



RÉSUMÉ THÉORIQUE – FILIÈRE DÉVELOPPEMENT WEB FULL STACK

Compétence M113 : « Développer en back-end »

Elaboré par :

Mariam MAHDAOUI Formatrice à ISTA Sidi Moumen CASABLANCA

Mohamed CHCHAB Formateur à ISTA BENGUERIR

120 heures



Equipe de rédaction de validation



Equipe de rédaction :

Mme MARIAM MAHDAOUI : Formatrice en Développement Digital

M. MOHAMED CHCHAB : Formateur en Développement Digital

SOMMAIRE



1. Découvrir le Framework PHP Laravel

Découvrir les notions fondamentales des Frameworks PHP
Préparer l'environnement de Laravel

2. Programmer avec Laravel

Installation des outils pour Laravel
Créer un nouveau projet Laravel
Architecture d'un projet Laravel
Laravel Artisan
Lancer un projet Laravel

3. Approfondir la programmation Laravel

Gérer la sécurité Interagir
avec la base de données
Manipuler l'ORM Eloquent

4. Administrer un site à l'aide d'un CMS

Manipuler les éléments essentiels d'un CMS
Personnaliser graphiquement un site à l'aide d'un CMS
Manipuler les outils avancés d'un CMS

MODALITÉS PÉDAGOGIQUES



1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life



3

DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life



4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life



5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage



PARTIE 1

Découvrir le Framework PHP Laravel

Dans ce module, vous allez :

- Découvrir les notions fondamentales des Frameworks PHP
- Préparer l'environnement de Laravel



20 heures



CHAPITRE 1

Découvrir les notions fondamentales des Frameworks PHP

Ce que vous allez apprendre dans ce chapitre :

1. Présentation des Frameworks PHP
2. L'architecture MVC
3. Intérêt du Framework Laravel

 02 heures

Présentation des Frameworks PHP

Qu'est-ce qu'un Framework web ?

Un Framework est **une boîte à outils** pour **aider** les développeurs dans la réalisation de leurs tâches. **Frame (cadre) et work (travail).**

Un Framework contient des composants autonomes qui permettent de **résoudre les problèmes souvent rencontrés par les développeurs** (CRUD, arborescence, normes, sécurités, etc.).

Un Framework n'est pas uniquement une boîte à outils. Il peut aussi désigner une **méthodologie**.

Un framework est donc perçu comme un squelette, regroupant chaque os et articulations (ici composants/librairies) de manière harmonieuse mais surtout de manière fiable !

Pourquoi utiliser un framework ?

L'objectif de l'utilisation d'un Framework est de :

- maximiser la productivité du développeur qui l'utilise.
- faciliter la **maintenance** du l'application Web réalisée.

Voici les trois raisons à utiliser un Framework :

- **Rapidité** : le Framework permet un gain de temps et une livraison beaucoup plus rapide qu'un développement à zéro.
- **Organisation** : l'architecture d'un Framework favorise une bonne organisation du code source (modèle MVC par exemple).
- **Maintenabilité** : l'organisation du Framework facilite la maintenance du logiciel et la gestion des évolutions.

Les critères de choix d'un Framework PHP

1. la courbe d'apprentissage du framework ne devrait pas être trop dure.
2. Le framework PHP doit répondre aux **exigences techniques** de votre projet. (version PHP minimale , supporter la ou les bases de données du projet, ...)
3. Un bon équilibre de **fonctionnalités du framework vis-à-vis du projet**. Choisir un framework minimal pour un petit projet.
4. Une **bonne documentation et un bon support** sont importants pour que vous puissiez tirer le meilleur parti de votre framework PHP.

Les Frameworks PHP les plus populaires

Il est difficile d'obtenir une liste définitive des frameworks PHP. Wikipédia liste 40 frameworks PHP.

Voici quelques-uns des meilleurs frameworks PHP en usage aujourd'hui :

1. Laravel
2. Symfony
3. Zend Framework / Laminas Project
4. Yii Framework
5. ...

Le framework Symphony

Symfony est un excellent choix pour les sites web et les applications qui doivent être évolutifs, il a plusieurs fonctionnalités:

- Supporte la plupart des bases de données.
- a son propre ORM **Doctrine**.
- Utilise le moteur de templating **Twig**, qui est facile à apprendre, rapide et sûr.
- **Packagist** * liste plus de **4 000 paquets Symfony** téléchargeable prête à utiliser.
- dispose d'un support **support professionnel** (support commercial de Sensio Labs) , contrairement à la plupart des autres frameworks PHP.
- S'intègre facilement avec des frameworks populaires comme Drupal.



* Packagist est une sorte de répertoire public de packages PHP utilisables via Composer.

Composer est un gestionnaire de dépendances entre applications et librairies.

Composer permet de gérer pour chaque projet, la liste des modules et bibliothèques nécessaires à son fonctionnement ainsi que leurs versions. Il est utilisable via la console en ligne de commande. De plus, il permet de mettre en place un système d'autoload pour les bibliothèques compatibles.

Présentation des Frameworks PHP



Le framework Laravel

Laravel est présenté comme « **Le framework PHP pour les artisans du web** ».

Lancé en 2011, Laravel se trouve en tête des classements grâce à ses fonctionnalités:

- Offre une solution MVC complète.
- a son propre ORM **Eloquent ORM** .
- Utilise le moteur de templating **Blade**. On peut utiliser PHP dans Blade, contrairement aux autres.
- **Packalyst**, une collection de paquets Laravel, compte plus de 15 000 paquets utilisable .
- Dispose de l'outil de ligne de commande **Artisan Console** qui permet d'automatiser les tâches répétitives et de générer rapidement du code squelette.
- Bénéficie d'une large communauté de développeurs



Le Zend framework/ Laminas Project

Zend Framework est un framework PHP établi de longue date qui est maintenant en transition vers Laminas Project.

La migration vers Laminas est fortement recommandée, car Zend n'est plus mis à jour.

- C'est un framework complètement orienté objet.
- Basé sur une méthodologie agile.
- Il suit strictement le modèle de conception MVC
- La communauté de Laminas dispose d'un forum et d'un groupe Slack pour la collaboration et le support.
- Le seul inconvénient de Zend réside dans le fait qu'il n'est pas aussi facile à maîtriser.



Le Yii Framework

Le nom de ce framework, **Yii**, signifie « simple et évolutif » en chinois. Il signifie également « **Yes, It Is !** ».

Les principaux avantages de l'utilisation du framework Yii sont:

- C'est assez facile à installer et à démarrer
- Il est préchargé avec un modèle Bootstrap
- Il est livré avec un design élégant pour commencer rapidement
- C'est un framework PHP qui est également livré avec un débogueur
- Il est livré avec un générateur de code de classe extrêmement puissant appelé **Gii**. Le **générateur de code Gii** peut rapidement construire un squelette de code pour vous, ce qui permet de gagner du temps.
- La communauté Yii offre un support en direct via Slack ou IRC.



Ce que vous devez savoir avant d'utiliser un framework PHP

Avant d'utiliser un framework PHP, il faut connaître :

- Le **langage PHP** lui-même. La plupart des frameworks fonctionnent avec PHP version 7.2 ou supérieure.
- Le PHP **orienté objet**, car la plupart des frameworks PHP modernes sont orientés objet.
- **Les bases de données et la syntaxe SQL**. Chaque framework PHP a sa propre liste de bases de données supportées.
- **ORM** (Object-Relational-Mapping) est une méthode d'accès aux données d'une base de données utilisant une syntaxe orientée objet au lieu d'utiliser le langage SQL. De nombreux frameworks PHP ont leur propre ORM intégré.
- Le fonctionnement **les serveurs web** comme Apache et Nginx.
- Architecture **MVC** (Modèle – Vue – Controleur). Les frameworks PHP suivent généralement le modèle de conception MVC
- L'utilisation d'une **interface en ligne de commande** (commande-line interface ou CLI) permet de se sentir à l'aise dans un framework PHP.



CHAPITRE 1

Découvrir les notions fondamentales des Frameworks PHP

Ce que vous allez apprendre dans ce chapitre :

1. Présentation des Frameworks PHP
2. L'architecture MVC
3. Intérêt du Framework Laravel

 02 heures

L'architecture MVC

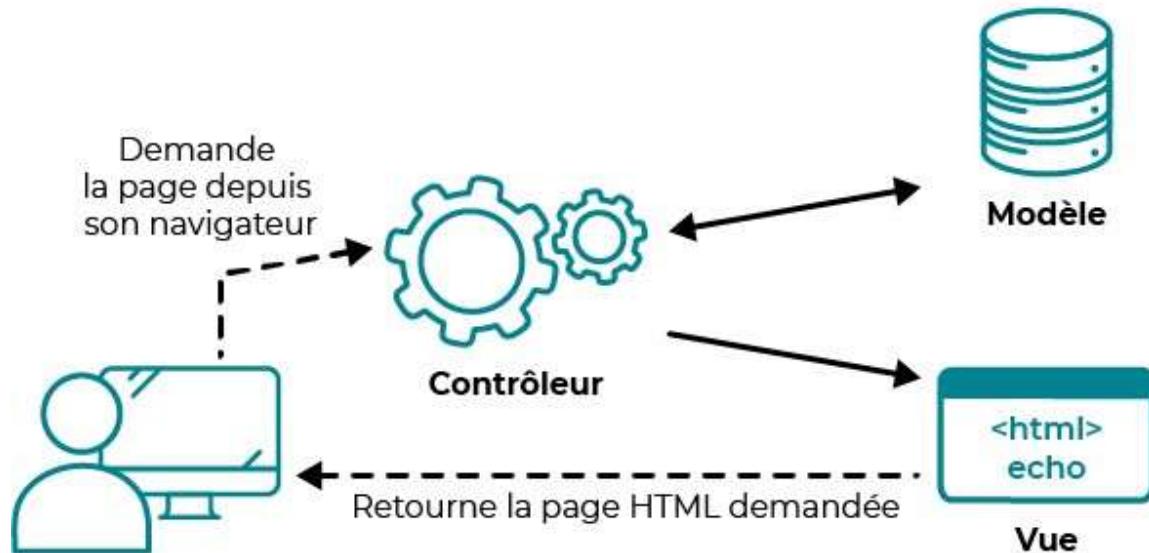
Les frameworks PHP suivent généralement le plus célèbres *design patterns* qui s'appelle MVC, qui signifie **Modèle - Vue - Contrôleur**.

Le but du pattern MVC est de séparer la logique du code en trois parties que l'on retrouve dans des fichiers distincts.

- **Modèle** : cette partie gère la **logique métier**. Le modèle communique avec la BDD. C'est lui qui s'occupe de récupérer un article en base par exemple. On y trouve des algorithmes complexes et des requêtes SQL.
- **Vue** : cette partie se concentre sur **l'affichage**. Elle affiche à l'utilisateur des données contenues dans des variables. On y trouve essentiellement du code HTML mais aussi quelques boucles et conditions PHP très simples.
- **Contrôleur** : cette partie gère les **échanges** avec l'utilisateur. C'est en quelque sorte l'intermédiaire entre l'utilisateur, le modèle et la vue. Le modèle lui transmet les données récupérées en base de données, puis il transmet ces données à la vue.

Laravel propose le modèle MVC mais ne l'impose pas.

L'architecture MVC



Le client et l'architecture MVC

1. Le contrôleur reçoit des requêtes de l'utilisateur.
2. Le contrôleur va demander au modèle d'effectuer certaines actions (lire des informations depuis une base de données,...) et de lui renvoyer les résultats.
3. Le contrôleur va *adapter* ce résultat et le donner à la vue.
4. Enfin, le contrôleur va renvoyer la nouvelle page HTML, générée par la vue, à l'utilisateur.



CHAPITRE 1

Découvrir les notions fondamentales des Frameworks PHP

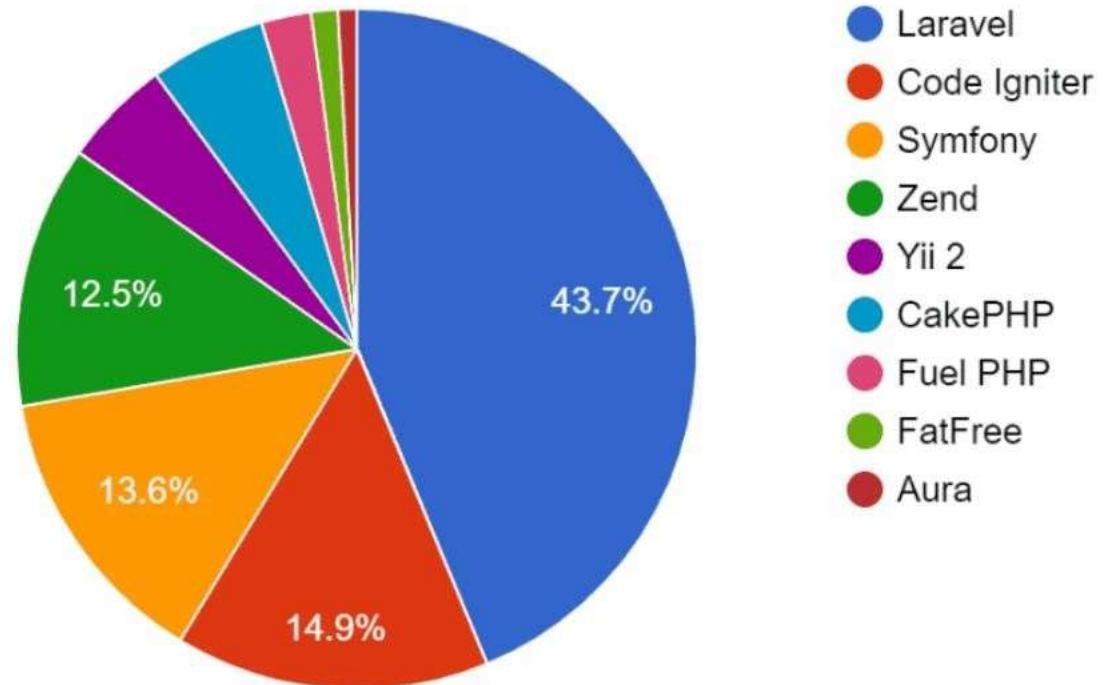
Ce que vous allez apprendre dans ce chapitre :

1. Présentation des Frameworks PHP
2. L'architecture MVC
3. Intérêt du Framework Laravel

02 heures

Pourquoi choisir le Framework Laravel ?

Laravel est le framework PHP le plus utilisé au monde



<https://laravel.sillo.org/cours-laravel-8-les-bases-presentation-generale/>

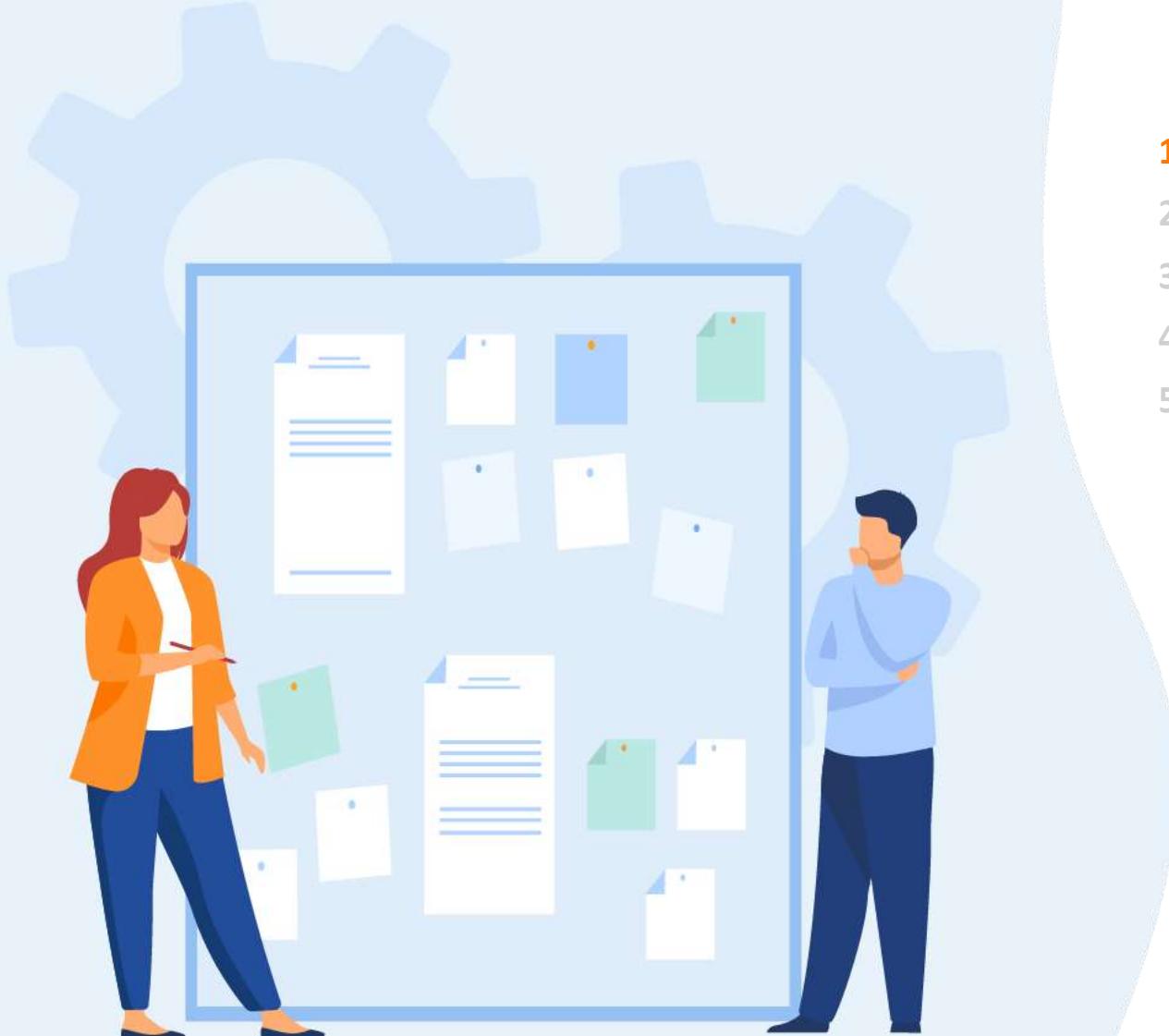
Intérêt du Framework Laravel

Pourquoi choisir le Framework Laravel ?

1. **une sécurité de haut niveau:** Avec Laravel, votre application Web ne présente aucun risque d'injections SQL involontaires et cachées.
2. **Performance:** Laravel propose divers outils qui aident les développeurs à améliorer leurs performances.
3. **Bibliothèques orientées objet:** Laravel possède des bibliothèques orientées objet et d'autre pré installées, qui ne se trouvent dans aucun autre framework PHP. Comme la bibliothèque d'authentification.
4. **Documentation et communauté:** Laravel possède une puissante communauté de développeurs qui fournit en une assistance en permanence.
5. **Tests unitaires:** Avec les tests unitaires de Laravel, chaque module de votre application Web est testé avant la mise en ligne du site.
6. **Intégration aux services de messagerie :** Il fournit également des pilotes pour Mailgun, SMTP, Mandrill, SparkPost, la fonction de courrier électronique de PHP et Amazon SES.
7. **Créateur d'applications multilingues:** Le framework Laravel vous aide donc à créer facilement et rapidement votre application Web dans différentes langues.
8. **Tutoriels Laracasts :** Laravel propose des fonctionnalités de Laracasts, un mélange de tutoriels vidéo gratuits et payants qui vous montrent comment utiliser Laravel pour le développement.

Résumé

- Un framework fait gagner du temps et donne l'assurance de disposer de composants bien codés et fiables.
- Laravel est un framework novateur, complet, qui utilise les possibilités les plus récentes de PHP et qui est impeccamment codé et organisé.
- La documentation de Laravel est complète, précise et de plus en plus de tutoriels et exemples apparaissent sur la toile.
- Laravel adopte le patron MVC, mais ne l'impose pas.
- Laravel est totalement orienté objet.



CHAPITRE 2

Préparer l'environnement de Laravel

1. Installation des outils pour Laravel
2. Créer un nouveau projet Laravel
3. Architecture d'un projet Laravel
4. Laravel Artisan
5. Lancer le projet Laravel

02 - Préparer l'environnement de Laravel

Installation des outils pour Laravel



Les outils pour Laravel

Pour travailler avec Laravel, vous avez besoin de :

1. PHP
2. un serveur Local (Apache ou Nginx), permettant l'execution du PHP
3. un serveur de base de données pour gérer vos bases de données.
4. Composer, un gestionnaire de dépendances pour PHP
5. IDE et Editeurs de code (Visual Studio Code, Atom, PHPStorm ...)
6. Navigateurs Web : Laravel est compatible sur tout les navigateurs récents (Chrome, Firefox ou Safari)

Nous vous recommandons d'installer les dernières versions de ces logiciels ayant une version PHP plus récente
(PHP 8.X pour Laravel 9, PHP 7.4 pour Laravel 8).

Document officielle : <https://laravel.com/docs/9.x>

02 - Préparer l'environnement de Laravel

Installation des outils pour Laravel



Téléchargement et installation de PHP

- Pour installer PHP vous allez devoir vous rendre sur la page de téléchargement Windows du site officiel : <http://windows.php.net/download/>.
- Installer la version 7.3 ou plus
- Vérifier que php est bien installé en exécutant la commande dans la console php --version.



Configuration des extensions nécessaires pour PHP

Les extensions **PDO**, **Tokenizer**, **OpenSSL** et **Mbstring** de PHP doivent être activées.

Dans le fichier Le fichier de configuration de PHP **php.ini** , décommenter les lignes ci-dessous en enlevant le point virgule au début des lignes :

```
;extension=php_mbstring.dll  
;extension=php_openssl.dll
```

02 - Préparer l'environnement de Laravel

Installation des outils pour Laravel



Composer - Définition

Composer est un logiciel gestionnaire de dépendances libre écrit en PHP. Il permet à ses utilisateurs de déclarer et d'installer les bibliothèques dont le projet principal a besoin. Il permet de télécharger et de mettre à jour des bibliothèques externes.

Les bibliothèques externes permettent de réutiliser le code écrit par d'autres personnes pour simplifier le développement.

Exemple:

- Pour gérer des dates, vous pouvez utiliser **Carbon**
- Pour gérer les paiements Paypal, vous pouvez utiliser la bibliothèque officielle **PayPal PHP SDK**.

Composer permet également de créer des projets Laravel et de télécharger le framework.

Le framework Laravel est un simple assemblage de plusieurs dizaines de bibliothèques.



Installation de Composer

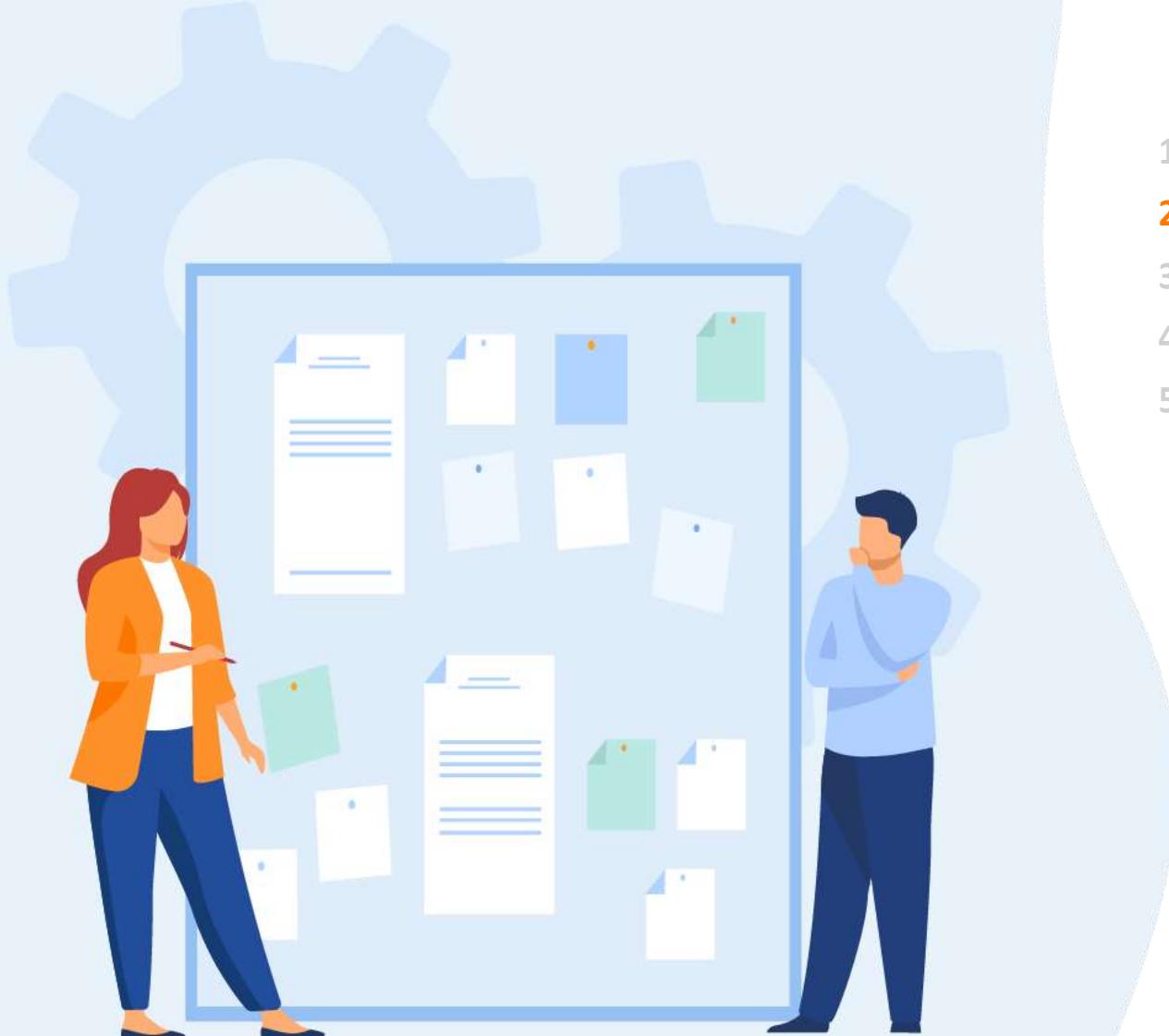
Pour installer Composer, il suffit de télécharger un installateur <https://getcomposer.org/download/> et téléchargez **Composer-Setup.exe**.

Vérifiez lors de l'installation que le chemin par défaut vers PHP est bien C:\PHP\php.exe, car Composer est un fichier PHP et a besoin d'être exécuté.

Vérifications

Pour vérifier que tout fonctionne exécuter composer sur la ligne de commande comme suit:

ici on peut voir que j'ai la version 2.3.10 de composer installée.



CHAPITRE 2

Préparer l'environnement de Laravel

1. Installation des outils pour Laravel
2. **Créer un nouveau projet Laravel**
3. Architecture d'un projet Laravel
4. Laravel Artisan
5. Lancer le projet Laravel

02 - Préparer l'environnement de Laravel

Installation de Laravel



Utiliser l'installateur de Laravel

- Installer Laravel dans notre ordinateur : Ouvrez votre terminal (invite de commande) et tapez la ligne suivante:

```
C:\ Invite de commandes  
Microsoft Windows [version 10.0.19043.1766]  
(c) Microsoft Corporation. Tous droits réservés.  
C:\Users\HINNOVIS>composer global require laravel/installer
```

require: ajoute la bibliothèque en paramètre au fichier *composer.json* et l'installe.

Créer notre premier projet laravel (*project_laravel*) – Méthode 1 :

Taper la commande ***laravel new project_laravel*** en plaçant le terminal dans le répertoire de travail (ex. www ou htdocs):

```
C:\ Invite de commandes - laravel new project_laravel  
Microsoft Windows [version 10.0.19043.1766]  
(c) Microsoft Corporation. Tous droits réservés.  
C:\Users\HINNOVIS>cd C:\Apache24\htdocs  
C:\Apache24\htdocs>laravel new project_laravel
```

02 - Préparer l'environnement de Laravel

Créer un nouveau projet Laravel



Créer un nouveau projet Laravel avec Composer – Méthode 2:

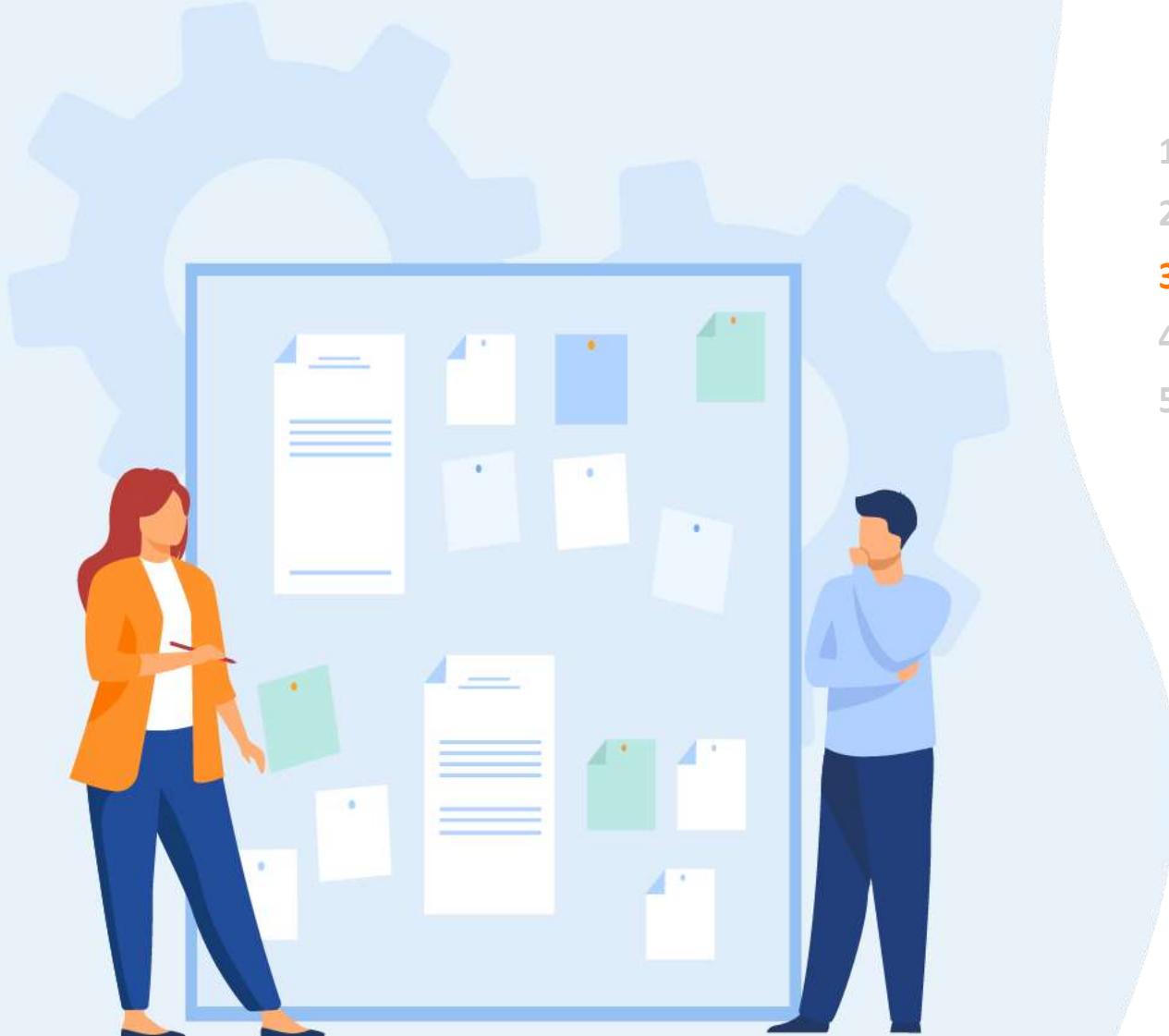
Créer un deuxième projet à la racine du serveur nommé `project_laravel2` : avec la commande suivante:

- Dirigez-vous vers votre dossier à la racine de votre serveur www ou htdocs et exécuter la commande :
`composer create-project laravel/laravel project_laravel2`

```
C:\Invite de commandes  
Microsoft Windows [version 10.0.19043.1766]  
(c) Microsoft Corporation. Tous droits réservés.  
  
C:\Users\HINNOVIS>cd C:\Apache24\htdocs  
  
C:\Apache24\htdocs>composer create-project laravel/laravel project_laravel2
```

Ici nous avons demandé à composer de créer une installation Laravel dans le dossier «blog».

Cette commande installera automatiquement la dernière version stable de Laravel.



CHAPITRE 2

Préparer l'environnement de Laravel

1. Installation des outils pour Laravel
2. Créer un nouveau projet Laravel
- 3. Architecture d'un projet Laravel**
4. Laravel Artisan
5. Lancer le projet Laravel

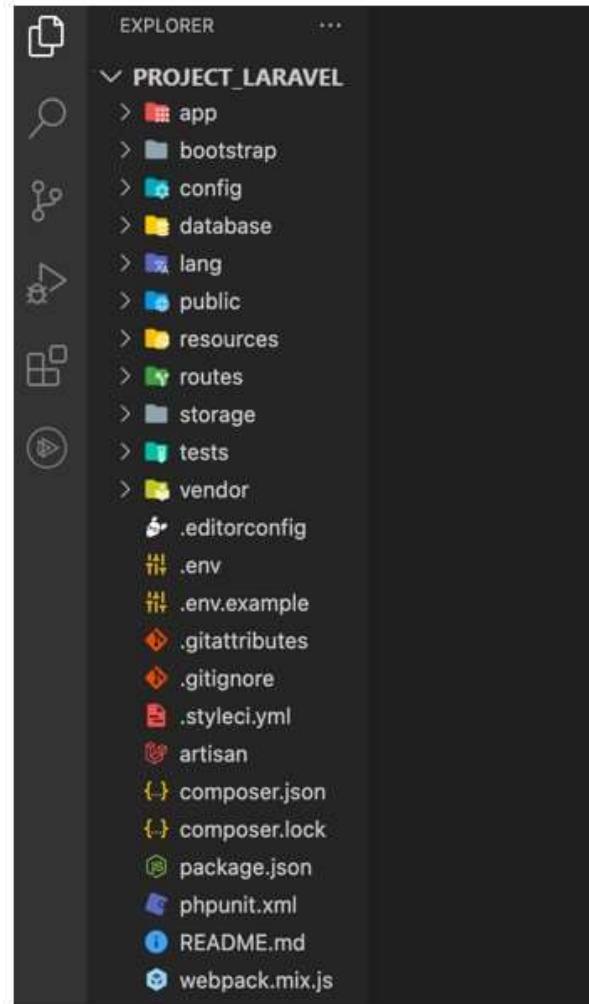
02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel



Architecture d'un projet Laravel

Ouvrez le dossier project_laravel avec l'éditeur Visual Code ou autre ,
vous devez avoir ceci comme résultat :



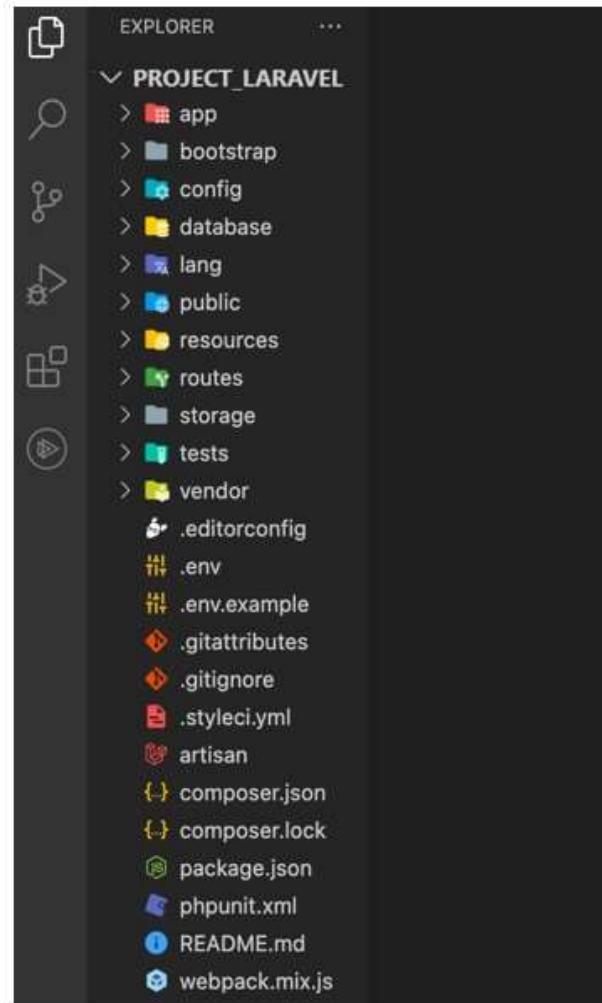
02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel



Les répertoires racines d'un projet Laravel

- ❑ **/app** : contient le code principal de votre application. Il contient deux répertoires principal http et Models.
 - ❑ **Le répertoire Http** : contient les contrôleurs et les middleware.
 - ❑ **Le répertoire Models** : contient toutes les classes de modèle Eloquent.
- ❑ **/config** : contient tous les fichiers de configuration de votre application, authentification, namespace, mails, base de données etc...
- ❑ **/database** : contient vos migrations de base de données, les données factices (faux data) de vos modèles.
- ❑ **/lang**: contient tous les fichiers de langue de votre application. Vous permettant de gérer plusieurs langues dans votre site web. *Ce répertoire est devenu racine que à partir de la version 9 de Laravel, dans les versions ultérieures, le répertoire lang se trouve dans le dossier public.*



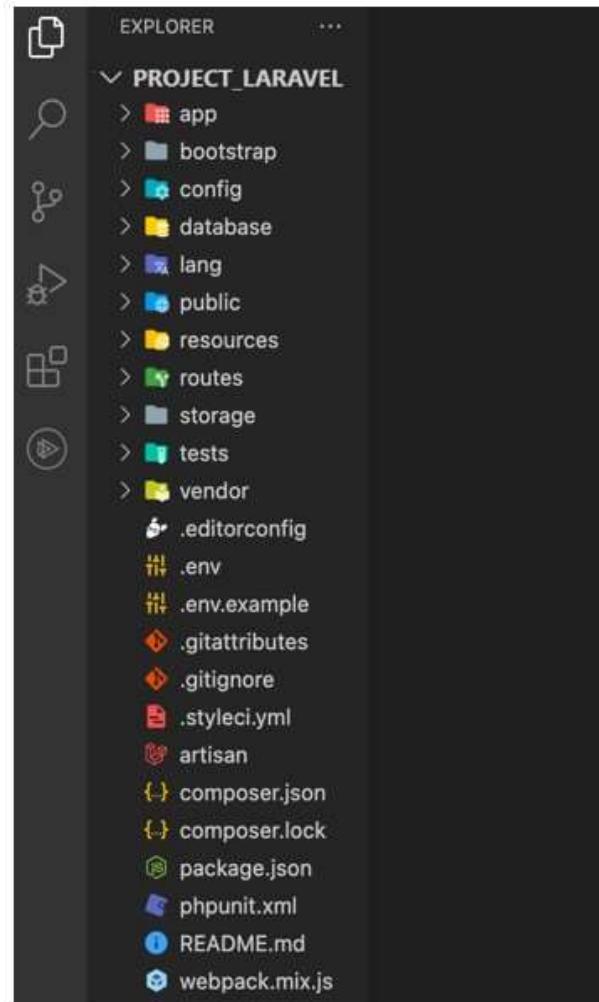
02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel



Les répertoires racines d'un projet Laravel

- ❑ **/public:** par convention comme pour la majorité des frameworks il s'agit du seul dossier accessible depuis le serveur où les fichiers sont accessibles depuis votre site (images, feuilles de style et scripts principalement). Contient le fichier **index.php**, qui est le point d'entrée pour toutes les demandes entrant dans votre application et configure le chargement automatique.
- ❑ **/resources:** contient vos **vues** ainsi que vos fichiers bruts non compilés tels que CSS, SASS ou JavaScript
- ❑ **/routes :**contient toutes les définitions des URLs pour votre application.
Par défaut, 4 fichiers de route sont inclus avec Laravel : web.php, api.php, console.php et channels.php.
 - **Le fichier web.php:** contient des routes placés dans le **RouteServiceProvider** (web groupe middleware), qui fournit l'état de la session, la protection CSRF et le cryptage des cookies.



