⬅Revenir à la théorie.

Les formulaires

1. Introduction

Les formulaires sont faciles à mettre en place. Nous allons voir comment faire un formulaire, appeler la route, valider les données reçues et la protection CSRF.

Sur la page de contacts de notre site Cupcake, il n'y a pas de formulaire de contacts. Nous y avons juste mis des icones vers réseaux sociaux.

Nous allons créer le formulaire de contact, les 2 routes (une pour l'affichage du formulaire, l'autre pour la confirmation), le contrôleur et la validation du formulaire.

2. Ajout d'un formulaire de contact

Soit la vue contacts.blade.php qui contient maintenant un formulaire:

```
<x-layout>
    <x-slot name="title">Contacts </x-slot>

    <div class="container bg-light mt-2 border rounded">
        <h1 class="text-center">Contacts</h1>
        <p>Nous sommes joignable via les plateformes suivantes:</p>
        <p><i class="bi bi-facebook" aria-label="facebook"></i> <i class="bi bi-twitter" a

        <p>N'hésitez pas à nous poser vos questions.</p>
        <form action="{{ url('contacts') }}" method="POST">
                @csrf
                <div class="form-group">
                    <input type="text" class="form-control  @error('nom') is-invalid @ende
                    @error('nom')
                        <div class="invalid-feedback">{{ $message }}</div>
                    @enderror
                </div>
                <div class="form-group">
                    <input type="email" class="form-control  @error('email') is-invalid @e
                    @error('email')
                        <div class="invalid-feedback">{{ $message }}</div>
                    @enderror
                </div>
                <div class="form-group">
                    <textarea class="form-control  @error('message') is-invalid @enderror"
                    @error('message')
                        <div class="invalid-feedback">{{ $message }}</div>
                    @enderror
                </div>
                <button type="submit" class="btn btn-secondary">Envoyer !</button>
            </form>
    </div>
</x-layout>
```

Nous discuterons plus tard de @error, @enderror et old('variable'). ☺

On voit que l'on va utiliser la méthode POST. De plus, dans l'attribut action on indique l'url ('contacts'). C'est à dire que l'on va utiliser ici la route 'contacts' avec la méthode POST.

3. Création du contrôleur

Nous allons créer un contrôleur nomm" ContactController avec artisan:

```
php artisan make:controller ContactController
```

Nous allons ajouter deux méthodes: create et store

```
class ContactController extends Controller
{
    public function create(){
        return view('contacts');
    }

    public function store(){
        return view('confirm');
    }
}
```

4. Les routes

Nous avons notre formulaire qui indique l'url et la méthode POST. Créons maintenant nos différentes routes 'contacts':

```
use App\Http\Controllers\ContactController;

Route::get('contacts', [ContactController::class, 'create']);
Route::post('contacts', [ContactController::class, 'store']);
```

Que constatez-vous ? Qu'est-ce qui est commun aux deux routes ?

5. Page de confirmation

On aura besoin d'une page de confirmation que nous avons bien reçu les informations de l'utilisation:

Page confirm.blade.php

```
<x-layout>
<x-slot name='title'>Confirmation formulaire de contacts</x-slot>
    <br>
    <div class="container">
        <p>Madame/Monsieur {{ request()->get('nom') }}, vous recevrez une réponse rapideme
        <p>Votre demande sera traîtée dans les plus brefs délais:</p>
        <p class='font-italic border p-2'>{{ request()->get('message')}}</p>
    </div>
</x-layout>
```

On constate que l'on peut réafficher des informations reçues du formulaire avec blade grâce à request()->('nomduchamp').

6. Validation des informations

Notre formulaire est toujours envoyé même si on n'a rien mis dans le formulaire. Ce qui n'est pas normal.

Pour valider ces champs, nous allons utiliser les requêtes de formulaires qui sont faciles à mettre en place.

Avec artisan, nous allons créer une request:

```
php artisan make:request ContactRequest
```

Un fichier a été créé dans Ap\Http\Requests nommé ContactRequest.php

Nous allons le modifier pour qu'il ressemble à ceci:

```php
namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class ContactRequest extends FormRequest
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'nom' => 'bail|required|between:5,20|alpha',
            'email' => 'bail|required|email',
            'message' => 'bail|required|max:250'
        ];
    }
}
```

- Dans la fonction authorize: Normalement on peut/doit y fait un contrôle de sécurité mais pour simplifier on va retourner toujours true indiquant que n'importe qui peut faire une requête.
- rules: c'est ici qu'on va faire la validation du formulaire. Notre return va renvoyer un tableau associatif avec le nom des champs. Comme valeur on va indiquer les règles de gestion:
    - bail : on arrête de vérifier dès qu'une règle n'est pas respectée.

- required : une valeur est requise, donc le champ ne doit pas être vide.
- between : nombre de caractères entre une valeur minimale et une valeur maximale.
- alpha : on n'accepte que les caractères alphabétiques.
- email : la valeur doit être une adresse email valide.

On peut mixer les valeurs avec un | entre chaque règle.
Il y en a d'autres que vous pouvez consulter sur la documentation officielle de Laravel.

Dans notre contrôleur ContactController, nous allons le modifier pour prendre en compte notre ContactRequest.

