Campus Tampico

Alumno:

Blanca Leticia Badillo Guzmán 511262

Materia:

Laboratorio de Desarrollo de Aplicaciones Web

Tutor:

Roberto Guevara González

Crear migraciones

php artisan make:migration nombre

Por convención se utiliza la sintáxis del nombre “create_users_table”. Habrá casos en los que se utilizarán migraciones para agregar campos a tablas ya existentes o algún otro tipo de alteración, por lo que nombrar la migración describiendo su función será muy útil para dar mantenimiento.

Ejemplo de creación de tabla

```
Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name')->unique();
            $table->string('first_name')->nullable();
            $table->string('last_name')->nullable();
            $table->string('email')->unique()->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->boolean('activated')->default(false);
            $table->string('token');
            $table->ipAddress('signup_ip_address')->nullable();
            $table->ipAddress('signup_confirmation_ip_address')->nullable();
            $table->ipAddress('signup_sm_ip_address')->nullable();
            $table->ipAddress('admin_ip_address')->nullable();
            $table->ipAddress('updated_ip_address')->nullable();
            $table->ipAddress('deleted_ip_address')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
```

increments	Entero que se va incrementando
String	Cadena de texto
Unique	Atributo único
rememberToken	Token con cifrado de laravel
Nullable	Este campo puede ser nulo
Boolean	Campo de tipo booleano
ipAddress	Campo de dirección ip
Foreign	Llave foránea
References	Se utiliza cuando se hace referencia a otra table
Timestamps	Crea los campos created_at y updated_at
Morphs	Crea los campos para utilizar relaciones polimórficas

Seeders

Los seeders son clases que se mandan llamar para popular la base de datos con información pre-existente que se ocupa para el funcionamiento del sistema.

php artisan make:seeder NameSeeder -> to create seeder

php artisan db:seed –class=NameSeeder ->to call seeder

Otra opción es php artisan migrate -seed, que correrá las migraciones y luego llenaría la base de datos con los datos puestos en los seeders. Para esto hay que añadir $this->call(NameSeeder::class); al DatabaseSeeder.php

```
if (Permission::where('name', '=', 'Can View Users')->first() === null) {
            Permission::create([
                'name'        => 'Can View Users',
                'slug'        => 'view.users',
                'description' => 'Can view users',
                'model'       => 'Permission',
            ]);
        }
```

En este caso se valida que exista el permiso, si no existe se crea. También podría utilizarse el método FirstOrCreate.

Modelo

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Activation extends Model
{
    /**
     * The database table used by the model.
     *
     * @var string
     */
    /**
     * Indica a qué tabla hace referencia el modelo.
     * De no ponerse se toma el nombre del modelo en plural en minúsculas
     * Es decir "activations"
    */
    protected $table = 'activations';

    /**
     * The attributes that are not mass assignable.
     *
     * @var array
     */
    protected $guarded = [
        'id',
    ];
     /**
      * Al hacer un query como Activation::all()
      * los siguientes campos no serán devueltos
      */
    protected $hidden = [
        'user_id',
        'token',
        'ip_address',
    ];

    /**
     * Relación con la tabla usuario
```

```
     * Quiere decir que cada activación pertenece a un usuario
     */
    public function user()
    {
        return $this->belongsTo(User::class);
    }
}
```

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Profile extends Model
{
    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'profiles';

    /**
     * The attributes that are not mass assignable.
     *
     * @var array
     */
    protected $guarded = [
        'id',
    ];

    /**
     * Fillable fields for a Profile.
     *
     * @var array
     */
    /**
     * Los siguientes campos podrán ser insertados mediante mass aignment
     */
    protected $fillable = [
        'theme_id',
        'location',
        'bio',
        'twitter_username',
```

```
        'github_username',
        'user_profile_bg',
        'avatar',
        'avatar_status',
    ];

    /**
     * Le da un tipo de dato al atributo
     */

    protected $casts = [
        'theme_id' => 'integer',
    ];

    /**
     * A profile belongs to a user.
     *
     * @return mixed
     */
    public function user()
    {
        return $this->belongsTo('App\Models\User');
    }

    /**
     * Profile Theme Relationships.
     * Cada usuario estará relacionado a UN theme
     * @var array
     */
    public function theme()
    {
        return $this->hasOne('App\Models\Theme');
    }
}
```

Otras funciones útiles son las de validación.

Por ejemplo, en el modelo Theme

```php
/**
     * Get a validator for an incoming registration request.
     *
     * @param array $data
     *
     * @return array
     */
    public static function rules($id = 0, $merge = [])
    {
        return array_merge(
            [
                'name'   => 'required|min:3|max:50|unique:themes,name'.($id ? ",$id" : ''),
                'link'   => 'required|min:3|max:255|unique:themes,link'.($id ? ",$id" : ''),
                'notes'  => 'max:500',
                'status' => 'required',
            ],
            $merge);
    }
```

Se da un arreglo con el nombre del atributo y las características a validar.