02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel



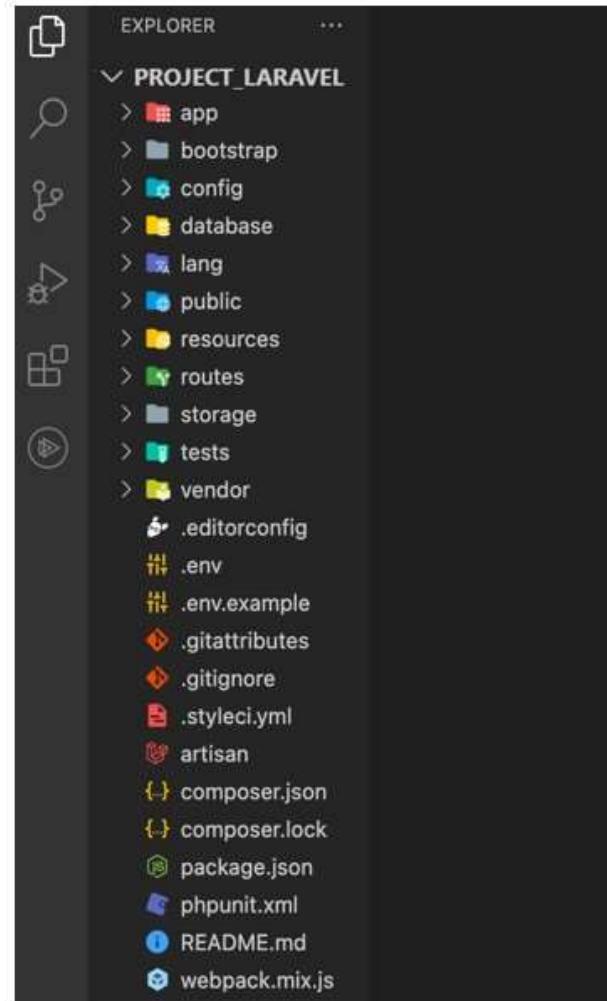
composer.json

- **/storage:** contient vos logs, modèles de vues (blades) compilés, sessions basées sur des fichiers, fichiers caches et autres fichiers générés par le framework.

Ce répertoire est divisé en répertoires **app**, **framework** et **logs**.

- **storage/app** peut être utilisé pour stocker tous les fichiers générés par votre application.
- **storage/framework** est utilisé pour stocker les fichiers et les caches générés par le framework.
- **storage/logs** contient les fichiers journaux de votre application.
- **storage/app/public**: peut être utilisé pour stocker des fichiers générés par l'utilisateur, tels que des avatars de profil, les images des vos publications, qui doivent être accessibles au public.
 - Vous devez créer un lien symbolique(raccourcis) **public/storage** pointant vers ce répertoire.
 - Vous pouvez créer le lien à l'aide de la commande Artisan : **php artisan storage:link**

- **/bootstrap** : démarrage de l'app.



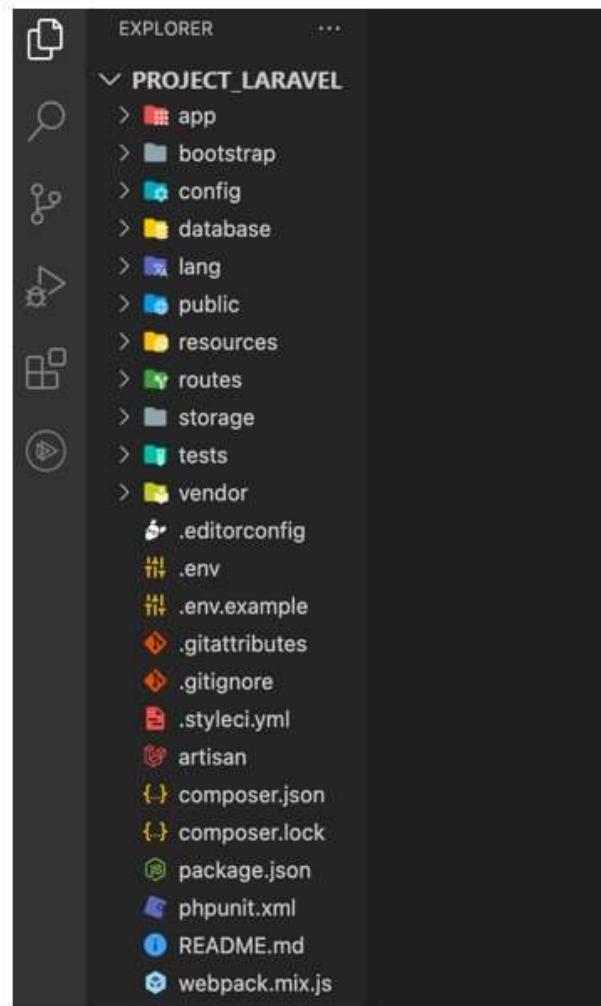
02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel



Les répertoires racines d'un projet Laravel

- ❑ **/tests:** contient vos tests automatisés. Des exemples de tests unitaires PHPUnit et de tests de fonctionnalités sont fournis prêts à l'emploi.
 - Chaque classe de test doit être suffixée par le mot Test.
 - Vous pouvez exécuter vos tests à l'aide des commandes `php unit` ou `php vendor/bin/phpunit`.
 - Ou, si vous souhaitez une représentation plus détaillée et plus belle de vos résultats de test, vous pouvez exécuter vos tests à l'aide de la commande Artisan `php artisan test`.



02 - Préparer l'environnement de Laravel

Architecture d'un projet Laravel

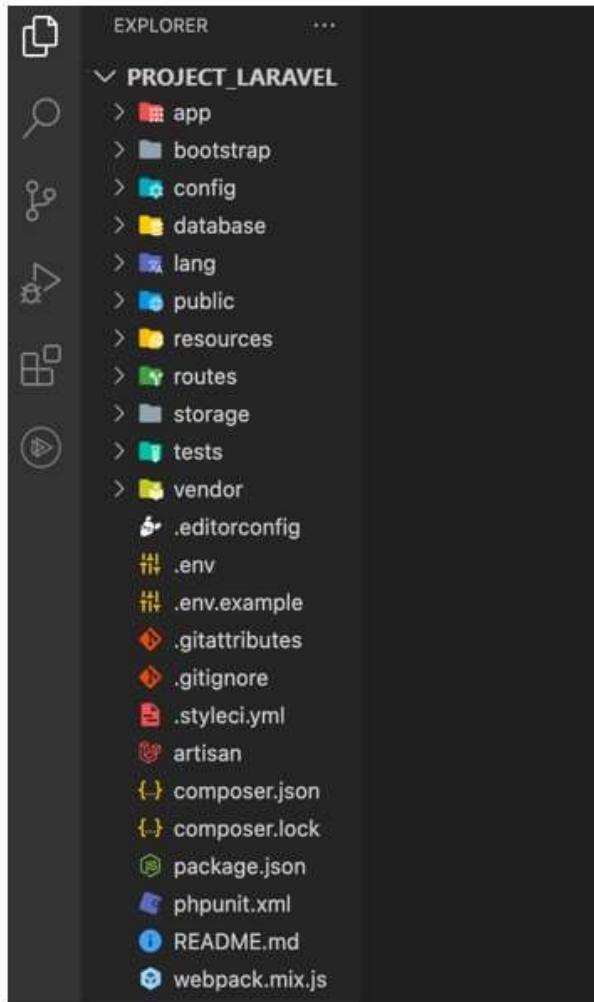


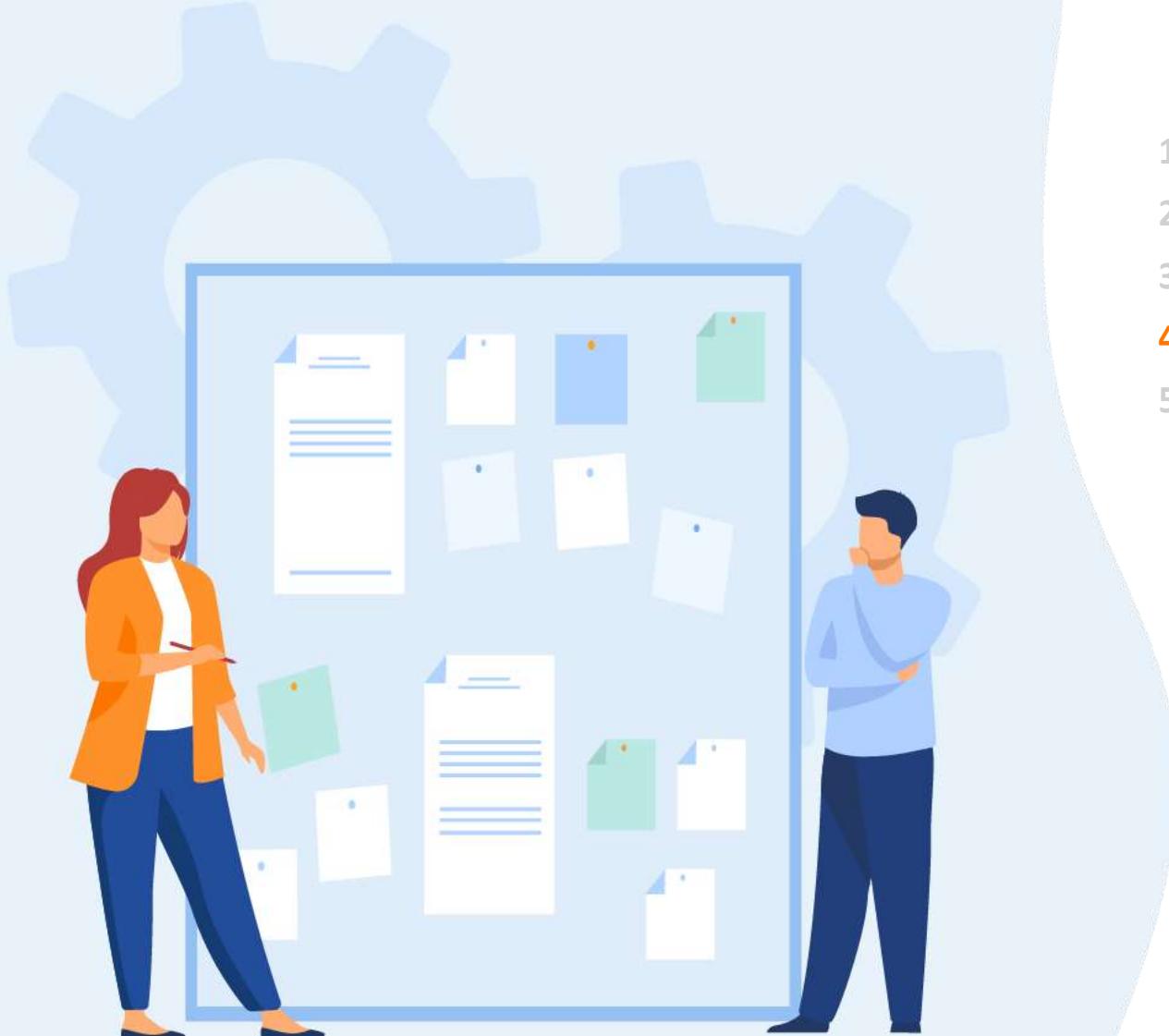
Qu'est ce que le fichier composer.json ?

C'est le fichier qui stocke tous les paquets (dépendances) d'un projet, ainsi que leurs versions, mais peut aussi être utilisé pour définir un projet quand il est publié sur Git, par exemple.

```
{
  "name": "laravel/laravel",
  "type": "project",
  "description": "The Laravel Framework.",
  "keywords": [ "framework", "laravel" ],
  "license": "MIT",
  "require": {
    "php": "^7.1.3",
    "fideloper/proxy": "^4.0",
    "laravel/framework": "5.7.*",
    "laravel/tinker": "^1.0"
  },
  [...]
}
```

(ici ma version de Laravel est la 5.7 et nécessite une version php qui soit au moins égal à la 7.1.3)





CHAPITRE 2

Préparer l'environnement de Laravel

1. Installation des outils pour Laravel
2. Créer un nouveau projet Laravel
3. Architecture d'un projet Laravel
4. **Laravel Artisan**
5. Lancer le projet Laravel

02 - Préparer l'environnement de Laravel

Laravel Artisan



Qu'est ce que Laravel Artisan?

Laravel Artisan est une Interface en Ligne de Commande (CLI) qui va vous permettre de gérer votre application en lançant des commandes via le terminal (effacer le cache de l'application, gérer des modèles, des contrôleurs, des routes...)

Laravel Artisan: commandes utiles

1. Afficher la liste des commandes artisan

```
php artisan list
```

2. Pour afficher un écran d'aide, faites précéder le nom de la commande de help

```
php artisan help migrate
```

3. Pour créer un modèle

```
php artisan make:model NomDuModel
```

4. Vérifier la version de Laravel installée

```
php artisan -version
```

02 - Préparer l'environnement de Laravel

Laravel Artisan



Laravel Artisan: commandes utiles (suite)

5. lancer le serveur , lancer le projet Laravel

```
php artisan serve
```

6. voir toutes les routes de votre application

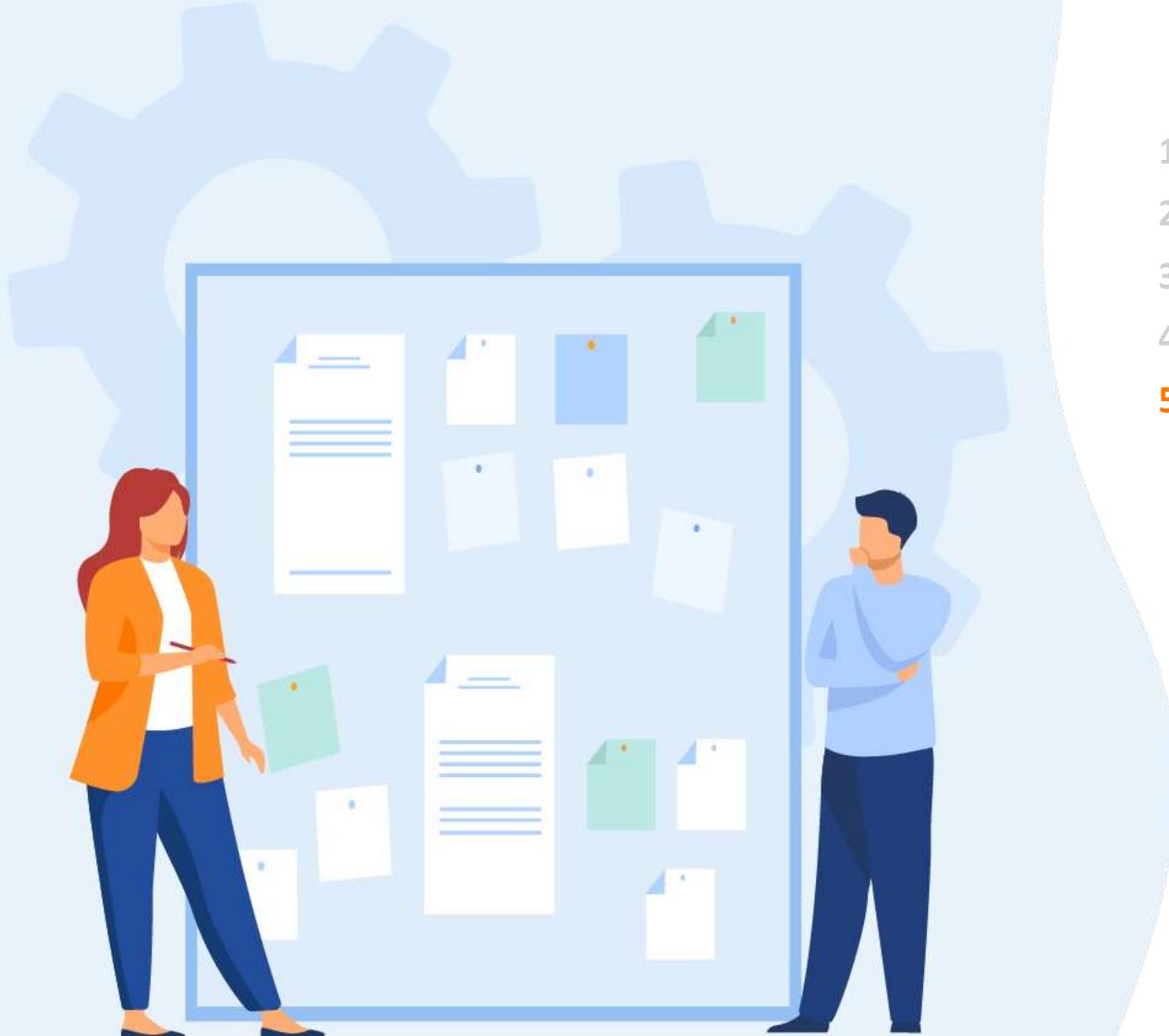
```
artisan route:list
```

7. Exécuter toutes les migrations:

```
php artisan migrate
```

8. faire un retour en arrière pour la dernière migration

```
php artisan migrate : rollback
```



CHAPITRE 2

Préparer l'environnement de Laravel

1. Installation des outils pour Laravel
2. Créer un nouveau projet Laravel
3. Architecture d'un projet Laravel
4. Laravel Artisan
5. Lancer le projet Laravel

02 - Préparer l'environnement de Laravel

Lancer le projet Laravel



Lancer le projet Laravel

1. Placez le terminal dans le dossier racine de votre projet et taper les commandes: ***php artisan key:generate*** ensuite ***php artisan serve*** :

```
C:\Apache24\htdocs>cd project_laravel  
C:\Apache24\htdocs\project_laravel>php artisan key:generate  
    INFO Application key set successfully.  
  
C:\Apache24\htdocs\project_laravel>php artisan serve  
    INFO Server running on [http://127.0.0.1:8000].  
    Press Ctrl+C to stop the server
```

Sous windows 10 , en cas d'erreur exécuter la commande suivante : **composer install --ignore-platform-reqs**

02 - Préparer l'environnement de Laravel

Lancer le projet Laravel



Lancer le projet Laravel

1. Ouvrez le navigateur et allez sur <http://localhost:8000>, et vous obtiendrez le résultat suivant :

The screenshot shows a web browser window with the URL `127.0.0.1:8000` in the address bar. The page itself is the Laravel welcome screen, featuring the red Laravel logo and the word "Laravel". Below the logo, there are four main sections: "Documentation", "Laracasts", "Laravel News", and "Vibrant Ecosystem". Each section has a small icon and a brief description. At the bottom of the page, there are links for "Shop" and "Sponsor", and a footer note "Laravel v9.22.1 (PHP v8.1.6)".

Documentation
Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

Laracasts
Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News
Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

Vibrant Ecosystem
Laravel's robust library of first-party tools and libraries, such as Forge, Vapor, Nova, and Envoyer help you take your projects to the next level. Pair them with powerful open source libraries like Cashier, Dust, Echo, Horizon, Sanctum, Telescope, and more.

Résumé

- Pour son installation et sa mise à jour, Laravel utilise le gestionnaire de dépendances Composer.
- La création d'une application Laravel se fait à partir de la console avec une simple ligne de commande.
- Laravel est organisé en plusieurs dossiers.
- Le dossier public est le seul qui doive être accessible par le serveur.



PARTIE 1

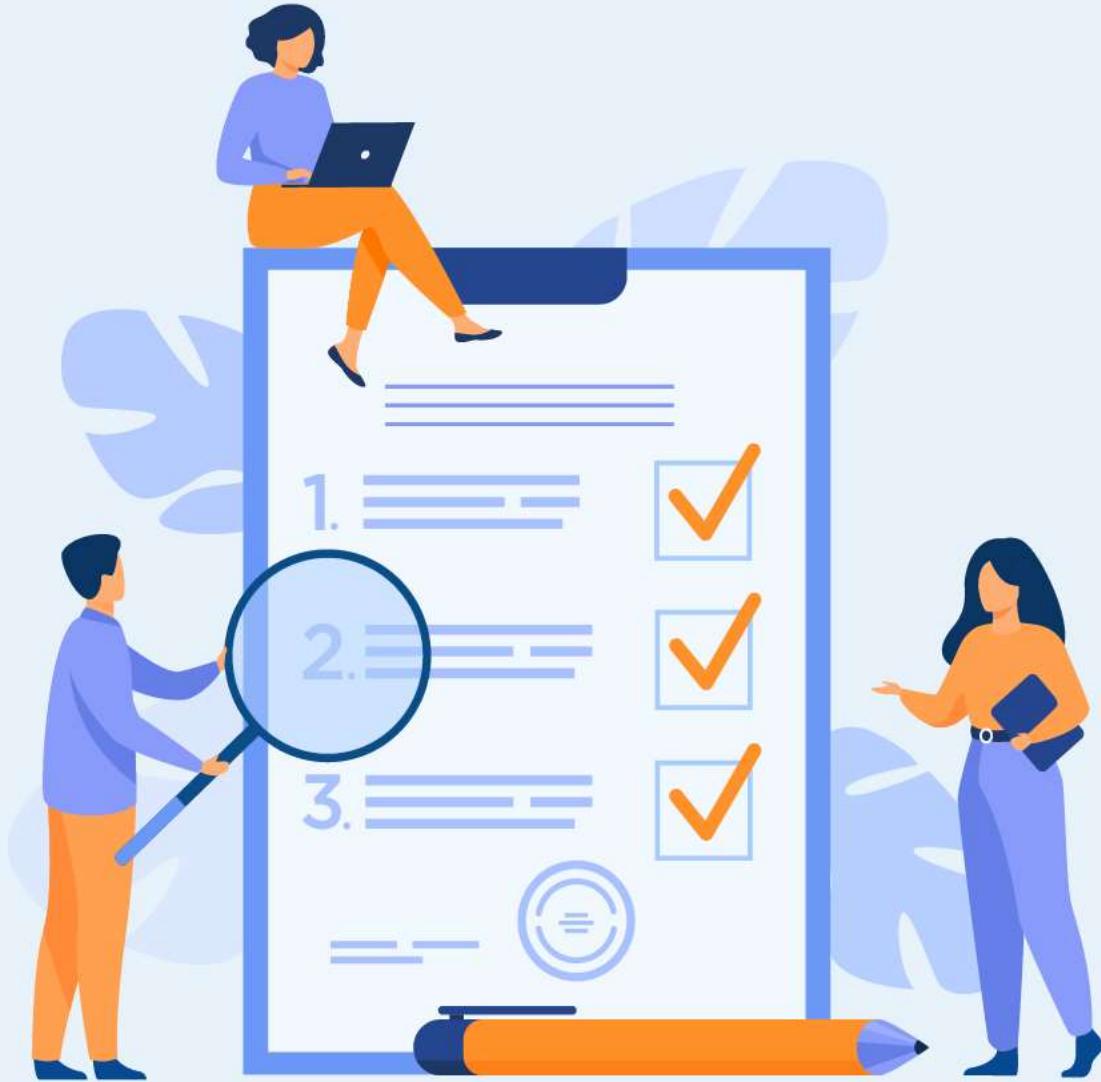
Programmer avec Laravel

Dans ce module, vous allez :

- Connaître les fondements du modèle MVC Laravel
- Maîtriser le Framework Laravel



30 heures



CHAPITRE 1

Gestion du routage

Ce que vous allez apprendre dans ce chapitre :

- Routage de base (redirection, affichage, liste de routes)
- Paramètres de routage
- Routes nommées
- Groupe de routage
- Liaisons de modèles de routes
- Routes de repli
- (spoofing) de la méthode du formulaire
- Accès à la route courante
- Cross Origin Resource Sharing (CORS)
- Mise en cache des routes



05 heures



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

01 - Gestion du routage

Routage de base



- Les routes **Laravel** les plus basiques acceptent un URI et une fermeture, fournissant une méthode très simple et expressive de définition des routes et du comportement sans fichiers de configuration de routage compliqués :

```
use Illuminate\Support\Facades\Route;
```

```
Route::get('/greeting', function () {  
    return 'Hello World';  
});
```

Les fichiers de routage par défaut

- Toutes les routes **Laravel** sont définies dans vos fichiers de route, qui se trouvent dans le répertoire **routes**. Ces fichiers sont automatiquement chargés par le fichier **App\Providers\RouteServiceProvider**. Le **routes/web.php** fichier définit les itinéraires qui sont destinés à votre interface Web. Ces routes sont affectées au webgroupe middleware, qui fournit des fonctionnalités telles que l'état de session et la protection CSRF. Les routes dans **routes/api.php** sont sans état et sont affectées au **api** groupe middleware.
- Pour la plupart des applications, vous commencerez par définir des routes dans votre fichier **routes/web.php**. Les itinéraires définis dans **routes/web.php** sont accessibles en saisissant l'URL de l'itinéraire défini dans votre navigateur. Par exemple, vous pouvez accéder à l'itinéraire suivant en naviguant vers <http://example.com/user> dans votre navigateur :

```
use Illuminate\Support\Facades\Route;
```

```
Route::get('/greeting', function () {  
    return 'Hello World';  
});
```

- Les routes définies dans le fichier **routes/api.php** sont imbriquées dans un groupe de routes par le **RouteServiceProvider**. Dans ce groupe, le préfixe URI **/api** est automatiquement appliqué, vous n'avez donc pas besoin de l'appliquer manuellement à chaque route du fichier. Vous pouvez modifier le préfixe et d'autres options de groupe de routage en modifiant votre classe **RouteServiceProvider**.

Méthodes de routeur disponibles

- Le routeur vous permet d'enregistrer des routes qui répondent à n'importe quel verbe HTTP :

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

- Parfois, vous devrez peut-être enregistrer une route qui répond à plusieurs verbes HTTP. Vous pouvez le faire en utilisant la méthode **match**. Ou, vous pouvez même enregistrer une route qui répond à tous les verbes HTTP en utilisant la méthode **any** :

```
Route::match(['get', 'post'], '/', function () {
// 
});

Route::any('/', function () {
// 
});
```

01 - Gestion du routage

Routage de base



Injection de dépendance

- Vous pouvez indiquer toutes les dépendances requises par votre itinéraire dans la signature de rappel de votre itinéraire. Les dépendances déclarées seront automatiquement résolues et injectées dans le rappel par le conteneur de service Laravel. Par exemple, vous pouvez donner un indice de type à la classe **Illuminate\Http\Request** pour que la requête HTTP actuelle soit automatiquement injectée dans votre rappel de route :

```
use Illuminate\Http\Request;

Route::get('/users', function (Request $request) {
// ...
});
```

Figure 1 : les navigateurs les plus utilisés

Redirection

- Si vous définissez une route qui redirige vers un autre **URI**, vous pouvez utiliser la méthode **Route::redirect**. Cette méthode fournit un raccourci pratique pour que vous n'ayez pas à définir une route ou un contrôleur complet pour effectuer une simple redirection :

```
Route::redirect('/ici', '/là-bas');
```

- Par défaut, **Route::redirect** renvoie un code d'état **302**. Vous pouvez personnaliser le code d'état à l'aide du troisième paramètre facultatif :

```
Route::redirect('/ici', '/là-bas', 301);
```

- Ou, vous pouvez utiliser la méthode **Route::permanentRedirect** pour renvoyer un code d'état **301**:

```
Route::permanentRedirect ('/ici', '/là-bas');
```

Afficher les routes

- Si votre itinéraire ne doit renvoyer qu'une vue , vous pouvez utiliser la méthode **Route::view**. Comme la méthode **redirect**, cette méthode fournit un raccourci simple pour que vous n'ayez pas à définir une route ou un contrôleur complet. La méthode **view** accepte un URI comme premier argument et un nom de vue comme second argument. De plus, vous pouvez fournir un tableau de données à transmettre à la vue en tant que troisième argument facultatif :

```
Route::view('/welcome', 'welcome');
```

```
Route::view('/welcome', 'welcome', ['name' => 'Ahmed']);
```

La liste des routes

- La commande **Artisan route:list** peut facilement fournir une vue d'ensemble de toutes les routes définies par votre application :

```
php artisan route:list
```

- Par défaut, le middleware de route affecté à chaque route ne sera pas affiché dans la sortie **route:list** ; cependant, vous pouvez demander à **Laravel** d'afficher le middleware de route en ajoutant l'option **-v** à la commande :

```
php artisan route:list -v
```

- Vous pouvez également demander à **Laravel** de n'afficher que les routes commençant par un **URI** donné :

```
php artisan route:list --path=api
```

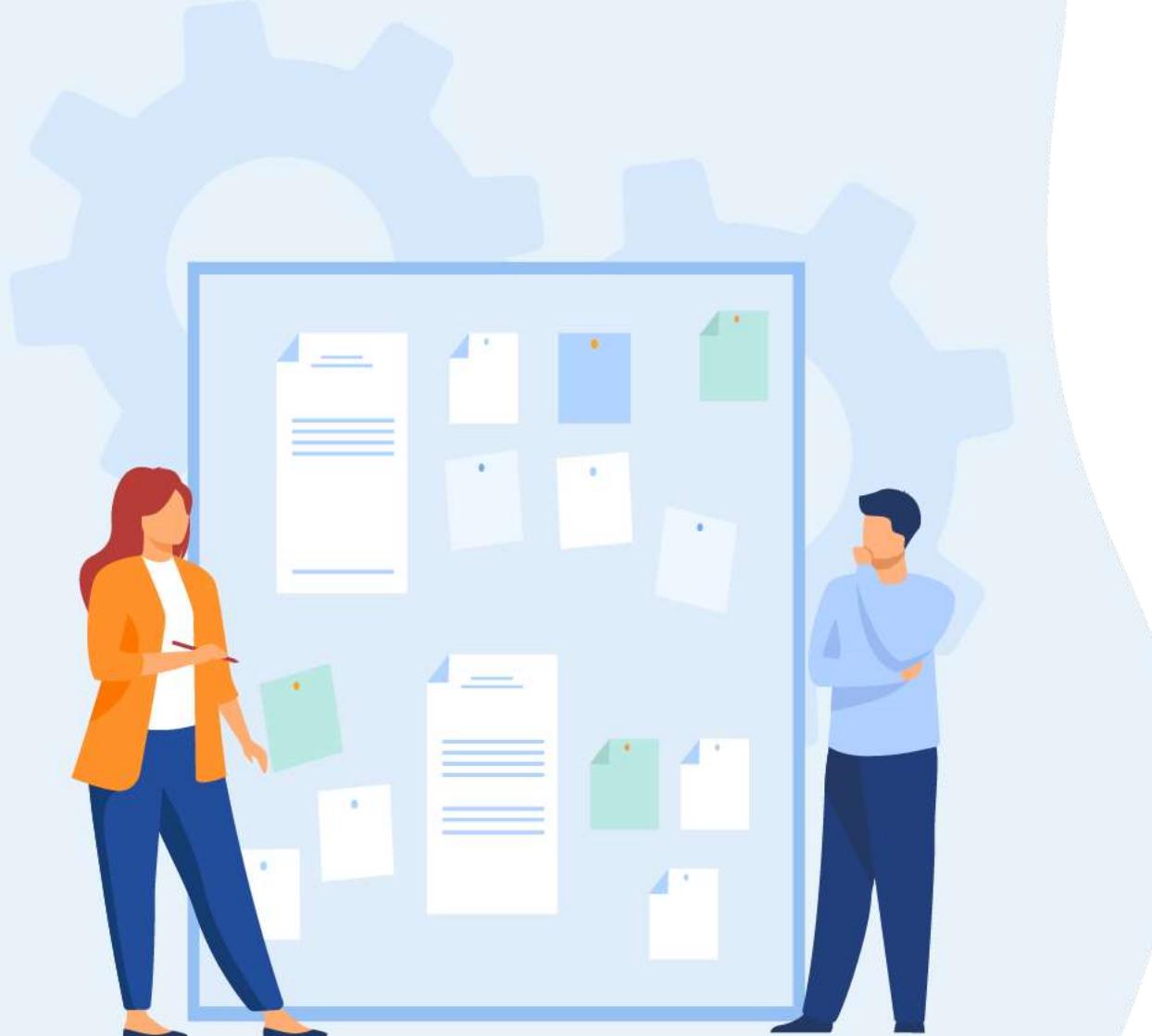
- De plus, vous pouvez demander à **Laravel** de masquer toutes les routes définies par des packages tiers en fournissant l'option **--except-vendor** lors de l'exécution de la commande **route:list** :

```
php artisan route:list --except-vendor
```

La liste des routes

- De même, vous pouvez également demander à **Laravel** de n'afficher que les routes définies par des packages tiers en fournissant l'option **--only-vendor** lors de l'exécution de la commande **route:list** :

```
php artisan route:list
```



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. **Paramètres de routage**
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

01 - Gestion du routage

Paramètres de routage



Paramètres requis

- Parfois, vous devrez capturer des segments de l'URI dans votre route. Par exemple, vous devrez peut-être capturer l'ID d'un utilisateur à partir de l'URL. Vous pouvez le faire en définissant les paramètres de route :

```
Route::get('/user/{id}', function ($id) {  
    return 'User '.$id;  
});
```

- Vous pouvez définir autant de paramètres de route que requis par votre route :

```
Route::get('/posts/{post}/comments/{comment}', function ($postId, $commentId) {  
    //  
});
```

- Les paramètres de route sont toujours entourés d'accolades {} et doivent être composés de caractères alphabétiques. Les traits de soulignement (_) sont également acceptables dans les noms de paramètre de route. Les paramètres de route sont injectés dans les rappels/contrôleurs de route en fonction de leur ordre - les noms des arguments de rappel/contrôleur de route n'ont pas d'importance.

