LARAVEL (8-2021) – FRAMEWORK PHP 5

SOMMAIRE

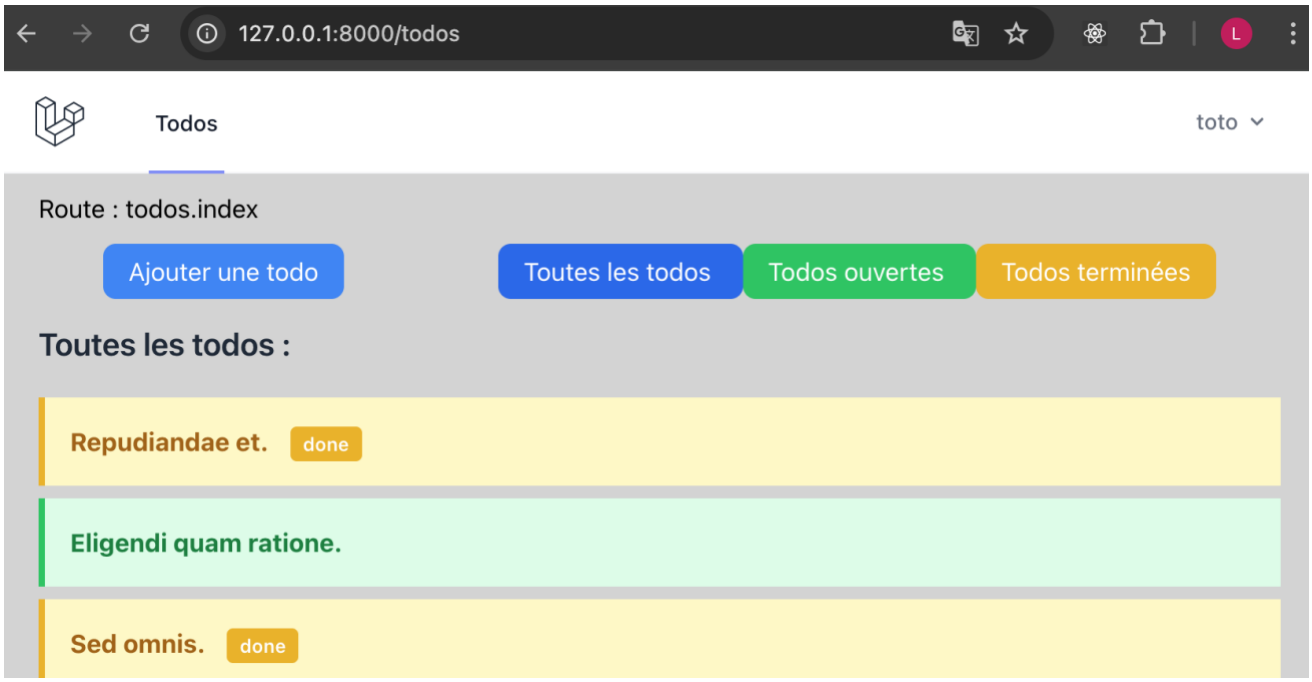
ETAPE 5

1 –Boutons de gestion des todos v12-13

Menu principal : gestions des todos

Objectif

- Une barre à 4 boutons : ajouter, toutes, ouvertes, terminés
- Des todos colorées selon l'état
- Un menu général limité à ce qui nous intéresse.
- Un débug : l'affichage du nom de la route
- Le titre de la liste : d'abord toutes les todos (auxquelles amène le menu Todos).




127.0.0.1:8000/todos/undone
Todos
toto
Route : todos.undone
Ajouter une todo
Toutes les todos
Todos ouvertes
Todos terminées
Todos ouvertes :
Eligendi quam ratione.
Possimus explicabo est.


127.0.0.1:8000/todos/done
Todos
toto
Route : todos.done
Ajouter une todo
Toutes les todos
Todos ouvertes
Todos terminées
Todos terminées :
Repudiandae et.
done
Sed omnis.
done

Principes de résolution

- On gère les Todos : c'est du CRUD avec le contrôleur de ressources : TodoController
  ⇒ Mettre à jour les routes
  ⇒ Créer les contrôleurs (les méthodes dans le contrôleur de ressources)
  ⇒ Créer / mettre à jour les vues
    ⇒ Les vues appellent des routes dans les <a> et les <form>
- Simplifier le menu principal :
  ⇒ Le logos amène sur la route apropos
  ⇒ On n'a que le menu todos
  ⇒ Quand on arrive, on a l'équivalent de la page apropos.

