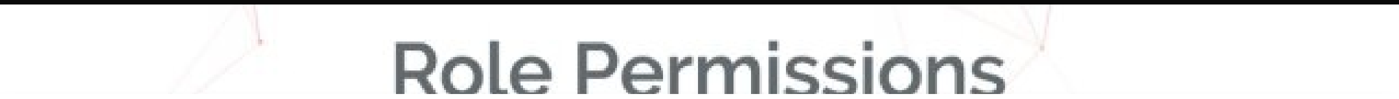


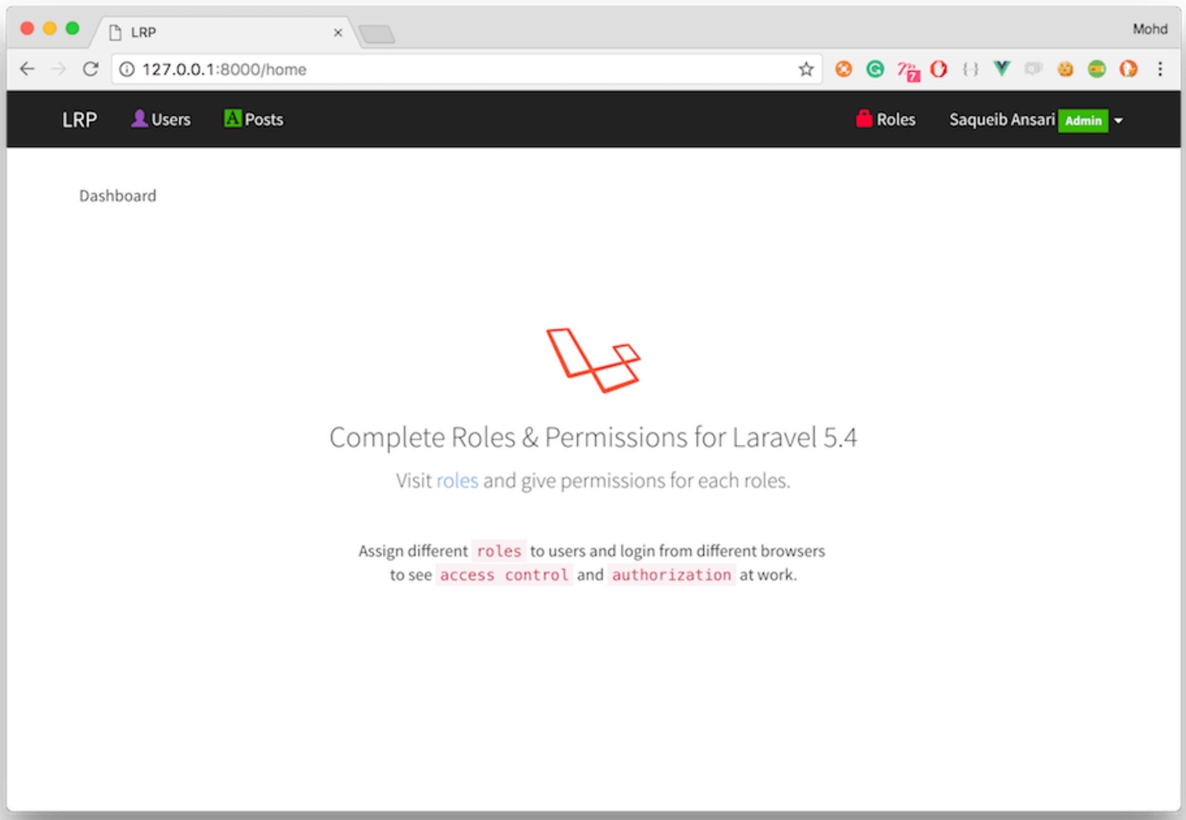
Easy roles and permissions in Laravel 5.4

By Mohd Saqueib May 1, 2017 under Laravel

Laravel comes with Authentication and Authorization out of the box, I have implemented many role and permissions based system in the past, using laravel, it's peace of cake. In this post, we are going to implement a fully working and extensible roles and permissions on laravel 5.4. When we finish we will have a starter kit which we can use for our any future project which needs roles and permissions based access control (ACL).

