

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 "eloquent" 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

Introduction

Dans ce TP nous allons réaliser une application / site web, le but de cette application/site web est de faire de la prise de note de « TODO » ou aussi appelé liste de tâches.

Une TODO List est un procédé qui se veut simple et efficace pour gérer les tâches d'un projet. Ces tâches peuvent être indépendantes ou devoir, au contraire, être accomplies dans un certain ordre.

Voilà la liste des fonctionnalités de l’application que l'on va créer :

- Lister les tâches.
- Ajouter une tâche.
- Marquer comme terminé une tâche.
- Suppression d'une tâche.

Technologie

Dans ce TP nous allons utiliser les technologies suivantes :

- Laravel (Framework PHP)
- ORM Eloquent
- Bootstrap 4 (Framework HTML / CSS / JS)
- NodeJS (Utilisation de Webpack)

Initialiser le projet

La première étape lors d'un projet Laravel est la création de la structure avec une simple ligne de commande, dans notre cas :

👉 Attention, le projet va être créé dans le dossier `laravel-todo` dans le dossier courant.

```
> laravel new laravel-todo
Crafting application...
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
[...]
```

La création prend environ 1 minute, un certains nombres de librairies sont téléchargées. Une fois terminé, prenez quelques instants pour regarder les fichiers qui ont été créés.

Questions

- Où sont les librairies ?
- Quel est l'intérêt à votre avis d'utiliser un outil pour initialiser un projet ?

Configurer le projet, le `.env`

Le projet qui vient d'être créé est « générique » c'est à dire qu'il ne possède aucune personnalisation et peut donc servir de base quelques soit votre développement.

La première étape est donc d'éditer le fichier `.env` pour configurer les options de base de votre projet tel que :

- Le nom : `ToDoList`
- Le type de base de données
- ...

Le reste du fichier `.env`

Nous n'allons pas toucher aux autres paramètres mais certains sont tout de même intéressants. Je vous laisse donc consulter la documentation en ligne de laravel.

Conception de la base de données

Voilà à quoi va ressembler notre base de données :

TODO (id, texte, termine, timestamp)

todos	
PK	<u>id</u>
	texte
	termine
	timestamp

Création de la « migration »

Laravel utilise un ORM pour manipuler la base de données. Pour rappel un ORM est

Un mapping objet-relationnel (en anglais object-relational mapping ou ORM) est une technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances entre cette base de données et les objets du langage utilisé.

L'ORM utilisé par Laravel est Eloquent, il est à la fois puissant et relativement simple. Autrement dit avec un ORM dans la plus part des cas vous n'écrivez plus de requête SQL mais vous manipulez des objets représentant la base de données.

Comme pour la création du projet, la création de la définition de la table « todos » va se faire via une requête dans votre terminal en utilisant Artisan.

```
migrate
migrate:fresh      Drop all tables and re-run all migrations
migrate:install    Create the migration repository
migrate:refresh    Reset and re-run all migrations
migrate:reset      Rollback all database migrations
migrate:rollback   Rollback the last database migration
migrate:status     Show the status of each migration
```

```
php artisan make:migration create_todo_table --create=todo
ou
php artisan migration :install
```

L'option `--create=todo` dans la première méthode permet d'indiquer le nom du modèle de la table à créer.

👉 Le contenu du fichier créé est fictif, il est là pour illustrer comment travailler. Nous allons le modifier pour mettre les informations relatives à notre table.

La commande vient de créer un nouveau fichier dans le dossier `database/migrations`. Dans mon cas le fichier se nomme :

```
/laravel-todo/database/migrations/2018_11_12_102501_create_todo_table.php
```

Questions

- Il y a deux autres fichiers dans votre répertoire de migration. A quoi servent ils ?
- Quels sont les champs par défaut dans la création d'une table simple ?
- Pourquoi ?

Nous allons définir notre schéma (à savoir la définition de la table) pour y ajouter les 2 colonnes qui nous seront utiles `texte` et `termine`. L'ORM étant une librairie objet, la définition de nouveaux champs se fait via une méthode, dans notre cas :

```
$table->string('texte');
```

```
$table->boolean('termine');
```

Ajouter les deux champs dans la méthode `up` du fichier.

