Laravel

(suite)

Florian Rodriguez

Plan

Formulaire

Création d'un formulaire

On créer une nouvelle page pour créer un projet et une nouvelle route:

```
Route::get('/project', [ProjectController::class, 'index']);
Route::get('/project/create', [ProjectController::class, 'create']);
```

Copy

Dans `app/Http/Controller/ProjectController.php`:

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

Copy

Puis on créer `resources/views/project/create.blade.php` avec la même structure html

Optionnel: on peut ajouter un lien sous notre liste de projet

```
<a href="/project/create">Créer un nouveau projet</a>
```

Copy

resources/views/project/create.blade.php

Copy

```
@extends('layout')
@section('title','Project Create')

@section('content')
<h2>Créer un nouveau projet</h2>
<form>
    <div>
        <input type="text" name="title" placeholder="Titre du projet">
    </div>
    <div>
        <textarea name="description" placeholder="Description du projet"></textarea>
    </div>
    <div>
        <button type="submit">Créer le projet</button>
    </div>
</form>
@endsection
```

→ on check et on test avec un submit

→ remarquer les paramètres get: http://localhost:8000/projects/create?title=test+titre&description=test+description

Gestion des soumissions de formulaires

- On change le formulaire en post et on ajoute une action

```
<form method="POST" action="/project">
```

Copy

- On a déjà une route (`route/web.php`)

```
Route::get('/project', [ProjectController::class, 'index']);
```

Copy

→ Comment cela va donc marcher? On test -> HTTP exception

- C'est une route en GET
- On créer une route en POST:

```
Route::post('/project', [ProjectController::class, 'store']);
//(store est une convention qu'il est conseillé de suivre)
```

Copy

- Puis on va créer la méthode store dans le controller qui sera chargée de traiter le formulaire.

CSRF

- Dans le controller `ProjectController` on ajoute la méthode store

```php
public function store()
{
    return request()->all(); //pour récupérer les informations du formulaire
}
```

Copy

- On test → On obtient une erreur 419
- https://www.google.fr/search?q=laravel+419&oq=laravel+419
- Problème de token
- Ajout d’un helper spécial `@csrf` dans le `<form>`

```php
<form method="POST" action="/projects">
     @csrf
```

Copy

- On peut regarder la source du formulaire (champs hidden) et on check l’envoi du formulaire
- Pour info https://fr.wikipedia.org/wiki/Cross-site_request_forgery

Gestion soumission des formulaires et sauvegarde en BDD

- On modifie la méthode `store()` dans le controller `ProjectController` pour sauvegarder en BDD

```
public function store()
 {
     $project = new Project(); //on instancie un nouveau projet

     $project->title = request('title'); //on set le titre avec la donnée envoyée du formulaire
     $project->description = request('description');

     $project->save(); // on enregistre dans la base

     return redirect('/project'); // méthode pour rediriger vers une autre url (en get par défaut)
 }
```

Copy

- On test
- On peut aller voir dans la DB

Mass assignment

https://laravel.com/docs/master/eloquent#mass-assignment

- Plutôt que d’instancier un model, assigner ses attributs et sauvegarder:

```php
$project = new Project(); //on instancie un nouveau projet
$project->title = request('title'); //on set le titre avec la donnée envoyée du formulaire
$project->description = request('description');
$project->save(); // on enregistre dans la base
```

Copy

- On peut assigner directement

```php
Project::create(request(['title', 'description']));
```

Copy

- Il faut le spécifier au model

```php
protected $fillable = [
    'title','description'
];
//protected $guarded = [];
```

Copy

Recap

- On peut créer des formulaire HTML dans les vues, comme d'habitude.
- Les routes peuvent être déclarées en *get* ou en *post*.
- On peut donc utiliser la même route et appeler un méthode différente si la requête est en *post*
- On peut utiliser `request()->all()` ou `request('title')` pour récupérer les données envoyées en post, puis on utilise la méthode Eloquent `save()` pour sauvegarder dans la base de données.

