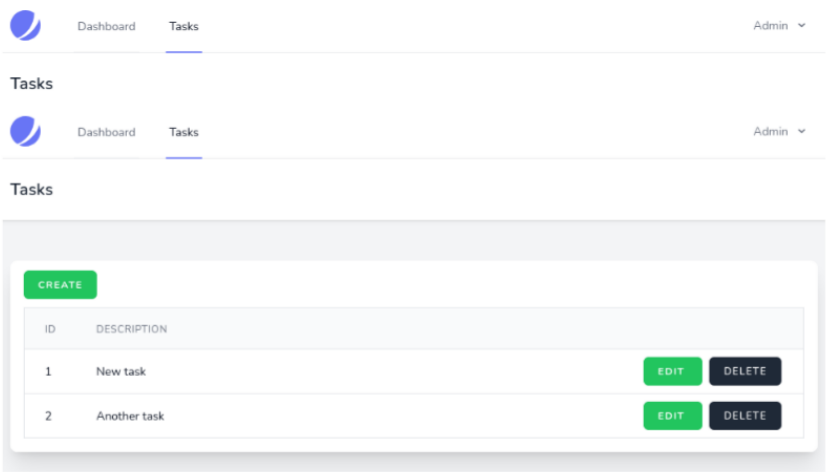
September 23, 2022 · 15 mins, 2823 words

Laravel Inertia Roles & Permissions: Breeze/Jetstream Examples

Laravel has a few starter kits like Breeze and Jetstream, but they don't have roles/permissions functionality. This time, let's talk specifically about Vue Inertia versions of those starter kits: how to add the roles and permissions there?



Before we get to the roles and permissions, we will prepare the actual project with the data structure and Inertia CRUD. So, let's go step-by-step.

Preparation Step 1: Install Breeze Inertia

First, we will take a look at Laravel Breeze implementation, and at the end of the article, we will touch on Jetstream, which will be really similar.

So, after creating a fresh Laravel project, we install Breeze:

```
composer require laravel/breeze --dev
```

Then we use Breeze scaffolding with Vue version specifically:

```
php artisan breeze:install vue
npm run dev
```

Preparation Step 2: Tasks CRUD

For using permissions, let's create a simple Tasks CRUD. The command below will create a Model, Migrations, and Controller:

```
php artisan make:model Task -mc
```

For migration, let's add only one field for the description

```
public function up()
{
    Schema::create('tasks', function (Blueprint $table) {
        $table->id();
        $table->text('description');
        $table->timestamps();
    });
}
```

Don't forget to add the `description` field to our Model's `$fillable`:

app/Models/Task.php:

```
protected $fillable = ['description'];
```

Our Controller for this example is a very typical CRUD Controller. The only difference from a typical Laravel application is that we render the Inertia page instead of Blade.

app/Http/Controllers/TasksController.php:

```
class TasksController extends Controller
{
    public function index()
    {
        $tasks = Task::all();
        return Inertia::render('Tasks/Index', compact('tasks'));
    }

    public function create()
    {
        return Inertia::render('Tasks/Create');
    }

    public function store(StoreTaskRequest $request)
    {
        Task::create($request->validated());
        return redirect()->route('tasks.index');
    }

    public function edit(Task $task)
    {
        return Inertia::render('Tasks/Edit', compact('task'));
    }

    public function update(UpdateTaskRequest $request, Task $task)
    {
        $task->update($request->validated());
        return redirect()->route('tasks.index');
    }

    public function destroy(Task $task)
    {
        $task->delete();
        return redirect()->route('tasks.index');
    }
}
```

For validation, we use Form Requests with only one rule:

```
public function rules(): array
{
    return [
        'description' => ['required', 'string'],
    ];
}
```

Now we need to create a new directory in `resources/js/Pages` named `Tasks`, and add three files in the new directory for showing CRUD in front-end:

- Index.vue
- Create.vue
- Edit.vue

Inside those Vue files, it's going to be very basic code using Breeze components.

