

## Proyecto Laravel

### 1.1 Instalación de laravel

```
composer global require laravel/installer
```

### 1.2 Crear proyecto

Creamos un proyecto con autenticación y gestión de usuarios. En nuestro caso el proyecto se llamará `project-20`.

```
Laravel new project-20 --Auth
```

### 1.3 Base de datos

#### 1.3.1 Create database

Creamos la base de datos de nuestro proyecto mediante la instrucción sql correspondiente, en nuestro caso la base de datos se llamará `project-20` también.

```
Create database project-20;
```

#### 1.3.2 Configuramos BD en Laravel

Para la configuración del acceso a esta base de datos buscamos en el directorio raíz de la estructura de carpetas de Laravel el archivo `env`. y especificamos el nombre de la base de datos que acabamos de crear en la sección correspondiente.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=project20
DB_USERNAME=root
DB_PASSWORD=
```

#### 1.3.3 Migraciones

Por defecto cuando se crea un proyecto laravel ya vienen creada las siguientes migraciones:

- `Create_users_table`
- `Create_password_resets_table`
- `Create_failed_jobs_table`

Con ello no nos es suficiente, puesto que nuestra gestión de usuarios va a contemplar roles, con distintos privilegios cada uno de ellos. Un usuario puede tener varios `roles` y un rol puede ser asignado a varios usuarios, por lo tanto la relación entre usuarios y roles es del tipo M:N, por lo que habrá que crear una nueva tabla `role_user` para establecer ese tipo de relación.

Así pues las migraciones que tenemos que crear son:

- Create\_roles\_table
- Create\_role\_user\_table

#### 1.3.3.1 Migración `create_users_table`

Esta migración está creada sólo habrá que modificarla si deseamos añadir o modificar alguna columna de la tabla `users`.

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

class CreateUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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

#### 1.3.3.2 Migración `create_roles_table`

Creamos la migración con el siguiente comando

```
php artisan make:migration create_roles_table --table=roles
```

Busco el archivo en la carpeta `database/migrations`

```
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->id();
            $table->string('name', 20);
            $table->string('description', 100);
            $table->timestamps();
        });
    }

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

#### 1.3.3.3 *Migración create\_role\_user\_table*

Igualmente usamos el comando siguiente para crear la migración

```
php artisan make:migration create_role_user_table --table=role_user
```

En esta migración además de las columnas de la tabla tenemos que especificar y esto es muy importante las restricciones foreign key a las columnas `user_id` y `role_id`

```
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->id();
            $table->unsignedBigInteger('user_id');
            $table->unsignedBigInteger('role_id');
            $table->timestamps();
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade')->onUpdate('cascade');
            $table->foreign('role_id')->references('id')->on('roles')->onDelete('cascade')->onUpdate('cascade');
        });
    }

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

```

#### 1.3.4 Modelos

Por defecto los modelos se guardarán como clases PHP dentro de la carpeta `app`, sin embargo Laravel nos da libertad para colocarlos en otra carpeta si queremos, como por ejemplo la carpeta `app/Models`. Pero en este caso tendremos que asegurarnos de indicar correctamente el espacio de nombres.

En nuestro proyecto tendremos que crear los modelos para las tablas `users` y `roles`. Se recuerda que la nomenclatura de Laravel aconseja que si bien el nombre de las tablas ha de ir en plural el nombre del modelo se usa el singular.

##### 1.3.4.1 Modelo User

El modelo `user` no tengo que crearlo puesto que por defecto ya lo instala Laravel.

Sobre el fichero ya creado sólo tengo que añadir las líneas **resaltadas** para establecer la relación M:N entre usuarios y roles.

```
-- user.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',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function roles()
    {
        return $this->belongsToMany('App\Role');
    }
}

```

#### 1.3.4.2 Modelo Role

El modelo para la tala Roles si que tengo que crearlo por lo que ejecuto primero el siguiente comando.

```
Php artisan make:model Role
```

Genera automáticamente el fichero `role.php` al cual le tengo que añadir las siguientes líneas **resaltadas** para establecer la relación donde un rol puede ser asignado a varios usuarios, es decir, la relación M:N ya comentada entre Roles e Usuarios.

```
-- Role.php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Role extends Model
{
    public function users()
    {
        return $this->belongsToMany('App\User');
    }
}
```

### 1.3.5 Seeders

#### 1.3.5.1 RolesTableSeeder

Este seeder lo usaremos para llenar de datos la tabla Roles en la que insertaremos 3 registros con los tres roles de pruebas que vamos a crear: admin, author y user.

