**Objectif**: Ajouter un **contrôleur** nommé FournisseurController avec une **route**, une **vue** (fournisseur.blade.php) et une méthode pour calculer le **montant total**.

**Rappel : Contrôleur**

Le contrôleur est la partie qui gère la logique de traitement. Il reçoit les données de la requête, exécute les opérations nécessaires (comme récupérer des données de la base de données, valider les données, etc.), et renvoie les résultats à une vue.

**Étape 1 : Créer le Contrôleur**

php artisan make:controller FournisseurController

* **Ajouter des Méthodes RESTful Automatiquement**

Pour générer un contrôleur avec des méthodes de base RESTful (index, create, store, show, edit, update, destroy), ajoutez l'option --resource :

**Syntaxe :**

php artisan make:controller NomDuController --resource

* Créer un contrôleur FournisseurController avec des méthodes RESTful :

php artisan make:controller FournisseurController --resource

* **Créer un Contrôleur RESTful API**

Si vous travaillez sur une API, vous pouvez inclure uniquement les méthodes essentielles pour une API en ajoutant --api :

php artisan make:controller NomDuController --api

* **Créer un Contrôleur dans un Sous-Dossier**

Si vous souhaitez organiser vos contrôleurs dans des sous-dossiers, utilisez une barre oblique (/) pour indiquer le chemin.

**Exemple** : Créer un contrôleur FournisseurController dans le dossier Admin :

php artisan make:controller Admin/FournisseurController

Le contrôleur sera placé dans app/Http/Controllers/Admin/FournisseurController.php.

* **Ajouter un Modèle Associé**

Un modèle dans Laravel (et dans le modèle de conception MVC, en général) est une classe qui représente les données et la logique métier d'une application. Il est principalement utilisé pour interagir avec la base de données, en définissant la structure des données, les relations entre les entités, et la logique de gestion des données.

**Rôle du Modèle**

Le modèle gère toutes les opérations de manipulation de données, comme la création, la lecture, la mise à jour et la suppression (opérations CRUD). Dans Laravel, les modèles sont basés sur Eloquent, l'ORM (Object-Relational Mapping) intégré, qui simplifie considérablement l'interaction avec la base de données.

Vous pouvez créer un contrôleur qui est directement lié à un modèle en utilisant l'option --model. Cela ajoute des injections de dépendances pour le modèle dans les méthodes générées.

**Exemple** : Créer un contrôleur FournisseurController avec un modèle Fournisseur :

php artisan make:controller FournisseurController --resource --model=Fournisseur

* **Afficher Toutes les Options Disponibles**

Pour voir toutes les options disponibles pour la commande make:controller, utilisez :

php artisan help make:controller

* **A quoi sert les controleurs Laravel RESTful ?**

Contrôleurs Laravel RESTful pour gérer les opérations de base (CRUD : Create, Read, Update, Delete) sur des ressources. Elles sont créées automatiquement lorsque vous utilisez l'option --resource avec la commande make:controller.

**1. index : Afficher une Liste de Ressources**

**Rôle** : Cette méthode est utilisée pour récupérer et afficher une liste de toutes les ressources (par exemple, une liste d'articles, d'utilisateurs, de produits, etc.).

**Type de Requête** : GET

**Exemple**

public function index()

{

$users = User::all(); // Récupérer tous les utilisateurs

return view('users.index', compact('users')); // Afficher la liste

}

* **Route associée :**

Route::get('/users', [UserController::class, 'index']);

**2. store : Créer une Nouvelle Ressource**

**Rôle** : Cette méthode est utilisée pour valider les données et enregistrer une nouvelle ressource dans la base de données.

**Type de Requête** : POST

**Exemple**

public function store(Request $request)

{

$validated = $request->validate([

'nom' => 'required|min:3',

'email' => 'required|email|unique:users',

]);

User::create($validated); // Créer un nouvel utilisateur

return redirect()->route('users.index')->with('success', 'Utilisateur créé avec succès!');

}

* Route **associée** :

Route::post('/users', [UserController::class, 'store']);

**3. show : Afficher une Ressource Spécifique**

**Rôle** : Cette méthode est utilisée pour afficher les détails d'une ressource spécifique, identifiée par son ID.

**Type de Requête** : GET

**Exemple**

public function show($id)

{

$user = User::findOrFail($id); // Trouver l'utilisateur ou retourner une erreur 404

return view('users.show', compact('user'));

}

**Route associée** :

Route::get('/users/{id}', [UserController::class, 'show']);

**4. update : Mettre à Jour une Ressource**

**Rôle** : Cette méthode est utilisée pour valider les données et mettre à jour une ressource existante dans la base de données.