Questions

- À quoi correspond la méthode `up` et `down` ?
- L'ordre des champs est-il important ?

Création en base

Maintenant que le script est terminé, nous allons indiquer à Laravel d'effectuer « la migration » c'est-à-dire de transformer votre définition PHP en instruction SQL pour créer réellement la base de données. Retour dans la ligne de commande :

```
> php artisan migrate:install
[...]  
Migrating: 2018_09_09_150442_create_todos_table  
Migrated: 2018_09_09_150442_create_todos_table
```

Vérifier votre console et votre travail !

ERREUR :

Remplacer dans `config/database.php`

```
'charset' => 'utf8mb4',  
'collation' => 'utf8mb4_unicode_ci',  
  
Par  
  
'charset' => 'utf8',  
'collation' => 'utf8_unicode_ci',
```

Votre base de données est maintenant prête à être utilisée. Vous pouvez aller voir le contenu grâce à phpMyAdmin.

COURS :

L'ORM eloquent permet de créer une correspondance entre les classes PHP et la base de données. On peut créer une classe `maTable` associée à une table `maTable`. On pourra accéder aux colonnes de la table au travers des propriétés de la classe et bénéficier de méthodes pour charger un élément ou une liste d'éléments, enregistrer, mettre à jour ...

Exemple :

```
$user = User::find(5) ;  
Echo $user->name ;  
$user->age=32 ;  
$user->save() ;
```

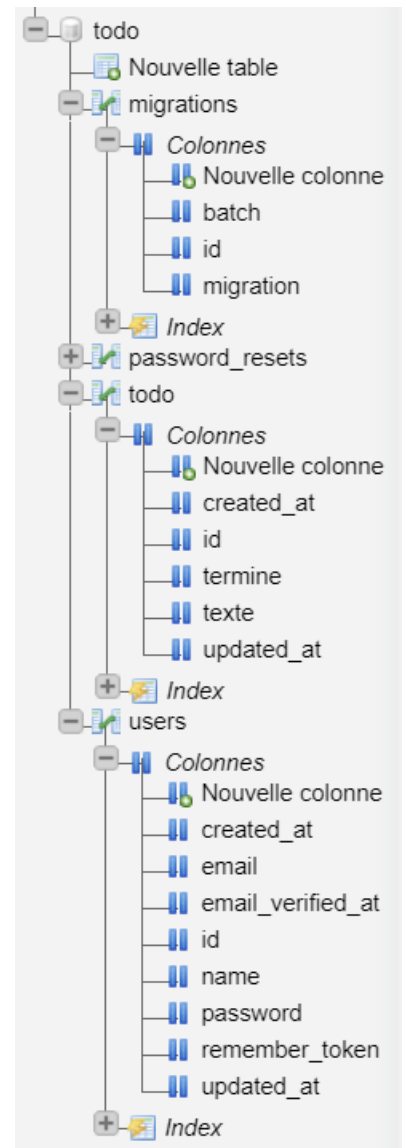
Modèle

Le modèle

Maintenant que nous avons fait le script de création / migration, nous allons définir notre modèle.

```
$ php artisan make:model Todos
```

La commande va créer le fichier `Todos.php` dans le dossier `app/`.



Ajouter dans la class :

```
protected $fillable = ['texte', 'termine'];
```

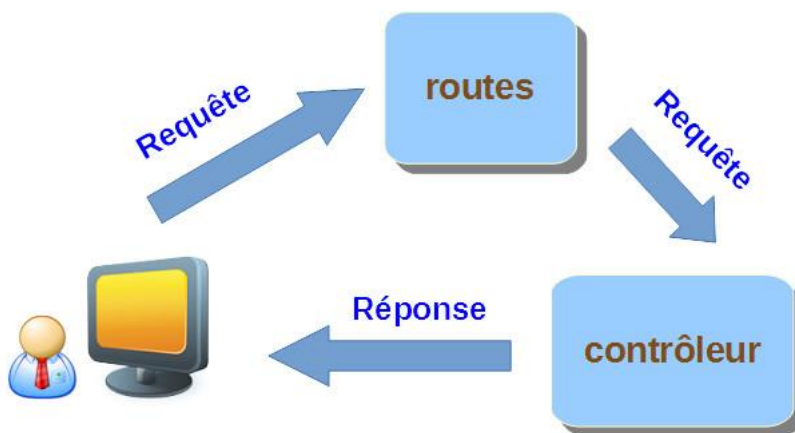
⚠ Cette propriété est optionnelle, elle permet vous autorisera plus tard à faire du « mass-assignment » c'est à dire à créer un objet « Todos » depuis par exemple le POST HTTP.