Crear el seeder RolesTableSeeder

```
Php artisan make:seeder RolesTableSeeder
```

Al archivo creado le tenemos que añadir las siguientes líneas resaltadas

```
-- RolesTableSeeder

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;
use App\Role;

class RolesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
```

```

        DB::statement('SET FOREIGN_KEY_CHECKS = 0');

        Role::truncate();

        Role::create(['name' => 'admin', 'description' => 'Todos los
privilegios']);
        Role::create(['name' => 'author', 'description' => 'No todos los
privilegios' ]);
        Role::create(['name' => 'user', 'description' => 'Sin privilegios']);

        DB::statement('SET FOREIGN_KEY_CHECKS = 1');

    }
}

```

### 1.3.5.2 *UsersTableSeeder*

Crear seeders `UsersTableSeeder`

```
Php artisan make:seeder UsersTableSeeder
```

Este seeder lo vamos a emplear no sólo para añadir los registros a la tabla `Users`, también a la tabla `role_user`.

```

-- UsersTableSeeder

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\DB;
use App\User;
use App\Role;

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        DB::statement('SET FOREIGN_KEY_CHECKS = 0');
        User::truncate();
        DB::table('role_user')->truncate();
    }
}

```

```

$adminRole = Role::where('name', 'admin')->first();
$authorRole = Role::where('name', 'author')->first();
$userRole = Role::where('name', 'user')->first();

$yomismo = User::create([
    'name' => 'Juan Carlos Moreno',
    'email' => 'nerom24@gmail.com',
    'password' => Hash::make('casitachica')
]);

$admin = User::create([
    'name' => 'Administrador Name',
    'email' => 'admin@gmail.com',
    'password' => Hash::make('password')
]);

$author = User::create([
    'name' => 'Author Name',
    'email' => 'author@gmail.com',
    'password' => Hash::make('password')
]);

$user = User::create([
    'name' => 'User Name',
    'email' => 'user@gmail.com',
    'password' => Hash::make('password')
]);

// con estas líneas añado los registros a la tabla role_user

$yomismo->roles()->attach($adminRole);
$admin->roles()->attach($adminRole);
$author->roles()->attach($authorRole);
$user->roles()->attach($userRole);

DB::statement('SET FOREIGN_KEY_CHECKS = 1');

}
}

```

### 1.3.5.3 DatabaseSeeder

Este seeder se crea por defecto con la instalación por lo que no hay que crearlo y lo que hará es llamar al resto de los seeder para que se ejecuten.

```
use Illuminate\Database\Seeder;
```



```

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $this->call(RolesTableSeeder::class);
        $this->call(UsersTableSeeder::class);
    }
}

```

#### 1.3.5.4 Ejecutar los seeders

Ahora para ejecutar los seeders y llenar de datos ejemplos nuestra base de datos, así ejecutamos el siguiente comando.

```
Php artisan db:seed
```

Cada vez que lo ejecute se perderán todos los datos y se restaurarán los datos ejemplos incluidos en los seeders.

## 1.4 Controladores

Creamos el controlador para la gestión de usuarios, indicando el nombre del modelo como parámetro.

```
php artisan make:controller Admin\UsersController --resource --model=User
```

Así creamos el controlador `UsersController` al que en principio no le voy a añadir ningún código.

Los métodos que se incluyen en este controlador al especificar el parámetro `--resource` en su creación son

- `Index()`
- `Create()`
- `Store()`
- `Show()`
- `Edit()`
- `Update()`
- `Destroy()`

Ahora a cada método de dicho controlador habrá que asignarle una ruta

## 1.5 Rutas

Para establecer las rutas al controlador `UserController` añadido al archivo `web.php` el siguiente código

```
Route::resource('/admin/users', 'Admin\UsersController', ['except' => ['show', 'create', 'store']]);
```

Esto implica que ahora voy a tener disponibles las siguientes rutas

```
Php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	admin/users	users.index	App\Http\Controllers\Admin\UsersController@index	web,auth
	DELETE	admin/users/{user}	users.destroy	App\Http\Controllers\Admin\UsersController@destroy	web,auth
	PUT PATCH	admin/users/{user}	users.update	App\Http\Controllers\Admin\UsersController@update	web,auth
	GET HEAD	admin/users/{user}/edit	users.edit	App\Http\Controllers\Admin\UsersController@edit	web,auth
	GET HEAD	api/user		Closure	api,auth:api
	GET HEAD	home	home	App\Http\Controllers\HomeController@index	web
	POST	login	login	App\Http\Controllers\Auth\LoginController@login	web,guest
	GET HEAD	login	login	App\Http\Controllers\Auth\LoginController@showLoginForm	web,guest
	POST	logout	logout	App\Http\Controllers\Auth\LoginController@logout	web
	POST	password/confirm		App\Http\Controllers\Auth\ConfirmPasswordController@confirm	web,auth
	GET HEAD	password/confirm	password.confirm	App\Http\Controllers\Auth\ConfirmPasswordController@showConfirmForm	web,auth
	POST	password/email	password.email	App\Http\Controllers\Auth\ForgotPasswordController@sendResetLinkEmail	web
	GET HEAD	password/request	password.request	App\Http\Controllers\Auth\ForgotPasswordController@showLinkRequestForm	web
	POST	password/reset	password.update	App\Http\Controllers\Auth\ResetPasswordController@reset	web
	GET HEAD	password/reset/{token}	password.reset	App\Http\Controllers\Auth\ResetPasswordController@showResetForm	web
	POST	register	register	App\Http\Controllers\Auth\RegisterController@register	web,guest
	GET HEAD	register	register	App\Http\Controllers\Auth\RegisterController@showRegistrationForm	web,guest

Lo cambio por este otro para añadir el prefijo admin a las acciones destinadas a las tareas de administración

