LARAVEL (8-2021) – FRAMEWORK PHP 3

SOMMAIRE

https://laravel.com/
https://laravel.com/docs/10.x
https://laravel.com/docs/11.x

ETAPE 3

https://laravel.com/
https://laravel.com/docs/10.x
https://laravel.com/docs/11.x

0-1 - Présentation

⇒ L'étape 3 :

⇒ Présente le MVC

⇒ Découvre la création des routes

⇒ Découvre la création des controleurs

⇒ Découvre la création des vues

⇒ On fait ça avec un page « apropos » en MVC avec sa route, son contrôleur et sa vue

⇒ Mise à jour de la page d'accueil

⇒ L'objectif de l'étape 3 est de se familiariser avec le MVC dans Laravel.

⇒ Ca vient approfondir l'étape 3.0 sur le contrôleur de ressources.

Principe du document

- Les parties surlignées de bleu correspondent à des installation ou exercices à faire.
- Les parties surlignées en jaune sont des concepts importants.
- Les parties marquées (on peut sauter) peuvent être sautées.

Installation de la nouvelle étape

On va tout réinstaller

➢ ***Installer Laravel***

Dans un terminal, exécuter la commande suivante :

```
composer create-project laravel/laravel matodolist_e3
cd matodolist_e3
```

Puis, ouvrir le projet dans VS Code.

➢ ***Configurer l'environnement***

Dans `.env`, configurer la base de données :

```
DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE= matodolist_e3
DB_USERNAME=root
DB_PASSWORD=
```

➢ ✅ ***Créer la base de données dans MySQL (`todo_list`).***

```
CREATE DATABASE matodolist_e3 ;
```

➢ ✅ ***Installer Laravel Breeze***

Breeze gère **l'authentification** (login, inscription, réinitialisation de mot de passe).

Dans un terminal (dans le dossier « matodolist_e3 » , exécuter les commandes suivantes, une par une pour y voir clair :

```
composer require laravel/breeze --dev
```
`php artisan breeze:install` : faire tous les choix par défaut
```
php artisan migrate
npm install && npm run dev
```

Dans un autre terminal (dans le dossier « matodolist_e3 » , exécuter cette commande :

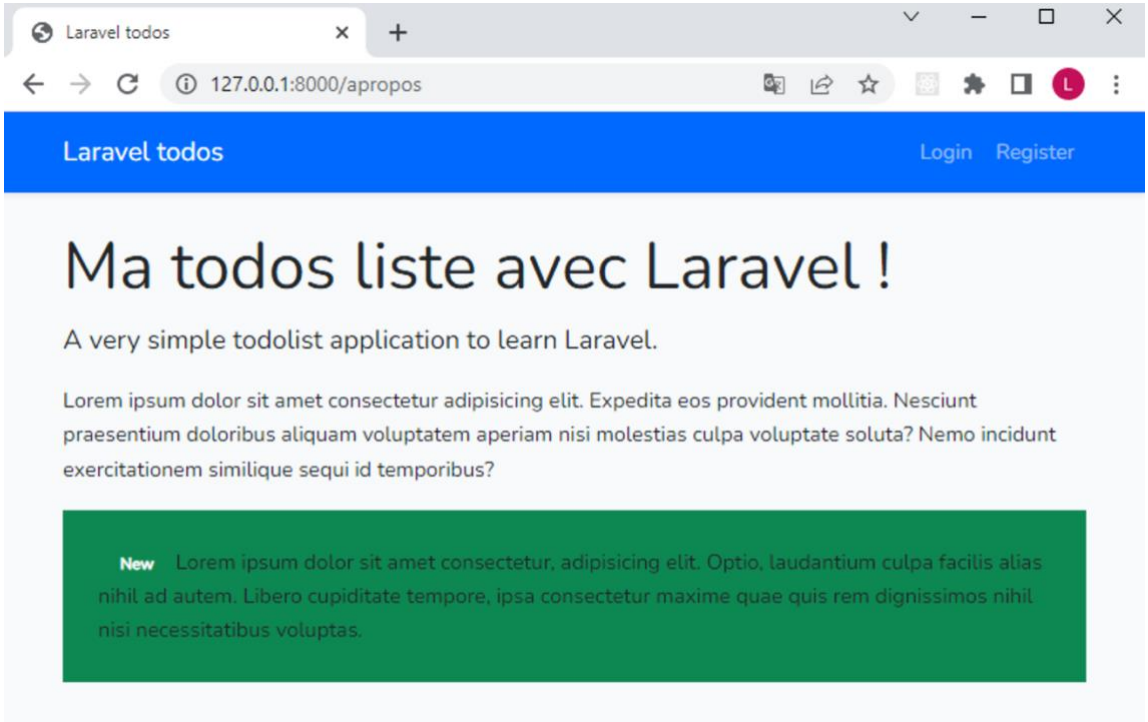
```
php artisan serve
```

➢ ✅ ***Laravel Breeze est prêt !***

On peut tester l'inscription et la connexion.

Objectifs de l'étape 3

- Créer la route "apropos" avec son controleur et sa vue.
- Faire de la route « apropos » la page d'accueil.