01 - Gestion du routage

Paramètres de routage



Paramètres et injection de dépendance

- Si votre route a des dépendances que vous aimeriez que le conteneur de service **Laravel** injecte automatiquement dans le rappel de votre route, vous devez lister vos paramètres de route après vos dépendances.

```
use Illuminate\Http\Request;

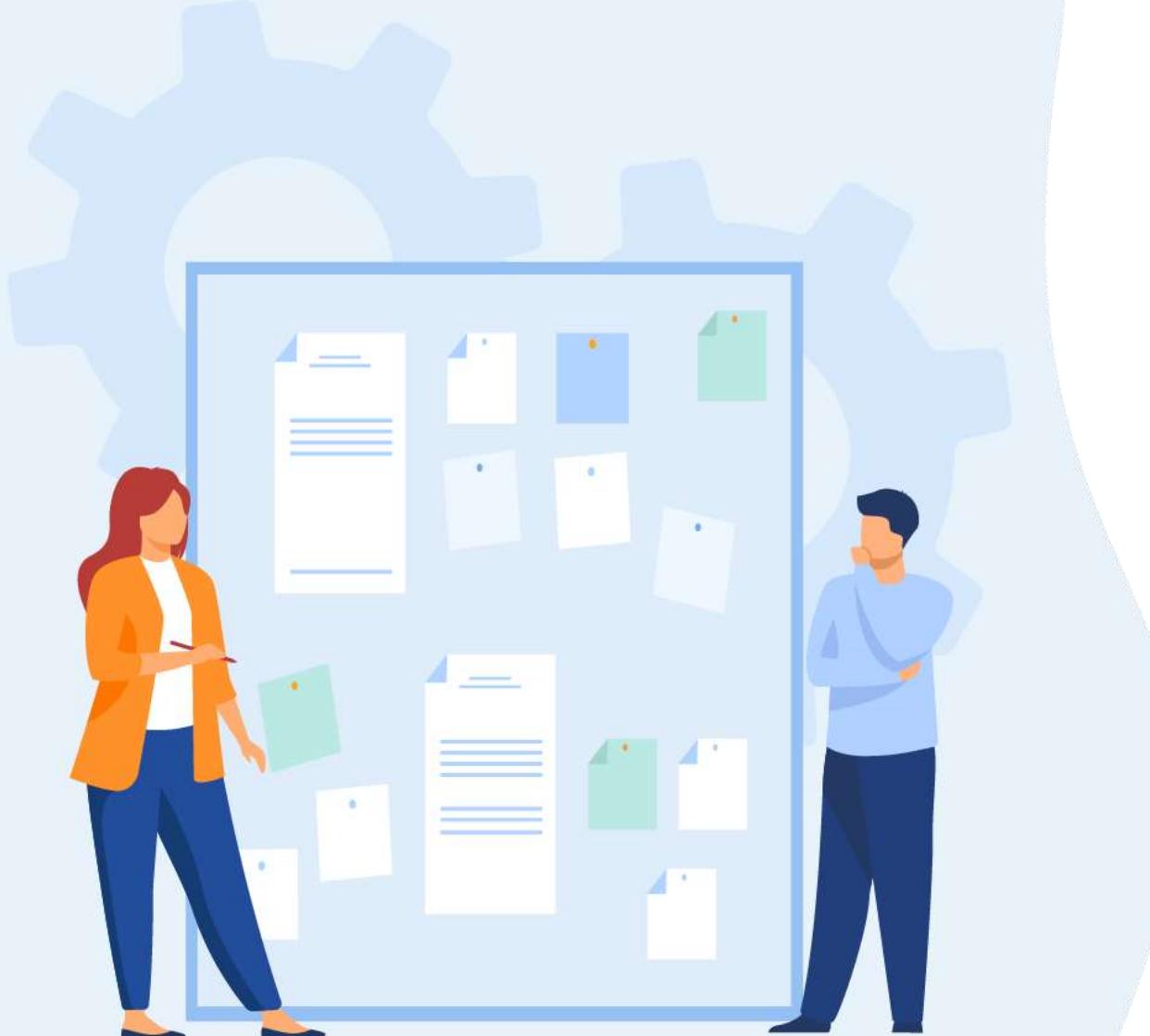
Route::get('/user/{id}', function (Request $request, $id) {
    return 'User '.$id;
});
```

Paramètres facultatifs

- Parfois, vous devrez peut-être spécifier un paramètre de route qui n'est pas toujours présent dans l'URI. Vous pouvez le faire en plaçant une ? marque après le nom du paramètre. Assurez-vous de donner une valeur par défaut à la variable correspondante de la route :

```
Route::get('/user/{name?}', function ($name = null) {  
    return $name;  
});
```

```
Route::get('/user/{name?}', function ($name = Ahmed') {  
    return $name;  
});
```



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. **Routes nommées**
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

01 - Gestion du routage

Routes nommées



- Les routes nommées permettent la génération pratique d'URL ou de redirections pour des routes spécifiques. Vous pouvez spécifier un nom pour une route en enchaînant la méthode **name** sur la définition de route :

```
Route::get('/user/profile', function () {  
    //  
    })->name('profile');
```

- Vous pouvez également spécifier des noms de route pour les actions du contrôleur :

```
Route::get(  
    '/user/profile',  
    [UserProfileController::class, 'show']  
    )->name('profile');
```

01 - Gestion du routage

Routes nommées



Génération d'URL vers des routes nommées

- Une fois que vous avez attribué un nom à une route donnée, vous pouvez utiliser le nom de la route lors de la génération d'URL ou de redirections via les fonctions de **Laravel route** et d'assistance **redirect** :

```
// Generating URLs...
$url = route('profile');
```

```
// Generating Redirects...
return redirect()->route('profile');
```

- Si la route nommée définit des paramètres, vous pouvez passer les paramètres comme deuxième argument à la fonction **route**. Les paramètres donnés seront automatiquement insérés dans l'**URL** générée dans leurs positions correctes :

```
Route::get('/user/{id}/profile', function ($id) {
    //
})->name('profile');

$url = route('profile', ['id' => 1]);
```

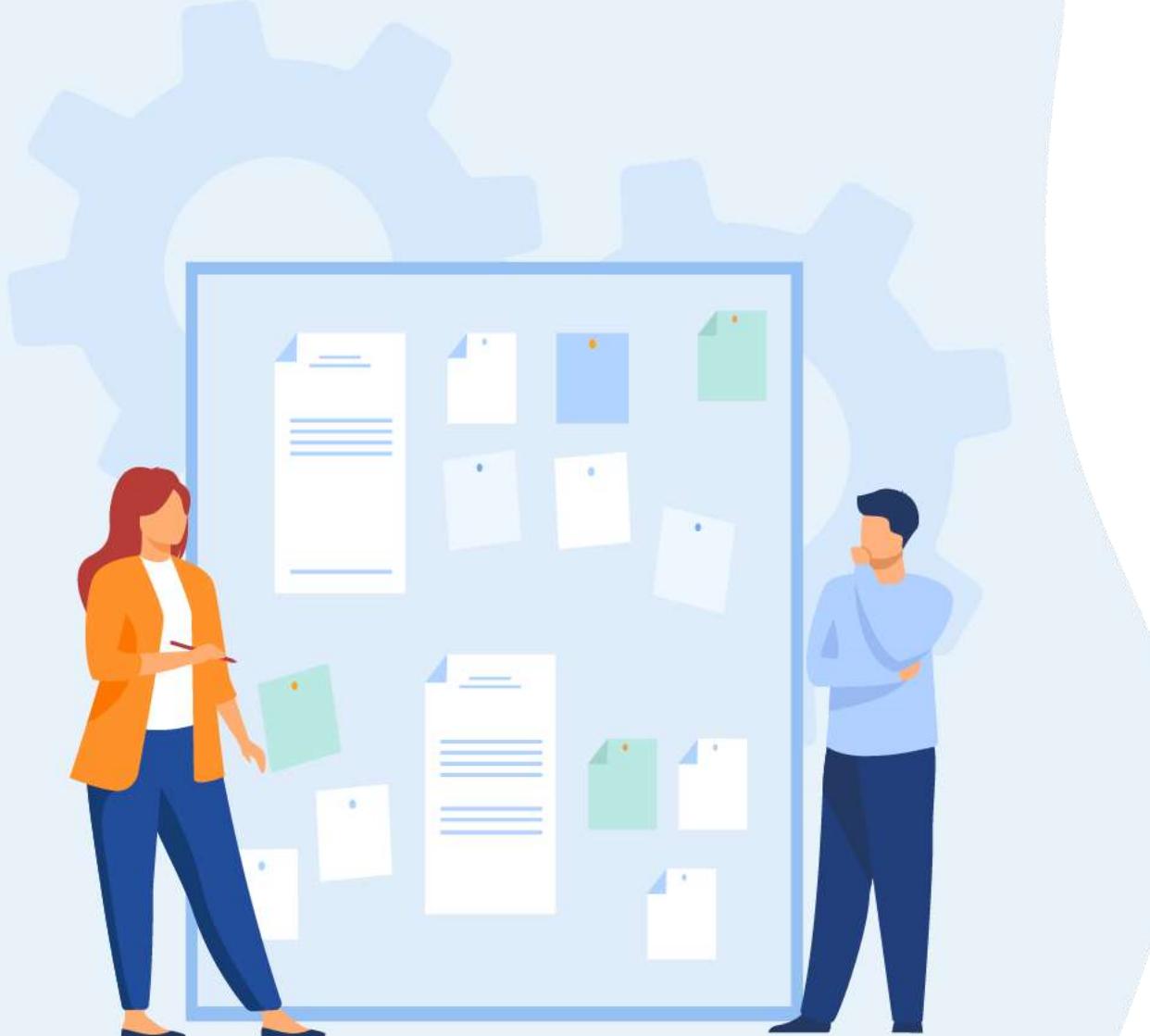
01 - Gestion du routage

Routes nommées



- Si vous transmettez des paramètres supplémentaires dans le tableau, ces paires **clé/valeur** seront automatiquement ajoutées à la chaîne de requête de l'URL générée :

```
Route::get('/user/{id}/profile', function ($id) {  
    //  
    })->name('profile');  
  
$url = route('profile', ['id' => 1, 'photos' => 'yes']);  
  
// /user/1/profile?photos=yes
```



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. **Groupe de routage**
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

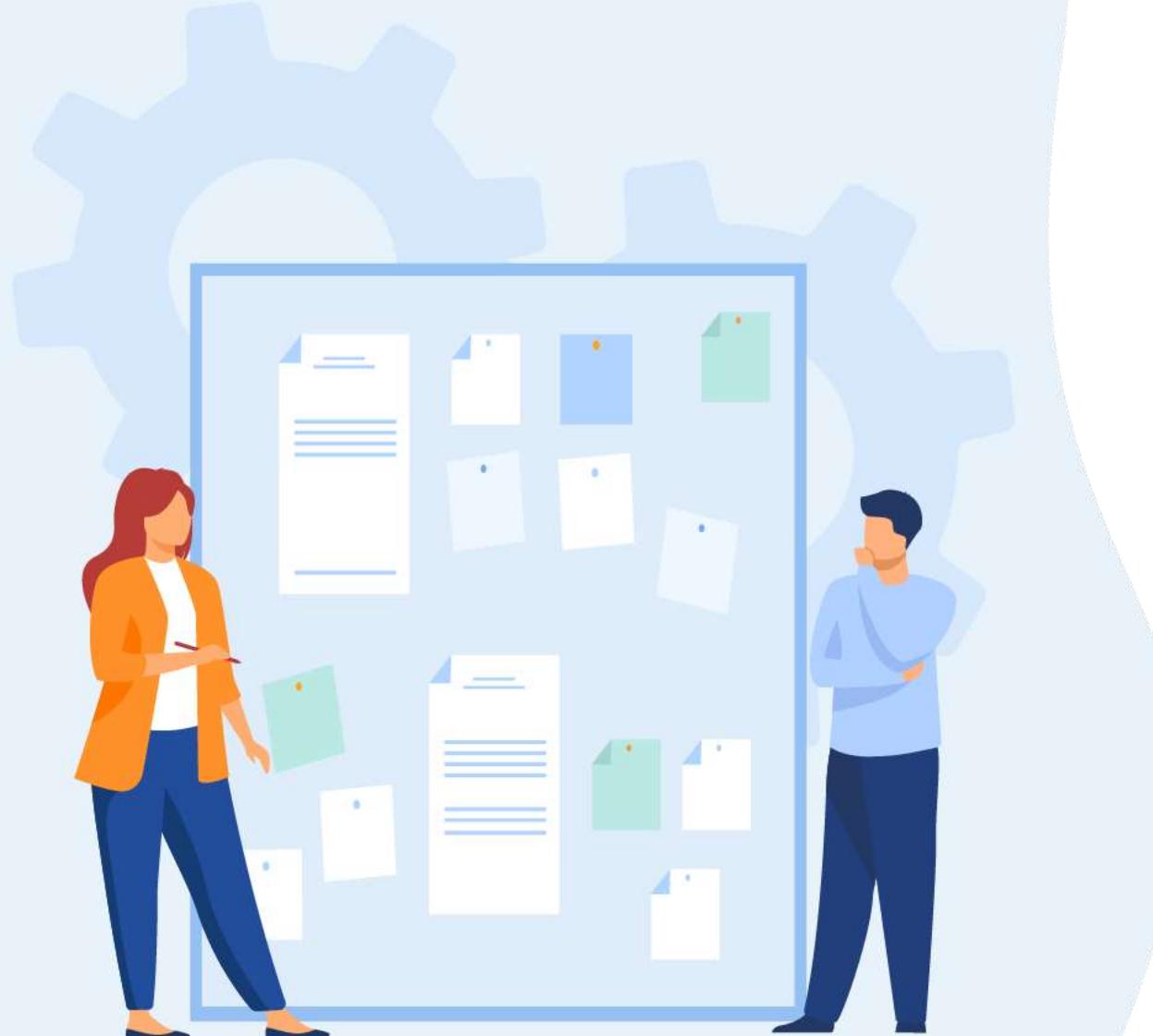
01 - Gestion du routage

Groupes de routage



- Les routes peuvent être regroupées pour éviter la répétition du code.
- Les groupes de routage vous permettent de partager des attributs de routage, tels que le middleware, sur un grand nombre de routages sans avoir à définir ces attributs sur chaque routage individuel.
- Disons que tous les **URI** avec un préfixe de **/admin** utilisent un certain middleware appelé **admin** et qu'ils vivent tous dans l'espace de noms **App\Http\Controllers\Admin** .
 - Une manière propre de représenter cela en utilisant des groupes de routage est la suivante:

```
Route::group([
    'namespace' => 'Admin',
    'middleware' => 'admin',
    'prefix' => 'admin'
], function () {
    // something.dev/admin
    // 'App\Http\Controllers\Admin\IndexController'
    // Uses admin middleware
    Route::get('/', ['uses' => 'IndexController@index']);
    // something.dev/admin/logs
    // 'App\Http\Controllers\Admin\LogsController'
    // Uses admin middleware
    Route::get('/logs', ['uses' => 'LogsController@index']);
});
```



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
- 5. Liaisons de modèles de routes**
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

Liaison Implicite

- **Laravel** résout automatiquement les modèles Eloquent définis dans les itinéraires ou les actions de contrôleur dont les noms de variable correspondent à un nom de segment de route. Par exemple:

```
Route::get('api/users/{user}', function (App\User $user) {  
    return $user->email;  
});
```

- Dans cet exemple, comme la variable utilisateur eloquent `$` définie sur la route correspond au segment `{utilisateur}` dans l'URI de la route, **Laravel** injectera automatiquement l'instance de modèle dont l'ID correspond à la valeur correspondante de l'URI de la demande. Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse **HTTP 404** sera automatiquement générée.
- Si le nom de la table du modèle est composé de plusieurs mots, pour que la liaison de modèle implicite fonctionne, la variable d'entrée doit être en minuscules;
- Par exemple, si l'utilisateur peut effectuer une action quelconque et que nous voulons accéder à cette action, l'itinéraire sera le suivant:

```
Route::get('api/useractions/{useraction}', function (App\UserAction $useraction) {  
    return $useraction->description;  
});
```

Liaison Explicite

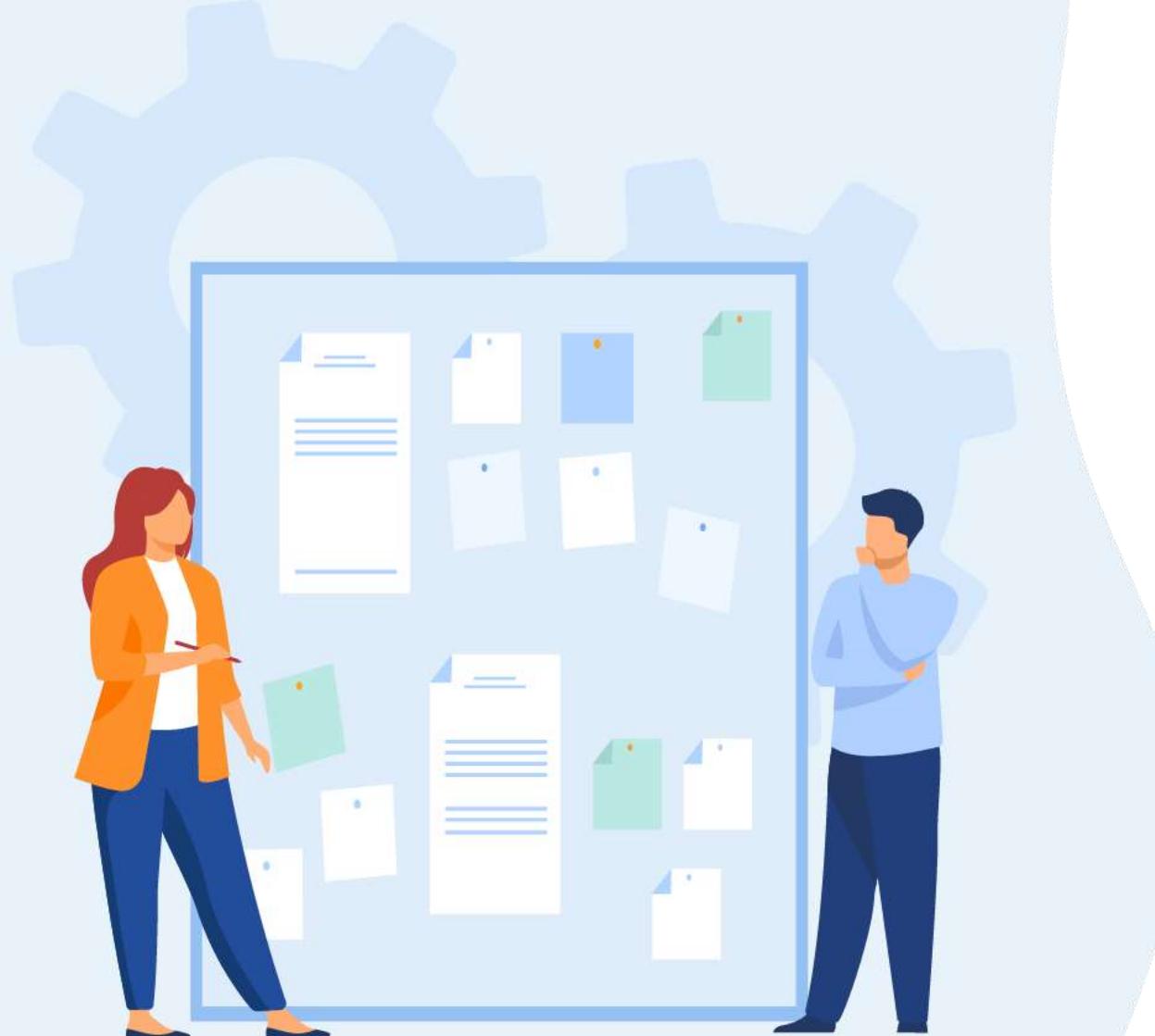
- Pour enregistrer une liaison explicite, utilisez la méthode du **modèle** du routeur pour spécifier la classe d'un paramètre donné. Vous devez définir vos liaisons de modèle explicites dans la méthode de démarrage de la classe **RouteServiceProvider**

```
public function boot()
{
    parent::boot();
    Route::model('user', App\User::class);
}
```

- Ensuite, nous pouvons définir une route contenant le paramètre {utilisateur}.

```
$router->get('profile/{user}', function(App\User $user) {
});
```

- Comme nous avons lié tous **{user}** paramètres **{user}** au modèle **App\User** , une instance **User** sera injectée dans la route. Ainsi, par exemple, une demande de profile/1 injectera l'instance d'utilisateur de la base de données dont l'identifiant est 1 .
- Si une instance de modèle correspondante n'est pas trouvée dans la base de données, une réponse HTTP 404 sera automatiquement générée



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. **Routes de repli**
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

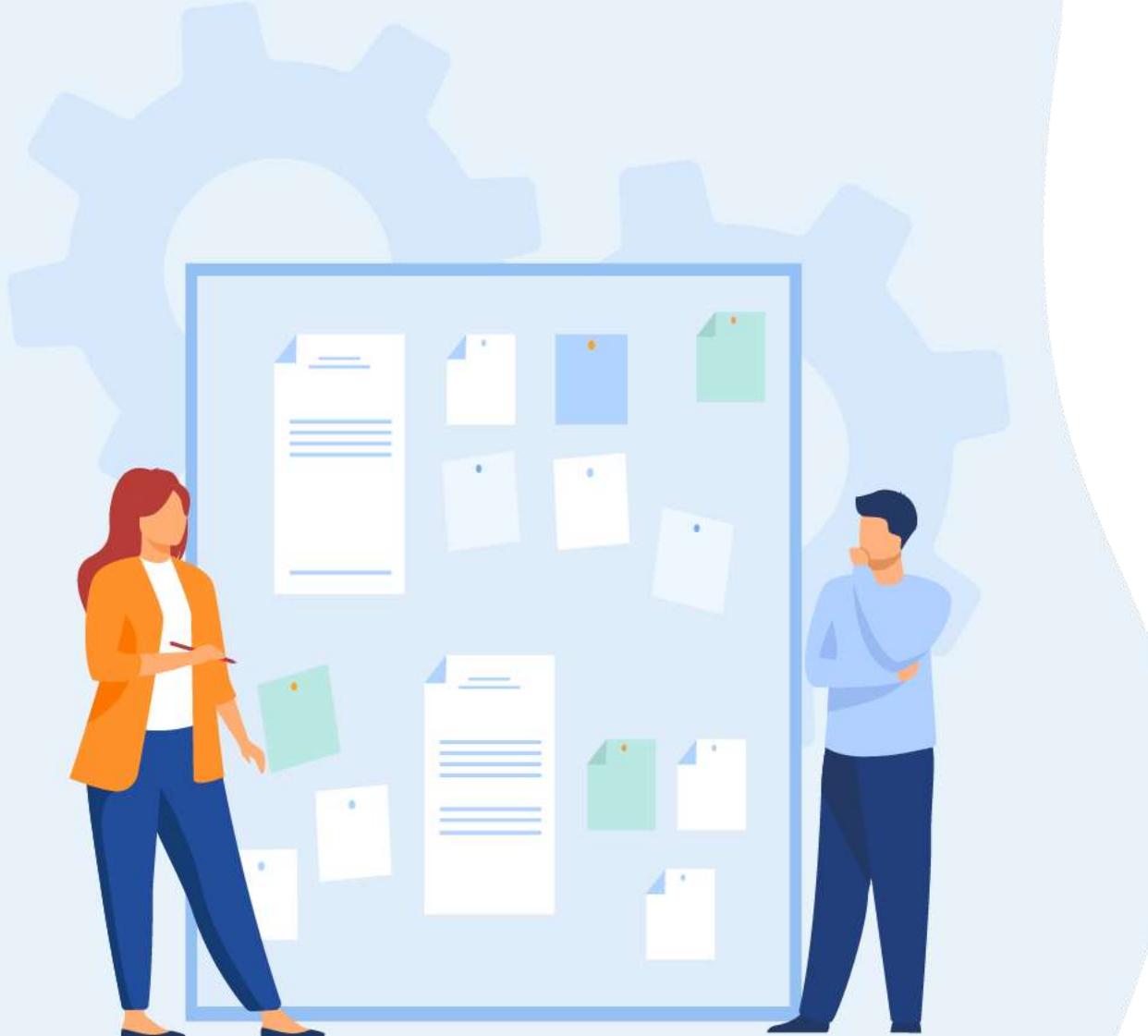
01 - Gestion du routage

Routes de repli



- En utilisant la méthode **Route::fallback**, vous pouvez définir une route qui sera exécutée lorsqu'aucune autre route ne correspond à la demande entrante.
- En règle générale, les requêtes non gérées afficheront automatiquement une page "**404**" via le gestionnaire d'exceptions de votre application. Cependant, comme vous définissez généralement la **fallbackroute** dans votre fichier **routes/web.php**, tous les middleware du groupe middleware **web** s'appliqueront à la route. Vous êtes libre d'ajouter un middleware supplémentaire à cette route si nécessaire :

```
Route::fallback(function () {  
    //  
});
```



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. **(spoofing) de la méthode du formulaire**
8. Accès à la route courante
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

01 - Gestion du routage

Usurpation (spoofing) de la méthode du formulaire

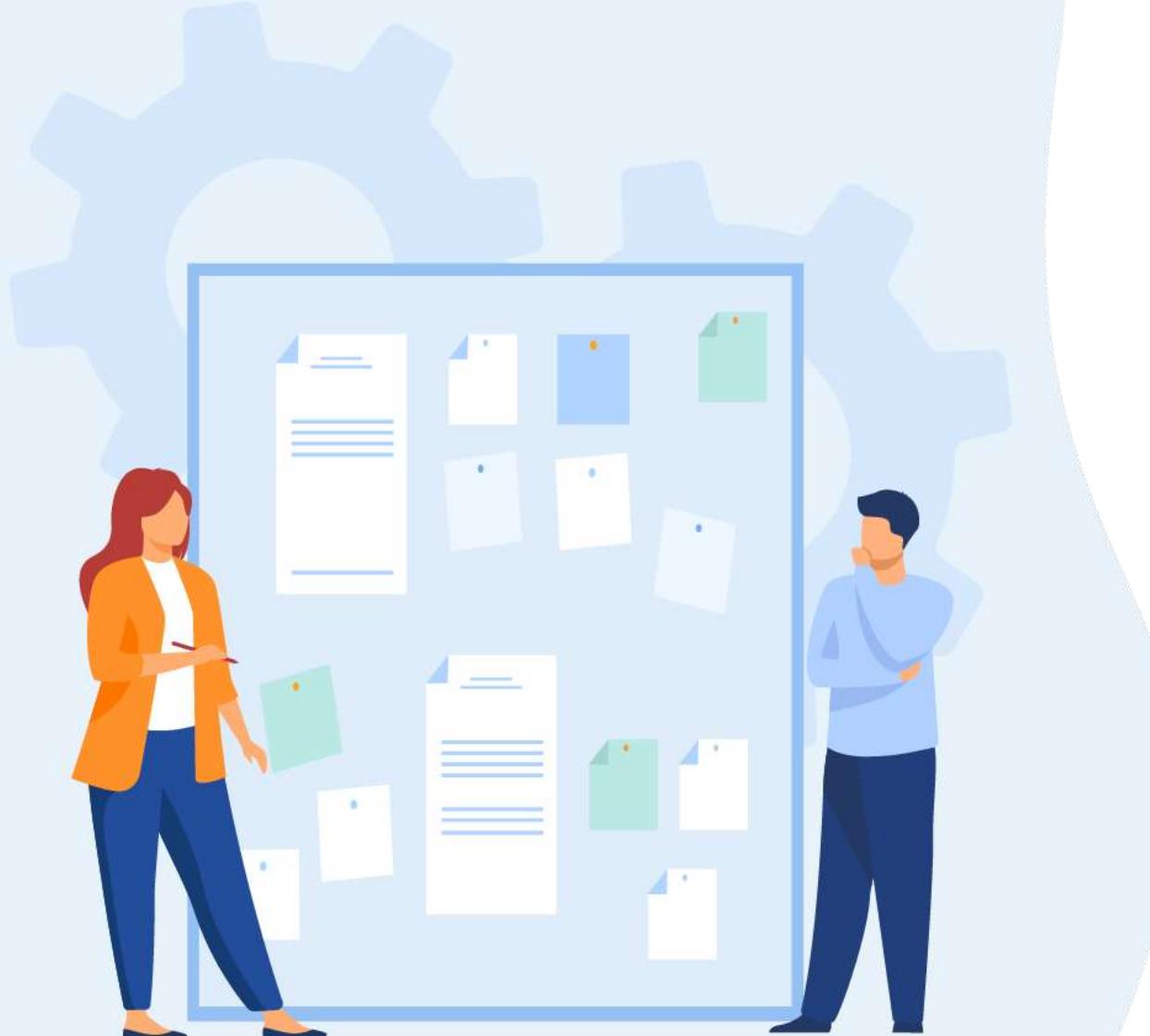


- Les formulaires HTML ne prennent pas en charge les actions **PUT**, **PATCH** ou **DELETE**. Ainsi, lors de la définition de routes appelées **PUT**, **PATCH** ou **DELETE** à partir d'un formulaire HTML, vous devrez ajouter un champ masqué **_method** au formulaire. La valeur envoyée avec le champ **_method** sera utilisée comme méthode de requête HTTP :

```
<form action="/example" method="POST">
    <input type="hidden" name="_method" value="PUT">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
</form>
```

- Pour plus de commodité, vous pouvez utiliser la **directive Blade @method** pour générer le champ **_method** de saisie :

```
<form action="/example" method="POST">
    @method('PUT')
    @csrf
</form>
```



CHAPITRE 1

Gestion du routage

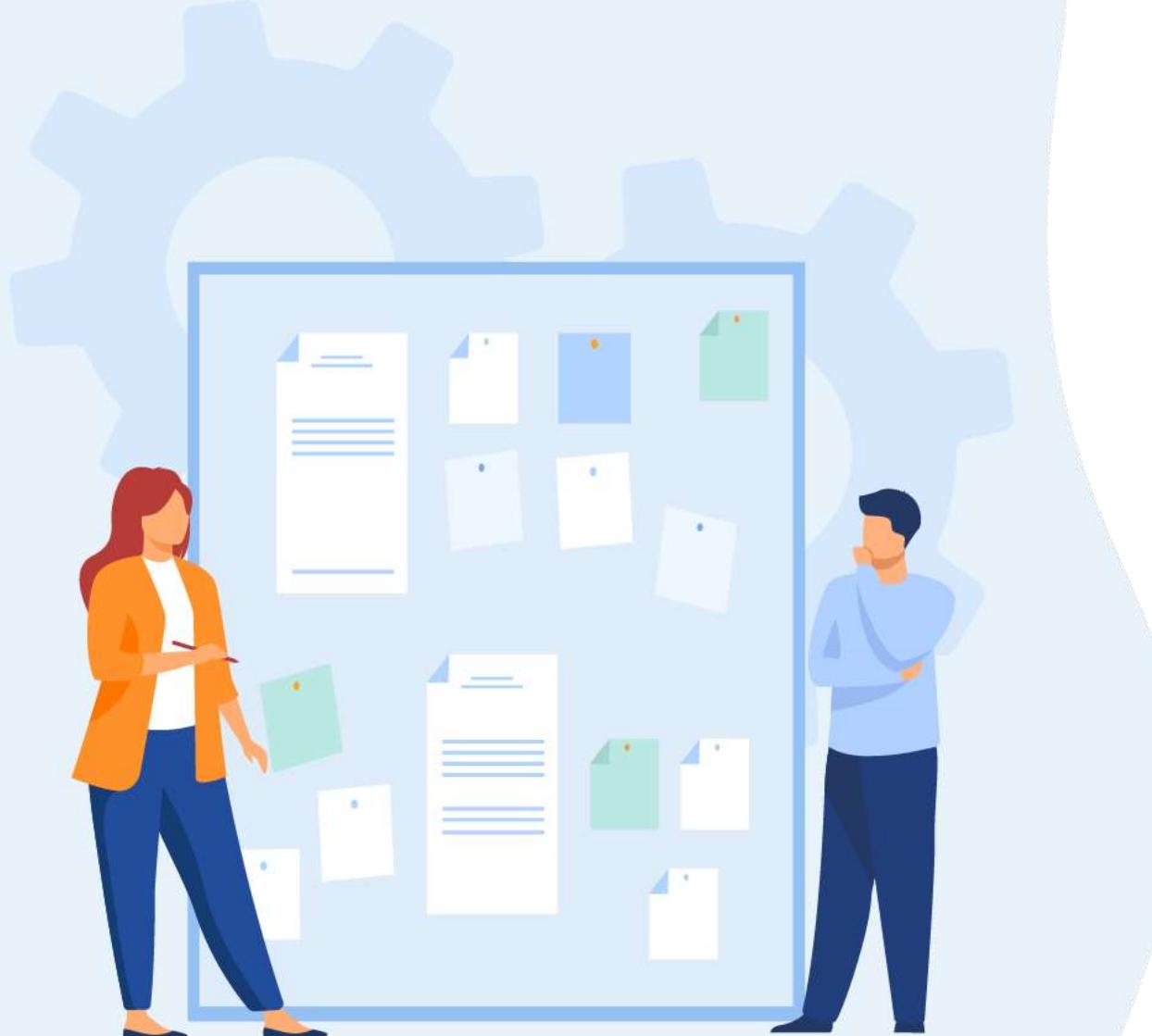
1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
- 8. Accès à la route courante**
9. Cross Origin Resource Sharing (CORS)
10. Mise en cache des routes

01 - Gestion du routage

Accès à la route courante

- Vous pouvez utiliser les méthodes **current**, **currentRouteName** et **currentRouteAction** sur la façade Route pour accéder aux informations sur la route traitant la requête entrante :

```
use Illuminate\Support\Facades\Route;  
  
$route = Route::current(); // Illuminate\Routing\Route  
$name = Route::currentRouteName(); // string  
$action = Route::currentRouteAction(); // string
```



CHAPITRE 1

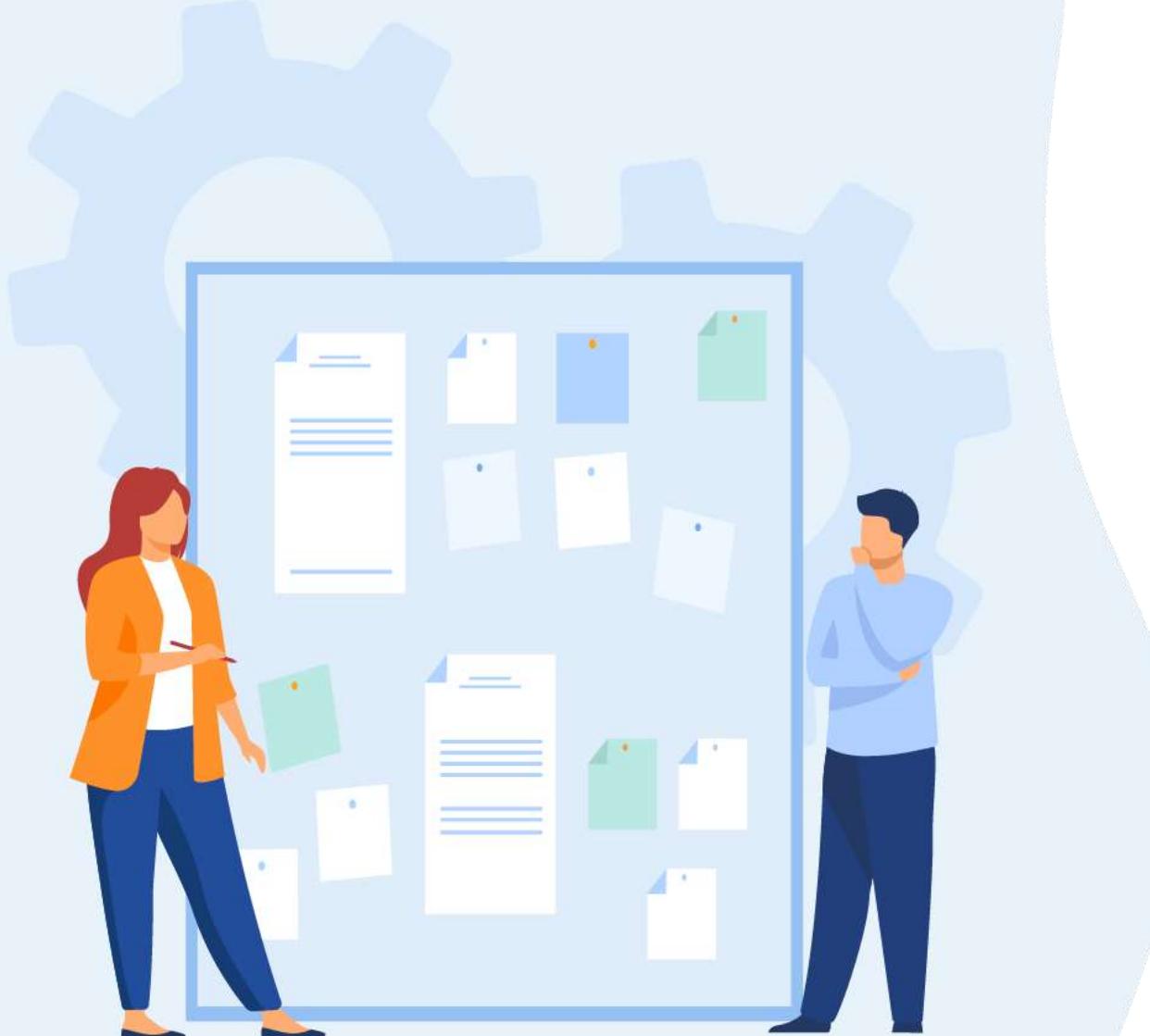
Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. **Partage des ressources d'origine croisée(CORS)**
10. Mise en cache des routes

01 - Gestion du routage

Partage des ressources d'origine croisée (CORS)

- Laravel peut répondre automatiquement aux requêtes **HTTP CORS OPTIONS** avec des valeurs que vous configurez. Tous les paramètres **CORS** peuvent être configurés dans le fichier `config/cors.php` de configuration de votre application. Les requêtes **OPTIONS** seront automatiquement traitées par le middleware `HandleCors` qui est inclus par défaut dans votre pile middleware globale. Votre pile middleware globale se trouve dans le noyau HTTP de votre application (`App\Http\Kernel`).