Le controller

Maintenant que nous avons la migration et le modèle de notre base de données, nous allons créer le controller. Pour rappel le controller va faire « le lien » entre la base de données et les appels HTTP. C'est ici que nous allons mettre la logique métier de notre application.

Rôle

La tâche d'un contrôleur est de réceptionner une requête (qui a déjà été triée par une route) et de définir la réponse appropriée, rien de moins et rien de plus. Voici une illustration du processus :



- Les contrôleurs servent à réceptionner les requêtes triées par les routes et à fournir une réponse au client.
- Artisan permet de créer facilement un contrôleur.
- Il est facile d'appeler une méthode de contrôleur à partir d'une route.
- On peut nommer une route qui pointe vers une méthode de contrôleur.

Pour commencer nous allons créer « la structure de base » de notre controller.

```
> php artisan make:controller TodosController
```

Le fichier `TodosController.php` viens d'être créé dans le chemin suivant `app/Http/Controllers/`.

Notre code est maintenant prêt. Nous allons créer les méthodes permettant la manipulation de notre base de données tout en répondant à nos problématique d'interface (liste, création, terminer, suppression).

Nous allons maintenant écrire une méthode pour chaque action. Avec les différentes conditions nécessaires au bon fonctionnement de l'application.

La méthode « Liste »

Corriger le chemin de la database
La méthode « Liste »

La méthode `liste` est certainement la plus simple, nous allons faire appel à la méthode `all()` de Eloquent (ORM pour l'accès à la base de données). Pour ça créez une nouvelle méthode dans la Class `TodoController` avec le code suivant.

Pour l'instant nous n'allons pas intégrer Éloquent, mais uniquement définir notre méthode.

```
public function liste() {  
    return "Liste";  
}
```

Rien de bien compliqué, comme vous pouvez le voir.

Les autres méthodes

✋ Pour l'instant nous allons nous arrêter là pour la partie code PHP. Cette méthode est suffisante pour « tester » le premier template que nous allons écrire.

Ajout route /

Nous allons tester la route /, pour ça nous allons remplacer le contenu du fichier `routes/web.php` par :

```
Route::get('/', "TodosController@liste");
```

Vous pouvez relancer votre serveur de test, vous devez maintenant voir afficher « Liste ».

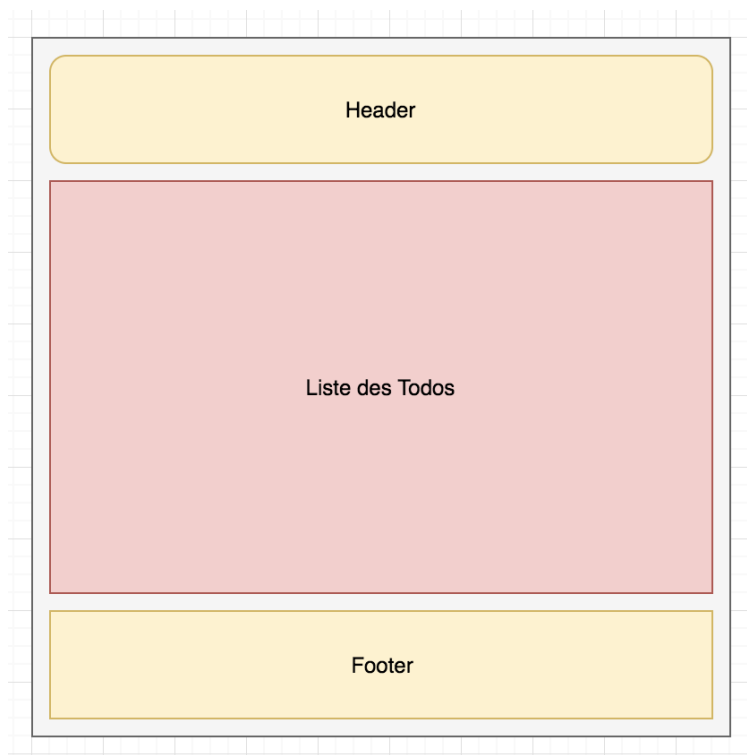
Créer les templates

Nous avons donc maintenant :

- La structure Laravel.
- La définition de notre base de données.
- Les dépendances clients (bootstrap 4...)

