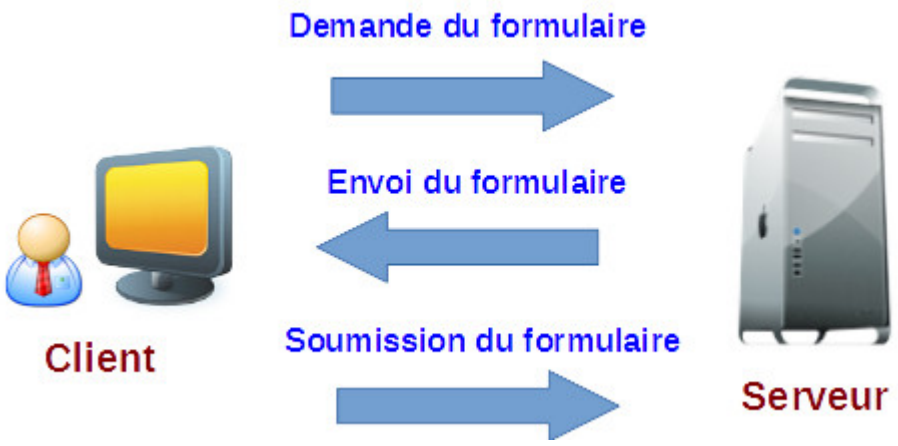
Cours Laravel 5.3 – les bases – formulaires et middlewares

Dans bien des circonstances, le client envoie des informations au serveur. La situation la plus générale est celle d'un formulaire. Nous allons voir dans ce chapitre comment créer facilement un formulaire avec Laravel, comment réceptionner les entrées et nous améliorerons notre compréhension du routage.

Nous verrons aussi l'importante notion de middleware.

Scénario et routes

Nous allons envisager un petit scénario avec une demande de formulaire de la part du client, sa soumission et son traitement :



On va donc avoir besoin de deux routes :

- une pour la demande du formulaire avec une méthode **get**,
- une pour la soumission du formulaire avec une méthode **post**.

On va donc créer ces deux routes dans le fichier routes/web.php :

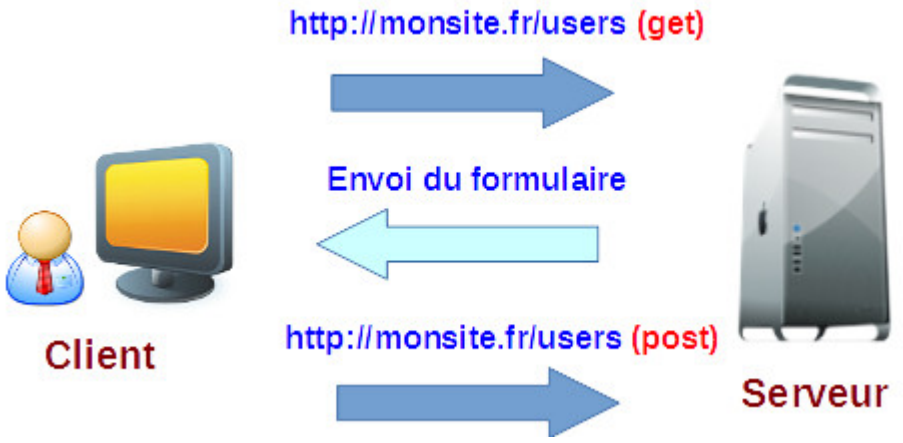
```
Route::get('users', 'UsersController@create');
Route::post('users', 'UsersController@store');
```

Jusque-là on avait vu seulement des routes avec le verbe **get**, on a maintenant aussi une route avec le verbe **post**.

Les urls correspondantes sont donc :

- http://monsite.fr/users avec la méthode **get**,
- http://monsite.fr/users avec la méthode **post**.

