

# Cours

Outils & administration du WEB

# Objectifs (1)

Ce cours a pour objectif de doter les étudiants d'outils puissants en adéquation avec leur formation et prisés par les entreprises.

Les étudiants, tout au long de ce cours, apprendront différentes notions de la programmation et des bonnes pratiques en entreprise.

Ils apprendront des notions clés :

- des environnements WEB,
- Des bases de données et les attentes en entreprise
- Du développement frontend avec un Framework CSS
- Du développement backend avec un Framework PHP

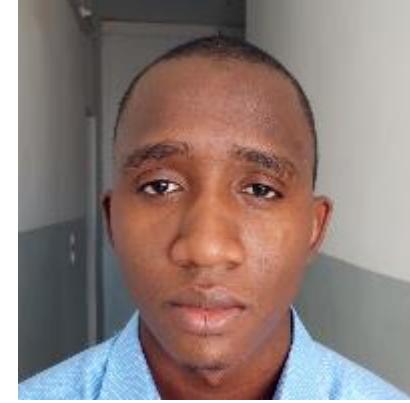
# Objectifs (2)

A l'issue de cette formation, les étudiants doivent être en mesure de :

- Créer un environnement complet PHP
- Créer un projet Laravel
- Gérer des bases de données
- Développer des applications complexes en utilisant les bonnes pratiques en entreprise
- Se familiariser avec le framework Bootstrap CSS pour une meilleure expérience utilisateur
- Rencontrer, déboguer et solutionner des problèmes réels
- Déployer des applications réelles en production

# Présentation

- M FOFANA
- Ingénieur en génie logiciel
- Doctorant en Sciences de données et Application
- Chef de division développement et innovation technologique à la DSI de l'UIDT
- Senior Project Manager



# Historique du WEB

Le chercheur britannique Tim Berners-Lee a inventé le World Wide Web en 1989, lorsqu'il travaillait au CERN.

L'idée de base du WWW était de combiner les technologies des ordinateurs personnels, des réseaux informatiques et de l'hypermédia pour créer un système d'information mondial, puissant et facile à utiliser.

# Historique & Evolution du WEB

# Evolution du WEB

- Le premier site Web créé au CERN, et dans le monde, était destiné au projet World Wide Web lui-même. Il était hébergé sur l'ordinateur NeXT de Tim Berners-Lee. En 2013, le CERN a entrepris de remettre en service ce premier site Web : info.cern.ch.
- Jusqu'au passage dans le domaine public de ces nouvelles technologies, la toile était avant tout réservée aux chercheurs des principales académies et laboratoires. Le Web a alors commencé son développement auprès du grand public. L'absence d'entité centrale du Web a permis à toute personne disposant du matériel nécessaire et d'une connexion à Internet de publier n'importe quoi, sans craindre la censure. On découvre alors des mots comme modem, navigateur, fournisseur d'accès à Internet, puis plus tard ADSL et fibre.

# Evolution du WEB

Le 30 avril 1993, le CERN a mis le logiciel du World Wide Web dans le domaine public. Puis il a émis une version suivante de l'application sous licence libre, une façon plus sûre de maximiser sa diffusion. Ce faisant, il a permis à la Toile de se tisser.

Au début, le Web était donc simplement un ensemble de sites statiques, chaque propriétaire de site publiant ses propres informations. Ensuite, le Web a commencé à devenir participatif. Les messageries instantanées ont créé les premiers vrais réseaux.

Mais le premier réseau social est apparu en 1997 et s'appelait Sixdegrees.com. Il permettait aux utilisateurs de créer des profils, des groupes et d'inviter des amis. Il n'a tenu que 4 ans, mais il avait tout de même réuni 3,5 millions d'usagers.

# Evolution du WEB

De nombreux utilisateurs communiquaient aussi grâce aux blogs pour partager leur opinion, raconter un voyage ou une expérience culinaire, mais les réseaux sociaux ont rapidement offert une alternative plus pratique.

Plus besoin d'inviter les gens à venir lire son blog : en postant sur les réseaux sociaux, la publication atteignait directement tous ses contacts.

C'est ainsi que sont apparus Friendster, puis son clone MySpace. Des réseaux sociaux plus connus sont arrivés peu après, comme LinkedIn en 2003, ou Facebook en 2004.

# Evolution du WEB

À chaque fois, s'intègrent de nouvelles fonctions, de plus en plus de membres, mais aussi de nouveaux usages comme le partage de vidéo personnelles sur YouTube, de sons sur SoundCloud, ou carrément de fichiers illégaux sur des plate-formes de téléchargement.

Aujourd'hui, les réseaux sociaux sont au cœur du Web, à la fois parce qu'ils permettent à l'internaute lambda de suivre, voire de converser, avec les plus grandes personnalités du showbizz, du sport ou de la politique, mais aussi parce que c'est devenu un véritable outil de communication, voire de propagande.

Facebook est même devenu le premier « pays » du monde avec plus de deux milliards de membres !

# Evolution du WEB

## Arrivée du Web Mobile

Le dernier grand phénomène bouleversant le Web a été l'apparition des smartphones.

L'accès à la toile via un mobile a commencé dès 1996, avec notamment le Nokia 9000. Cependant, l'équipement étant primitif et à un tarif prohibitif pour le grand public, il faudra attendre 2005 environ et l'apparition de smartphones 3G grand public pour commencer à le voir se développer sérieusement.

Les sites Web ont dû s'adapter, en proposant des versions de sites simplifiés fonctionnant sur des appareils peu puissants, avec un navigateur limité, tout en réduisant la quantité de données envoyées.