0-2 – Rappel : le MVC

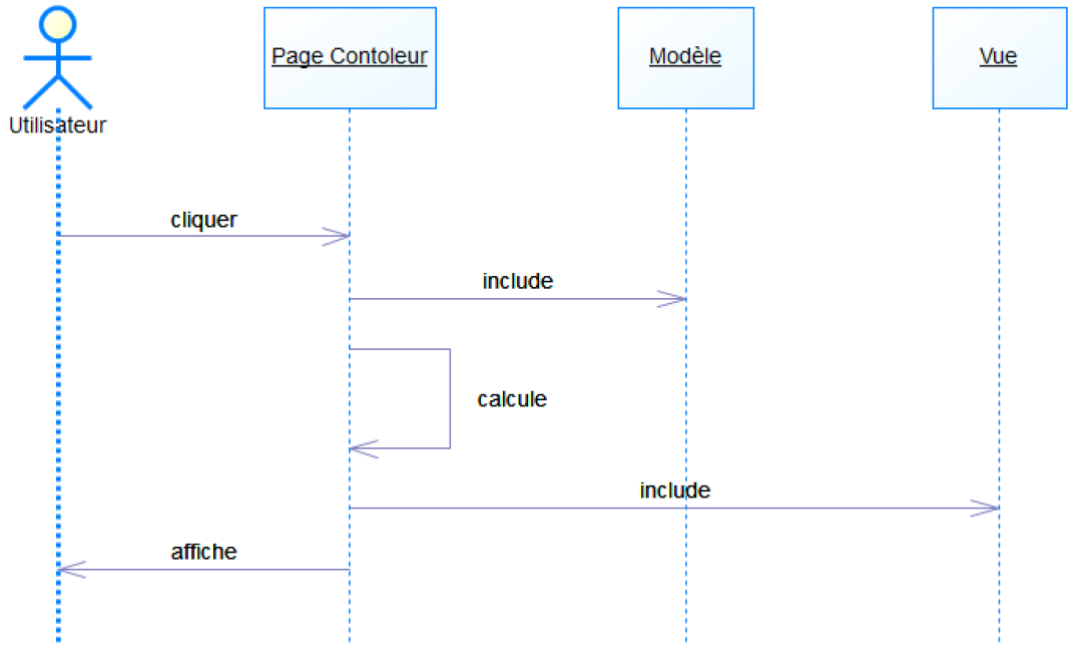
- Avant de présenter la création d'un controller dans Laravel, on fait un rappel du MVC.

MVC : Modèle - Vue - Contrôleur.

- L'architecture MVC sépare la logique du code en trois parties, trois ensembles de fichiers :
  - le **modèle** (qui correspond au SQL)
  - la **vue** (qui correspond au HTML)
  - le **contrôleur** (qui correspond au PHP faisant le lien entre les deux précédent).

- Cela rend le code plus facile à mettre à jour et permet d'organiser le travail en 3 parties et donc de travailler en parallèle.
- L'architecture MVC est une bonne pratique de programmation.
- La connaissance de l'architecture MVC rend capable de créer un site web de qualité et facile à maintenir.
- En pratique, les architectures MVC mises en œuvre s'appuient sur la théorie mais l'adaptent de façon pragmatique. Il y a donc plusieurs façons de mettre en œuvre le MVC.
- Les principaux framework sont développés en MVC : Laravel, Symfony, CodeIgniter, CakePHP, Zend Framework, etc.

Fonctionnement global



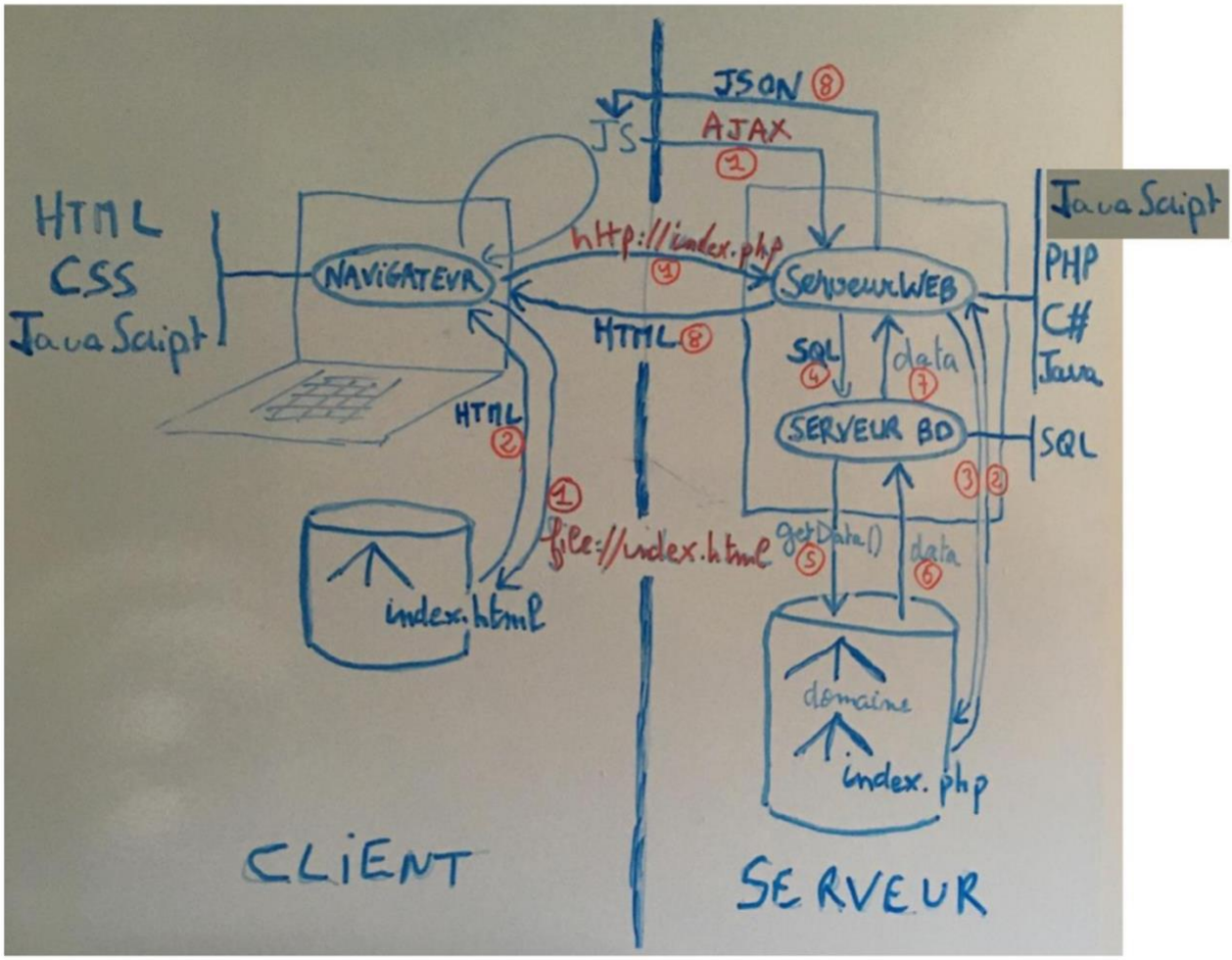
- Ce diagramme est un diagramme de séquence.
- Il est simplifié et adapté à du code PHP

1. Un utilisateur voit une page et clique sur une choix qui fait appel à un contrôleur (en général via un href ou un input)
2. Le contrôleur include un Modèle
3. On aurait pu ajouter que le contrôleur récupère des données de son modèle.
4. Le contrôleur fait des calculs.
5. Le contrôleur include la Vue en passant les datas récupérés et calculés en paramètre.
6. Le contrôleur affiche la vue pour l'utilisateur

- De là, l'utilisateur peut repartir à l'étape 1

Rappel d'une architecture WEB de base.