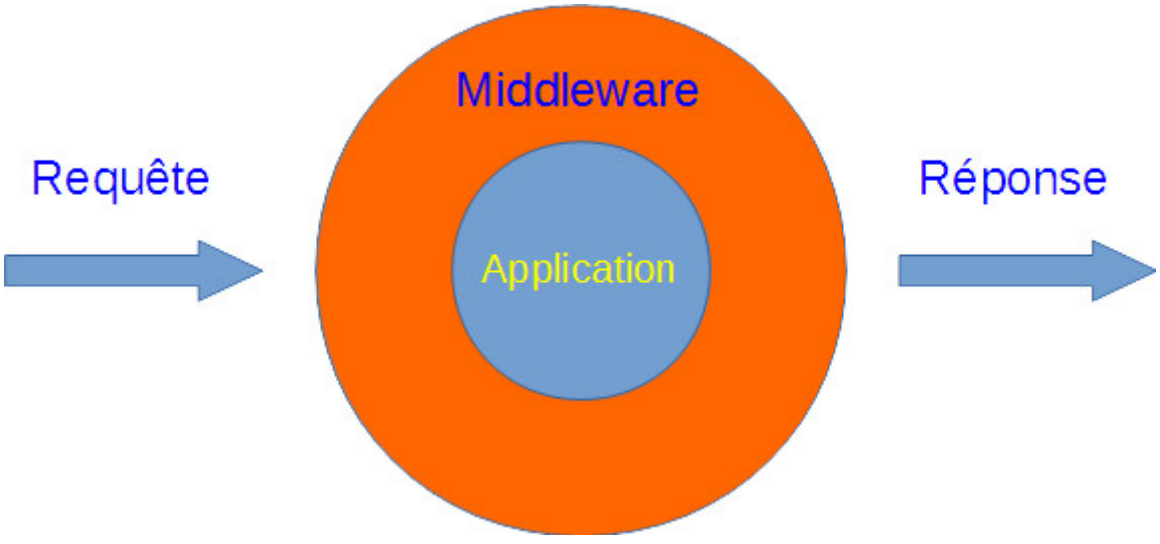
Donc on a la même url, seul le verbe diffère. Voici le scénario schématisé avec les urls :



Les middlewares

Les middlewares sont chargés de filtrer les requête HTTP qui arrivent dans l'application, ainsi que celles qui en partent (beaucoup moins utilisé). Le cas le plus classique est celui qui concerne la vérification de l'authentification d'un utilisateur pour qu'il puisse accéder à certaines ressources. On peut aussi utiliser un middleware par exemple pour démarrer la gestion des sessions.

Voici un schéma pour illustrer cela :



On peut avoir en fait plusieurs middlewares en pelures d'oignon, chacun effectue son traitement et transmet la requête ou la réponse au suivant.

Donc dès qu'il y a un traitement à faire à l'arrivée des requêtes (ou à leur départ) un middleware est tout indiqué.

Laravel peut servir comme application « web » ou comme « api ». Dans le premier cas on a besoin :

- de gérer les cookies,
- de gérer une session,
- de gérer la protection CSRF (dont je parle plus loin dans ce chapitre).

Si vous regardez dans le fichier app/Http/Kernel.php :

```
<?php
/**
 * The application's route middleware groups.
 *
 * @var array
 */
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
\Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],

    'api' => [
        'throttle:60,1',
        'bindings',
    ],
];
```

On trouve les deux middlewares de groupes (ils rassemblent plusieurs middlewares) « web » et « api ». On voit que dans le premier cas on active bien les cookies, les sessions et la vérification CSRF.

Par défaut toutes les routes que vous entrez dans le fichier routes/web.php sont incluses dans le groupe « web ». Si vous regardez dans le provider app/Providers/RouteServiceProvider.php vous trouvez cette inclusion :

```
<?php
/**
 * Define the "web" routes for the application.
 *
 * These routes all receive session state, CSRF protection, etc.
 *
 * @return void
 */
protected function mapWebRoutes()
{
    Route::group([
        'middleware' => 'web',
        'namespace' => $this->namespace,
    ], function ($router) {
        require base_path('routes/web.php');
    });
}
```

Le formulaire

Pour faire les choses correctement nous allons prévoir un template **resources/views/template.blade.php** :

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
</head>
<body>
        @yield('contenu')
</body>
</html>
```

Et une vue **resources/views/infos.blade.php** qui utilise ce template :

```
@extends('template')
```

```
@section('contenu')
    {!! Form::open(['url' => 'users']) !!}
        {!! Form::label('nom', 'Entrez votre nom : ') !!}
        {!! Form::text('nom') !!}
        {!! Form::submit('Envoyer !') !!}
    {!! Form::close() !!}
@endsection
```

Cette vue utilise le composant laravelcollective\html dont je vous ai déjà parlé dans le chapitre sur l'installation de Laravel et dont la documentation se trouve ici. Je pars donc du principe que vous l'avez installé. Si ce n'est pas le cas reportez vous à ce chapitre pour le faire sinon ce code ne fonctionnera pas.

Nous avons déjà vu comment s'organise une vue avec un template, par contre la création du formulaire mérite quelques commentaires. Pour créer un formulaire avec le composant **laravelcollective/html** il faut commencer par l'ouvrir :