RESTful

REST

REST (Representation State Transfert) est une façon de définir nos routes pour représenter nos fonctionnalités CRUD Dans un environnement REST, CRUD correspond souvent au méthodes http:

- POST
- GET
- PUT
- DELETE

Les 7 RESTful routes

Route / action	URL	Méthode HTTP	Description
Index	/projet	GET	Affiche la liste de tous les projets
Create	/projet/create	GET	Affiche un formulaire pour créer un nouveau projet
Store	/projet	POST	Ajoute un nouveau projet dans la base de donnée et redirige
Show	/projet/:id	GET	Affiche les information à propos d’un projet particulier
Edit	/projet/:id/edit	GET	Affiche un formulaire pour modifier un projet particulier
Update	/projet/:id	PUT	Met à jour un projet en particulier puis redirige
Destroy	/projet/:id	DELETE	Supprime un projet en particulier

Controllers resource

- Artisan crée la structure typique d’un CRUD:

```
php artisan help make:controller
php artisan make:controller -r ProjectController
```

Copy

- Dans la route:

```
Route::resource('/project', ProjectController::class);
```

Copy

- On peut enregistrer plusieurs contrôleurs “ressource” à la fois:

```
Route::resources([
    'photos' => ProjectController::class,
    'posts' => PostController::class
])
```

Copy

Méthodes spoofing

Les formulaires HTML ne peuvent pas réaliser des requêtes *PUT*, *PATCH* ou *DELETE* (les formulaires peuvent uniquement faire des requêtes *POST* ou *GET*), on doit ajouter un champ caché `@method`. ex: `@method('PUT')`

Routes ressources partielles

Il est possible de limiter les actions acceptées par une route ressource:

Copy

```
Route::resource('photos', PhotoController::class)->only([
    'index', 'show'
]);

Route::resource('photos', PhotoController::class)->except([
    'create', 'store', 'update', 'destroy'
]);
```

Route model binding

https://laravel.com/docs/master/routing#route-model-binding

- On peut spécifier le modèle lié à un contrôleur ressource:

```
php artisan make:controller -r ProjectController -m "Project"
```
Copy

```
public function show($id)
{
  $project = Project::find($id);
}
```
Copy

```
public function show(Project $project)
{
}
```
Copy

- Le paramètre de la route doit correspondre

```
Route::get('/project/{project}', [ProjectController::class, 'show']);
```
Copy

- On peut tout créer d'une seule commande:

```
php artisan make:model -a Project
```
Copy

Relationships

Eloquent Relationships

Copy

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Project extends Model
{
    // Get the user the project.
    public function user()
    {
        return $this->belongs(User::class);
    }
    // Get the tasks for the project.
    public function tasks()
    {
        return $this->hasMany(Task::class);
        // Select * from tasks where project_id = 1
    }
}
```

Copy

```php
<?php
$tasks = \App\Model\Project::find(1)->tasks;
foreach ($tasks as $task) {
    //
}
```

- hasOne
- hasMany
- belongsTo

Foreign keys

Copy

```
public function up()
{
    Schema::create('projects', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->unsignedBigInteger('user_id');
        $table->string('title');
        $table->text('description');
        $table->timestamps();

        $table->foreign('user_id')
            ->references('id')
            ->on('users')
            ->onDelete('cascade');
    });
}
```

Ou on peut simplifier avec

Copy

```php
public function up()
{
    Schema::create('projects', function (Blueprint $table) {
        $table->bigIncrements('id');
        $table->string('title');
        $table->text('description');
        $table->timestamps();

        $table->foreignId('user_id')
            ->constrained()
            ->onUpdate('cascade')
            ->onDelete('cascade');
    });
}
```

Factories & Seeder

Factories

```
php artisan make:factory ProjectFactory
```

Copy

```
 public function definition()
{
    return [
        'user_id' => \App\Models\User::factory(),
        'title' => $this->faker->sentence,
        'description' => $this->faker->paragraph
    ];
}
```

Copy

```
\App\Models\Project::factory(5)->create();
```

Copy

Seeders

- Dans `database/seeders/DatabaseSeeder.php` on peut appeler nos factories:

```php
public function run()
{
    \App\Models\User::factory(2)->create();
    \App\Models\Project::factory(10)->create();
}
```

Copy

- Puis

```
php artisan migrate:fresh --seed -v
```

Copy

- On peut lier nos factories et créer par exemple 5 projets par utilisateur:

```
public function run()
{
    $users = \App\Models\User::factory(2)->has(\App\Models\Project::factory()->count(5))->create();
}
```

Copy

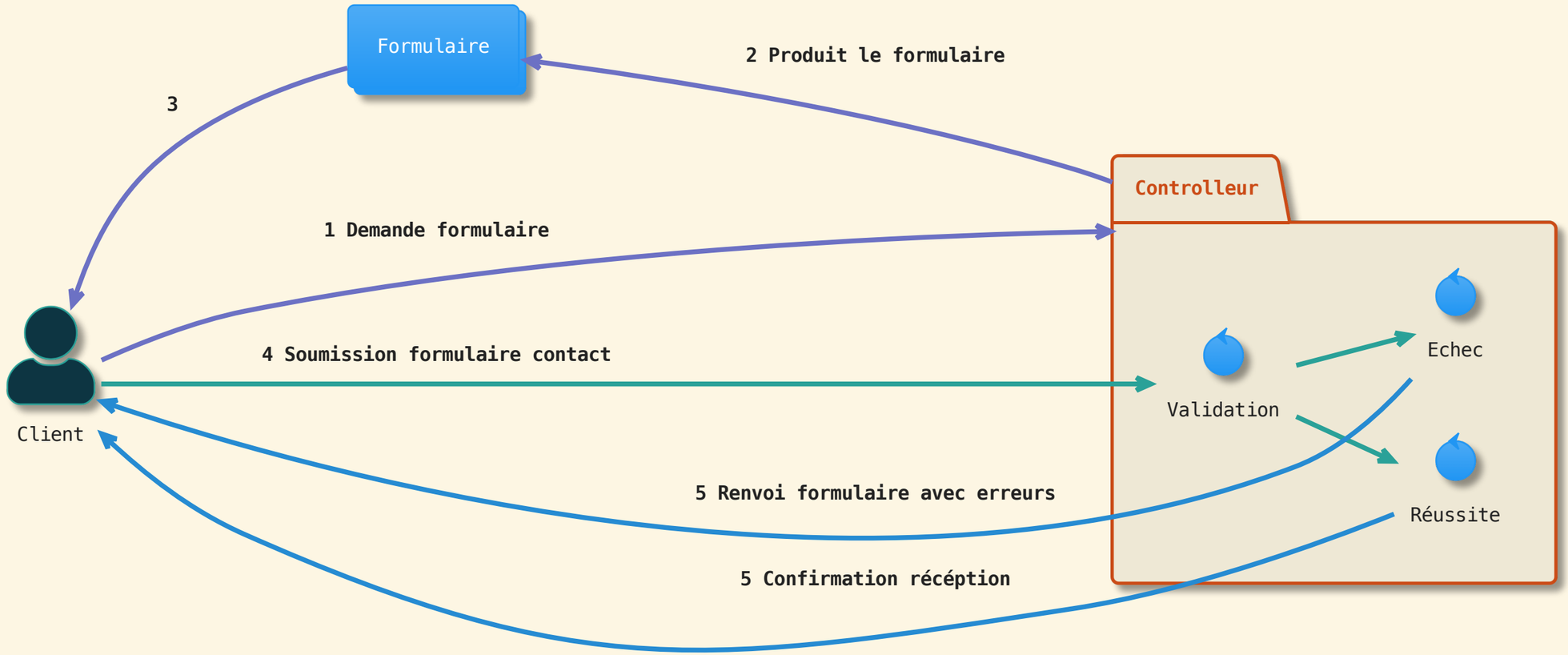
- Nous devons d’abords lier notre modèle User avec notre modèle Project, dans notre model user on ajoute la méthode:

```
public function project()
{
    return $this->hasMany(Project::class,'user_id');
}
```

Copy

Validation des formulaires

Scénario et routes



Validation des données

- La validation des données de formulaire est cruciale pour la sécurité et l'intégrité des données.
- Laravel offre une façon simple et efficace de *valider les données* soumises à l'aide de la méthode `validate()` disponible dans l'objet Request.