- On demande un fichier « contrôleur » qui est traduit par le serveur WEB : il commence par inclure le modèle, puis il fait le calculs du contrôleur et enfin il inclut la vue pour produire le code HTML à renvoyer au client.



Le Contrôleur (PHP)

- **Le contrôleur est la page appelée (le véritable index, autrement dit, le « main »).**
- Il fonctionne en trois étapes :

1. Il utilise les fonctions du Modèle (include et appels aux fonctions).
2. Il analyse et traite les données issues du Modèle mais aussi celles passées en paramètre à l'appel de la page ($_GET, $_POST, $_SESSION). Il détermine par exemple si le visiteur a le droit de voir la page ou non.
3. En fonction de ses calculs, il appelle la vue correspondante par un include.

- **C'est une page pur PHP.**

- **Le contrôleur est le « chef d'orchestre »** :
  - ⇒ Il récupère la demande de l'utilisateur à travers la vue (la page HTML) par un href, un formulaire ou un header.
  - ⇒ Il échange des données avec le modèle,
  - ⇒ Il fait les calculs (qui peuvent être complexes)
  - ⇒ Il choisit une vue à afficher en lui fournissant les variables.

- Le **rôle du contrôleur** peut se limiter à faire le lien entre le modèle et la vue : **de la colle** !

Le Modèle (SQL)

- Le modèle gère les données du site : essentiellement les accès à la BD. Mais aussi la gestion de fichiers.
- Il propose des fonctions pour faire des Insert, des Update, des Delete, des Select.
- Ces fonctions peuvent renvoyer des tableaux de données.
- Les résultats seront exploités par le contrôleur mais aussi par le HTML.
  - ⇒ **C'est une page pur PHP.**

- Les framework permettent de mettre à jour à la BD facilement et de façon itérative grâce à la commande php artisan migrate.

La Vue (HTML)

- La vue affiche la page HTML. Elles récupèrent des variables du Contrôleur et/ou du Modèle pour savoir ce qu'elles doivent afficher.

- **C'est une page HTML avec quelques boucles et conditions PHP très simples**.
  - ⇒ Le php sert à afficher données issues du Modèle.

- La vue contient le **DOCTYPE** mais elle ne peut fonctionner qu'avec le contexte du contrôleur.

MVC : design pattern tête la première

- Le MVC fonctionne aussi de la POO. En P.O.O. on parle de « Design Pattern ».
- Il y a un vieux livre sur les design pattern proposait la chanson du contrôleur :

*Car si **l'modèle** - est **essentiel***
*Et si la **vue** - est **magnifique***
J'suis p'têt feignant - oui mais c'est fou
*Ces lignes de code - qui sont **d'la colle***
C'code qui n'fait rien - d'vraiment magique
Ne fait qu'transmettre - que des valeurs

- Le contrôleur, c'est de la colle : il colle le modèle et la vue.

- On dit parfois que le MVC est un design pattern (DP). C'est en réalité un assemblage de DP élémentaires (un par lettre) : les DP « stratégie », « composite » et « observateur » (les DP ont des noms).
- Si on code réellement ces DP, alors on aura une mise à jour automatique des notifications (DP observateur).

1 – Création d'une route apropos (v6)

- 1.1 : Syntaxe des routes
- 1.2 : Création d'une page « a propos »
- 1.3 : TP Route vers la page « a propos » : ça appelle un contrôleur qui n'existe pas.

1.1 : Programmation de routes : routes/web.php

Route simple

➢ ***Principes:***

- On va modifier le fichier de routes : routes/web.php

➢ ***Syntaxe :***

```
Route:methode_http('/nom_route', action() )
```

- Une méthode HTTP : GET, POST, etc.
- Un nom de route : l'url pour notre application
- Une action : une fonction à coder

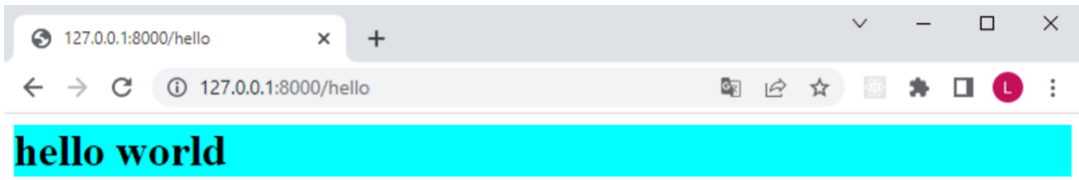
➢ ***Exemple***

```
Route::get('/hello', function(){
    return '<h1 style="background-color:aqua" >hello world</h1>';
});
```

- Une méthode http : get : on va uniquement afficher un résultat
- Un nom de route : /hello : on peut passer cette URL au navigateur
- Une action : la fonction return ce qui est renvoyé au navigateur.