Por ejemplo, el atributo “name” es un campo requerido, de mínimo 3 caracteres de extensión y un máximo de 50 caracteres. Además es único en themes y name. Esto también se puede crear en una validation request.

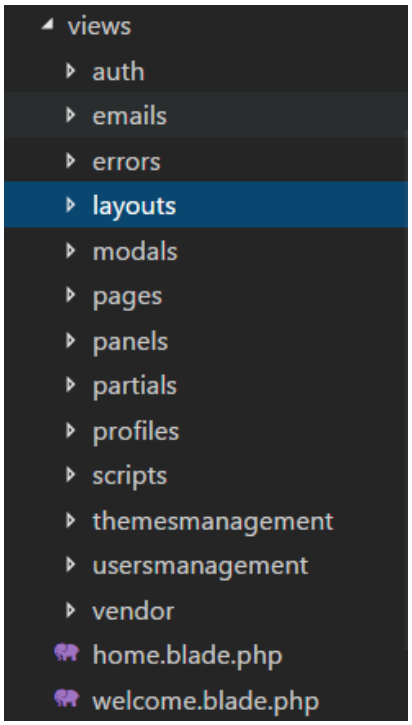
```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use jeremykenedy\LaravelRoles\Traits\HasRoleAndPermission;

class User extends Authenticatable
{
```

```
    use HasRoleAndPermission;
    use Notifiable;
    use SoftDeletes;

    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'users';

    /**
     * The attributes that are not mass assignable.
     *
     * @var array
     */

    protected $guarded = ['id'];

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'first_name',
        'last_name',
        'email',
        'password',
        'activated',
        'token',
        'signup_ip_address',
        'signup_confirmation_ip_address',
        'signup_sm_ip_address',
        'admin_ip_address',
        'updated_ip_address',
        'deleted_ip_address',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
```

```php
    protected $hidden = [
        'password',
        'remember_token',
        'activated',
        'token',
    ];
    /**
     * Estos campdaos serán tratados como fechas
     * por lo que se podrá hacer uso de la sintáxis Carbon
     * y aún así se guardarían con el formato correcto en la BD
     */
    protected $dates = [
        'deleted_at',
    ];

    /**
     * Build Social Relationships.
     *
     * @var array
     */
    /**
     * Un usuario tiene varias redes sociales
     */
    public function social()
    {
        return $this->hasMany('App\Models\Social');
    }

    /**
     * User Profile Relationships.
     *
     * @var array
     */
    /**
     * Habrá un perfil por usuario
     */
    public function profile()
    {
        return $this->hasOne('App\Models\Profile');
    }

    // User Profile Setup - SHould move these to a trait or interface...
    /**
     * Un usuario puede pertenecer a varios perfies
     */
```

```php
    public function profiles()
    {
        return $this->belongsToMany('App\Models\Profile')->withTimestamps();
    }

    /**
     * Regresa si un usuario tiene un perfil asignado
     */
    public function hasProfile($name)
    {
        foreach ($this->profiles as $profile) {
            if ($profile->name == $name) {
                return true;
            }
        }

        return false;
    }

    /**
     * Asigna un perfil en la pivot table
     */
    public function assignProfile($profile)
    {
        return $this->profiles()->attach($profile);
    }

    /**
     * Quita el perfil de la pivot table
     */
    public function removeProfile($profile)
    {
        return $this->profiles()->detach($profile);
    }
}
```

Vistas



Se recomienda ordenarlas en subcarpetas por módulos. Es decir, todo lo que tenga que ver con autenticación iría en la carpeta auth. Y así sucesivamente.

Se recomienda tener una carpeta de partials, donde se incluya código que deba repetirse constantemente en diferentes lugares. De esta forma sería más sencillo hacer referencia a esas secciones de código para incluirlas.

También se recomienda hacer un layout, en donde se haga una plantilla o esqueleto de las vistas. Un layout puede incuir la navbar, el footer, el menú, etc.

También es importante nombrar las vistas de forma descriptiva. Entre más fragmentado y descriptivo sea el código será mucho más sencillo dar mantenimiento y localizar la fuente de algún error.