```
Route::namespace('Admin')->prefix('admin')->name('admin.')->group(function() {

    Route::resource('/users', 'UsersController', ['except' => ['show', 'create', 'store']]);

});
```

El fichero `web.php` quedaría entonces de la siguiente forma

```
use Illuminate\Support\Facades\Route;

/*
|-----
| 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!
|
*/

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

```
Auth::routes();

Route::get('/home', 'HomeController@index')->name('home');

// Añadir el prefijo admin a las rutas de administración

Route::namespace('Admin')->prefix('admin')->name('admin.')->
>middleware('can:manage-users')->group(function() {

    Route::resource('/users', 'UserController', ['except' => ['show',
'create', 'store']]);

});
```

## 1.6 Lista usuarios

Se trata de la primera acción que vamos a asignar a nuestro proyecto.

### 1.6.1 Ruta

Para poder acceder a esata funcionalidad parto de la ruta

```
Admi.users.index
```

### 1.6.2 UsersController

Esta ruta está asociada al método `index()` de `UserController` por lo tanto tengo ahora que incluir en dicho método el siguiente código

```
public function index()
{
    // Comando Eloquent que extrae todos los usuarios
    $users= User::all();
    return view('admin.users.index', compact('users'));
}
```

### 1.6.3 Vista index.blade.php

En la carpeta `views/admin/users` creo la vista `index.blade.php` que se encargará de mostrar los usuarios de la tabla `users`.

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

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">Usuarios</div>
```

```

<div class="card-body">
  @if (session('status'))
    <div class="alert alert-success" role="alert">
      {{ session('status') }}
    </div>
  @endif

  {{-- Muestra los usuarios en una tabla --}}

  <table class="table">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">Nombre</th>
        <th scope="col">Correo</th>
        <th scope="col">Roles</th>
        <th scope="col">Acciones</th>
      </tr>
    </thead>
    <tbody>

      @forelse ($users as $user)
        <tr>
          <td>{{ $user->id }}</td>
          <td>{{ $user->name }}</td>
          <td>{{ $user->email }}</td>
          <td>{{-- Extra los roles del usuario --}}
            <td>{{ implode(', ', $user->roles()->get()-
>pluck('name')->toArray()) }}</td>
            <td>
              <a href="{{ route('admin.users.edit', $user-
>id) }}"> <button type="button" class="btn btn-primary
inline">Editar</button></a>
              <a href="{{ route('admin.users.destroy,
$user) }}"> <button type="button" class="btn btn-danger
inline">Eliminar</button></a>

            </td>
          </tr>
        @empty
          <li>No hay usuarios registrados.</li>
        @endforelse

      </tbody>
    </table>

```

```

        Total Usuarios: {{$users->count()}}

    </div>
</div>
</div>
</div>
</div>
@endsection

```

Dicha vista dará lugar a la siguiente pantalla

Laravel Juan Carlos Moreno ▾

Usuarios					
#	Nombre	Correo	Roles	Acciones	
1	Juan Carlos Moreno	nerom24@gmail.com	admin	<a href="#">Editar</a>	<a href="#">Eliminar</a>
2	Administrador Name	admin@gmail.com	admin	<a href="#">Editar</a>	<a href="#">Eliminar</a>
3	Author Name	author@gmail.com	author	<a href="#">Editar</a>	<a href="#">Eliminar</a>
4	User Name	user@gmail.com	user	<a href="#">Editar</a>	<a href="#">Eliminar</a>

Total Usuarios: 4

#### 1.6.4 Plantilla de diseño

La plantilla de diseño a partir de la cual se crearán todas las vistas de nuestro proyecto está en la carpeta `views/layout` y se llama `app.blade.php`. Ese archivo viene incluido en nuestra instalación Laravel –Auth

### 1.7 Editar Usuarios

Permitirá editar los datos de un usuario.

#### 1.7.1 Ruta

Para llevar a cabo esta funcionalidad necesito dos rutas

```

Admin.users.edit asociada a la url admin/users/{user}/edit
Admin.users.update asociada a la url admin/users/{user}

```

Con la primera ruta muestro en un formulario la edición de los datos de un usuario y con la segunda ruta, una vez modificados los datos procedo a su validación y actualización definitiva en la base de datos.

#### 1.7.2 UsersController

Necesito ahora añadir código a dos métodos de este controlador

- `Edit()` asociado a la ruta `admin.users.edit`
- `Update()` asociado a la ruta `admin.users.update`

Veamos el método **`edit()`**

```

public function edit(User $user)
{
    $roles = Role::all();
    return view('admin.users.edit', compact('user', 'roles'));
}

```

Veamos ahora el método **update()**

```

public function update(Request $request, User $user)
{
    // Sync Acepta una matriz de roles para colocar en la tabla user_role
    // los valores que no estén los elimina
    $user->roles()->sync($request->roles);
    return redirect()->route('admin.users.index');
}

```

### 1.7.3 Vista edit.blade.edit

Esta vista recibirá dos parámetros del controlador el `$roles` (lista de roles) y el objeto `$user`, con los datos del usuario que deseamos editar.

También hay que crearla en la carpeta `views/admin/users`