➢ ***Test***

```
:8080/hello : ça marche
:8080/hello2 : ça ne marche pas
```



➢ ***Consultation des routes***

```
>php artisan route:list
```

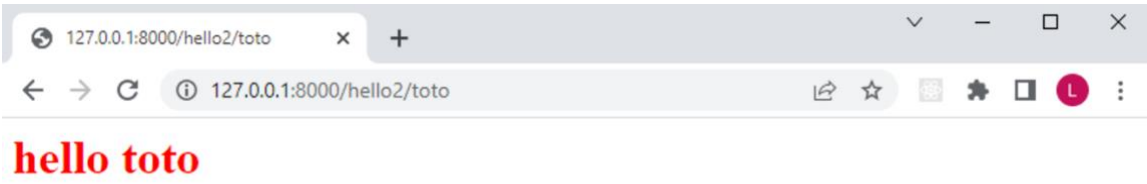
Route avec paramètre dans l'url

- ***Exemple***

```
Route::get('/hello2/{name}', function($name){
     return '<h1 style="color:red" >hello '.$name.'</h1>';
});
```

- ***Test***

```
:8080/hello2/toto : ça marche
:8080/hello2 : ça ne marche pas : le paramètre est obligatoire
```



- ***Consultation des routes***

```
>php artisan route:list
```

Route avec paramètre facultatif dans l'url

- ***Syntaxe***
  - Paramètre par obligatoire : ? et valeur par défaut

- ***Exemple***

```
Route::get('/hello3/{name?}', function($name='nobody'){
    return '<h1 style="color:red" >hello '.$name.'</h1>';
});
```

- ***Test***

```
:8080/hello3/toto : ça marche
:8080/hello3 : ça marche
```

- ***Consultation des routes***

```
>php artisan route:list
```

Attention à l'ordre et au nom des routes : lecture du fichier routes/web.php

- Si on met 2 fois la même route, la 2ème écrase la 1ère.
- Il faut donc faire attention à ne pas mettre 2 fois la même route.

Documentation Laravel : routing

- ***Site officiel***
  https://laravel.com/docs/12.x/routing
- ***Sur l'injection de dépendances***
  https://putaindecode.io/articles/injection-de-dependances-en-php/

1.2 : Création d'une page "apropos" dans notre site

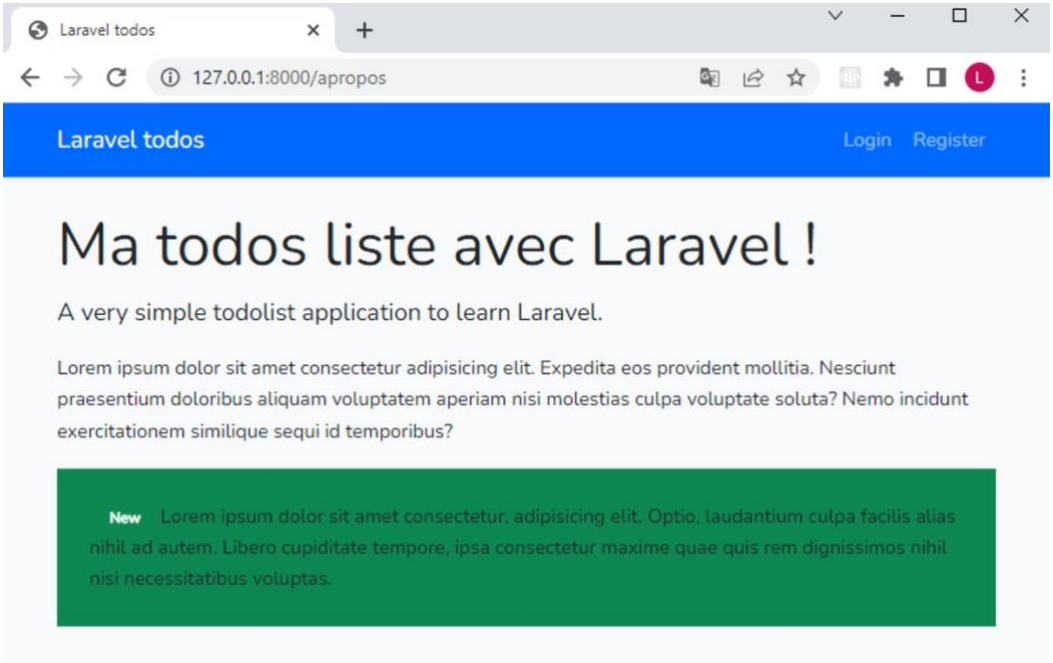
Objectifs

- Création d'une route « apropos »

  ⇒ Fichier \ routes \ web.php
- Création d'un contrôleur pour la route

  ⇒ Fichier \ App \ http \ Controllers \ AproposController.php
- Création d'une vue pour le contrôleur

  ⇒ Fichier \ ressources \ views \ apropos \ index.blade.php



Structure de la route « home » de notre site :

- ***Les routes dans routes/web.php***

  - Actuellement, on a une route home sur notre site :

```
Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');
```

  - C'est la route du « dashboard » qui appelle la route login si on n'est pas connecté.

  - Il y a aussi :

```
Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])-
>name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])-
>name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])-
>name('profile.destroy');
});
```

  - C'est la route du « profile » qui qui appelle la route login si on n'est pas connecté.

  - Variante :

    ⇒ La syntaxe peut changer selon les versions. Cette autre syntaxe fonctionne :

```
Route::get(
    '/home,
    [App\Http\Controllers\HomeController::class, 'index']
)->name('apropos');
```

- Dans le code de la route, on trouve :
  - ⇒ Une méthode HTTP : get et patch et delete
  - ⇒ Un nom de route : l'url pour notre application : /dashboard ou /profile
  - ⇒ Soit : directement le return de la vue
  - ⇒ Soit une classe (au sens POO) : ProfileController et une méthode associée qui sera à coder : « edit », « update », « destroy ». Le patch est pour l'update, le put pour l'insert.
  - ⇒ Un nom logique pour la route : ->name()
    - ⇒ On pourra récupérer l'url avec la fonction route('nom_de_route')
    - ⇒ Ça permet de changer les routes sans changer le code (il y a un nom physique et un nom logique).
    - ⇒ https://laravel.com/docs/9.x/routing#named-routes
- Précisions
  - ⇒ ProfileController : c'est le nom du fichier contrôleur. Tous les contrôleurs sont au même endroit : App\Http\Controllers
- Variante :
  - ⇒ Cette variante n'est pas dans notre code mais fonctionne :
  - ⇒ [App\Http\Controllers\HomeController::class, 'index']
  - ⇒ La classe du contrôleur, ici HomeController, est préfixée par tout le chemin.
  - ⇒ La méthode par défaut sera « index »

Création de la route « apropos »

- Dans le fichier routes/web.php, on reprend la même syntaxe que pour « home » imaginaire, avec le chemin :

➢ ***La route***

```
Route::get(
    '/apropos',
    [App\Http\Controllers\AproposController::class, 'index']
)->name('apropos');
```

⇒ Notez que cette une technique compliquée : il faut mettre des « \ » dans le chemin.