Nous allons donc pouvoir commencer la création des templates. L'organisation du code est quelques choses d'important, elle n'est pas à négliger. Un code organisé est un code agréable à rédiger.

Notre vue va être découpée en 3 partie :



- ✋ Pourquoi le découpage en « 3 templates » est-il important ?

- Quelle est l'avantage pour le développeur ?

Le template principal

Nous allons commencer par définir notre « Template principal » celui-ci va contenir l'ensemble des éléments partagés sur toutes nos pages à savoir :

- Les JS
- Les CSS
- La structure commune

```
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>@yield('title')</title>

    <link href="{{ asset('css/app.css') }}" rel="stylesheet">
    <script type="text/javascript" src="{{ asset('js/app.js') }}"></script>
  </head>
  <body>
    <nav class="navbar navbar-expand-md navbar-dark bg-dark fixed-top">
      <a class="navbar-brand" href="#">Ma Todo List</a>
    </nav>

    @yield('content')

  </body>
</html>
```

Maintenant que nous avons le contenu, nous devons créer un nouveau fichier.

- Créer un nouveau fichier `resources/views/template.blade.php`
- Copier-Coller le html dans le fichier.

👏👏 Bravo, vous venez de créer votre premier template.

Questions

- Pourquoi le fichier est-il nommé `...blade.php` ?
- À quoi correspond `{{ asset('...') }}` ?
- À quoi correspond la directive « `yield` » ? En quoi est-ce très important ?
- Est-il possible de définir une « zone » où d'autres ressources JS/CSS seront insérées lors de l'exécution ? [Voir la documentation](https://laravel.com/docs/5.6/blade#stacks) : <https://laravel.com/docs/5.6/blade#stacks>
- Maintenant que vous avez la réponse, ajouter une « stack » pour le script et le style dans l'entête.

Pour tester, mettez en commentaires la route vers le contrôleur et remplacez par une route vers le template.

Tester

Une fois votre serveur de développement lancé, allez sur <http://localhost:8000>.

Que se passe-t-il ?

→ Changer votre couleur de fond dans le css base pour voir ?
Visualiser votre code source de page. Que se passe-t-il ?

Installation des dépendances client

Maintenant que la partie base de données est prête, nous allons nous occuper de la partie visualisation de notre application / site web. Nous allons utiliser bootstrap pour gérer la problématique d'affichage et de responsive.

Attention méthode alternative en cas de soucis avec NodeJS et npm.

Si pas de souci, reprendre le TP à **Installation de Bootstrap 4 via NodeJS + NPM**

Ajout des libraries

La structure de base de Laravel n'intègre pas Bootstrap, par contre il est possible de l'installer dans votre projet.

La façon la plus rapide est d'intégrer directement dans le <head> de votre projet les librairies suivantes :

Éditer le fichier `scripts.blade.php` pour ajouter :

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
integrity="sha384-ZMP7rVo3mIyKv+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIpM49"
crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE24lrYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>
```

Éditer le fichier `styles.blade.php` pour ajouter :

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdKnLPMO"
crossorigin="anonymous">
```

Vous venez d'ajouter JQuery, Bootstrap, et l'ensemble des éléments nécessaires à son bon fonctionnement. Notre site a également besoin d'une CSS spécifique. Celle-ci doit être ajoutée manuellement.

Créer un fichier `main.css` dans `public/css`. Mettre le contenu suivant :

```
body{
    padding-top: 5rem;
}

form.add{
    padding-bottom: 10px;
}
```



```
.pull-right{
    float: right;
}

.action > .btn{
    padding: 1px 7px 1px;
}

.oi{
    font-size: small;
}
```

Ajouter à nouveau cette CSS dans le fichier `styles.template.php` de votre site :

```
<link href="{{ asset('css/main.css') }}" rel="stylesheet">
```

Installation de Bootstrap 4 via NodeJS + NPM

👉 Si vous lisez ceci c'est que vous êtes sur votre machine, si c'est le cas, sachez que c'est la meilleur façon de faire. (**Attention** à ne pas avoir les librairies en HTTP et Via NodeJS).