```

@extends('layouts.app')

@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">Editar {{ $user->name }}</div>

                <div class="card-body">
                    @if (session('status'))
                        <div class="alert alert-success" role="alert">
                            {{ session('status') }}
                        </div>
                    @endif

                    {{-- Formulario de edición con método PUT y csrf token de
seguridad --}}

                    <form action={{ route('admin.users.update', $user) }} method
= "POST">

                        @csrf
                        {{ method_field('PUT') }}

```

```

        {{-- Campo Name --}}
        <div class="form-group">
            <label class="form-label">{{ __('Nombre')
}}</label>

            <input id="name" type="text" name="name"
class="form-control @error('name') is-invalid @enderror" value="{{ $user-
>name }}">

            @error('name')
                <span class="invalid-feedback"
role="alert">

                    <strong>{{ $message }}</strong>
                </span>
            @enderror
        </div>

        {{-- Campo Email --}}
        <div class="form-group">
            <label class="form-label">{{ __('Email')
}}</label>

            <input id="name" type="email" name="email"
class="form-control @error('email') is-invalid @enderror" value="{{ $user-
>email }}">

            @error('email')
                <span class="invalid-feedback"
role="alert">

                    <strong>{{ $message }}</strong>
                </span>
            @enderror
        </div>

        <label class="form-label">{{ __('Roles') }}</label>
        @foreach ($roles as $role)

            <div class="form-check">
                <input class="form-check-input" type="checkbox"
name="roles[]" value="{{ $role->id }}" >
                <label class="form-check-label"
for="defaultCheck1">

                    {{ $role->name }}
                </label>
            </div>

        @endforeach
        <hr>
        <a class="btn btn-secondary"
href="{{ route('admin.users.index') }}" role="button">Cancelar</a>
        <button type="submit" class="btn btn-
primary">Actualizar</button>

```

```

        </form>

    </div>

</div>

</div>

</div>

</div>
@endsection

```

## 1.8 Control de accesos

En nuestro proyecto hemos creado los siguientes un conjunto de perfiles o roles con la asignación de una serie de privilegios.

## Tabla de perfiles y privilegios

Rol	Privilegios
admin	Edit y Delete
author	Edit
user	

Las modificaciones que tenemos que realizar en nuestro proyecto para incluir funcionalidad son las que se detallan a continuación.

### 1.8.1 Modelo User

En le modelo `User` debemos añadir los siguiéntes métodos

```
#Comprueba si el modelo actual de usuario tiene un rol determinado
public function hasRole($role) {

    if ($this->roles()->where('name', $role)->first()) {

        return true;
    }

    return false;
}

# Comprueba si un usuario tiene alguno de los roles recibidos en un #
array
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 authorizeRoles($roles) {

    if ($this->hasAnyRole($roles)) {
        return true;
    }

    abort(403, 'Acción No Autorizada');
}

```

### 1.8.2 Providers

En la carpeta `/providers` accedo al archivo `AuthServiceProvider.php` y añado el siguiente código

#### **AuthServiceProvider.php**

```

namespace App\Providers;

use Illuminate\Foundation\Support\Providers\AuthServiceProvider as
ServiceProvider;
use Illuminate\Support\Facades\Gate;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        // 'App\Model' => 'App\Policies\ModelPolicy',
    ];

    /**
     * Register any authentication / authorization services.
     *
     * @return void
     */
}

```

```

        */
        public function boot()
        {
            $this->registerPolicies();

            Gate::define('manage-users', function($user) {
                return $user->hasAnyRole(['admin', 'author']);
            });

            Gate::define('edit-users', function($user) {
                return $user->hasAnyRole(['admin', 'author']);
            });

            Gate::define('delete-users', function($user) {
                return $user->hasRole('admin');
            });

            //
        }
    }
}

```

### 1.8.3 UsersController

Añadimos en la clase `UsersController` el constructor.

```

public function __construct()
{
    $this->middleware('auth');
}

```

Ahora debemos establecer el filtro de acceso en los métodos `edit()`, `update()` y `delete()`

```

public function edit(User $user)
{
    // Control de acceso Gate
    if (Gate::denies('edit-users')) {
        return redirect(route('admin.users.index'));
    }

    $roles = Role::all();
    return view('admin.users.edit', compact('user', 'roles'));
}

/**
 * Update the specified resource in storage.
 *
 * @param \Illuminate\Http\Request $request
 */

```

```

* @param \App\User $user
* @return \Illuminate\Http\Response
*/
public function update(Request $request, User $user)
{
    if (Gate::denies('edit-users')) {
        return redirect(route('admin.users.index'));
    }

    // Sync Acepta una matriz de roles para colocar en la tabla user_role
    // los valores que no estén los elimina
    $user->roles()->sync($request->roles);
    return redirect()->route('admin.users.index');
}

/**
 * Remove the specified resource from storage.
 *
 * @param \App\User $user
 * @return \Illuminate\Http\Response
 */
public function destroy(User $user)
{
    // Control de acceso Gate
    if (Gate::denies('delete-users')) {
        return redirect(route('admin.users.index'));
    }