CHAPITRE 1

Gestion du routage

1. Routage de base (redirection, affichage, liste de routes)
2. Paramètres de routage
3. Routes nommées
4. Groupe de routage
5. Liaisons de modèles de routes
6. Routes de repli
7. (spoofing) de la méthode du formulaire
8. Accès à la route courante
9. **Partage des ressources d'origine croisée(CORS)**
10. Mise en cache des routes

01 - Gestion du routage

Mise en cache des routes



- Lors du déploiement de votre application en production, vous devez tirer parti du cache de routage de Laravel. L'utilisation du cache de route réduira considérablement le temps nécessaire pour enregistrer toutes les routes de votre application.
- Pour générer un cache de route, exécutez la commande Artisan **route:cache** :

```
php artisan route:cache
```

- Après avoir exécuté cette commande, votre fichier de routes en cache sera chargé à chaque requête. N'oubliez pas que si vous ajoutez de nouvelles routes, vous devrez générer un nouveau cache de routes. Pour cette raison, vous ne devez exécuter la **route:cache** commande que pendant le déploiement de votre projet.
- Vous pouvez utiliser la **route:clear** commande pour vider le cache de route :

```
php artisan route:clear
```

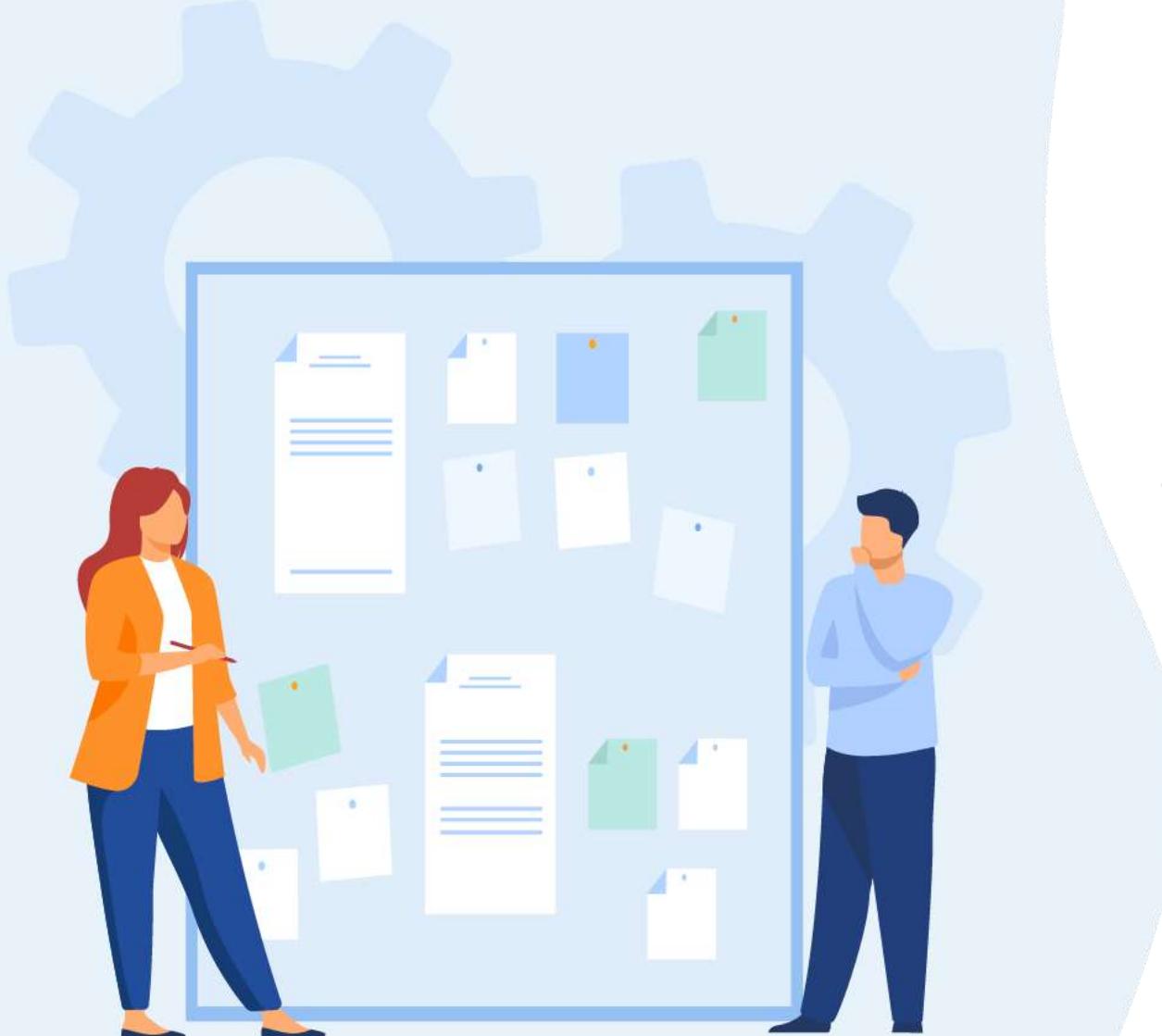


CHAPITRE 2

Utilisation des Middleware

Ce que vous allez apprendre dans ce chapitre :

- Définition
- Enregistrement
- Paramétrage
- Terminate



CHAPITRE 2

Utilisation des Middleware

1. Définition
2. Enregistrement
3. Paramétrage
4. Terminate

02 - Utilisation des Middleware

Définition

- Le **middleware** fournit un mécanisme pratique pour inspecter et filtrer les requêtes **HTTP** entrant dans votre application. Par exemple, **Laravel** inclut un **middleware** qui vérifie que l'utilisateur de votre application est authentifié. Si l'utilisateur n'est pas authentifié, le **middleware** redirigera l'utilisateur vers l'écran de connexion de votre application. Cependant, si l'utilisateur est authentifié, le **middleware** permettra à la demande de continuer plus loin dans l'application.
- Pour créer un nouveau middleware, utilisez la commande Artisan **make:middleware** :

```
php artisan make:middleware EnsureTokenIsValid
```

02 - Utilisation des Middleware

Définition



- Cette commande placera une nouvelle classe **EnsureTokenIsValid** dans votre répertoire **app/Http/Middleware**. Dans ce **middleware**, nous n'autoriserons l'accès à la route que si l'entrée **token** fournie correspond à une valeur spécifiée. Sinon, nous redirigerons les utilisateurs vers l'URI **home** :

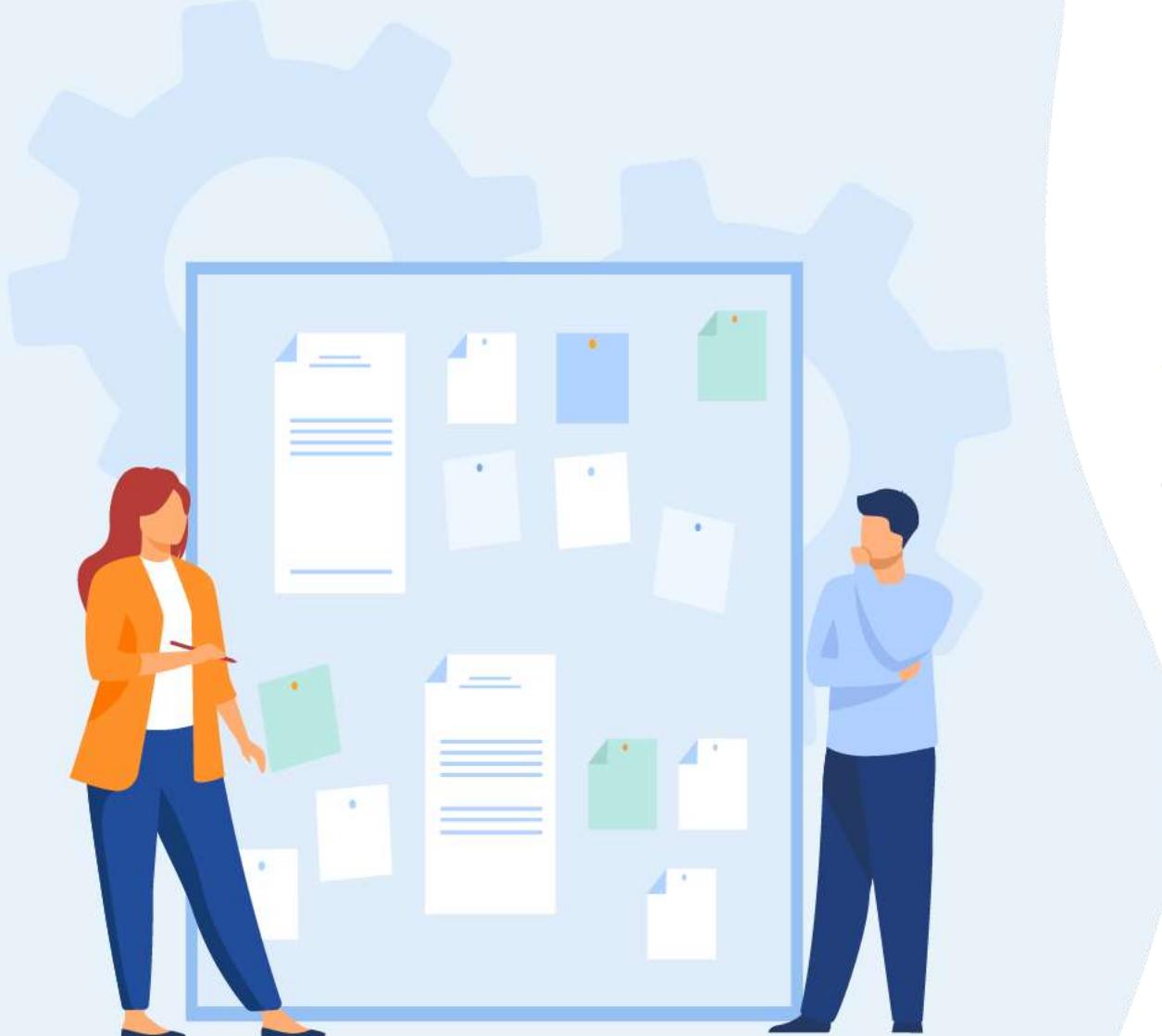
```
<?php
namespace App\Http\Middleware;
use Closure;
class EnsureTokenIsValid
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if ($request->input('token') !== 'my-secret-token') {
            return redirect('home');
        }
        return $next($request);
    }
}
```

[5] <https://code.visualstudio.com/>

02 - Utilisation des Middleware

Définition

- Comme vous pouvez le voir, si le donné **token** ne correspond pas à notre jeton secret, le **middleware** renverra une redirection HTTP au client ; sinon, la demande sera transmise plus loin dans l'application. Pour transmettre la demande plus profondément dans l'application (permettant au middleware de "passer"), vous devez appeler le rappel **\$next** avec le **\$request**.
- Il est préférable d'envisager le middleware comme une série de "couches" que les requêtes HTTP doivent traverser avant d'atteindre votre application. Chaque couche peut examiner la demande et même la rejeter entièrement.



CHAPITRE 2

Utilisation des Middleware

1. Définition
2. **Enregistrement**
3. Paramétrage
4. Terminate

02 - Utilisation des Middleware

Enregistrement

Middleware Globale

- Si vous souhaitez qu'un **middleware** s'exécute lors de chaque requête HTTP adressée à votre application, répertoriez la classe middleware dans la propriété **\$middleware** de votre classe **app/Http/Kernel.php**.

Affectation de middleware aux routes

- Si vous souhaitez affecter un **middleware** à des routes spécifiques, vous devez d'abord affecter au **middleware** une **clé** dans le fichier **app\Http\Kernel.php** de votre application. Par défaut, la propriété **\$routeMiddleware** de cette classe contient des entrées pour le **middleware** inclus avec **Laravel**. Vous pouvez ajouter votre propre **middleware** à cette liste et lui attribuer une clé de votre choix :

```
// Within App\Http\Kernel class...

protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
];
```

02 - Utilisation des Middleware

Enregistrement



- Une fois le middleware défini dans le noyau HTTP, vous pouvez utiliser la méthode **middleware** pour affecter le middleware à une route :

```
Route::get('/profile', function () {  
    //  
})->middleware('auth');
```

- Vous pouvez affecter plusieurs middlewares à la route en transmettant un tableau de noms de middlewares à la méthode **middleware** :

```
Route::get('/', function () {  
    //  
})->middleware(['first', 'second']);
```

- Lors de l'attribution du middleware, vous pouvez également transmettre le nom complet de la classe :

```
use App\Http\Middleware\EnsureTokenIsValid;
```

```
Route::get('/profile', function () {  
    //  
})->middleware(EnsureTokenIsValid::class);
```

L'exclusion de middleware

- Lors de l'attribution d'un middleware à un groupe de routes, vous devrez peut-être parfois empêcher le middleware de s'appliquer sur une route individuelle au sein du groupe. Vous pouvez accomplir cela en utilisant la méthode `withoutMiddleware` :

```
use App\Http\Middleware\EnsureTokenIsValid;

Route::middleware([EnsureTokenIsValid::class])->group(function () {
    Route::get('/', function () {
        //
    });

    Route::get('/profile', function () {
        //
    })->withoutMiddleware([EnsureTokenIsValid::class]);
});
```

02 - Utilisation des Middleware

Enregistrement



- Vous pouvez également exclure un ensemble donné de middleware d'un groupe entier de définitions de route :

```
use App\Http\Middleware\EnsureTokenIsValid;
```

```
Route::withoutMiddleware([EnsureTokenIsValid::class])->group(function () {  
    Route::get('/profile', function () {  
        //  
    });  
});
```

- La méthode **withoutMiddleware** ne peut supprimer que le middleware de route et ne s'applique pas au middleware global .

Groupes de middleware

- Parfois, vous souhaiterez peut-être regrouper plusieurs middlewares sous une seule clé pour faciliter leur affectation aux routes. Vous pouvez accomplir cela en utilisant la propriété `$middlewareGroups` de votre noyau HTTP.
- Prêt à l'emploi, Laravel est livré avec des groupes **middlewares Web** et **API** qui contiennent des **middlewares** communs que vous voudrez peut-être appliquer à vos routes Web et API. N'oubliez pas que ces groupes de middleware sont automatiquement appliqués par le fournisseur de services `App\Providers\RouteServiceProvider` de votre application aux routes dans vos fichiers de route **Web** et **API** correspondants :

```
/**  
 * The application's route middleware groups.  
 *  
 * @var array  
 */  
protected $middlewareGroups = [  
    'web' => [  
        \App\Http\Middleware\EncryptCookies::class,  
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,  
        \Illuminate\Session\Middleware\StartSession::class,  
        \Illuminate\View\Middleware\ShareErrorsFromSession::
```

```
        class,  
        \App\Http\Middleware\VerifyCsrfToken::class,  
        \Illuminate\Routing\Middleware\SubstituteBindings::class,  
    ],  
  
    'api' => [  
        'throttle:api',  
        \Illuminate\Routing\Middleware\SubstituteBindings::class,  
    ],
```

02 - Utilisation des Middleware

Enregistrement



- Les groupes middlewares peuvent être affectés aux routes et aux actions du contrôleur en utilisant la même syntaxe que les middlewares individuels. Encore une fois, les groupes de middlewares facilitent l'affectation simultanée de plusieurs middlewares à une route :

```
Route::get('/', function () {
    //
})->middleware('web');

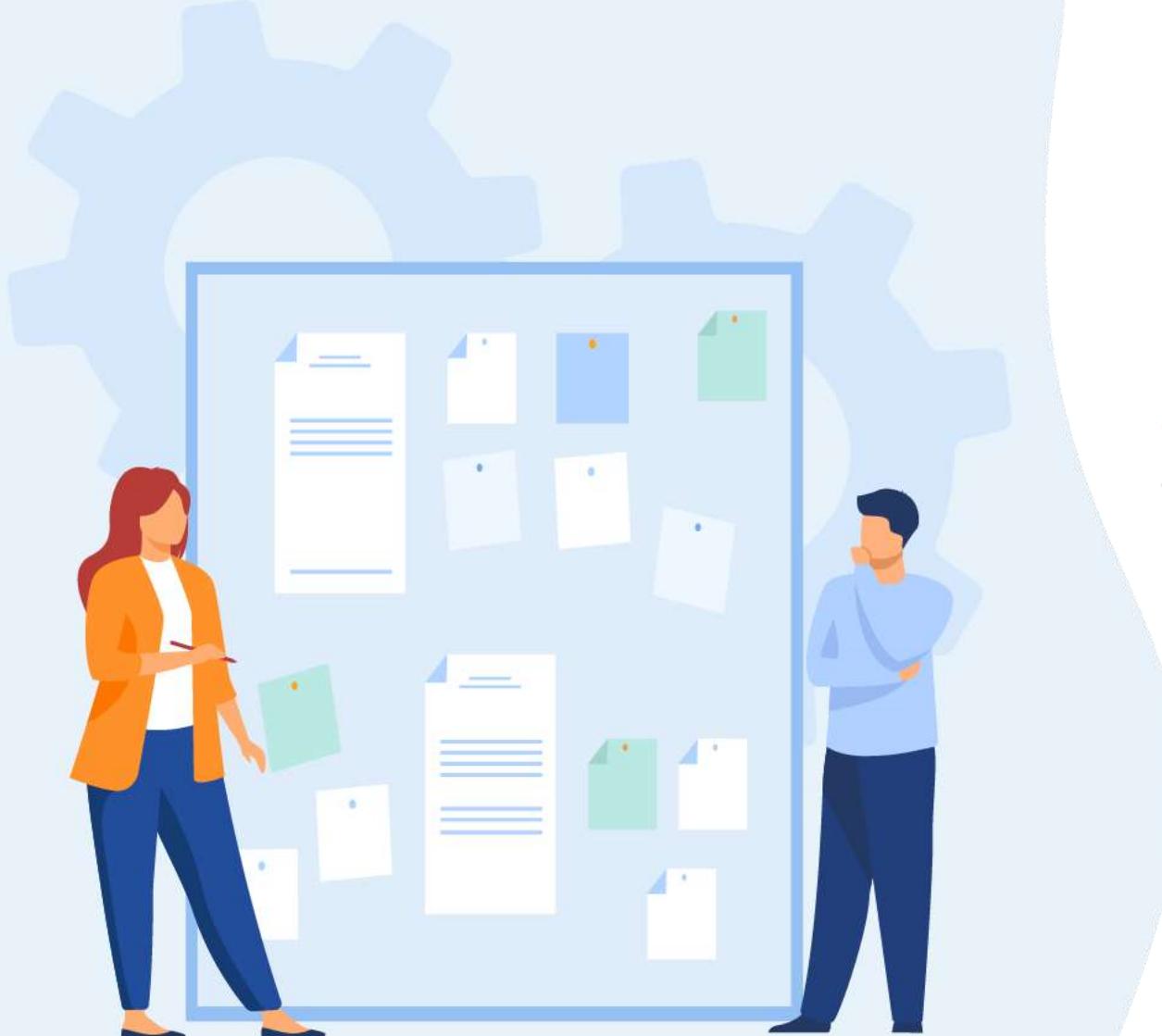
Route::middleware(['web'])->group(function () {
    //
});
```

Le tri de middleware

- Rarement, vous pouvez avoir besoin que votre middleware s'exécute dans un ordre spécifique, mais sans contrôler leur ordre lorsqu'ils sont affectés à la route. Dans ce cas, vous pouvez spécifier votre priorité middleware en utilisant la propriété **\$middlewarePriority** de votre fichier **app/Http/Kernel.php**. Cette propriété peut ne pas exister dans votre noyau HTTP par défaut. S'il n'existe pas, vous pouvez copier sa définition par défaut ci-dessous :

```
/**  
 * The priority-sorted list of middleware.  
 *  
 * This forces non-global middleware to always be in the  
 given order.  
 *  
 * @var string[]  
 */  
protected $middlewarePriority = [  
    \Illuminate\Cookie\Middleware\EncryptCookies::class,  
    \Illuminate\Session\Middleware\StartSession::class,  
    \Illuminate\View\Middleware\ShareErrorsFromSession::class,  
    \Illuminate\Contracts\Auth\Middleware\AuthenticatesR
```

```
equests::class,  
\Illuminate\Routing\Middleware\ThrottleRequests::class  
,  
\Illuminate\Routing\Middleware\ThrottleRequestsWithRedis::class,  
\Illuminate\Contracts\Session\Middleware\AuthenticateSessions::class,  
\Illuminate\Routing\Middleware\SubstituteBindings::class,  
\Illuminate\Auth\Middleware\Authorize::class,  
];
```



CHAPITRE 2

Utilisation des Middleware

1. Définition
2. Enregistrement
- 3. Paramétrage**
4. Terminate

02 - Utilisation des Middleware

Paramétrage



- Le middleware peut également recevoir des paramètres supplémentaires. Par exemple, si votre application doit vérifier que l'utilisateur authentifié a un "rôle" donné avant d'effectuer une action donnée, vous pouvez créer un middleware **EnsureUserHasRole** qui reçoit un nom de rôle comme argument supplémentaire.
- Des paramètres middleware supplémentaires seront transmis au middleware après l'argument **\$next** :

```
<?php  
  
namespace App\Http\Middleware;  
  
use Closure;  
  
class EnsureUserHasRole  
{  
    /**  
     * Handle the incoming request.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @param \Closure $next  
     * @param string $role  
     * @return mixed  
     */  
    public function handle($request, Closure $next, $role)  
    {  
        if (! $request->user()->hasRole($role)) {  
            // Redirect...  
        }  
  
        return $next($request);  
    }  
}
```

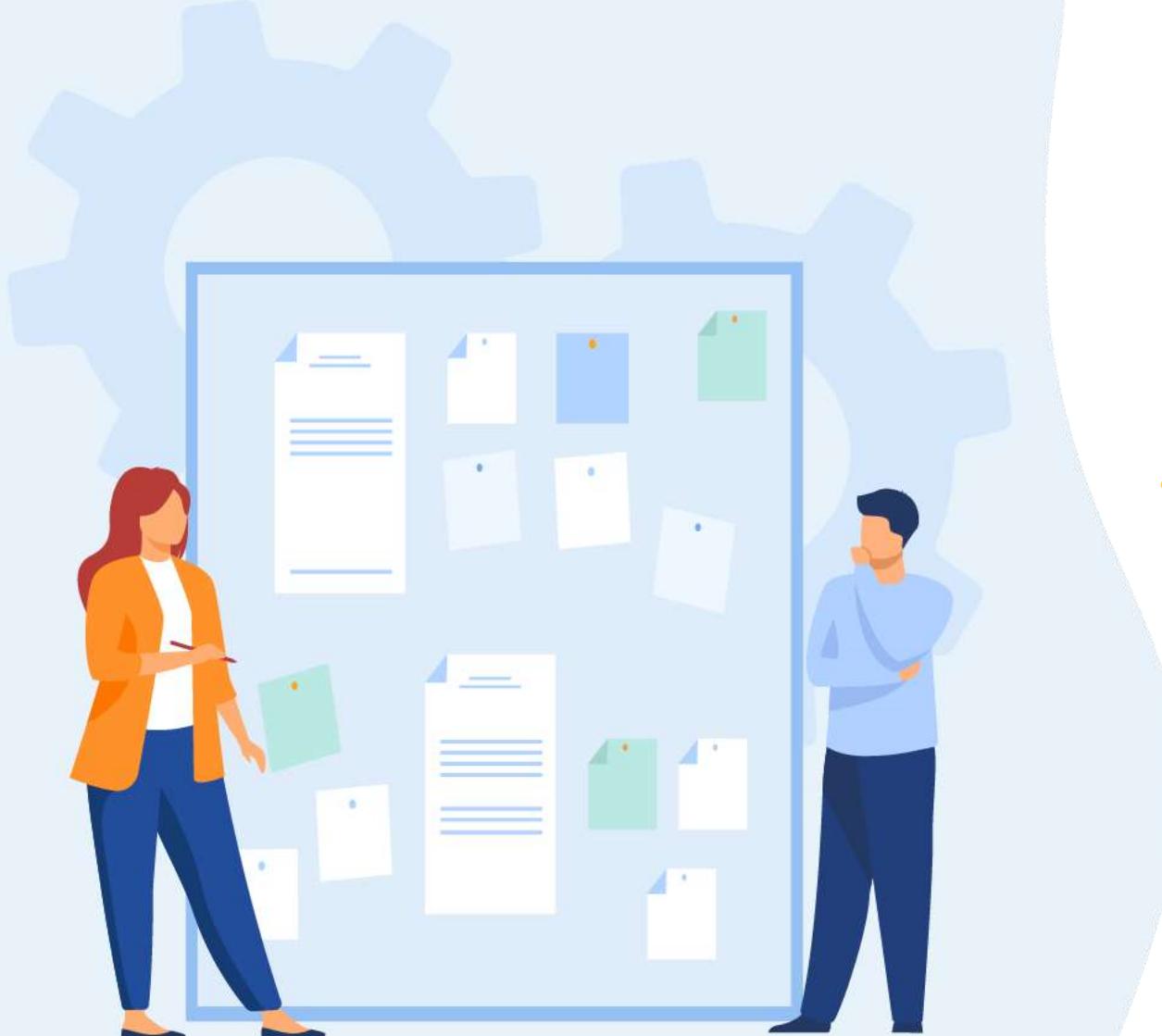
02 - Utilisation des Middleware

Paramétrage



- Les paramètres du middleware peuvent être spécifiés lors de la définition de la route en séparant le nom du middleware et les paramètres par un : . Plusieurs paramètres doivent être délimités par des **virgules** :

```
Route::put('/post/{id}', function ($id) {  
    //  
})->middleware('role:editor');
```



CHAPITRE 2

Utilisation des Middleware

1. Définition
2. Enregistrement
3. Paramétrage
4. **Terminate**

02 - Utilisation des Middleware

Terminate



- Parfois, un middleware peut avoir besoin de faire du travail après que la réponse HTTP a été envoyée au navigateur. Si vous définissez une méthode **terminate** sur votre middleware et que votre serveur web utilise **FastCGI**, la méthode **terminate** sera automatiquement appelée après l'envoi de la réponse au navigateur :

```
<?php  
  
namespace Illuminate\Session\Middleware;  
  
use Closure;  
  
class TerminatingMiddleware  
{  
    /**  
     * Handle an incoming request.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @param \Closure $next  
     * @return mixed  
     */  
    public function handle($request, Closure $next)  
    {  
        {  
            return $next($request);  
        }  
  
        /**  
         * Handle tasks after the response has been sent to the browser.  
         *  
         * @param \Illuminate\Http\Request $request  
         * @param \Illuminate\Http\Response $response  
         * @return void  
         */  
        public function terminate($request, $response)  
        {  
            // ...  
        }  
    }  
}
```

02 - Utilisation des Middleware

Terminate



- La méthode **terminate** doit recevoir à la fois la requête et la réponse. Une fois que vous avez défini un middleware terminable, vous devez l'ajouter à la liste des routes ou middleware global dans le fichier **app/Http/Kernel.php**.
- Lors de l'appel de la méthode **terminate** sur votre middleware, Laravel résoudra une nouvelle instance du middleware à partir du service container. Si vous souhaitez utiliser la même instance de middleware lorsque les méthodes **handle** et **terminate** sont appelées, enregistrez le middleware auprès du conteneur à l'aide de la méthode du conteneur **singleton**. Généralement, cela devrait être fait dans la méthode **register** de votre **AppServiceProvider**:

```
use App\Http\Middleware\TerminatingMiddleware;

/**
 * Register any application services.
 *
 * @return void
 */
public function register()
{
    $this->app->singleton(TerminatingMiddleware::class);
}
```

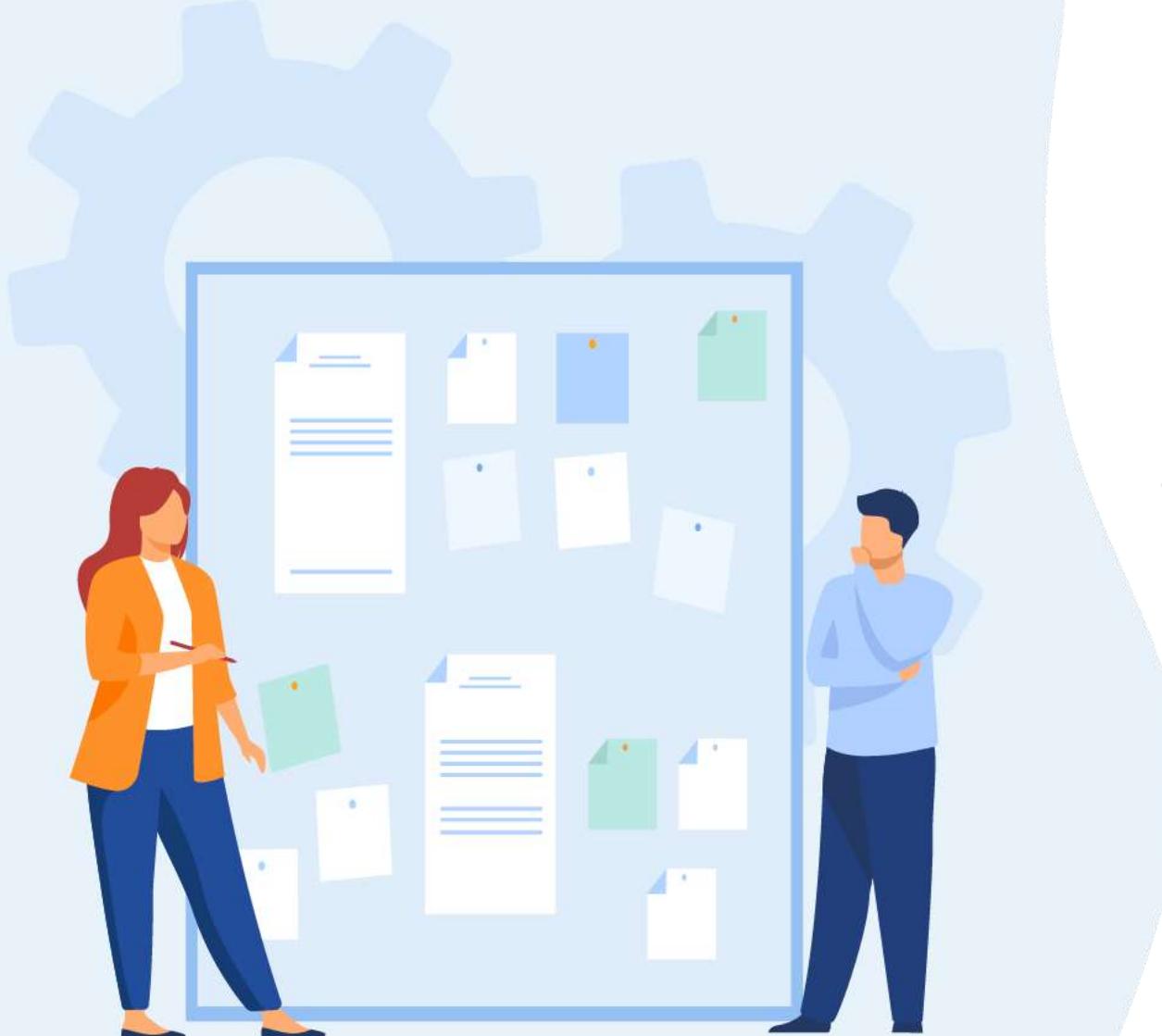


CHAPITRE 3

Protection CSRF

Ce que vous allez apprendre dans ce chapitre :

- Introduction
- Prévenir les requêtes CSRF
- Jeton X-CSRF-Token
- Jeton X-XSRF-Token



CHAPITRE 3

Protection CSRF

1. **Introduction**
2. Prévenir les requêtes CSRF
3. Jeton X-CSRF-Token
4. Jeton X-XSRF-Token

03 - Protection CSRF

Introduction



- Les falsifications de requêtes intersites sont un type d'exploit malveillant par lequel des commandes non autorisées sont exécutées au nom d'un utilisateur authentifié. Heureusement, Laravel facilite la protection de votre application contre les attaques de cross-site request forgery (CSRF)

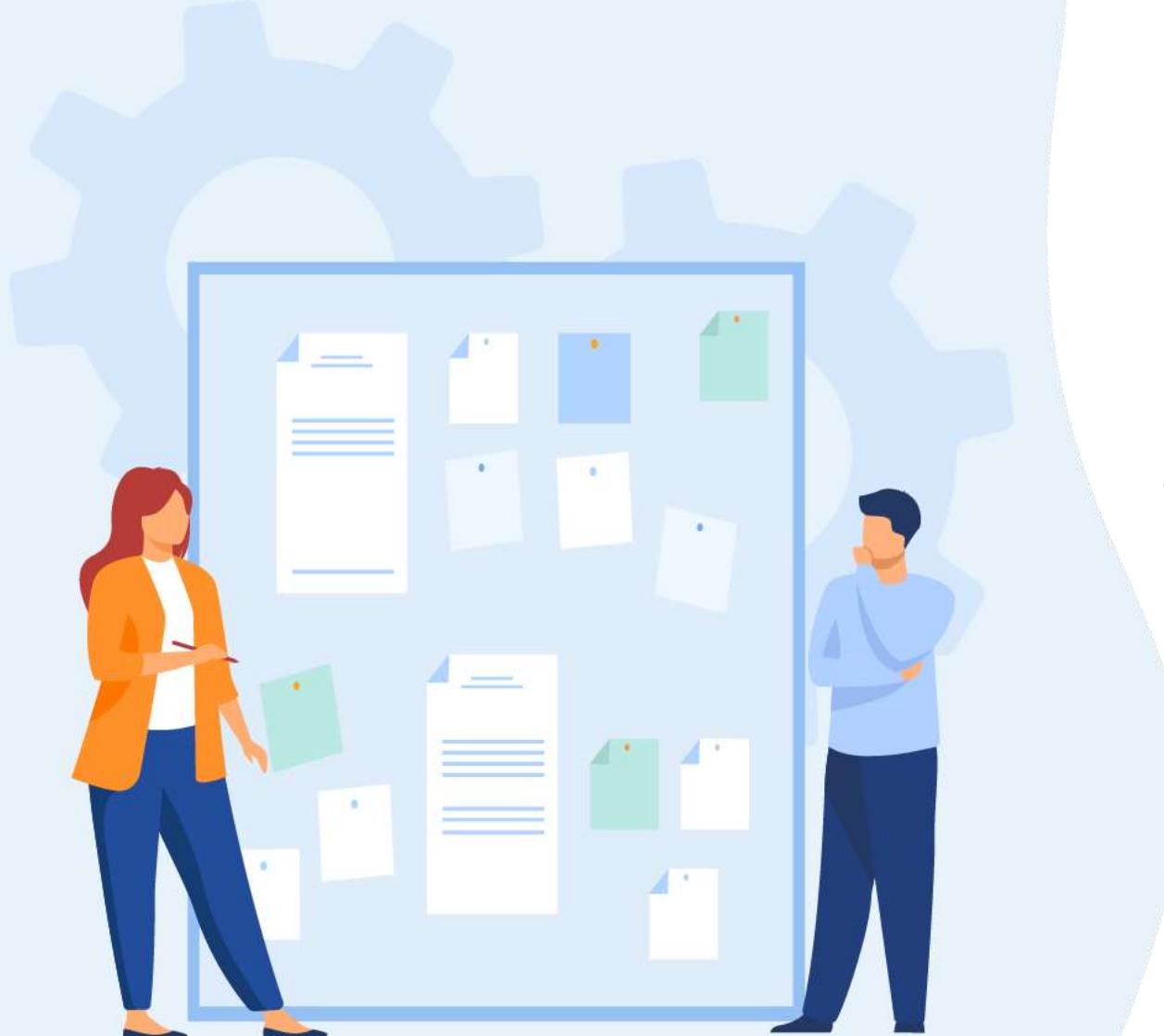
Une explication de la vulnérabilité

- Si vous n'êtes pas familier avec les falsifications de requêtes intersites, discutons d'un exemple de la façon dont cette vulnérabilité peut être exploitée. Imaginez que votre application dispose d'une route `/user/email` qui accepte une demande **POST** de modification de l'adresse e-mail de l'utilisateur authentifié. Très probablement, cette route s'attend à ce qu'un champ de saisie `email` contienne l'adresse e-mail que l'utilisateur souhaite commencer à utiliser.
- Sans protection **CSRF**, un site Web malveillant pourrait créer un formulaire HTML qui pointe vers la route `/user/email` de votre application et soumet la propre adresse e-mail de l'utilisateur malveillant :

```
<form action="https://your-application.com/user/email" method="POST">
    <input type="email" value="malicious-email@example.com">
</form>

<script>
    document.forms[0].submit();
</script>
```

- Si le site Web malveillant soumet automatiquement le formulaire lorsque la page est chargée, l'utilisateur malveillant n'a qu'à attirer un utilisateur sans méfiance de votre application pour qu'il visite son site Web et son adresse e-mail sera modifiée dans votre application.
- Pour éviter cette vulnérabilité, nous devons inspecter chaque demande entrante **POST**, **PUT**, **PATCH** ou **DELETE** pour une valeur de session secrète à laquelle l'application malveillante ne peut pas accéder.



