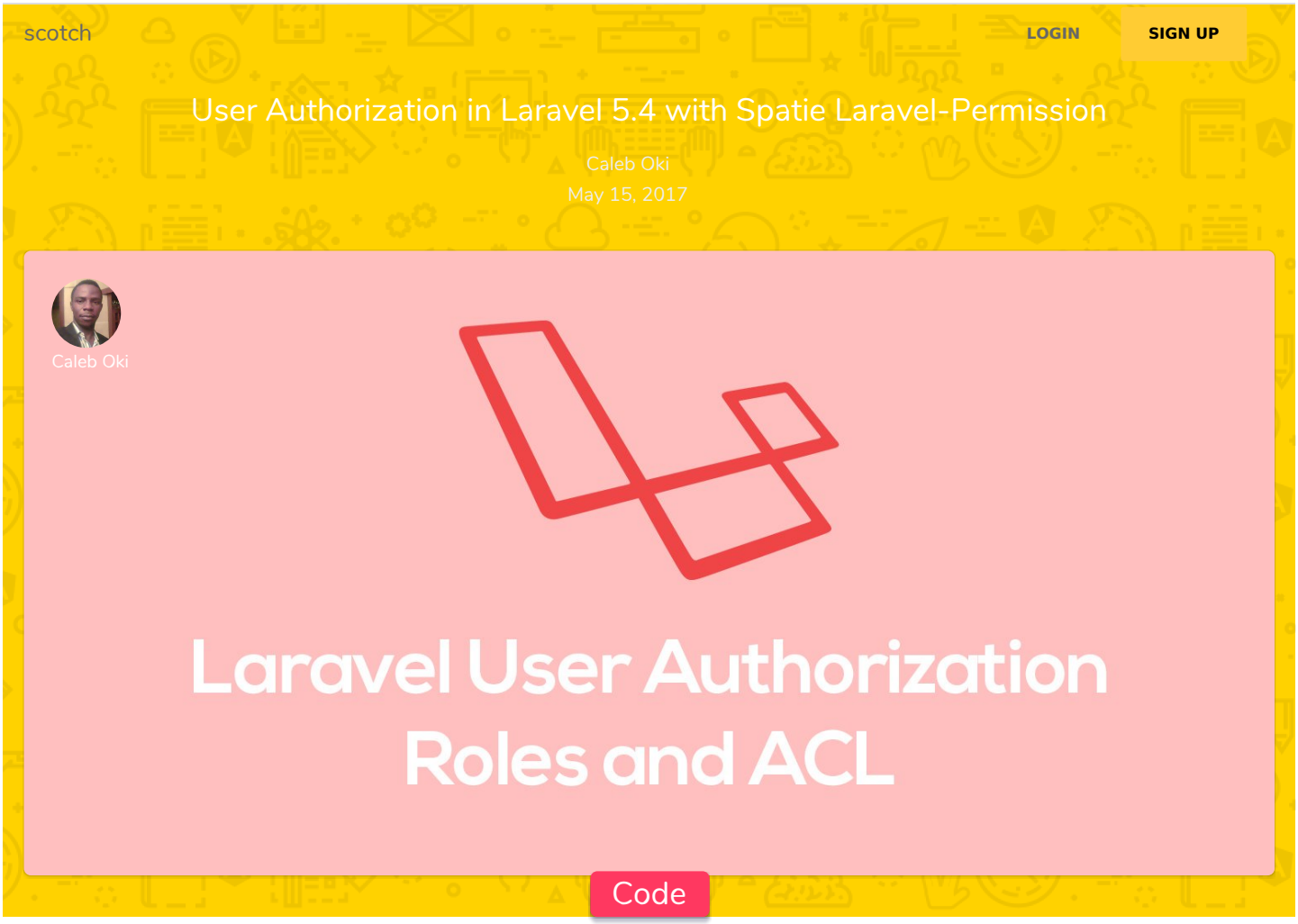

What We'll Build

When building an application, we often need to set up an access control list (ACL). An ACL specifies the level of permission granted to a user of an application. For example a user John may have the permission to read and write to a resource while another user Smith may have the permission only to read the resource.

In this tutorial, I will teach you how to add access control to a Laravel app usingLaravel-permission package. For this tutorial we will build a simple blog application where users can be assigned different levels of permission. Our user admin page will look like this:

Why Use Laravel-Permission

The Laravel-Permission package is built on top of Laravel's authorization features introduced in the 5.1.1 release. Although there are other packages that claim to offer similar functionalities, none of them have the same level of activity and maintenance as the laravel-permission package.

Development Environment and Installation

You can get Laravel up and running by first downloading the installer

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

Then add `$HOME/.composer/vendor/bin` to your $PATH so the `laravel` executable can be located by your system. Now you can install the latest stable version of Laravel by running

```
laravel new
```

To install the laravel-permission package run

```
composer require spatie/laravel-permission
```

Next include the package to our list of service providers, in `config/app.php` add `Spatie\Permission\PermissionServiceProvider::class` so our file looks like this

```
'providers' => [
    ...
    Spatie\Permission\PermissionServiceProvider::class,
];
```

Next publish the migration file for this package with the command

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider" --tag="migrations"
```

Database Setup and Migrations

Next create the database and update the `.env` file to include the database information. For example, for this tutorial the database information section of the `.env` looks like this:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=acl4
DB_USERNAME=root
DB_PASSWORD=
```

To build the tables, run

```
php artisan migrate
```

Please note that in Laravel 5.4 the default character set is changed to utf8mb4, therefore if you are running MariaDB or MYSQL version lower than 5.7.7 you may get this error when trying to run migration files

```
[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))

[PDOException]
SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long; max key length is 767 bytes
```

To fix this error edit the `app\Providers\AppServiceProvider.php` file, setting the default string length in the boot method

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

public function boot()
{
    Schema::defaultStringLength(191);
}
```

After that run the migration again. If it works as normal you would find the following tables in your database:

- `migrations` : This keeps track of migration process that have ran
- `users` : This holds the users data of the application
- `password_resets` : Holds token information when users request a new password
- `permissions` : This holds the various permissions needed in the application
- `roles` : This holds the roles in our application
- `role_has_permission` : This is a pivot table that holds relationship information between the permissions table and the role table
- `user_has_roles` : Also a pivot table, holds relationship information between the roles and the users table.
- `user_has_permissions` : Also a pivot table, holds relationship information between the users table and the permissions table.

Publish the configuration file for this package by running

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider" --tag="config"
```