Eléments techniques

- TailWind

  ⇒ On va modifier le fichier tailwind.config.js
- Blade : @if, @foreach
- SQL Laravel : https://laravel.com/docs/12.x/queries

Colorer les todos

En Jaune : les terminées
En Vert : les pas terminées

Div du foreach :

```
@foreach($datas as $data)
    <div
        class="{{
            $data->done ?
                'bg-yellow-100 border-l-4 border-yellow-500 text-yellow-700'
            :
                'bg-green-100 border-l-4 border-green-500 text-green-700'
        }}"
        role="alert"
    >
```

Menu principal : todos ouvertes (undone) et todos terminées (done)

Vue : views/todos/index.php

- On ajoute deux balises a pour les deux autres boutons :

```
<div>
  <a href={{ route('todos.create') }}>Ajouter</a>
  <a href={{ route('todos.index) }}>Toutes les todos</a>
  <a href={{ route('todos.undone) }}>Les todos terminées</a>
  <a href={{ route('todos.done) }}>Les todos ouvertes</a>
</div>
```

⇒ Ca plante ! Les routes « todos.undone » et « todos.done » n'existent pas.

Création des routes : avec un group fonction

- On crée toutes les routes du TodoController dans un group fonction

```
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');
});

Route::controller(TodoController::class)->group(function () {
  Route::get('/todos', 'index')->name('todos.index');
  Route::get('/todos/done', 'done')->name('todos.done');
  Route::get('/todos/undone', 'undone')->name('todos.undone');
});
```

⇒ Cette syntaxe est simple et permet des mises à jour aisées et lisibles des contrôleurs de ressources.

⇒ On renomme les routes URL (physique) en routes logiques : ->name()

⇒ https://laravel.com/docs/12.x/routing#generating-urls-to-named-routes

Contrôleurs : on crée des contrôleurs vides

- On va ajouter les méthodes done() et undone() dans le TodoController

  ⇒ Fichier : app/http/Controllers/TodoController.php

```
/**
 * Sélectionne les undone.
 */
public function undone()
{
    $datas = Todo::where('done', 0)->paginate(10);
    return view('todos.index', compact('datas'));
}

/**
 * Sélectionne les done.
 */
public function done()
{
    $datas = Todo::where('done', 1)->paginate(10);
    return view('todos.index', compact('datas'));
}
```

⇒ Ca marche !

Version TailWind des boutons

```
@php
/* une fonction pour calculer une classe de coloration */
if (!function_exists('getClassBouton')) { // on vérifie qu'elle n'a pas déjà été chargée
    function getClassBouton($color){
        return "bg-".$color."-500 text-white px-4 py-2 rounded-lg hover:bg-".$color."-600";
    }
}
@endphp

<div class="flex justify-around space-around space-x-4 mt-4">
    <div>
        <a href="{{ route('todos.create') }}" class="{{ getClassBouton('blue') }}">
            Ajouter une todo
        </a>
    </div>
    <div class="flex space-x-4" >
        <a href="{{ route('todos.index') }}" class="{{ getClassBouton('blue') }}">
            Toutes les todos
        </a>
        <a href="{{ route('todos.undone') }}" class="{{ getClassBouton('green') }}">

            Todos ouvertes
        </a>
        <a href="{{ route('todos.done') }}" class="{{ getClassBouton('yellow') }}">
            Todos terminées
        </a>
    </div>
</div>
```

- bg-blue-500, bg-yellow-500, bg-green-500 : Définit la couleur des boutons.
- text-white ou text-black : Définit la couleur du texte pour un bon contraste.
- px-4 py-2 : Ajoute du padding pour une meilleure apparence.
- rounded-lg : Ajoute des coins arrondis.
- hover:bg-blue-600, hover:bg-yellow-600, hover:bg-green-600 : Change la couleur au survol.
- flex space-x-4 : Affiche les boutons en ligne avec un espace entre eux.
- On utilise une fonction pour générer la class de coloration

Améliorer la présentation

➢ *Dans index.blade.php*

- On affiche :Toutes les todos, Todos ouvertes ou Todos terminées

```
<h3 class="text-xl font-semibold text-gray-800">
    @if (Route::currentRouteName() == 'todos.index')
        Toutes les todos :
    @elseif (Route::currentRouteName() == 'todos.undone')
        Todos ouvertes :
    @elseif (Route::currentRouteName() == 'todos.done')
        Todos terminées :
    @endif
</h3>
```

⇒ Le h3 n'est plus pris en compte avec TailWind : il faut créer la classe correspondante.