CHAPITRE 3

Protection CSRF

1. Introduction
2. **Prévenir les requêtes CSRF**
3. Jeton X-CSRF-Token
4. Jeton X-XSRF-Token

03 - Protection CSRF

Prévenir les requêtes CSRF



- Laravel génère automatiquement un "**token**" CSRF pour chaque session utilisateur active gérée par l'application. Ce jeton est utilisé pour vérifier que l'utilisateur authentifié est la personne qui fait réellement les requêtes à l'application. Étant donné que ce jeton est stocké dans la session de l'utilisateur et change à chaque régénération de la session, une application malveillante ne peut pas y accéder.
- Le jeton CSRF de la session en cours est accessible via la session de la requête ou via la fonction d'assistance **csrf_token** :

```
use Illuminate\Http\Request;

Route::get('/token', function (Request $request) {
    $token = $request->session()->token();

    $token = csrf_token();

    // ...
});
```

03 - Protection CSRF

Prévenir les requêtes CSRF



- Chaque fois que vous définissez un formulaire HTML "POST", "PUT", "PATCH" ou "DELETE" dans votre application, vous devez inclure un champ `_token` CSRF caché dans le formulaire afin que le middleware de protection **CSRF** puisse valider la demande. Pour plus de commodité, vous pouvez utiliser la directive `@csrf` pour générer le champ de saisie du jeton caché :

```
<form method="POST" action="/profile">
    @csrf

    <!-- Equivalent to... -->
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
</form>
```

- Le middleware `App\Http\Middleware\VerifyCsrfToken`, qui est inclus dans le groupe middleware par défaut `web`, vérifiera automatiquement que le jeton dans l'entrée de la requête correspond au jeton stocké dans la session. Lorsque ces deux jetons correspondent, nous savons que l'utilisateur authentifié est celui qui lance la requête.

Exclure les URI de la protection CSRF

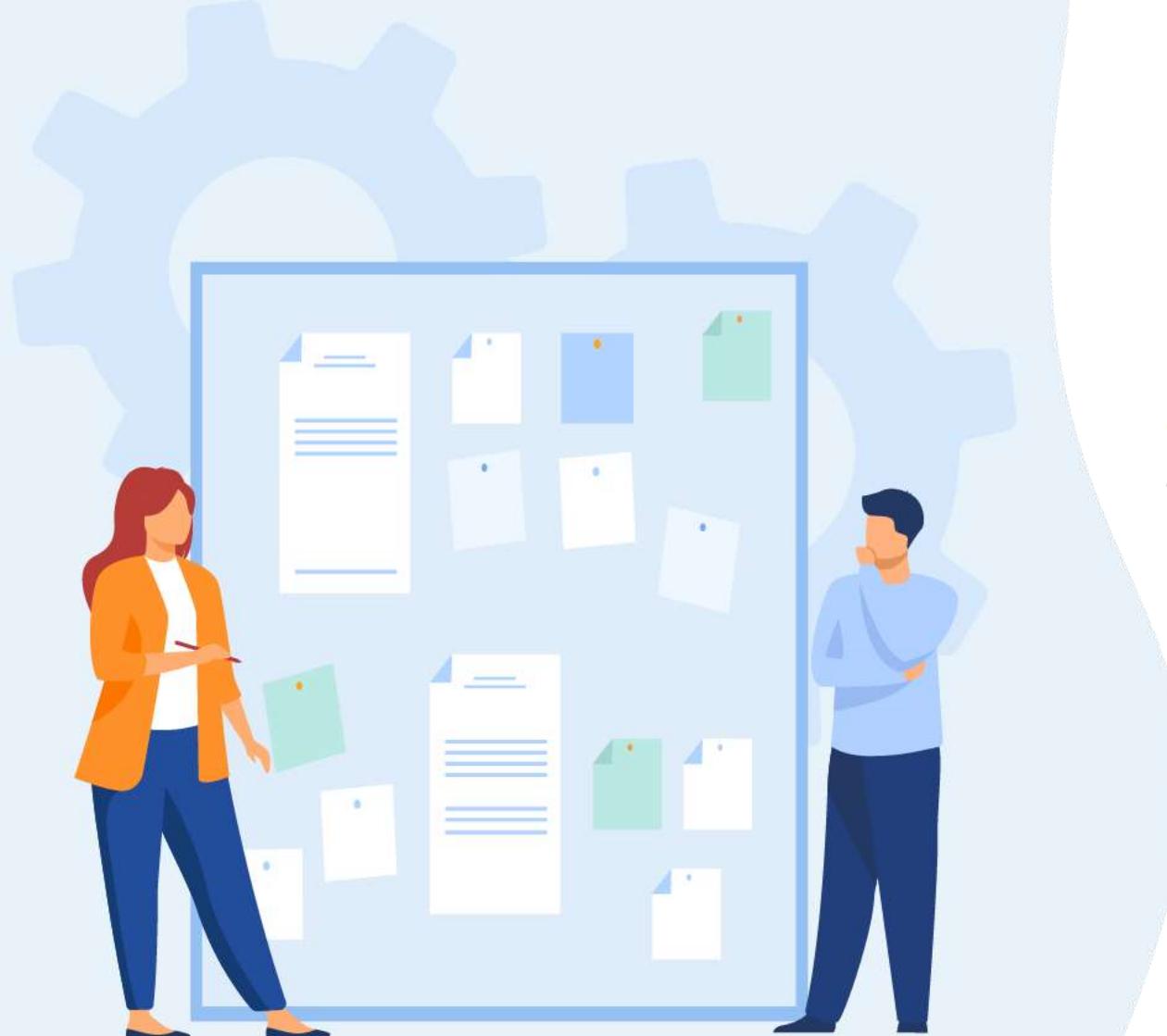
- Parfois, vous souhaiterez peut-être exclure un ensemble d'URI de la protection CSRF. Par exemple, si vous utilisez [Stripe](#) pour traiter les paiements et utilisez leur système de **webhook**, vous devrez exclure votre route de gestionnaire de **webhook Stripe** de la protection **CSRF** car **Stripe** ne saura pas quel jeton **CSRF** envoyer à vos routes.
- En règle générale, vous devez placer ces types de routes en dehors du groupe **web** de middleware **App\Providers\RouteServiceProvider** qui s'applique à toutes les routes du fichier **routes/web.php**. Cependant, vous pouvez également exclure les routes en ajoutant leurs URI à la propriété **\$except** du middleware **VerifyCsrfToken** :

```
<?php

namespace App\Http\Middleware;

use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken
as Middleware;

class VerifyCsrfToken extends Middleware
{
    /**
     * The URIs that should be excluded from CSRF
     verification.
     *
     * @var array
     */
    protected $except = [
        'stripe/*',
        'http://example.com/foo/bar',
        'http://example.com/foo/\*',
    ];
}
```



CHAPITRE 3

Protection CSRF

1. Introduction
2. Prévenir les requêtes CSRF
- 3. Jeton X-CSRF-Token**
4. Jeton X-XSRF-Token

03 - Protection CSRF

Jeton X-CSRF-TOKEN



- En plus de vérifier le jeton CSRF en tant que paramètre POST, le [middleware App\Http\Middleware\VerifyCsrfToken](#) vérifiera également l'en-tête de la requête **X-CSRF-TOKEN**. Vous pouvez, par exemple, stocker le jeton dans une balise HTML **meta** :

```
<meta name="csrf-token" content="{{ csrf_token() }}>
```

- Ensuite, vous pouvez demander à une bibliothèque comme [jQuery](#) d'ajouter automatiquement le jeton à tous les en-têtes de requête. Cela fournit une protection **CSRF** simple et pratique pour vos applications basées sur [AJAX](#) utilisant la technologie JavaScript héritée :

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

03 - Protection CSRF

Jeton X-XSRF-Token



- Laravel stocke le jeton **CSRF** actuel dans un cookie crypté **XSRF-TOKEN** qui est inclus avec chaque réponse générée par le framework. Vous pouvez utiliser la valeur du cookie pour définir l'en-tête de la demande **X-XSRF-TOKEN**.
- Ce cookie est principalement envoyé pour la commodité des développeurs, car certains frameworks et bibliothèques JavaScript, comme Angular et Axios, placent automatiquement sa valeur dans l'en-tête des requêtes- X-XSRF-TOKEN.

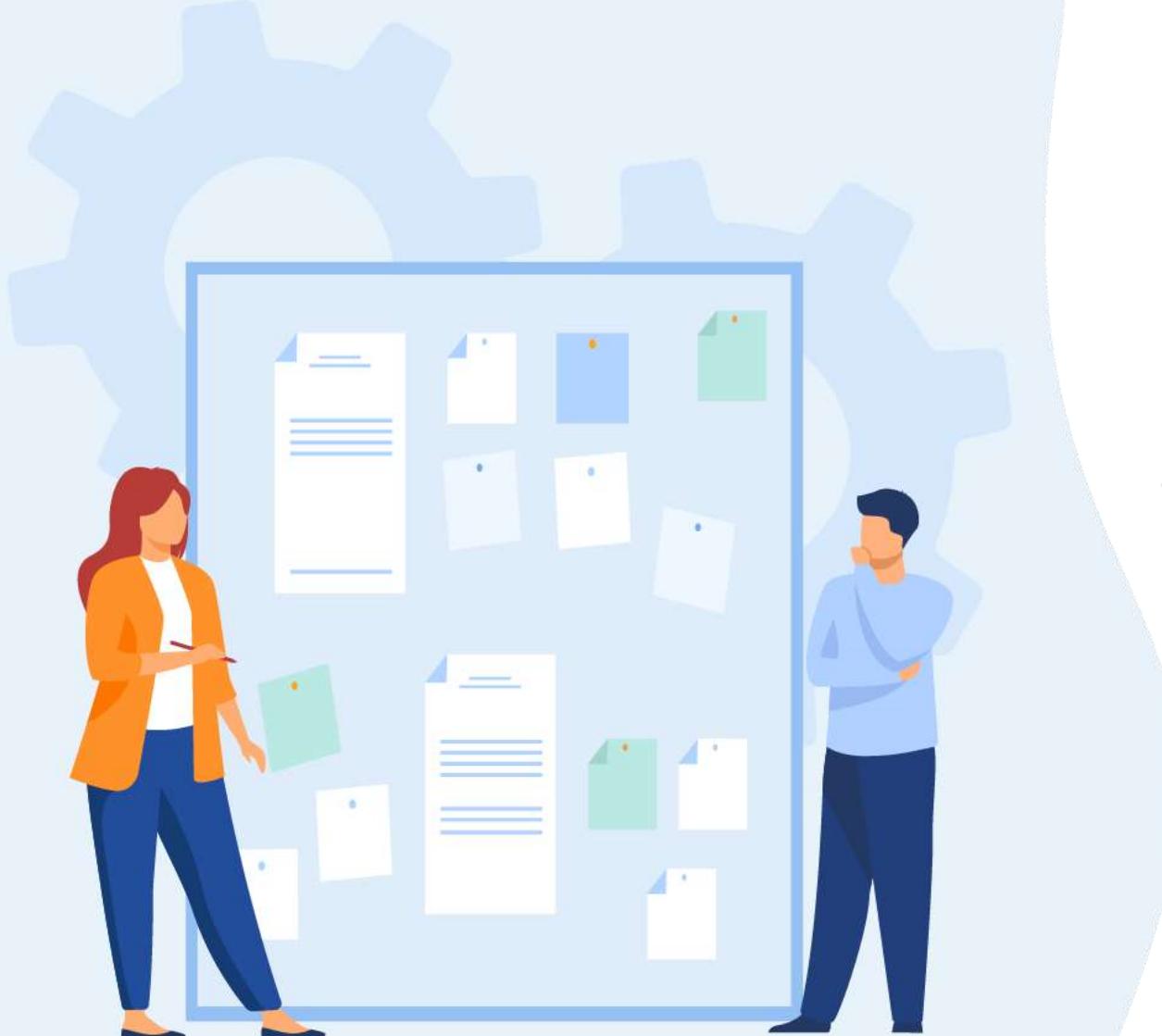


CHAPITRE 4

Manipulation des contrôleurs :

Ce que vous allez apprendre dans ce chapitre :

- Intérêt des contrôleurs
- Implémentation de contrôleur
- Contrôleur Middleware
- Contrôleur de ressources
- Injection de dépendances



CHAPITRE 4

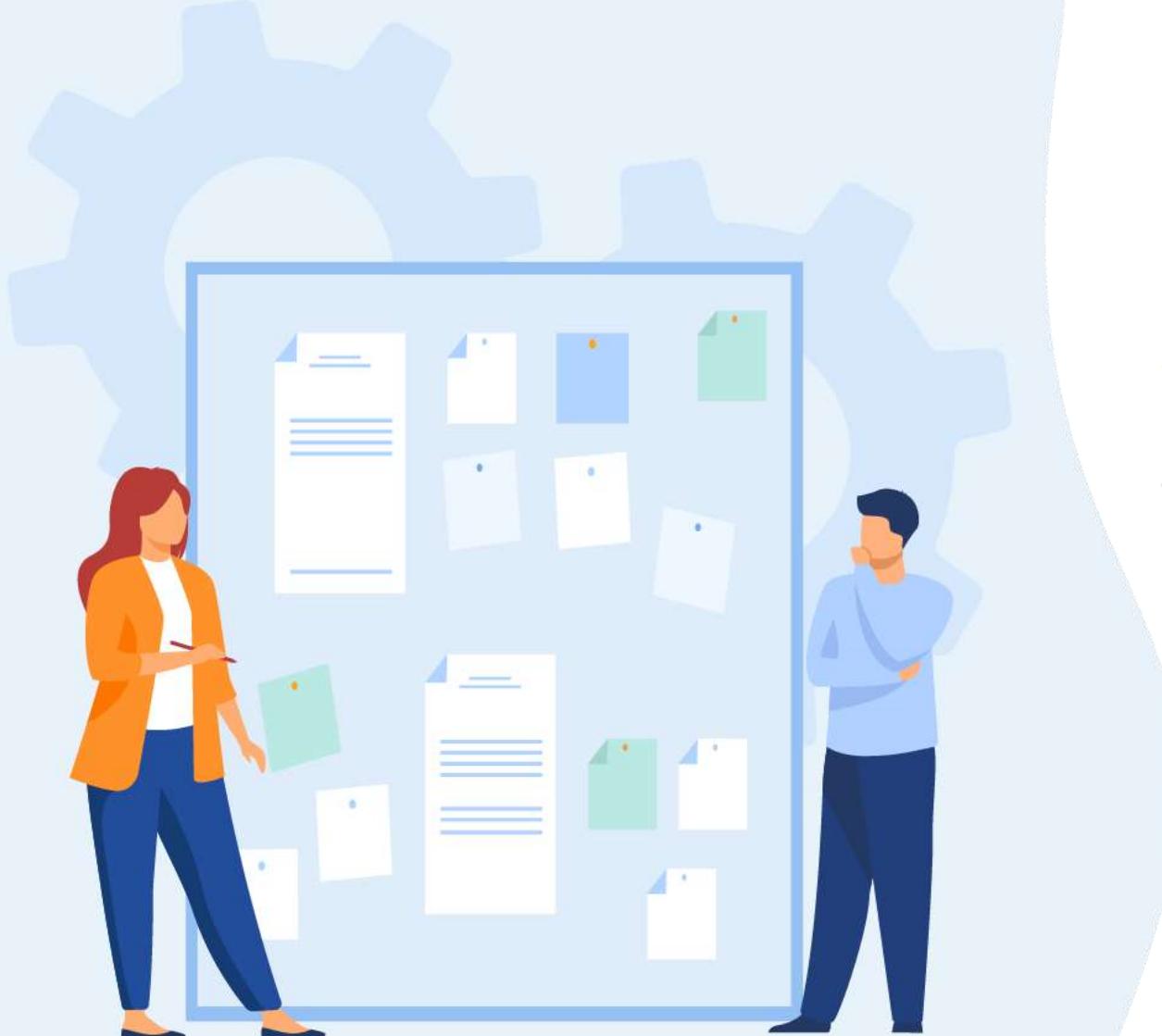
Manipulation des contrôleurs :

1. **Intérêt des contrôleurs**
2. Implémentation de contrôleur
3. Contrôleur Middleware
4. Contrôleur de ressources
5. Injection de dépendances

04 - Manipulation des contrôleurs :

Intérêt des contrôleurs

- Au lieu de définir toute votre logique de gestion des requêtes comme des fermetures dans vos fichiers de routage, vous pouvez organiser ce comportement à l'aide de classes "**contrôleur**". Les **contrôleurs** peuvent regrouper la logique de gestion des demandes associées dans une seule classe. Par exemple, une classe **UserController** peut gérer toutes les demandes entrantes liées aux utilisateurs, y compris l'affichage, la création, la mise à jour et la suppression d'utilisateurs. Par défaut, les contrôleurs sont stockés dans le répertoire **app/Http/Controllers**.



CHAPITRE 4

Manipulation des contrôleurs :

1. Intérêt des contrôleurs
2. **Implémentation de contrôleur**
3. Contrôleur Middleware
4. Contrôleur de ressources
5. Injection de dépendances

Contrôleurs de base

- Examinons un exemple de contrôleur de base. Notez que le contrôleur étend la classe de **contrôleur** de base incluse avec Laravel `App\Http\Controllers\Controller`

```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Http\Controllers\Controller;  
use App\Models\User;  
  
class UserController extends Controller  
{  
    /**  
     * Show the profile for a given user.  
     *  
     * @param int $id  
     * @return \Illuminate\View\View  
    */  
    public function show($id)  
    {  
        return view('user.profile', [  
            'user' => User::findOrFail($id)  
        ]);  
    }  
}
```

04 - Manipulation des contrôleurs :

Implémentation de contrôleur

- Vous pouvez définir une route vers cette méthode de contrôleur comme suit :

```
use App\Http\Controllers\UserController;  
  
Route::get('/user/{id}', [UserController::class, 'show']);
```

- Lorsqu'une requête entrante correspond à l'URI de route spécifié, la méthode **show** de la classe **App\Http\Controllers\UserController** est appelée et les paramètres de route sont transmis à la méthode.

Contrôleurs à simple action

- Si une action de contrôleur est particulièrement complexe, vous trouverez peut-être pratique de dédier une classe de contrôleur entière à cette action unique. Pour ce faire, vous pouvez définir une seule méthode `__invoke` dans le contrôleur :

```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Http\Controllers\Controller;  
use App\Models\User;  
  
class ProvisionServer extends Controller  
{  
    /**  
     * Provision a new web server.  
     *  
     * @return \Illuminate\Http\Response  
     */  
    public function __invoke()  
    {  
        // ...  
    }  
}
```

- Lors de l'enregistrement d'itinéraires pour des contrôleurs à action unique, vous n'avez pas besoin de spécifier une méthode de contrôleur. Au lieu de cela, vous pouvez simplement transmettre le nom du contrôleur au routeur :

```
use App\Http\Controllers\ProvisionServer;  
  
Route::post('/server', ProvisionServer::class);
```

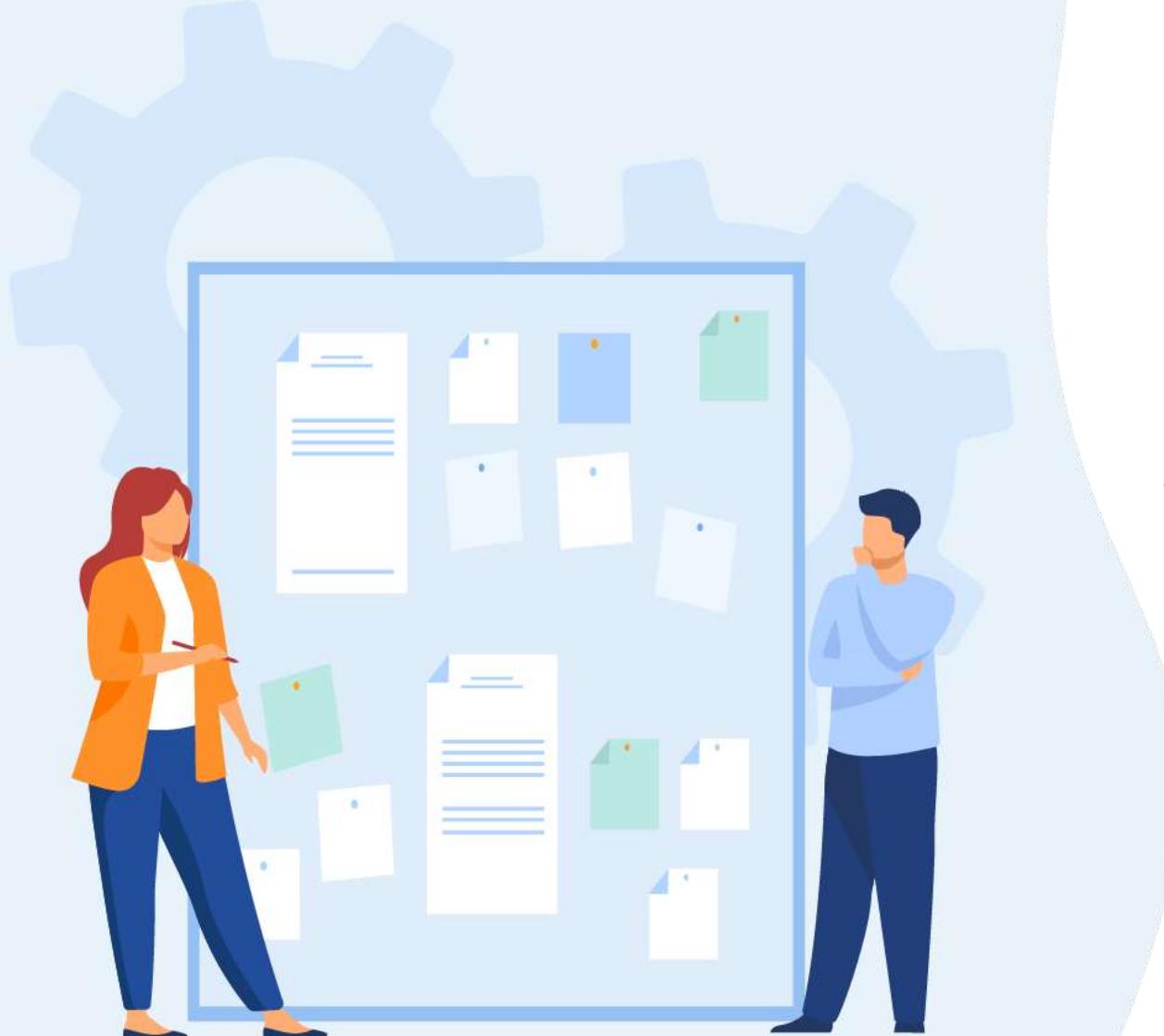
04 - Manipulation des contrôleurs :

Implémentation de contrôleur



- Vous pouvez générer un contrôleur invocable en utilisant l' option **--invokable** de la commande Artisan make:controller :

```
php artisan make:controller ProvisionServer --invokable
```



CHAPITRE 4

Manipulation des contrôleurs :

1. Intérêt des contrôleurs
2. Implémentation de contrôleur
3. **Contrôleur Middleware**
4. Contrôleur de ressources
5. Injection de dépendances

04 - Manipulation des contrôleurs :

Contrôleur Middleware



- Un middleware peut être affecté aux routes du contrôleur dans vos fichiers de routes :

```
Route::get('profile', [UserController::class, 'show'])->middleware('auth');
```

- Ou, vous trouverez peut-être pratique de spécifier un middleware dans le constructeur de votre contrôleur. En utilisant la méthode **middleware** dans le constructeur de votre contrôleur, vous pouvez affecter un middleware aux actions du contrôleur :

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
        $this->middleware('log')->only('index');
```

```
        $this->middleware('subscribed')->except('store');
    }
}
```

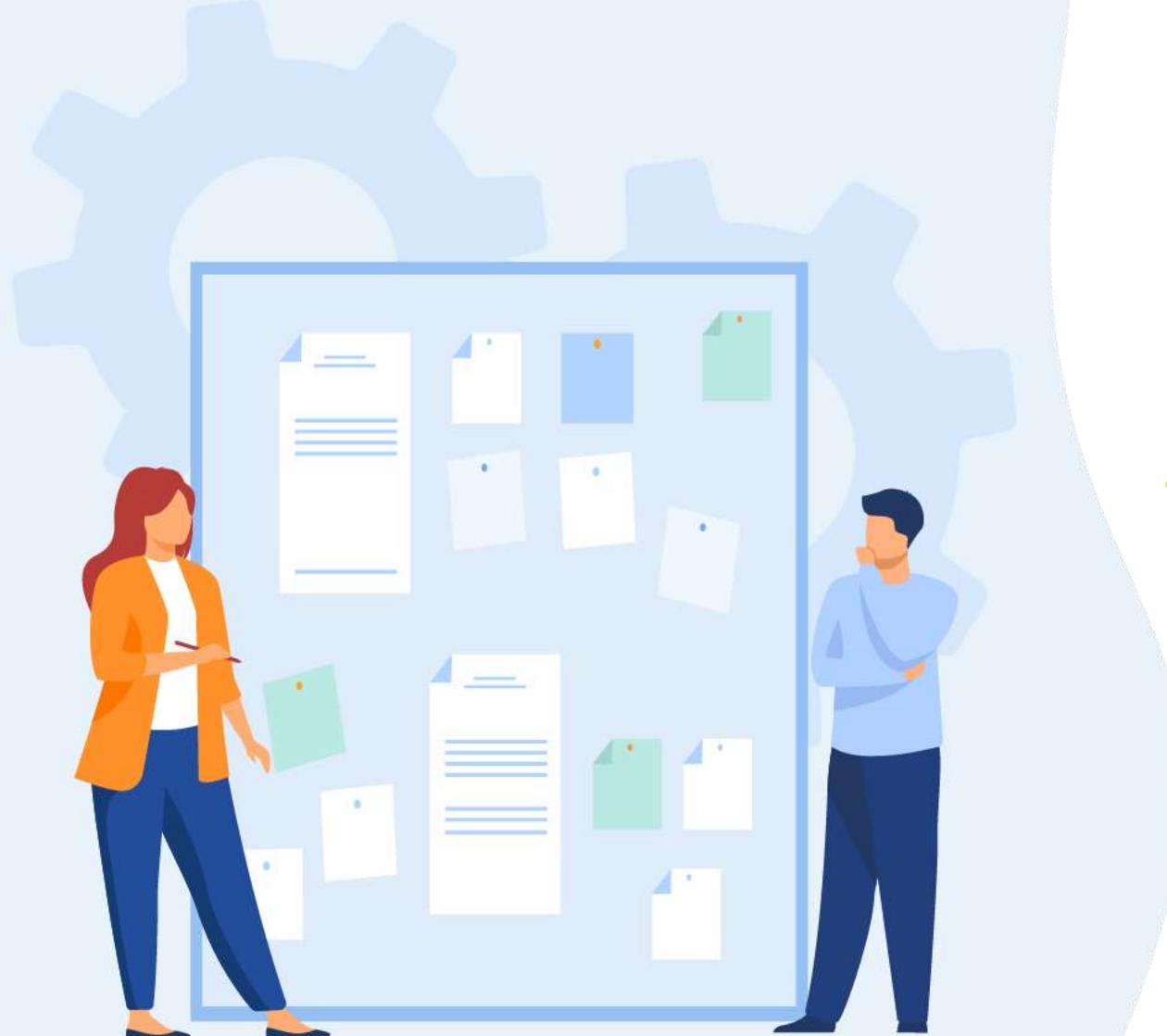
04 - Manipulation des contrôleurs :

Contrôleur Middleware



- Les contrôleurs vous permettent également d'enregistrer un middleware à l'aide d'une fermeture. Cela fournit un moyen pratique de définir un middleware en ligne pour un seul contrôleur sans définir une classe complète de middleware :

```
$this->middleware(function ($request, $next) {  
    return $next($request);  
});
```



CHAPITRE 4

Manipulation des contrôleurs :

1. Intérêt des contrôleurs
2. Implémentation de contrôleur
3. Contrôleur Middleware
4. **Contrôleur de ressources**
5. Injection de dépendances

04 - Manipulation des contrôleurs :

Contrôleur de ressources



- Si vous considérez chaque modèle Eloquent de votre application comme une "ressource", il est courant d'effectuer les mêmes ensembles d'actions sur chaque ressource de votre application. Par exemple, imaginez que votre application contient un modèle **Photo** et un modèle **Movie**. Il est probable que les utilisateurs puissent créer, lire, mettre à jour ou supprimer ces ressources.
- En raison de ce cas d'utilisation courant, le routage des ressources Laravel attribue les routes typiques de création, lecture, mise à jour et suppression ("CRUD") à un contrôleur avec une seule ligne de code. Pour commencer, nous pouvons utiliser l'option **make:controller** de la commande Artisan **--resource** pour créer rapidement un contrôleur pour gérer ces actions :

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

- Cette commande générera un contrôleur à **app/Http/Controllers/PhotoController.php**. Le contrôleur contiendra une méthode pour chacune des opérations de ressources disponibles. Ensuite, vous pouvez enregistrer une route de ressources qui pointe vers le contrôleur :

```
use App\Http\Controllers\PhotoController;
```

```
Route::resource('photos', PhotoController::class);
```

- Cette déclaration de route unique crée plusieurs routes pour gérer diverses actions sur la ressource. Le contrôleur généré aura déjà des méthodes stub pour chacune de ces actions. N'oubliez pas que vous pouvez toujours obtenir un aperçu rapide des routes de votre application en exécutant la commande Artisan **route:list**.

04 - Manipulation des contrôleurs :

Contrôleur de ressources



- Vous pouvez même enregistrer plusieurs contrôleurs de ressources à la fois en passant un tableau à la méthode **resources** :

```
Route::resources([
    'photos' => PhotoController::class,
    'posts' => PostController::class,
]);
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources



Actions générées par le contrôleur de ressources

Verbe	URI	Action	Nom de la route
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Personnalisation du comportement du modèle manquant

- En règle générale, une réponse HTTP 404 est générée si un modèle de ressource implicitement lié n'est pas trouvé. Cependant, vous pouvez personnaliser ce comportement en appelant la méthode **missing** lors de la définition de votre route de ressource. La méthode **missing** accepte une fermeture qui sera invoquée si un modèle lié implicitement ne peut être trouvé pour aucune des routes de la ressource :

```
use App\Http\Controllers\PhotoController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;

Route::resource('photos', PhotoController::class)
->missing(function (Request $request) {
    return Redirect::route('photos.index');
});
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources

Spécification du modèle de ressource

- Si vous utilisez la liaison de modèle de route et que vous souhaitez que les méthodes du contrôleur de ressources indiquent le type d'une instance de modèle, vous pouvez utiliser l'option **--model** lors de la génération du contrôleur :

```
php artisan make:controller PhotoController --model=Photo --resource
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources

Génération de demandes de formulaire

- Vous pouvez fournir l'option **--requests** lors de la génération d'un contrôleur de ressources pour demander à Artisan de générer des classes de requête de formulaire pour les méthodes de stockage et de mise à jour du contrôleur :

```
php artisan make:controller PhotoController --model=Photo --resource --requests
```

Routes de ressources partielles

- Lors de la déclaration d'une route de ressources, vous pouvez spécifier un sous-ensemble d'actions que le contrôleur doit gérer au lieu de l'ensemble complet d'actions par défaut :

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class)->only([
    'index', 'show'
]);

Route::resource('photos', PhotoController::class)->except([
    'create', 'store', 'update', 'destroy'
]);
```

Itinéraires de ressources d'API

- Lors de la déclaration des routes de ressources qui seront consommées par les API, vous souhaiterez généralement exclure les routes qui présentent des modèles HTML tels que `create` et `edit`. Pour plus de commodité, vous pouvez utiliser la méthode `apiResource` pour exclure automatiquement ces deux routes :

```
use App\Http\Controllers\PhotoController;  
  
Route::apiResource('photos', PhotoController::class);
```

- Vous pouvez enregistrer plusieurs contrôleurs de ressources d'API à la fois en passant un tableau à la méthode `apiResources` :

```
use App\Http\Controllers\PhotoController;  
use App\Http\Controllers\PostController;  
  
Route::apiResources([  
    'photos' => PhotoController::class,  
    'posts' => PostController::class,  
]);
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources



- Pour générer rapidement un contrôleur de ressources d'API qui n'inclut pas les méthodes `create` ou `update`, utilisez le commutateur lors de l'exécution de la commande : `edit--apimake:controller`

```
php artisan make:controller PhotoController --api
```

Ressources imbriquées

- Parfois, vous devrez peut-être définir des itinéraires vers une ressource imbriquée. Par exemple, une ressource photo peut avoir plusieurs commentaires qui peuvent être joints à la photo. Pour imbriquer les contrôleurs de ressources, vous pouvez utiliser la notation "point" dans votre déclaration de route :

```
use App\Http\Controllers\PhotoCommentController;  
  
Route::resource('photos.comments', PhotoCommentController::class);
```

- Cette route enregistrera une ressource imbriquée accessible avec des URI comme suit :

```
/photos/{photo}/comments/{comment}
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources

Nidification peu profonde

- Souvent, il n'est pas tout à fait nécessaire d'avoir à la fois les identifiants parent et enfant dans un URI puisque l'identifiant enfant est déjà un identifiant unique. Lorsque vous utilisez des identifiants uniques tels que des clés primaires à incrémentation automatique pour identifier vos modèles dans des segments d'URI, vous pouvez choisir d'utiliser une "imbrication peu profonde":

```
use App\Http\Controllers\CommentController;  
  
Route::resource('photos.comments', CommentController::class)->shallow();
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources



- Cette définition de route définira les routes suivantes :

Verbe	URI	Action	Nom de la route
GET	/photos/{photo}/comments	index	photos.comments.index
GET	/photos/{photo}/comments/create	create	photos.comments.create
POST	/photos/{photo}/comments	store	photos.comments.store
GET	/comments/{comment}	show	comments.show
GET	/comments/{comment}/edit	edit	comments.edit
PUT/PATCH	/comments/{comment}	update	comments.update
DELETE	/comments/{comment}	destroy	comments.destroy

04 - Manipulation des contrôleurs :

Contrôleur de ressources



Nommer les routes de ressources

- Par défaut, toutes les actions du contrôleur de ressources ont un nom de route ; cependant, vous pouvez remplacer ces noms en transmettant un tableau **names** avec les noms de route souhaités :

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class)->names([
    'create' => 'photos.build'
]);
```

Nommer les paramètres de route de ressource

- Par défaut, Route::resource créera les paramètres de route pour vos ressources de route en fonction de la version "singularisée" du nom de la ressource. Vous pouvez facilement remplacer cela par ressource à l'aide de la méthode **parameters**. Le tableau transmis à la méthode **parameters** doit être un tableau associatif de noms de ressources et de noms de paramètres :

```
use App\Http\Controllers\AdminUserController;

Route::resource('users', AdminUserController::class)->parameters([
    'users' => 'admin_user'
]);
```

- L'exemple ci-dessus génère l'URI suivant pour la route de la ressource **show** :

```
/users/{admin_user}
```

Évaluer les itinéraires des ressources

- La fonctionnalité de liaison de modèle implicite délimitée de Laravel peut automatiquement délimiter les liaisons imbriquées de sorte que le modèle enfant résolu soit confirmé comme appartenant au modèle parent. En utilisant la méthode **scoped** lors de la définition de votre ressource imbriquée, vous pouvez activer la portée automatique et indiquer à Laravel par quel champ la ressource enfant doit être récupérée :

```
use App\Http\Controllers\PhotoCommentController;
```

```
Route::resource('photos.comments', PhotoCommentController::class)->scoped([
    'comment' => 'slug',
]);
```

- Cette route enregistrera une ressource imbriquée délimitée accessible avec des URI comme suit :

```
/photos/{photo}/comments/{comment:slug}
```

Localisation des URI de ressource

- Par défaut, **Route::resource** créera des URI de ressource en utilisant des verbes anglais et des règles de pluriel. Si vous avez besoin de localiser les verbes d'action **create** et **edit**, vous pouvez utiliser la méthode **Route::resourceVerbs**. Cela peut être fait au début de la méthode **boot** dans votre application **App\Providers\RouteServiceProvider**:

```
/**  
 * Define your route model bindings, pattern filters, etc.  
 *  
 * @return void  
 */  
public function boot()  
{  
    Route::resourceVerbs([  
        'create' => 'crear',  
        'edit' => 'editar',  
    ]);  
  
    // ...  
}
```

04 - Manipulation des contrôleurs :

Contrôleur de ressources

- Le pluraliseur de Laravel prend en charge plusieurs langues différentes que vous pouvez configurer en fonction de vos besoins. Une fois que les verbes et le langage de pluralisation ont été personnalisés, un enregistrement de route de ressources tel que **Route::resource('publicacion', PublicacionController::class)** produira les URI suivants :