The config file allows us to set the location of the Eloquent model of the permission and role class. You can also manually set the table names that should be used to retrieve your roles and permissions. Next we need to add the `HasRoles` trait to the User model:

```
use Illuminate\Foundation\Auth\User as Authenticatable;
use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable {
    use HasRoles;

    // ...
}
```

Laravel Collective HTML Form builder

Next install Laravel Collective HTML Form builder as this will be useful further on when we are creating our forms:

```
composer require laravelcollective/html
```

Then add your new provider to the providers array of `config/app.php`:

```
'providers' => [
    ...
    Collective\Html\HtmlServiceProvider::class,
];
```

Finally, add two class aliases to the aliases array of `config/app.php`:

```
'aliases' => [
    // ...
    'Form' => Collective\Html\FormFacade::class,
    'Html' => Collective\Html\HtmlFacade::class,
    // ...
    ],
```

That's all the installation and configuration needed. A role can be created like a regular Eloquent model, like this:

```
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

$role = Role::create(['name' => 'writer']);
$permission = Permission::create(['name' => 'edit articles']);
```

You can also get the permissions associated to a user like this:

```
$permissions = $user->permissions;
```

And using the pluck method, `pluck()` you can get the role names associated with a user like this:

```
$roles = $user->roles()->pluck('name');
```

Other methods available to us include:

- `givePermissionTo()` : Allows us to give persmission to a user or role
- `revokePermissionTo()` : Revoke permission from a user or role
- `hasPermissionTo()` : Check if a user or role has a given permission
- `assignRole()` : Assigns role to a user
- `removeRole()` : Removes role from a user
- `hasRole()` : Checks if a user has a role
- `hasAnyRole(Role::all())` : Checks if a user has any of a given list of roles
- `hasAllRoles(Role::all())` : Checks if a user has all of a given list of role

The methods `assignRole` , `hasRole` , `hasAnyRole` , `hasAllRoles` and `removeRole` can accept a string, a `Spatie\Permission\Models\Role-object` or an `\Illuminate\Support\Collection` object. The `givePermissionTo` and `revokePermissionTo` methods can accept a string or a `Spatie\Permission\Models\Permission` object.

Laravel-Permission also allows to use Blade directives to verify if the logged in user has all or any of a given list of roles:

```
@role('writer')
    I'm a writer!
@else
    I'm not a writer...
@endrole

@hasrole('writer')
    I'm a writer!
@else
    I'm not a writer...
@endhasrole

@hasanyrole(Role::all())
    I have one or more of these roles!
@else
    I have none of these roles...
@endhasanyrole

@hasallroles(Role::all())
    I have all of these roles!
@else
    I don't have all of these roles...
@endhasallroles
```

The Blade directives above depends on the users role. Sometimes we need to check directly in our view if a user has

a certain permission. You can do that using Laravel's native `@can` directive:

```
@can('Edit Post')
    I have permission to edit
@endcan
```

Controllers, Authentication and Views

You will need a total of four controllers for this application. Let's use resource controllers, as this automatically adds stub methods for us. Our controllers will be called

1. PostController
2. UserController
3. RoleController
4. PermissionController

Before working on these controllers let's create our authentication system. With one command Laravel provides a quick way to scaffold all of the routes and views needed for authentication.

```
php artisan make:auth
```

After running this command you would notice two new links for user login and registration in the home page.

This command also creates a `HomeController` (you can delete this as it won't be needed), a `resources/views/layouts/app.blade.php` file which contains markup that would be shared by all our views and an `app/Http/Controllers/Auth` directory which contains the controllers for registration and login. Switch into this directory and open the `RegisterController.php` file. Remove the `bcrypt` function in the `create` method, so the the method looks like this

```
protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => $data['password'],
    ]);
}
```

Instead let's define a mutator in `app\User.php` which would encrypt all our password fields. In `app\User.php` add this method:

```
public function setPasswordAttribute($password)
{
    $this->attributes['password'] = bcrypt($password);
}
```

This would provide the same functionality as before but now you don't need to write the `bcrypt` function when dealing with the password field in subsequent controllers.

Also in the `RegisterController.php` file. Change the `$redirectTo` property to:

```
protected $redirectTo = '/';
```

Do the same thing in the `LoginController.php` file.

Since the `HomeController` has been deleted our users are now redirected to the home page which would contain a list of our blog posts.