Se puede crear un layout con código HTML que compartan varias vistas. Por ejemplo, el título de la página, el encabezado, etc.

```
<!DOCTYPE html>
<html lang="{{ config('app.locale') }}">
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        {{-- CSRF Token --}}
        <meta name="csrf-token" content="{{ csrf_token() }}">

        <title>@if (trim($__env-
>yieldContent('template_title')))@yield('template_title') | @endif {{
config('app.name', Lang::get('titles.app')) }}</title>
        <meta name="description" content="">
        <meta name="author" content="Jeremy Kenedy">
        <link rel="shortcut icon" href="/favicon.ico">
```

```
        {{-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and
media queries --}}
        <!--[if lt IE 9]>
            <script
src="https://oss.maxcdn.com/html5shiv/3.7.2/html5shiv.min.js"></script>
            <script
src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>
        <![endif]-->

        {{-- Fonts --}}
        @yield('template_linked_fonts')

        {{-- Styles --}}
        <link href="{{ mix('/css/app.css') }}" rel="stylesheet">

        @yield('template_linked_css')

        <style type="text/css">
            @yield('template_fastload_css')

            @if (Auth::User() && (Auth::User()->profile) && (Auth::User()-
>profile->avatar_status == 0))
                .user-avatar-nav {
                    background: url({{ Gravatar::get(Auth::user()->email)
}}) 50% 50% no-repeat;
                    background-size: auto 100%;
                }
            @endif

        </style>

        {{-- Scripts --}}
        <script>
            window.Laravel = {!! json_encode([
                'csrfToken' => csrf_token(),
            ]) !!};
        </script>

        @if (Auth::User() && (Auth::User()->profile) && $theme->link != null
&& $theme->link != 'null')
            <link rel="stylesheet" type="text/css" href="{{ $theme->link
}}">
        @endif

        @yield('head')
```

```
    </head>
    <body>
        <div id="app">

            @include('partials.nav')

            <div class="container">

                @include('partials.form-status')

            </div>

            @yield('content')

        </div>

        {{-- Scripts --}}
        <script src="{{ mix('/js/app.js') }}"></script>

        @if(config('settings.googleMapsAPIStatus'))
            {!!
HTML::script('//maps.googleapis.com/maps/api/js?key='.env("GOOGLEMAPS_API_KE
Y").'&libraries=places&dummy=.js', array('type' => 'text/javascript')) !!}
        @endif

        @yield('footer_scripts')

    </body>
</html>
```

@yield(‘nombre’)

Hace referencia al código que iría en esa sección. En una vista se pondría

@section(‘nombre’)

<p> Todo el código escrito dentro de esta sección se pondría en donde pusimos el @yield

@endsection

En el caso del @include(‘nombre’), se utiliza para incluir el código del archivo en cierta sección del código.

@if @else @endif

Funcionan como una condición if normal.

En las vistas también se puede accesar a variables pasadas por medio del controlador.

Ejemplo

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

@section('template_title')
    Routing Information
@endsection

@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-12">

                @include('partials.form-status')

                <div class="panel panel-default">
                    <div class="panel-heading">
                        Routing Information
                        <span class="badge badge-primary pull-right">{{
count($routes) }} routes</span>
                    </div>
                    <div class="panel-body">
                        <div class="table-responsive">
                            <table class="table table-striped table-
condensed data-table">
                                <thead>
                                    <tr class="success">
                                        <th>URI</th>
                                        <th>Name</th>
                                        <th>Type</th>
                                        <th>Method</th>
                                    </tr>
                                </thead>
                                <tbody>
                                    @foreach ($routes as $route)
                                        <tr>
                                        <td>{{$route->uri}}</td>
                                        <td>{{$route->getName()}}</td>
                                        <td>{{$route->getPrefix()}}</td>
                                        <td>{{$route-
>getActionMethod()}}</td>
                                        </tr>
                                    @endforeach
                                </tbody>
```

```
                            </table>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
@endsection
```

Aquí se hace uso de la variable $routes y sus atributos, se imprime su contenido en una tabla.

Controladores

Todos los controladores pueden tener una función de constructor. En esta se pueden agregar lo middlewares.

```
/**
     * Create a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {   //únicamente pueden accesar a estas funciones usuarios autenticados
        $this->middleware('auth');
    }
```

También se pueden hacer validaciones mediante el controlador

```
/**
     * Get a validator for an incoming registration request.
     *
     * @param array $data
     *
     * @return \Illuminate\Contracts\Validation\Validator
     */
    public function profile_validator(array $data)
    {   //Esta validación puede ir en el modelo o en una custom request
        return Validator::make($data, [
            'theme_id'         => '',
            'location'         => '',
            'bio'              => 'max:500',
            'twitter_username' => 'max:50',
            'github_username'  => 'max:50',
            'avatar'           => '',
```

```
            'avatar_status'    => '',
        ]);
    }
```

Rutas

```
<?php

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
| Middleware options can be located in `app/Http/Kernel.php`
|
*/

// Homepage Route
Route::get('/', 'WelcomeController@welcome')->name('welcome');

// Authentication Routes
Auth::routes();