    // Elimina los roles de este usuario
    $user->roles()->detach();
    $user->delete();

    return redirect()->route('admin.users.index');
}

```

#### 1.8.4 Modificaciones en las vistas

Se muestra la parte de la vista `index.blade.php` donde se establecerán los filtros para mostrar o no un botón de acción usando la directiva `@can`

```

@forelse ($users as $user)
    <tr>
        <td>{{ $user->id }}</td>
        <td>{{ $user->name }}</td>
        <td>{{ $user->email }}</td>
        {{-- Extra los roles que tiene asignado el usuario --}}
        <td>{{ implode(', ', $user->roles()->get()-
>pluck('name')->toArray()) }}</td>
        <td>

```

```

                @can('edit-users')
                <a href="{{ route('admin.users.edit', $user-
>id) }}"> <button type="button" class="btn btn-primary
inline">Editar</button></a>
                @endcan

                @can('delete-users')
                <form action="{{ route('admin.users.destroy',
$user) }}" method = "POST" class="list-inline-item">
                    @csrf
                    {{ method_field('DELETE') }}
                    <button type="submit" class="btn btn-
danger">Eliminar</button></a>
                </form>
                @endcan

            </td>
        </tr>
        @empty
            <li>No hay usuarios registrados.</li>
        @endforelse
    
```

## 1.9 Asignar rol a usuario registrado

Vamos a la carpeta `Http/Controllers/Auth` y abrimos el controlador

`RegisterController.php` en la función o método `create()` realizamos las siguientes modificaciones.

```

protected function create(array $data)
{
    $user = User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);

    // Usuario que se acaba de registrar le asignamos perfil user
    $role = Role::select('id')->where('name','user')->first();
    $user->roles()->attach($role);

    return $user;
}
    
```

## 1.10 Gestión de Mensajes

La gestión de mensajes es muy importante añadirla a nuestro proyecto ya que nos va a permitir mostrar cualquier tipo de mensaje ya sea de carácter informativo o bien cualquier tipo de error que se haya podido producir.

### 1.10.1 Alerts.blade.php

Creamos la carpeta `views/partials` el archivo `alerts.blade.php` con el siguiente contenido

```
-- alerts.blade.php

@if(session('success'))
    <div class="alert alert-success" role="alert">
        <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
        <b>Mensaje: </b>{{ session('success') }}
    </div>
@endif

@if(session('error'))
    <div class="alert alert-danger" role="alert">
        <button type="button" class="close" data-dismiss="alert" aria-label="Close"><span aria-hidden="true">&times;</span></button>
        <b>Error: </b>{{ session('error') }}
    </div>
@endif
```

### 1.10.2 Modificación en las vistas

Ahora en las vistas donde queremos que se muestre un mensaje usamos la directiva `@include` de blade.

Por ejemplo en la vista `index.blade.php`

```
@section('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            @include('partials.alerts')
            <div class="card">
                <div class="card-header">Usuarios</div>
            </div>
        </div>
    </div>
</div>
```

### 1.10.3 Controladores

En los métodos o funciones de los controladores que hacen llamada a las distintas vistas se incluirá el siguiente código. En nuestro ejemplo generamos un mensaje en el método `update()` de controlador `UserController` una vez que se han actualizado los datos de un registro.

```
public function update(Request $request, User $user)
{
    if (Gate::denies('edit-users')) {
        return redirect(route('admin.users.index'));
    }
}
```

```

        // Sync Acepta una matriz de roles para colocar en la tabla user_role
        // los valores que no estén los elimina
        $user->roles()->sync($request->roles);

        // Actualizar user sin validación
        $user->name = $request->name;
        $user->email = $request->email;

        if($user->save()) {
            $request->session()->flash('success', $user->name. ' ha sido
actualizado');
        } else {
            $request->session()->flash('error', 'Ocurrió un error en la
actualización');
        }

        return redirect()->route('admin.users.index');
    }

```

### 1.11 Redirect

Cuando se autentifique un usuario con rol admin queremos que se cargue directamente la ruta `admin.users.index` y no vaya al home como hasta ahora.

Para ello en la carpeta `App/Http/Controller/Auth` abrimos el controlador `LoginController.php` e incluimos la función `redirectTo()`.

```

public function redirectTo()
{
    if (Auth::user()->hasRole('admin')) {
        $this->redirectTo = route('admin.users.index');
    }
    return $this->redirectTo;
}

```

### 1.12 Crear usuario

Esta fase del proyecto consiste en habilitar el botón `Nuevo` en el panel de control de usuarios, permitiendo añadir un nuevo usuario a la tabla `Users`.

Usuarios

Nuevo

#	Nombre	Correo	Roles	Acciones
1	Juan	nerom24@gmail.com	admin, author	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
2	Administrador	admin@gmail.com	admin, author	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
3	Author Name	author@gmail.com	author	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>

Hay que decir que esta utilidad ya está incluida en el proyecto mediante la opción `register`, pero es conveniente incluirla también en este panel de control. También hay que tener en cuenta que una vez que se añada un nuevo usuario, automáticamente se le asignará el rol de User, en la tabla `role_user`.

#### 1.12.1 Rutas

En la siguiente se describen las nuevas rutas que tenemos que establecer para poder realizar esta fase.

URL	Name	Action
admin/users/create	admin.users.create	UserController@create
admin/users	admin.users.store	UserController@store

Para ello en la carpeta `Routes` actualizo el archivo `web` de la siguiente forma.

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

Auth::routes();

Route::get('/home', 'HomeController@index')->name('home');

// Añadir el prefijo admin a las rutas de administración

Route::namespace('Admin')->prefix('admin')->name('admin.')-
>middleware('can:manage-users')->group(function() {

    Route::resource('/users', 'UserController', ['except' => ['show']]);
});
```

```
});
```

