MISSION LARAVEL en PHP : TP ToDo D'après le travail de Valentin Brousseau

Savoirs et savoir-faire développés :
Utiliser le framework « Laravel », Utiliser l'outil d'ORM "eloqent" en PHP

A4.1.2	Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.2	Conception ou adaptation de l'interface utilisateur d'une solution applicative
A4.1.3	Conception ou adaptation d'une base de données
A4.1.4	Définition des caractéristiques d'une solution applicative
A4.1.5	Prototypage de composants logiciels
A4.1.6	Gestion d'environnements de développement et de test
A4.1.7	Développement, utilisation ou adaptation de composants logiciels

Pré-requis : Une installation Laravel Propre !

Gestion de Projet :

Pour ce cours, nous utiliserons un outil de suivi collaboratif SLACK.
Vous pouvez rejoindre le panneau de ce cours : https://btssaintsauveur.slack.com/ en suivant le fil #tptodo

Application Todo Liste

Action d'ajout

Maintenant que nous avons implémenté la liste, nous allons pouvoir faire le code pour la partie « ajout d'une tâche ». La méthodologie sera la même que pour la liste à savoir :

- Ajout du code dans le contrôleur.
- Ajout de la route.
- Modification du template pour implémenter la fonctionnalité.

Le contrôleur

Nous allons faire un mapping automatique entre la requête HTTP et le modèle `Todos`

```
public function saveTodo(Request $request){
    Todos::create($request->all());
    return redirect()->action('TodosController@liste');
}
```

Que va t'il se passer lors de l'appel ? L'objet `$request` contient tous les paramètres de l'appel HTTP, la méthode `all()` permet de les récupérer. L'objet `Todos` possède une méthode permettant de créer un nouvel enregistrement en base de données. Les valeurs passées en paramètre de `create()` permette de renseigner automatiquement les champs en base de données.

✋ Tester l'ajout. Normalement ça ne doit pas fonctionner… Pourquoi ? Tout simplement car nous ne spécifions pas l'ensemble des champs nécessaire à la création de notre objet.

- Quel est l'autre solution possible ?

La première approche est la plus rapide mais elle sous entend que tous les paramètres soient bien initialisées dans « l'input » HTTP. Dans cette version la méthode est plus complète et gère la création de l'objet Todos manuellement, en récupérant les différents éléments dans la requette HTTP.

```
public function saveTodo(Request $request){
    $texte = $request->input('texte');

    if($texte){
      $todo = new Todos();
      $todo->texte = $texte;
      $todo->termine = 0;
      $todo->save();
    }

    return redirect()->action('TodosController@liste');
}
```

Et c'est tout …

La Route

Pour la route modifier le fichier `routes/web.php` :

```
Route::post('/action/add', "TodosController@saveTodo");
```

😳 Questions

- À quoi correspond le mot clef « post » ?
- Que se passe-t-il si on fait un appel de type GET (ou PUT, …) ?

Tester

Maintenant que nous avons notre action d'ajout, nous allons pouvoir tester notre Todo List réellement.

- Démarrer votre serveur de test (`php artisan serve`).
- Ajouter une nouvelle todo dans l'interface.
- Vous devez avoir une erreur 419. À quoi correspond-t-elle ?

Correction de l'erreur 419

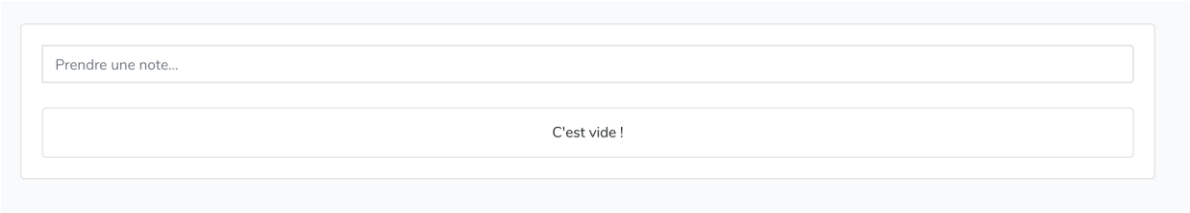
L'erreur 419 indique que votre Token `CSRF` (anti-rejeu) est expiré, ou plutôt dans notre cas que vous ne l'avez pas fourni. C'est une sécurité intégrée à Laravel pour l'ensemble des requêtes POST qui arrive dans votre code.

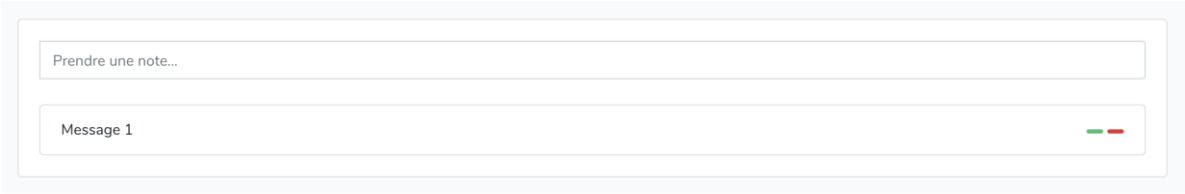
Nous devons donc ajouter un input « caché » dans notre formulaire pour envoyer en plus du texte une valeur dite de sécurité.