➢ *Dans app.blade.php (le layout principal)*

- On va afficher la route avant la liste des todos avec la méthode : Route::currentRouteName()

```
        <main>
            <h2> Route : {{ Route::currentRouteName() }}</h2>
            {{ $slot }}
        </main>
```

Subtilités de TailWind : tailwind.config.js

Certaines définies dynamiquement class peuvent ne pas être prises en compte.

Pour régler le problème :

1) Videz les caches :

```
php artisan optimize:clear
php artisan config:cache
```

2) Modifiez le fichier tailwind.config.js

Ajoutez :

```
/*
   On ajoute ça pour pouvoir avoir du CSS dynamique avec TailWind
   On précise les fichiers concernés
   On précise les class HTML concernées
*/
module.exports = {
   content: [
     './resources/views/**/*.blade.php',
     './resources/js/**/*.vue',
   ],
   safelist: [
     'bg-blue-500', 'bg-blue-600',
     'bg-green-500', 'bg-green-600',
     'bg-yellow-500', 'bg-yellow-600'
   ],
 }
```

2 - Ajout de la todo : todos/create

Ajouter une todo

Objectifs :

- On va faire fonctionner le bouton : « ajouter ».

Situation de départ : route todos/create

- On a déjà une route http://127.0.0.1:8000/todos/create qui a été créée par défaut avec php artisan make:controller.
- On a aussi une méthode « create() » dans la classe App\Http\Controllers\TodoController.

  ⇒ Cette méthode a été créée avec php artisan make:controller

  ⇒ Elle ne fait rien. On va la remplir.
- On peut visualiser cette route :

```
php artisan route:list
```

⇒ On a une route todos/create

⇒ Qui correspond à une route logique todos.create

⇒ Qui appelle la méthode create() de la classe TodoController

```
|Method    |URI           | nom logique   | Action
-------------------------------------------------------------------
|GET|HEAD. |todos/create   | todos.create  | TodoController@create
```

Controller : fonction create()

- Dans le TodoControler, la fonction create ne fait rien.
- Cette fonction a pour but de permettre la saisie d'une nouvelle todo et son insertion en BD.
- L'insertion en BD sera gérée par la fonction store() qui est déjà prévue dans le contrôleur de ressources de Todos.
- La fonction create() ne fait donc pas de calcul. Elle ne fait qu'appeler la vue : todos.create

```
public function create()
{
    return view('todos.create');
}
```

⇒ Quand on ajoute cette ligne : ça plante ! On ne trouve pas la vue totos.create.

⇒ Il va falloir créer cette vue.

Vue todos/create.blade.php

- Il faut créer une vue associée avec un formulaire :

  ⇒ create.blade.php dans views/todos

- Ce formulaire est un simple code HTML avec 2 spécificités :

  ⇒ L'action : {{ route('todos.store') }}

  ⇒ La route store a été créé par le php artisan make:controller

  ⇒ @csrf : cette directive permet de sécuriser la saisie.

  ⇒ https://laravel.com/docs/8.x/csrf

```
<x-app-layout>
<div class="max-w-lg mx-auto mt-10 bg-white shadow-md rounded-lg p-6">
   <h2 class="text-2xl font-bold mb-4 text-center">Ajouter une tâche</h2>

   <form action="{{ route('todos.store') }}" method="POST" class="space-y-4">
      @csrf
      <!-- Champ titre -->
      <div>
         <label for="name" class="block text-gray-700 font-semibold">Titre</label>
         <input type="text" name="name" id="name"
                class="w-full p-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"
                placeholder="Entrez le titre"
                required>
      </div>
      <!-- Champ description -->
      <div>
         <label for="description" class="block text-gray-700 font-semibold">Description</label>
         <input type="text" name="description" id="description"
                class="w-full p-2 border border-gray-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500"
                placeholder="Entrez une description"
                required>
      </div>
      <!-- Bouton de soumission -->
      <div class="text-center">
         <button type="submit"
                 class="bg-blue-500 text-white px-4 py-2 rounded-lg hover:bg-blue-600 transition">
            Ajouter
         </button>
      </div>
   </form>
</div>
```

```
</x-app-layout>
```

⇒ A ce stade, ça plante : la route n'est pas définie.