1.3 : TP Routes

TP routing de base

- Tester les exemples présentés : routes hello, hello2, hello3
- Regardez la documentation Laravel

TP : route « apropos »

- La route « apropos » appelle le controller : App\Http\Controllers\AproposController.
- La route apropos plante : le controller n'est pas défini.

2 - Création d'un contrôleur (v6)

- 2.1 : dossier des contrôleurs : App/http/Controllers
- 2.2 : php artisan make:controller : pour créer un fichier controleur
- 2.3 : TP Controller vers la view « a propos » : ça appelle une vue qui n'existe pas.

2.1 : Le dossier des contrôleurs : App /Http /Controllers

- Il y a déjà un contrôleur : ProfileController.php.
- On va ajouter le contrôleur : AproposController.php

2.2 : Ajout d'un contrôleur avec php artisan make :controller

php artisan make

- On peut créer ce contrôleur en utilisant la commande « php artisan make »
- Liste des commandes php artisan

```
> php artisan   ->  liste les commandes
...
make
...
  make:controller      Create a new controller class
...
```

- Aide php artisan make:controller

```
> php artisan make:controller –help
...
Arguments:
  name                    The name of the class
...
```

Ajout avec php artisan make :controller

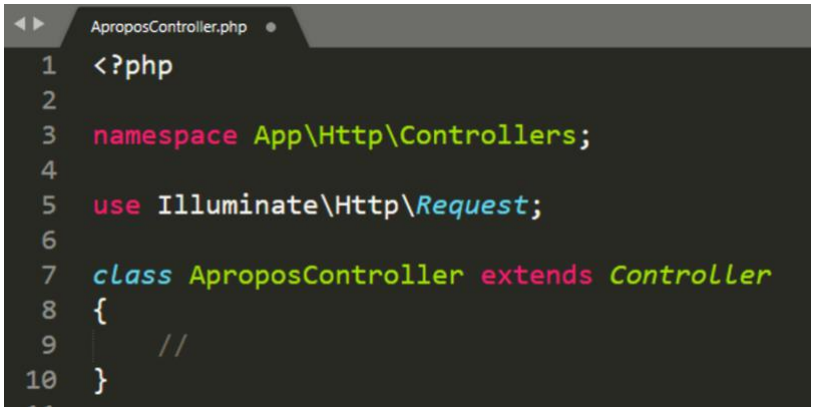
- Création de la classe

```
> php artisan make:controller AproposController
```

⇒ Dans le dossier App /Http /Controllers : AproposController.php

⇒ C'est un squelette :

⇒



- Dans le fichier AproposController.php appelle la vue :

⇒ Dans la classe on ajoute la méthode index.

⇒ Cette méthode ne fera que retourné un fichier de views nommé « apropos »

⇒ Ce sera un fichier « blade » : un template qui intègre du HTML et du PHP.

```
public function index()
{
    return view('apropos'); # fichier resources/views/apropos.blade.php
}
```

Test de la route « apropos »

- La route « apropos » appelle le controller : App \ http \ Controllers \ AproposController.
- Le controller appelle la vue : ressources \ views \ apropos.blade.php
- La route apropos plante : la vue n'est pas définie.

Ajout « à la main »

- On peut aussi créer ce contrôleur « à la main » :

  ⇒ On peut créer le fichier complètement

  ⇒ Ou faire un copier-coller d'un existant et mettre à jour en commentant de façon adaptée.

2.3 : TP Contrôleur

Création du contrôleur AproposController avec php artisan

- Suivre les étapes de la création avec php artisan et tester le résultat.

Ensuite « à la main »

- Suivre les étapes de la création à la main et tester le résultat.
- Travailler sur une 2ème route avec un 2ème contrôleur : apropos2

3 - Création d'une vue (v6)

- 3.1 : dossier des vues : ressources/views.
- 3.2 : vue avec TailWind : garder le menu Breeze de connexion dans nos pages.
- 3.3 : TP Vue : version simple h1, version TailWind. La page A propos fonctionne.

3.1 : Le dossier des vues : ressources / views

- 2 approches :

  ⇒ L'une ou la vue est directement dans views.

  ⇒ L'autre ou la vue est dans un dossier qu'on crée : on gardera celle-là.

Première approche : /views/apropos.blade.php

- On crée un fichier dans le répertoire « views » : apropos.blade.php
- Dans ce fichier on met :

```
<h1 style="color:blue">A propos : /views/apropos.blade.php </h1>
```

- La route apropos fonctionne :8080/apropos

  ⇒ La route mène au controler qui mène à la vue

Deuxième approche : /views/apropos/index.blade.php

- Dans le répertoire « views », on crée le dossier apropos et le fichier index.blade.php
- Dans ce fichier on met :

```
<h1 style="color:red">A propos : /views/apropos/index.blade.php </h1>
```

- On change le fichier AproposController.php :

```
public function index()
{
    return view('apropos/index'); # fichier apropos.blade.php
}
```

