Tutoriel Laravel 8 : opérations CRUD

Tout d'abord, CRUD de l'anglais (Create, Read, Update, Delete) qui sont les 4 opérations principales d'une base de données (Insérer, Lire, Mettre à jour et Supprimer).

Pour créer une application CRUD dans Laravel 8, votre machine doit avoir une version plus récente de **PHP**, la version PHP >= 7.3 et **composer** avec les extensions supplémentaires suivantes.

Extension PHP **BCMath**

Extension PHP **Ctype**

Extension PHP **OpenSSL**

Extension PHP **PDO**

Extension PHP de **Tokenizer**

Extension PHP **XML**

Extension PHP **Fileinfo**

Extension PHP **JSON**

Extension PHP **Mbstring**

Étape 1: Installation de Laravel 8

Vous pouvez installer Laravel 8 avec **Laravel Valet**. Pour en savoir plus, consultez le guide de mise à niveau de Laravel Valet .

Vous pouvez également installer Laravel 8 à l'aide de la commande suivante.

```
composer create-project laravel/laravel --prefer-dist laravel8cars
```

Étape 2: configurer la base de données MySQL

Nous utiliserons une base de données **MySQL** pour créer une base de données et revenir au projet. utiliser Wampserver, Mamp ou tout autre server Local pour cette operation.

Laravel fournit le fichier **.env** pour ajouter des informations d'identification. Ouvrez le fichier et modifiez le code suivant.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel8cars
DB_USERNAME=root
DB_PASSWORD=root
```

Le nom d'utilisateur et le mot de passe seront différents pour les vôtres en fonction des informations d'identification de votre base de données.

Laravel est livré avec certaines migrations par défaut comme **users**, **password_resets** et la table **create_failed_jobs**. Accédez maintenant au terminal et tapez la commande suivante pour exécuter vos migrations.

```
php artisan migrate
```

Vous pouvez voir dans votre base de données que ces tables sont créées.

Étape 3: créer un modèle et une migration personnalisée

Nous allons créer un projet autour des **marques des voitures**. Ainsi, les utilisateurs peuvent créer des voitures, éditer et les supprimer. Alors, créons un modèle de voiture et une migration.

```
php artisan make:model Car -m
```

Cela créera deux fichiers :

- Modèle **Car.php**
- migration **create_cars_table.php**

Ajoutez les nouveaux champs dans le fichier de migration ***create_cars_table.php***.

```
// -> create_cars_table.php

public function up()

{
    Schema::create('cars', function (Blueprint $table) {
        $table->id();
        $table->string('marque');
        $table->integer('prix');
        $table->timestamps();
    });
}
```

Les champs ***id*** et ***timestamps*** sont créés par défaut par Laravel. La ***marque*** et le ***prix*** sont nos champs personnalisés que l'utilisateur peut ajouter via les formulaires Web.

Vous pouvez exécuter la migration pour créer la table dans la base de données.

```
php artisan migrate
```

Étape 4: Créez un contrôleur Laravel 8.

Le routage des ressources Laravel attribue les routes « **CRUD** » typiques à un contrôleur avec une seule ligne de code. Puisque notre application a des opérations crud de base, nous utiliserons le contrôleur de ressources pour ce petit projet.

```
php artisan make:controller CarController --resource
```

Ce que vous pouvez faire ici, c'est ouvrir le fichier ***App\Providers\RouteServiceProvider.php*** et ajouter le code suivant dans la méthode boot().

```
// RouterServiceProvider.php

Route::middleware('web')
            ->namespace('App\Http\Controllers')
            ->group(base_path('routes/web.php'));
```

Voilà !. Maintenant, il peut trouver le contrôleur. Si vos fichiers de contrôleur sont ailleurs, vous devez affecter le chemin dans l'espace de noms.

Notez ici que j'ai ajouté le flag ***–resource***, qui définira six méthodes à l'intérieur du **CarController**, à savoir:

Index: (La méthode index() est utilisée pour afficher une liste de voitures).

Create: (La méthode create() affichera le formulaire ou la vue pour créer une voiture).

Store: (La méthode store() est utilisée pour insérer une voiture dans la base de données. Remarque: la méthode create submit les données du formulaire à la méthode store()).

Show: (La méthode show() affichera une voiture spécifié).

Edit: (La méthode edit() affichera le formulaire pour éditer une voiture. Le formulaire sera rempli avec les données de la voiture existante).

Update: (La méthode update() est utilisée pour mettre à jour une voiture dans la base de données. Remarque: la méthode edit() submit les données du formulaire à la méthode update()).

Destroy: (La méthode destroy() est utilisée pour supprimer la voiture specifique spécifié).

Par défaut, le fichier **CarController.php** est créé dans le dossier des contrôleurs **app > Http >**

```php
<?php

// CarController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;


class CarsController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */

    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        //
    }


    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
```

```
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
}
```

Vous pouvez voir que le fichier contient des opérations CRUD sous la forme de différentes fonctions. Nous utiliserons ces fonctions, une par une, pour créer des opérations specifiques.

Le flag ***–resource*** appellera la méthode internal *resource()* par Laravel pour générer les routes suivantes.

Vous pouvez consulter la liste des routes à l'aide de la commande suivante.