```
{!! Form::open(['url' => 'users']) !!}
```

La sémantique est simple : on veut pour un formulaire (**Form**), ouvrir (**open**) celui-ci, et qu'il pointe vers l'url « users ».

Ensuite on veut une étiquette (**label**) :

```
{!! Form::label('nom', 'Entrez votre nom : ') !!}
```

On veut un contrôle de type « text » qui se nomme « nom » :

```
{!! Form::text('nom') !!}
```

On veut enfin un bouton de soumission (**submit**) avec le texte « Envoyer ! » :

```
{!! Form::submit('Envoyer !') !!}
```

Et finalement on veut clore (**close**) le formulaire :

```
{!! Form::close() !!}
```

Le code généré pour le formulaire sera alors le suivant :

```
<form method="POST" action="<a href="<a href="http://monsite.fr/users">http://monsite.fr/users</a>"><a
```

```
href="http://monsite.fr/users">http://monsite.fr/users</a></a>"
accept-charset="UTF-8">
                <input    name="_token"    type="hidden"
value="pV1vWWdUqFDfYsBjKag43C3NvzbIC0lHtMnv9BpI">
        <label for="nom">Entrez votre nom : </label>
        <input name="nom" type="text" id="nom">
        <input type="submit" value="Envoyer !">
</form>
```

Quelques remarques sur cette génération :

- la méthode par défaut est **post**, on n'a pas eu besoin de le préciser,
- l'action est bien générée,
- il y a un contrôle caché () destiné à la protection **CSRF** dont je parlerai plus loin,
- l'étiquette est bien créée avec son attribut for,
- le contrôle de texte est du bon type avec le nom correct, il est en plus généré un id pour qu'il fonctionne avec son étiquette,
- le bouton de soumission a été généré avec son texte.

Le résultat sera un formulaire sans fioriture :

Vous n'êtes pas obligé d'utiliser le composant **laravelcollective/html** pour créer des formulaires mais je vous y encourage parce qu'il simplifie le codage et je l'utilise dans de ce cours. Par exemple pour créer le formulaire sans l'utiliser il nous faudrait écrire ceci :

```
@extends('template')

@section('contenu')
        <form method="POST" action="{!! url('users') !!}" accept-
charset="UTF-8">
                {!! csrf_field() !!}
                <label for="nom">Entrez votre nom : </label>
                <input name="nom" type="text" id="nom">
                <input type="submit" value="Envoyer !">
        </form>
```

```
@endsection
```

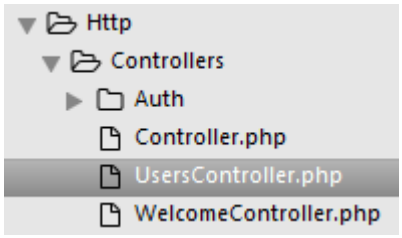
C’est quand même plus simple et lisible avec le composant !

Le contrôleur

Il ne nous manque plus que le contrôleur pour faire fonctionner tout ça. Utilisez Artisan pour générer un contrôleur :

```
php artisan make:controller UsersController
```

Vous devez le retrouver ici :



Modifiez ensuite son code ainsi :

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UsersController extends Controller
{
    public function create()
    {
        return view('infos');
    }

    public function store(Request $request)
    {
        return 'Le nom est ' . $request->input('nom');
    }
}
```

Le contrôleur possède deux méthodes :

