dInstallation Laravel

# Installer PHP

Pour installer php nous devons nous rendre sur le site de php.com afin de récupérer dans l’ongler « download » le .exe qui lancera le téléchargement de php. Laravel a cependant besoin d’une version supérieur à la

Artisan commande

1. php artisan serve - cette commande lance un serveur de développement local pour votre application Laravel.
2. php artisan make:model - cette commande crée un nouveau modèle pour votre application Laravel.
3. php artisan make:migration - cette commande crée une nouvelle migration pour votre base de données.
4. php artisan migrate - cette commande exécute toutes les migrations en attente.
5. php artisan make:controller - cette commande crée un nouveau contrôleur pour votre application Laravel.
6. php artisan make:middleware - cette commande crée un nouveau middleware pour votre application Laravel.
7. php artisan make:auth - cette commande crée les vues et les contrôleurs nécessaires pour l'authentification utilisateur.
8. php artisan route:list - cette commande affiche une liste de toutes les routes disponibles pour votre application Laravel.
9. php artisan tinker - cette commande ouvre une console interactive où vous pouvez tester votre application Laravel.
10. php artisan make:seeder - cette commande crée un nouveau seeder pour votre application Laravel.
11. php artisan db:seed - cette commande exécute tous les seeders en attente.
12. php artisan optimize - cette commande optimise les performances de votre application Laravel en regroupant les fichiers en un seul.
13. php artisan config:cache - cette commande met en cache les fichiers de configuration de votre application Laravel pour améliorer les performances.
14. php artisan make:request - cette commande crée une nouvelle classe de demande pour valider les entrées utilisateur.
15. php artisan make:factory - cette commande crée une nouvelle factory pour générer des données aléatoires pour vos tests.

Dossier Laravel

1. app - ce dossier contient la logique de l'application, y compris les modèles, les contrôleurs, les middlewares et les fournisseurs de services.
2. bootstrap - ce dossier contient les fichiers qui initialisent l'application Laravel, y compris le fichier app.php qui configure le conteneur de service.
3. config - ce dossier contient les fichiers de configuration de l'application Laravel, y compris les fichiers de configuration de la base de données, de l'authentification et de la session.
4. database - ce dossier contient les migrations de la base de données, les seeders et les factories pour générer des données de test.
5. public - ce dossier contient les fichiers qui sont accessibles publiquement à partir de l'application, y compris le fichier index.php qui est le point d'entrée de l'application.
6. resources - ce dossier contient les fichiers de ressources de l'application Laravel, y compris les fichiers de vue, de langues et de fichiers Sass ou Less.
7. routes - ce dossier contient les fichiers de définition de routes pour l'application Laravel, y compris le fichier web.php qui définit les routes pour l'interface utilisateur et le fichier api.php qui définit les routes pour l'API.
8. storage - ce dossier contient les fichiers générés par l'application Laravel, y compris les fichiers de cache, de sessions et les fichiers journaux.
9. tests - ce dossier contient les tests automatisés de l'application Laravel.
10. vendor - ce dossier contient les dépendances de l'application Laravel installées via Composer.
11. .env - ce fichier contient les variables d'environnement pour l'application Laravel, y compris les informations de base de données et les paramètres de configuration.
12. .env.example - ce fichier est un exemple de fichier .env qui doit être personnalisé pour votre environnement.
13. composer.json - ce fichier contient les dépendances PHP requises par l'application Laravel.
14. package.json - ce fichier contient les dépendances JavaScript requises par l'application Laravel.

Explication fichiers routes

1. web.php: ce fichier est utilisé pour définir les routes web de votre application. Les routes web sont les routes qui sont accessibles via un navigateur web et qui retournent une vue HTML. Par exemple, lorsque vous visitez la page d'accueil de votre application, c'est une route web qui est utilisée pour retourner la vue HTML correspondante. Les routes définies dans web.php sont généralement protégées par des sessions et des cookies, ce qui signifie que l'utilisateur doit être authentifié pour y accéder.
2. api.php: ce fichier est utilisé pour définir les routes de l'API de votre application. Les routes de l'API sont utilisées pour fournir une interface de programmation pour votre application, ce qui permet à d'autres applications ou services de communiquer avec votre application en utilisant des requêtes HTTP. Les routes définies dans api.php sont généralement protégées par un jeton d'authentification ou un middleware d'authentification similaire.
3. console.php: ce fichier est utilisé pour définir les commandes de la console de votre application. Les commandes de la console sont des commandes qui peuvent être exécutées à partir de la ligne de commande en utilisant la commande php artisan. Les commandes de la console sont généralement utilisées pour effectuer des tâches telles que la migration de la base de données ou l'envoi de courriels.
4. channels.php: ce fichier est utilisé pour définir les canaux de diffusion de votre application. Les canaux de diffusion sont utilisés pour envoyer des messages en temps réel à des clients connectés à votre application, par exemple en utilisant des websockets ou des connexions HTTP longue durée. Les canaux définis dans channels.php sont généralement protégés par un middleware d'authentification similaire à celui utilisé pour les routes web.