resources/js/Pages/Tasks/Index.vue

```
<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import { Head, Link } from '@inertiajs/inertia-vue3';
import BreezeButton from '@/Components/PrimaryButton.vue';
import { Inertia } from '@inertiajs/inertia'

const props = defineProps({
    tasks: Object
})

function destroy(id) {
    Inertia.delete(route('tasks.destroy', id),{
      onBefore: () => confirm('Are you sure you want to delete?'),
    })
}
</script>

<template>
    <Head title="Tasks" />

    <AuthenticatedLayout>
        <template #header>
            <h2 class="font-semibold text-xl text-gray-800 leading-tight">
                Tasks List
            </h2>
        </template>

        <div class="py-12">
            <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
                <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
                    <div class="p-6 bg-white border-b border-gray-200">

                        <div class="mb-4" v-if="can.createTask">
                            <Link :href="route('tasks.create')" class="bg-green-500 hover:
                                Create
                            </Link>
                        </div>

                        <div class="min-w-full align-middle">
                            <table class="min-w-full divide-y divide-gray-200 border">
                                <thead>
                                    <tr>
                                        <th class="px-6 py-3 bg-gray-50 text-left w-10">
                                            <span class="text-xs leading-4 font-medium tex
                                        </th>
                                        <th class="px-6 py-3 bg-gray-50 text-left">
                                            <span class="text-xs leading-4 font-medium tex
                                        </th>
                                        <th class="px-6 py-3 bg-gray-50 text-left w-48">
```

```
                                    </th>
                                </tr>
                            </thead>

                            <tbody class="bg-white divide-y divide-gray-200 divide-sol
                                <tr class="bg-white" v-for="task in tasks" :key="task.
                                    <td class="px-6 py-4 whitespace-no-wrap text-sm le
                                        {{ task.id }}
                                    </td>
                                    <td class="px-6 py-4 whitespace-no-wrap text-sm le
                                        {{ task.description }}
                                    </td>
                                    <td class="space-x-2">
                                        <Link v-if="can.editTask" :href="route('tasks.
                                            Edit
                                        </Link>
                                        <BreezeButton v-if="can.destroyTask" @click="d
                                            Delete
                                        </BreezeButton>
                                    </td>
                                </tr>
                            </tbody>
                        </table>
                    </div>

                </div>
            </div>
        </div>
    </AuthenticatedLayout>
</template>
```

resources/js/Pages/Tasks/Create.vue:

```
<script setup>
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import TextInput from '@/Components/TextInput.vue';
import InputError from '@/Components/InputError.vue';
import InputLabel from '@/Components/InputLabel.vue';
import PrimaryButton from '@/Components/PrimaryButton.vue';
import { Head, useForm } from '@inertiajs/inertia-vue3';

const form = useForm({
    description: '',
});

const submit = () => {
    form.post(route('tasks.store'));
};
</script>

<template>
    <Head title="Create Task" />

    <AuthenticatedLayout>
        <template #header>
            <h2 class="font-semibold text-xl text-gray-800 leading-tight">
                Create Task
            </h2>
        </template>

        <div class="py-12">
            <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
                <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
                    <div class="p-6 bg-white border-b border-gray-200">

                        <form @submit.prevent="submit">
```

```
                            <div>
                                <InputLabel for="description" value="Description" />
                                <TextInput id="description" type="text" class="mt-1 block
                                <InputError class="mt-2" :message="form.errors.description
                            </div>

                            <div class="mt-4">
                                <PrimaryButton :class="{ 'opacity-25': form.processing }"
                                    Save
                                </PrimaryButton>
                            </div>
                        </form>

                    </div>
                </div>
            </div>
        </div>
    </AuthenticatedLayout>
</template>
```