Next let's edit the `resources/views/layouts/app.blade.php` file to include: an extra drop-down 'Admin' link to view all users and an errors file which checks if our form produced any error. The 'Admin' link would only be viewed by users with the 'Admin' Role. We would also create a custom `styles.css` which would have extra styling for our `resources/views/posts/index.blade.php` view. The styling is just a paragraph in the teaser of our index view, the file should be located in `public/css/styles.css`

```
{{-- resources/views/layouts/app.blade.php --}}
<!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">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1">

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

    <title>{{ config('app.name', 'Laravel') }}</title>

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

    <link href="{{ asset('css/styles.css') }}" rel="stylesheet">

    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384


    <!-- Scripts -->
    <script>
        window.Laravel = {!! json_encode([
            'csrfToken' => csrf_token(),
        ]) !!};
    </script>
    <script src="https://use.fontawesome.com/9712be8772.js"></script>
</head>
<body>
    <div id="app">
        <nav class="navbar navbar-default navbar-static-top">
            <div class="container">
                <div class="navbar-header">

                    <!-- Collapsed Hamburger -->
                    <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#app-navbar

                        <span class="sr-only">Toggle Navigation</span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>

                    <!-- Branding Image -->
                    <a class="navbar-brand" href="{{ url('/') }}">
                        {{ config('app.name', 'Laravel') }}
                    </a>
                </div>

                <div class="collapse navbar-collapse" id="app-navbar-collapse">
                    <!-- Left Side Of Navbar -->
                    <ul class="nav navbar-nav">
                        <li><a href="{{ url('/') }}">Home</a></li>
                        @if (!Auth::guest())
                            <li><a href="{{ route('posts.create') }}">New Article</a></li>
                         @endif
                    </ul>

                    <!-- Right Side Of Navbar -->
                    <ul class="nav navbar-nav navbar-right">
                        <!-- Authentication Links -->
                        @if (Auth::guest())
                            <li><a href="{{ route('login') }}">Login</a></li>
                            <li><a href="{{ route('register') }}">Register</a></li>
                        @else
                            <li class="dropdown">
                                <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-expanded="

                                    {{ Auth::user()->name }} <span class="caret"></span>
                                </a>
```

```
                                <ul class="dropdown-menu" role="menu">
                                    <li>
                                        @role('Admin') {{-- Laravel-permission blade helper --}}
                                        <a href="#"><i class="fa fa-btn fa-unlock"></i>Admin</a>
                                        @endrole
                                        <a href="{{ route('logout') }}" onclick="event.preventDefault(); document.getElem

                                            Logout
                                        </a>

                                        <form id="logout-form" action="{{ route('logout') }}" method="POST" style="displa

                                            {{ csrf_field() }}
                                        </form>
                                    </li>
                                </ul>
                            </li>
                        @endif
                    </ul>
                </div>
            </div>
        </nav>

        @if(Session::has('flash_message'))
            <div class="container">
                <div class="alert alert-success"><em> {!! session('flash_message') !!}</em>
                </div>
            </div>
        @endif

        <div class="row">
            <div class="col-md-8 col-md-offset-2">
                @include ('errors.list') {{-- Including error file --}}
            </div>
        </div>

        @yield('content')

    </div>

    <!-- Scripts -->
    <script src="{{ asset('js/app.js') }}"></script>
</body>
</html>
```

The error file is:

```
{{-- resources\views\errors\list.blade.php --}}
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

and the `styles.css` file is simply:

```
p.teaser {
    text-indent: 30px;
    }
```

Post Controller

First, let's create the migration and model files for the `PostController`

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

This command generates a migration file in `app/database/migrations` for generating a new MySQL table named posts in our database and a model file `Post.php` in the `app` directory. Let's edit the migration file to include title and body fields of our post. Add a title and body field so the migration file looks like this:

```php
<?php
//database\migrations\xxxx_xx_xx_xxxxxx_create_posts_table.php

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

class CreatePostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->increments('id');
            $table->string('title');
            $table->text('body');
            $table->timestamps();
        });
    }

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

After saving the file, run migration again

```
php artisan migrate
```

You can now check the database for the `post` table and columns.

Next make the title and body field of the `Post` model mass assignable

```
namespace App;
use Illuminate\Database\Eloquent\Model;

class Post extends Model {
    protected $fillable = [
        'title', 'body'
    ];
}
```

Now let's generate our resource controller.

```
php artisan make:controller PostController --resource
```

This will create our controller with all the stub methods needed. Edit this file to look like this

```
<?php
// app/Http/Controllers/PostController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Post;
use Auth;
use Session;

class PostController extends Controller {

    public function __construct() {
        $this->middleware(['auth', 'clearance'])->except('index', 'show');
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */

    public function index() {
        $posts = Post::orderby('id', 'desc')->paginate(5); //show only 5 items at a time in descending order

        return view('posts.index', compact('posts'));
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
```

```
     */
    public function create() {
        return view('posts.create');
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request) {

    //Validating title and body field
        $this->validate($request, [
            'title'=>'required|max:100',
            'body' =>'required',
            ]);

        $title = $request['title'];
        $body = $request['body'];

        $post = Post::create($request->only('title', 'body'));

    //Display a successful message upon save
        return redirect()->route('posts.index')
            ->with('flash_message', 'Article,
             '. $post->title.' created');
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id) {
        $post = Post::findOrFail($id); //Find post of id = $id

        return view ('posts.show', compact('post'));
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id) {
        $post = Post::findOrFail($id);

        return view('posts.edit', compact('post'));
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id) {
        $this->validate($request, [
            'title'=>'required|max:100',
            'body'=>'required',
```

```
        ]);

        $post = Post::findOrFail($id);
        $post->title = $request->input('title');
        $post->body = $request->input('body');
        $post->save();

        return redirect()->route('posts.show',
            $post->id)->with('flash_message',
            'Article, '. $post->title.' updated');

    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id) {
        $post = Post::findOrFail($id);
        $post->delete();

        return redirect()->route('posts.index')
            ->with('flash_message',
             'Article successfully deleted');

    }
}
```

Here the `Post` class was imported from our model and the `Auth` class which was generated with the `make:auth` command earlier. These were imported so that you would be able to make Eloquent queries on the `Post` table and so as to be able to have access to authentication information of our users. In the constructor two middlewares were called, one is `auth` which restricts access to the `PostController` methods to authenticated users the other is a custom middleware is yet to be created. This would be responsible for our Permissions and Roles system. Next, `index` and `show` are passed into the except method to allow all users to be able to view posts.

The `index()` method lists all the available posts. It queries the `post` table for all posts and passes this information to the view. `Paginate()` allows us to limit the number of posts in a page, in this case five.

The `create()` method simply returns the `posts/create` view which would contain a form for creating new posts. The `store()` method saves the information input from the `posts/create` view. The information is first validated and after it is saved, a flash message is passed to the view `posts/index`.

Our `show()` method of the `PostController` allows us to display a single post. This method takes the `post` id as an argument and passes it to the method `Post::find()`. The result of the query is then sent to our `posts/show` view.

The `edit()` method, similar to the `create()` method simply returns the `posts/edit` view which would contain a form for creating editing posts. The `update()` method takes the information from the `posts/edit` view and updates the record. The `destroy()` method let's us delete a post.

Now that you have the `PostController` you need to set up the routes. Edit your `app/routes/web.php` file to look like this:

```php
<?php

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

Auth::routes();

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

Route::resource('users', 'UserController');

Route::resource('roles', 'RoleController');

Route::resource('permissions', 'PermissionController');

Route::resource('posts', 'PostController');
```

The `/` route is the route to our home page, here it was renamed to `home` The `Auth` route was generated when you ran the `make:auth` command. It handles authentication related routes. The other four routes are for resources that would be created later.