Ajouter Bootstrap 4

La structure de base de Laravel n'intègre pas Bootstrap, par contre elle intègre un système de gestion de dépendances. Nous allons nous servir de cette gestion de dépendance pour ajouter bootstrap (version 4). Dans la ligne de commande :

```
$ php artisan preset none
$ php artisan preset bootstrap
```

Patiencez quelques instants... bootstrap est maintenant disponible pour votre projet. Mais pour qu'il soit accessible pour vos templates nous devons « le compiler ».

Préparation des « Assets » bootstrap ...

Laravel inclut une configuration « `webpack.mix.js` », celle-ci permet de fusionner l'ensemble des JS et CSS en un seul fichier pour gagner en performance.

Sans entrer dans le détail, la compilation des ressources (assets) est réalisée avec Webpack. Webpack est un outils NodeJS très puissant mais qui peut être complexe, nous allons donc uniquement l'utiliser.

- Installer [NodeJS version current](https://nodejs.org/en/download/current/) (<https://nodejs.org/en/download/current/>)

Une fois installé retourner dans le dossier de votre projet faites les commandes suivantes :

laravel-mix ?

laravel-mix est un outil fourni de base dans Laravel qui gèrent la partie libraries clientes. La configuration de celui-ci se fait dans le fichier `webpack.mix.js`

Le contenu initial est :

```
const mix = require('laravel-mix');

/*
|-----
| Mix Asset Management
|-----
|
```

```

| Mix provides a clean, fluent API for defining some Webpack build steps
| for your Laravel application. By default, we are compiling the Sass
| file for the application as well as bundling up all the JS files.
|
*/

mix.js('resources/js/app.js', 'public/js')
    .sass('resources/sass/app.scss', 'public/css');
Modifier le SCSS

```

Remplacer le fichier `resources/sass/app/scss` par :

```

// Fonts
@import url('https://fonts.googleapis.com/css?family=Nunito');

// Variables
@import 'variables';

// Bootstrap
@import '~bootstrap/scss/bootstrap';

.navbar-laravel {
    background-color: #fff;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.04);
}

body{
    padding-top: 5rem;
}

form.add{
    padding-bottom: 10px;
}

.pull-right{
    float: right;
}

.action > .btn{
    padding: 1px 7px 1px;
}

.oi{
    font-size: small;
}

```

Transpiler

```

> npm install
> npm run production

```

🔍 [Plus d'informations Webpack.mix.js](#)

Questions

Questions

- Quels fichiers ont été créés ?
- Que contient le fichier `webpack.mix.js` ?

- `webpack.mix.js` fait référence à des fichiers dans `ressources/js/*` et `ressources/sass/*`, allez y jeter un coup d'oeil (même si dans ce projet nous n'allons rien modifier).
- Pourquoi la fusion / compilation des ressources est-elle si importante ?

Modifications demandés

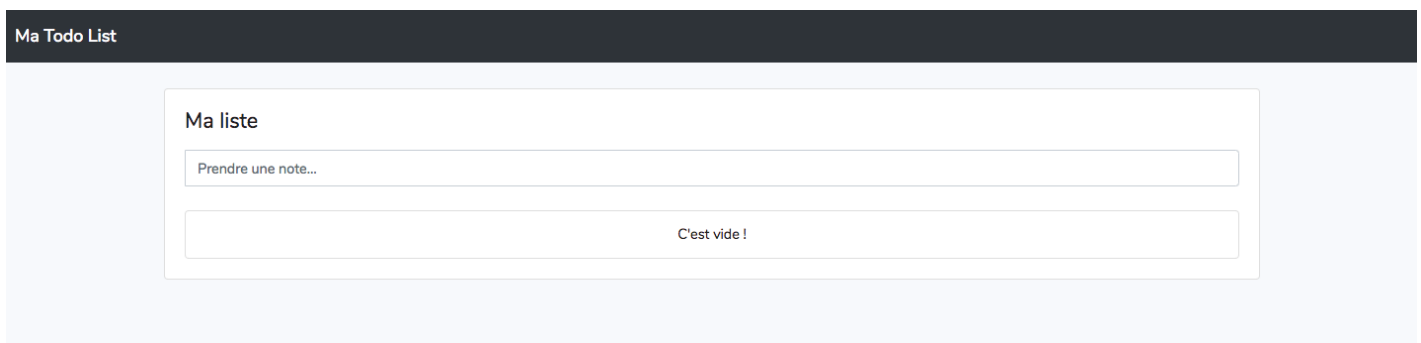
👉 Cette modification n'est nécessaire que dans le cas où vous n'utilisez **pas** NodeJS + NPM.