```
/publicacion/crear
```

```
/publicacion/{publicaciones}/editar
```

04 - Manipulation des contrôleurs :

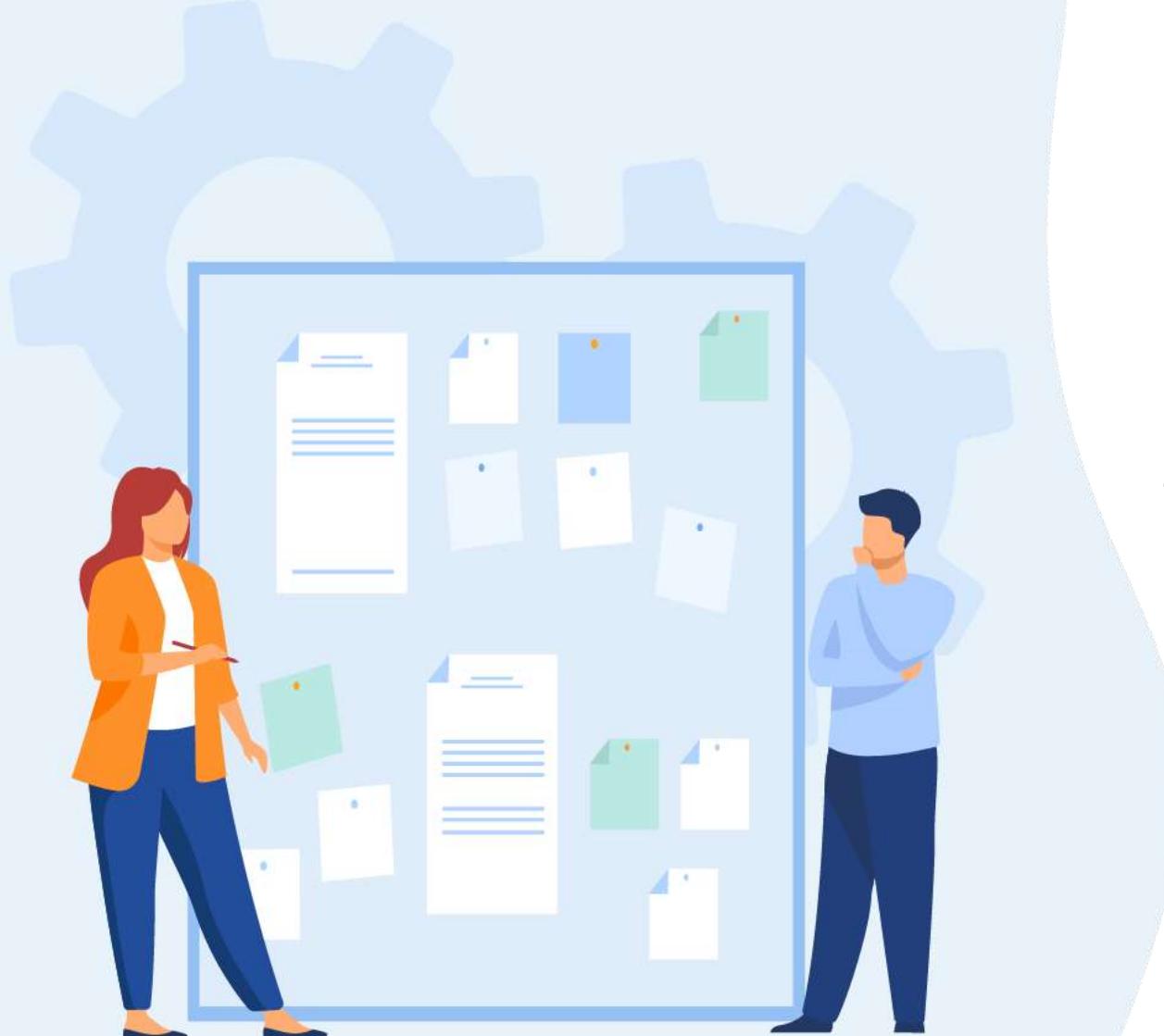
Contrôleur de ressources



Compléter les contrôleurs de ressources

- Si vous devez ajouter des routes supplémentaires à un contrôleur de ressources au-delà de l'ensemble de routes de ressources par défaut, vous devez définir ces routes avant votre appel à la méthode **Route::resource** ; sinon, les routes définies par la méthode **resource** peuvent involontairement prendre le pas sur vos routes supplémentaires :

```
use App\Http\Controller\PhotoController;  
  
Route::get('/photos/popular', [PhotoController::class, 'popular']);  
Route::resource('photos', PhotoController::class);
```



CHAPITRE 4

Manipulation des contrôleurs :

1. Intérêt des contrôleurs
2. Implémentation de contrôleur
3. Contrôleur Middleware
4. Contrôleur de ressources
5. **Injection de dépendances**

Injection de constructeur

- Le conteneur de service Laravel est utilisé pour résoudre tous les contrôleurs Laravel. En conséquence, vous pouvez indiquer toutes les dépendances dont votre contrôleur peut avoir besoin dans son constructeur. Les dépendances déclarées seront automatiquement résolues et injectées dans l'instance du contrôleur :

```
<?php  
  
namespace App\Http\Controllers;  
  
use App\Repositories\UserRepository;  
  
class UserController extends Controller  
{  
    /**  
     * The user repository instance.  
     */  
    protected $users;  
  
    /**
```

```
     * Create a new controller instance.  
     *  
     * @param \App\Repositories\UserRepository $users  
     * @return void  
     */  
    public function __construct(UserRepository $users)  
    {  
        $this->users = $users;  
    }  
}
```

Injection de méthode

- En plus de l'injection de constructeur, vous pouvez également créer des dépendances d'indication de type sur les méthodes de votre contrôleur. Un cas d'utilisation courant pour l'injection de méthode consiste à injecter l'instance `Illuminate\Http\Request` dans les méthodes de votre contrôleur :

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class UserController extends Controller  
{  
    /**  
     * Store a new user.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @return \Illuminate\Http\Response  
     */  
    public function store(Request $request)  
    {  
        $name = $request->name;  
        //  
    }  
}
```

04 - Manipulation des contrôleurs : Injection de dépendances



- Si votre méthode de contrôleur attend également une entrée d'un paramètre de route, lister vos arguments de route après vos autres dépendances. Par exemple, si votre route est définie comme ceci :

```
use App\Http\Controllers\UserController;  
  
Route::put('/user/{id}', [UserController::class, 'update']);
```

04 - Manipulation des contrôleurs : Injection de dépendances



- Vous pouvez toujours taper le **Illuminate\Http\Request** et accéder à votre paramètre **id** en définissant votre méthode de contrôleur comme suit :

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class UserController extends Controller  
{  
    /**  
     * Update the given user.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @param string $id  
     * @return \Illuminate\Http\Response  
     */
```

```
    public function update(Request $request, $id)  
    {  
        //  
    }  
}
```



CHAPITRE 5

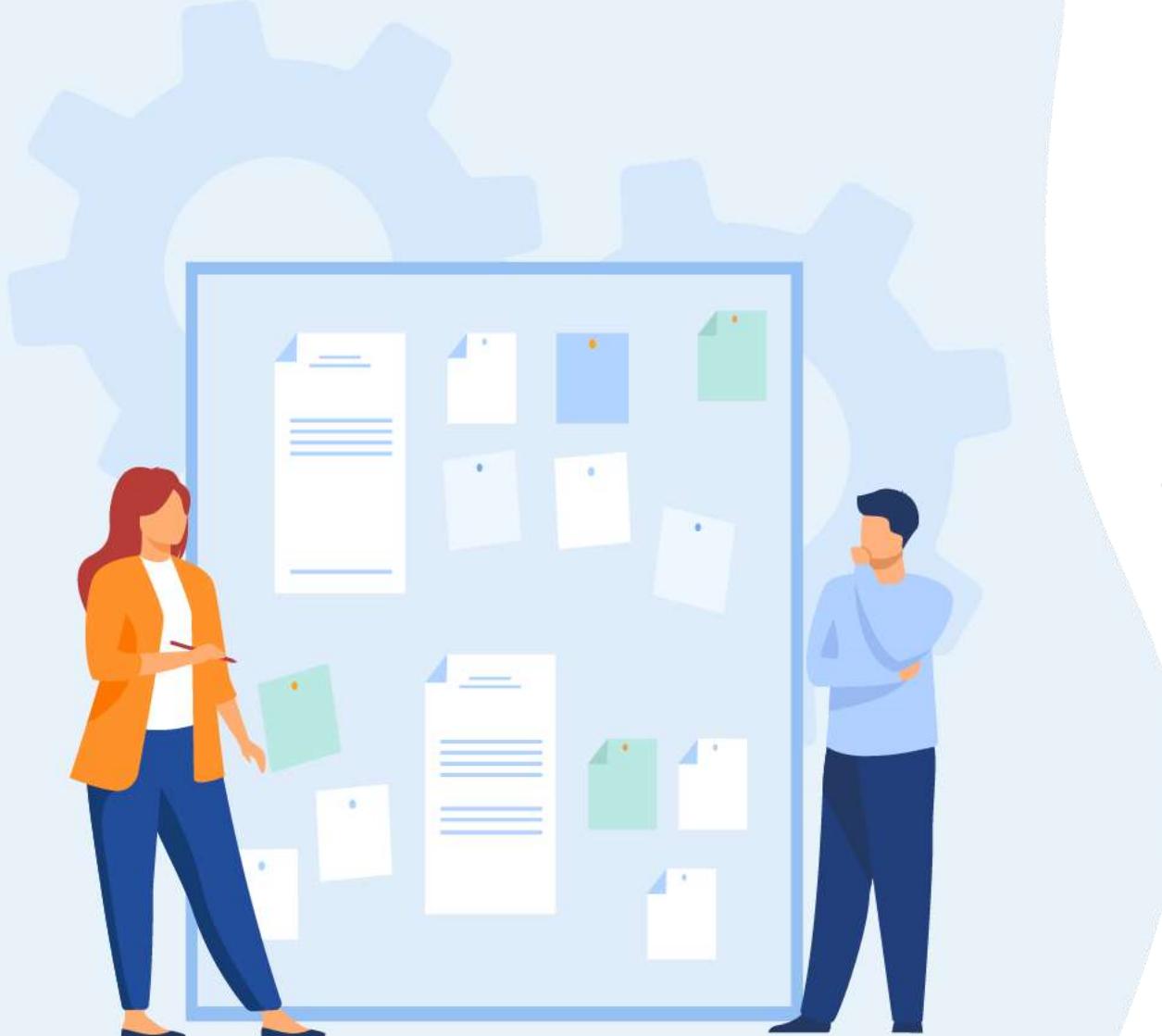
Manipulation des requêtes HTTP

Ce que vous allez apprendre dans ce chapitre :

- Introduction
- Interaction avec les requêtes
- Input
- Fichiers
- Configuration des proxys de confiance
- Configuration des hôtes approuvés



05heure



CHAPITRE 5

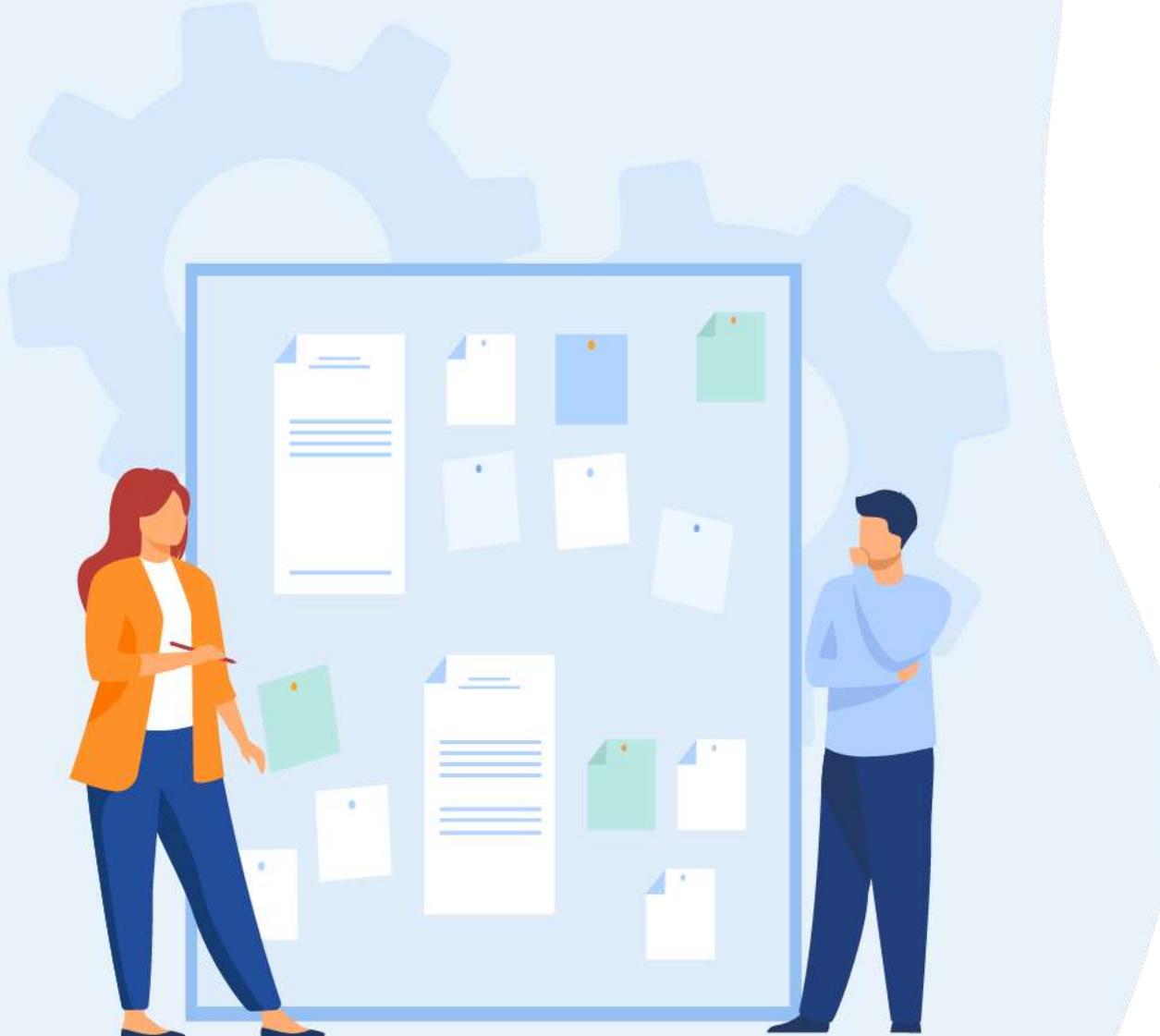
Manipulation des requêtes HTTP

1. **Introduction**
2. Interaction avec les requêtes
3. Input
4. Fichiers
5. Configuration des proxys de confiance
6. Configuration des hôtes approuvés

05 - Manipulation des requêtes http :

Introduction

- La classe de `Laravel Illuminate\Http\Request` fournit un moyen orienté objet d'interagir avec la requête HTTP actuelle gérée par votre application, ainsi que de récupérer l'entrée, les cookies et les fichiers qui ont été soumis avec la requête.



CHAPITRE 5

Manipulation des requêtes HTTP

1. Introduction
2. **Interaction avec les requêtes**
3. Input
4. Fichiers
5. Configuration des proxys de confiance
6. Configuration des hôtes approuvés

05 - Manipulation des requêtes http :

Interaction avec les requêtes



Accéder à la demande

- Pour obtenir une instance de la requête HTTP actuelle via l'injection de dépendances, vous devez indiquer la classe `Illuminate\Http\Request` sur votre méthode de fermeture de route ou de contrôleur. L'instance de requête entrante sera automatiquement injectée par le conteneur de service Laravel :

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class UserController extends Controller  
{  
    /**  
     * Store a new user.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @return \Illuminate\Http\Response  
     */
```

```
    public function store(Request $request)  
    {  
        $name = $request->input('name');  
  
        //  
    }  
}
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



- Comme mentionné, vous pouvez également indiquer la classe **Illuminate\Http\Request** sur une fermeture d'itinéraire. Le conteneur de service injectera automatiquement la requête entrante dans la fermeture lors de son exécution :

```
use Illuminate\Http\Request;

Route::get('/', function (Request $request) {
    //
});
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



Paramètres d'injection de dépendance et de routage

- Si votre méthode de contrôleur attend également une entrée d'un paramètre de route, vous devez lister vos paramètres de route après vos autres dépendances. Par exemple, si votre route est définie comme ceci :

```
use App\Http\Controllers\UserController;  
  
Route::put('/user/{id}', [UserController::class, 'update']);
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



- Vous pouvez toujours taper le **Illuminate\Http\Request** et accéder à votre paramètre **id** route en définissant votre méthode de contrôleur comme suit :

```
<?php  
  
namespace App\Http\Controllers;  
  
use Illuminate\Http\Request;  
  
class UserController extends Controller  
{  
    /**  
     * Update the specified user.  
     *  
     * @param \Illuminate\Http\Request $request  
     * @param string $id  
     * @return \Illuminate\Http\Response  
    */  
    public function update(Request $request, $id)  
    {  
        //  
    }  
}
```

Récupération du chemin de la requête

- La méthode `path` renvoie les informations de chemin de la requête. Ainsi, si la requête entrante est ciblée sur [`http://example.com/foo/bar`](http://example.com/foo/bar), la méthode `path` retournera `foo/bar`:

```
$uri = $request->path();
```

Inspecter le chemin/la route de la demande

- La méthode **is** vous permet de vérifier que le chemin de la demande entrante correspond à un modèle donné. Vous pouvez utiliser le caractère * comme caractère générique lors de l'utilisation de cette méthode :

```
if ($request->is('admin/*')) {  
    //  
}
```

- En utilisant la méthode **routels**, vous pouvez déterminer si la requête entrante correspond à une route nommée :

```
if ($request->routels('admin.*')) {  
    //  
}
```

Récupération de l'URL de la requête

- Pour récupérer l'URL complète de la requête entrante, vous pouvez utiliser les méthodes **url** ou **.fullUrl**. La méthode **url** renverra l'URL sans la chaîne de requête, tandis que la méthode **fullUrl** inclut la chaîne de requête :

```
$url = $request->url();
```

```
$urlWithQueryString = $request->fullUrl();
```

- Si vous souhaitez ajouter des données de chaîne de requête à l'URL actuelle, vous pouvez appeler la méthode **fullUrlWithQuery**. Cette méthode fusionne le tableau donné de variables de chaîne de requête avec la chaîne de requête actuelle :

```
$request->fullUrlWithQuery(['type' => 'phone']);
```

Récupération de l'hôte de requête

- Vous pouvez récupérer "l'hôte" de la requête entrante via les méthodes **host**, **httpHost** et **schemeAndHttpHost**:

```
$request->host();  
$request->httpHost();  
$request->schemeAndHttpHost();
```

Récupération de la méthode Request

- La méthode **method** renverra le verbe HTTP pour la requête. Vous pouvez utiliser la méthode **isMethod** pour vérifier que le verbe HTTP correspond à une chaîne donnée :

```
$method = $request->method();  
  
if ($request->isMethod('post')) {  
    //  
}
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



En-têtes de demande

- Vous pouvez récupérer un en-tête de requête à partir de l'instance Illuminate\Http\Request à l'aide de la méthode header. Si l'en-tête n'est pas présent sur la requête, null sera renvoyé. Cependant, la méthode header accepte un deuxième argument facultatif qui sera retourné si l'en-tête n'est pas présent sur la requête :

```
$value = $request->header('X-Header-Name');
```

```
$value = $request->header('X-Header-Name', 'default');
```

- La méthode hasHeader peut être utilisée pour déterminer si la requête contient un en-tête donné :

```
if ($request->hasHeader('X-Header-Name')) {  
//  
}
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes

- Pour plus de commodité, le procédé `bearerToken` peut être utilisé pour récupérer un jeton de support à partir de l'en-tête `Authorization`. Si aucun en-tête de ce type n'est présent, une chaîne vide sera renvoyée :

```
$token = $request->bearerToken();
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



Demander l'adresse IP

- La méthode ip peut être utilisée pour récupérer l'adresse IP du client qui a fait la requête à votre application :

```
$ipAddress = $request->ip();
```

Négociation de contenu

- Laravel fournit plusieurs méthodes pour inspecter les types de contenu demandés par la requête entrante via l'en-tête Accept. Tout d'abord, la méthode `getAcceptableContentTypes` renverra un tableau contenant tous les types de contenu acceptés par la requête :

```
$contentTypes = $request->getAcceptableContentTypes();
```

La méthode `accepts` accepte un tableau de types de contenu et renvoie `true` si l'un des types de contenu est accepté par la requête. Dans le cas contraire, `false` sera renvoyé :

```
if ($request->accepts(['text/html', 'application/json'])) {  
    // ...  
}
```

Vous pouvez utiliser la méthode `prefers` pour déterminer quel type de contenu parmi un tableau donné de types de contenu est le plus préféré par la requête. Si aucun des types de contenu fournis n'est accepté par la requête, `null` sera renvoyé :

```
$preferred = $request->prefers(['text/html', 'application/json']);
```

05 - Manipulation des requêtes http :

Interaction avec les requêtes



- Étant donné que de nombreuses applications ne servent que du HTML ou du JSON, vous pouvez utiliser la méthode expectsJson pour déterminer rapidement si la requête entrante attend une réponse JSON :

```
if ($request->expectsJson()) {  
    // ...  
}
```

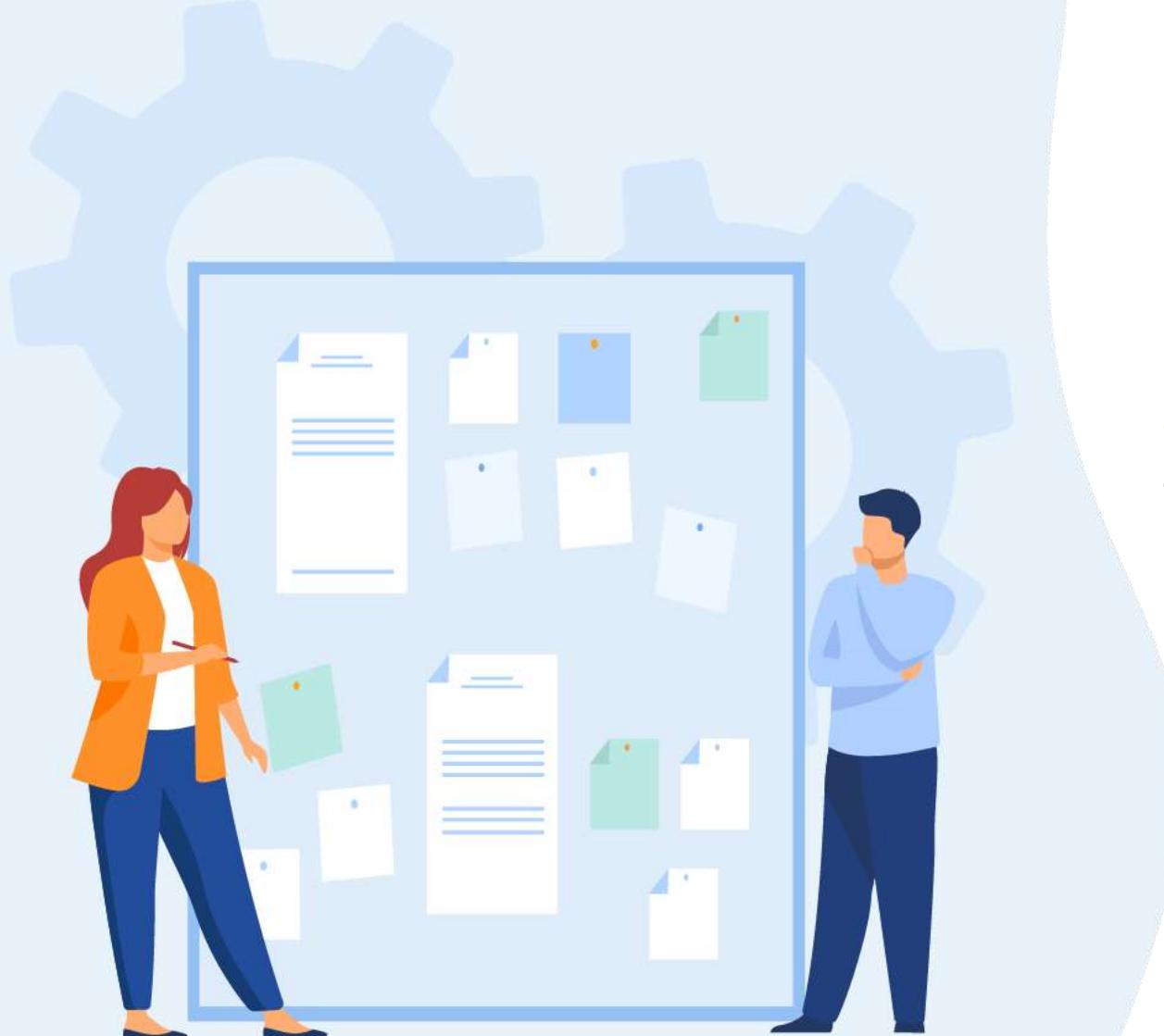
Demandes PSR-7

- La norme PSR-7 spécifie les interfaces pour les messages HTTP, y compris les requêtes et les réponses. Si vous souhaitez obtenir une instance d'une requête PSR-7 au lieu d'une requête Laravel, vous devrez d'abord installer quelques bibliothèques. Laravel utilise le composant Symfony HTTP Message Bridge pour convertir les requêtes et réponses typiques de Laravel en implémentations compatibles PSR-7 :

```
composer require symfony/psr-http-message-bridge  
composer require nyholm/psr7
```

Une fois que vous avez installé ces bibliothèques, vous pouvez obtenir une requête PSR-7 en tapant l'interface de requête sur votre méthode de fermeture de route ou de contrôleur :

```
use Psr\Http\Message\ServerRequestInterface;  
  
Route::get('/', function (ServerRequestInterface $request) {  
    //  
});
```



CHAPITRE 5

Manipulation des requêtes HTTP

1. Introduction
2. Interaction avec les requêtes
- 3. Input**
4. Fichiers
5. Configuration des proxys de confiance
6. Configuration des hôtes approuvés

Récupération de toutes les données d'entrée

- Vous pouvez récupérer toutes les données d'entrée de la demande entrante sous forme d'un array en utilisant la méthode all. Cette méthode peut être utilisée que la requête entrante provienne d'un formulaire HTML ou soit une requête XHR :

```
$input = $request->all();
```

- En utilisant la méthode collect, vous pouvez recuperer toutes les donnees d'entree de la requette entrante sous la forme d'une collection :

```
$input = $request->collect();
```

- La méthode collect vous permet également de récupérer un sous-ensemble de l'entrée de requête entrante sous forme de collection :

```
$request->collect('users')->each(function ($user) {  
    // ...  
});
```

Récupération d'une valeur d'entrée

- À l'aide de quelques méthodes simples, vous pouvez accéder à toutes les entrées utilisateur de votre instance Illuminate\Http\Request sans vous soucier du verbe HTTP utilisé pour la requête. Quel que soit le verbe HTTP, la méthode input peut être utilisée pour récupérer l'entrée utilisateur.

```
$name = $request->input('name');
```

- Vous pouvez passer une valeur par défaut comme second argument de la méthode input. Cette valeur sera renvoyée si la valeur d'entrée demandée n'est pas présente dans la requête :

```
$name = $request->input('name', 'Sally');
```

- Lorsque vous travaillez avec des formulaires contenant des entrées de tableau, utilisez la notation "point" pour accéder aux tableaux :

```
$name = $request->input('products.0.name');
```

```
$names = $request->input('products.*.name');
```

05 - Manipulation des requêtes http :

Input



- Vous pouvez appeler la méthode input sans aucun argument afin de récupérer toutes les valeurs d'entrée sous forme de tableau associatif :

```
$input = $request->input();
```

Récupération de l'entrée de la chaîne de requête

- Alors que la méthode input récupère les valeurs de l'intégralité de la charge utile de la requête (y compris la chaîne de requête), la méthode query ne récupère que les valeurs de la chaîne de requête :

```
$name = $request->query('name');
```

- Si les données de valeur de chaîne de requête demandées ne sont pas présentes, le deuxième argument de cette méthode sera renvoyé :

```
$name = $request->query('name', 'Helen');
```

- Vous pouvez appeler la méthode query sans aucun argument afin de récupérer toutes les valeurs de la chaîne de requête sous forme de tableau associatif :

```
$query = $request->query();
```

Récupération des valeurs d'entrée JSON

- Lors de l'envoi de requêtes JSON à votre application, vous pouvez accéder aux données JSON via la méthode input tant que l'en-tête Content-Type de la requête est correctement défini sur application/json. Vous pouvez même utiliser la syntaxe "point" pour récupérer des valeurs imbriquées dans des tableaux JSON :

```
$name = $request->input('user.name');
```

05 - Manipulation des requêtes http :

Input



Récupération de valeurs d'entrée stringables

- Au lieu de récupérer les données d'entrée de la requête en tant que primitive string, vous pouvez utiliser la méthode string pour récupérer les données de la requête en tant qu'instance de Illuminate\Support\Stringable:

```
$name = $request->string('name')->trim();
```

05 - Manipulation des requêtes http :

Input

Récupération des valeurs d'entrée booléennes

- Lorsqu'il s'agit d'éléments HTML tels que des cases à cocher, votre application peut recevoir des valeurs "véridiques" qui sont en fait des chaînes. Par exemple, "vrai". Pour plus de commodité, vous pouvez utiliser la méthode boolean pour récupérer ces valeurs sous forme de booléens. La méthode boolean renvoie true pour 1, "1", vrai, "vrai" et "oui". Toutes les autres valeurs renverront false :

```
$archived = $request->boolean('archived');
```

05 - Manipulation des requêtes http :

Input



Récupération des valeurs d'entrée de date

- Pour plus de commodité, les valeurs d'entrée contenant des dates/heures peuvent être récupérées en tant qu'une instances à l'aide de la méthode date. Si la requête ne contient pas de valeur d'entrée avec le nom donné, null sera renvoyé :

```
$birthday = $request->date('birthday');
```

05 - Manipulation des requêtes http :

Input



Récupération des valeurs d'entrée de date

- Pour plus de commodité, les valeurs d'entrée contenant des dates/heures peuvent être récupérées en tant qu'une instances à l'aide de la méthode date. Si la requête ne contient pas de valeur d'entrée avec le nom donné, null sera renvoyé :

```
$birthday = $request->date('birthday');
```

- Les deuxième et troisième arguments acceptés par la méthode date peuvent être utilisés pour spécifier respectivement le format et le fuseau horaire de la date :

```
$elapsed = $request->date('elapsed', '!H:i', 'Europe/Madrid');
```

- Si la valeur d'entrée est présente mais a un format invalide, un InvalidArgumentException sera lancé ; par conséquent, il est recommandé de valider l'entrée avant d'appeler la méthode date.

Récupération des valeurs d'entrée Enum

- Les valeurs d'entrée qui correspondent aux énumérations PHP peuvent également être extraites de la requête. Si la demande ne contient pas de valeur d'entrée avec le nom donné ou si l'énumération n'a pas de valeur de sauvegarde qui correspond à la valeur d'entrée, null sera renvoyé. La méthode enum accepte le nom de la valeur d'entrée et la classe enum comme premier et deuxième arguments :

```
use App\Enums>Status;  
  
$status = $request->enum('status', Status::class);
```

Récupération de l'entrée via les propriétés dynamiques

- Vous pouvez également accéder aux entrées utilisateur à l'aide des propriétés dynamiques de l'instance Illuminate\Http\Request. Par exemple, si l'un des formulaires de votre application contient un champ name, vous pouvez accéder à la valeur du champ comme suit :

```
$name = $request->name;
```

- Lors de l'utilisation de propriétés dynamiques, Laravel recherchera d'abord la valeur du paramètre dans la charge utile de la requête. S'il n'est pas présent, Laravel recherchera le champ dans les paramètres de la route correspondante,

05 - Manipulation des requêtes http :

Input



Récupération d'une partie des données d'entrée

- Si vous avez besoin de récupérer un sous-ensemble des données d'entrée, vous pouvez utiliser les méthodes `only` et `except`. Ces deux méthodes acceptent un seul array ou une liste dynamique d'arguments :

```
$input = $request->only(['username', 'password']);  
  
$input = $request->only('username', 'password');  
  
$input = $request->except(['credit_card']);  
  
$input = $request->except('credit_card');
```

05 - Manipulation des requêtes http :

Input



Déterminer si l'entrée est présente

- Vous pouvez utiliser la méthode has pour déterminer si une valeur est présente sur la demande. La méthode has retourne true si la valeur est présente sur la requête :

```
if ($request->has('name')) {  
    //  
}
```

- Lorsqu'on lui donne un tableau, la méthode has déterminera si toutes les valeurs spécifiées sont présentes :

```
if ($request->has(['name', 'email'])) {  
    //  
}
```

- La méthode whenHas exécutera la fermeture donnée si une valeur est présente sur la requête :

```
$request->whenHas('name', function ($input) {  
    //  
});
```

05 - Manipulation des requêtes http :

Input



- Une deuxième fermeture peut être passée à la méthode whenHas qui sera exécutée si la valeur spécifiée n'est pas présente sur la requête :

```
$request->whenHas('name', function ($input) {  
    // The "name" value is present...  
}, function () {  
    // The "name" value is not present...  
});
```

- La méthode hasAny renvoie true si l'une des valeurs spécifiées est présente :

```
if ($request->hasAny(['name', 'email'])) {  
    //  
}
```

- Si vous souhaitez déterminer si une valeur est présente sur la requête et n'est pas vide, vous pouvez utiliser la méthode filled :

```
if ($request->filled('name')) {  
    //  
}
```

05 - Manipulation des requêtes http :

Input



- La méthode `whenFilled` exécutera la fermeture donnée si une valeur est présente sur la requête et n'est pas vide :

```
$request->whenFilled('name', function ($input) {  
    //  
});
```

- Une deuxième fermeture peut être passée à la méthode `whenFilled` qui sera exécutée si la valeur spécifiée n'est pas "remplie":

```
$request->whenFilled('name', function ($input) {  
    // The "name" value is filled...  
}, function () {  
    // The "name" value is not filled...  
});
```

- Pour déterminer si une clé donnée est absente de la requête, vous pouvez utiliser la méthode `missing` :

```
if ($request->missing('name')) {  
    //  
}
```

05 - Manipulation des requêtes http :

Input



Fusion d'entrées supplémentaires

- Parfois, vous devrez peut-être fusionner manuellement des entrées supplémentaires dans les données d'entrée existantes de la demande. Pour ce faire, vous pouvez utiliser la méthode merge :

```
$request->merge(['votes' => 0]);
```

- La méthode mergeIfMissing peut être utilisée pour fusionner l'entrée dans la requête si les clés correspondantes n'existent pas déjà dans les données d'entrée de la requête :

```
$request->mergeIfMissing(['votes' => 0]);
```

Ancienne entrée

- Laravel vous permet de conserver les entrées d'une requête lors de la prochaine requête. Cette fonctionnalité est particulièrement utile pour remplir à nouveau les formulaires après avoir détecté des erreurs de validation. Cependant, si vous utilisez les fonctionnalités de validation incluses de Laravel , il est possible que vous n'ayez pas besoin d'utiliser manuellement ces méthodes de flashage d'entrée de session directement, car certaines des fonctionnalités de validation intégrées de Laravel les appelleront automatiquement.

05 - Manipulation des requêtes http :

Input



Entrée clignotante à la session

- La méthode flash sur la classe Illuminate\Http\Request fera clignoter l'entrée actuelle de la session afin qu'elle soit disponible lors de la prochaine requête de l'utilisateur à l'application :

```
$request->flash();
```

- Vous pouvez également utiliser les méthodes flashOnly et flashExcept pour flasher un sous-ensemble des données de la demande dans la session. Ces méthodes sont utiles pour conserver des informations sensibles telles que les mots de passe hors de la session :

```
$request->flashOnly(['username', 'email']);
```

```
$request->flashExcept('password');
```

Entrée clignotante puis redirection

- Étant donné que vous souhaiterez souvent flasher l'entrée dans la session, puis rediriger vers la page précédente, vous pouvez facilement enchaîner l'entrée clignotante sur une redirection en utilisant la méthode `withInput` :

```
return redirect('form')->withInput();

return redirect()->route('user.create')->withInput();

return redirect('form')->withInput(
    $request->except('password')
);
```

05 - Manipulation des requêtes http :

Input

Récupération de l'ancienne entrée

- Pour récupérer l'entrée flashée de la requête précédente, appelez la méthode old sur une instance de Illuminate\Http\Request. La méthode old extraira les données d'entrée précédemment flashées de la session :

```
$username = $request->old('username');
```

- Laravel fournit également une aide old globale. Si vous affichez d'anciennes entrées dans un modèle Blade , il est plus pratique d'utiliser l'assistant old pour remplir à nouveau le formulaire. Si aucune ancienne entrée n'existe pour le champ donné, null sera renvoyé :

```
<input type="text" name="username" value="{{ old('username') }}">
```

05 - Manipulation des requêtes http :

Input



Récupération des cookies à partir des demandes

- Tous les cookies créés par le framework Laravel sont cryptés et signés avec un code d'authentification, ce qui signifie qu'ils seront considérés comme invalides s'ils ont été modifiés par le client. Pour récupérer une valeur de cookie à partir de la requête, utilisez la méthode cookie sur une instance Illuminate\Http\Request :

```
$value = $request->cookie('name');
```

Ajustement et normalisation d'entrée

- Par défaut, Laravel inclut le middleware `App\Http\Middleware\TrimStrings` et `App\Http\Middleware\ConvertEmptyStringsToNull` dans la pile middleware globale de votre application. Ces middlewares sont répertoriés dans la pile middleware globale par `App\Http\Kernel`. Ces middlewares enlèveront automatiquement tous les champs de chaîne entrants sur la demande, ainsi que convertiront tous les champs de chaîne vides en `null`. Cela vous permet de ne pas avoir à vous soucier de ces problèmes de normalisation dans vos routes et vos contrôleur.