Laravel Permissions

Although laravel comes with Policies to handle the authorization but I wanted to have an option to just create permissions in the database which we can manage by a UI in the admin panel, pretty standard. we can implement our own role and permission from scratch but I am going to use spatie/laravel-permission package for this. This package was inspired by Jeffrey ways screencast and it's very well maintained and very easy to use. It has everything we need and plays very well with Laravel Gate and Policies implementations.



Complete Roles & Permissions for Laravel 5.4

Visit roles and give permissions for each roles.

Assign different `roles` to users and login from different browsers to see `access control` and `authorization` at work.

Source Code

Scaffold app

Let's get started by creating a fresh laravel app by running `laravel new rpl` or if you don't have laravel command line tool, you can use composer to get it `composer create-project --prefer-dist laravel/laravel rpl`. rpl is the name of app which is an abbreviation for Role Permissions Laravel.

Once the installation is done, make necessary changes in `.env` so you can connect to a database.

Setup packages

We will use laravel authorization which comes bundled, let's scaffold it using `auth:make` command. It will create a fully functional login, register and password reset features. Now that's done, let's pull packages we will need in this app.

Edit your `composer.json` to add below dependencies.

```
    "require": {
            ...
            "spatie/laravel-permission": "^2.1",
            "laracasts/flash": "^3.0",
            "laravelcollective/html": "^5.3.0"
        },
```

Apart from permissions package, I have also grabbed flash to show notification alerts and laravelcollective html to create forms with the option to model bind them.

Now Add them in **ServiceProvider** and in **aliases** array, open `config/app.php`

```
    'providers' => [

        ...

        Spatie\Permission\PermissionServiceProvider::class,

        Laracasts\Flash\FlashServiceProvider::class,

        Collective\Html\HtmlServiceProvider::class,

        ...

    ],


    'aliases' => [

        ...

        'Form' => Collective\Html\FormFacade::class,

        'Html' => Collective\Html\HtmlFacade::class,

    ]
```

Cool, now let’s start by publishing migration which comes from permissions package, which will create tables for roles and permissions.

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

Great, now we need to create our Resource, I am going to create **Post**, **Role**, **Permission** models with resource controller. **User** model is already present so we will use it.

```
 // Create Post model with migration and resource controller
php artisan make:model Post -m -c --resource

// Create Role model and resource controller
php artisan make:model Role -c --resource

// Create Permission model and resource controller
php artisan make:model Permission -c --resource
```

Spatie\Permission already have role and permissions model, we will just extend them in order to do any changes if needed in future.

```
 // Permission Model
class Permission extends \Spatie\Permission\Models\Permission { }

// Role Model
class Role extends \Spatie\Permission\Models\Role { }
```

We need to add HasRoles trait provided by the package to give the user all the power of laravel permissions.

```
 use Spatie\Permission\Traits\HasRoles;

class User extends Authenticatable
{
    use HasRoles;
    ...
```