Explication fonction route dans web.php

Dans une application Laravel, le fichier "web.php" est généralement utilisé pour définir des routes qui répondent aux requêtes HTTP pour les pages web. La fonction "Route" est utilisée pour définir une route, en utilisant la syntaxe suivante :

Route::[méthode HTTP]('[URL]', '[action]');

Cette fonction prend trois paramètres :

La méthode HTTP : il s'agit de la méthode HTTP utilisée pour accéder à cette route (par exemple, "get", "post", "put", "patch", "delete", etc.).

L'URL : il s'agit de l'URL qui sera associée à cette route. Cette URL peut contenir des variables de paramètre, qui seront passées à la méthode de contrôleur associée à cette route.

L'action : il s'agit de l'action à effectuer lorsque cette route est appelée. Cette action peut être une fonction anonyme, une fonction d'un contrôleur, une classe de contrôleur avec une méthode, etc.

Par exemple, la route suivante définit une route pour l'URL "/example" avec la méthode HTTP "get", qui appelle la fonction anonyme pour renvoyer une réponse à l'utilisateur :

Route::get('/example', function () {

return "Hello, world!";

});

Les routes définies dans le fichier "web.php" sont généralement destinées à être utilisées pour les pages web, donc elles utilisent souvent les méthodes HTTP "get" et "post" pour récupérer et soumettre des données, respectivement.

Blade commandes importantes

@extends('layout') : cette commande permet d'étendre un layout existant en y ajoutant le contenu de la vue actuelle. Il est possible de préciser le nom du layout que l'on souhaite étendre.

@section('nom\_section') ... @endsection : cette commande permet de définir une section de contenu dans la vue actuelle. Le nom de la section est passé en paramètre de la commande @section, et le contenu de la section est placé entre les commandes @section et @endsection.

@yield('nom\_section') : cette commande permet d'indiquer où insérer le contenu de la section nommée dans le layout parent. Le nom de la section à insérer est passé en paramètre de la commande @yield.

@include('nom\_vue') : cette commande permet d'inclure le contenu d'une vue dans la vue actuelle. Le nom de la vue à inclure est passé en paramètre de la commande @include.

@if ... @elseif ... @else ... @endif : cette commande permet de conditionner l'affichage d'un bloc de contenu en fonction d'une condition. La commande @elseif permet de rajouter une condition supplémentaire et la commande @else permet de définir un bloc de contenu alternatif si toutes les conditions précédentes ne sont pas remplies.

@foreach ... @endforeach : cette commande permet de boucler sur un tableau et d'afficher le contenu de chaque élément. La variable à boucler est passée en paramètre de la commande @foreach, et le contenu à afficher est placé entre les commandes @foreach et @endforeach. Il est possible d'utiliser la commande @empty pour afficher un contenu alternatif si le tableau est vide.

@php ... @endphp : cette commande permet d'écrire du code PHP directement dans la vue. Le code PHP est placé entre les commandes @php et @endphp.

@stack('nom\_stack') : cette commande permet d'empiler du contenu dans une pile nommée. Le nom de la pile est passé en paramètre de la commande @stack. Le contenu à empiler est placé entre les commandes @push et @endpush. Le contenu empilé peut ensuite être affiché en utilisant la commande @stack dans le layout parent.

@isset(...) et @empty(...) : ces commandes permettent de vérifier si une variable est définie ou non, ou si elle est vide ou non. Elles peuvent être utilisées dans des conditions pour afficher du contenu conditionnel.

@verbatim ... @endverbatim : cette commande permet de désactiver le parsing des commandes Blade à l'intérieur d'un bloc de texte. Le contenu du bloc est considéré comme du texte brut et est affiché tel quel.

MVC Laravel

Bien sûr ! Le modèle MVC (Model-View-Controller) est une architecture de conception couramment utilisée dans les applications Web modernes. Laravel est construit sur cette architecture et fournit des outils pour faciliter l'implémentation du modèle MVC.

