Documentation Laravel

Retour à toute les documentations

Règles

- "`Saisie utilisateur`"
- '`Elément cliquable/sélectionnable`'
- `Nom de fichier, dossier ou autre`
- <Élément à remplacer>

> lien, raccourci clavier et phrase de demande de saisie

```
commande, extrait code et extrait de fichier
```

Table des matières

Installation de Laravel

- Documentation officiel, complète et très bien expliqué

  > https://laravel.com/docs/10.x

- Source

  > https://www.webhi.com/how-to/how-to-install-laravel-on-ubuntu-debian-apache-nginx/

- Installer les différentes dépendances PHP dont Laravel à besoin

```
sudo apt install php php-cli php-common php-mbstring php-xml php-zip php-mysql php-pgsql php-sqlite3 php-json php-bcmath php-gd php-tokenizer php-xmlwriter
```

- Installer Composer
- Installer le serveur web Apache pour php

Installation de Laravel CLI

- Installation de Laravel CLI grâce à `Composer`

```
composer global require laravel/installer
```

Utilisation de Laravel

Création d'un projet Laravel

- Créer un projet Laravel grâce à `Composer`

```
composer create-project --prefer-dist laravel/laravel your-project-name
```

- Créer un projet Laravel grâce à la commande `laravel`

```
laravel new your-project-name
```

- Créer un projet Laravel avec une version spécifique

```
composer create-project --prefer-dist laravel/laravel your-project-name "8.*"
```

Gestion de la base de données

- Créer une migration

```
php artisan make:migration create_<table-name>_table
```

- Créer un modèle

```
php artisan make:model <ModelName>
```

- Créer un modèle et une migration

```
php artisan make:model <ModelName> -m
```

- Modifier le fichier de Model pour ajouter le nom de la table

```
protected $fillable = [
  'name',
  'email',
];
```

- Éxécuter les migrations

```
php artisan migrate
```

- Éxécuter une migration spécifique

```
php artisan migrate --path=/path/to/migration.php
```

- Créer un contrôleur

```
php artisan make:controller <ControllerName>
```

Intégration de plusieurs base de données

- Source

> https://arjunamrutiya.medium.com/laravel-multiple-database-connectivity-a-step-by-step-guide-72cecb5d9223

Migrations

- Source

Les migrations sont des fichiers qui permettent de creer, modifier ou supprimer des tables dans une base de donnees. Les migrations sont executees en premier lieu pour configurer la base de donnees de l'application.

Si vous voulez modifier une table existante, il ne faut pas modifier la migration existante. Il faut créer une nouvelle migration pour ajouter ou modifier des colonnes. le but étant que vous ayez toujours la possibilité de revenir en arrière et que vous puissiez les utiliser sans supprimer les données de la base de données.

- Créer une migration

```
php artisan make:migration <nom_de_la_migration>
```

  - **Convention de nommage** : Lorsque l'on nomme un fichier de migration dans le terminal bash, il est recommandé de le créer de la forme suivante `create_<nom_de_la_table>_table`

- Ouvrez le fichier créé dans le dossier `/database/migrations` et ajoutez le code qui permet de créer la table. L'aspect du fichier de migration est déjà configuré, il contient déjà les méthodes `up` et `down` qui permettent de lancer et d'annuler la migration. Utilisez ce template pour compléter le code en remplaçant les `<>` par les valeurs appropriées :

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```php
class <nom_de_la_migration> extends Migration
{
    $connection = <connection_name>;

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('<nom_de_la_table>', function (Blueprint $table) {
            $table->id();
            $table-><type>('<column_name>')-><attribute>();
            $table->string('exemple')->nullable()->default('default_value');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('<nom_de_la_table>');
    }
}
```

- `nom_de_la_migration` : Nom de la migration (Laravel le complète automatiquement)
- `connection_name` : Nom de la connexion à la base de données (Par défaut `mysql`)
- `nom_de_la_table` : Nom de la table à créer (Laravel le complète automatiquement mais vérifiez que le nom est correct)
- `id()` : Crée une colonne `id` de type `bigIncrements` (Vous pouvez utiliser `id('id_name')` pour changer le nom de la colonne de l'id)
- `type` : Type de la colonne (integer, string, text, etc.)
- `column_name` : Nom de la colonne
- `attribute` : Attribut de la colonne (nullable, default, etc.)
- `string('exemple')->nullable()->default('default_value')` : Crée une colonne `exemple` de type `string` qui peut être `null` et qui a une valeur par défaut `default_value`
- `timestamps()` : Crée deux colonnes `created_at` et `updated_at` de type `timestamp`

- Exécuter les migrations