Désactivation de la normalisation d'entrée

- Si vous souhaitez désactiver ce comportement pour toutes les requêtes, vous pouvez supprimer les deux middlewares de la pile middleware de votre application en les supprimant de la propriété `$middleware` de votre classe `App\Http\Kernel`.
- Si vous souhaitez désactiver le découpage des chaînes et la conversion des chaînes vides pour un sous-ensemble de requêtes adressées à votre application, vous pouvez utiliser la méthode `skipWhen` proposée par les deux middleware. Cette méthode accepte une fermeture qui doit renvoyer `true` ou `false` pour indiquer si la normalisation des entrées doit être ignorée. En règle générale, la méthode `skipWhen` doit être appelée dans la méthode `boot` du fichier `AppServiceProvider`.

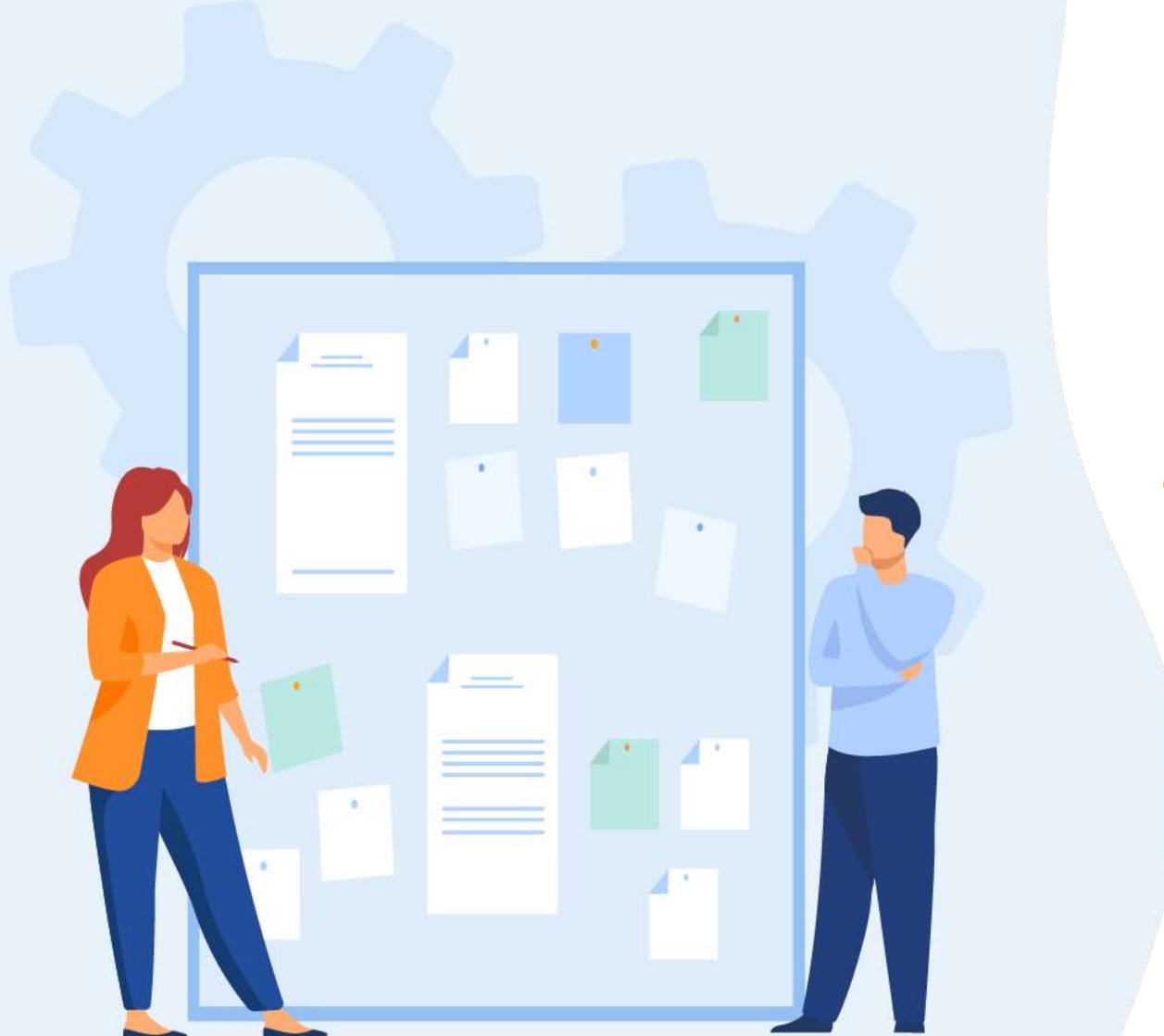
05 - Manipulation des requêtes http :

Input

```
use App\Http\Middleware\ConvertEmptyStringsToNull;
use App\Http\Middleware\TrimStrings;

/**
 * Bootstrap any application services.
 *
 * @return void
 */
public function boot()
{
    TrimStrings::skipWhen(function ($request) {
        return $request->is('admin/*');
    });

    ConvertEmptyStringsToNull::skipWhen(function ($request) {
        // ...
    });
}
```



CHAPITRE 5

Manipulation des requêtes HTTP

1. Introduction
2. Interaction avec les requêtes
3. Input
- 4. Fichiers**
5. Configuration des proxys de confiance
6. Configuration des hôtes approuvés

Récupération des fichiers téléchargés

- Vous pouvez récupérer des fichiers téléchargés à partir d'une instance Illuminate\Http\Request à l'aide de la méthode file ou à l'aide de propriétés dynamiques. La méthode file renvoie une instance de la classe Illuminate\Http\UploadedFile, qui étend la classe PHP SplFileInfo et fournit une variété de méthodes pour interagir avec le fichier :

```
$file = $request->file('photo');
```

```
$file = $request->photo;
```

- Vous pouvez déterminer si un fichier est présent sur la requête en utilisant la méthode hasFile :

```
if ($request->hasFile('photo')) {  
    //  
}
```

Validation des téléchargements réussis

- En plus de vérifier si le fichier est présent, vous pouvez vérifier qu'il n'y a eu aucun problème lors du téléchargement du fichier via la méthode isValid :

```
if ($request->file('photo')->isValid()) {  
    //  
}
```

Chemins de fichiers et extensions

- La classe UploadedFile contient également des méthodes pour accéder au chemin complet du fichier et à son extension. La méthode extension tentera de deviner l'extension du fichier en fonction de son contenu. Cette extension peut être différente de l'extension fournie par le client :

```
$path = $request->photo->path();  
  
$extension = $request->photo->extension();
```

05 - Manipulation des requêtes http : Fichiers

Autres méthodes de fichier

- Il existe une variété d'autres méthodes disponibles sur les instances UploadedFile.

Stockage des fichiers téléchargés

- Pour stocker un fichier téléchargé, vous utiliserez généralement l'un de vos systèmes de fichiers configurés . La classe UploadedFile a une méthode store qui déplacera un fichier téléchargé vers l'un de vos disques, qui peut être un emplacement sur votre système de fichiers local ou un emplacement de stockage en nuage.
- La méthode store accepte le chemin où le fichier doit être stocké par rapport au répertoire racine configuré du système de fichiers. Ce chemin ne doit pas contenir de nom de fichier, car un identifiant unique sera automatiquement généré pour servir de nom de fichier.
- La méthode store accepte également un deuxième argument facultatif pour le nom du disque qui doit être utilisé pour stocker le fichier. La méthode renverra le chemin du fichier par rapport à la racine du disque :

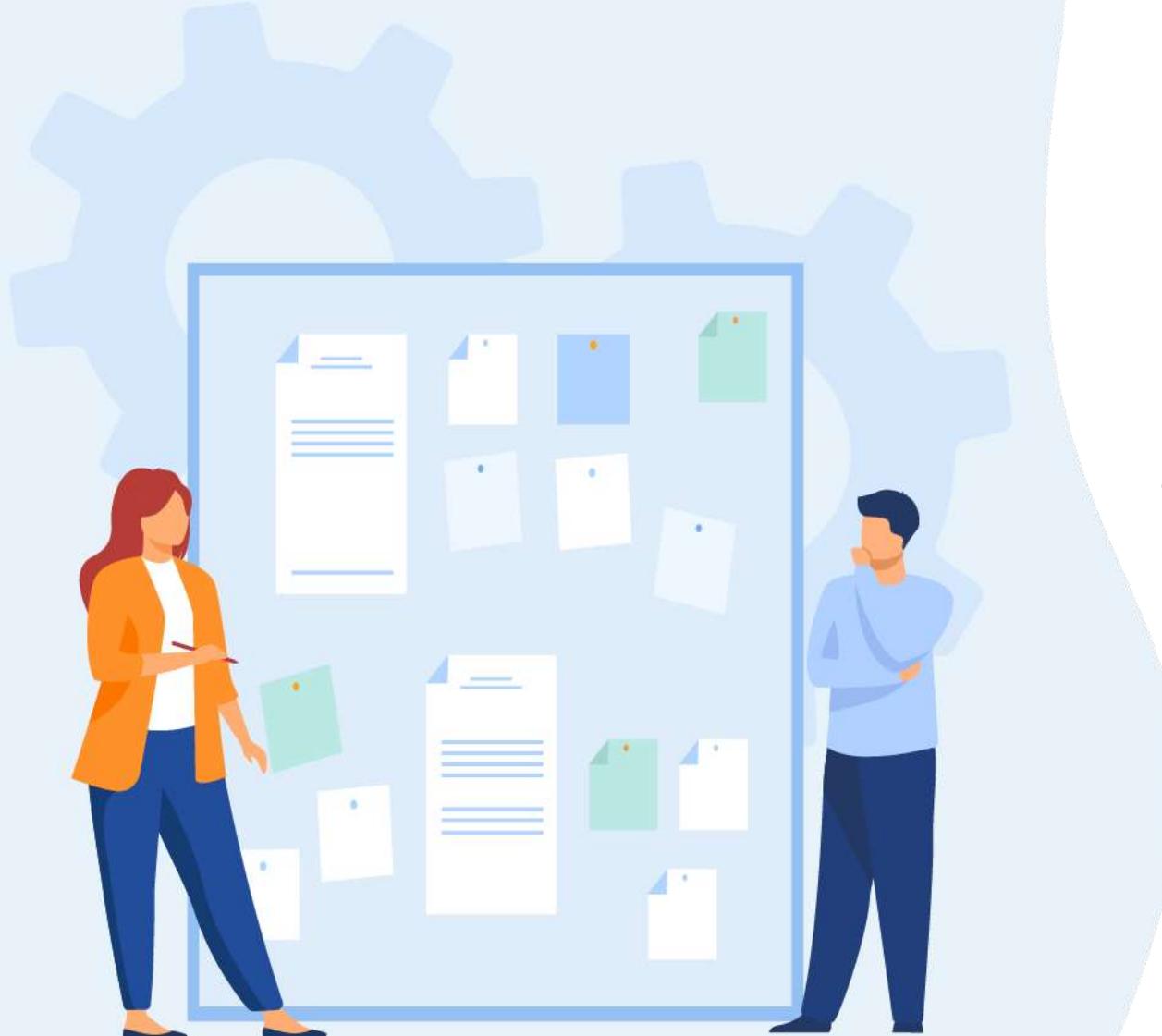
```
$path = $request->photo->store('images');
```

```
$path = $request->photo->store('images', 's3');
```

- Si vous ne souhaitez pas qu'un nom de fichier soit généré automatiquement, vous pouvez utiliser la méthode storeAs, qui accepte le chemin, le nom de fichier et le nom du disque comme arguments :

```
$path = $request->photo->storeAs('images', 'filename.jpg');
```

```
$path = $request->photo->storeAs('images', 'filename.jpg', 's3');
```



CHAPITRE 5

Manipulation des requêtes HTTP

1. Introduction
2. Interaction avec les requêtes
3. Input
4. Fichiers
5. **Configuration des proxys de confiance**
6. Configuration des hôtes approuvés

05 - Manipulation des requêtes http :

Configuration des proxys de confiance



- Lorsque vous exécutez vos applications derrière un équilibrEUR de charge avec les certificats TLS/SSL, vous remarquerez peut-être que votre application ne génère parfois pas de liens HTTPS lors de l'utilisation de l'assistant url. Cela est généralement dû au fait que votre application transfère le trafic de votre équilibrEUR de charge sur le port 80 et ne sait pas qu'elle doit générer des liens sécurisés.
- Pour résoudre ce problème, vous pouvez utiliser le middleware App\Http\Middleware\TrustProxies inclus dans votre application Laravel, qui vous permet de personnaliser rapidement les équilibrEURS de charge ou les proxys auxquels votre application doit faire confiance. Vos proxys de confiance doivent être répertoriés sous forme de tableau sur la propriété \$proxies de ce middleware. Outre la configuration des proxys de confiance, vous pouvez configurer le proxy \$headers qui doit être approuvé :
-

05 - Manipulation des requêtes http :

Configuration des proxys de confiance



```
<?php

namespace App\Http\Middleware;

use Illuminate\Http\Middleware\TrustProxies as
Middleware;
use Illuminate\Http\Request;

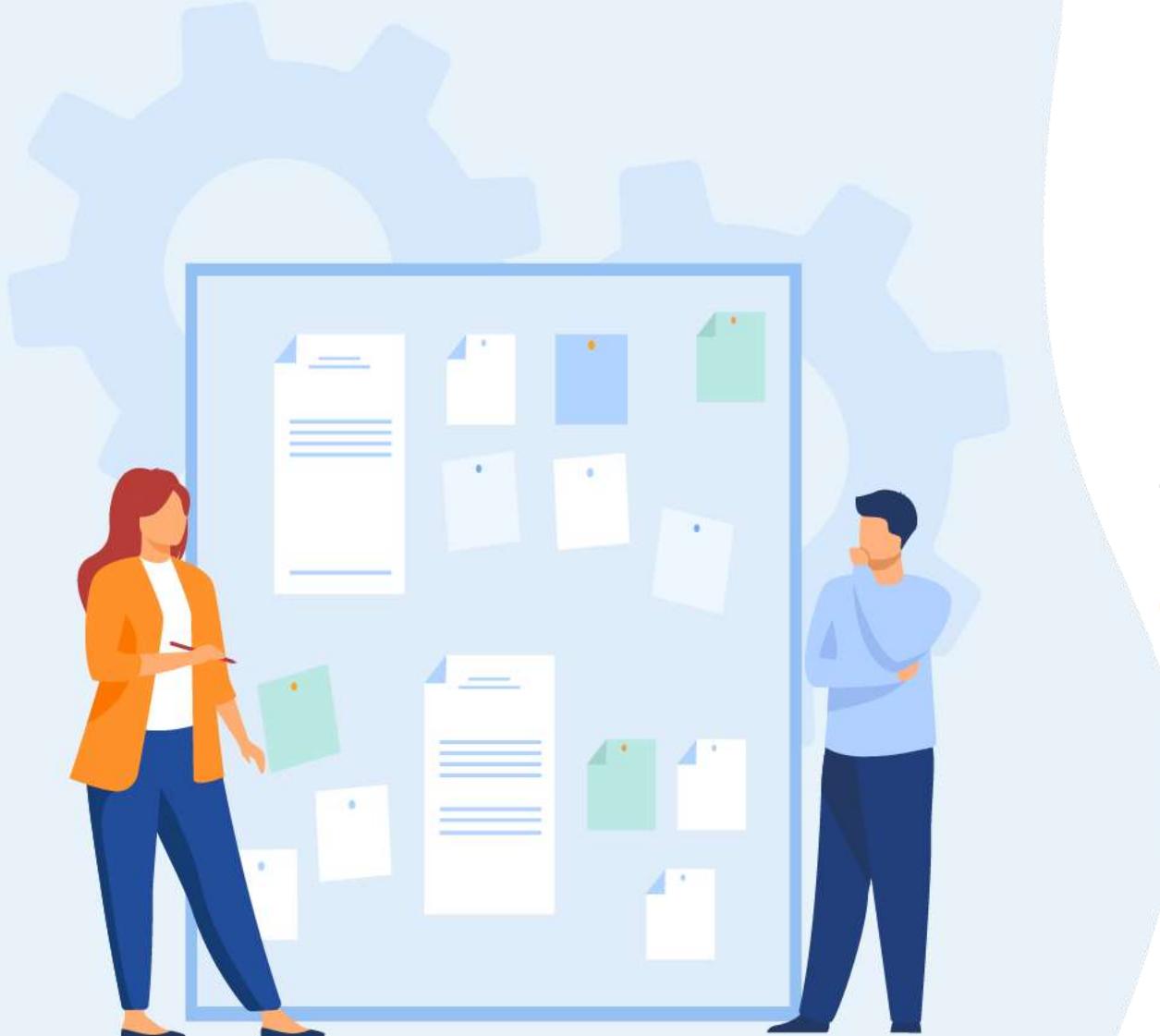
class TrustProxies extends Middleware
{
    /**
     * The trusted proxies for this application.
     *
     * @var string|array
     */
    protected $proxies = [
        '192.168.1.1',
        '192.168.1.2',
    ];

    /**
     * The headers that should be used to detect proxies.
     *
     * @var int
     */
    protected $headers =
Request::HEADER_X_FORWARDED_FOR |
Request::HEADER_X_FORWARDED_HOST |
Request::HEADER_X_FORWARDED_PORT |
Request::HEADER_X_FORWARDED_PROTO;
}
```

Faire confiance à tous les proxys

- Si vous utilisez un fournisseur d'équilibreur de charge "cloud", vous ne connaissez peut-être pas les adresses IP de vos équilibriseurs réels. Dans ce cas, vous pouvez utiliser * pour faire confiance à tous les proxy :

```
/**  
 * The trusted proxies for this application.  
 *  
 * @var string|array  
 */  
protected $proxies = '*';
```



CHAPITRE 5

Manipulation des requêtes HTTP

1. Introduction
2. Interaction avec les requêtes
3. Input
4. Fichiers
5. Configuration des proxys de confiance
- 6. Configuration des hôtes approuvés**

05 - Manipulation des requêtes http :

Configuration des hôtes approuvés



- Par défaut, Laravel répondra à toutes les requêtes qu'il reçoit, quel que soit le contenu de l'en-tête de la requête HTTP Host. De plus, la valeur Host de l'en-tête sera utilisée lors de la génération d'URL absolues vers votre application lors d'une requête Web.
- En règle générale, vous devez configurer votre serveur Web, tel que Nginx ou Apache, pour n'envoyer à votre application que les requêtes qui correspondent à un nom d'hôte donné. Cependant, si vous n'avez pas la possibilité de personnaliser directement votre serveur Web et que vous devez demander à Laravel de ne répondre qu'à certains noms d'hôte, vous pouvez le faire en activant le middleware App\Http\Middleware\TrustHosts pour votre application.

05 - Manipulation des requêtes http :

Configuration des hôtes approuvés



- Le middleware TrustHosts est déjà inclus dans la pile \$middleware de votre application ; cependant, vous devez le décommenter pour qu'il devienne actif. Dans la méthode de ce middleware hosts, vous pouvez spécifier les noms d'hôte auxquels votre application doit répondre. Les requêtes entrantes avec d'autres en-têtes Host de valeur seront rejetées :

```
/**  
 * Get the host patterns that should be trusted.  
 *  
 * @return array  
 */  
public function hosts()  
{  
    return [  
        'laravel.test',  
  
        $this->allSubdomainsOfApplicationUrl(),  
    ];  
}
```

La méthode allSubdomainsOfApplicationUrl d'aide renverra une expression régulière correspondant à tous les sous-domaines de la valeur app.url de configuration de votre application. Cette méthode d'aide fournit un moyen pratique d'autoriser tous les sous-domaines de votre application lors de la création d'une application qui utilise des sous-domaines génériques.

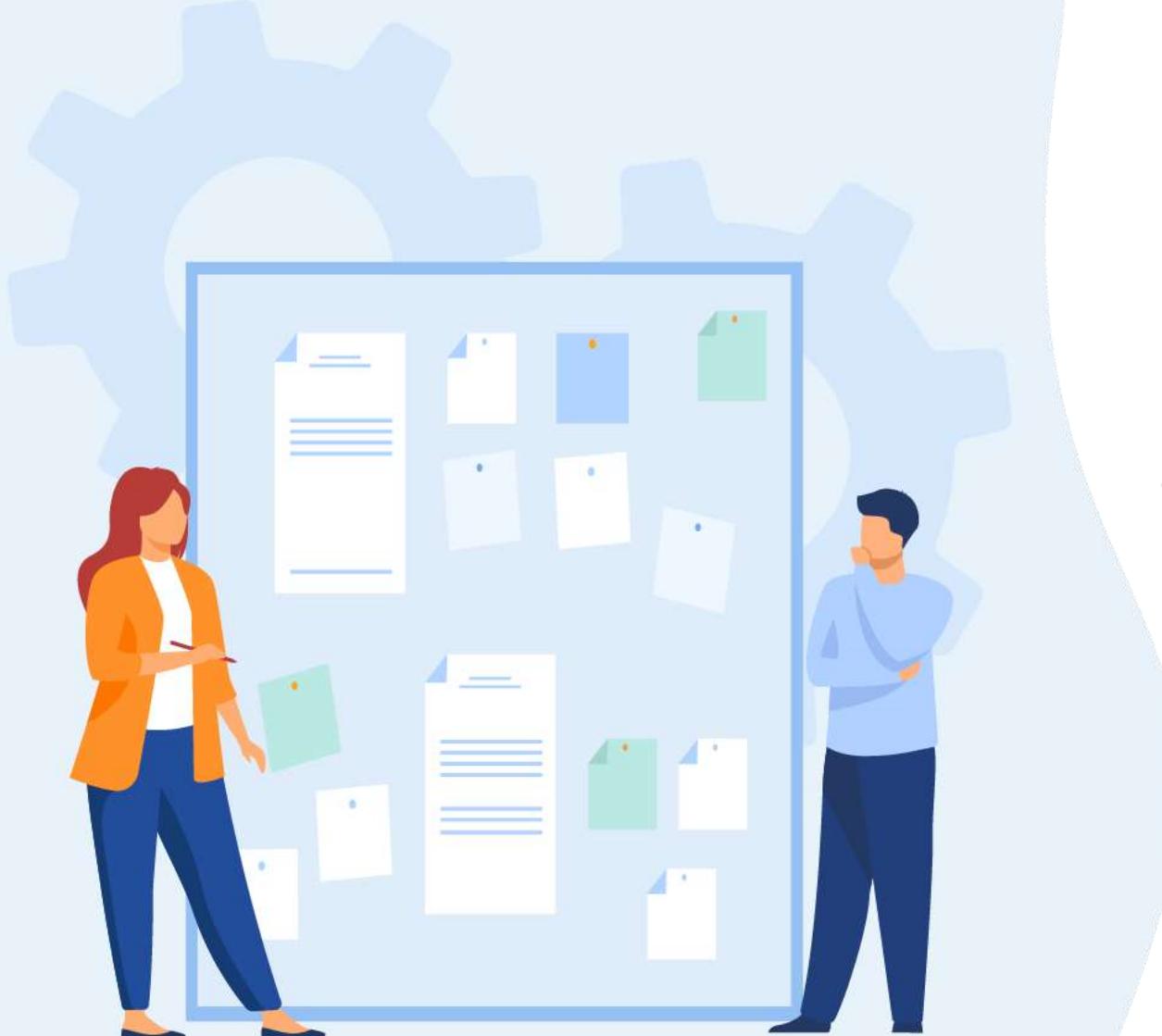


CHAPITRE 6

Manipulation des réponses http

Ce que vous allez apprendre dans ce chapitre :

- Création
- Redirection
- Types de réponses
- Réponse Marco



CHAPITRE 6

Manipulation des réponses HTTP

1. **Création**
2. Redirection
3. Types de réponses
4. Réponse Marco

Chaînes et tableaux

- Toutes les routes et tous les contrôleurs doivent renvoyer une réponse à renvoyer au navigateur de l'utilisateur. Laravel propose plusieurs façons différentes de renvoyer des réponses. La réponse la plus basique consiste à renvoyer une chaîne à partir d'une route ou d'un contrôleur. Le framework convertira automatiquement la chaîne en une réponse HTTP complète :

```
Route::get('/', function () {
    return 'Hello World';
});
```

- En plus de renvoyer des chaînes à partir de vos routes et de vos contrôleurs, vous pouvez également renvoyer des tableaux. Le framework convertira automatiquement le tableau en une réponse JSON :

```
Route::get('/', function () {
    return [1, 2, 3];
});
```

Objets de réponse

- En règle générale, vous ne renverrez pas simplement de simples chaînes ou des tableaux à partir de vos actions de routage. Au lieu de cela, vous renverrez des instances Illuminate\Http\Response ou des vues complètes .
- Le renvoi d'une instance complète Response vous permet de personnaliser le code d'état HTTP et les en-têtes de la réponse. Une instance Response hérite de la classe Symfony\Component\HttpFoundation\Response, qui fournit diverses méthodes pour créer des réponses HTTP :

```
Route::get('/home', function () {
    return response('Hello World', 200)
        ->header('Content-Type', 'text/plain');
});
```

Modèles et collections éloquentes

- Vous pouvez également renvoyer des modèles et des collections ORM Eloquent directement à partir de vos routes et de vos contrôleurs. Lorsque vous le faites, Laravel convertira automatiquement les modèles et collections en réponses JSON tout en respectant les attributs cachés du modèle :

```
use App\Models\User;

Route::get('/user/{user}', function (User $user) {
    return $user;
});
```

Attacher des en-têtes aux réponses

- Gardez à l'esprit que la plupart des méthodes de réponse peuvent être chaînées, ce qui permet la construction fluide d'instances de réponse. Par exemple, vous pouvez utiliser la méthode header pour ajouter une série d'en-têtes à la réponse avant de la renvoyer à l'utilisateur :

```
return response($content)
    ->header('Content-Type', $type)
    ->header('X-Header-One', 'Header Value')
    ->header('X-Header-Two', 'Header Value');
```

- Ou, vous pouvez utiliser la méthode withHeaders pour spécifier un tableau d'en-têtes à ajouter à la réponse :

```
return response($content)
    ->withHeaders([
        'Content-Type' => $type,
        'X-Header-One' => 'Header Value',
        'X-Header-Two' => 'Header Value',
    ]);
```

Middleware de contrôle du cache

- Laravel inclut un middleware `cache.headers`, qui peut être utilisé pour définir rapidement l'en-tête Cache-Control d'un groupe de routes. Les directives doivent être fournies en utilisant l'équivalent "snake case" de la directive `cache-control` correspondante et doivent être séparées par un point-virgule. Si `etag` est spécifié dans la liste des directives, un hachage MD5 du contenu de la réponse sera automatiquement défini comme identifiant ETag :

```
Route::middleware('cache.headers:public;max_age=2628000;etag')->group(function () {  
    Route::get('/privacy', function () {  
        // ...  
    });  
  
    Route::get('/terms', function () {  
        // ...  
    });  
});
```

Attacher des cookies aux réponses

- Vous pouvez attacher un cookie à une instance Illuminate\Http\Response sortante à l'aide de la méthode cookie. Vous devez transmettre le nom, la valeur et le nombre de minutes pendant lesquelles le cookie doit être considéré comme valide à cette méthode

```
return response('Hello World')->cookie(  
    'name', 'value', $minutes  
)
```

- La méthode cookie accepte également quelques arguments supplémentaires qui sont utilisés moins fréquemment. Généralement, ces arguments ont le même but et la même signification que les arguments qui seraient donnés à la méthode native setcookie de PHP :

```
return response('Hello World')->cookie(  
    'name', 'value', $minutes, $path, $domain, $secure, $httpOnly  
)
```

06 - Manipulation des réponses HTTP : Création



- Si vous souhaitez vous assurer qu'un cookie est envoyé avec la réponse sortante mais que vous n'avez pas encore d'instance de cette réponse, vous pouvez utiliser la façade Cookie pour "mettre en file d'attente" les cookies à attacher à la réponse lorsqu'elle est envoyée. La méthode queue accepte les arguments nécessaires pour créer une instance de cookie. Ces cookies seront joints à la réponse sortante avant qu'elle ne soit envoyée au navigateur :

```
use Illuminate\Support\Facades\Cookie;  
  
Cookie::queue('name', 'value', $minutes);
```

Génération d'instances de cookies

- Si vous souhaitez générer une instance Symfony\Component\HttpFoundation\Cookie pouvant être attachée à une instance de réponse ultérieurement, vous pouvez utiliser l'assistant cookie global. Ce cookie ne sera pas renvoyé au client sauf s'il est attaché à une instance de réponse :

```
$cookie = cookie('name', 'value', $minutes);  
  
return response('Hello World')->cookie($cookie);
```

Expiration anticipée des cookies

- Vous pouvez supprimer un cookie en le faisant expirer via la méthode `withoutCookie` d'une réponse sortante :

```
return response('Hello World')->withoutCookie('name');
```

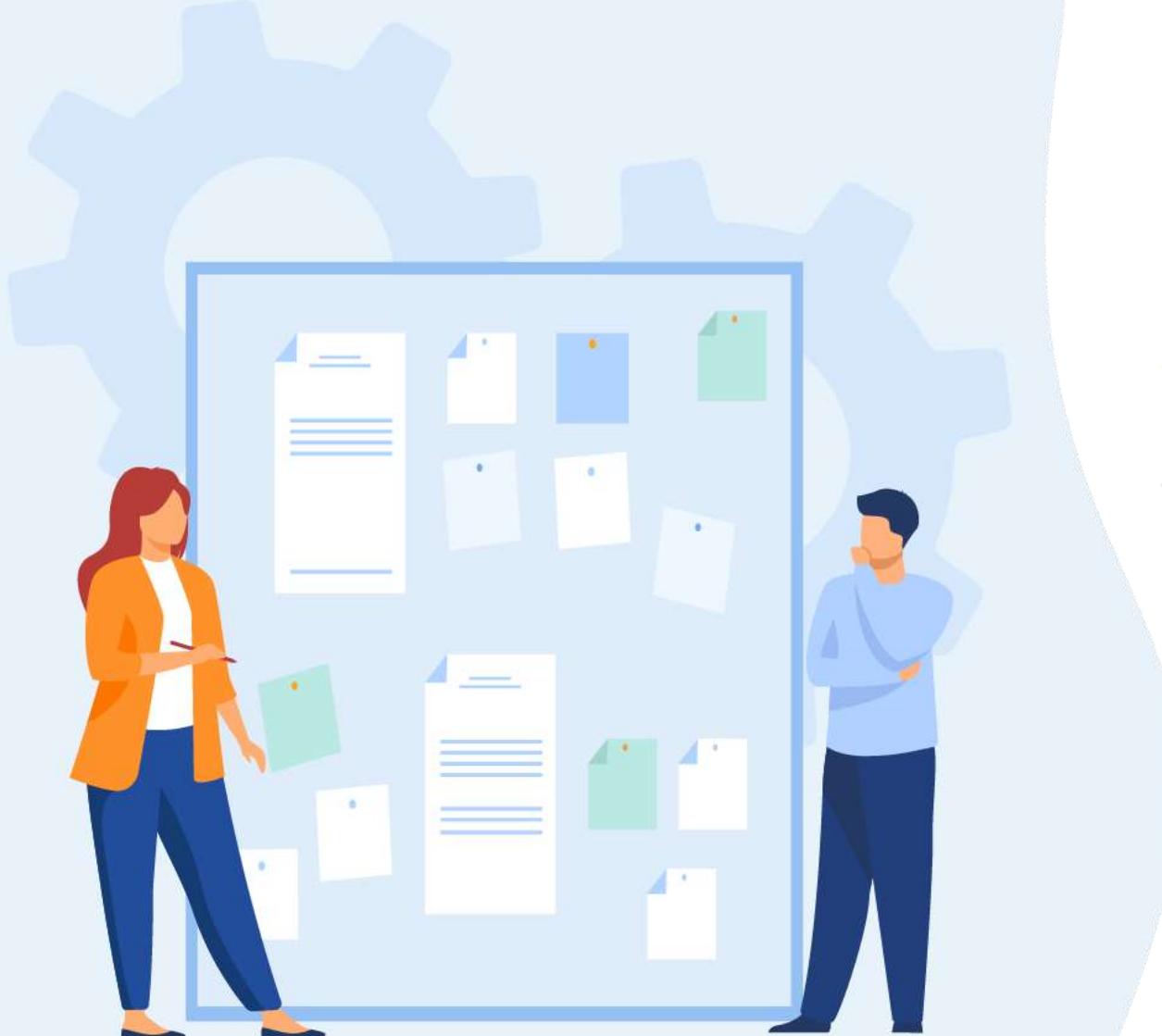
- Si vous n'avez pas encore d'instance de la réponse sortante, vous pouvez utiliser la méthode `expire` de la façade `Cookie` pour faire expirer un cookie :

```
Cookie::expire('name');
```

Cookies et cryptage

- Par défaut, tous les cookies générés par Laravel sont cryptés et signés afin qu'ils ne puissent pas être modifiés ou lus par le client. Si vous souhaitez désactiver le chiffrement pour un sous-ensemble de cookies générés par votre application, vous pouvez utiliser la propriété `$except` du middleware `App\Http\Middleware\EncryptCookies`, qui se trouve dans le répertoire `app/Http/Middleware` :

```
/**  
 * Les noms de cookies qui ne doivent pas être cryptés :  
 *  
 * @var array  
 */  
protected $except = [  
    'cookie_name',  
];
```



CHAPITRE 6

Manipulation des réponses HTTP

1. Création
2. **Redirection**
3. Types de réponses
4. Réponse Marco

Introduction

Les réponses de redirection sont des instances de la classe `Illuminate\Http\RedirectResponse` et contiennent les en-têtes appropriés nécessaires pour rediriger l'utilisateur vers une autre URL. Il existe plusieurs façons de générer une instance `RedirectResponse`. La méthode la plus simple consiste à utiliser l'assistant global `redirect` :

```
Route::get('/dashboard', function () {
    return redirect('home/dashboard');
});
```

Parfois, vous souhaiterez peut-être rediriger l'utilisateur vers son emplacement précédent, par exemple lorsqu'un formulaire soumis n'est pas valide. Vous pouvez le faire en utilisant la fonction d'assistance globale `back`. Étant donné que cette fonctionnalité utilise la `session`, assurez-vous que la route appelant la fonction `back` utilise le groupe middleware `web` :

```
Route::post('/user/profile', function () {
    // Validate the request...

    return back()->withInput();
});
```

Redirection vers des routes nommées

- Lorsque vous appelez l'assistant redirect sans paramètre, une instance de Illuminate\Routing\Redirector est renvoyée, vous permettant d'appeler n'importe quelle méthode sur l'instance Redirector. Par exemple, pour générer un RedirectResponse vers une route nommée, vous pouvez utiliser la méthode route :

```
return redirect()->route('login');
```

- Si votre route a des paramètres, vous pouvez les passer comme deuxième argument à la méthode route :

```
// pour une route avec l'URI suivant: /profile/{id}  
  
return redirect()->route('profile', ['id' => 1]);
```

06 - Manipulation des réponses HTTP : Redirections



Remplir les paramètres via des modèles éloquents

- Si vous redirigez vers une route avec un paramètre "ID" qui est renseigné à partir d'un modèle Eloquent, vous pouvez transmettre le modèle lui-même. L'ID sera extrait automatiquement :

```
// pour une route avec l'URI suivant : /profile/{id}  
  
return redirect()->route('profile', [$user]);
```

06 - Manipulation des réponses HTTP : Redirections



- Si vous souhaitez personnaliser la valeur placée dans le paramètre route, vous pouvez spécifier la colonne dans la définition du paramètre route (`/profile/{id:slug}`) ou vous pouvez remplacer la méthode `getRouteKey` sur votre modèle Eloquent :

```
/**  
 * Obtenir la valeur de la clé du modèle route.  
 *  
 * @return mixed  
 */  
public function getRouteKey()  
{  
    return $this->slug;  
}
```

Redirection vers les actions du contrôleur

- Vous pouvez également générer des redirections vers [les actions du contrôleur](#). Pour ce faire, passez le contrôleur et le nom de l'action à la méthode action :

```
use App\Http\Controllers\UserController;  
  
return redirect()->action([UserController::class, 'index']);
```

- Si votre route de contrôleur nécessite des paramètres, vous pouvez les passer comme deuxième argument à la méthode action :

```
return redirect()->action(  
    [UserController::class, 'profile'], ['id' => 1]  
)
```

Redirection vers des domaines externes

- Parfois, vous devrez peut-être rediriger vers un domaine en dehors de votre application. Vous pouvez le faire en appelant la méthode `away`, qui crée un `RedirectResponse` sans aucun codage, validation ou vérification d'URL supplémentaire :

```
return redirect()->away('https://www.google.com');
```

Redirection avec des données de session flashées

- La redirection vers une nouvelle URL et le flashage des données vers la session sont généralement effectués en même temps. En règle générale, cela se fait après avoir effectué une action avec succès lorsque vous envoyez un message de réussite à la session. Pour plus de commodité, vous pouvez créer une instance RedirectResponse et envoyer des données flash à la session dans une seule chaîne de méthodes fluide :

```
Route::post('/user/profile', function () {
    // ...

    return redirect('dashboard')->with('status', 'Profile updated!');
});
```

06 - Manipulation des réponses HTTP : Redirections



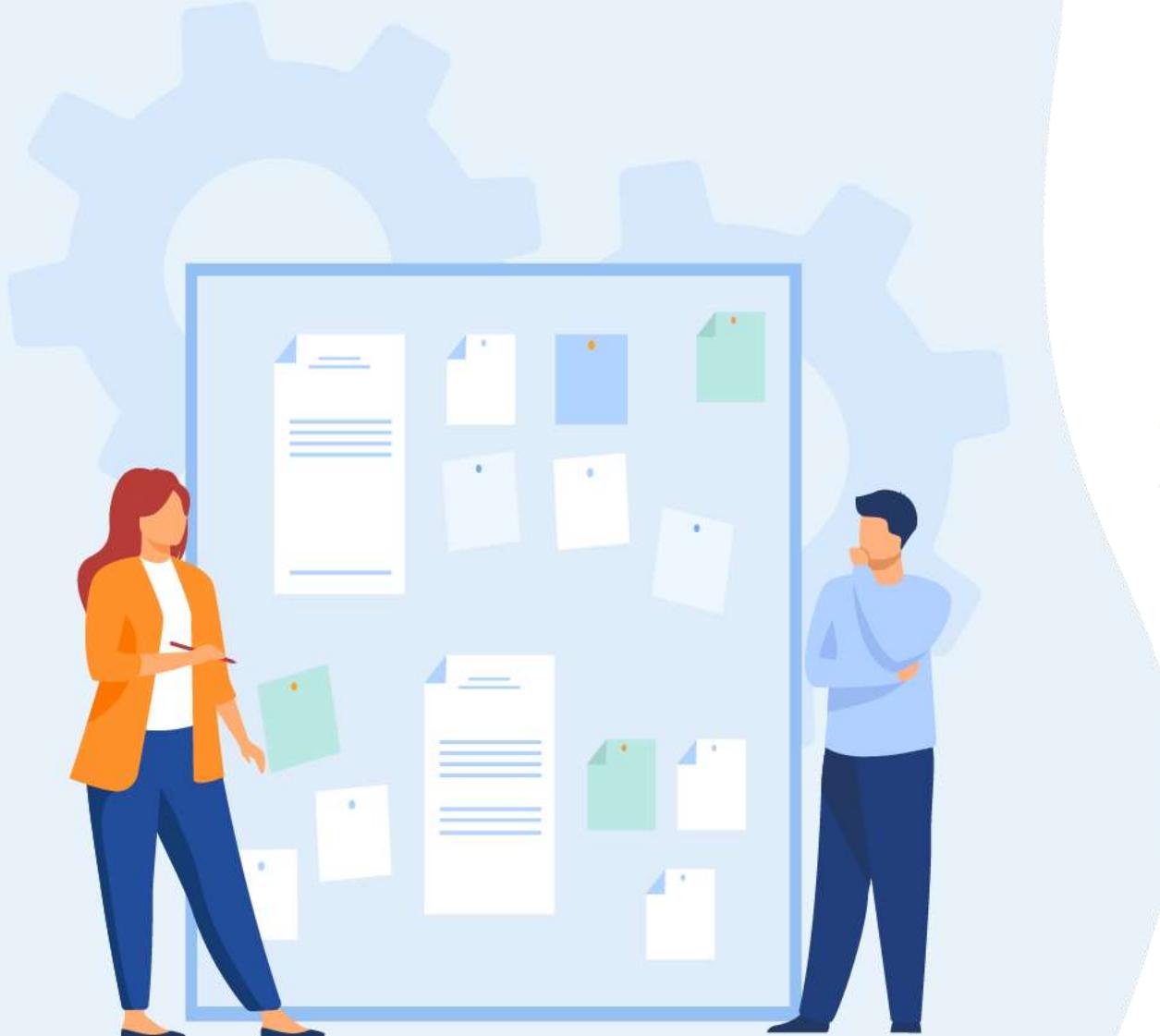
- Une fois l'utilisateur redirigé, vous pouvez afficher le message flashé de la session . Par exemple, en utilisant la syntaxe Blade :

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

Redirection avec entrée

- Vous pouvez utiliser la méthode `withInput` fournie par l'instance `RedirectResponse` pour flasher les données d'entrée de la demande actuelle dans la session avant de rediriger l'utilisateur vers un nouvel emplacement. Cela se fait généralement si l'utilisateur a rencontré une erreur de validation. Une fois l'entrée flashée à la session, vous pourrez facilement la récupérer lors de la prochaine requête pour repeupler le formulaire :

```
return back()->withInput();
```



CHAPITRE 6

Manipulation des réponses HTTP

1. Création
2. Redirection
3. **Types de réponses**
4. Réponse Marco

Introduction

- L'assistant response peut être utilisé pour générer d'autres types d'instances de réponse. Lorsque l'assistant response est appelé sans arguments, une implémentation du contrat Illuminate\Contracts\Routing\ResponseFactory est renvoyée. Ce contrat fournit plusieurs méthodes utiles pour générer des réponses.