// Public Routes
//Todas usan los middlewares de web y activity
Route::group(['middleware' => ['web', 'activity']], function () {
    //get routes only
    // Activation Routes
    Route::get('/activate', ['as' => 'activate', 'uses' =>
'Auth\ActivateController@initial']);

    Route::get('/activate/{token}', ['as' => 'authenticated.activate',
'uses' => 'Auth\ActivateController@activate']);
    Route::get('/activation', ['as' => 'authenticated.activation-resend',
'uses' => 'Auth\ActivateController@resend']);
    Route::get('/exceeded', ['as' => 'exceeded', 'uses' =>
'Auth\ActivateController@exceeded']);

    // Socialite Register Routes
    Route::get('/social/redirect/{provider}', ['as' => 'social.redirect',
'uses' => 'Auth\SocialController@getSocialRedirect']);
```

```
    Route::get('/social/handle/{provider}', ['as' => 'social.handle', 'uses'
=> 'Auth\SocialController@getSocialHandle']);

    // Route to for user to reactivate their user deleted account.
    Route::get('/re-activate/{token}', ['as' => 'user.reactivate', 'uses' =>
'RestoreUserController@userReActivate']);
});

// Registered and Activated User Routes
Route::group(['middleware' => ['auth', 'activated', 'activity']], function
() {

    // Activation Routes
    Route::get('/activation-required', ['uses' =>
'Auth\ActivateController@activationRequired'])->name('activation-required');
    Route::get('/logout', ['uses' => 'Auth\LoginController@logout'])-
>name('logout');
});

// Registered and Activated User Routes
Route::group(['middleware' => ['auth', 'activated', 'activity', 'twostep']],
function () {

    //  Homepage Route - Redirect based on user role is in controller.
    Route::get('/home', ['as' => 'public.home',   'uses' =>
'UserController@index']);

    // Show users profile - viewable by other users.
    Route::get('profile/{username}', [
        'as'   => '{username}',
        'uses' => 'ProfilesController@show',
    ]);
});

// Registered, activated, and is current user routes.
Route::group(['middleware' => ['auth', 'activated', 'currentUser',
'activity', 'twostep']], function () {

    // User Profile and Account Routes
    //Resource crea las rutas para hacer el CRUD del modelo profile. En este
caso solo crea show, edit, update y create. No se crean index y delete
    Route::resource(
        'profile',
        'ProfilesController', [
            'only' => [
```

```
                'show',
                'edit',
                'update',
                'create',
            ],
        ]
    );
    //Put es para el método PUT que es para updates
    Route::put('profile/{username}/updateUserAccount', [
        'as'   => '{username}',
        'uses' => 'ProfilesController@updateUserAccount',
    ]);
    Route::put('profile/{username}/updateUserPassword', [
        'as'   => '{username}',
        'uses' => 'ProfilesController@updateUserPassword',
    ]);
    //Delete method en forms
    Route::delete('profile/{username}/deleteUserAccount', [
        'as'   => '{username}',
        'uses' => 'ProfilesController@deleteUserAccount',
    ]);

    // Route to show user avatar
    Route::get('images/profile/{id}/avatar/{image}', [
        'uses' => 'ProfilesController@userProfileAvatar',
    ]);

    // Route to upload user avatar.
    Route::post('avatar/upload', ['as' => 'avatar.upload', 'uses' =>
'ProfilesController@upload']);
});

// Registered, activated, and is admin routes.
Route::group(['middleware' => ['auth', 'activated', 'role:admin',
'activity', 'twostep']], function () {
    Route::resource('/users/deleted', 'SoftDeletesController', [
        'only' => [
            'index', 'show', 'update', 'destroy',
        ],
    ]);

    Route::resource('users', 'UsersManagementController', [
        'names' => [
            'index'   => 'users',
            'destroy' => 'user.destroy',
```

```
        ],
        //lo opuesto al only
        'except' => [
            'deleted',
        ],
    ]);
    Route::post('search-users', 'UsersManagementController@search')-
>name('search-users');

    Route::resource('themes', 'ThemesManagementController', [
        //Se le puede dar nombre a las rutas para su fácil acceso en las
vistas. En lugar del url se pone {{url('themes.destroy')}}
        'names' => [
            'index'   => 'themes',
            'destroy' => 'themes.destroy',
        ],
    ]);

    Route::get('logs',
'\Rap2hpoutre\LaravelLogViewer\LogViewerController@index');
    Route::get('routes', 'AdminDetailsController@listRoutes');
    Route::get('active-users', 'AdminDetailsController@activeUsers');
});
//Si el usuario ingresa a la ruta /php lo reedirige a la ruta /phpinfo
Route::redirect('/php', '/phpinfo', 301);
```