```
php artisan migrate
```

Seeders

- Source

Un seeder est un fichier qui permet de remplir les tables de la base de donnees avec des donnees. Les seeders sont executees apres les migrations pour fournir des donnees a l'application.

- Créer un seeder

```
php artisan make:seeder <nom_du_seeder>
```

- **Convention de nommage** : Lorsque l'on nomme un fichier seeder dans le terminal bash, il est recommandé de le créer de la forme suivante `<NomDeLaTable>Seeder`. S'il y a un `s` avant `Seeder`, il est conseillé de supprimer

ce `s`.

- Ouvrez le fichier de seeder situé dans le dossier `/database/seeders` et ajoutez le code nécessaire pour remplir la table de la base de données. Le fichier de seeder contient deja la classe `<nom_du_seeder>` qui contient la méthode `run` qui permet de lancer le remplissage de la table. Utilisez ce template pour compléter le code en remplaçant les `<>` par les valeurs appropriées :

```php
<?php

namespace Database\Seeders;

use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class <nom_du_seeder> extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        DB::connection(<connection_name>)->table(<nom_de_la_table>)->insert([
            [
                <nom_du_primaire> => <valeur_du_primaire>,
                <nom_de_la_colonne> => <valeur_de_la_colonne>,
                ...
            ]
        ]);
    }
}
```

  - `nom_du_seeder` : Nom du seeder (Laravel le complète automatiquement)
  - `connection_name` : Nom de la connexion à la base de données (Par défaut `mysql`)
  - `nom_de_la_table` : Nom de la table à remplir
  - `nom_du_primaire` : Nom de la colonne de la clé primaire
  - `valeur_du_primaire` : Valeur de la clé primaire
  - `nom_de_la_colonne` : Nom de la colonne à remplir
  - `valeur_de_la_colonne` : Valeur de la colonne

- Exécuter les seeders

```
php artisan db:seed
```

Lancement d'un projet Laravel en local grâce au serveur web Apache

- Si vous n'avez pas créer le projet Laravel dans le dossier `/var/www/html` il faut créer un lien symbolique entre le dossier du projet Laravel et le dossier `/var/www/html`

```
sudo ln -s /path/to/your-project-name /var/www/html/your-project-name
```

- Créer un fichier de configuration pour le projet Laravel

```
sudo nano /etc/apache2/sites-available/your-project-name.conf
```

- Ajouter le code suivant dans le fichier de configuration

```
```

```
<VirtualHost *:80>
    ServerName your-domain-or-ip
    DocumentRoot /var/www/html/your-project-name/public
    <Directory /var/www/html/your-project-name>
        AllowOverride All
    </Directory>
</VirtualHost>
```

- Activez le module de réécriture Apache :

```
sudo a2enmod rewrite
```

- Activez l'hôte virtuel :

```
sudo a2ensite your-project-name.conf
```

- Redémarrez Apache pour que les modifications prennent effet :

```
sudo systemctl restart apache2
```

Lancement d'un projet Laravel en local grâce au serveur de développement de Laravel

- Lancez le serveur web de développement de Laravel

```
php artisan serve
```

  - Si vous avez cette erreur `The stream or file "/path/to/your-project-name/storage/logs/laravel.log" could not be opened: failed to open stream: Permission denied` il faut modifier le créateur du dossier `storage` et de son contenu
    - Source de la solution

      > https://stackoverflow.com/questions/30639174/laravel-5-ubuntu-14-04-permission-denied-on-storage-log

    ```
    sudo chown -R ${USER}:www-data /path/to/your-project-name/storage
    ```

    - Redonnez les droits au dossier `storage` si nécessaire

      ```
      sudo chmod -R 775 /path/to/your-project-name/storage
      ```

    - Relancez le serveur web de développement de Laravel

      ```
      php artisan serve
      ```

Rendre le serveur de développement de laravel accessible sur tout les appareils d'un réseau local

- Lancer le serveur laravel avec la commande suivante