# Evolution du WEB

## Arrivée du Web Mobile

La mise sur le marché du premier iPhone en 2007 a eu pour effet de faire exploser le nombre d'utilisateurs mobile, et d'offrir un navigateur Web mobile complet. Pour s'adapter à ces écrans, les créateurs de sites détectaient simplement s'il s'agissait d'un mobile, et offraient une version simplifiée. Mais avec un Web toujours présent dans la poche, le smartphone est devenu un enjeu majeur.

Désormais le « mobile-first » s'impose. Les sites doivent être conçus d'abord pour le mobile. Même Google a lancé en 2017 son indexation mobile-first, qui référence le contenu mobile avant tout. Cette dernière phase est encore en cours. Et elle risque d'apporter encore bien des bouleversements puisque l'ordinateur, au cœur de la naissance d'Internet et de son fonctionnement, est désormais dépassé par les smartphones capables de prendre des photos et des vidéos, mais aussi d'enregistrer du son pour les partager sur la toile ou même de regarder la télévision et des séries TV.

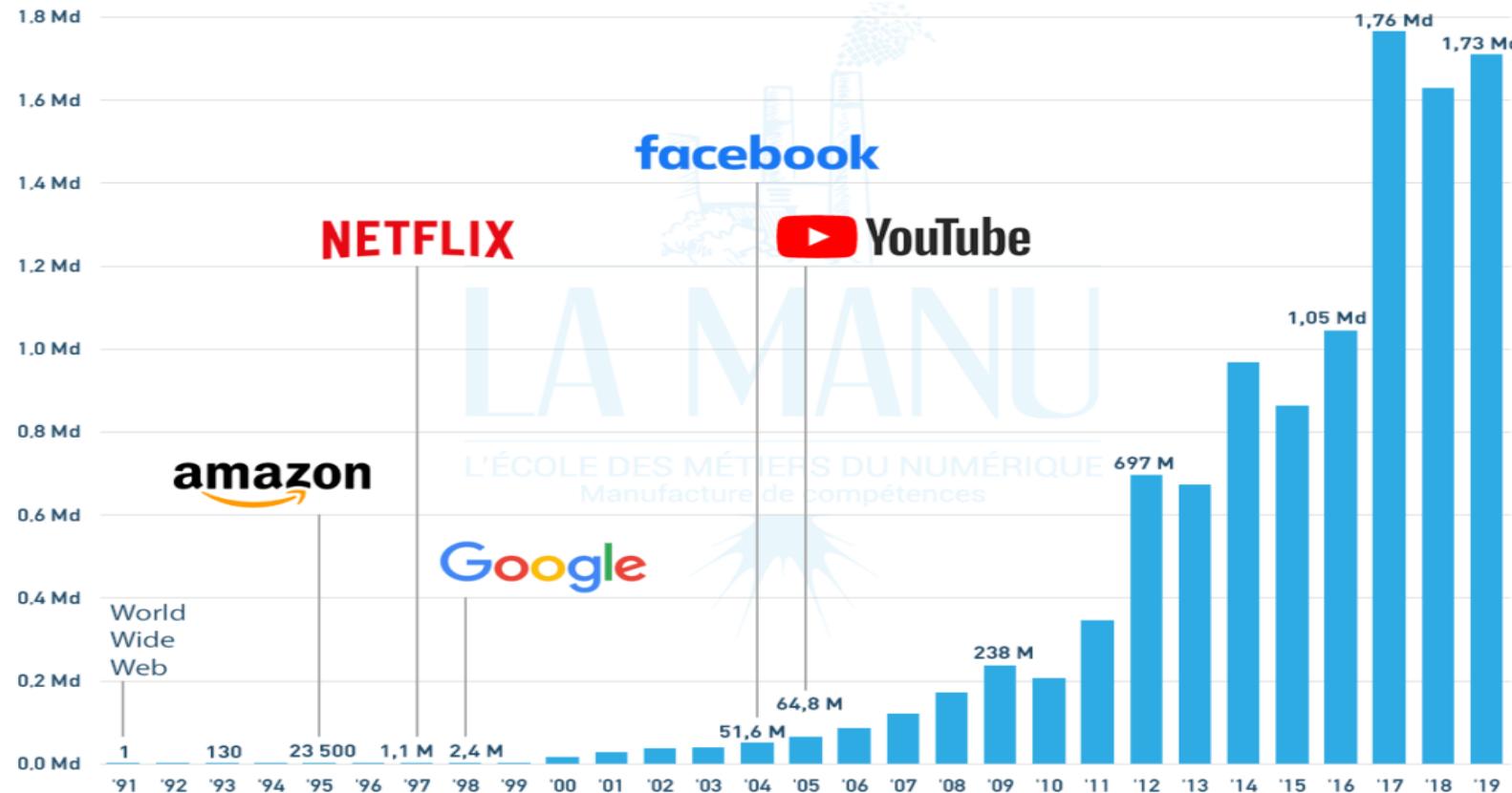
# Evolution du WEB

## Ce qu'il faut retenir

- L'idée d'un réseau mondial d'échange d'informations est née en 1989.
- En 2019, les pages Web se comptent par milliards et l'Internet est devenu l'outil numéro 1 de communication.
- La révolution mobile a séparé l'Internet de l'ordinateur pour un usage en temps réel, n'importe où et n'importe quand.

# Evolution du WEB

## Évolution du nombre de sites Internet



Source : NetCraft / InternetLiveStats

# Le développement WEB

# Définition

- Le développement Web désigne de manière générale les tâches associées au développement de sites Web destinés à être hébergés via un intranet ou Internet. Le processus de développement web comprend, entre autres, la conception de sites web, le développement de contenu web, l'élaboration de scripts côté client ou côté serveur et la configuration de la sécurité du réseau.
- Le développement Web est le codage ou la programmation qui permet de faire fonctionner un site Web, selon les exigences du propriétaire. Il traite principalement de l'aspect non conceptuel de la création de sites Web, qui comprend le codage et l'écriture du balisage.
- Le développement Web va de la création de pages en texte brut à des applications Web complexes, des applications de réseaux sociaux et des applications commerciales électroniques.