Now we have all the boilerplate out of the way, let's create simple database seeder so we can build and test our app, you can read more about advance database seeding in this post, open `database/seeds/DatabaseSeeder.php` and add this code.

```
    public function run()
        {
            // Ask for db migration refresh, default is no
            if ($this->command->confirm('Do you wish to refresh migration before seeding, it w
                // Call the php artisan migrate:refresh
                $this->command->call('migrate:refresh');
                $this->command->warn("Data cleared, starting from blank database.");
            }

            // Seed the default permissions
            $permissions = Permission::defaultPermissions();

            foreach ($permissions as $perms) {
                Permission::firstOrCreate(['name' => $perms]);
            }

            $this->command->info('Default Permissions added.');

            // Confirm roles needed
            if ($this->command->confirm('Create Roles for user, default is admin and user? [y|

                // Ask for roles from input
                $input_roles = $this->command->ask('Enter roles in comma separate format.', 'A

                // Explode roles
                $roles_array = explode(',', $input_roles);

                // add roles
                foreach($roles_array as $role) {
                    $role = Role::firstOrCreate(['name' => trim($role)]);
                    $role = Role::firstOrCreate(['name' => trim($role)]);

                    if( $role->name == 'Admin' ) {
                        // assign all permissions
                        $role->syncPermissions(Permission::all());
                        $this->command->info('Admin granted all the permissions');
                    } else {
                        // for others by default only read access
                        $role->syncPermissions(Permission::where('name', 'LIKE', 'view_%')->ge
                    }

                    // create one user for each role
                    $this->createUser($role);
                }

                $this->command->info('Roles ' . $input_roles . ' added successfully');

            } else {
                Role::firstOrCreate(['name' => 'User']);
                $this->command->info('Added only default user role.');
            }

            // now lets seed some posts for demo
            factory(\App\Post::class, 30)->create();
            $this->command->info('Some Posts data seeded.');
            $this->command->warn('All done :)');
        }

        /**
         * Create a user with given role
         *
         * @param $role
         */
```

```
        private function createUser($role)
        {
            $user = factory(User::class)->create();
            $user->assignRole($role->name);


            if( $role->name == 'Admin' ) {
                $this->command->info('Here is your admin details to login:');
                $this->command->warn($user->email);
                $this->command->warn('Password is "secret"');
            }
        }
```

Now add `defaultPermissions` in Permissions Model which we have created.

```
    public static function defaultPermissions()
    {
        return [
            'view_users',
            'add_users',
            'edit_users',
            'delete_users',

            'view_roles',
            'add_roles',
            'edit_roles',
            'delete_roles',

            'view_posts',
            'add_posts',
            'edit_posts',
            'delete_posts',
        ];
    }
```

Post model factory contains only 2 fileds, **title** and **body.** You should setup the migration and factory. Now run the seeder using `php artisan db:seed` it should give an admin user which you can use to login.

Now you can login with admin user but there is no access control in place, I will create the **User** Resource first.

Create the **UserController** and add below code.

```
    public function index()
    {
        $result = User::latest()->paginate();
        return view('user.index', compact('result'));
    }
```

```php
    public function create()

    {

        $roles = Role::pluck('name', 'id');

        return view('user.new', compact('roles'));

    }


    public function store(Request $request)

    {

        $this->validate($request, [

            'name' => 'bail|required|min:2',

            'email' => 'required|email|unique:users',

            'password' => 'required|min:6',

            'roles' => 'required|min:1'

        ]);


        // hash password

        $request->merge(['password' => bcrypt($request->get('password'))]);


        // Create the user

        if ( $user = User::create($request->except('roles', 'permissions')) ) {

            $this->syncPermissions($request, $user);

            flash('User has been created.');

        } else {

            flash()->error('Unable to create user.');

        }


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

    }


    public function edit($id)

    {

        $user = User::find($id);
        $user = User::find($id);
        $roles = Role::pluck('name', 'id');
        $permissions = Permission::all('name', 'id');

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

    public function update(Request $request, $id)
    {
        $this->validate($request, [
            'name' => 'bail|required|min:2',
            'email' => 'required|email|unique:users,email,' . $id,
            'roles' => 'required|min:1'
        ]);

        // Get the user
        $user = User::findOrFail($id);

        // Update user
        $user->fill($request->except('roles', 'permissions', 'password'));

        // check for password change
        if($request->get('password')) {
            $user->password = bcrypt($request->get('password'));
        }

        // Handle the user roles
        $this->syncPermissions($request, $user);

        $user->save();
        flash()->success('User has been updated.');
        return redirect()->route('users.index');
    }
    public function destroy($id)

    {

        if ( Auth::user()->id == $id ) {

            flash()->warning('Deletion of currently logged in user is not allowed :(')->import

            return redirect()->back();

        }


        if( User::findOrFail($id)->delete() ) {

            flash()->success('User has been deleted');

        } else {

            flash()->success('User not deleted');

        }


        return redirect()->back();

    }


    private function syncPermissions(Request $request, $user)

    {

        // Get the submitted roles

        $roles = $request->get('roles', []);

        $permissions = $request->get('permissions', []);


        // Get the roles

        $roles = Role::find($roles);


        // check for current role changes

        if( ! $user->hasAllRoles( $roles ) ) {

            // reset all direct permissions for user

            $user->permissions()->sync([]);

        } else {

            // handle permissions

            $user->syncPermissions($permissions);
        }


        $user->syncRoles($roles);

        return $user;

    }
```