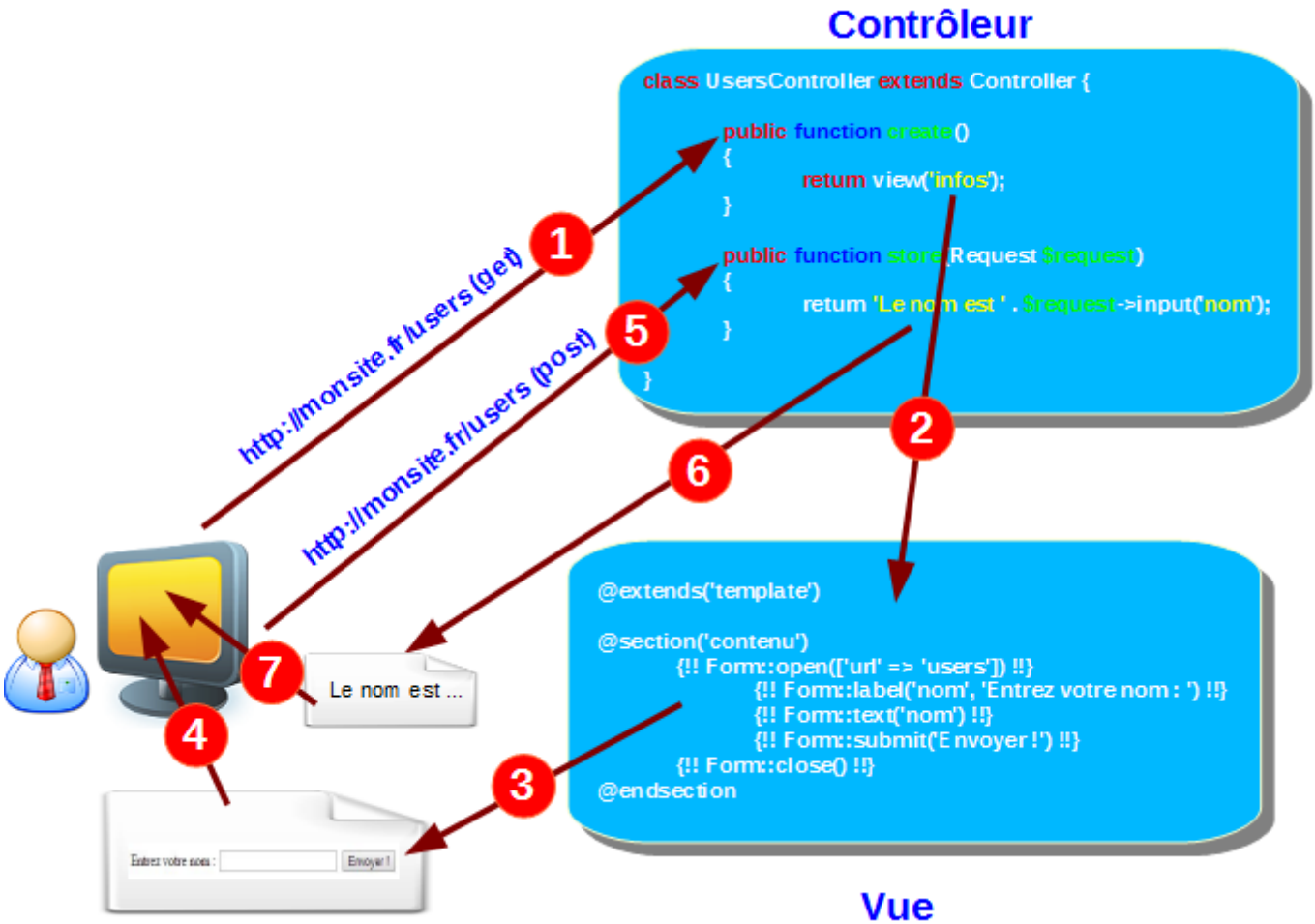
- la méthode create qui reçoit l’url **http://monsite.fr/users**

avec le verbe get et qui retourne le formulaire,
- la méthode store qui reçoit l'url **http://monsite.fr/users** avec le verbe post et qui traite les entrées.

Pour la première méthode il n'y a rien de nouveau et je vous renvoie aux chapitres précédents si quelque chose ne vous paraît pas clair. Par contre nous allons nous intéresser à la seconde méthode.

Dans cette seconde méthode on veut récupérer l'entrée du client. Encore une fois la syntaxe est limpide : on veut dans la requête (**request**) les entrées (**input**) récupérer celle qui s'appelle **nom**.

Si vous faites fonctionner tout ça vous devez au final obtenir l'affichage du nom saisi. Voici une schématisation du fonctionnement qui exclue les routes pour simplifier :



(1) le client envoie la requête de demande du formulaire qui est transmise au contrôleur par la route (non représentée sur le schéma),

(2) le contrôleur crée la vue « infos »,
(3) la vue « infos » crée le formulaire,
(4) le formulaire est envoyé au client,
(5) le client soumet le formulaire, le contrôleur reçoit la requête de soumission par l'intermédiaire de la route (non représentée sur le schéma),
(6) le contrôleur génère la réponse,
(7) la réponse est envoyée au client.