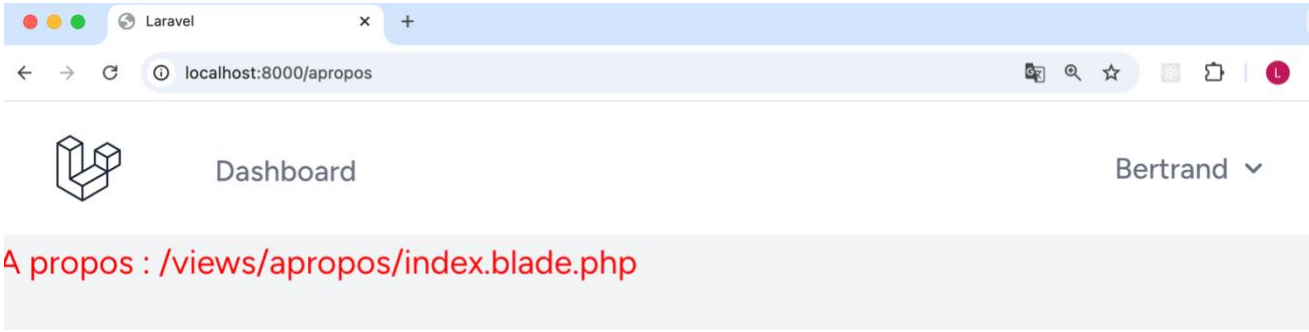
- La route apropos fonctionne :8080/apropos

  ⇒ La route mène au contrôleur qui mène à la vue

3.2 : Vue avec TailWind - Intégration de breeze

D'abord sur notre <h1>

- On veut garder les menus de Breeze dans l'apropos :



- Dans /views/apropos/index.blade.php :

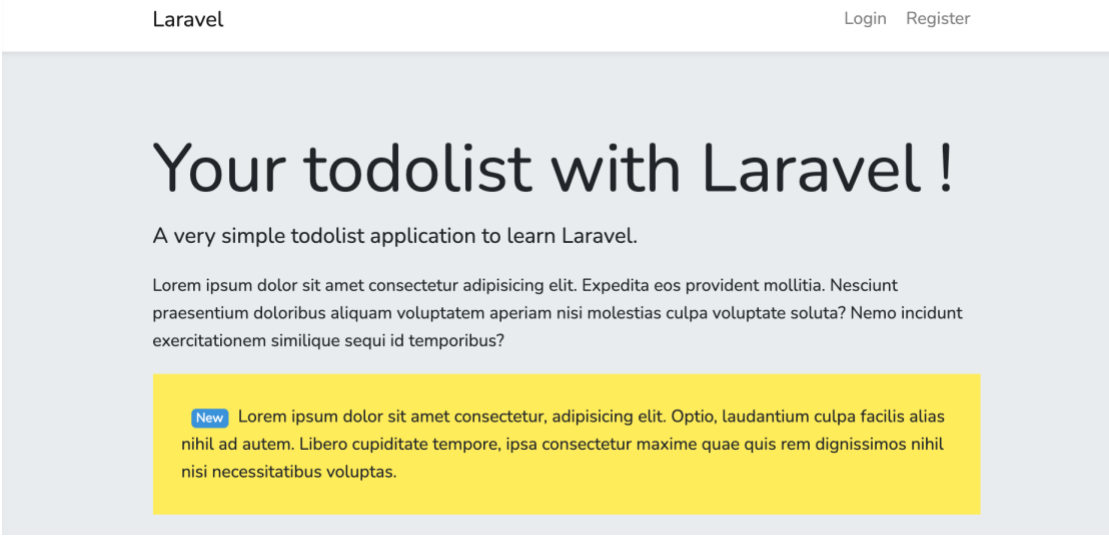
```
<x-app-layout>
    <h1 style="color:red">A propos : /views/apropos/index.blade.php </h1>
</x-app-layout>
```

- Avec cette écriture, notre h1 va se placer dans le « layout » :

  ⇒ C'est le fichier /views/layouts/app.blade.php

  ⇒ Notre fichier se place dans {{ slot }} qui est dans la balise <main>.

Ensuite pour produire cet apropos :



- Le HTML TailWind :

  ⇒ TailWind ressemble à Bootstrap, mais c'est du TailWind !

  ⇒ Il faut démarrer Vite pour que ça fonctionne : nmp run dev dans un terminal à part.

```
<main class="container mx-auto p-4">
  <div class="bg-gray-200 py-12">
    <div class="container mx-auto text-center">
      <h1 class="text-5xl font-extrabold text-gray-800">Your todolist with Laravel!</h1>
      <p class="mt-4 text-lg text-gray-600">A very simple todolist application to learn Laravel.</p>
      <p class="mt-4 text-gray-700">
        Lorem ipsum dolor sit amet consectetur adipisicing elit.
        Expedita eos provident mollitia.
        Nesciunt praesentium doloribus aliquam voluptatem aperiam nisi molestias culpa voluptate soluta?
        Nemo incidunt exercitationem similique sequi id temporibus?
      </p>
      <p class="mt-4 bg-yellow-300 p-4 text-gray-800">
        <span class="bg-blue-500 text-white px-2 py-1 rounded-full mx-2">New</span>
        Lorem ipsum dolor sit amet consectetur, adipisicing elit. Optio, laudantium culpa facilis alias nihil ad autem.
        Libero cupiditate tempore, ipsa consectetur maxime quae quis rem dignissimos nihil nisi necessitatibus voluptas.
      </p>
    </div>
  </div>
</main>
```

- La route apropos fonctionne :8080/apropos

Modification de la mise en page : CSS :

- ***dans apropos / index.blade.php :***

  ⇒ On peut paramétrer des éléments TailWind : des couleurs, des tailles, etc.