# La fonction de développeur WEB

- Un développeur ou programmeur Web est une personne qui prend un projet Web, qui a été pensée et réfléchie par un client ou une équipe de conception, et la transforme en site Web. Ils le font en écrivant des lignes de code compliqué. Pour les écrire, ils utilisent différents langages avec une spécificité et une utilité pour chacun d'entre eux.
- Les développeurs Web ont un travail assez difficile, car ils doivent prendre une langue que nous comprenons, comme l'anglais, et la traduire dans une langue qu'un ordinateur comprend, comme Python ou HTML.
- Comme vous pouvez l'imaginer, cela va prendre beaucoup de temps et d'efforts pour apprendre différents langages de programmation ainsi que leurs utilisations. Différents types de développeurs se spécialisent dans différents domaines, ce qui signifie que les grands projets Web sont généralement une collaboration entre plusieurs développeurs différents.

# Les types de développement WEB

Les trois principaux types de développeurs sont front-end, back-end et full-stack.

- Les développeurs front-end s'occupent de la partie d'un site Web que les gens voient et avec laquelle ils interagissent.
- Les développeurs back-end sont responsables eux, du code en arrière-plan. Ce code va servir à contrôler la façon dont un site Web va se charger et s'exécuter.
- Les développeurs full-stack font un peu de tout .
- Les termes « front-end » et « back-end » sont utilisés dans le métier de programmeurs pour décrire les couches d'un site web.

# « DÉVELOPPEMENT FRONT-END » ?

- Un développeur front-end va prendre en charge la conception du site Web et écrire le code nécessaire pour l'implémenter sur le Web. Il se doit de maîtriser au moins trois **langages de programmation** que sont, **HTML, CSS et JavaScript**. Il a aussi des responsabilités dans le référencement naturel (**SEO**), la conception graphique et l'édition des images et photos du site. Mais aussi les différents tests (utilisabilité et accessibilité) ainsi que la performance du site web et sa compatibilité avec les différents navigateurs et format d'affichage (mobile, desktop) autrement dit « responsive design ».

# Langages utilisé en frontend ?

**HTML** permet d'intégrer du contenu à un site Web tout en le divisant en titres, paragraphes et tableaux. C'est un langage essentiel. Il indique au site Web quel contenu afficher et, dans une certaine mesure, comment l'afficher.

Des commandes telles que «lien», «image» ou «vidéo», indiquent à votre navigateur quoi afficher, d'où obtenir le contenu et comment l'afficher.

# Langages utilisé en frontend ?

**CSS** est utilisé en conjonction avec HTML. Alors que HTML indique à la page Web quel contenu afficher, CSS indique à la page Web comment afficher le contenu – c'est un langage de «**style**». Il va notamment permettre le changement de couleurs, de polices, le positionnement des éléments ainsi que le changement de taille du texte.

# Langages utilisé en frontend ?

**JavaScript** va quand à lui, rendre votre site Web réactif, interactif et attrayant pour vos visiteurs. Un développeur JavaScript va travailler sur des extraits de code JavaScript qui, souvent seront intégrés au code source HTML d'un site Web. Bien qu'il ait toujours été un langage front-end, JavaScript devient également de plus en plus populaire pour le développement principal.

# « DÉVELOPPEMENT BACK-END » ?

Alors que les développeurs front-end sont responsables de la programmation côté client, les développeurs back-end doivent s'occuper du côté **serveur**.

Cela signifie qu'ils doivent créer le code et les programmes qui alimentent le serveur du site web, les bases de données et toutes les applications que contient le site internet. Ils utilisent un large éventail de langages différents côté serveur afin de construire des programmes compliqués.

Parmi les langages les plus utilisés, citons PHP, Python, Java et Ruby, tandis que le SQL est couramment utilisé pour gérer et analyser les données dans les bases de données des sites web.

La vitesse d'un site web étant une considération majeure lorsqu'il s'agit d'optimisation pour les moteurs de recherche (SEO), elle est un facteur important lors du développement du back-end.

# Les langages du back-end

**PHP** a dominé les définitions de ce qu'est le développement web. Connu comme le langage de l'internet, il est actuellement utilisé sous une forme ou une autre sur plus de 80 % des sites web existants. Le PHP est très populaire auprès des développeurs. Notamment sur les sites web construits sur des plateformes comme WordPress. PHP joue un rôle particulièrement important en matière de développement WordPress .

Ce language permet de réaliser des choses comme des boîtes de dialogue s'ouvrant sur un écran en réponse aux actions d'un utilisateur. Mais aussi des chatbots répondant au comportement définit de l'utilisateur avec des messages correspondants. Ou encore une animation qui se produit lorsqu'un utilisateur défile au-delà d'un certain point sur une page. C'est-à-dire, toutes les fonctions de site Web dynamiques qui doivent se produire à l'écran sans que l'utilisateur ait à recharger manuellement un site.

# Les langages du back-end

- **Python** est un langage de programmation orienté objet de haut niveau utilisé principalement pour le développement de sites Web et d'applications. C'est un langage de programmation à usage général, ce qui est une autre façon de dire qu'il peut être utilisé pour presque tout. Plus important encore, il s'agit d'un langage interprété, ce qui signifie que le code écrit n'est pas réellement traduit dans un format lisible par ordinateur au moment de l'exécution.
- Le cas d'utilisation le plus basique de Python est un langage de script et d'automatisation. Il est également utilisé pour automatiser les interactions avec les navigateurs Web ou les interfaces graphiques des applications.

# Les langages du back-end

**Java** est un vieux langage qui est populaire pour toute une série d'utilisation de programmation différentes.

Du point de vue du développement web, Java est utilisé pour créer des applications web réactives et évolutives qui sont utilisées pour la conception rapide de sites web.