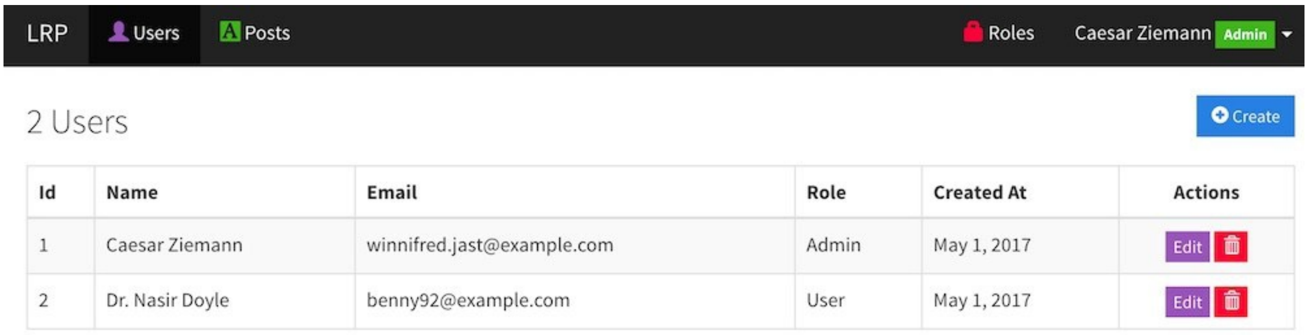
Everything is commented and self-explanatory, it's a simple controller to perform CRUD operation with user level permission override so you can give access to certain permissions for the specific user. Now register the route for all the resource controllers.

```
    Route::group( ['middleware' => ['auth']], function() {

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

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

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

    });
```

Go ahead and create the **PostController** by yourself, you can always access the source code if need help.

Authorization

This is the main part, authorization will be in 2 level, first is at the controller level and second in view level. In view, if you don't have permission to `add_users` then it doesn't make sense to show **Create** button. this can be done using `@can('add_users')` directive in blade template.

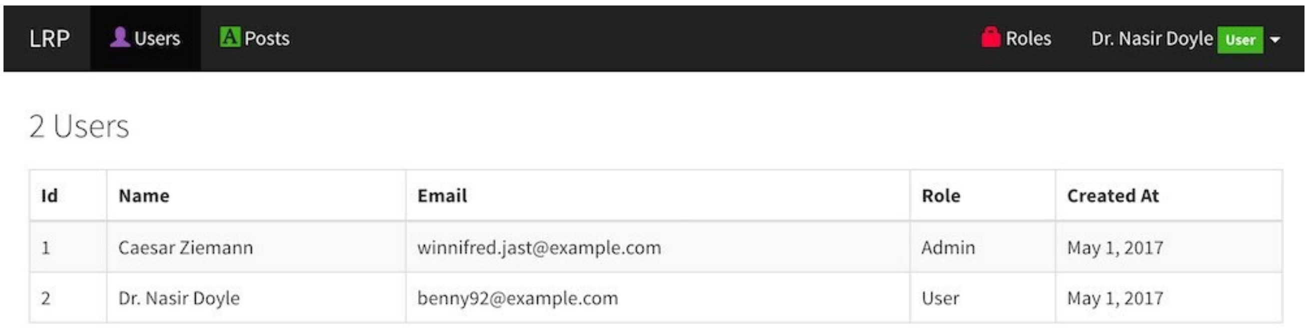


2 Users

Id	Name	Email	Role	Created At	Actions
1	Caesar Ziemann	winnifred.jast@example.com	Admin	May 1, 2017	Edit
2	Dr. Nasir Doyle	benny92@example.com	User	May 1, 2017	Edit

```
 ...
@can('add_users')
    <a href="{{ route('users.create') }}" class="btn btn-primary btn-sm">
        <i class="glyphicon glyphicon-plus-sign"></i> Create
    </a>
@endcan
...
```

Similarly, if you don't have access to `edit_users` you should not see the edit button or delete button in the table.