Comme vous l'avez vu avec le fichier `main.css` il est possible d'avoir un CSS / JS en local dans le dossier public.

- Télécharger l'ensemble de librairies `http / https` (css + js) dans le dossier public (attention à bien les ranger) et les utiliser dans votre head.

Question Liste des todos

Maintenant que nous avons défini notre template de base nous allons pouvoir définir notre page principale, la page « liste des todos ». Une fois terminée celle-ci va ressembler à :



Avant de commencer la réalisation de ce template, regardons ce que l'on peut y voir :

- Un formulaire « form »
- Une « liste »

Nous allons donc avoir besoin de composant bootstrap. Première étape regarder [la documentation de bootstrap \(https://getbootstrap.com/docs/4.0/components/alerts/\)](https://getbootstrap.com/docs/4.0/components/alerts/) !

Questions

- Quels composants (components) allons-nous avoir besoins ?
Gestion des messages d'erreur pour gérer la saisie dans le formulaire
- Est-ce que ce sont les seuls ?

Définition du template « Liste » / « Home »

Nous allons créer un 2nd template celui qui va être chargé d'afficher la liste des todos.

Créer un nouveau fichier `resources/views/home.blade.php` et y mettre le contenu suivant :

```
@extends("template")

@section("title", "Ma Todo List")

@section("content")
    <div class="container">
        <div class="card">
            <div class="card-body">
                <!-- Action -->
                <form action="/action/add" method="post" class="add">
                    <div class="input-group">
                        <span class="input-group-addon" id="basic-addon1"><span
class="oi oi-pencil"></span></span>
                        <input id="texte" name="texte" type="text" class="form-
control" placeholder="Prendre une note..." aria-label="My new idea" aria-
describedby="basic-addon1">
                    </div>
                </form>

                <!-- Liste -->
                <ul class="list-group">
                    @foreach ($todos as $todo)
                        <li class="list-group-item">
                            <span>{{ $todo->texte }}</span>
                            <!-- Action à ajouter pour Terminer et supprimer -->
                        </li>
                    @empty
                        <li class="list-group-item text-center">C'est vide !</li>
                    @endforeach
                </ul>
            </div>
        </div>
    </div>
@endsection
```

Questions

- @extends ? À quoi sert cette directive, pourquoi « template »?
- Modifier le titre affiché dans la barre.
- À quoi correspond le @foreach ... @empty ... @endforeach?

Et maintenant ?

Bien... récapitulons ! Nous avons :

- Le modele.
- Le contrôleur.
- Les templates (template et home).

Il faut maintenant assembler l'ensemble pour que votre page s'affiche lors d'une requête.

Création des routes

La définition des routes se fait dans le fichier `routes/web.php` pour l'instant vous devez avoir qu'une seule route de définie.

Supprimer là, et ajouter :

```
Route::get('/', "TodosController@liste");
```

Question

- À quoi correspond la notation `TodosController@liste` ?

Tester

Vous voyez « Liste » ? C'est OK !

Affichage de la liste

Bon afficher « Liste » c'est un bon début... Modifier la méthode `liste()` pour qu'elle ressemble à :

```
return view("home", ["todos" => Todos::all()]);
```

Tester

La page s'affiche ? Super !

- Ajouter une entrée « à la main » grâce à l'explorateur de base de données de PHPStorm.

Rafraichissez la page, vous devez maintenant voir votre texte.