### 1.12.2 UsersController

Este es el controlador asociado a las rutas anteriores y en él tengo que añadir importantes modificaciones, como son añadir código a:

- método `create()` para que muestre el formulario de creación de usuario
- método `store()` para que valide el formulario, muestre los errores en caso de no validación y si procede dar de alta el registro.

#### 1.12.2.1 Create()

Este método sólo lo podrá ejecutar el rol `admin`, por lo que previamente hay que crear un gate en el archivo `AuthServiceProvider` dentro de la carpeta `App/Providers`

```
Gate::define('create-users', function($user) {  
    return $user->hasRole('admin');  
});
```

Una vez declarada una puerta de acceso para dicho método en `UsersController` añadimos el siguiente código al método `create()`

```
public function create()  
{  
    // Control de acceso mediante Gate  
    if (Gate::denies('create-users')) {  
        session()->flash('error','Acción no autorizada');  
        return redirect(route('admin.users.index'));  
    }  
  
    return view('admin.users.create');  
}
```

#### 1.12.2.2 Store()

Este método es un poco más complicado que el anterior puesto que tiene que blindar el método para que sólo lo pueda usar el rol `admin`, también tiene que validar el formulario y en caso de validación procederá a añadir el nuevo usuario a la tabla `Users`.

La validación es la misma que se encuentra en el controlador `RegisterController.php` de la carpeta `App/Http/Controllers/Auth`

```
public function store(Request $request)  
{  
    // Control de acceso Gate  
    if (Gate::denies('create-users')) {  
        return redirect(route('admin.users.index'));  
    }  
}
```



```
// Validación del formulario
$validatedData = $request->validate
([
    'name' => ['required', 'string', 'max:255'],
    'email' => ['required', 'string', 'email', 'max:255',
'unique:users'],
    'password' => ['required', 'string', 'min:8', 'confirmed'],
]);

$user = User::create([
    'name' => $request['name'],
    'email' => $request['email'],
    'password' => Hash::make($request['password']),
]);

// Usuario que se acaba de registrar le asignamos perfil user
$role = Role::select('id')->where('name','user')->first();
$user->roles()->attach($role);

if($user) {
    $request->session()->flash('success','Usuario creado con éxito');
}else {
    $request->session()->flash('error','Formulario no validado');
}

return redirect()->route('admin.users.index');
}
```

### 1.12.3 Vista create.blade.index

Para poder completar esta fase tendremos que crear la vista asociada al método `create()` de `UserController.php`.

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

@section('content')
<div class="container">
  <div class="row justify-content-center">
    <div class="col-md-8">
      @include('partials.alerts')
      <div class="card">
        <div class="card-header">Nuevo usuario</div>

        <div class="card-body">
          <form method="POST" action="{{ route('admin.users.store')
}}">

            @csrf

            <div class="form-group row">
```

```

<label for="name" class="col-md-4 col-form-label
text-md-right">Nombre</label>

<div class="col-md-6">
    <input id="name" type="text" class="form-
control @error('name') is-invalid @enderror" name="name" value="{{ old('name')
}}" required autocomplete="name" autofocus>

    @error('name')
        <span class="invalid-feedback"
role="alert">

            <strong>{{ $message }}</strong>
        </span>
    @enderror
</div>
</div>

<div class="form-group row">
    <label for="email" class="col-md-4 col-form-label
text-md-right">Email</label>

    <div class="col-md-6">
        <input id="email" type="email" class="form-
control @error('email') is-invalid @enderror" name="email" value="{{
old('email') }}" required autocomplete="email">

        @error('email')
            <span class="invalid-feedback"
role="alert">

                <strong>{{ $message }}</strong>
            </span>
        @enderror
    </div>
</div>

<div class="form-group row">
    <label for="password" class="col-md-4 col-form-
label text-md-right">Password</label>

    <div class="col-md-6">
        <input id="password" type="password"
class="form-control @error('password') is-invalid @enderror" name="password"
required autocomplete="new-password">

        @error('password')
            <span class="invalid-feedback"
role="alert">

                <strong>{{ $message }}</strong>
            </span>

```

```
@enderror
    </div>
</div>

<div class="form-group row">
    <label for="password-confirm" class="col-md-4 col-
form-label text-md-right">Confirmar Password</label>

        <div class="col-md-6">
            <input id="password-confirm" type="password"
class="form-control" name="password_confirmation" required autocomplete="new-
password">