```
php artisan serve --host=0.0.0.0 --port=8000
```

- Récupérer l'adresse IP ou le nom de la machine sur laquel le serveur laravel est lancé
    - Récupérer l'adresse IP sous Linux

      ```
      hostname -I
      ```

    - Récupérer le nom de la machine sous Linux

      ```
      hostname
      ```

    - Récupérer l'adresse IP sous Windows

      ```
      ipconfig
      ```

    - Récupérer l'adresse IP sous Windows grâce à l'inteface graphique
        - https://support.microsoft.com/en-us/windows/find-your-ip-address-in-windows-f21a9bbc-c582-55cd-35e0-73431160a1b9
- Ouvrir un navigateur sur un autre appareil connecté au même réseau
- Entrer l'adresse IP de la machine sur lequel le serveur laravel est lancé suivi du numéro de port, en l'occurence "`:8000`"

  ```
  <adresse_ip>:8000
  ```

  OU

  ```
  <nom>:8000
  ```

    - En règle général l'adresse IP est de la forme `192.168.1.XX`

      ```
      192.168.1.XX:8000
      ```

Configuration de votre projet Laravel

- Configuration des permissions dans votre projet Laravel (normalement vous n'avez pas besoin de faire cette étape)

  ```
  sudo chown -R www-data:www-data /path/to/your-project-name
  sudo chmod -R 755 /path/to/your-project-name
  ```

Configuration du .env

Voici une explication détaillée des différentes variables du fichier `.env` et leur utilité dans une application Laravel :

Configuration de l'application

- **`APP_NAME`** : Nom de l'application. Utilisé dans les notifications, les emails, ou dans certaines parties du frontend pour afficher le nom de l'application.
- **`APP_ENV`** : Environnement de l'application (`local`, `production`, `staging`, etc.). Permet d'ajuster le comportement de l'application en fonction de l'environnement.
- **`APP_KEY`** : Clé utilisée pour chiffrer des données sensibles. Générée par Laravel avec la commande `php artisan key:generate`.

- `APP_DEBUG` : Active ou désactive le mode débogage (`true` ou `false`). En mode débogage, les erreurs s'affichent directement dans le navigateur.
- `APP_TIMEZONE` : Définit le fuseau horaire de l'application, comme `UTC`, `Europe/Paris`, etc.
- `APP_URL` : URL de base de l'application. Utilisée pour générer des liens absolus dans l'application.

Langue et localisation

- `APP_LOCALE` : Langue par défaut de l'application, comme `en` pour l'anglais ou `fr` pour le français.
- `APP_FALLBACK_LOCALE` : Langue utilisée si la traduction dans `APP_LOCALE` n'est pas disponible.
- `APP_FAKER_LOCALE` : Langue utilisée par la librairie `Faker` pour générer des données fictives (noms, adresses, etc.).

Maintenance

La maintenance dans Laravel fait référence à un mode spécial où l'application devient temporairement inaccessible aux utilisateurs finaux. Cela permet aux développeurs d'effectuer des tâches comme des mises à jour, des modifications ou des réparations sans interrompre l'expérience utilisateur avec des erreurs ou des bugs.

- `APP_MAINTENANCE_DRIVER` : Détermine comment l'état de maintenance est géré (fichier, base de données, etc.).
- `APP_MAINTENANCE_STORE` : Emplacement pour stocker les données de maintenance (si nécessaire).
- **Fonctionnement :**
    - En mode maintenance, Laravel affiche une page par défaut ou personnalisée indiquant que le site est temporairement hors service.
    - Les commandes comme `php artisan down` activent le mode maintenance, tandis que `php artisan up` le désactive.
    - Tu peux spécifier des utilisateurs ou des IP autorisés à contourner ce mode avec `--allow`.

Ce n'est pas obligatoire. Cependant, c'est une bonne pratique dans les environnements de production pour éviter que les utilisateurs ne rencontrent des bugs ou des incohérences pendant des opérations critiques.

Sécurité

- `BCRYPT_ROUNDS` : Nombre de tours pour hacher les mots de passe avec bcrypt. Plus la valeur est élevée, plus le processus est sécurisé mais gourmand en ressources.
- **Valeur de référence :**
    - Laravel utilise une valeur par défaut de 10. Cette valeur offre un bon compromis entre sécurité et performances.