# DIFFÉRENCE ENTRE WEB-DESIGN ET WEB-DÉVELOPPEMENT ?

Le web design et le développement web sont deux choses différentes, mais bien sûr, cela dépend de la façon dont vous définissez « design » et « développement ». Il faut des connaissances différentes pour comprendre ce qu'est un concepteur de site web et ce qu'est un développeur de site web. Ces professions sont différentes et elles ont les rôles suivants :

- Le web designer est la personne qui est responsable de la création du concept du site web. Il peut décider que le site doit avoir une certaine couleur, un certain contenu et des pages spécifiques. C'est un expert en infographie, et dispose de grandes compétences techniques pour réaliser des logos et vidéos. Il indique au développeur où ces éléments doivent être placés sur la page web. Cependant, ils ne participent pas à la construction du site web ou du code sous-jacent.
- Le développeur web prend les concepts du designer et crée le code. Il utilise les différentes technologies et ressources vues précédemment pour transformer et concevoir une idée en un site web, et les mettre à la disposition de personnes comme vous.

# Quelques notions clés - HTTP

HTTP définit un ensemble de méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée.

- **GET**

La méthode GET demande une représentation de la ressource spécifiée. Les requêtes GET doivent uniquement être utilisées afin de récupérer des données.

- **HEAD**

La méthode HEAD demande une réponse identique à une requête GET pour laquelle on aura omis le corps de la réponse (on a uniquement l'en-tête).

- **POST**

La méthode POST est utilisée pour envoyer une entité vers la ressource indiquée. Cela entraîne généralement un changement d'état ou des effets de bord sur le serveur.

- **PUT**

La méthode PUT remplace toutes les représentations actuelles de la ressource visée par le contenu de la requête.

- **DELETE**

La méthode DELETE supprime la ressource indiquée.

- **Ect.**

# Frameworks WEB

- Un Framework est un ensemble d'outils et de composants logiciels organisés conformément à un plan d'architecture et des patterns, l'ensemble formant ou promouvant un « squelette » de programme, un canevas. Il est souvent fourni sous la forme d'une bibliothèque logicielle et accompagné du plan de l'architecture cible du Framework.
- Un Framework est conçu en vue d'aider les programmeurs dans leur travail. L'organisation du Framework vise la productivité maximale du programmeur qui va l'utiliser — gage de baisse des coûts de construction et maintenance du programme. Le contenu exact du Framework est dicté par le type de programme et l'architecture cible pour lequel il est conçu.
- Liste de Framework : <https://fr.wikipedia.org/wiki/Framework#Exemples>
  - Source: WIKIPEDIA

# Frameworks PHP

- Selon le W3Techs, le PHP est utilisé par environ 79 % de tous les sites web. Il est 8 fois plus populaire que ASP.NET, son concurrent le plus proche dans les langages de programmation côté serveur.
- Un framework PHP est une plate-forme permettant de créer des applications web en PHP. Les frameworks PHP fournissent des bibliothèques de code pour les fonctions les plus courantes, ce qui réduit la quantité de code original à écrire.

# Pourquoi utiliser un Framework PHP ?

## 1. Un développement plus rapide

Comme les frameworks PHP disposent de bibliothèques et d'outils intégrés, le temps nécessaire au développement est moindre.

Par exemple, le framework CakePHP dispose de l'outil en ligne de commande Bake qui peut rapidement créer tout code de squelette dont vous avez besoin dans votre application.

Plusieurs frameworks PHP populaires intègrent la bibliothèque PHPUnit pour faciliter les tests.

# Pourquoi utiliser un Framework PHP ?

## 2. Moins de code à écrire

L'utilisation de fonctions intégrées au framework permet de ne pas avoir à écrire autant de code original.

# Pourquoi utiliser un Framework PHP ?

## 3. Bibliothèques pour les tâches communes

De nombreuses tâches que les développeurs devront effectuer dans les applications web sont courantes. Il s'agit par exemple de la validation des formulaires, du nettoyage des données et des opérations CRUD (Create, Read, Update, and Delete). Plutôt que d'avoir à écrire vos propres fonctions pour ces tâches, vous pouvez simplement utiliser celles qui font partie du framework.

# Pourquoi utiliser un Framework PHP ?

## 4. Suivre les bonnes pratiques de codage

Les Frameworks PHP suivent généralement les meilleures pratiques de codage. Par exemple, ils divisent le code en plusieurs répertoires selon la fonction.

Ils vous obligent à organiser le code d'une manière plus propre, plus nette et plus facile à maintenir.

Les Frameworks ont également leurs propres conventions de dénomination des entités que vous devez suivre.

# Pourquoi utiliser un Framework PHP ?

## 5. Plus sûr que d'écrire vos propres applications

Il existe de nombreuses menaces à la sécurité de PHP, notamment les scripts cross-sites, les attaques par injection SQL et les falsifications de requêtes cross-sites. Si vous ne prenez pas les bonnes mesures pour sécuriser votre code, vos applications web PHP seront vulnérables.

L'utilisation d'un framework PHP ne remplace pas l'écriture d'un code sécurisé, mais elle minimise les risques d'exploitation par des pirates. Les bons frameworks intègrent un nettoyage des données et des défenses contre les menaces courantes mentionnées ci-dessus.

# Pourquoi utiliser un Framework PHP ?

## 6. Un meilleur travail d'équipe

Les projets avec plusieurs développeurs peuvent se tromper s'il n'y a pas de clarté au sujet de :

- La documentation
- Les décisions du design
- Les normes du code

L'utilisation d'un framework définit des règles de base claires pour votre projet. Même si un autre développeur n'est pas familier avec le framework, il devrait être capable d'apprendre rapidement les ficelles du métier et de travailler en collaboration.