Modifier le template `home.blade.php` pour ajouter le code suivant dans la partie `<form>` :

```
<form […]>
@csrf <!-- << L'annotation ici ! -->
[…]
</form>
```

➔ Re-tester





Action : marquer comme terminé

Pour l'action terminé nous allons devoir updater un enregistrement en base de données, pour ça nous allons le récupérer puis mettre le `boolean termine` à 1.

```
public function markAsDone($id){
    $todo  = Todos::find($id);
    if($todo){
        $todo->termine = 1;
        $todo->save();
    }
    return redirect()->action('TodosController@liste');
}
```

Route

Ajouter une route de type `get` avec un paramètre nommé `{id}` dans le fichier `routes/web.php`.

Le lien doit-être : `/action/done/{id}` et la méthode dans le contrôleur `TodosController@markAsDone` en vous inspirant des exemples précédent ajouter la bonne instruction dans le fichier `routes/web.php`.

➔ Vérifier votre travail !
Comment avez-vous fait ?

Action de suppression

Pour la partie suppression, nous allons devoir dans un premier temps récupérer la todo par son ID.

```
public function deleteTodo($id){
        $todo  = Todos::find($id);
        if($todo){
            $todo->delete();
        }

        return redirect()->action('TodosController@liste');
}
```

Route

Ajouter une route de type `get` avec un paramètre nommé `{id}` dans le fichier `routes/web.php`.

Le lien doit-être : `/action/delete/{id}` et la méthode dans le contrôleur `TodosController@deleteTodo` en vous inspirant des exemples précédent ajouter la bonne instruction dans le fichier `routes/web.php`.

Questions

- Un `delete` de type `get` est-ce normal ?

- Quelle est l'autre solution ?

Ajouter les actions dans le template

Maintenant que nous avons déclaré nos « 3 actions » dans notre contrôleur (et dans le fichier de route) nous allons les utiliser dans notre template « home » voici les étapes :

- Éditer le fichier `resources/views/home.blade.php`.
- Ajouter sur chaque ligne de la boucle « foreach » deux liens qui vont « suprimer » et « terminer ».

Aide :
Pour accéder à une variable exemple `id` : `$todo->id`.
Exemple : `<a href="lien{{ $todo->id }}">Terminer</a>`

Ajout d'une 2nd page

Ajouter une nouvelle page dans votre site web cette page sera la page « À propos »,
Aucune aide autre que :

- Route.
- Méthode dans le contrôleur.
- Template qui « @extends » du gabarit / template de base.
- Ajouter un lien pour accèder à cette page dans le header du site.

Évolution souhaitée : Nommer les routes

Comme vous l'avez constaté, nous utilisons les routes comme une simple URL. Avec Laravel il est possible de faire mieux que ça ! Il est possible de nommer les routes (exemple `todo.save`) pour les utiliser directement dans notre template. Ça va permettre d'améliorer votre code (changement de contrôleur plus simple par exemple) et surtout de le rendre plus lisible.

Modifier le fichier route

Éditer le fichier `routes/web.php` pour remplacer le contenu par :

```
<?php

Route::get('/', "TodosController@liste")->name("todo.list");
Route::post('/action/add', "TodosController@saveTodo")->name('todo.save');
Route::get('/action/done/{id}', "TodosController@markAsDone")->name('todo.done');
Route::get('/action/delete/{id}', "TodosController@deleteTodo")->name('todo.delete');
```

- Avez-vous vu la différence ? Des `->name("…")` sont en plus, vos routes sont maintenant nommées

Éditer votre template « home »

Maintenant que nous avons édité nos routes, il faut les utiliser dans le template pour ça modifier les différents liens (dans le form et dans les `<a>` d'action) :

```
[…]
    <form action="{{ route('todo.save') }}" method="post" class="add">
[…]

<a href="{{ route('todo.done', ['id' => $todo->id]) }}">[…]</a>
<a href="{{ route('todo.delete', ['id' => $todo->id]) }}">[…]</a>
```

- Avez vous vu la différence ?

Il est également possible de les utiliser dans le contrôleur via la directive :

```
return redirect()->route('todo.list');
```

Modifier votre code pour l'utiliser.

Question

- Quel est l'avantage d'utiliser les routes nommées ?

Évolution souhaitée : Ajout de contrôle

Seul les `Todos` marqués comme terminée peuvent être supprimées, il faudra donc contrôler l'état avant de faire le `delete()` en base de données.

- Modifier la méthode `deleteTodo` contrôleur pour ajouter la règle de gestion (Indice `$todo->termine`)
- Ajouter la directive `@if` dans le template afficher uniquement les bonnes actions en fonction de l'état de la todo. Voir la documentation du if avec blade
- Pour les boutons d'actions, utiliser des icônes :
  - Exemple fontawesome
  - Choisir une icône
  - Utilisation : `<i class="fas fa-stroopwafel"></i>`
- « Encapsuler » votre icône autour d'un `a` exemple

```
<a href="/actions/done/{{$todo->id}}" class="btn btn-success">
<i class="fas fa-check"></i></a>
```

Évolution souhaitée : Message en cas d'erreur

Avertir l'utilisateur en cas d'erreur est important! Comme vous l'avez constaté aucune gestion de message d'erreur n'est présente dans le contrôleur. Implémenter des messages (basiques) grâce à l'aide suivante :

Ajouter un message à afficher

```
[…]
Session::flash('message', "Message d'erreur de traitement à définir.");
[…]
```

Afficher le message dans votre template

```
@if(Session::has('message'))
    <p class="alert alert-danger">{{ Session::get('message') }}</p>
@endif
```

Explication sur la méthode Flash