    - Dans des environnements nécessitant une sécurité accrue (par exemple, avec des serveurs puissants), tu peux augmenter ce nombre à 12 ou 14.
    - Éviter des valeurs trop élevées (comme 20+), car elles ralentiront considérablement le processus de connexion.

Logs

- `LOG_CHANNEL` : Définit le canal de journalisation principal (fichier, stack, daily, etc.).
- `LOG_STACK` : Permet d'activer une pile de canaux de journalisation.
- `LOG_DEPRECATIONS_CHANNEL` : Canal pour capturer les messages d'obsolescence de code.

Base de données

- `DB_CONNECTION` : Type de base de données utilisé (`mysql`, `sqlite`, `pgsql`, etc.).
- `DB_HOST` : Adresse du serveur de base de données.
- `DB_PORT` : Port utilisé pour la connexion à la base de données.
- `DB_DATABASE` : Nom de la base de données.
- `DB_USERNAME` : Nom d'utilisateur pour se connecter à la base.
- `DB_PASSWORD` : Mot de passe pour la connexion.

Sessions

- `SESSION_DRIVER` : Méthode de gestion des sessions (`file`, `cookie`, `database`, `redis`, etc.).
- `SESSION_LIFETIME` : Durée de vie des sessions en minutes.
- `SESSION_ENCRYPT` : Détermine si les sessions doivent être chiffrées (`true` ou `false`).
- `SESSION_PATH` : Chemin où les cookies de session sont accessibles.
    - Définit le chemin sur lequel les cookies de session sont valides.
    - **Exemple :**
        - Si la valeur est /app, les cookies ne seront accessibles que pour les URLs commençant par /app.
    - Par défaut, Laravel utilise /, ce qui signifie que les cookies sont valides pour toutes les routes de ton domaine.
- `SESSION_DOMAIN` : Domaine associé aux cookies de session.
    - Définit le domaine pour lequel les cookies de session sont valides.
    - **Exemple :**
        - Si ton site est `example.com` et que tu as un sous-domaine `admin.example.com` :
            - Valeur par défaut (vide) : Les sessions sont spécifiques à chaque sous-domaine.
            - Valeur `.example.com` : Partage les sessions entre `example.com` et `admin.example.com`.
    - **Utilité :**
        - Ces paramètres sont utiles pour gérer les sessions dans des applications multi-domaines ou ayant des chemins spécifiques nécessitant des restrictions.

Diffusion (Broadcast)

Le broadcasting est un mécanisme utilisé pour envoyer des mises à jour en temps réel aux utilisateurs sans qu'ils aient besoin de recharger la page.

Laravel propose le broadcasting pour diffuser des événements via des canaux comme `WebSocket`, `Pusher`, ou d'autres services compatibles.

- Exemple d'utilisation :
    - **Une application de chat** : Quand un utilisateur envoie un message, tous les autres utilisateurs dans la même discussion voient ce message en temps réel.
    - **Notifications en direct** : Quand une commande est validée dans un e-commerce, l'administrateur est notifié en direct.
- Configurations courantes :
    - `pusher` : Utilise le service Pusher pour diffuser les événements.
    - `log` : Enregistre les événements dans les logs pour déboguer (utile pour tester sans configurer un service).
    - `null` : Désactive complètement le broadcasting.

Utilisé cette configuration uniquement si l'application a des fonctionnalités en temps réel.

- `BROADCAST_CONNECTION` : Connexion utilisée pour le la diffusion (broadcasting) en temps réel.

Filesystems

- `FILESYSTEM_DISK` : Disque utilisé par défaut pour le stockage de fichiers (`local`, `s3`, etc.).

File d'attente

Les queues (files d'attente) permettent d'exécuter des tâches lourdes ou non urgentes en arrière-plan, au lieu de les exécuter immédiatement.

- **Exemples de tâches :**
    - Envoyer un email après une inscription.
    - Générer un rapport PDF.
    - Redimensionner une image.
- Comment ça fonctionne ?
    - Lorsqu'une tâche est ajoutée à la file d'attente, elle est mise en attente jusqu'à ce qu'un worker (processus en arrière-plan) la traite. Cela évite de bloquer le serveur ou de ralentir l'application.
- Configurations courantes :
    - `sync` : Les tâches sont exécutées immédiatement (pas vraiment en file d'attente).