Post Views

Only four views are needed for our `PostController`. Create the files `\resources\views\posts\index.blade.php`, `\resources\views\posts\create.blade.php`, `\resources\views\posts\show.blade.php`, `\resources\views\posts\edit.blade.php`

Edit the `index.blade.php` file to look like this

```
@extends('layouts.app')
@section('content')
    <div class="container">
        <div class="row">
            <div class="col-md-10 col-md-offset-1">
                <div class="panel panel-default">
                    <div class="panel-heading"><h3>Posts</h3></div>
                    <div class="panel-heading">Page {{ $posts->currentPage() }} of {{ $posts->lastPage() }}</div>
                    @foreach ($posts as $post)
                        <div class="panel-body">
                            <li style="list-style-type:disc">
                                <a href="{{ route('posts.show', $post->id ) }}"><b>{{ $post->title }}</b><br>
                                    <p class="teaser">
                                       {{  str_limit($post->body, 100) }} {{-- Limit teaser to 100 characters --}}
                                    </p>
                                </a>
                            </li>
                        </div>
                    @endforeach
                    </div>
                    <div class="text-center">
                        {!! $posts->links() !!}
                    </div>
                </div>
            </div>
        </div>
@endsection
```

Notice that this file extends `views\layouts\app.php` file, which was generated earlier by the `make:auth` command.

The `create.blade.php` file looks like this

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

@section('title', '| Create New Post')

@section('content')
    <div class="row">
        <div class="col-md-8 col-md-offset-2">

        <h1>Create New Post</h1>
        <hr>

    {{-- Using the Laravel HTML Form Collective to create our form --}}
        {{ Form::open(array('route' => 'posts.store')) }}

        <div class="form-group">
            {{ Form::label('title', 'Title') }}
            {{ Form::text('title', null, array('class' => 'form-control')) }}
            <br>

            {{ Form::label('body', 'Post Body') }}
            {{ Form::textarea('body', null, array('class' => 'form-control')) }}
            <br>

            {{ Form::submit('Create Post', array('class' => 'btn btn-success btn-lg btn-block')) }}
            {{ Form::close() }}
        </div>
        </div>
    </div>

@endsection
```

The `show` view looks like this:

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

@section('title', '| View Post')

@section('content')

<div class="container">

    <h1>{{ $post->title }}</h1>
    <hr>
    <p class="lead">{{ $post->body }} </p>
    <hr>
    {!! Form::open(['method' => 'DELETE', 'route' => ['posts.destroy', $post->id] ]) !!}
    <a href="{{ url()->previous() }}" class="btn btn-primary">Back</a>
    @can('Edit Post')
    <a href="{{ route('posts.edit', $post->id) }}" class="btn btn-info" role="button">Edit</a>
    @endcan
    @can('Delete Post')
    {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
    @endcan
    {!! Form::close() !!}

</div>

@endsection
```

Here the `can` directive checks if a user has the permission to Edit or Delete Posts, if so the Edit and Delete button will be displayed. If the user does not have these permissions, only the Back button would be displayed.

The `edit` view just displays a edit form that will be used to update records:

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

@section('title', '| Edit Post')

@section('content')
<div class="row">

    <div class="col-md-8 col-md-offset-2">

        <h1>Edit Post</h1>
        <hr>
            {{ Form::model($post, array('route' => array('posts.update', $post->id), 'method' => 'PUT')) }}
            <div class="form-group">
            {{ Form::label('title', 'Title') }}
            {{ Form::text('title', null, array('class' => 'form-control')) }}<br>

            {{ Form::label('body', 'Post Body') }}
            {{ Form::textarea('body', null, array('class' => 'form-control')) }}<br>

            {{ Form::submit('Save', array('class' => 'btn btn-primary')) }}

            {{ Form::close() }}
    </div>
    </div>
</div>

@endsection
```

If you visit the home page you would see this

User Controller

The `UserController` will handle displaying all users, creating of new users, editing users, assigning roles to users and deleting users. As before generate the controller by running

```
php artisan make:controller UserController --resource
```

Then replace the content of this file with:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\User;
use Auth;

//Importing laravel-permission models
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

//Enables us to output flash messaging
use Session;

class UserController extends Controller {

    public function __construct() {
        $this->middleware(['auth', 'isAdmin']); //isAdmin middleware lets only users with a //specific permission pe
rmission to access these resources
    }

    /**
    * Display a listing of the resource.
    *
    * @return \Illuminate\Http\Response
    */
    public function index() {
    //Get all users and pass it to the view
        $users = User::all();
        return view('users.index')->with('users', $users);
    }

    /**
    * Show the form for creating a new resource.
    *
    * @return \Illuminate\Http\Response
    */
    public function create() {
    //Get all roles and pass it to the view
        $roles = Role::get();
        return view('users.create', ['roles'=>$roles]);
    }

    /**
    * Store a newly created resource in storage.
    *
    * @param  \Illuminate\Http\Request  $request
    * @return \Illuminate\Http\Response
    */
    public function store(Request $request) {
    //Validate name, email and password fields
        $this->validate($request, [
            'name'=>'required|max:120',
            'email'=>'required|email|unique:users',
            'password'=>'required|min:6|confirmed'
        ]);

        $user = User::create($request->only('email', 'name', 'password')); //Retrieving only the email and password
data
```

```php
        $roles = $request['roles']; //Retrieving the roles field
    //Checking if a role was selected
        if (isset($roles)) {

            foreach ($roles as $role) {
            $role_r = Role::where('id', '=', $role)->firstOrFail();
            $user->assignRole($role_r); //Assigning role to user
            }
        }
    //Redirect to the users.index view and display message
        return redirect()->route('users.index')
            ->with('flash_message',
             'User successfully added.');
    }

    /**
    * Display the specified resource.
    *
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function show($id) {
        return redirect('users');
    }

    /**
    * Show the form for editing the specified resource.
    *
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function edit($id) {
        $user = User::findOrFail($id); //Get user with specified id
        $roles = Role::get(); //Get all roles

        return view('users.edit', compact('user', 'roles')); //pass user and roles data to view

    }

    /**
    * Update the specified resource in storage.
    *
    * @param  \Illuminate\Http\Request  $request
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function update(Request $request, $id) {
        $user = User::findOrFail($id); //Get role specified by id

    //Validate name, email and password fields
        $this->validate($request, [
            'name'=>'required|max:120',
            'email'=>'required|email|unique:users,email,'.$id,
            'password'=>'required|min:6|confirmed'
        ]);
        $input = $request->only(['name', 'email', 'password']); //Retreive the name, email and password fields
        $roles = $request['roles']; //Retreive all roles
        $user->fill($input)->save();

        if (isset($roles)) {
            $user->roles()->sync($roles);  //If one or more role is selected associate user to roles
        }
        else {
            $user->roles()->detach(); //If no role is selected remove exisiting role associated to a user
        }
```

```
        return redirect()->route('users.index')
            ->with('flash_message',
             'User successfully edited.');
    }

    /**
    * Remove the specified resource from storage.
    *
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function destroy($id) {
    //Find a user with a given id and delete
        $user = User::findOrFail($id);
        $user->delete();

        return redirect()->route('users.index')
            ->with('flash_message',
             'User successfully deleted.');
    }
}
```