2 Users

Id	Name	Email	Role	Created At
1	Caesar Ziemann	winnifred.jast@example.com	Admin	May 1, 2017
2	Dr. Nasir Doyle	benny92@example.com	User	May 1, 2017

```
 <table class="table table-bordered table-striped table-hover" id="data-table">
    <thead>
    <tr>
        ....
        <th>Created At</th>
        @can('edit_users', 'delete_users')
            <th class="text-center">Actions</th>
        @endcan
    </tr>
    </thead>
    <tbody>
    @foreach($result as $item)
        <tr>
            ...
            @can('edit_users')
            <td class="text-center">
                // action buttons
            </td>
            @endcan
        </tr>
    @endforeach
    </tbody>
</table>
```

Now that's fine, but some malicious user can still directly visit the URL and he will be able to access the protected route. To prevent that we need protection on controller@method level.

Authorizable Trait

We can add `$user->can()` check in every method to handle authorization but it will make it more difficult to maintain and since we will be using this in multiple resource controller it will be good to extract out this logic in a trait which will handle the authorization automatically, sounds good! let's do it.

```php
namespace App;

trait Authorizable
{
    private $abilities = [
        'index' => 'view',
        'edit' => 'edit',
        'show' => 'view',
        'update' => 'edit',
        'create' => 'add',
        'store' => 'add',
        'destroy' => 'delete'
    ];

    /**
     * Override of callAction to perform the authorization before
     *
     * @param $method
     * @param $parameters
     * @return mixed
     */
    public function callAction($method, $parameters)
    {
        if( $ability = $this->getAbility($method) ) {
            $this->authorize($ability);
        }

        return parent::callAction($method, $parameters);
    }

    public function getAbility($method)
    {
        $routeName = explode('.', \Request::route()->getName());
```

```
            $action = array_get($this->getAbilities(), $method);


            return $action ? $action . '_' . $routeName[0] : null;

        }


        private function getAbilities()

        {

            return $this->abilities;

        }


        public function setAbilities($abilities)

        {

            $this->abilities = $abilities;

        }

    }
```

In this trait we have to override the **callAction** method which gets called by the router to trigger respective method on the resource controller, that's good place to check the permission, so we get the route name in users case it will be **users.index**, **users.store**, **users.update** etc.. we have mapped the abilities to our resource controller route naming conventions.

what happens is when a user visits route named `users.index`, it gets translated into `view_users` ability to check against in `authorize($ability)` method by getAbility() method, when user visits edit page route **users.edit** it will be translated as `edit_users` and so on. by extracting this logic in a Trait we will be able to apply authorization on any resource controller we wanted.

Add our trait on **UserController**.

```
 use App\Authorizable;


class UserController extends Controller

{

    use Authorizable;

    ...
```

That’s it, your controller is protected, the only user who has permissions to visit certain route can access it.

When user doesn’t have permission they get `AuthorizationException` exception, which is not very friendly to end user, let’s handle this in global **Handler** so we can display a notification and send the user back to dashboard if the try to visit some route which they don’t suppose to access.

AuthorizationException Exception Handler

Open the **app/Exceptions/Handler.php** and add this in render method

```
    public function render($request, Exception $exception)
    {
        if ($exception instanceof AuthorizationException) {
            return $this->unauthorized($request, $exception);
        }

        return parent::render($request, $exception);
    }

    private function unauthorized($request, Exception $exception)
    {
        if ($request->expectsJson()) {
            return response()->json(['error' => $exception->getMessage()], 403);
        }

        flash()->warning($exception->getMessage());
        return redirect()->route('home');
    }
```

This will redirect back to home route with flash notification if user doesn't have access to the action.