# Pourquoi utiliser un Framework PHP ?

## 7. Plus facile à entretenir

Les Frameworks PHP encouragent le remaniement du code et favorisent le développement DRY (Don't Repeat Yourself). La base de code ainsi allégée nécessite moins de maintenance.

Vous n'avez pas non plus à vous soucier de la maintenance du cœur du Framework, puisque les développeurs le font pour vous.

# Préalables

La première chose que vous devez connaître avant d'utiliser un Framework PHP est PHP lui-même ! Si vous n'avez pas une bonne maîtrise du langage, vous aurez du mal à vous choisir un Framework. La plupart des Frameworks fonctionnent avec PHP version 7.2 ou supérieure.

Il est également indispensable de connaître le PHP orienté objet, car la plupart des Frameworks PHP modernes sont orientés objet. Assurez-vous de comprendre les concepts tels que les classes, les objets, l'héritage, les méthodes, les traits et les modificateurs d'accès.

Comme de nombreuses applications web se connectent à une base de données, vous devez connaître les bases de données et la syntaxe SQL. Chaque Framework PHP a sa propre liste de bases de données supportées.

# Le Framework Laravel

# Pourquoi Laravel ?

Il existe une variété d'outils et de cadres à votre disposition lors de la création d'une application Web. Cependant, nous pensons que Laravel est le meilleur choix pour créer des applications Web modernes et complètes.

- Un Framework progressiste
- Un Framework évolutif
- Une communauté engagée

# Outils & Prérequis

- Installer :
  - composer : <https://getcomposer.org/download/>
  - Vscode : <https://code.visualstudio.com/download>
  - Installer un serveur PHP selon le système d'exploitation (wamp ou xamp our windows) : pour mon cas « wamp server 3 » et ajouter la commande php aux variables d'environnement système
  - Installer Git
  - Installer Heroku Client

# Comment installer ?

- Soit passer par composer
  - *composer create-project –prefer-dist laravel/laravel TuttoLara*
  - *cd TuttoLara*
  - *php artisan serve*
- Soit installer l'installateur Laravel

- *composer global require laravel/installer*
  - *laravel new TuttoLara*
  - *cd TuttoLara*
  - *php artisan serve*

# Créer un nouveau projet avec Laravel Installer

```
C:\Windows\System32\cmd.exe - laravel new gestion-note
Microsoft Windows [version 10.0.19042.867]
(c) 2020 Microsoft Corporation. Tous droits réservés.

C:\wamp64\www\laravel>laravel new gestion-note
[REMOVED]
```

```
C:\Windows\System32\cmd.exe - laravel new gestion-note
Microsoft Windows [version 10.0.19042.867]
(c) 2020 Microsoft Corporation. Tous droits réservés.

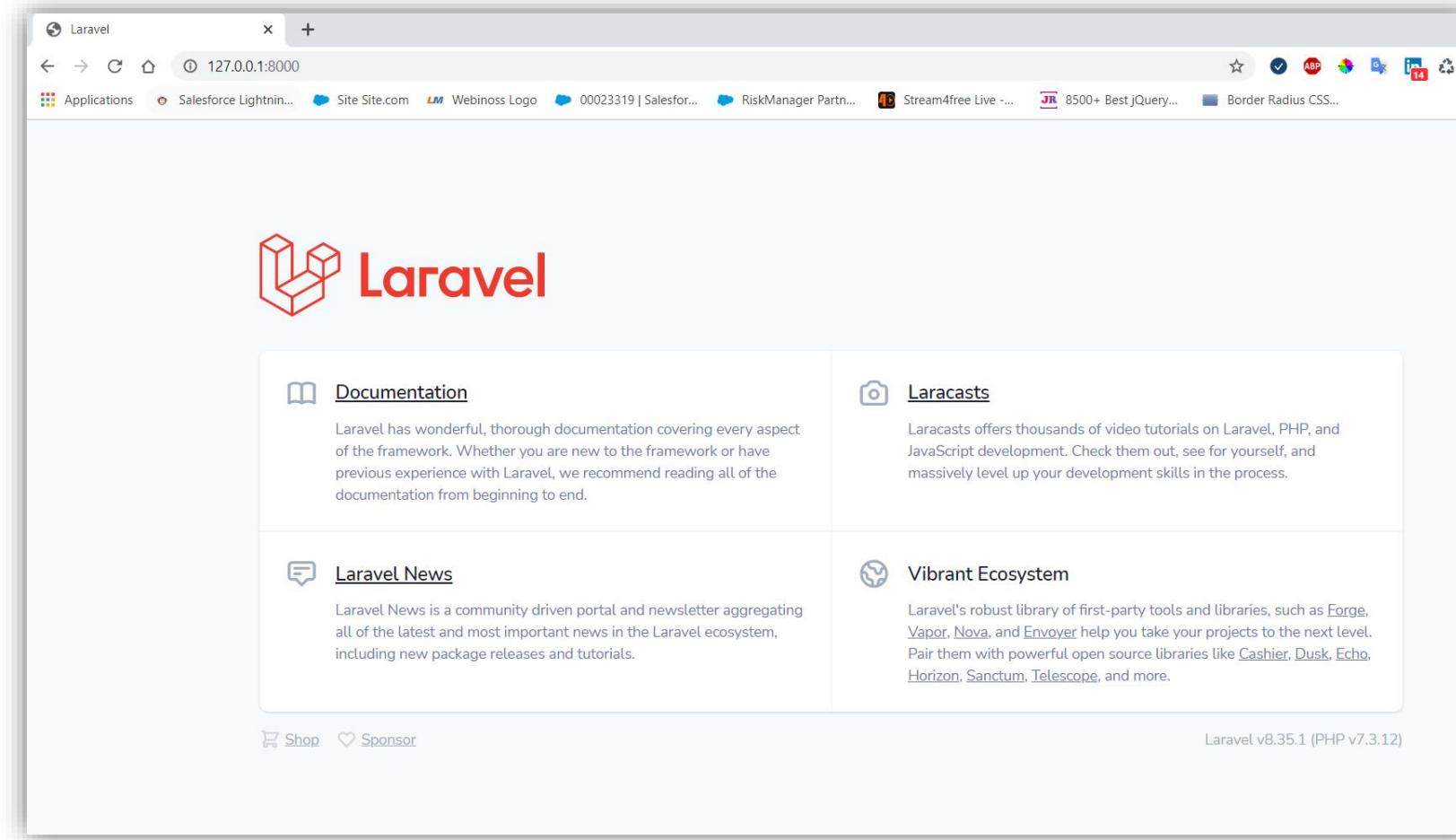
C:\wamp64\www\laravel>laravel new gestion-note
[REMOVED]

Warning from https://repo.packagist.org: You are using an outdated version of Composer. Composer 2 is now available
and you should upgrade. See https://getcomposer.org/2
Installing laravel/laravel (v8.5.15)
- Installing laravel/laravel (v8.5.15): Loading from cache
  Created project in C:\wamp64\www\laravel/gestion-note
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
  Loading composer repositories with package information
  Warning from https://repo.packagist.org: You are using an outdated version of Composer. Composer 2 is now available
  and you should upgrade. See https://getcomposer.org/2
  Updating dependencies (including require-dev)
```