```
php artisan route:list
```

Étape 5: définir les routes

Pour définir une route dans Laravel, vous devez ajouter le code de la route dans le fichier **routes > web.php .**

```
// web.php

Route::resource('cars', 'CarController');
```

Étape 6: Configurer Bootstrap dans Laravel 8

Laravel fournit le framework **Bootstrap** et **Vue** qui se trouve dans le package **laravel/ui**, qui peut être installé à l'aide de Composer.

```
composer require laravel/ui
```

Une fois le package *laravel//ui* installé, vous pouvez installer Bootstrap à l'aide de la commande Artisan suivante.

```
php artisan ui bootstrap
```

Maintenant, exécutez "**npm install**" puis "**npm run dev**" pour compiler votre nouvelle structure.

Si vous n'avez pas installer NPM, installez nodejs

Étape 7: créer des vues dans Laravel 8

Les vues contiennent le code HTML servi par votre application et séparent la logique de votre contrôleur/application de votre logique de présentation. Les vues sont stockées dans le répertoire *resources/views*.

Dans le répertoire views, nous devons également créer un fichier de mise en page. Nous allons donc créer le fichier dans le répertoire views appelé **layout.blade.php**. Ajoutez le code suivant dans le fichier *layout*.

```
<!-- layout.blade.php -->

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Tutoriel Laravel 8 CRUD</title>
  <link href="{{ asset('css/app.css') }}" rel="stylesheet" type="text/css" />
</head>

<body>
  <div class="container">
    @yield('content')
  </div>
  <script src="{{ asset('js/app.js') }}" type="text/js"></script>
</body>
</html>
```

Dans le dossier ***resources/views***, créez les trois fichiers suivants.

index.blade.php

create.blade.php

edit.blade.php

Dans le fichier **create.blade.php**, écrivez le code suivant.

```
<!-- create.blade.php -->

@extends('layout')

@section('content')
<style>
  .uper {
    margin-top: 40px;
  }
</style>

<div class="card uper">
  <div class="card-header">
    Ajouter une Voiture
  </div>

  <div class="card-body">
    @if ($errors->any())
      <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
        </ul>
      </div><br />
    @endif

      <form method="post" action="{{ route('cars.store') }}">
.         @csrf
          <div class="form-group">
              <label for="marque">Marque de Voiture:</label>
              <input type="text" class="form-control" name="marque"/>
          </div>

          <div class="form-group">
              <label for="prix">Prix :</label>
              <input type="text" class="form-control" name="prix"/>
          </div>
          <button type="submit" class="btn btn-primary">Ajouter</button>
      </form>
  </div>
</div>
@endsection
```

Dans ce code, nous avons défini l'action qui appellera la méthode **store()** du controller **CarController**.

N'oubliez pas que nous avons utilisé le contrôleur de ressources.

Maintenant, nous devons retourner cette vue de création à partir de la méthode **create()** de CarController.

Écrivez donc le code suivant dans la méthode create().

```
// CarController.php


public function create()
{
   return view('create')
}
```

Accédez à **http://localhost:8000/cars/create**, et vous verrez quelque chose comme ci-dessous.

Étape 8: ajoutez des règles de validation et stockez les données.

Dans cette étape, nous ajouterons une validation de formulaire Laravel.

Maintenant, importer dans le fichier **CarController.php** l'espace de noms du modèle Car.php.

```php
<?php

// CarController.php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Car;
```

Maintenant, écrivez le code suivant à l'intérieur du CarController.php dans la fonction **store().**

```php
// CarController.php

public function store(Request $request)
{

    $validatedData = $request->validate([
        'marque' => 'required|max:255',
        'prix' => 'required',
    ]);

    $car = Car::create($validatedData);

    return redirect('/cars')->with('success', 'Voiture créer avec succèss');
}
```

Nous utilisons la méthode **$request->validate()** pour la validation, qui reçoit un tableau de règles de validation.

Les règles de validation [] sont un tableau associatif. La clé sera le **field_name** et la valeur étant les règles de validation. Le deuxième paramètre est un tableau facultatif pour les messages de validation personnalisés. Les règles sont séparées par un signe "| ". Dans cet exemple, nous utilisons les règles de validation les plus élémentaires. Si la validation échoue, elle sera redirigée vers la page de formulaire avec des messages d'erreur. Si la validation réussit, il créera la nouvelle voiture et la sauvegardera dans la base de données.

En cas d'erreur, nous devons parcourir ces messages d'erreur dans le fichier **create.blade.php**, ce que nous avons déjà fait.

Si vous laissez tous les champs du formulaire vides, vous trouverez le message d'erreur comme sur cette image.

Comme nous pouvons voir que nous avons obtenu les erreurs, mais si nous remplissons toutes les données correctes, vous ne pourrez toujours pas enregistrer les données dans la base de données en raison d'une exception d'affectation en masse.

Pour éviter l'exception d'affectation en masse , nous devons ajouter un tableau **$fillable** dans le modèle **Car.php**.