- ***Quelques infos TailWind :***

  ⇒ https://v2.tailwindcss.com/docs

3.3 : TP Vues

TP de création de la vue A propos

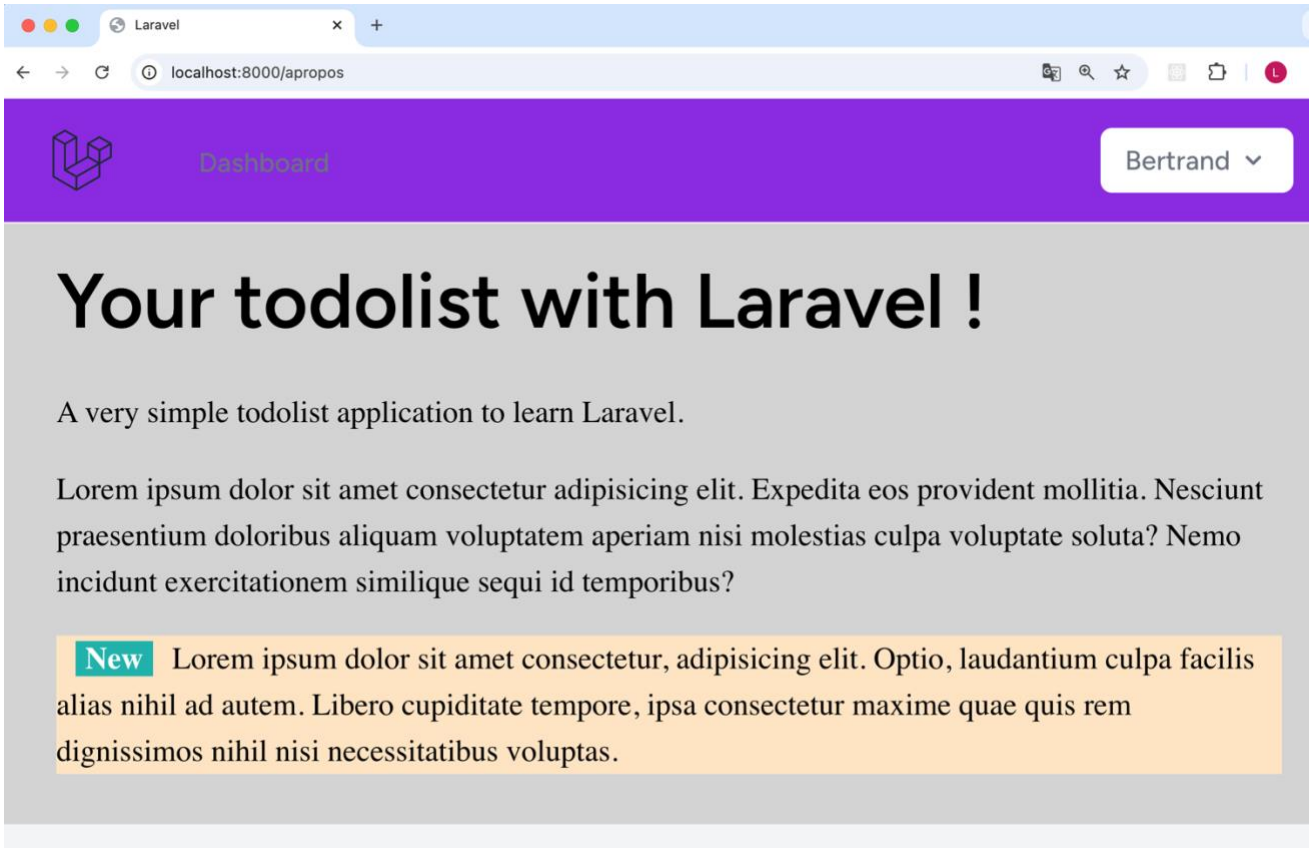
- ***Version <h1> dans /views/apropos.blade.php »***
  - Suivre les étapes et tester le résultat.
- ***Version <h1> dans /views/apropos/index.blade.php »***
  - Suivre les étapes et tester le résultat.
  - Mettre à jour le contrôleur.
- ***Version TailWind***
  - Suivre les étapes et tester le résultat.

    ⇒ Mettez-en commentaire la version <h1>

4 – CSS natif !

- Ce chapitre montre comment intégrer du code HTML et du CSS natif dans notre application Laravel.

On part de HTML de base



On peut aussi mettre du HTML de base :

```
<main class="matodolist">
    <h1>YYour todolist with Laravel !</h1>
    <p>A very simple todolist application to learn Laravel.</p>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Expedita eos
        provident mollitia. Nesciunt praesentium doloribus aliquam voluptatem
        aperiam nisi molestias culpa voluptate soluta?
        Nemo incidunt exercitationem similique sequi id temporibus?</p>
    <div class="new">
        <p><span>New</span>Lorem ipsum dolor sit amet consectetur, adipisicing elit.
            Optio, laudantium culpa facilis alias nihil ad autem.
            Libero cupiditate tempore, ipsa consectetur maxime quae quis
            rem dignissimos nihil nisi necessitatibus voluptas.</p>
    </div>
```

```
</main>
```

On gére du CSS spécifique : custom !

Une autre difficulté est que Tailwind annule le CSS de base du HTML : il faudra tout recoder (on peut demander à chatGPT à chaque fois).

On crée un fichier /ressources/css/custom.css
Dedans :

```css
.custom_menu{
    background-color: blueviolet;
}
main.matodolist{
    background-color: lightgray;
    padding: 10px 30px 10px 30px;
}
main.matodolist h1 {
    font-size: 2.5rem;  /* Exemple de taille de police */
    font-weight: bold;  /* S'assurer que le texte est gras */
}

main.matodolist p {
    margin-top: 1em;
    margin-bottom: 1em;
    font-family: serif; /* par défaut dans de nombreux navigateurs */
    line-height: 1.5;
}

main.matodolist .new {
    background-color: bisque;
}

main.matodolist span  {
    background-color: lightseagreen;
    display: inline;
    padding: 0px 5px 0px 5px;
    margin: 0px 10px 0px 10px;
    font-weight: bold;
    color : white
}
```

Dans app.css : on l'importe

```css
@import 'custom.css';
```

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

On gére le CSS spécifique du Breeze

Dans /ressources/views/layout/navigation.blade.php
On met tout le code dans

```
<div class="custom_menu">


</div>
```

Et on modifie la ligne 2 :

```
class="bg-white
```

on l'annulera :

```
class="bg--white
```

Il faut quand même exécuter Vite (compiler le code) et le laisser tourner (pas pratique !) :
npm run dev

5 - Petits paramétrages

Gestion de quelques petits paramétrages :

- 5.1 : changer le logo de l'application et le nom « dashboard »
- 5.2 : protéger un accès non connecté à une route

5.1 : Changer le logo de l'application et changer le nom « dashboard »

Dans le fichier /ressources/views/layouts/navigation.blade.php

```
div class="flex">
    <!-- Logo  -->
    <div class="shrink-0 flex items-center">
        <a href="{{ route('dashboard') }}">
            <!--
            <x-application-logo class="block h-9 w-auto fill-current text-gray-800" />
            {{ config('app.name', 'MonApplication') }}
            -->
            <img src="{{ asset('images/logo.png') }}" alt="Mon Logo" class="h-10 w-auto">
        </a>
    </div>

    <!-- Navigation Links -->
    <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
        <x-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">
            <span class="font-extrabold">
            <!-- {{ __('Dashboard') }} -->
            {{ __('Ma Todos liste') }}
            </span>
        </x-nav-link>
    </div>
</div>
```

On met une image « logo.png » dans /public/images

5.2 : Problème si on accède à la route apropos sans être connecté

- Si on accède à la route apropos sans être connecté, ça plante : en effet, le code veut accéder à des infos d'utilisateur connecté.