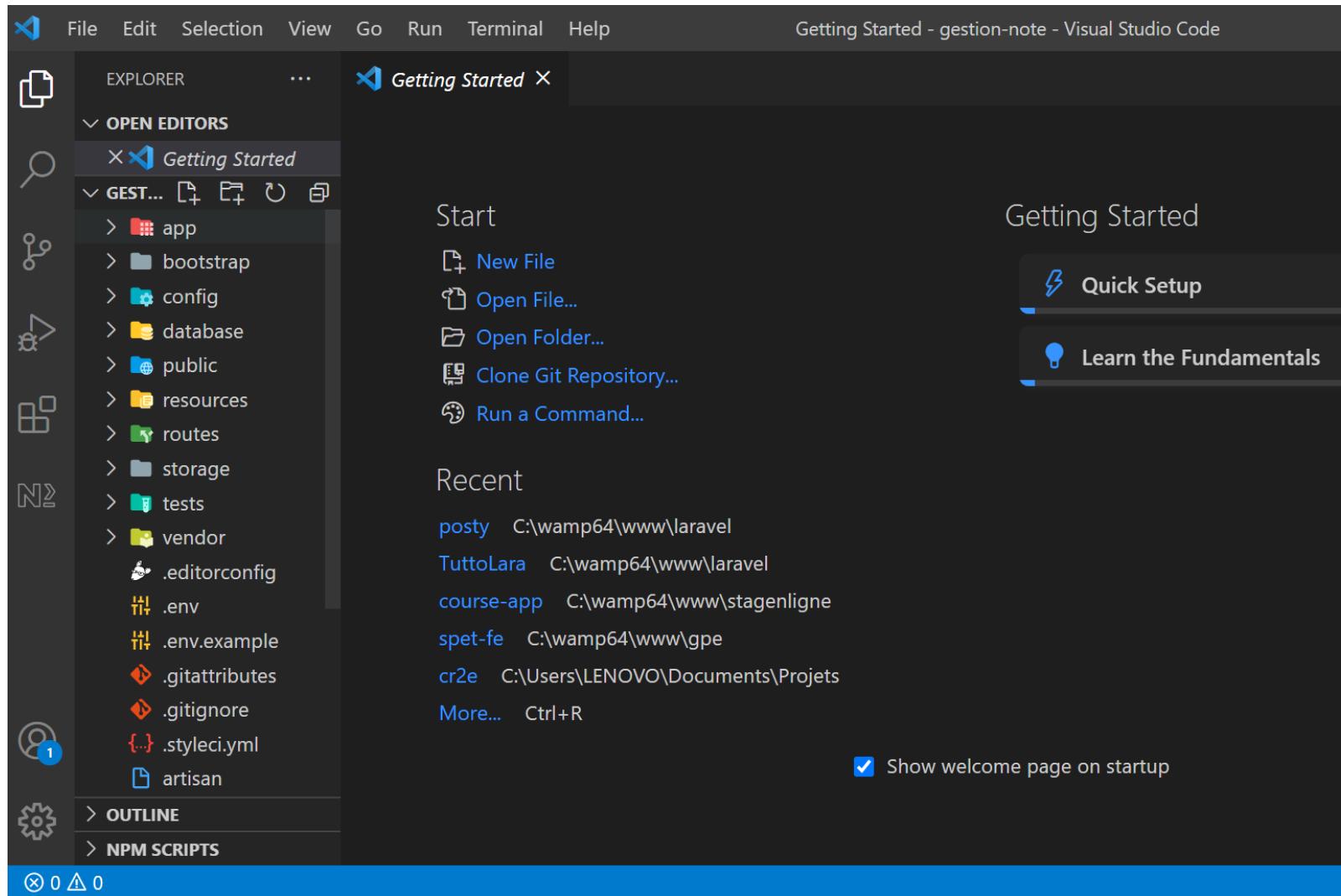
# Démarrer le projet

- Se placer dans le répertoire du projet
  - *cd gestion-note*
- Démarrer le serveur d'application
  - *php artisan serve*
- Ouvrir l'application dans un navigateur
  - ✓ ouvrir ce lien dans votre navigateur : <http://127.0.0.1:8000>