⇒ On va l'ajouter

Route todos.store : POST

- Il faut ajouter une route pour le POST du formulaire

  ⇒ C'est la route /todos/store dont le contrôleur est prédéfini.

- On va le mettre dans le goupe de route du TodoController
- Et on va aussi mettre la route create, ce qui permettra de voir toutes les routes du TodoController et de les nommer :

```
Route::controller(TodoController::class)->group(function () {
  Route::get('/todos', 'index')->name('todos.index');
  Route::get('/todos/done', 'done')->name('todos.done');
  Route::get('/todos/undone', 'undone')->name('todos.undone');
  Route::get('/todos/create', 'create')->name('todos.create');
  Route::post('/todos/store', 'store')->name('todos.store');
);
```

⇒ A ce stade ça fonctionne, mais la méthode store() ne fait rien.

L'enregistrement en BD : la méthode store

- On va coder la méthode store.

Principe de résolution

- On va instancier une Todo du modèle et la remplir avec le $request qui contient les données du formulaire.
- Il suffit ensuite de faire un todo->save() pour avoir une sauvegarde en BD. La méthode save() vient du Model.
- Puis de revenir à la page des todos : la route todos.index. La méthode redirect() est une fonction globale de Laravel : https://laravel.com/docs/9.x/helpers#method-redirect

Code

```
$todo = new Todo();
$todo->name = $request->name; // le request est en paramètre
$todo->description = $request->description; // le request est en
paramètre
$todo->done=0;
$todo->save(); // sauvegarde en BD
return redirect()->route('todos.index');
```

⇒ Ca marche !

Variante : on donne la valeur par défaut à l'attribut done avec une migration

Objectifs :

- On modifie la BD pour que l'attribut ait 0 en valeur par défaut.
- On va faire ça avec une migration.

Création de la migration

- Commande php artisan make:migration
- On précise la table concernée avec –table todos : ca génère un code automatique plus adapté.
- On donne un nom explicite : update_done_field_with_default_value_to_todos_table

```
php artisan make:migration
update_done_field_with_default_value_to_todos_table --table todos
```

Résultats de la création de la migration :

➢ ***On vérifie le statut des migrations :***

```
php artisan migrate:status
```

- La migration est là. Elle n'a pas été exécutée.

➢ ***On regarde le fichier de migration créé :***

- Database/migrations/… update_done_field_with_default_value_to_todos_table.php
- Les function up et down font appel à Shema ::table('todos',) qui reste à écrire

➢ ***On vérifie la situation en BD***

```
Mysql > desc todos ;

+-----------+-------------------+----+---+-------+--------------+
|Field      |Type               |Null|Key|Default|Extra         |
+-----------+-------------------+----+---+-------+--------------+
|done       |tinyint(1)         | NO |   | NULL  |              |
+-----------+-------------------+----+---+-------+--------------+
```

Code de la modification de l'attribut « done »

- On code up() :

```
    $table->boolean('done')->default(0)->change();
    $table->boolean('done')->nullable(true)->change();
```

- Dans down : on fait les opérations inverses :

```
    $table->boolean('done')->default(NULL)->change();
    $table->boolean('done')->nullable(false)->change();
```

- Puis on migre :

```
    php artisan migrate
```

⇒ Ca bogue.

⇒ Il faut installer installer doctrine/dbal

```
>composer require doctrine/dbal
```

⇒ https://laravel.com/docs/8.x/migrations#modifying-columns

```
    php artisan migrate
```

⇒ Ca marche !!!