    - `database` : Les tâches sont stockées dans la base de données et exécutées par un worker.

    - `redis` : Utilise Redis pour une file d'attente rapide et performante.

Il est possible d'utiliser sync par défaut. Mais si l'application a beaucoup de tâches, passer à une file d'attente comme database ou redis est recommandé.

- `QUEUE_CONNECTION` : Type de gestion des files d'attente (`sync`, `database`, `redis`, etc.).

Cache

- `CACHE_STORE` : Méthode de stockage du cache (`file`, `redis`, `database`, etc.).
- `CACHE_PREFIX` : Préfixe utilisé pour identifier les clés de cache.

Memcached

Memcached est un système de gestion de cache en mémoire rapide, utilisé pour stocker des données temporaires (comme des résultats de requêtes) afin d'améliorer les performances en réduisant les appels à la base de données. Ce système est optionnel.

- `MEMCACHED_HOST` : Adresse IP du serveur Memcached (si utilisé).

Redis

- Redis est une base de données en mémoire rapide et légère, souvent utilisée comme :
    - **Cache** : Pour stocker des données temporairement.
    - **File d'attente** : Pour gérer les files d'attente des tâches (plus rapide que la base de données traditionnelle).
    - **Session store** : Pour gérer les sessions utilisateur.
- Laravel peut fonctionner sans Redis, mais l'utiliser peut améliorer les performances, surtout dans les applications à grande échelle.
- **Alternatives** : Tu peux utiliser des systèmes de cache comme `file`, `database`, ou `Memcached`.
- `REDIS_CLIENT` : Client Redis utilisé (`predis` ou `phpredis`).
- `REDIS_HOST` : Adresse du serveur Redis.
- `REDIS_PASSWORD` : Mot de passe pour le serveur Redis (si requis).
- `REDIS_PORT` : Port utilisé pour la connexion à Redis.

Email

- `MAIL_MAILER` : Service utilisé pour envoyer des emails (`smtp`, `mailgun`, `sendmail`, etc.).
- `MAIL_DRIVER` : Ancien nom pour `MAIL_MAILER` (encore utilisé dans certaines versions).
- `MAIL_HOST` : Adresse du serveur SMTP. (Exemple : `smtp.gmail.com` pour Gmail).
- `MAIL_PORT` : Port utilisé pour le serveur SMTP. (Exemple : `465` pour Gmail).
- `MAIL_USERNAME` : Identifiant pour le serveur SMTP.
- `MAIL_NAME` : Nom de l'expéditeur (affiché dans les emails envoyés).
- `MAIL_PASSWORD` : Mot de passe pour le serveur SMTP. Disponible sur Gmail via la propriété `Mot de passe d'application` disponible uniquement si l'authentification à deux facteurs est activée.
- `MAIL_ENCRYPTION` : Méthode de cryptage pour les emails (`tls`, `ssl`).
- `MAIL_FROM_ADDRESS` : Adresse email de l'expéditeur par défaut.
- `MAIL_FROM_NAME` : Nom affiché pour l'expéditeur.

ViteJS

- `VITE_APP_NAME` : Nom de l'application pour le système de bundling Vite.js.

Intégration de Tailwind CSS

- Installation de Tailwind CSS

```
npm install -D tailwindcss postcss autoprefixer
```

- Création du fichier de configuration de Tailwind CSS

```
npx tailwindcss init -p
```

- Ajoutez le code suivant dans le fichier `tailwind.config.js`

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./resources/**/*.blade.php",
    "./resources/**/*.js",
    "./resources/**/*.vue",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

- Ajoutez le code suivant dans le fichier `ressources/css/app.css`

```
@tailwind base;
@tailwind components;
@tailwind utilities;
@tailwind forms;
```

- Vous pouvez compiler les fichiers CSS de Tailwind CSS en utilisant la commande suivante
    - Vous pouvez voir ça comme l'activation du style Tailwind CSS. Il faudra refaire cette commande à chaque fois que vous ajouter un nouveau style dans votre projet

```
npm run build
```

Intégration de Livewire

Ajout de Livewire à un projet Laravel

- Installer LiveWire grâce à `Composer`

```
composer require livewire/livewire
```

Création d'un composant Livewire

- Éxécuter la commande suivante dans le dossier racine de votre projet Laravel pour créer un composant Livewire

```
php artisan make:livewire <NomDuComposant>
```

Licence

Authors: Floris Robart

Retour à toute les documentations