Role Management

Let's create the roles management resource controller. which admin can use to create the new role and give or change permission to them.

```
    class RoleController extends Controller
    {
        use Authorizable;

        public function index()
        {
            $roles = Role::all();
            $permissions = Permission::all();

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

        public function store(Request $request)
        {
            $this->validate($request, ['name' => 'required|unique:roles']);

            if( Role::create($request->only('name')) ) {
                flash('Role Added');
            }

            return redirect()->back();
        }

        public function update(Request $request, $id)
        {
            if($role = Role::findOrFail($id)) {
                // admin role has everything
                if($role->name === 'Admin') {
                    $role->syncPermissions(Permission::all());
                    return redirect()->route('roles.index');
                }

                $permissions = $request->get('permissions', []);
                $role->syncPermissions($permissions);
                flash( $role->name . ' permissions has been updated.');
            } else {
                flash()->error( 'Role with id '. $id .' note found.');
            }

            return redirect()->route('roles.index');
            return redirect()->route('roles.index');

        }

    }
```

Now lets add the views for role, create **resources/views/role/index.blade.php** with below markup.

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


@section('title', 'Roles & Permissions')


@section('content')

    <!-- Modal -->

    <div class="modal fade" id="roleModal" tabindex="-1" role="dialog" aria-labelledby="ro

        <div class="modal-dialog" role="document">

            {!! Form::open(['method' => 'post']) !!}


            <div class="modal-content">

                <div class="modal-header">

                    <button type="button" class="close" data-dismiss="modal" aria-label="C

                    <h4 class="modal-title" id="roleModalLabel">Role</h4>

                </div>

                <div class="modal-body">

                    <!-- name Form Input -->

                    <div class="form-group @if ($errors->has('name')) has-error @endif">

                        {!! Form::label('name', 'Name') !!}

                        {!! Form::text('name', null, ['class' => 'form-control', 'placehol

                        @if ($errors->has('name')) <p class="help-block">{{ $errors->first

                    </div>

                </div>

                <div class="modal-footer">

                    <button type="button" class="btn btn-default" data-dismiss="modal">Clo
                    <!-- Submit Form Button -->
                    {!! Form::submit('Submit', ['class' => 'btn btn-primary']) !!}
                </div>
                {!! Form::close() !!}
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-md-5">
            <h3>Roles</h3>
        </div>
        <div class="col-md-7 page-action text-right">
            @can('add_roles')
                <a href="#" class="btn btn-sm btn-success pull-right" data-toggle="modal"
            @endcan
        </div>
    </div>



    @forelse ($roles as $role)
        {!! Form::model($role, ['method' => 'PUT', 'route' => ['roles.update',  $role->id