```php
$request->validate([
    'nom' => 'required|max:255',
    'email' => 'required|email|unique:users,email_address',
]);
```

Copy

- Ceci garantit que les données soumises respectent les règles spécifiées.
- Si les données ne passent pas la validation, Laravel renvoie automatiquement l'utilisateur au formulaire avec les messages d'erreur correspondants.
- Pour voir toutes les règles de validations possibles: https://laravel.com/docs/master/validation#available-validation-rules

Les vues (Formulaire et confirmation)

- En cas de réception du formulaire suite à des erreurs, on reçoit une variable `$errors` qui contient un tableau avec comme clés les noms des contrôles et comme valeurs les textes identifiant les erreurs.
- La variable `$errors` est générée systématiquement pour toutes les vues.

- On teste la présence d'une erreur pour chaque contrôle en ajustant le style et en affichant le texte de l'erreur si nécessaire avec la méthode first :

```
<p style="color:red">{{$errors->first('title')}}</p>
```

Copy

- S'il n'y a aucune erreur rien n'est renvoyé et donc rien n'est affiché, sinon on récupère la première (first) et on respecte le format imposé.

```
@if ($errors->has('title'))
<p style="color:red">{{$errors->first('title')}}</p>
@endif
```

Copy

Ou

```
@error('title')
<p style="color:red">{{$errors->first('title')}}</p>
@enderror
```

Copy

On peut aussi

```
<input @error('title') style="border-color:red" @enderror
```

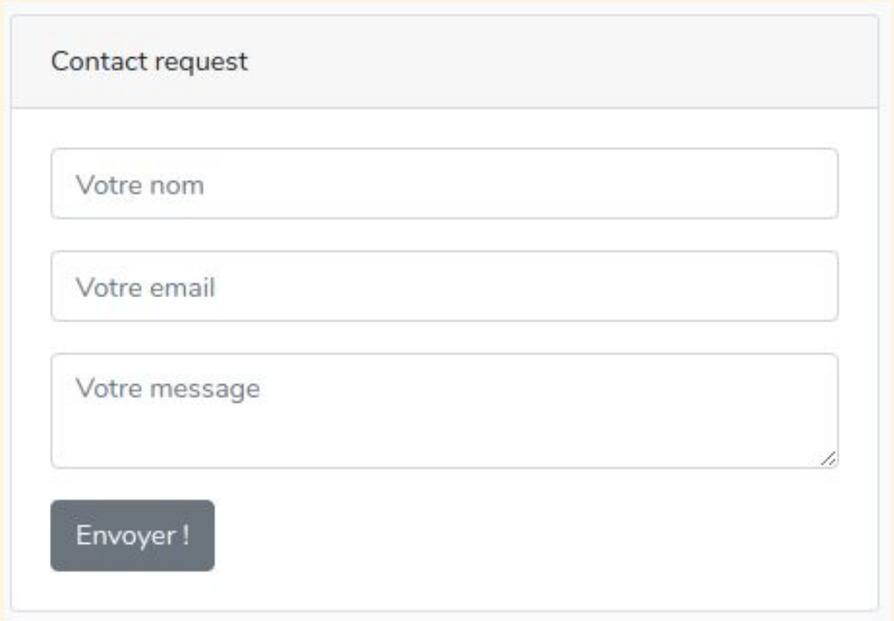
Copy

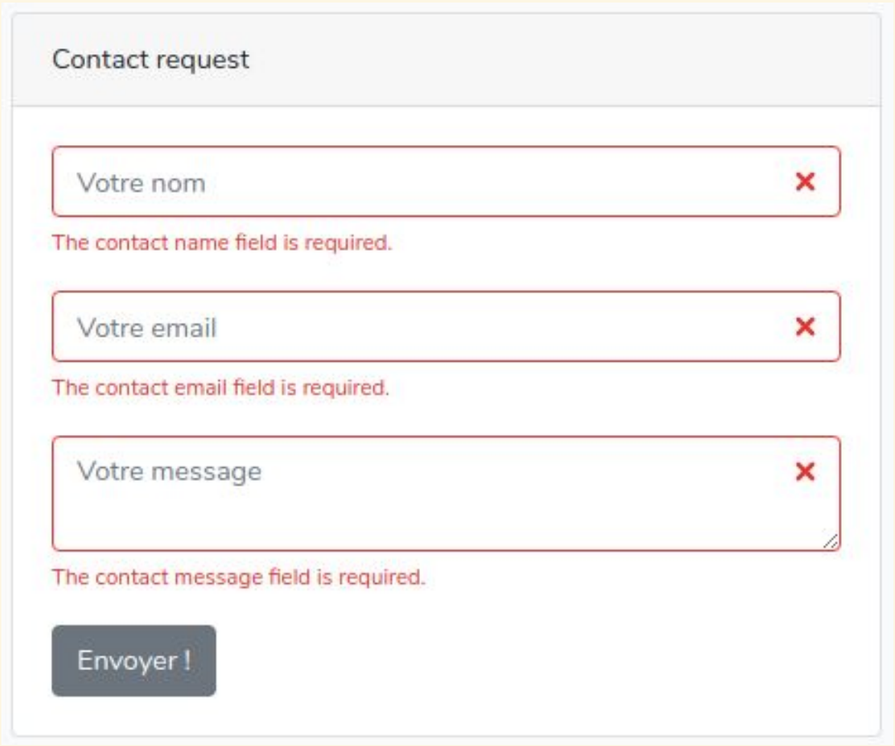
- Enfin en cas d’erreur de validation, les anciennes valeurs saisies sont retournées au formulaire et récupérées avec le helper old :