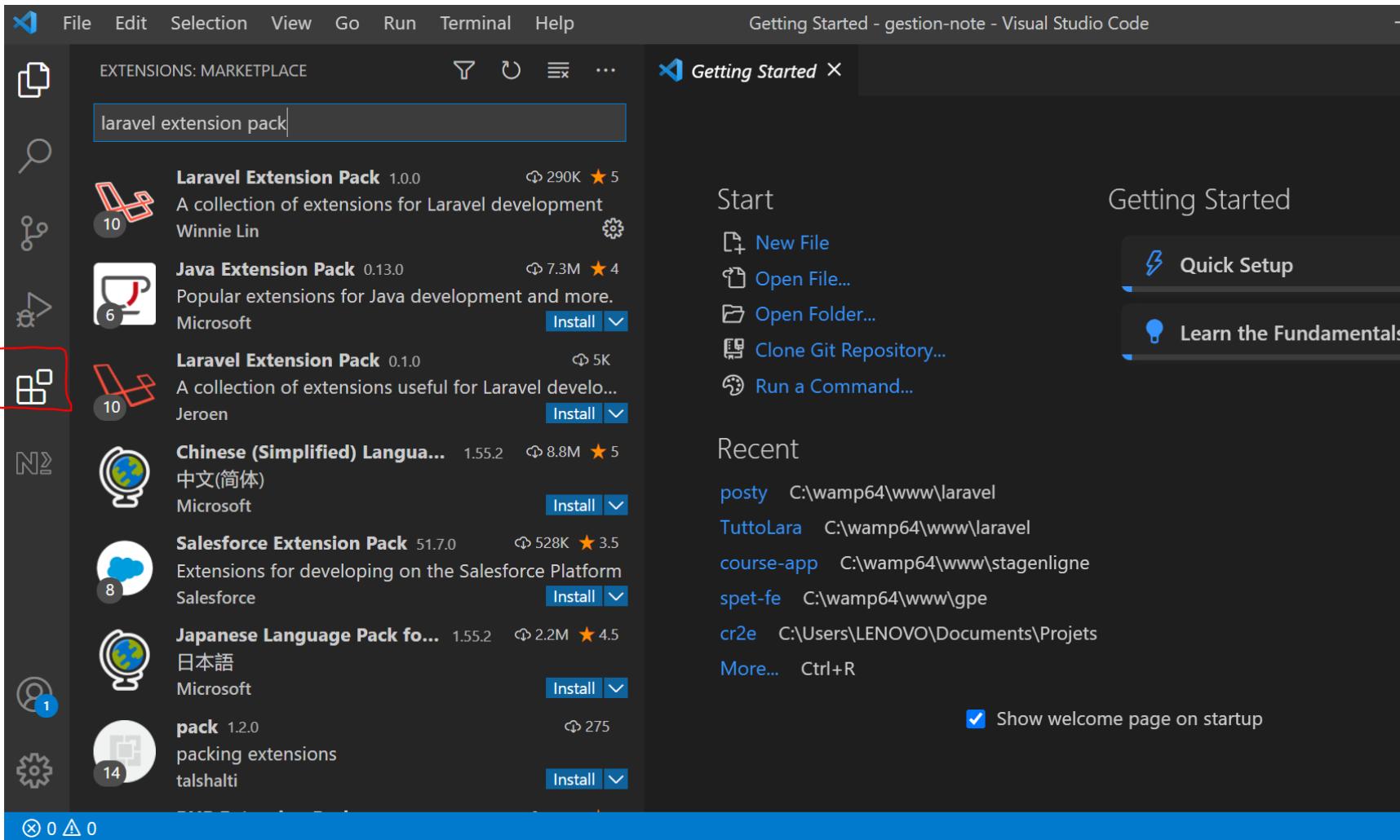
# Première page



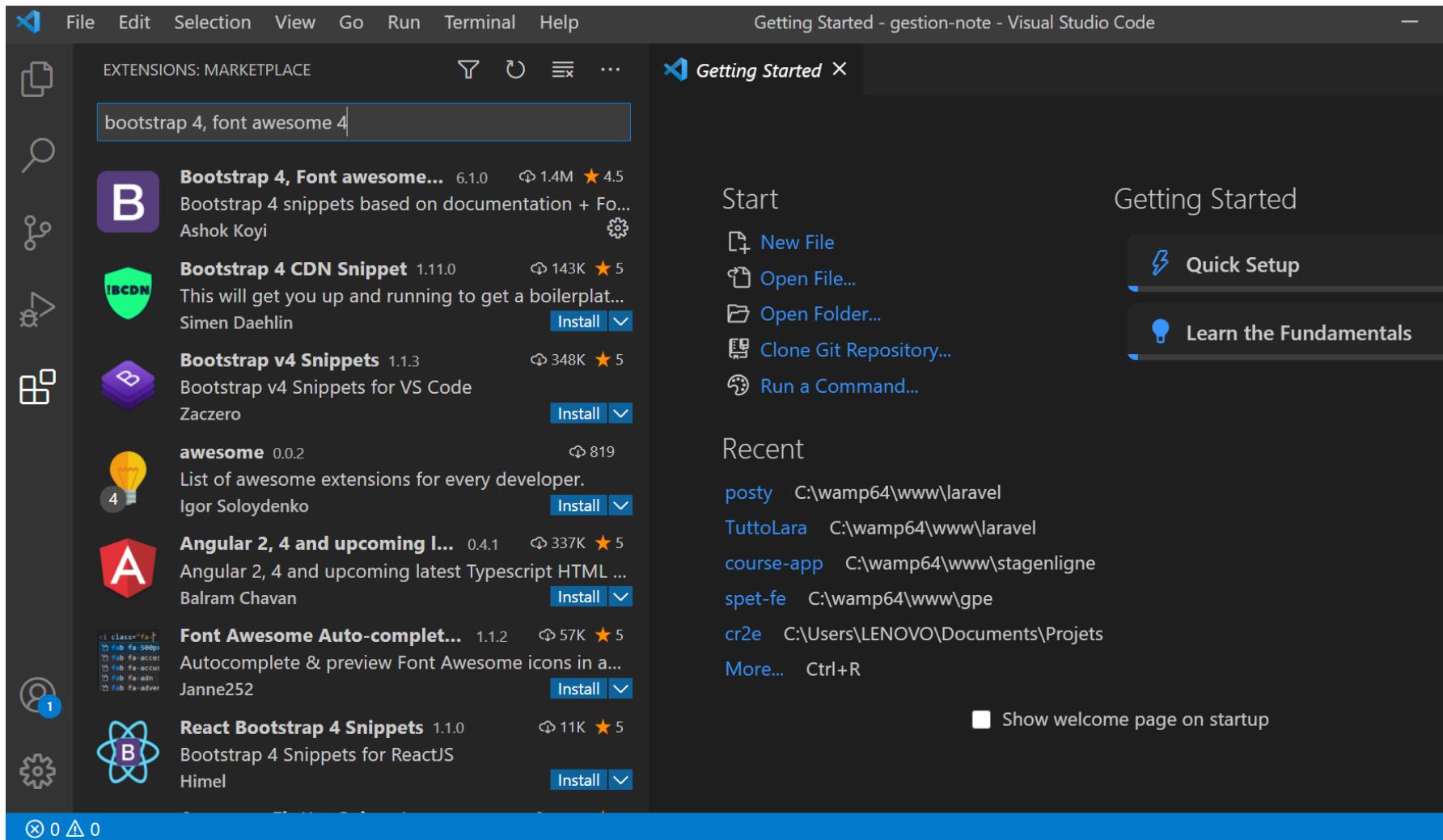
# Ouvrir le projet dans l'éditeur de code: VsCode