Voici une explication détaillée de chaque composant du modèle MVC de Laravel :

Le Modèle : Le Modèle représente les données et la logique métier de l'application. Il est responsable de la gestion de la couche d'accès aux données et fournit une interface pour accéder et manipuler ces données. Dans Laravel, les modèles sont généralement stockés dans le répertoire "app/Models" et étendent la classe de base "Illuminate\Database\Eloquent\Model".

La Vue : La Vue est responsable de l'affichage des données à l'utilisateur. Elle reçoit les données du Contrôleur et les formate pour l'affichage. Dans Laravel, les vues sont stockées dans le répertoire "resources/views" et utilisent la syntaxe de modèle Blade pour afficher les données.

Le Contrôleur : Le Contrôleur est responsable de la gestion des requêtes utilisateur et de la récupération des données à partir du Modèle. Il traite les requêtes, récupère les données nécessaires auprès du Modèle, et les passe à la Vue pour l'affichage. Dans Laravel, les contrôleurs sont stockés dans le répertoire "app/Http/Controllers".

Dans une application Laravel typique, la requête utilisateur est d'abord acheminée vers une route définie dans le fichier "routes/web.php". Cette route est associée à une méthode d'un contrôleur, qui traite la requête et renvoie une réponse. La méthode du contrôleur peut récupérer des données à partir d'un Modèle et les passer à une Vue pour l'affichage.

En utilisant cette architecture MVC, il est possible de créer des applications robustes et évolutives, en séparant les préoccupations de l'interface utilisateur, de la logique métier et de la gestion des données

REQUETE SQL

# Permet de savoir qui doit de l’argent à quelle personne

SELECT

u.useName AS User,

SUM(e.expAmount) AS TotalExpenses,

COALESCE(SUM(eu.expAmountPaid), 0) AS TotalPayments,

SUM(e.expAmount) - COALESCE(SUM(eu.expAmountPaid), 0) AS Balance,

GROUP\_CONCAT(CASE WHEN eu.expAmountPaid IS NULL THEN CONCAT(a.actTitle, ' - €', e.expAmount) END SEPARATOR ', ') AS ExpensesOwed,

GROUP\_CONCAT(CASE WHEN eu.expAmountPaid IS NOT NULL THEN CONCAT(a.actTitle, ' - €', eu.expAmountPaid) END SEPARATOR ', ') AS PaymentsMade

FROM t\_users u

LEFT JOIN t\_expenses e ON u.idUsers = e.fkUsers

LEFT JOIN t\_activityExpenses ae ON e.idExpenses = ae.fkExpenses

LEFT JOIN t\_activities a ON ae.fkActivity = a.idActivities

LEFT JOIN t\_expenseUsers eu ON e.idExpenses = eu.fkExpense AND u.idUsers = eu.fkUsers

GROUP BY u.idUsers;

MiddleWare user approuved

Créer une colonne "approuvé" dans la table "users" avec une valeur par défaut de "0".

Créer une vue et une méthode de contrôleur pour l'administrateur afin qu'il puisse approuver les utilisateurs inscrits.

Créer un middleware pour vérifier si l'utilisateur est approuvé.

Enregistrer le middleware dans le fichier Kernel.php.

Ajouter le middleware à toutes les routes de l'application sauf pour les pages d'inscription et de connexion.

Voici un exemple de middleware que vous pouvez utiliser :

php

Copy code

<?php

namespace App\Http\Middleware;

use Closure;

use Illuminate\Support\Facades\Auth;

class ApproveUser

{

public function handle($request, Closure $next)

{

if (Auth::check() && Auth::user()->approved == 0) {

return redirect()->route('approval.required');

}

return $next($request);

}

}

Ce middleware vérifie si l'utilisateur est connecté et si son compte est approuvé. Si l'utilisateur n'est pas approuvé, il est redirigé vers une page "approval.required".

Ensuite, vous pouvez enregistrer ce middleware dans le fichier Kernel.php :

php

Copy code

protected $routeMiddleware = [

'approve' => \App\Http\Middleware\ApproveUser::class,

];

Enfin, vous pouvez ajouter ce middleware à toutes les routes de l'application sauf pour les pages d'inscription et de connexion :

php

Copy code

Route::group(['middleware' => ['approve']], function () {

// toutes les routes de l'application sauf les routes d'inscription et de connexion

});

De cette façon, l'utilisateur ne pourra accéder à aucune page de l'application tant que son compte n'a pas été approuvé par l'administrateur.