Here the `User` class, the `Role` class, the `Permission` class and the `Auth` class are imported. In the constructor the `auth` middleware is called to make sure only authenticated users have access to the User resource. A custom middleware `isAdmin` is also called. This checks if the authenticated user has administrator privileges. This middleware will be created later.

The `index()` method gets all users from the Users table and passes it to the index view which will display all users in a table. The `create()` method first gets all the Roles from the Roles table and passes it to the create view. This is so that Roles can be added when creating a User.

The `store()` method saves the input from the create view, after validating the input, looping through the Roles that was passed in the form and assigning these Roles to the User. The `show()` method just redirects back to the users page as for this demonstration, we wont need to show each user individually.

The `edit()` method gets the user corresponding to the id passed, then gets all `roles` and passes it to the edit view. The `update()` method validates data from the edit view and saves the updated name and password fields. It gets all `roles` from the `roles` table and while looping through them, removes any `role` assign to the user. It then takes the `role` data inputted from the form, matches them with the values in the databases and assigns these `roles` to the user.

The `destroy()` method allows us to delete a `user` along with it's corresponding `role`.

User Views

Three views are needed here: `index`, `create` and `edit` views. The `index` view would contain a table that lists all our `users` and their `roles`.

```
{{-- \resources\views\users\index.blade.php --}}
@extends('layouts.app')

@section('title', '| Users')

@section('content')

<div class="col-lg-10 col-lg-offset-1">
    <h1><i class="fa fa-users"></i> User Administration <a href="{{ route('roles.index') }}" class="btn btn-default pull-
Roles</a>
    <a href="{{ route('permissions.index') }}" class="btn btn-default pull-right">Permissions</a></h1>
    <hr>
    <div class="table-responsive">
        <table class="table table-bordered table-striped">

            <thead>
                <tr>
                    <th>Name</th>
                    <th>Email</th>
                    <th>Date/Time Added</th>
                    <th>User Roles</th>
                    <th>Operations</th>
                </tr>
            </thead>

            <tbody>
                @foreach ($users as $user)
                <tr>

                    <td>{{ $user->name }}</td>
                    <td>{{ $user->email }}</td>
                    <td>{{ $user->created_at->format('F d, Y h:ia') }}</td>
                    <td>{{  $user->roles()->pluck('name')->implode(' ') }}</td>{{-- Retrieve array of roles associat
ed to a user and convert to string --}}
                    <td>
                    <a href="{{ route('users.edit', $user->id) }}" class="btn btn-info pull-left" style="margin-right: 3p
Edit</a>

                    {!! Form::open(['method' => 'DELETE', 'route' => ['users.destroy', $user->id] ]) !!}
                    {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
                    {!! Form::close() !!}

                    </td>
                </tr>
                @endforeach
            </tbody>

        </table>
    </div>

    <a href="{{ route('users.create') }}" class="btn btn-success">Add User</a>

</div>

@endsection
```

The `create` view is just a form that allows us to create new `users` and assign `roles` to them.

```
{{-- \resources\views\users\create.blade.php --}}
@extends('layouts.app')

@section('title', '| Add User')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>

    <h1><i class='fa fa-user-plus'></i> Add User</h1>
    <hr>

    {{ Form::open(array('url' => 'users')) }}

    <div class="form-group">
        {{ Form::label('name', 'Name') }}
        {{ Form::text('name', '', array('class' => 'form-control')) }}
    </div>

    <div class="form-group">
        {{ Form::label('email', 'Email') }}
        {{ Form::email('email', '', array('class' => 'form-control')) }}
    </div>

    <div class='form-group'>
        @foreach ($roles as $role)
            {{ Form::checkbox('roles[]',  $role->id ) }}
            {{ Form::label($role->name, ucfirst($role->name)) }}<br>

        @endforeach
    </div>

    <div class="form-group">
        {{ Form::label('password', 'Password') }}<br>
        {{ Form::password('password', array('class' => 'form-control')) }}

    </div>

    <div class="form-group">
        {{ Form::label('password', 'Confirm Password') }}<br>
        {{ Form::password('password_confirmation', array('class' => 'form-control')) }}

    </div>

    {{ Form::submit('Add', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}

</div>

@endsection
```