3 – Un peu de SQL Laravel

Principes

- La documentation est là : https://laravel.com/docs/12.x/queries

  ⇒ On y trouve la documentation pour faire du SQL avec Laravel

- Par exemple :

  ⇒ La clause WHERE : https://laravel.com/docs/12.x/queries#basic-where-clauses

Application

Vue : afficher le nombre de todos

- Avant la pagination, on va afficher en plus le nombre total de todos.
- Après le endforeach, avant le link de la pagination, on ajoute :

```
    @endforeach
    <p>Nombre total : {{ $countTodos }}</p>
    {{ $datas->links()}}
```

⇒ C'est du HTML classique : on affiche une variable, ici avec Blade.

⇒ Il faut calculer $countTodos.

Controller : compter le nombre de todos

- On va compter le nombre de todos dans les fonctions index(), done() et undone() du contrôleur de ressources et retourner la valeur.
- Pour la fonction index :

```
public function index() {
   $countTodos = Todo::all()->count();
   return view(
      'todos.index',
      compact('datas'),
      compact('countTodos')
);
```

- Pour la fonction done :

```
$countTodos = Todo::where('done', 1)->count();
```

- Pour la fonction undone :

```
$countTodos = Todo::where('done', 0)->count();
```

Syntaxe SQL en Laravel :

2 usage du SQL en Laravel :

- ***A partir de la méthode DB::table()***
  - where() : https://laravel.com/docs/12.x/queries#basic-where-clauses
  - count() : https://laravel.com/docs/12.x/queries#aggregates
  - Pour utiliser la classe DB : il faut l'inclure :

```
use Illuminate\Support\Facades\DB;
```

⇒ Il faut regarder la documentation et les exemples pour en savoir plus.

- ***A partir des classes du modèles :***
  - all(), where(), orderBy() : https://laravel.com/docs/12.x/eloquent#retrieving-models

⇒ Il faut regarder la documentation et les exemples pour en savoir plus.

4 – Finalisation du menu

Objectifs

- Avoir une présentation plus aboutie.
- Faire du ménage dans le code.

Avoir une présentation plus aboutie

Une arrivée sur la page « apropos » avec breeze

On met duplique welcome, on l'appelle apropos
A la place de la div qui est sous le if route, on met :

```
<section class="bg-gray-200 p-8 rounded-lg shadow-md">
    <h1 class="text-4xl font-bold text-center text-gray-800">
        Your todolist with Laravel!
    </h1>
    <p class="mt-4 text-lg text-gray-700 text-center">
        A very simple todolist application to learn Laravel.
    </p>
    <p class="mt-4 text-gray-700 leading-relaxed">
        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="mt-6 p-4 bg-orange-200 rounded-lg shadow rounded border-2 border-red-500">
        <p>
            <span class="bg-teal-500 text-red-500 font-bold px-2 py-1">
                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>
</section>
```

Dans le head, pour charger TailWind, on met, à la fin :

```
<!-- Scripts -->
```

```
    @vite(['resources/css/app.css', 'resources/js/app.js'])
```

Le menu de navigation : dans navigation.blade.php

On veut un menu todos : on va dans la zone « navigation links » : on veut accéder à todos.index

Un logo qui amène sur le « apropos » : on va dans la zone « Logo » : on met la route apropos

Arrivée directe dans la page apropos avec Breeze quand on se loggue :

dans app/Providers/RouteServiceProvider :

```
public const HOME = '/apropos';
```

5 : Boutons d'action sur chaque todo v14

- La suite consistera à ajouter 3 boutons d'action sur chaque todo : done/undone, supprimer, editer

  ⇒ Set done : jaune, se transforme en set undone : vert

  ⇒ effacer : rouge

  ⇒ éditer: bleu

Ajout des boutons

Div du foreach :

```
@foreach($datas as $data)
    <div
        class="flex items-center justify-between p-4 mb-2 {{
```

- flex Active Flexbox pour aligner les éléments à l'intérieur horizontalement.
- items-center Aligne verticalement les éléments au centre.
- justify-between Répartit l'espace entre les éléments : le texte est à gauche et les boutons à droite.
- p-4 Ajoute du padding de 1rem (16px) à l'intérieur du div.
- mb-2 Ajoute une marge en bas de 0.5rem (8px) pour espacer les lignes entre elles.