        @if($role->name === 'Admin')
            @include('shared._permissions', [
                          'title' => $role->name .' Permissions',
                          'options' => ['disabled'] ])
        @else
            @include('shared._permissions', [
                          'title' => $role->name .' Permissions',
                          'model' => $role ])
            @can('edit_roles')
                {!! Form::submit('Save', ['class' => 'btn btn-primary']) !!}
```

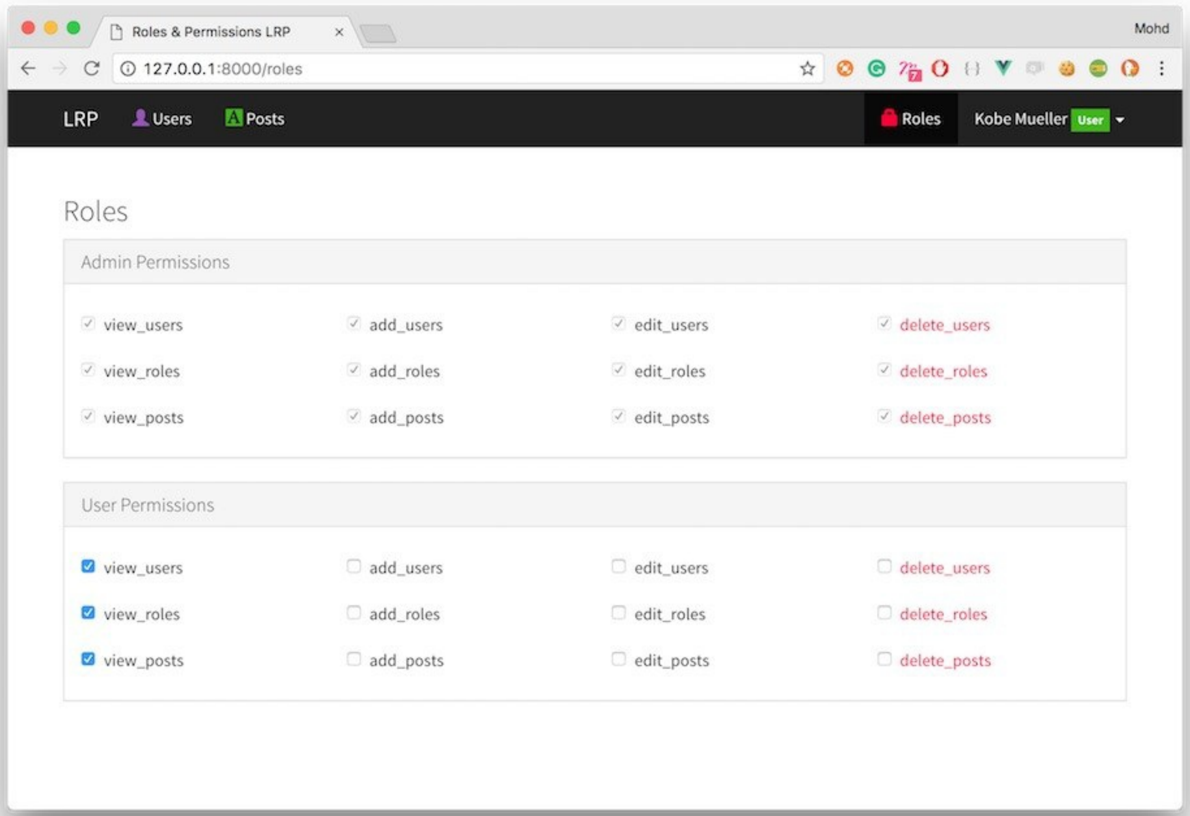
```
                {!! Form::submit('Save', ['class' => 'btn btn-primary']) !!}
            @endcan
        @endif

        {!! Form::close() !!}

    @empty
        <p>No Roles defined, please run <code>php artisan db:seed</code> to seed some dumm
    @endforelse
@endsection
```

Let’s add `resources/views/shared/_permissions.blade.php` template.

```
<div class="panel panel-default">
    <div class="panel-heading" role="tab" id="{{ isset($title) ? str_slug($title) :  'perm
        <h4 class="panel-title">
            <a role="button" data-toggle="collapse" data-parent="#accordion" href="#dd-{{
                {{ $title or 'Override Permissions' }} {!! isset($user) ? '<span class="te
            </a>
        </h4>
    </div>
    <div id="dd-{{ isset($title) ? str_slug($title) :  'permissionHeading' }}" class="pane
        <div class="panel-body">
            <div class="row">
                @foreach($permissions as $perm)
                    <?php
                        $per_found = null;

                        if( isset($role) ) {
                            $per_found = $role->hasPermissionTo($perm->name);
                        }
```

```
                                if( isset($user)) {
                                    $per_found = $user->hasDirectPermission($perm->name);
                                }
                            ?>

                            <div class="col-md-3">
                                <div class="checkbox">
                                    <label class="{{ str_contains($perm->name, 'delete') ? 'text-d
                                        {!! Form::checkbox("permissions[]", $perm->name, $per_foun
                                    </label>
                                </div>
                            </div>
                        @endforeach
                    </div>
                </div>
            </div>
        </div>
```

If you visit now `/roles` you can find will be able to manage the roles and permission.



Roles

Admin Permissions

✓ view_users	✓ add_users	✓ edit_users	✓ delete_users
✓ view_roles	✓ add_roles	✓ edit_roles	✓ delete_roles
✓ view_posts	✓ add_posts	✓ edit_posts	✓ delete_posts

User Permissions

✓ view_users	☐ add_users	☐ edit_users	☐ delete_users
✓ view_roles	☐ add_roles	☐ edit_roles	☐ delete_roles
✓ view_posts	☐ add_posts	☐ edit_posts	☐ delete_posts

Permissions Management

Permissions are not going to be changed very often in most cases, you can just add them directly into the database. I am leaving the implementation for this. It's simple CRUD, if you wanted you can implement it. One thing we can do to create a command which we can run to create permissions, something like `php artisan auth:permission tasks`, which will create `'view_tasks', 'add_tasks', 'edit_tasks', 'delete_tasks'` the permissions for tasks model, and if we pass `--remove` it will delete permissions on it.

Create our command by running `php artisan make:command AuthPermissionCommand`.