The `edit` view is a form that allows us to edit `users` and their `roles`. Using Laravel's form model binding the form is automatically populated with the previous values.

```
{{-- \resources\views\users\edit.blade.php --}}

@extends('layouts.app')

@section('title', '| Edit User')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>

    <h1><i class='fa fa-user-plus'></i> Edit {{$user->name}}</h1>
    <hr>

    {{ Form::model($user, array('route' => array('users.update', $user->id), 'method' => 'PUT')) }}{{-- Form model bin
ding to automatically populate our fields with user data --}}

    <div class="form-group">
        {{ Form::label('name', 'Name') }}
        {{ Form::text('name', null, array('class' => 'form-control')) }}
    </div>

    <div class="form-group">
        {{ Form::label('email', 'Email') }}
        {{ Form::email('email', null, array('class' => 'form-control')) }}
    </div>

    <h5><b>Give Role</b></h5>

    <div class='form-group'>
        @foreach ($roles as $role)
            {{ Form::checkbox('roles[]',  $role->id, $user->roles ) }}
            {{ Form::label($role->name, ucfirst($role->name)) }}<br>

        @endforeach
    </div>

    <div class="form-group">
        {{ Form::label('password', 'Password') }}<br>
        {{ Form::password('password', array('class' => 'form-control')) }}

    </div>

    <div class="form-group">
        {{ Form::label('password', 'Confirm Password') }}<br>
        {{ Form::password('password_confirmation', array('class' => 'form-control')) }}

    </div>

    {{ Form::submit('Add', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}

</div>

@endsection
```

Permission Controller

Now let's tackle the `PermissionController` Create the file and paste the following code:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Auth;

//Importing laravel-permission models
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

use Session;

class PermissionController extends Controller {

    public function __construct() {
        $this->middleware(['auth', 'isAdmin']); //isAdmin middleware lets only users with a //specific permission pe
rmission to access these resources
    }

    /**
    * Display a listing of the resource.
    *
    * @return \Illuminate\Http\Response
    */
    public function index() {
        $permissions = Permission::all(); //Get all permissions

        return view('permissions.index')->with('permissions', $permissions);
    }

    /**
    * Show the form for creating a new resource.
    *
    * @return \Illuminate\Http\Response
    */
    public function create() {
        $roles = Role::get(); //Get all roles

        return view('permissions.create')->with('roles', $roles);
    }

    /**
    * Store a newly created resource in storage.
    *
    * @param  \Illuminate\Http\Request  $request
    * @return \Illuminate\Http\Response
    */
    public function store(Request $request) {
        $this->validate($request, [
            'name'=>'required|max:40',
        ]);

        $name = $request['name'];
        $permission = new Permission();
        $permission->name = $name;

        $roles = $request['roles'];
```

```php
        $permission->save();

        if (!empty($request['roles'])) { //If one or more role is selected
            foreach ($roles as $role) {
                $r = Role::where('id', '=', $role)->firstOrFail(); //Match input role to db record

                $permission = Permission::where('name', '=', $name)->first(); //Match input //permission to db record

                $r->givePermissionTo($permission);
            }
        }

        return redirect()->route('permissions.index')
            ->with('flash_message',
             'Permission'. $permission->name.' added!');

    }

    /**
    * Display the specified resource.
    *
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function show($id) {
        return redirect('permissions');
    }

    /**
    * Show the form for editing the specified resource.
    *
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function edit($id) {
        $permission = Permission::findOrFail($id);

        return view('permissions.edit', compact('permission'));
    }

    /**
    * Update the specified resource in storage.
    *
    * @param  \Illuminate\Http\Request  $request
    * @param  int  $id
    * @return \Illuminate\Http\Response
    */
    public function update(Request $request, $id) {
        $permission = Permission::findOrFail($id);
        $this->validate($request, [
            'name'=>'required|max:40',
        ]);
        $input = $request->all();
        $permission->fill($input)->save();

        return redirect()->route('permissions.index')
            ->with('flash_message',
             'Permission'. $permission->name.' updated!');

    }

    /**
    * Remove the specified resource from storage.
    *
```

```
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
     public function destroy($id) {
         $permission = Permission::findOrFail($id);

     //Make it impossible to delete this specific permission
     if ($permission->name == "Administer roles & permissions") {
             return redirect()->route('permissions.index')
             ->with('flash_message',
              'Cannot delete this Permission!');
         }

         $permission->delete();

         return redirect()->route('permissions.index')
             ->with('flash_message',
              'Permission deleted!');

     }
 }
```

In the `store()` method, we are making it possible for a `role` to be selected as a `permission` is created. After validating and saving the `permission` name field, a check is done if a `role` was selected if it was, a `permission` is assigned to the selected `role`.

Permission View

Three views are needed here as well. The `index` view would list in a table all the available permissions, the `create` view is a form which would be used to create a new `permission` and the edit view is a form that let's us edit existing `permission`.

```
{{-- \resources\views\permissions\index.blade.php --}}
@extends('layouts.app')

@section('title', '| Permissions')

@section('content')

<div class="col-lg-10 col-lg-offset-1">
    <h1><i class="fa fa-key"></i>Available Permissions

    <a href="{{ route('users.index') }}" class="btn btn-default pull-right">Users</a>
    <a href="{{ route('roles.index') }}" class="btn btn-default pull-right">Roles</a></h1>
    <hr>
    <div class="table-responsive">
        <table class="table table-bordered table-striped">

            <thead>
                <tr>
                    <th>Permissions</th>
                    <th>Operation</th>
                </tr>
            </thead>
            <tbody>
                @foreach ($permissions as $permission)
                <tr>
                    <td>{{ $permission->name }}</td>
                    <td>
                    <a href="{{ URL::to('permissions/'.$permission->id.'/edit') }}" class="btn btn-info pull-left" style=
Edit</a>

                    {!! Form::open(['method' => 'DELETE', 'route' => ['permissions.destroy', $permission->id] ]) !!
}
                    {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
                    {!! Form::close() !!}

                    </td>
                </tr>
                @endforeach
            </tbody>
        </table>
    </div>

    <a href="{{ URL::to('permissions/create') }}" class="btn btn-success">Add Permission</a>

</div>

@endsection
```

The following is the `create` view

```
{{-- \resources\views\permissions\create.blade.php --}}
@extends('layouts.app')

@section('title', '| Create Permission')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>

    <h1><i class='fa fa-key'></i> Add Permission</h1>
    <br>

    {{ Form::open(array('url' => 'permissions')) }}

    <div class="form-group">
        {{ Form::label('name', 'Name') }}
        {{ Form::text('name', '', array('class' => 'form-control')) }}
    </div><br>
    @if(!$roles->isEmpty()) //If no roles exist yet
        <h4>Assign Permission to Roles</h4>