resources/js/Pages/Tasks/Edit.vue:

```
<script setup>
import { defineProps } from "vue";
import AuthenticatedLayout from '@/Layouts/AuthenticatedLayout.vue';
import TextInput from '@/Components/TextInput.vue';
import InputError from '@/Components/InputError.vue';
import InputLabel from '@/Components/InputLabel.vue';
import PrimaryButton from '@/Components/PrimaryButton.vue';
import { Head, useForm } from '@inertiajs/inertia-vue3';

const props = defineProps({
    task: {
        type: Object,
    }
})

const form = useForm({
    id: props.task.id,
    description: props.task.description,
});

const submit = () => {
    form.patch(route('tasks.update', form.id));
};
</script>

<template>
    <Head title="Edit Task" />

    <AuthenticatedLayout>
        <template #header>
            <h2 class="font-semibold text-xl text-gray-800 leading-tight">
                Edit Task
            </h2>
        </template>

        <div class="py-12">
            <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
                <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
                    <div class="p-6 bg-white border-b border-gray-200">

                        <form @submit.prevent="submit">
                            <div>
                                <InputLabel for="description" value="Description" />
                                <TextInput id="description" type="text" class="mt-1 block
                                <InputError class="mt-2" :message="form.errors.description
                            </div>
```

```
                        <div class="mt-4">
                            <PrimaryButton :class="{ 'opacity-25': form.processing }"
                                Save
                            </PrimaryButton>
                        </div>
                    </form>

                </div>
            </div>
        </div>
    </div>
</AuthenticatedLayout>
</template>
```

Roles/Permissions Back-end: Models/Migrations

Now, as we have our CRUD, let's get to the roles/permissions: first, we need to add the logic on the back-end. In this case, we won't use any external packages like Spatie Permission, but just create some data structure to check the permissions ourselves.

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

Both models will only have one field `title`, so add that to your Model and Migration files. Model:

```
protected $fillable = ['title'];
```

Migration:

```
$table->string('title');
```

Roles can have many permissions, so let's add a relation to the Role model:

app/Models/Role.php

```
public function permissions(): BelongsToMany
{
    return $this->belongsToMany(Permission::class);
}
```

Also, we create a new table for Many to Many relation:

```
php artisan make:migration "create permission role table"
```

```
public function up()
{
    Schema::create('permission_role', function (Blueprint $table) {
        $table->foreignId('permission_id')->constrained();
        $table->foreignId('role_id')->constrained();
    });
}
```

Next, users can also have many roles. So, let's create a Many to Many relation.

```
php artisan make:migration "create role user table"
```

```
public function up()
```

```
{
    Schema::create('role_user', function (Blueprint $table) {
        $table->foreignId('role_id')->constrained();
        $table->foreignId('user_id')->constrained();
    });
}
```

Finally, add a relation to the User model.

app/Models/User.php:

```
public function roles(): BelongsToMany
{
    return $this->belongsToMany(Role::class);
}
```

Roles/Permissions Back-end: Seeders

Let's seed some data for testing users and roles. First, let's create two users, one with the `admin` role, and another with the `user` role.

database/seeders/UsersTableSeeder.php:

```
public function run()
{
    $users = [
        [
            'id' => 1,
            'name' => 'Admin',
            'email' => 'admin@admin.com',
            'password' => bcrypt('password'),
            'remember_token' => null,
        ],
        [
            'id' => 2,
            'name' => 'User',
            'email' => 'user@user.com',
            'password' => bcrypt('password'),
            'remember_token' => null,
        ],
    ];

    User::insert($users);
}
```

Next, we will create Roles and attach them to users.

database/seeders/RolesTableSeeder.php:

```
public function run()
{
    $roles = [
        [
            'id' => 1,
            'title' => 'Admin',
        ],
        [
            'id' => 2,
            'title' => 'User',
        ],
    ];

    Role::insert($roles);
}
```

database/seeders/RoleUserTableSeeder.php:

```
User::findOrFail(1)->roles()->sync(1);
User::findOrFail(2)->roles()->sync(2);
```

Now we have users with roles, we also need permissions for the roles.

database/seeders/PermissionsTableSeeder.php:

```
public function run()
{
    $permissions = [
        [
            'id' => 1,
            'title' => 'task_create',
        ],
        [
            'id' => 2,
            'title' => 'task_edit',
        ],
        [
            'id' => 3,
            'title' => 'task_destroy',
        ],
    ];

    Permission::insert($permissions);
}
```

Finally, we attach permissions to correct roles. The admin role will have all three permissions, and the user role will only have a `task_create` permission.

database/seeders/PermissionRoleTableSeeder.php:

```
Role::findOrFail(1)->permissions()->sync([1, 2, 3]);
Role::findOrFail(2)->permissions()->sync([1]);
```