        </div>
    </div>

    <div class="form-group row mb-0">
        <div class="col-md-6 offset-md-4">
            <a class="btn btn-secondary"
href="{{route('admin.users.index')}}" role="button">Cancelar</a>
            <button type="submit" class="btn btn-primary">
                Añadir
            </button>
        </div>
    </div>
</form>
</div>
</div>
</div>
</div>
@endsection
```

### 1.13 Email

El usuario que se acaba de registrar debe recibir un email con los datos de registro.

### 1.13.1 Configuración Laravel

Vamos a configurar nuestro proyecto Laravel para que el envío de email se haga mediante servidor SMTP de vuestra cuenta de GMAIL, ya que ofrece de forma gratuita este servicio a sus usuarios.

Archivo .ENV

Una vez obtenidos los datos de nuestro servidor SMTP que nos proporciona gmail, configuramos el archivo `.ENV` de nuestro proyecto Laravel.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=nerom24@gmail.com
```

```
MAIL_PASSWORD=fvkphrcuunqfvemv
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="{APP_NAME}"
```

## Mail.php

A continuación en la carpeta `config` tenemos que modificar el fichero `mail.php` también con alguna de las opciones de configuración del servidor SMTP

```
'smtp' => [
    'transport' => 'smtp',
    'host' => env('MAIL_HOST', 'smtp.gmail.com'),
    'port' => env('MAIL_PORT', 587),
    'encryption' => env('MAIL_ENCRYPTION', 'tls'),
    'username' => env('MAIL_USERNAME'),
    'password' => env('MAIL_PASSWORD'),
    'timeout' => null,
],

...

'from' => [
    'address' => env('MAIL_FROM_ADDRESS', 'nerom24@gmail.com'),
    'name' => env('MAIL_FROM_NAME', 'DWES - Curso 19/20'),
],
```

### 1.13.2 Clase Mail

A continuación tenemos que crear una clase de tipo `Mail` la cuál se va a encargar tanto del envío como de su diseño del email.

Si el email que vamos a enviar contiene un mensaje de bienvenida después de un registro de usuario le podríamos llamar `MailRegister`, así para crear esta clase a partir de la clase `Mail` ejecutamos desde consola el siguiente comando:

```
php artisan make:mail MailRegister --markdown=emails.register
```

En ese comando también incluimos `--markdown=emails.register` esto procede a crear dentro de la carpeta `resources/views/email` el archivo `register.blade.php`, en este archivo se definirá el contenido del email, y prácticamente tendrá la misma funcionalidad que una vista aunque se trata de un markdown porque incluye una serie de comandos o componentes específicos.

Una vez ejecutado el comando anterior desde consola en la carpeta `App/Mail` se encontrará el archivo `MailRegister.php` que contendrá la clase que se encargará de gestionar nuestro email.

El contenido de `MailRegister.php`

```
-- MailRegister.php
```

```

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;
use App\User;

class MailRegister extends Mailable
{
    use Queueable, SerializesModels;

    private $user;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->markdown('emails.register' )->with('user', $this->user)
            ->from('nerom24@gmail.com', 'DWES - DAW 19/20')
            ->subject('Bienvenido a Laravel');
    }
}

```

El constructor de la clase `MailRegister` recibe un parámetro muy importante que es un objeto de la clase `User`, así los datos del usuario puedan ser usados para la personalización del mensaje cuyo contenido se definirá en el markdow `register.blade.php`.

### 1.13.3 Markdown

En la carpeta `resources/views/email` se ha creado el archivo `register.blade.php`, donde se definirá el contenido del mensaje.

```
-- register.blade.php
```

```

@component('mail::message')
# Proyecto Laravel Curso 19/20
# DWES 19/20 - ieslosremedios.org

Estimado {{ $user->name }}:

Gracias por haberse registrado en nuestra aplicación. Ha sido incluido
en nuestra lista de usuarios con perfil de usuario registrado o genérico.

Sus datos de acceso son:

@component('mail::table')
| Usuario | Email |
| ----- | ----- |
| {{ $user->name }} | {{ $user->email }} |
@endcomponent

Ahora deberá activar su cuenta mediante el siguiente enlace

@component('mail::button', ['url' => ''])
Button Text
@endcomponent

Thanks,<br>
{{ config('app.name') }}
@endcomponent

```

#### 1.13.4 Controladores

Ahora tendremos que actualizar dos controladores:

- RegisterController.php
- UserController.php

En ambos controladores se podrán añadir usuarios en la tabla User, el primero mediante un registro y el segundo mediante el panel de control.

RegisterControoler.php en la carpeta App/Http/Controllers/Auth

```

namespace App\Http\Controllers\Auth;
use App\Http\Controllers\Controller;
use App\Providers\RouteServiceProvider;
use App\User;
use App\Role;
use Illuminate\Foundation\Auth\RegistersUsers;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Facades\Mail;

```

```

use App\Mail\MailRegister;

...

protected function create(array $data)
{
    $user = User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);

    // Enviar email de confirmación al usuario registrado
    Mail::to($user->email)->send(new MailRegister($user));

    // Usuario que se acaba de registrar le asignamos perfil user
    $role = Role::select('id')->where('name','user')->first();
    $user->roles()->attach($role);

    return $user;
}