        @foreach ($roles as $role)
            {{ Form::checkbox('roles[]',  $role->id ) }}
            {{ Form::label($role->name, ucfirst($role->name)) }}<br>

        @endforeach
    @endif
    <br>
    {{ Form::submit('Add', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}

</div>

@endsection
```

And finally the `edit` view:

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

@section('title', '| Edit Permission')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>

    <h1><i class='fa fa-key'></i> Edit {{$permission->name}}</h1>
    <br>
    {{ Form::model($permission, array('route' => array('permissions.update', $permission->id), 'method' => 'PUT')) }}{
{-- Form model binding to automatically populate our fields with permission data --}}

    <div class="form-group">
        {{ Form::label('name', 'Permission Name') }}
        {{ Form::text('name', null, array('class' => 'form-control')) }}
    </div>
    <br>
    {{ Form::submit('Edit', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}

</div>

@endsection
```

Role Controller

The `RoleController` is quite similar to the `UserController`. This controller will allow us to create `roles` and assign one or more `permissions` to a `role`. Create the file and paste the following code:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use Auth;
//Importing laravel-permission models
use Spatie\Permission\Models\Role;
use Spatie\Permission\Models\Permission;

use Session;

class RoleController extends Controller {

    public function __construct() {
        $this->middleware(['auth', 'isAdmin']);//isAdmin middleware lets only users with a //specific permission per
mission to access these resources
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
```

```
     */
    public function index() {
        $roles = Role::all();//Get all roles

        return view('roles.index')->with('roles', $roles);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create() {
        $permissions = Permission::all();//Get all permissions

        return view('roles.create', ['permissions'=>$permissions]);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request) {
    //Validate name and permissions field
        $this->validate($request, [
            'name'=>'required|unique:roles|max:10',
            'permissions' =>'required',
            ]
        );

        $name = $request['name'];
        $role = new Role();
        $role->name = $name;

        $permissions = $request['permissions'];

        $role->save();
    //Looping thru selected permissions
        foreach ($permissions as $permission) {
            $p = Permission::where('id', '=', $permission)->firstOrFail();
         //Fetch the newly created role and assign permission
            $role = Role::where('name', '=', $name)->first();
            $role->givePermissionTo($p);
        }

        return redirect()->route('roles.index')
            ->with('flash_message',
             'Role'. $role->name.' added!');
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id) {
        return redirect('roles');
    }

    /**
     * Show the form for editing the specified resource.
     *
```

```
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id) {
        $role = Role::findOrFail($id);
        $permissions = Permission::all();

        return view('roles.edit', compact('role', 'permissions'));
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id) {

        $role = Role::findOrFail($id);//Get role with the given id
    //Validate name and permission fields
        $this->validate($request, [
            'name'=>'required|max:10|unique:roles,name,'.$id,
            'permissions' =>'required',
        ]);

        $input = $request->except(['permissions']);
        $permissions = $request['permissions'];
        $role->fill($input)->save();

        $p_all = Permission::all();//Get all permissions

        foreach ($p_all as $p) {
            $role->revokePermissionTo($p); //Remove all permissions associated with role
        }

        foreach ($permissions as $permission) {
            $p = Permission::where('id', '=', $permission)->firstOrFail(); //Get corresponding form //permission in
db

            $role->givePermissionTo($p);  //Assign permission to role
        }

        return redirect()->route('roles.index')
            ->with('flash_message',
             'Role'. $role->name.' updated!');
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        $role = Role::findOrFail($id);
        $role->delete();

        return redirect()->route('roles.index')
            ->with('flash_message',
             'Role deleted!');

    }
}
```

Roles View

Three views are needed here as well. The `index` view to display available `roles` and associated `permissions`, the `create` view to add a new `role` and a view to edit an existing `role`. Create the `index.blade.php` file and paste the following:

```
{{-- \resources\views\roles\index.blade.php --}}
@extends('layouts.app')

@section('title', '| Roles')

@section('content')

<div class="col-lg-10 col-lg-offset-1">
    <h1><i class="fa fa-key"></i> Roles

    <a href="{{ route('users.index') }}" class="btn btn-default pull-right">Users</a>
    <a href="{{ route('permissions.index') }}" class="btn btn-default pull-right">Permissions</a></h1>
    <hr>
    <div class="table-responsive">
        <table class="table table-bordered table-striped">
            <thead>
                <tr>
                    <th>Role</th>
                    <th>Permissions</th>
                    <th>Operation</th>
                </tr>
            </thead>

            <tbody>
                @foreach ($roles as $role)
                <tr>

                    <td>{{ $role->name }}</td>

                    <td>{{ str_replace(array('[',']','""'),'', $role->permissions()->pluck('name')) }}</td>{{-- Retri
eve array of permissions associated to a role and convert to string --}}
                    <td>
                    <a href="{{ URL::to('roles/'.$role->id.'/edit') }}" class="btn btn-info pull-left" style="margin-righ
Edit</a>

                    {!! Form::open(['method' => 'DELETE', 'route' => ['roles.destroy', $role->id] ]) !!}
                    {!! Form::submit('Delete', ['class' => 'btn btn-danger']) !!}
                    {!! Form::close() !!}

                    </td>
                </tr>
                @endforeach
            </tbody>

        </table>
    </div>

    <a href="{{ URL::to('roles/create') }}" class="btn btn-success">Add Role</a>

</div>

@endsection
```

For the `create` view:

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

@section('title', '| Add Role')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>

    <h1><i class='fa fa-key'></i> Add Role</h1>
    <hr>

    {{ Form::open(array('url' => 'roles')) }}

    <div class="form-group">
        {{ Form::label('name', 'Name') }}
        {{ Form::text('name', null, array('class' => 'form-control')) }}
    </div>

    <h5><b>Assign Permissions</b></h5>

    <div class='form-group'>
        @foreach ($permissions as $permission)
            {{ Form::checkbox('permissions[]',  $permission->id ) }}
            {{ Form::label($permission->name, ucfirst($permission->name)) }}<br>

        @endforeach
    </div>

    {{ Form::submit('Add', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}

</div>

@endsection
```

And for the `edit` view:

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

@section('title', '| Edit Role')

@section('content')

<div class='col-lg-4 col-lg-offset-4'>
    <h1><i class='fa fa-key'></i> Edit Role: {{$role->name}}</h1>
    <hr>

    {{ Form::model($role, array('route' => array('roles.update', $role->id), 'method' => 'PUT')) }}

    <div class="form-group">
        {{ Form::label('name', 'Role Name') }}
        {{ Form::text('name', null, array('class' => 'form-control')) }}
    </div>

    <h5><b>Assign Permissions</b></h5>
    @foreach ($permissions as $permission)

        {{Form::checkbox('permissions[]',  $permission->id, $role->permissions ) }}
        {{Form::label($permission->name, ucfirst($permission->name)) }}<br>

    @endforeach
    <br>
    {{ Form::submit('Edit', array('class' => 'btn btn-primary')) }}

    {{ Form::close() }}
</div>

@endsection
```

Middleware

To restrict access to the `roles` and `permissions` page, a middleware was included called `isAdmin` in our `PermissionController` and `RoleController`. This middleware counts how many users are in the Users table, and if there are more than one users, it checks if the current authenticated User has the permission to 'Administer roles & permissions'. To create a permission visit `http://localhost:8000/permissions/create`. Then go to `http://localhost:8000/roles/create` to create a `role`, to which you can now assign the `permission` you created. For example you can create a permission called 'Administer roles & permissions' and a 'Admin' `role` to which you would assign this `permission`. Create the `AdminMiddleware` in the directory `app/Http/Middleware/` and enter the following code:

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;
use App\User;

class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        $user = User::all()->count();
        if (!($user == 1)) {
            if (!Auth::user()->hasPermissionTo('Administer roles & permissions')) //If user does //not have this per
mission
        {
                abort('401');
            }
        }

        return $next($request);
    }
}
```

A middleware called `clearance` was also included in our `PostController`. This middleware would check if a `user` has the `permissions` Administer roles & permissions, Create Post, Edit Post and Delete Post.

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class ClearanceMiddleware {
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next) {
        if (Auth::user()->hasPermissionTo('Administer roles & permissions')) //If user has this //permission
    {
            return $next($request);
        }

        if ($request->is('posts/create'))//If user is creating a post
         {
            if (!Auth::user()->hasPermissionTo('Create Post'))
         {
                abort('401');
            }
         else {
                return $next($request);
            }
        }

        if ($request->is('posts/*/edit')) //If user is editing a post
         {
            if (!Auth::user()->hasPermissionTo('Edit Post')) {
                abort('401');
            } else {
                return $next($request);
            }
        }

        if ($request->isMethod('Delete')) //If user is deleting a post
         {
            if (!Auth::user()->hasPermissionTo('Delete Post')) {
                abort('401');
            }
         else
         {
                return $next($request);
            }
        }

        return $next($request);
    }
}
```

Add `AdminMiddleware::class` and `ClearanceMiddleware::class` to the `$routeMiddleware` property of `/app/Http/kernel.php` like this:

```
protected $routeMiddleware = [
        'auth' => \Illuminate\Auth\Middleware\Authenticate::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'isAdmin' => \App\Http\Middleware\AdminMiddleware::class,
        'clearance' => \App\Http\Middleware\ClearanceMiddleware::class,
];
```

In both middelwares a 401 exception would be thrown if the conditions are not meet. Let's create a custom 401 error page:

```
{{-- \resources\views\errors\401.blade.php --}}
@extends('layouts.app')

@section('content')
    <div class='col-lg-4 col-lg-offset-4'>
        <h1><center>401<br>
        ACCESS DENIED</center></h1>
    </div>

@endsection
```

Wrapping Up

First lets create an 'Admin' user and then create the necessary permissions and roles. Click on Register and create a user, then go to `http://localhost:8000/permissions` and create permissions to `Create Post`, `Edit Post`, `Delete Post` and `Administer roles & permissions`. After creating these permissions, your permissions page should look like this:

Next, you need to create `roles` to which you would add the Create, Edit and Delete Permissions. Click on Roles and create these `roles`:

- Admin- A user assigned to this role would have all permissions
- Owner- A user assigned to this role would have selected permissions assigned to it by Admin

Finally assign the Role of 'Admin' to the currently logged in User. Click on Users and then Edit. Check the Admin box under Give Role:

After assigning the 'Admin' `role` to our `user`, notice that you now have a new Admin link in the drop of the navigation, this links to our users page. Now create a new `user` and give it the more restrictive `role` of Owner. If you login as this user and try to visit the User, Role or Permission pages you get this as expected:

The Owner `role` does not have permission to `Administer Roles & Users` hence the exception is thrown.

To demonstrate how this works for `posts`, create a `post` by clicking on New Article. After creating the post, view the `post` and you would notice you have along with the Back button, an Edit and Delete button as shown below:

Now if you logout and view the post only the Back button will be available to us. This also works if you have a logged in user who does not have permissions to Edit or Delete Post.

Conclusion

The laravel-permission package makes it relatively easy to build a role and permission system. To recap we have considered installation of the laravel-permission package, laravel-permission blade directives, creating a custom middleware and implementing an access control list in a Laravel application. You can look at the final product on Github and if you have any questions or comments, don't hesitate to post them below.

Caleb Oki

Laravel and Vue.js web developer

Latest Video Courses

See More Courses

Free

2.1 hours

Getting Started with Angular v4

1.4 hours

Getting Started with React

Free

4.7 hours

Getting Started with JavaScript for Web Development

0.9 hours

Get to Know Git

Join Scotch

High Quality Content

The best tutorials and content that you'll find for web development. Guides, courses, tutorials, and more great content to learn with.

Build Real Apps

We won't just go over concepts and "Hello Worlds"; we'll**build real apps together** that you can use at your job or for your portfolio.

Not Just How, But Why

There are many different ways to code the same project. We'll show**best practices** and why certain choices are better than others.

Scotch Free

- Write your own posts
- Watch free lessons
- Like favorite posts
- Bookmark content for reference
- Post in the forums

Free

Scotch Premium

- All of the free features
- Access to **all premium content**