**Type de Requête** : PUT ou PATCH

**Exemple**

public function update(Request $request, $id)

{

$validated = $request->validate([

'nom' => 'required|min:3',

'email' => 'required|email|unique:users,email,'.$id,

]);

$user = User::findOrFail($id);

$user->update($validated); // Mettre à jour l'utilisateur

return redirect()->route('users.index')->with('success', 'Utilisateur mis à jour avec succès!');

}

**Route associée** :

Route::put('/users/{id}', [UserController::class, 'update']);

**5. destroy : Supprimer une Ressource**

**Rôle** : Cette méthode est utilisée pour supprimer une ressource existante de la base de données.

**Type de Requête** : DELETE

**Exemple**

public function destroy($id)

{

$user = User::findOrFail($id);

$user->delete(); // Supprimer l'utilisateur

return redirect()->route('users.index')->with('success', 'Utilisateur supprimé avec succès!');

}

**Route associée** :

Route::delete('/users/{id}', [UserController::class, 'destroy']);

**Mode opératoire de l’exercice** :

* **la Vue fournisseur.blade.php**

Créez un fichier fournisseur.blade.php dans le dossier resources/views.

**Nb :**

Dans Laravel (et plus généralement dans Blade, son moteur de templates), le **@** est un caractère spécial utilisé pour indiquer qu'une directive Blade suit.

* **Qu'est-ce qu'une Directive Blade ?**

Une directive Blade est une instruction spéciale que Blade interprète et traduit en code PHP avant d'envoyer le contenu final au navigateur. Par exemple :

@if, @foreach, @csrf sont des directives Blade.

Elles simplifient l'écriture du code PHP dans les fichiers de vue.

**Formalisme :**

@if($user->isAdmin())

<p>Bienvenue, Cher Administrateur !</p>

@endif

* Pendant le rendu, Blade traduit ce code en PHP pur comme :

<?php if($user->isAdmin()): ?>

<p>Bienvenue, Administrateur !</p>

<?php endif; ?>

**$errors->any()** est une méthode utilisée pour vérifier si des erreurs de validation sont présentes. Elle appartient à l'objet **$errors**, qui est une instance de la classe

**json\_encode($data)** est une fonction en PHP qui permet de convertir une structure de données PHP (comme un tableau ou un objet) en une chaîne de caractères au format JSON (JavaScript Object Notation).

**$errors->all()** est une méthode utilisée pour accéder à tous les messages d'erreur générés lors de la validation des formulaires ou d'autres processus de validation.

<!DOCTYPE html>

<html>

<head>

<title>Fournisseur</title>

<link rel="stylesheet" href="{{ asset('css/style.css') }}"> <!-- Lien vers une feuille de style si nécessaire -->

</head>

<body>

<h1> Fournisseur</h1>

<!-- Afficher les erreurs de validation -->

@if ($errors->any())

<div style="color: red;">

<ul>

@foreach ($errors->all() as $error)

<li>{{ $error }}</li>

@endforeach

</ul>

</div>

@endif

<!-- Afficher le montant total s'il est calculé -->

@isset($montant\_total)

<p><strong>Montant total :</strong> {{ $montant\_total }} </p>

<p><strong>Infos du fournisseur :</strong> {{ json\_encode($data) }}</p>

@endisset

<!-- Formulaire -->

<form action="{{ url('/fournisseur') }}" method="POST">

@csrf

<label for="id">ID :</label>

<input type="number" id="id" name="id" required>

<br><br>

<label for="nom">Nom :</label>

<input type="text" id="nom" name="nom" required>

<br><br>

<label for="tel">Téléphone :</label>

<input type="text" id="tel" name="tel" required>

<br><br>

<label for="bp">Boîte Postale (BP) :</label>

<input type="text" id="bp" name="bp" required>

<br><br>

<label for="designation">Désignation :</label>

<input type="text" id="designation" name="designation" required>

<br><br>

<label for="prix\_unitaire">Prix Unitaire :</label>

<input type="number" step="0.01" id="prix\_unitaire" name="prix\_unitaire" required>

<br><br>

<label for="quantite">Quantité :</label>

<input type="number" id="quantite" name="quantite" required>

<br><br>

<button type="submit">Calculer le Montant Total</button>

</form>

</body>

</html>

* **Ajouter une Méthode dans le Contrôleur**

Ouvrez le fichier FournisseurController.php dans app/Http/Controllers et ajoutez deux méthodes :

**index** : pour afficher la vue du formulaire.

**calculateTotal** : pour calculer le montant total.

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FournisseurController extends Controller

{

// Afficher le formulaire

public function index()

{

return view('fournisseur');

}

// Calculer le montant total

public function calculateTotal(Request $request)

{

// Valider les données du formulaire

$request->validate([

'id' => 'required|numeric',

'nom' => 'required|string|max:100',

'tel' => 'required|numeric',

'bp' => 'required|string|max:50',

'designation' => 'required|string|max:255',

'prix\_unitaire' => 'required|numeric|min:0',

'quantite' => 'required|integer|min:0',

]);

// Calcul du montant total

$prixUnitaire = $request->input('prix\_unitaire');

$quantite = $request->input('quantite');

$montantTotal = $prixUnitaire \* $quantite;

// Passer les données à la vue

return view('fournisseur', [

'montant\_total' => $montantTotal,

'data' => $request->all(),

]);

}

}

* **Les Routes**

Ajoutez les routes dans le fichier routes/web.php :

use App\Http\Controllers\FournisseurController;

// Route pour afficher le formulaire

Route::get('/fournisseur', [FournisseurController::class, 'index']);

// Route pour traiter le formulaire et calculer le montant total

Route::post('/fournisseur', [FournisseurController::class, 'calculateTotal']);