```
value="{{ old('nom') }}"
```

Copy

- Au départ le formulaire se présente ainsi :





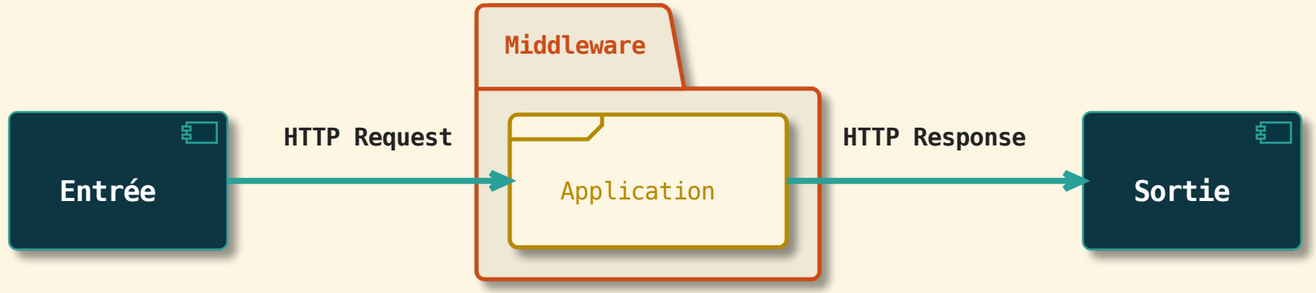
- Pour afficher toutes les erreurs…

Copy

```
@if ($errors->any())
<div class="alert alert-danger">
    <ul>
        @foreach ($errors->all() as $error)
        <li>{{ $error }}</li>
        @endforeach
    </ul>
</div>
@endif
```

Middleware

- C’est un mécanisme pour *filtrer les requêtes HTTP* qui entrent dans l’application
- Par exemple, vérifier si un utilisateur est authentifié ou non :
  - S’il n’est pas authentifié, on le redirige vers une page de login
  - S’il est authentifié, on laisse l’application s'exécuter normalement
- D’autres exemples: la génération de fichiers logs (fichiers qui listent les événements qui se sont produit dans l’application), vérifier si l’application est en mode maintenance ou non, les redirections, etc.
- Laravel inclut plusieurs middleware par défaut, pour l’authentification et la protection CRSF.
- Ils sont localisés dans le répertoire `app/Http/Middleware`
- Voir `app/Http/Kernel` pour le chargement.


Middleware
Entrée
HTTP Request
Application
HTTP Response
Sortie

Authentification

- https://laravel.com/docs/master/authentication
- https://laravel.com/docs/master/starter-kits

Installation

- Exemple avec laravel breeze:
  https://laravel.com/docs/master/starter-kits#laravel-breeze
  Attention cela va écraser certain fichier comme web.php ou des assets

```
composer require laravel/breeze --dev
php artisan breeze:install
npm install
npm run dev
php artisan migrate
```

Copy

- On peut voir `php artisan route:list`
  http://localhost:8000/login

Utilisation

- Pour «protéger» une route, on utilise le middleware `auth`

```php
Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');
```

Copy

- Ou un groupe de route

```php
Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});
```

Copy

- Autre example

```php
Route::get('/', [PagesController::class, 'home'])->middleware('auth');
```

Copy

→ On peut tester authentifié ou non

- Dans la vue:

```
@auth
Bonjour {{ Auth::user()->name }}!
@else
Bonjour vous n'êtes pas authentifié <a href="{{ route('login') }}">Log in</a>
@endauth
```

Copy

- ou l'inverse