ContactController.php:

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Requests\ContactRequest;


class ContactController extends Controller
{
    public function create(){
        return view('contacts');
    }

    public function store(ContactRequest $request){
        return view('confirm');
    }
}
```

Avant notre classe, on a ajouté un use App\Http\Requests\ContactRequest; pour pouvoir utiliser ContactRequest dans notre contrôleur.

Dans la méthode store qui va appeler la vue confirm.blade.php, on indique qu'on aura un paramètre $request qui sera du type ContactRequest.

Et c'est là que la magie s'opère. Et que notre formulaire sera réaffiché si des champs ne respectent pas les règles définies dans ContactRequest.php

Mais il y a encore un problème, nos messages sont en anglais. Nous verrons comment les mettre en français plus tard.

7. @error('variable') et @enderror

Ces directives Laravel permettent de faire un traîtement en cas d'erreur sur une variable donnée. Grâce à bootstrap on pourra rendre plus sympathique la présentation de nos erreurs de formulaire grâce à la classe is-invalid.

```
<input type="text" class="form-control @error('nom') is-invalid @enderror" name="nom" id="
```

Cette classe sera ajoutée si le nom rencontre une erreur de validation.
On peut aussi ajouter du code html si une erreur est rencontrée:

```
    @error('nom')
        <div class="invalid-feedback">{{ $message }}</div>
    @enderror
```

8. old('variable')

Il est souvent bien pratique de réafficher un formulaire avec les valeurs précédentes. Sinon, notre utilisateur devra tout retaper, ce qui est assez pénible.

Il est possible de récupérer la précédente valeur grâce à old('variable').

Par exemple old('nom'), on récupère la valeur qui a été mise pour le champ nom.

Idem pour old('email') et old('message').

9. Attaque CSRF

Nativement Laravel vous protège des attaques du type CSRF.

9.1 Définition

> En sécurité des systèmes d'information, le cross-site request forgery, abrégé CSRF (parfois prononcé sea-surf en anglais) ou XSRF, est un type de vulnérabilité des services d'authentification web.
>
> L'objet de cette attaque est de transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits. L'utilisateur devient donc complice d'une attaque sans même s'en rendre compte. L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

Source: Wikipedia

9.2 Caractéristiques

> - Implique un site qui repose sur l'authentification globale d'un utilisateur ;
> - Exploite cette confiance dans l'authentification pour autoriser des actions implicitement ;
> - Envoie des requêtes HTTP à l'insu de l'utilisateur qui est dupé pour déclencher ces actions.
>
> Pour résumer, les sites sensibles au CSRF sont ceux qui acceptent les actions sur le simple fait de l'authentification à un instant donné de l'utilisateur et non sur une autorisation explicite de l'utilisateur pour une action donnée.
>
> Il est important de souligner que ces attaques peuvent aussi être menées sur des intranets pour permettre à un attaquant de récupérer des informations sur ledit intranet.

Source: Wikipedia

9.3 Prévention

> - Demander des confirmations à l'utilisateur pour les actions critiques, au risque d'alourdir l'enchaînement des formulaires.
> - Demander une confirmation de l'ancien mot de passe à l'utilisateur pour changer celui-ci ou changer l'adresse mail du compte.
> - Utiliser des jetons de validité (ou Token) dans les formulaires est basé sur la

> création du token via le chiffrement d'un identifiant utilisateur, un nonce et un horodatage. Le serveur doit vérifier la correspondance du jeton envoyé en recalculant cette valeur et en la comparant avec celle reçue1.
> - Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions : cette technique va naturellement éliminer des attaques simples basées sur les images, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.
> - Effectuer une vérification du référent dans les pages sensibles : connaître la provenance du client permet de sécuriser ce genre d'attaques. Ceci consiste à bloquer la requête du client si la valeur de son référent est différente de la page d'où il doit théoriquement provenir.

Source: Wikipedia

C'est pour ça aussi que parfois on vous demande de faire un capcha (images, calculs, etc.), de retaper votre ancien mot de passe avant de le changer, etc.

10. Exemple d'attaque CSRF

On va imaginer que votre courrier électronique accepte d'afficher les images automatiquement et qu'il affiche le code html (C'est à déconseiller !).

```
<img src="https://tutubanque.be/transfert?montant=10000&vers_compte=13579" height='1' wid
```

Si pendant la lecture de votre email, vous êtes connecté sur le site non protégé banque.be et que l'url est correcte. Alors une transaction de 10000€ vers le compte 13579 pourrait être effectuée... L'attaque est la même dans une page web sur un autre site ou via js...

11. Laravel et la directive @csrf

Laravel refusera de traiter toute demande d'un formulaire qui ne contiendra pas la directive @csrf. Cette directive sera ignorée si elle n'est pas lue depuis un fichier .blade.php.

Il en est de même pour le formulaire créé plus haut.

Le site retournera l'erreur: '419 PAGE EXPIRED'

Donc, si vous grand hacker vous mettez @csrf dans votre formulaire d'attaque, cela ne sert à rien. Car le serveur recevra via la méthode POST que les champs de notre formulaire. En effet, @csrf est interprêté côté serveur et n'est jamais affiché dans un formulaire côté client. En fait, Laravel quand il "voit" @csrf va générer une balise de formulaire ressemblant à <input type="hidden" name="_token" value="7GqWidw0PLOFbLjeC3nJmGswYWZGpWaCuSCKHbyJ">

On pourrait créer nous-même, comme Laravel, cette balise en se passant de @csrf de la manière suivante:

```
<input type="hidden" name="_token" value="<?php echo csrf_token(); ?>">
```

Côté serveur, Laravel a généré un token de sécurité avec une valeur. Ce token vient du serveur et si dans votre formulaire d'attaque vous avez aussi un token dont vous avez inventé une valeur, le serveur répondra '419 PAGE EXPIRED' car le token ne correspond pas à celui qui a été généré par Laravel. Merci Laravel ! 😀

⬅Revenir à la théorie.