Afficher les réponses

- Si vous avez besoin de contrôler le statut et les en-têtes de la réponse, mais que vous devez également renvoyer une vue comme contenu de la réponse, vous devez utiliser la méthode view :

```
return response()  
    ->view('hello', $data, 200)  
    ->header('Content-Type', $type);
```

- Bien entendu, si vous n'avez pas besoin de transmettre un code d'état HTTP personnalisé ou des en-têtes personnalisés, vous pouvez utiliser la fonction d'assistance globale view.

Réponses JSON

- La méthode json définira automatiquement l'en-tête Content-Type sur application/json, ainsi que convertira le tableau donné en JSON à l'aide de la fonction json_encode PHP :

```
return response()->json([
    'name' => 'Abigail',
    'state' => 'CA',
]);
```

- Si vous souhaitez créer une réponse JSONP, vous pouvez utiliser la méthode json en combinaison avec la méthode withCallback :

```
return response()
    ->json(['name' => 'Abigail', 'state' => 'CA'])
    ->withCallback($request->input('callback'));
```

Téléchargements de fichiers

- La méthode `download` peut être utilisée pour générer une réponse qui force le navigateur de l'utilisateur à télécharger le fichier au chemin donné. La méthode `download` accepte un nom de fichier comme deuxième argument de la méthode, qui déterminera le nom de fichier qui est vu par l'utilisateur téléchargeant le fichier. Enfin, vous pouvez passer un tableau d'en-têtes HTTP comme troisième argument à la méthode :

```
return response()->download($pathToFile);  
  
return response()->download($pathToFile, $name, $headers);
```

! Symfony HttpFoundation, qui gère les téléchargements de fichiers, exige que le fichier en cours de téléchargement ait un nom de fichier ASCII.

Téléchargements en streaming

- Parfois, vous souhaiterez peut-être transformer la réponse de chaîne d'une opération donnée en une réponse téléchargeable sans avoir à écrire le contenu de l'opération sur le disque. Vous pouvez utiliser la méthode `streamDownload` dans ce scénario. Cette méthode accepte un rappel, un nom de fichier et un tableau facultatif d'en-têtes comme arguments :

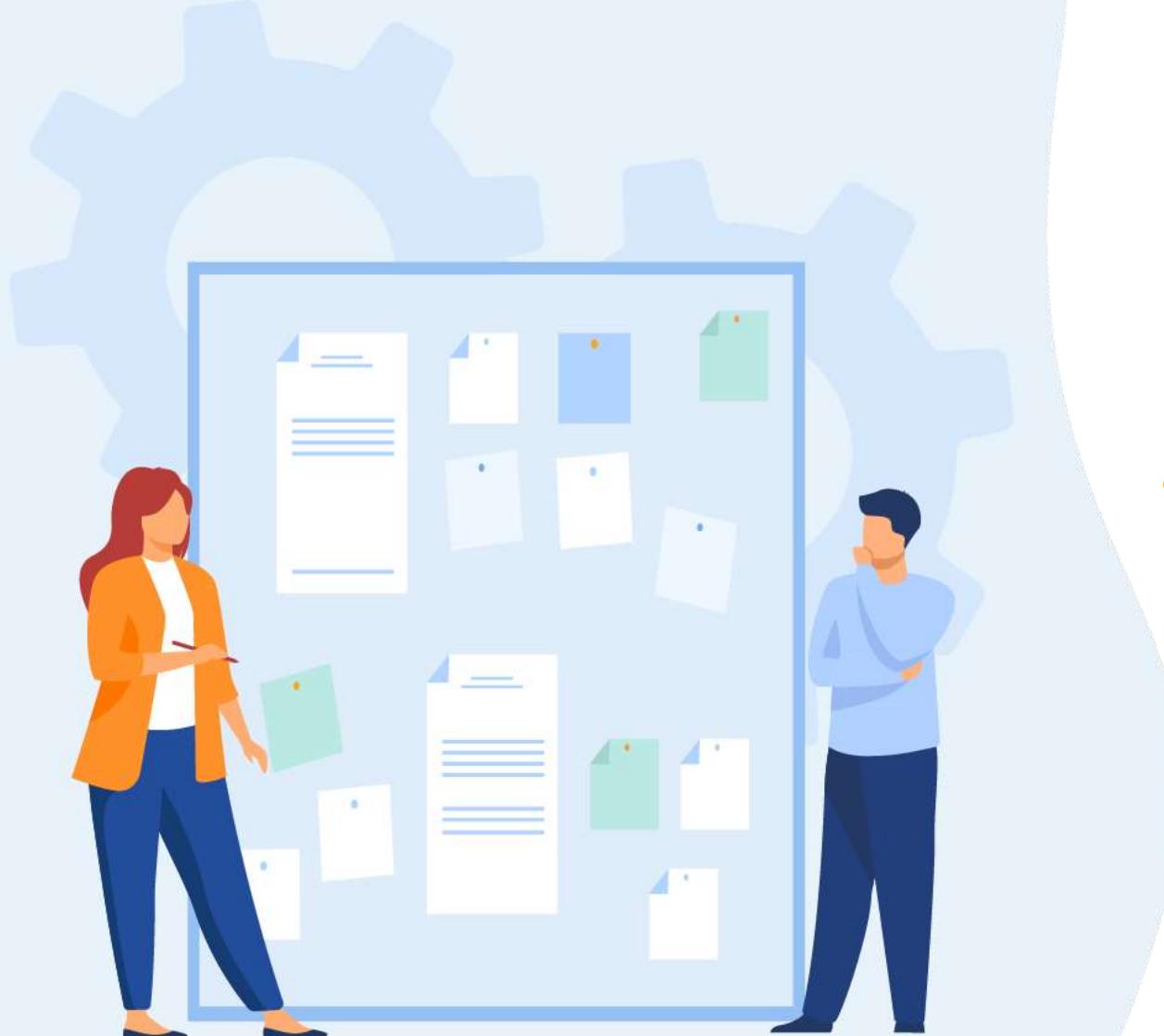
```
use App\Services\GitHub;

return response()->streamDownload(function () {
    echo GitHub::api('repo')
        ->contents()
        ->readme('laravel', 'laravel')['contents'];
}, 'laravel-readme.md');
```

Fichier de réponses

- La méthode file peut être utilisée pour afficher un fichier, tel qu'une image ou un PDF, directement dans le navigateur de l'utilisateur au lieu de lancer un téléchargement. Cette méthode accepte le chemin d'accès au fichier comme premier argument et un tableau d'en-têtes comme deuxième argument :

```
return response()->file($pathToFile);  
  
return response()->file($pathToFile, $headers);
```



CHAPITRE 6

Manipulation des réponses HTTP

1. Création
2. Redirection
3. Types de réponses
4. Réponse Marco

Introduction

- Si vous souhaitez définir une réponse personnalisée que vous pouvez réutiliser dans une variété de vos routes et contrôleurs, vous pouvez utiliser la méthode macro sur la façade Response. En règle générale, vous devez appeler cette méthode à partir de la méthode boot de l'un des fournisseurs de services de votre application , tel que le fournisseur de services : App\Providers\AppServiceProvider

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Response;
use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider
{
    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Response::macro('caps', function ($value) {
            return Response::make(strtoupper($value));
        });
    }
}
```

06 - Manipulation des réponses HTTP : Réponse Marco



- La fonction macro accepte un nom comme premier argument et une fermeture comme deuxième argument. La fermeture de la macro sera exécutée lors de l'appel du nom de la macro depuis une implémentation ResponseFactory ou le responsehelper :

```
return response()->caps('foo');
```

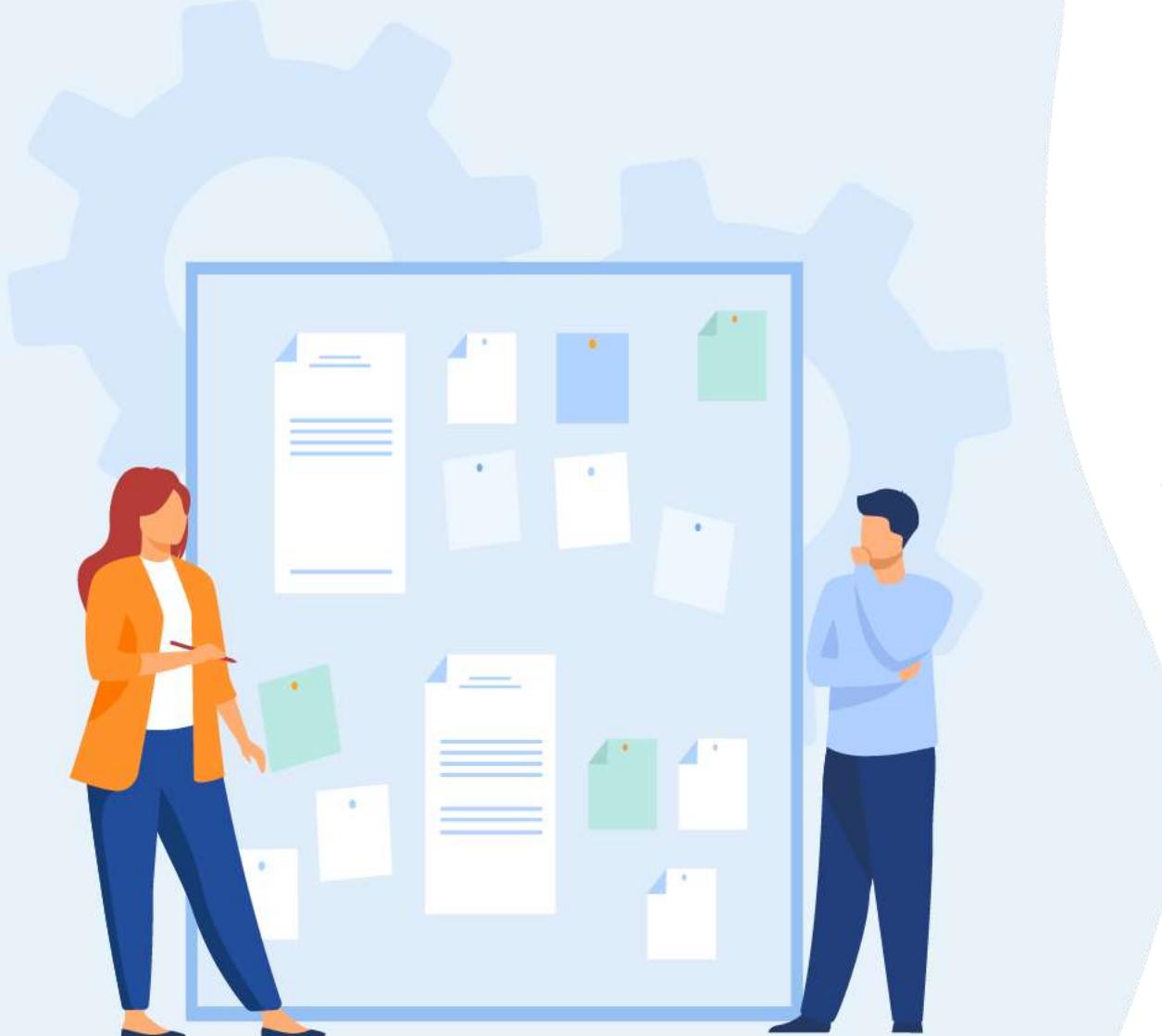


CHAPITRE 7

Manipulation des vues

Ce que vous allez apprendre dans ce chapitre :

- Création
- Transmission des données
- Affichage des compositeurs
- Optimisation des vues



CHAPITRE 7

Manipulation des vues

1. **Création**
2. Transmission des données
3. Affichage des compositeurs
4. Optimisation des vues

Introduction

- Bien sûr, il n'est pas pratique de renvoyer des chaînes de documents HTML entières directement à partir de vos routes et de vos contrôleurs. Heureusement, les vues offrent un moyen pratique de placer tout notre code HTML dans des fichiers séparés. Les vues séparent la logique de votre contrôleur/application de votre logique de présentation et sont stockées dans le répertoire resources/views. Une vue simple pourrait ressembler à ceci :

```
<!-- Vue stockée dans resources/views/greeting.blade.php -->

<html>
  <body>
    <h1>Hello, {{ $name }}</h1>
  </body>
</html>
```

07 - Manipulation des vues :

Création



- Étant donné que cette vue est stockée dans resources/views/greeting.blade.php, nous pouvons la renvoyer en utilisant l'assistant global view comme ceci :

```
Route::get('/', function () {
    return view('greeting', ['name' => 'James']);
});
```

Création et rendu des vues

- Vous pouvez créer une vue en plaçant un fichier avec l'extension .blade.php dans le répertoire resources/views de votre application. L'extension .blade.php informe le framework que le fichier contient un modèle Blade . Les modèles de lame contiennent du HTML ainsi que des directives Blade qui vous permettent de faire facilement écho des valeurs, de créer des instructions "if", d'itérer sur les données, etc.
- Une fois que vous avez créé une vue, vous pouvez la renvoyer à partir de l'une des routes ou des contrôleurs de votre application à l'aide de l'assistant global view:

```
Route::get('/', function () {
    return view('greeting', ['name' => 'James']);
});
```

07 - Manipulation des vues :

Création



- Les vues peuvent également être renvoyées à l'aide de la façade View :

```
use Illuminate\Support\Facades\View;  
  
return View::make('greeting', ['name' => 'James']);
```

- Comme vous pouvez le voir, le premier argument passé au viewhelper correspond au nom du fichier de vue dans le répertoire resources/views. Le deuxième argument est un tableau de données qui doivent être mises à la disposition de la vue. Dans ce cas, nous transmettons la variable name, qui est affichée dans la vue en utilisant la syntaxe Blade .

Répertoires de vue imbriqués

- Les vues peuvent également être imbriquées dans des sous-répertoires du répertoire resources/views. La notation "Point" peut être utilisée pour référencer des vues imbriquées. Par exemple, si votre vue est stockée dans resources/views/admin/profile.blade.php, vous pouvez la renvoyer depuis l'une des routes/contrôleurs de votre application comme suit :

```
return view('admin.profile', $data);
```

Création de la première vue disponible

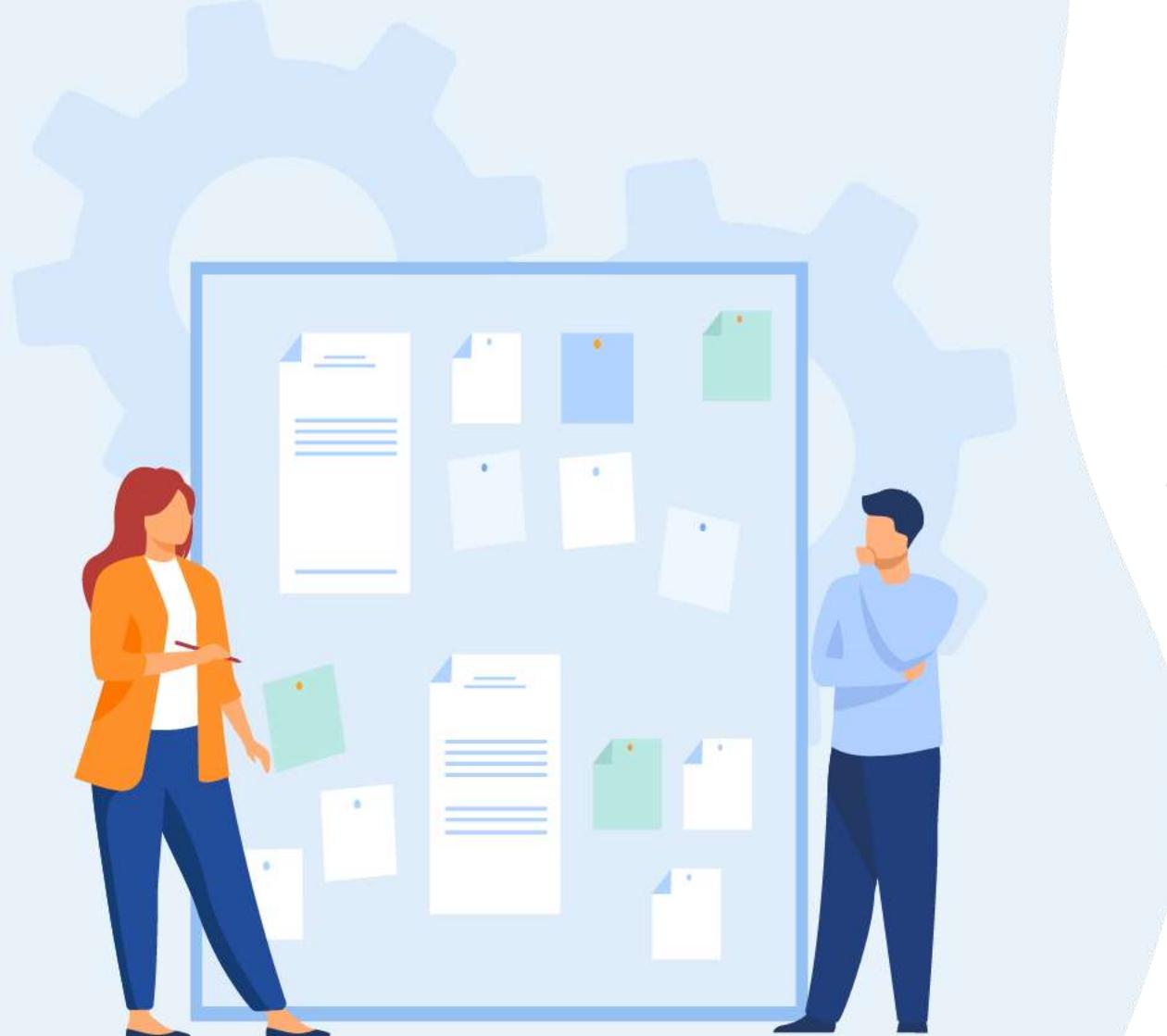
- En utilisant la méthode View de la façade first, vous pouvez créer la première vue qui existe dans un tableau de vues donné. Cela peut être utile si votre application ou votre package permet de personnaliser ou d'écraser les vues :

```
use Illuminate\Support\Facades\View;  
  
return View::first(['custom.admin', 'admin'], $data);
```

Déterminer si une vue existe

- Si vous avez besoin de déterminer si une vue existe, vous pouvez utiliser la façade View. La méthode exists retournera true si la vue existe :

```
use Illuminate\Support\Facades\View;  
  
if (View::exists('emails.customer')) {  
    //  
}
```



CHAPITRE 7

Manipulation des vues

1. Création
2. **Transmission des données**
3. Affichage des compositeurs
4. Optimisation des vues

Introduction

- Comme vous l'avez vu dans les exemples précédents, vous pouvez passer un tableau de données aux vues pour rendre ces données disponibles à la vue :

```
return view('greetings', ['name' => 'Victoria']);
```

- Lors de la transmission d'informations de cette manière, les données doivent être un tableau avec des paires clé/valeur. Après avoir fourni des données à une vue, vous pouvez ensuite accéder à chaque valeur de votre vue à l'aide des clés de données, telles que <?php echo \$name; ?>.

- Au lieu de transmettre un tableau complet de données à la fonction d'assistance view, vous pouvez utiliser la méthode with pour ajouter des éléments de données individuels à la vue. La méthode with renvoie une instance de l'objet view afin que vous puissiez continuer à enchaîner les méthodes avant de renvoyer la vue :

```
return view('greeting')
    ->with('name', 'Victoria')
    ->with('occupation', 'Astronaut');
```

Partage de données avec toutes les vues

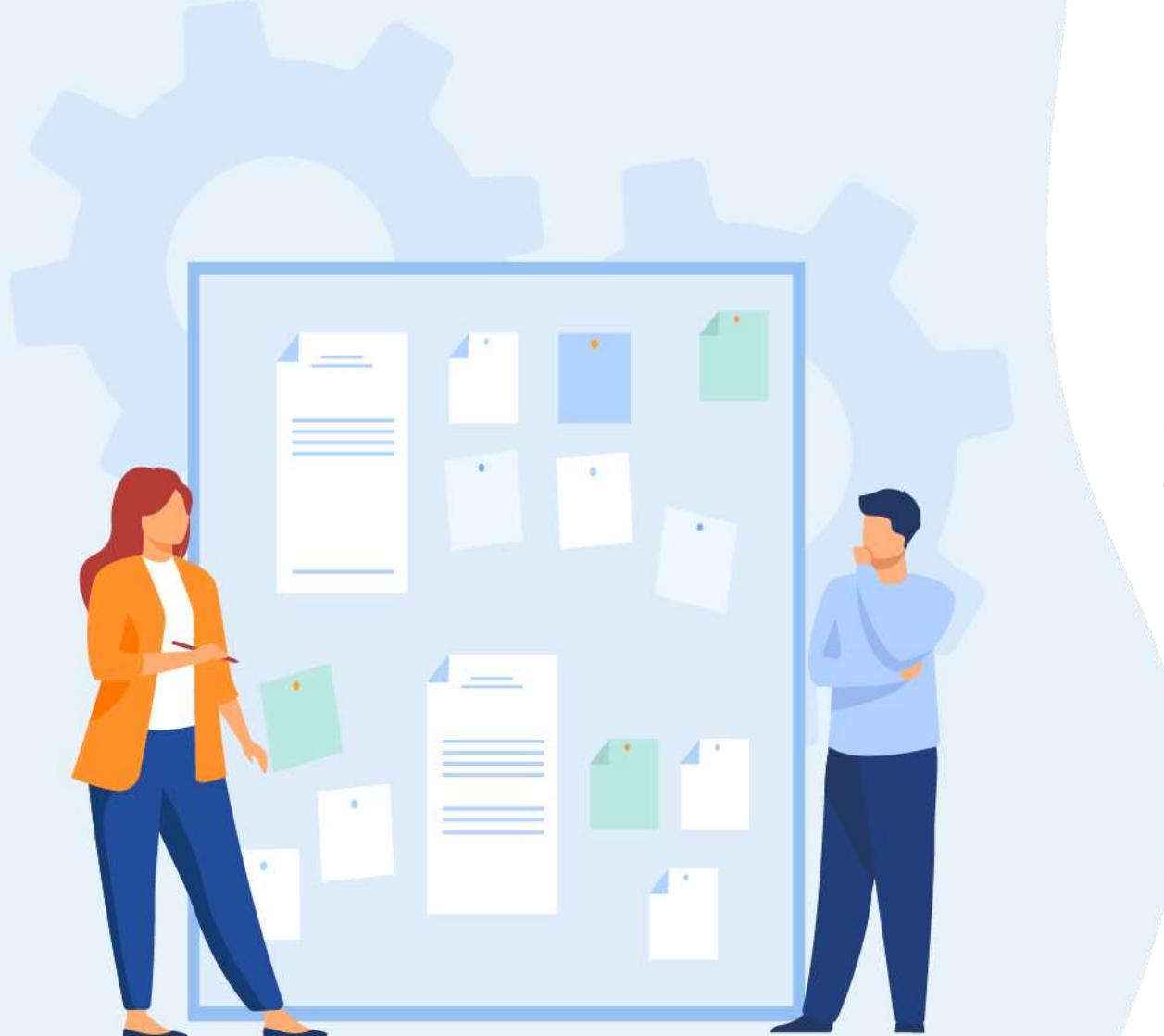
- Parfois, vous devrez peut-être partager des données avec toutes les vues rendues par votre application. Vous pouvez le faire en utilisant la méthode View de la façade share. En règle générale, vous devez placer des appels à la méthode share dans la méthode d'un fournisseur de services boot. Vous êtes libre de les ajouter à la classe App\Providers\AppServiceProvider ou de générer un fournisseur de services distinct pour les héberger :

```
<?php
namespace App\Providers;

use Illuminate\Support\Facades\View;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     * @return void
     */
    public function register()
    {
        // ...
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        View::share('key', 'value');
    }
}
```



CHAPITRE 7

Manipulation des vues

1. Création
2. Transmission des données
- 3. Affichage des compositeurs**
4. Optimisation des vues

Introduction

- Les compositeurs de vue sont des rappels ou des méthodes de classe qui sont appelées lorsqu'une vue est rendue. Si vous avez des données que vous souhaitez lier à une vue chaque fois que cette vue est rendue, un compositeur de vue peut vous aider à organiser cette logique en un seul emplacement. Les compositeurs de vues peuvent s'avérer particulièrement utiles si la même vue est renvoyée par plusieurs routes ou contrôleurs au sein de votre application et a toujours besoin d'un élément de données particulier.
- En règle générale, les compositeurs de vues seront enregistrés auprès de l'un des fournisseurs de services de votre application . Dans cet exemple, nous supposerons que nous avons créé un nouveau App\Providers\ViewServiceProvider pour héberger cette logique.

07 - Manipulation des vues :

Affichage des compositeurs



- Nous utiliserons la méthode composer de la façade View pour enregistrer le composeur de vue. Laravel n'inclut pas de répertoire par défaut pour les compositeurs de vues basés sur les classes, vous êtes donc libre de les organiser comme vous le souhaitez. Par exemple, vous pouvez créer un répertoire app/View/Composers pour héberger tous les compositeurs de vues de votre application :

```
<?php
namespace App\Providers;

use App\View\Composers\ProfileComposer;
use Illuminate\Support\Facades\View;
use Illuminate\Support\ServiceProvider;

class ViewServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        // Using class based composers...
        View::composer('profile', ProfileComposer::class);

        // Using closure based composers...
        View::composer('dashboard', function ($view) {
            //
        });
    }
}
```

07 - Manipulation des vues :

Affichage des compositeurs



- Maintenant que nous avons enregistré le composeur, la méthode compose de la classe App\View\Composers\ProfileComposer sera exécutée à chaque profile de la vue est rendu. Examinons un exemple de la classe composer :

```
<?php
namespace App\View\Composers;

use App\Repositories\UserRepository;
use Illuminate\View\View;

class ProfileComposer
{
    /**
     * The user repository implementation.
     *
     * @var \App\Repositories\UserRepository
     */
    protected $users;

    /**
     * Create a new profile composer.
     *
     * @param \App\Repositories\UserRepository $users
     * @return void
     */
    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }

    /**
     * Bind data to the view.
     *
     * @param \Illuminate\View\View $view
     * @return void
     */
    public function compose(View $view)
    {
        $view->with('count', $this->users->count());
    }
}
```

07 - Manipulation des vues : Affichage des compositeurs

- Comme vous pouvez le voir, tous les compositeurs de vue sont résolus via le conteneur de service , vous pouvez donc indiquer toutes les dépendances dont vous avez besoin dans le constructeur d'un compositeur.

Attacher un compositeur à plusieurs vues

- Vous pouvez attacher un compositeur de vues à plusieurs vues à la fois en passant un tableau de vues comme premier argument à la méthode composer :

```
use App\Views\Composers\MultiComposer;

View::composer(
    ['profile', 'dashboard'],
    MultiComposer::class
);
```

07 - Manipulation des vues : Affichage des compositeurs



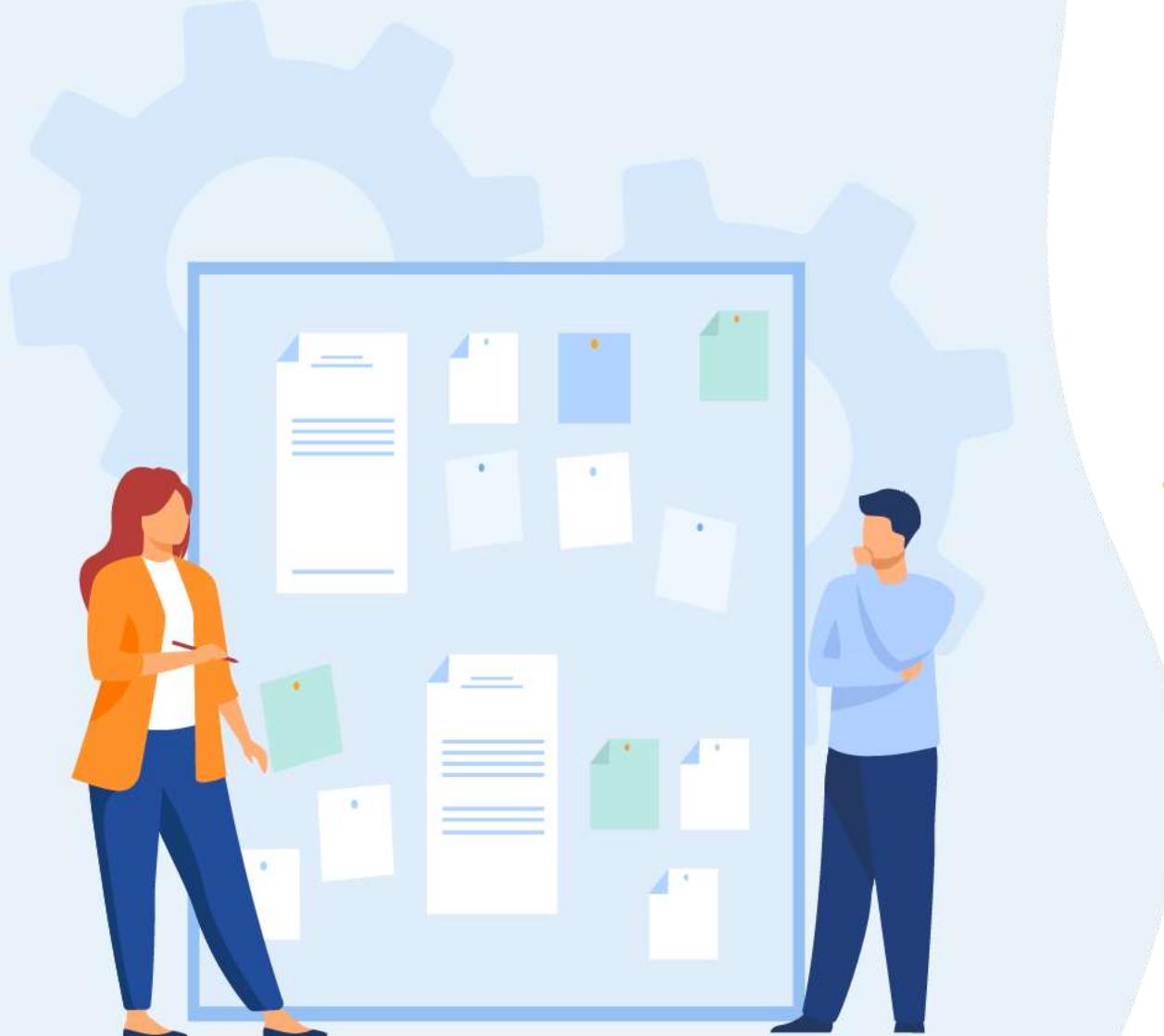
- La méthode composer accepte également le caractère * comme caractère générique, vous permettant d'attacher un compositeur à toutes les vues :

```
View::composer('*', function ($view) {  
    //  
});
```

Afficher les créateurs

- Les « créateurs » de vue sont très similaires aux compositeurs de vue ; cependant, ils sont exécutés immédiatement après linstanciation de la vue au lieu dattendre que la vue soit sur le point de s'afficher. Pour enregistrer un créateur de vue, utilisez la méthode creator :

```
use App\View\Creators\ProfileCreator;  
use Illuminate\Support\Facades\View;  
  
View::creator('profile', ProfileCreator::class);
```



CHAPITRE 7

Manipulation des vues

1. Création
2. Transmission des données
3. Affichage des compositeurs
4. **Optimisation des vues**

07 - Manipulation des vues : Optimisation des vues

Introduction

- Par défaut, les vues du modèle Blade sont compilées à la demande. Lorsqu'une requête est exécutée qui rend une vue, Laravel déterminera si une version compilée de la vue existe. Si le fichier existe, Laravel déterminera alors si la vue non compilée a été modifiée plus récemment que la vue compilée. Si la vue compilée n'existe pas ou si la vue non compilée a été modifiée, Laravel recompilera la vue.

07 - Manipulation des vues : Affichage des compositeurs



- La compilation des vues pendant la requête peut avoir un léger impact négatif sur les performances, c'est pourquoi Laravel fournit la commande `view:cache` Artisan pour précompiler toutes les vues utilisées par votre application. Pour des performances accrues, vous pouvez exécuter cette commande dans le cadre de votre processus de déploiement :

```
php artisan view:cache
```

- Vous pouvez utiliser la commande `view:clear` pour vider le cache de la vue :

```
php artisan view:clear
```