

Authentication/Authorization and Inertia

First, make sure to have installed the Laravel CLI tool:

1. Run: `composer global require laravel/installer`
2. Check the directory where composer install binaries are by running:
`composer global config bin-dir --absolute`
3. Add the directory to the PATH by opening the file:
`nano ~/.bashrc`
4. Add this to the end of that file (unless the directory given in step 2 is different):
`export PATH="$PATH:$HOME/.config/composer/vendor/bin"`
5. Save the file and reload the service:
`source ~/.bashrc`
6. Try using the laravel command: `laravel --version`

*** For this to work, make sure you have npm (v10 or newer) and node (v20 or newer) installed in your machine**

Now, you can just create a new laravel project using the following:

```
laravel new my-app
```

You can follow the instructions to [customize your own project](#). In this example, we will use React as a frontend and laravel's built in authorisation/authentication.

After creating the project, you can serve it using the following:

```
cd my-app
npm install && npm run build
composer run dev
```

After that, you will already have the utilities for user registration, log in and log out. You can try them to check everything is working fine.

In your project's resources/js/ folder, you will see a variety of React components already created for you.

Implementing Authorization (Role-Based Access Control)

Step 1: Prepare the model

By default, Laravel does not include a full roles and permissions system, but we can implement basic authorization by adding a `role` field to the `users` table and then using Laravel's built-in `middleware` or Gates to restrict access based on user roles.

Add a Role to the User Model by making a migration

```
php artisan make:migration add_role_to_users_table --table=users
```

Editing the just created migration to have a `role` field:

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('role')->default('user'); // Default role is 'user'
    });
}
```

Running the migration:

```
php artisan migrate
```

And updating the User Model to include functions to identify the `role`.

```
class User extends Authenticatable
{
    protected $fillable = [
        'name', 'email', 'password', 'role',
    ];

    // Optionally, you can add a helper function to check for roles
    public function isAdmin()
    {
```

```

        return $this->role === 'admin';
    }

    public function isUser()
    {
        return $this->role === 'user';
    }
}

```

Step 2: Preparing the Gates

Instead of custom middleware, [Laravel Gates](#) provide a clean and reusable authorization mechanism.

Define the Gate

app/Providers/AuthServiceProvider.php

```

use Illuminate\Support\Facades\Gate;

public function boot(): void {
    Gate::define('admin', fn ($user) => $user->isAdmin());
}

```

Protect routes using the Gate

Use Laravel's built-in [can middleware](#) to restrict access.

routes/web.php

```

use App\Http\Controllers\AdminController;

Route::middleware(['auth', 'can:admin'])->group(function () {
    Route::get('/admin', [AdminController::class, 'index'])
        ->name('admin.dashboard');
});

```

If a non-admin user accesses this route, Laravel automatically returns **403 Forbidden**.

Step 3: The Admin Controller

Create the [controller](#):

```
php artisan make:controller AdminController
```

Controller implementation:

```
use Inertia\Inertia;

class AdminController extends Controller {
    public function index() {
        return Inertia::render('Admin/Dashboard');
    }
}
```

Step 4: Create the React Admin page

Create the page component:

```
resources/js/Pages/Admin/Dashboard.jsx

export default function Dashboard() {

    return ( <div> <h1 className="text-2xl font-bold">Admin
Dashboard</h1> <p>Only users with the admin role can access this
page.</p> </div> );

}
```

Step 5: Assign the admin role to a user

You can now assign roles when you create or update users. For example, you might want to make sure the first user you create is an admin, or you can manually update users via tinker or a database query.

```
php artisan tinker
$user = App\Models\User::find(1);
$user->role = 'admin';
$user->save();
```

Now You can test everything is working!