Div des boutons

```
<div class="flex space-x-2">
    @if($data->done)
        <form action="{{ route('todos.makeundone', $data) }}" method="POST">
            @csrf
            @method("PUT")
            <button
                type="submit"
                class="bg-green-500 text-white px-4 py-2 rounded-md hover:bg-green-600">
                set undone
            </button>
        </form>
    @else
        <form action="{{ route('todos.makedone', $data) }}" method="POST">
            @csrf
            @method("PUT")
            <button
                type="submit"
                class="bg-yellow-500 text-white px-4 py-2 rounded-md hover:bg-yellow-600">
                set done
            </button>
        </form>
    @endif
    <form action="#" method="POST">
        <button
            type="submit"
            class="bg-blue-500 text-white px-4 py-2 rounded-md hover:bg-blue-600">
            Editer
        </button>
    </form>
    <form action="#" method="POST">
        <button
            type="submit"
            class="bg-red-500 text-white px-4 py-2 rounded-md hover:bg-red-600">
            Effacer
        </button>
    </form>
</div>
```

- flex : Active le mode flexbox, qui permet d'aligner les éléments enfants horizontalement.

- space-x-2 : Ajoute un espacement horizontal (margin-right) de 0.5rem (8px) entre chaque élément enfant du div.
- Ca buggue : les route n’existent pas
- On va expliquer les routes après.

6 - Actions done et undone – routes et controleur

Présentation : l'architecture est MVC

- L'objectif est simple : passer chaque todo de « done » à « undone » et réciproquement.
- Quand on clique sur « set done » :
    1. On appelle une route : todos/makedone/{idTodo} qui est associé à un contrôleur.
        ⇒ On va donc d'abord définir la route avec son contrôleur associé
        ⇒ On va définir le controleur qui met à jour la BD.
        ⇒ On va mettre à jour la vue.
- Idem quand on clique sur « set undone »

routes

- Routes :

⇒ On crée les routes todos/makedone{todo}, todos/makeundone{todo}
⇒ Ce sont des routes « put » : on va modifier le contenu de la BD.
⇒ Les routes appellent des méthode du contrôleur

Dans le group(fonction), on ajoute :

```
Route::put('/todos/makedone{todo}', 'makedone')->name('todos.makedone');
Route::put('/todos/makedone/{todo}', 'makeundone')->name('todos.makeundone');
```

⇒ On aura par exemple, la route : http://localhost:8000/todos/makedone/1

Vue appelante

HTML

- On met à jour la vue appelante pour pouvoir accéder à la route : on l'a déjà fait.

```
<form action="{{ route('todos.makedone', $data) }}" method="POST">
    @csrf
    @method("PUT")
    <button
        type="submit"
        class="bg-yellow-500 text-white px-4 py-2 rounded-md hover:bg-yellow-600">
        set done
    </button>
</form>
```

- A noter :

  ⇒ $data c'est l'id de la todo. C'est un paramètre de la méthode route().

  ⇒ @crsf : pour gérer la sécurité

  ⇒ @method("PUT") : nécessaire pour une bonne prise en compte des mises à jour de la BD.

Contrôleur

- On vient de définir deux fonctions pour le contrôleur : makedone() et makeundone()

  ⇒ Ce sont des fonctions pour App.Http/Controllers/todoControler.php

  ⇒ Elles ont un id de todo en paramètre

- Ces fonctions sont simples : elles mettent à jour l'attribut « done ».
- Il suffit de faire appel à la méthode « update() » mettre à jour « todo » dans la BD.
- Il faudra aussi récupérer l'objet à partir de l'id de Todo

```php
/**
 * Passe une vue à undone
 */
public function makeundone($idTodo)
{
    //dd($idTodo);
    $todo = Todo::findOrFail($idTodo); // Récupère l'objet Todo à partir de l'ID
    $todo->done = 0;
    $todo->update();
    return back();  // pour revenir à la page précédente
                    // autrement dit pour rester sur la page
}
```

- A noter :

  ⇒ return back(); // pour revenir à la page précédente

Tests

- On peut tester l'application : ça donne de bons résultats.
- On peut aussi voir les données dans la BD.

7 – Bilan : CRUD quasi complet

A ce stade, on sait gérer un CRUD quasi complet : CR et U partiel.
C'est assez simple.
On finit en retirant tout le code inutile : on constate qu'on a un code assez simple.