```
@guest
@endguest
```

Copy

Assets bundling (Vite)

Rappel frontend / vocabulaire

- Ecosystème javascript
  - node
  - npm
  - package.json
- Scss

Vite

https://laravel.com/docs/master/vite

- install

```
npm install
```
Copy

- vite.config.js
- charger les styles et scripts

```
<! doctype html>
<head>
  {{-- ... --}}

  @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
```
Copy

Mail

- Notre application peut avoir besoin d'envoyer des mails. Ex: password reset
- On doit avoir un service / serveur de mail
- Dans un premier temps on va «envoyer» nos mails dans un fichier de log
- Configuration dans le `.env` → driver log `MAIL_MAILER=log`
  → fichier de log dans storage/logs/ (on peut faire un `tail -f`)

- Envoi d'un mail:

```
Mail::raw('Ça marche', function ($message) {
    $message->to('email@test.com')
        ->subject('Salut');
});
```

Copy

- Utilisation de mailtrap.io

Exemple création projet

Lié un projet avec l'utilisateur authentifié

- On modifie la méthode `store()` de notre `ProjectController`

```
Project::create(array_merge(request(['title', 'description']), ['user_id' => auth()->id()]));
```

Copy

→ Ne pas oublié d'ajouter `user_id` dans les fillables de project

- Ou mieux on surcharge la méthode save() du model `Project`

```
public function save(array $options = array())
{
    $this->user_id = auth()->id();
    parent::save($options);
}
```

Copy

On envoi un mail à la création du projet

- On modifie la méthode `store()` de notre `ProjectController`

```
Mail::raw('Projet '.request('title').': '.request('description'), function ($message) {
       $message->to('admin@monsite.com')
           ->subject('Projet créé: '. request('title') );
});
```

Copy

- Ajouter en haut du controller

```
use Illuminate\Support\Facades\Mail;
```

Copy

- C'est un example «en dur», laravel propose une gestion plus complète des mails. Voir la doc: https://laravel.com/docs/master/mail#generating-mailables

Ajout d'un message flash

- https://laravel.com/docs/master/responses#redirecting-with-flashed-session-data
- On peut ajouter un message en passant de la «donnée flash» à la session lors de la redirection

- Dans la méthode `store()` du controller Project

```
return redirect('/project')
        ->with('message', 'Projet créé!');
```

Copy

- Puis dans la vue

```
@if (session('message'))
<div>
    {{ session('message')}}
</div>
@endif
```

Copy

Évenements

Patern observeur / observable

shéma

Création Event / Listener

- Creation Event

```
php artisan event:list
php artisan make:event ProjectCreated
```

Copy

- On déclanche l'event dans le controller
  `app\Http\Controller\ProjectController.php`

```
ProjectCreated::dispatch($project);
```

Copy

  → « Un projet a été crée si quelqu'un à besoin de le savoir»

- Création Listener

```
php artisan make:listener SuiviManager -e ProjectCreated
```

Copy

- `App\Listeners\SuiviManager.php`

```
public function handle(ProjectCreated $event)
{
    //
    var_dump($event);
```

Copy

Lier Event / Listener

- Dans `app\Providers\EventServiceProvider.php`

```
use App\Events\ProjectCreated;
use App\Listeners\SuiviManager;
…
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    ProjectCreated::class => [
        SuiviManager::class
    ]
];
```

Copy

→ La liaison dans le provider c'est bien c'est explicite mais difficile à maintenir on peut supprimer

- Event discovery `app\Providers\EventServiceProvider.php`

```
public function shouldDiscoverEvents()
{
    return true;
}
```

Copy

-> va scanner le répértoire listener

Exemple

- On peut envoyer un mail dans l'event

```php
use Illuminate\Support\Facades\Mail;
…
public function handle(ProjectCreated $event): void
{
    //
    Mail::raw('Projet '.$event->project->title.': '.$event->project->description,
              function ($message) use ($event){
                  $message->to('admin@monsite.com')
                      ->subject('Projet créé: '. $event->project->title);
              });

    //die('Die SuiviManager listener');
}
```

Injection de dépendance, conteneur et façades

Tests et Déploiement

Bonnes pratiques pour tester et déployer des applications Laravel.

Conclusion