Roles/Permissions Back-end: Defining Gates

Now, we need to register our permissions and define Laravel Gates from them. In `app/Providers/AuthServiceProvider.php` let's create new a method `registerUserAccessToGates()` and call it in the `boot()`. `registerUserAccessToGates` will look something like below:

```
protected function registerUserAccessToGates()
{
    try {
        foreach (Permission::pluck('title') as $permission) {
            Gate::define($permission, function ($user) use ($permission) {
                return $user->roles()->whereHas('permissions', function ($q) use ($permiss
                    $q->where('title', $permission);
                })->count() > 0;
            });
        }
    } catch (\Exception $e) {
        info('registerUserAccessToGates: Database not found or not yet migrated. Ignoring
    }
}
```

Basically, this code defines Gates for all permissions.

Notice: we're using a try-catch block here, because the first time, when running the project, AuthServiceProvider may be executed before the database table even exists.

Roles/Permissions Front-end: Breeze Inertia Implementation

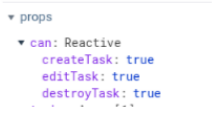
Before using our permissions in Vue we need to somehow pass the permissions data. For that, Inertia provides shared data which uses Laravel middleware called `HandleInertiaRequests`. Let's add a new key `can` to `auth` and pass all users permissions.

```
'auth' => [
    'user' => $request->user(),
    'can'  => $request->>user()?->loadMissing('roles.permissions')
        ->roles->flatMap(function ($role) {
            return $role->permissions;
        })->map(function ($permission) {
            return [$permission['title'] => auth()->user()->can($permission['title'])];
        })->collapse()->all(),
],
```

This will output json of user permissions, which can be accessed on every page using props like `$page.props.auth.can`

```
{"task_create":true,"task_edit":true,"task_destroy":true}
```

If you use Vue Devtools, you should see something like this:



Now, we can prevent the buttons from showing, if the user doesn't have a permission. Let's find a "Create Task" button and add `v-if="$page.props.auth.can.task_create"`.

resources/js/Pages/Tasks/Index.vue:

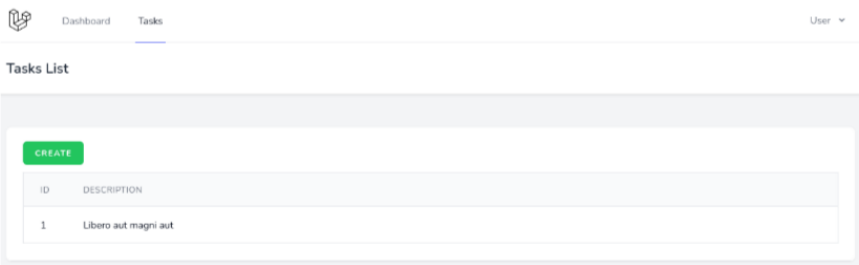
```
<div class="mb-4" v-if="$page.props.auth.can.task_create">
    <Link :href="route('tasks.create')" class="bg-green-500 hover:bg-green-700 text-white
        Create
    </Link>
</div>
```

Now, let's do the same for the edit and delete buttons.

resources/js/Pages/Tasks/Index.vue:

```
<Link v-if="$page.props.auth.can.task_edit" :href="route('tasks.edit', task)" class="bg-gr
    Edit
</Link>
<PrimaryButton v-if="$page.props.auth.can.task_destroy" @click="destroy(task.id)">
    Delete
</PrimaryButton>
```

After this, if you log in with the user and create a new task, you shouldn't see `Edit` and `Delete` buttons.



