Guide du Projet Laravel : Gestion des Employés

Introduction

Ce document présente les solutions pour la mise en uvre d'un projet Laravel nommé "gestion_des_employes". Il couvre la création du projet, la configuration de la base de données, les migrations, les routes, les contrôleurs, les vues, les seeders et le middleware d'autorisation.

1 Création du Projet Laravel

Pour initialiser le projet Laravel, exécutez la commande suivante dans le terminal :

```
composer create-project laravel/laravel gestion_des_employes
```

2 Configuration de la Base de Données

Modifiez le fichier `.env` pour configurer la connexion à la base de données MySQL :

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

3 Commandes Artisan

Créez le modèle `Employee` avec sa migration, son contrôleur et sa factory en une seule commande :

```
php artisan make:model Employee -mcr
```

4 Migration pour la Table Employés

Définissez la structure de la table `employes` dans le fichier de migration :

```
public function up()
{
    Schema::create('employes', function (Blueprint $table) {
        $table->id();
        $table->string('NomEmployee');
        $table->string('PrenomEmployee');
        $table->string('Adresse');
        $table->string('Telephone');
        $table->foreignId('Departement_Id')->constrained('
            departements');
        $table->timestamps();
    });
}
```

5 Exécution des Migrations

Exécutez les migrations pour créer les tables dans la base de données :

```
php artisan migrate
```

6 Route de Ressources

Définissez une route de ressource pour gérer les opérations CRUD dans `routes/web.php` :

```
use App\Http\Controllers\EmployeeController;

Route::resource('employee', EmployeeController::class);
```

7 Fonction create() dans EmployeeController

La méthode `create` affiche le formulaire d'ajout d'un employé :

```
public function create()
{
    return view('AjouteEmployeeblade');
}
```

8 Fonction store() dans EmployeeController

La méthode `store` valide et enregistre un nouvel employé :

```
public function store(Request $request)
{
    $validated = $request->validate([
```

```
        'NomEmployee' => 'required',
        'PrenomEmployee' => 'required',
        'Adresse' => 'required',
        'Telephone' => 'required',
        'Departement_Id' => 'required|exists:departements,id'
    ]);

    Employee::create($validated);

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

9 Formulaire dans AjouteEmployeeblade.php

Créez un formulaire pour ajouter un employé dans `AjouteEmployeeblade.php` :

```
<form action="{{ route('employee.store') }}" method="POST">
    @csrf
    <!-- Champs du formulaire -->
</form>
```

10 Fonction index() dans EmployeeController

La méthode `index` affiche la liste paginée des employés :

```
public function index()
{
    $employes = Employee::paginate(10);
    return view('ListEmployes', compact('employes'));
}
```

11 Fonction edit() dans EmployeeController

La méthode `edit` affiche le formulaire de modification d'un employé :

```
public function edit(Employee $employee)
{
    return view('EditEmploye', compact('employee'));
}
```

12 Fonction update() dans EmployeeController

La méthode `update` valide et met à jour les informations d'un employé :

```
public function update(Request $request, Employee $employee)
{
    $validated = $request->validate([
```

```
        'NomEmployee' => 'required',
        'PrenomEmployee' => 'required',
        'Adresse' => 'required',
        'Telephone' => 'required',
        'Departement_Id' => 'required|exists:departements,id'
    ]);

    $employee->update($validated);

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

13 Fonction destroy() dans EmployeeController

La méthode `destroy` supprime un employé :

```
public function destroy(Employee $employee)
{
    $employee->delete();
    return redirect()->route('employee.index');
}
```

14 Vue de Confirmation de Suppression

Créez une vue pour confirmer la suppression d'un employé dans `DeleteConfirm.blade.php` :

```
<form action="{{ route('employee.destroy', $employee->id) }}"
   method="POST">
    @csrf
    @method('DELETE')
    <p>Êtes-vous sûr de vouloir supprimer cet employé?</p>
    <button type="submit">Confirmer</button>
</form>
```

15 Création d'un Seeder

Créez un seeder pour générer des données d'exemple :

```
php artisan make:seeder EmployeeSeeder
```

Dans `database/seeders/EmployeeSeeder.php` :

```
public function run()
{
    Employee::factory()->count(50)->create();
}
```

Exécutez le seeder :

```
php artisan db:seed --class=EmployeeSeeder
```

16 Middleware d'Autorisation

Créez un middleware pour restreindre l'accès aux modifications des employés :

```
php artisan make:middleware CheckEmployeeModification
```

Dans app/Http/Middleware/CheckEmployeeModification.php :

```
public function handle(Request $request, Closure $next)
{
    if (!auth()->user()->can_modify_employees) {
        abort(403, 'Accès non autorisé');
    }

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

Enregistrez le middleware dans app/Http/Kernel.php :

```
protected $routeMiddleware = [
    // ...
    'can.modify' => \App\Http\Middleware\
       CheckEmployeeModification::class,
];
```

Appliquez le middleware aux routes :

```
Route::resource('employee', EmployeeController::class)->
   middleware('can.modify');
```