```

UserController.php en la carpeta App/Http/Controllers/Admin

```

namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use App\Mail\MailRegister;
use App\User;
use App\Role;
use Illuminate\Support\Facades\Gate;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Mail;
use Illuminate\Validation\Rule;

--

public function store(Request $request)
{
    // Control de acceso Gate
    if (Gate::denies('create-users')) {
        return redirect(route('admin.users.index'));
    }

    // Validación del formulario
    $validatedData = $request->validate(
        [
            'name' => ['required', 'string', 'max:255'],

```

```

        'email' => ['required', 'string', 'email', 'max:255',
'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ]);

    $user = User::create([
        'name' => $request['name'],
        'email' => $request['email'],
        'password' => Hash::make($request['password']),
    ]);

    // Usuario que se acaba de registrar le asignamos perfil user
    $role = Role::select('id')->where('name','user')->first();
    $user->roles()->attach($role);

    if($user) {
        $request->session()->flash('success','Usuario creado con éxito');
    }else {
        $request->session()->flash('error','Formulario no validado');
    }

    // Enviar email de confirmación al usuario registrado
    Mail::to($user->email)->send(new MailRegister($user));

    return redirect()->route('admin.users.index');
}

```

## 1.14 Búsquedas en Laravel

Laravel

Juan Carlos Moreno ▾

Logout  
Gestor Usuarios

Usuarios

Nuevo

#	Nombre	Correo	Roles	Acciones
1	Juan Carlos Moreno	nerom24@gmail.com	admin	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
2	Administrador Name	admin@gmail.com	admin	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
3	Author Name	author@gmail.com	author	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>
4	User Name	user@gmail.com	user	<input type="button" value="Editar"/> <input type="button" value="Eliminar"/>

Página 1 / 1

Total Usuarios: 4

Se trata de habilitar la opción de buscar usuarios en nuestro proyecto laravel.



### 1.14.1 Descripción del proceso de búsqueda

El proceso consistirá en escribir en el formulario de búsqueda una expresión de búsqueda y nuestra aplicación deberá mostrar aquellos registros coincidentes con dicha expresión.

Para seleccionar los registros coincidentes con la expresión de búsqueda debe tener en cuenta las columnas `id`, `name` y `email` de la tabla `users` así como la columna `nombre` de la tabla `roles`

### 1.14.2 Menu.blade.php

Ahora como vamos a dar funcionalidad al formulario buscar de la vista `index.blade.php` de la carpeta `views/admin/users`. El problema es que el menú se encuentra en un `partials` que se incluye en el archivo anterior con una directiva `@include` por lo que en realidad tendremos que modificar el archivo `menu.blade.php` que se encuentra en la carpeta `views/admin/users/partials`

Las modificaciones que tenemos que hacer en dicho archivo son las siguientes

```
-- menu.blade.php

<nav class="navbar navbar-expand-lg navbar">

    <div class="collapse navbar-collapse" id="navbarSupportedContent">

        <ul class="navbar-nav mr-auto">

            <li class="nav-item active">

                <a class="nav-link" href="{route('admin.users.create')}}" title="Nuevo usuario">Nuevo <span class="sr-only">(current)</span></a>

            </li>

            {{-- <li class="nav-item">

                <a class="nav-link" href="#">Link</a>

            </li> --}}

        </ul>

        <form class="form-inline my-2 my-lg-0" action="{route('admin.users.index')}}" method="GET">

            <input class="form-control mr-sm-2" type="search" placeholder="Search" name="search" aria-label="Search">

            <button class="btn btn-outline-secondary my-2 my-sm-0" type="submit">Buscar</button>

        </form>

    </div>

</nav>
```

### 1.14.3 Modelo User.php

El modelo de usuario **user.php** debemos añadir un método `scope` que se usa precisamente para hacer filtrados sobre los modelos de nuestro proyecto.

#### Reglas de scope

- Todos los scopes deben recibir la variable `$query`.
- Todos los nombres de los scopes deben comenzar con la palabra «scope» y luego el nombre que queramos utilizar para invocarlo.
- Cuando queremos utilizar el scope no utilizamos `User::scopeSearch()`, sino que hacemos `User::search()`. Por lo tanto, nuestro IDE no puede detectar el scope y no aparece en la lista de métodos disponibles cuando hacemos `Ctrl + space` (en el caso de PhpStorm y otros IDEs).

Este es el método scope que tengo que añadir al model `user.php` que se encuentra en la carpeta `App` de nuestro proyecto.

```
# Habilitar las búsquedas en el panel de control de usuarios
public function scopeSearch($query, $search) {

    if ($search) {

        return $query->where('id', 'LIKE', "%$search%")
            ->orWhere('name', 'LIKE', "%$search%")
            ->orWhere('email', 'LIKE', "%$search%");

    }

}
```

#### 1.14.4 UsersController.php

En la carpeta `App/Http/Controllers/Admin` está el controlador `UserController.php`, en el método `index()` debo filtrar el acceso a los usuarios, para ello he de recibir como parámetro `$request` de esta forma puedo acceder con `$request->search` al valor que ha escrito el usuario en la casilla de búsqueda.

```
-- UsersController.php

public function index(Request $request) {
    $users= User::search($request->search)->paginate(8);
    return view('admin.users.index', compact('users'));
}
```