```
class AuthPermissionCommand extends Command
{
    protected $signature = 'auth:permission {name} {--R|remove}';
    ...

    public function handle()
    {
        $permissions = $this->generatePermissions();

        // check if its remove
        if( $is_remove = $this->option('remove') ) {
            // remove permission
            if( Permission::where('name', 'LIKE', '%'. $this->getNameArgument())->delete()
                $this->warn('Permissions ' . implode(', ', $permissions) . ' deleted.');
            }  else {
                $this->warn('No permissions for ' . $this->getNameArgument() .' found!');
            }

        } else {
            // create permissions
            foreach ($permissions as $permission) {
                Permission::firstOrCreate(['name' => $permission ]);
            }

            $this->info('Permissions ' . implode(', ', $permissions) . ' created.');
        }

        // sync role for admin
        if( $role = Role::where('name', 'Admin')->first() ) {
            $role->syncPermissions(Permission::all());
            $this->info('Admin permissions');
        }
    }

    private function generatePermissions()
    {
        $abilities = ['view', 'add', 'edit', 'delete'];
        $name = $this->getNameArgument();

        return array_map(function($val) use ($name) {
            return $val . '_'. $name;
        }, $abilities);
    }
```

```
        private function getNameArgument()
        {
            return strtolower(str_plural($this->argument('name')));
        }
    }
```

Our command is ready, next let's register it in Kernel, open the **app/Console/Kernel.php** and add.

```
     App\Console\Commands\AuthPermissionCommand;

    class Kernel extends ConsoleKernel
    {
        protected $commands = [
            AuthPermissionCommand::class
        ];
        ...
```

The `auth:permission` command is ready, now you can run it to add/remove permissions.

> If you added permissions manually in the db, don't forget to run `php artisan cache:forget spatie.permission.cache`, otherwise new permissions wont work.

Finally, we have a starter kit which you can use for any new project required roles and permissions. Check the source code & I hope you found it useful, let me know in the comments if you have any question.

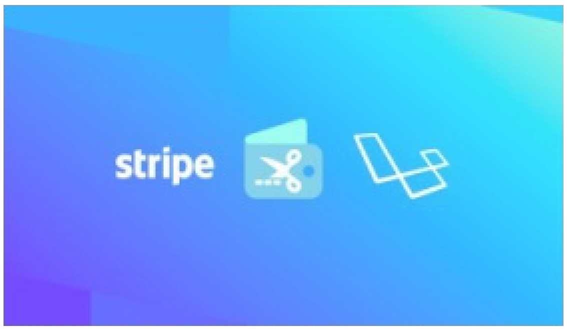
Source Code

Share this:

49 6

Related

Advance interactive database seeding in Laravel
December 20, 2016
In "Laravel"



Subscription with coupon using Laravel Cashier & Stripe
January 7, 2017
In "Laravel"

Use MySQL JSON field in Laravel
July 1, 2017
In "Laravel"

Tags: authorization, laravel, permissions, roles, user management

Build complete RSS feed reader app – Front-end with Vue.js, Vuex & Element UI
Convert Vue.js app to Native desktop app using Electron

Hi, Its me **Saqueib Ansari**, a web developer building cool things using PHP, Laravel, Angular, Vue and HTML CSS, This is my blog where I will be posting my thoughts and Tutorials.

Categories

- JavaScript
- Laravel
- PHP
- Quick Tips
- Trends
- Vue.js
- WordPress

Recent Posts

- Ready to use UUID in your next laravel app?
- VueJS pagination component to paginate anything in Laravel
- Use Laravel View Composer to share data in

partial views

- Reusable upload component in Laravel with Dropzone.js
- Building front-end for Twitter Like app on VueJS