  ⇒ Pour éviter que ça plante, on ajoute une flèche sur la route :

```
Route::get(
    '/apropos',
    [App\Http\Controllers\AproposController::class, 'index']
)->name('apropos')->middleware('auth');
// avec le middleware, on force la connexion avant d'arriver sur la route apropos
```

Une autre solution consiste à choisir la vue dans le contrôleur :

```
public function index()
{
    if (!Auth::check()) {
        return redirect('/')->with('error', 'Vous devez être connecté pour accéder à cette page.');
    }

    return view('apropos.index');
}
```

6 – Mettre à jour la page d'accueil

6.1 : changer la vue : Welcome.blade.php.
6.2 : garantir que les routes fonctionnent on n'est pas logué.

Changer la vue : Welcome.blade.php

- On va mettre le contenu de apropos dans la page d'accueil.
- On change le Welcome : on part de layouts / app.blade.php
- On laisse le début du body de l'ancien Welcome : le route has login

  ⇒ Mais on supprime le cas si Auth
- Pour finir, on met le main de l'apropos

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <meta name="csrf-token" content="{{ csrf_token() }}">

        <title>{{ config('app.name', 'Laravel') }}</title>

        <!-- Fonts -->
        <link rel="preconnect" href="https://fonts.bunny.net">
        <link href="https://fonts.bunny.net/css?family=figtree:400,500,600&display=swap" rel="stylesheet" />

        <!-- Scripts -->
        @vite(['resources/css/app.css', 'resources/js/app.js'])
    </head>
    <body class="antialiased">
        <div class="relative sm:flex sm:justify-center sm:items-center min-h-screen bg-dots-darker bg-center bg-gray-100
dark:bg-dots-lighter dark:bg-gray-900 selection:bg-red-500 selection:text-white">
            @if (Route::has('login'))
                <div class="sm:fixed sm:top-0 sm:right-0 p-6 text-right z-10">
                    @auth
                    @else
                        <a href="{{ route('login') }}" class="font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-400
dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Log in</a>

                        @if (Route::has('register'))
```

```
                    <a href="{{ route('register') }}" class="ml-4 font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-
400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">Register</a>
                @endif
            @endauth
        </div>
    @endif

    <main class="container mx-auto p-4">
        <div class="bg-gray-200 py-12">
            <div class="container mx-auto text-center">
                <h1 class="text-5xl font-extrabold text-gray-800">Your todolist with Laravel!</h1>
                <p class="mt-4 text-lg text-gray-600">A very simple todolist application to learn Laravel.</p>
                <p class="mt-4 text-gray-700">
                    Lorem ipsum dolor sit amet consectetur adipisicing elit.
                    Expedita eos provident mollitia.
                    Nesciunt praesentium doloribus aliquam voluptatem aperiam nisi molestias culpa voluptate soluta?
                    Nemo incidunt exercitationem similique sequi id temporibus?
                </p>
                <p class="mt-4 bg-yellow-300 p-4 text-gray-800">
                    <span class="bg-red-500 text-white px-2 py-1 rounded-full mx-2">New</span>
                    Lorem ipsum dolor sit amet consectetur, adipisicing elit. Optio, laudantium culpa facilis alias nihil ad
autem.
                    Libero cupiditate tempore, ipsa consectetur maxime quae quis rem dignissimos nihil nisi necessitatibus
voluptas.
                </p>
            </div>
        </div>
    </main>

</div>
</body>
</html>
```

Petites redirections

- Le logo redirige sur apropos : c'est dans layout / navigation.blade.php
- L'index du contrôleur d'apropos redirige si on n'est pas authentifié

```php
use Illuminate\Support\Facades\Auth;


class AproposController extends Controller
{
    //
    public function index()
    {
        if (!Auth::check()) {
            return redirect('/');
        }
        else{
            return view('apropos/index'); # fichier apropos.blade.php
        }
    }
}
```

7 - Synthèse

Synthèse

- On a créé (étape 2) une page en installant un package : Breeze. Ça installe plusieurs routes.
- On a créé (étape 3) une page « à la main » avec une seule route et on a travailler la circulation des routes.

Commandes et fichiers modifiés

Commandes

- Vider les caches

```
php artisan config:cache     # par prudence, on vide les caches
php artisan cache:clear      # par prudence, on vide les caches
```

- Migration

```
php artisan migrate
```

- npm

```
npm install
npm run dev
```

- Lister les routes

```
>php artisan route:list
```

- Création d'un contrôleur pour la route apropos

```
> php artisan make:controller AproposController
```

- Syntaxe Blade

```
     Le PHP est entre {{    }}
     Le PHP est précédé de @
```

Fichier manipulés

- / routes / web.php
- / App / Http / Controllers / AproposController.php
- / ressources / views / apropos / index.blade.php
- / ressources / views / layouts / app.blade.php
- / ressources / views / layout / navigation.blade.php
- / ressources / css / app.css
- /ressources / css / custom.css

8 – TP complet

- Faire les TP de 1, 2, 3, 5 et 6
- 1, 2, 3 : une page apropos avec TailWind
- 5 et 6 :

  ⇒ Changer le logo et contrôler l'apropos non logué.

  ⇒ Changer l'accueil pour qu'il corresponde a l'apropos. Faire que le logo ramène sur l'apropos et le nom de l'application sur le dashboard.
- 4 : tester du HTML et CSS natif.