```php
<?php

// Car.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Car extends Model
{
    use HasFactory;

    protected $fillable = ['marque', 'prix'];
}
```

Maintenant, si vous remplissez les champs de formulaire corrects, cela crée une nouvelle ligne dans la base de données.

Étape 9: Affichez les voitures

Pour afficher la liste des voitures, nous devons écrire le code HTML dans le fichier **index.blade.php**. Mais avant cela, écrivons la fonction *index()* du fichier CarController.php pour obtenir le tableau de données de la base de données.

```php
// CarController.php

public function index()
{
    $voitures = Car::all();

    return view('index', compact('voitures'));
}
```

Maintenant, écrivez le code suivant dans le fichier **index.blade.php**.

```
<!-- index.blade.php -->

@extends('layout')

@section('content')

<style>
  .uper {
    margin-top: 40px;
  }
</style>

<div class="uper">

  @if(session()->get('success'))
    <div class="alert alert-success">
      {{ session()->get('success') }}
    </div><br />
  @endif

  <table class="table table-striped">

    <thead>
        <tr>
```

```
        <td>ID</td>
        <td>Marque</td>
        <td>Prix</td>
        <td colspan="2">Action</td>
      </tr>
    </thead>

    <tbody>
      @foreach($voitures as $voiture)
      <tr>
        <td>{{$voiture->id}}</td>
        <td>{{$voiture->marque}}</td>
        <td>{{$voiture->prix}}</td>
        <td><a href="{{ route('cars.edit', $voiture->id)}}" class="btn btn-primary">Modifier</a></td>
        <td>
          <form action="{{ route('cars.destroy', $voiture->id)}}" method="post">
            @csrf
            @method('DELETE')
            <button class="btn btn-danger" type="submit">Supprimer</button>
          </form>
        </td>
      </tr>
      @endforeach
    </tbody>
  </table>
<div>
@endsection
```

Nous avons ajouté deux boutons nommés modifier et supprimer pour effectuer les opérations respectives.

Étape 10: Terminez la modification et la mise à jour

Pour pouvoir modifier les données, nous avons besoin des données de la base de données. Ajoutez le code suivant dans la fonction d'édition du fichier CarController.php.

```
// CarController.php

public function edit($id)

{
    $car = Car::findOrFail($id);

    return view('edit', compact('car'));
}
```

Maintenant, créez le nouveau fichier dans le dossier views appelé **edit.blade.php** et ajoutez le code suivant.

```
@extends('layout')

@section('content')

<style>
  .uper {
    margin-top: 40px;
  }
</style>

<div class="card uper">
  <div class="card-header">
    Modifier la voiture
  </div>
```

```
  <div class="card-body">

    @if ($errors->any())
      <div class="alert alert-danger">
        <ul>
          @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
          @endforeach
        </ul>
      </div><br />
    @endif

      <form method="post" action="{{ route('cars.update', $car->id ) }}">
        <div class="form-group">
          @csrf
          @method('PATCH')
          <label for="marque">Marque :</label>
          <input type="text" class="form-control" name="marque" value="{{ $car->marque }}"/>
        </div>

        <div class="form-group">
          <label for="cases">Prix :</label>
          <input type="text" class="form-control" name="prix" value="{{ $car->prix }}"/>
        </div>
        <button type="submit" class="btn btn-primary">Modifier</button>
      </form>
  </div>
</div>
@endsection
```

Maintenant, allez à la page d'index, puis allez à la page d'édition d'une voiture spécifique, et vous verrez le formulaire avec les valeurs remplies.

Maintenant, ajoutez le code suivant dans la fonction ***update()*** du CarController.

```
// CarController.php

public function update(Request $request, $id)
{
    $validatedData = $request->validate([
        'marque' => 'required|max:255',
        'prix' => 'required'
    ]);

    Car::whereId($id)->update($validatedData);

    return redirect('/cars')->with('success', 'Voiture mise à jour avec succèss');
}
```

Vous pouvez maintenant mettre à jour toutes les données dans la base de données.

Étape 11: Créer une fonctionnalité de suppression

Pour supprimer des données de la base de données, nous utiliserons la fonction **destroy()** de CarController.

```
// CarController.php


public function destroy($id)
```

```
{
    $car = Car::findOrFail($id);
    $car->delete();

    return redirect('/cars')->with('success', 'Voiture supprimer avec succèss');
}
```

La fonction **delete ()** est fournie par Laravel pour supprimer les données de la base de données.