Laravel 4 : chapitre 12 : Les formulaires

Ouvrir et fermer un formulaire

Commençons par quelque chose de simple : ouvrir et fermer un formulaire. Entrez ce code dans le fichier app/routes.php :

```
Route::get('/', function() {
    echo Form::open(array('url' => 'test'));
    echo Form::close();
});
```

Je pars toujours du principe que j'ai un serveur en localhost dans un dossier laravel. Si j'utilise l'URL http://localhost/laravel/public je vais avoir en retour une page vide mais avec ce code HTML :

```
<form method="POST" action="http://localhost/laravel/public/test" accept-charset="UTF-8"></form>
```

On constate que l'on a par défaut la méthode POST. On voit aussi que l'URL de retour se construit tout seul. On a aussi par défaut le charset en UTF-8.

Si on désire une autre méthode que POST il faut le préciser, par exemple pour GET :

```
Route::get('/', function() {
    echo Form::open(array('url' => 'test', 'method' => 'GET'));
    echo Form::close();
});
```

Avec comme résultat :

```
<form method="GET" action="http://localhost/laravel/public/test" accept-charset="UTF-8"></form>
```

On peut aussi préciser une méthode moins courante comme PUT ou DELETE :

```
Route::get('/', function() {
          echo Form::open(array('url' => 'test', 'method' =>
'PUT'));
        echo Form::close();
});
```

Comme les navigateurs ignorent ces méthodes Laravel génère un champ caché pour signaler la méthode :

```
<input name="_method" type="hidden" value="PUT">
```

Protection CSRF

Pour tout formulaire qui n'utilise pas GET il est ajouté automatiquement un champ caché pour la sécurité CSRF :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::close();
});
```

Dans le code généré apparaît un champ caché pour mémoriser la clé de sécurité :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
                <input    name="_token"    type="hidden"
value="i0SN6gBv5W9UVYENJZGi6A9A7BB3MySAWyM4SZFt">
</form>
```

Au retour il suffit de tester cette clé en créant un filtre :

```
Route::any('test', array('before' => 'csrf', function() {

}));
```

Les champs texte

Voyons maintenant comment créer des champs de type « texte ». Voici un formulaire avec les 5 types disponibles :

```
Route::get('/', function() {
    echo Form::open(array('url' => 'test'));
    echo Form::text('text', 'zone de texte');
```

```
    echo Form::password('password');
    echo Form::email('email');
    echo Form::hidden('hidden');
    echo Form::textarea('textarea');
    echo Form::close();
});
```

Résultat :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <input name="text" type="text" value="zone de texte">
    <input name="password" type="password" value="">
    <input name="email" type="email">
    <input name="hidden" type="hidden">
    <textarea name="textarea" cols="50" rows="10"></textarea>
</form>
```

On peut prévoir une valeur par défaut comme je l'ai fait pour le champ text.

Les nouveau types du HTML 5

Le HTML 5 a introduit de nouveau types : color, date, month… Pour les obtenir il faut utiliser la méthode input. Voici un exemple avec quelques types :

```
Route::get('/', function() {
    echo Form::open(array('url' => 'test'));
    echo Form::input('color', 'color');
    echo Form::input('date', 'date');
    echo Form::input('month', 'month');
    echo Form::input('number', 'number');
    echo Form::close();
});
```

Résultat :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
        <input name="color" type="color">
        <input name="date" type="date">
        <input name="month" type="month">
        <input name="number" type="number">
```

```
</form>
```

Les légendes

Voici comment générer une légende :

```
Route::get('/', function() {
    echo Form::open(array('url' => 'test'));
    echo Form::label('nom', 'Nom : ');
    echo Form::text('nom');
    echo Form::close();
});
```

Code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <label for="nom">Nom : </label>
    <input name="nom" type="text" id="nom">
</form>
```

Vous remarquez que la liaison entre le champ et la légende se fait automatiquement.

Les boutons radio

Créer des boutons radio est aussi très facile :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::label('reponse', 'oui');
        echo Form::radio('reponse', 'oui', true);
        echo Form::label('reponse', 'non');
        echo Form::radio('reponse', 'non');
        echo Form::close();
});
```

Avec ce code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <label for="reponse">oui</label>
      <input checked="checked" name="reponse" type="radio"
value="oui" id="reponse">
```

```
    <label for="reponse">non</label>
    <input name="reponse" type="radio" value="non" id="reponse">
</form>
```

Mais on constate un défaut, on se retrouve avec deux id identiques et les légendes non différenciées. Cela est dû au mode de génération. Lorsqu'un champ suit un label et s'il a le même nom que ce label, alors il se retrouve avec un id de ce même nom. C'est parfait pour un champ isolé de type texte mais ça ne convient plus pour des champs groupés comme les boutons radio. Alors il faut un peu ruser avec le générateur :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::label('oui', 'oui');
         echo Form::radio('reponse', 'oui', true, array('id' =>
'oui'));
        echo Form::label('non', 'non');
         echo Form::radio('reponse', 'non', false, array('id' =>
'non'));
        echo Form::close();
});
```

On se retrouve alors avec un code correct :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <label for="oui">oui</label>
    <input id="oui" checked="checked" name="reponse" type="radio"
value="oui">
    <label for="non">non</label>
    <input id="non" name="reponse" type="radio" value="non">
</form>
```

Les cases à cocher

Les cases à cocher fonctionnent sur le même principe que les boutons radio, alors voici l'exemple précédent mais avec cette fois des cases à cocher :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::label('oui', 'oui');
```

```
        echo Form::checkbox('reponse', 'oui', true, array('id' =>
'oui'));
        echo Form::label('non', 'non');
        echo Form::checkbox('reponse', 'non', false, array('id' =>
'non'));
        echo Form::close();
});
```