Roles/Permissions Back-end: Secure Controller

Now let's protect the backend part. In `app/Http/Controllers/TasksController.php`, in every method let's add a check for permissions using `$this->authorize()`. Now our controller will looks like this:

app/Http/Controllers/TasksController.php:

```
public function index()
{
    $tasks = Task::all();

    return Inertia::render('Tasks/Index', [
        'tasks' => $tasks,
        'can'   => [
            'createTask' => auth()->user()->can('task_create'),
            'editTask' => auth()->user()->can('task_edit'),
            'destroyTask' => auth()->user()->can('task_destroy'),
        ],
    ]);
}

public function create()
{
    $this->authorize('task_create');

    return Inertia::render('Tasks/Create');
}

public function store(StoreTaskRequest $request)
{
    $this->authorize('task_create');

    Task::create($request->validated());

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

public function edit(Task $task)
{
    $this->authorize('task_edit');

    return Inertia::render('Tasks/Edit', compact('task'));
}

public function update(UpdateTaskRequest $request, Task $task)
{
    $this->authorize('task_edit');

    $task->update($request->validated());

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

public function destroy(Task $task)
{
    $this->authorize('task_destroy');
```

```
    $task->delete();

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

Laravel Jetstream Roles and Permissions

Now, with Jetstream, most of the above functionality will work almost the same, but I will still briefly show you the steps.

Just like with Breeze, after creating a fresh Laravel project, we install Jetstream:

```
composer require laravel/jetstream
```

Then we use Jetstream scaffolding with Vue version specifically:

```
php artisan jetstream:install inertia
npm run dev
```

For Jetstream, all the back-end models, migrations, and controllers will be the same as we did with Breeze. The only difference is in JS part.

Also, in `app/Http/Middleware/HandleInertiaRequests.php` we don't see the `user` array, so just add `can` directly to `share()` method.

```
public function share(Request $request): array
{
    return array_merge(parent::share($request), [
        'can' => auth()->user()->loadMissing('roles.permissions')
            ->roles->flatMap(function ($role) {
                return $role->permissions;
            })->map(function ($permission) {
                return [$permission['title'] => auth()->user()->can($permission['title'])]
            })->collapse()->all(),
    ]);
}
```

And in JS, the only difference is that instead of `$page.props.auth.can` we use `$page.props.can`. The `auth` part is gone here.

So in Vue files, our check will look like this:

resources/js/Pages/Tasks/Index.vue:

```
<div class="mb-4" v-if="$page.props.can.task_create">
    <Link :href="route('tasks.create')" class="bg-green-500 hover:bg-green-700 text-white
        Create
    </Link>
</div>
```

Now, let's do the same for the edit and delete buttons.

resources/js/Pages/Tasks/Index.vue:

```
<Link v-if="$page.props.can.task_edit" :href="route('tasks.edit', task)" class="bg-green-5
    Edit
</Link>
<PrimaryButton v-if="$page.props.can.task_destroy" @click="destroy(task.id)">
```

```
    Delete
</PrimaryButton>
```

Get notified when participating

bokele wakiza franck 7 months ago

How to install **vue** with **Jetstream** without using **Inertia** in ?

Povilas Korop 7 months ago

I don't think it's possible, Jetstream has the Vue version specifically with Inertia. If you want pure Vue, you shouldn't/can't use Jetstream.

Leave a reply

Mahmudul Haque 4 months ago

is it possible to create a short video about **loadMissing()**? I do not understand, how does actually work.

thank you

1

Leave a reply

Leave a comment

You can use Markdown

Comment

Recent Premium Tutorials

October 20, 2022 · 25 mins, 4982 words · PREMIUM

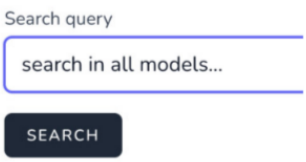
Laravel Breeze with User Areas: Student, Teacher, Admin

November 03, 2022 · 11 mins, 2046 words · PREMIUM

Laravel: Simple Two-Factor Auth OTP via Email and SMS

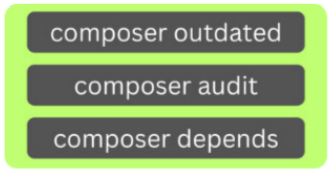
December 01, 2022 · 9 mins, 1768 words · PREMIUM

Value Objects and Data (Transfer) Objects in Laravel



December 15, 2022 · 9 mins, 1747 words · PREMIUM

January 05, 2023 · 11 mins, 2004 words · PREMIUM



April 27, 2023 · 7 mins, 1251 words · PREMIUM

Composer in Laravel: 9 Useful Features

Laravel Multiple Model Search: Queries, Scout, Packages

Laravel Testing: Mocking/Faking External 3rd Party APIs