La protection CSRF

On a vu que le formulaire généré par Laravel comporte un contrôle caché avec une valeur particulière :

```
<input name="_token" type="hidden" value="pV1vWWdUqFDfYsBjKag43C3NvzbIC0lHtMnv9BpI">
```

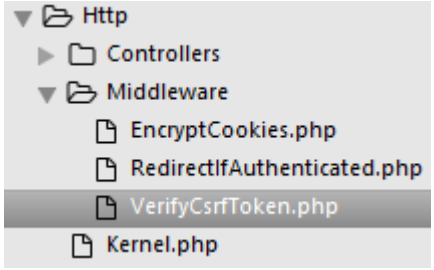
A quoi cela sert-il ?

Tout d'abord **CSRF** signifie **Cross-Site Request Forgery**. C'est une attaque qui consiste à faire envoyer par un client une requête à son insu. Cette attaque est relativement simple à mettre en place et consiste à envoyer à un client authentifié sur un site un script dissimulé (dans une page web ou un email) pour lui faire accomplir une action à son insu.

Pour se prémunir contre ce genre d'attaque Laravel génère une valeur aléatoire (**token**) associée au formulaire de telle sorte qu'à la soumission cette valeur est vérifiée pour être sûr de l'origine.

Vous vous demandez peut-être où se trouve ce middleware CSRF ?

Il est bien rangé dans le dossier **app/Http/Middleware** :

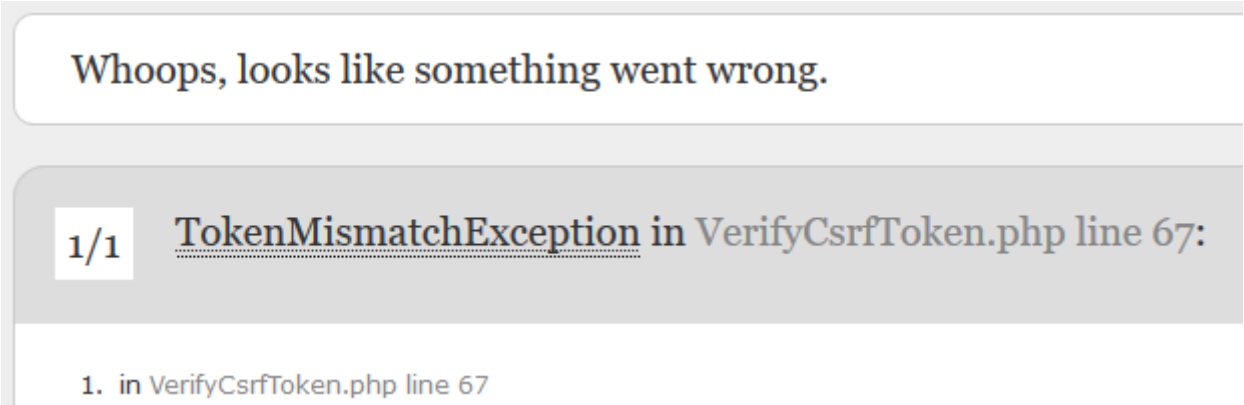


Pour tester l'efficacité de cette vérification essayez un envoi de formulaire sans le token en modifiant ainsi la vue (adaptez la valeur de l'action selon votre contexte) :

```
@extends('template')

@section('contenu')
        <form method="POST" action="{!! url('users') !!}" accept-charset="UTF-8">
                <label for="nom">Entrez votre nom : </label>
                <input name="nom" type="text" id="nom">
                <input type="submit" value="Envoyer !">
        </form>
@endsection
```

Vous tomberez sur cette erreur (avec un code 500) à la soumission :



C'est bien pratique que le composant **laravelcollective/html** le place automatiquement dans les formulaires, du coup on n'a même pas besoin d'y penser. Ça ne sera évidemment plus du tout le cas en cas de soumission avec Ajax par exemple, nous verrons cela plus tard…

En résumé

- Laravel permet de créer des routes avec différents verbes : get, post…
- Un middleware permet de filtrer les requêtes.
- Un formulaire peut facilement être créé avec le composant laravelcollective/html.

- Les entrées du client sont récupérées dans la requête.
- On peut se prémunir contre les attaques CSRF, cette défense est mise en place automatiquement par Laravel.