Avec ce code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <label for="oui">oui</label>
       <input  id="oui"  checked="checked"  name="reponse"
type="checkbox"  value="oui">
    <label for="non">non</label>
    <input id="non" name="reponse" type="checkbox" value="non">
</form>
```

Les listes de choix

Vous pouvez aussi générer une liste de choix :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::select('Animaux',
                array(
                        'C' => 'Chimpanzé',
                        'E' => 'Eléphant',
                        'G' => 'Girafe',
                        'S' => 'Serpent',
                        'Z' => 'Zèbre')
                );
        echo Form::close();
});
```

Code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <select name="Animaux">
        <option value="C">Chimpanzé</option>
        <option value="E">Eléphant</option>
        <option value="G">Girafe</option>
```

```
        <option value="S">Serpent</option>
        <option value="Z">Zèbre</option>
    </select>
</form>
```

Par défaut on a une liste déroulante avec aucune option sélectionnée. Il est possible de rendre la liste non déroulante en précisant son attribut size, la sélection multiple avec l'attribut multiple. On peut aussi préciser une option sélectionnée :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::select('Animaux',
                array(
                        'C' => 'Chimpanzé',
                        'E' => 'Eléphant',
                        'G' => 'Girafe',
                        'S' => 'Serpent',
                        'Z' => 'Zèbre'),
                        'E',
                           array('size' => 4, 'multiple' => 
'multiple')
                );
        echo Form::close();
});
```

Résultat :

```
<form method="POST" action="http://localhost/laravel/public/test" 
accept-charset="UTF-8">
    <select size="4" multiple="multiple" name="Animaux">
        <option value="C">Chimpanzé</option>
        <option value="E" selected="selected">Eléphant</option>
        <option value="G">Girafe</option>
        <option value="S">Serpent</option>
        <option value="Z">Zèbre</option>
</select>
</form>
```

Les boutons

Créer un bouton est simple :

```
Route::get('/', function() {
```

```
        echo Form::open(array('url' => 'test'));
        echo Form::button('Bouton');
        echo Form::close();
});
```

Code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <button type="button">Bouton</button>
</form>
```

Pour un bouton de soumission de formulaire c'est cette syntaxe :

```
Route::get('/', function() {
        echo Form::open(array('url' => 'test'));
        echo Form::submit('Envoyer !');
        echo Form::close();
});
```

Code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
<input type="submit" value="Envoyer !">
</form>
```

Pointer sur une route nommée

Voici comment faire pointer un formulaire sur une route nommée :

```
Route::get('/', function() {
        echo Form::open(array('route' => 'test'));
        echo Form::submit('Envoyer !');
        echo Form::close();
});
Route::any('test', array('as' => 'test', function() {
        return 'Route "test"';
}));
```

Créer des macros

Si les possibilités de la classe Form ne vous suffisent pas vous pouvez créer vos propres macros. Par exemple si vous utilisez le

framework CSS Bootstrap vous pouvez trouver pratique ce genre de macro :

```
Form::macro('group', function($name, $title, $type) {
    return '<div class="control-group">'
            .Form::label($name, $title, array('class' => 'control-
label'))
            .'<div class="controls">'
                .(($type == 'text')? Form::text($name):
Form::password($name))
            .'</div></div>';
});

Route::get('/', function() {
    echo Form::open(array('route' => 'test'));
    echo Form::group('name', 'Nom :', 'text');
    echo Form::group('password', 'Mot de passe :', 'password');
    echo Form::close();
});
```

Ça a le mérite de clarifier le code pour construire le formulaire.

Code généré :

```
<form method="POST" action="http://localhost/laravel/public/test"
accept-charset="UTF-8">
    <div class="control-group">
        <label for="name" class="control-label">Nom :</label>
        <div class="controls">
            <input name="name" type="text" id="name">
        </div>
    </div>
    <div class="control-group">
         <label for="password" class="control-label">Mot de passe
:</label>
        <div class="controls">
              <input name="password" type="password" value=""
id="password">
        </div>
    </div>
</form>
```

Il y a un package intéressant pour Bootstrap.

Renseigner les champs d'un formulaire

Il arrive souvent d'avoir une erreur de validation et de représenter à l'utilisateur un formulaire et il est alors judicieux de renseigner certains champs pour éviter une nouvelle saisie. On peut le faire simplement avec une session flash :

```
Route::get('/', array('as' => 'log', function() {
        echo Form::open(array('route' => 'test'));
        echo Form::label('email', 'Email : ');
        echo Form::email('email');
        echo Form::submit('Envoyer');
        echo Form::close();
}));
Route::any('test', array('as' => 'test', function() {
        return Redirect::route('log')->withInput();
}));
```

La redirection s'effectue avec la valeur saisie et le champ est à nouveau rempli.

Supposons maintenant que vous voulez remplir automatiquement les champs d'un formulaires à partir d'un objet. Regardez cet exemple :

```
Route::get('/', function() {
        $user = new User();
        $user->email = 'moi@chezmoi.fr';
        echo Form::model($user, array('route' => 'test'));
        echo Form::label('email', 'Email : ');
        echo Form::email('email');
        echo Form::submit('Envoyer');
        echo Form::close();
});
```

Le champ email est automatiquement rempli à partir de la propriété de l'objet $user !