# Installer les extensions pour le Framework Laravel extension pack



# Installer les extensions pour le Framework Bootstrap 4, Font awesome 4

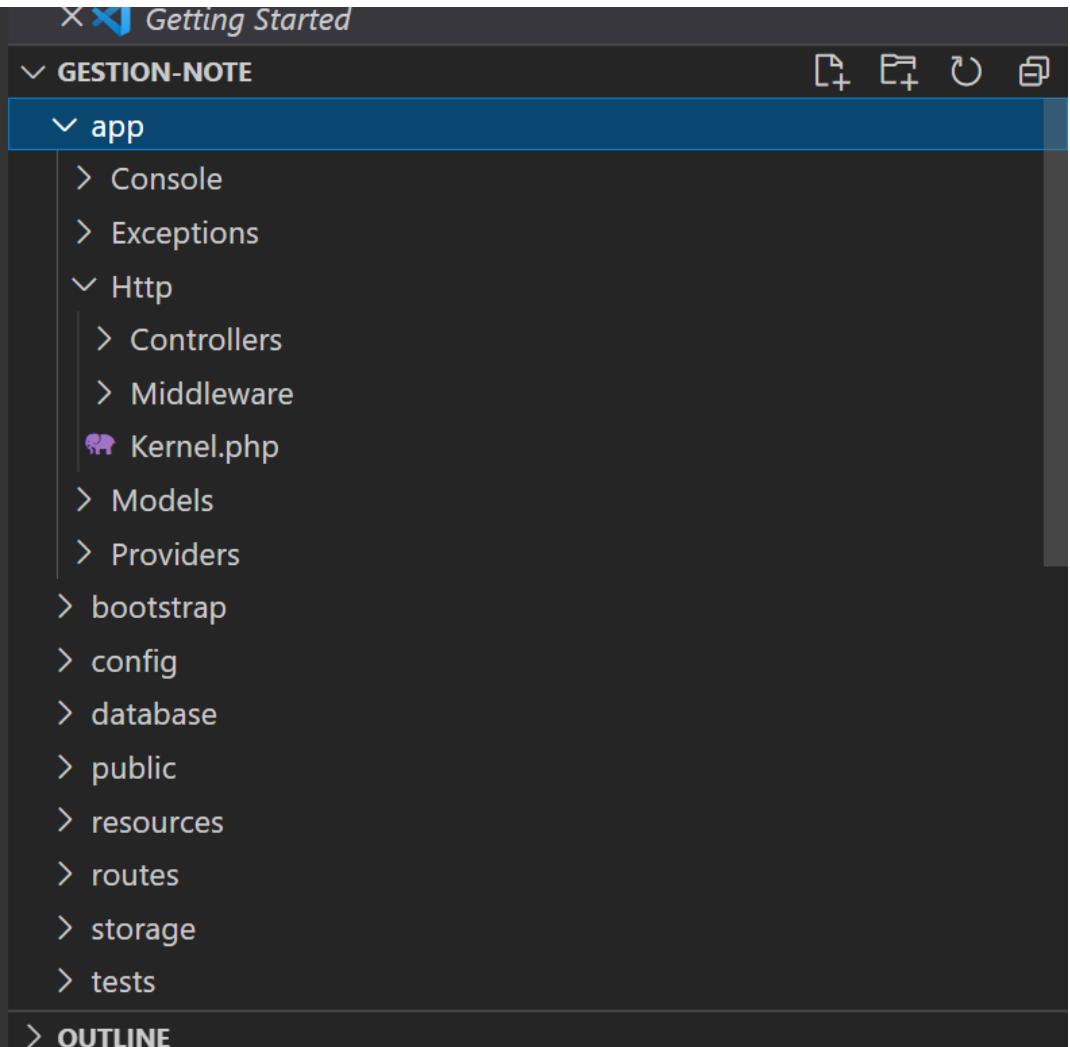


# Structure du projet

Un projet Laravel est ainsi organisé pour permettre à tous les développeurs de suivre la même structuration afin de se retrouver plus facilement.

# Structure du projet

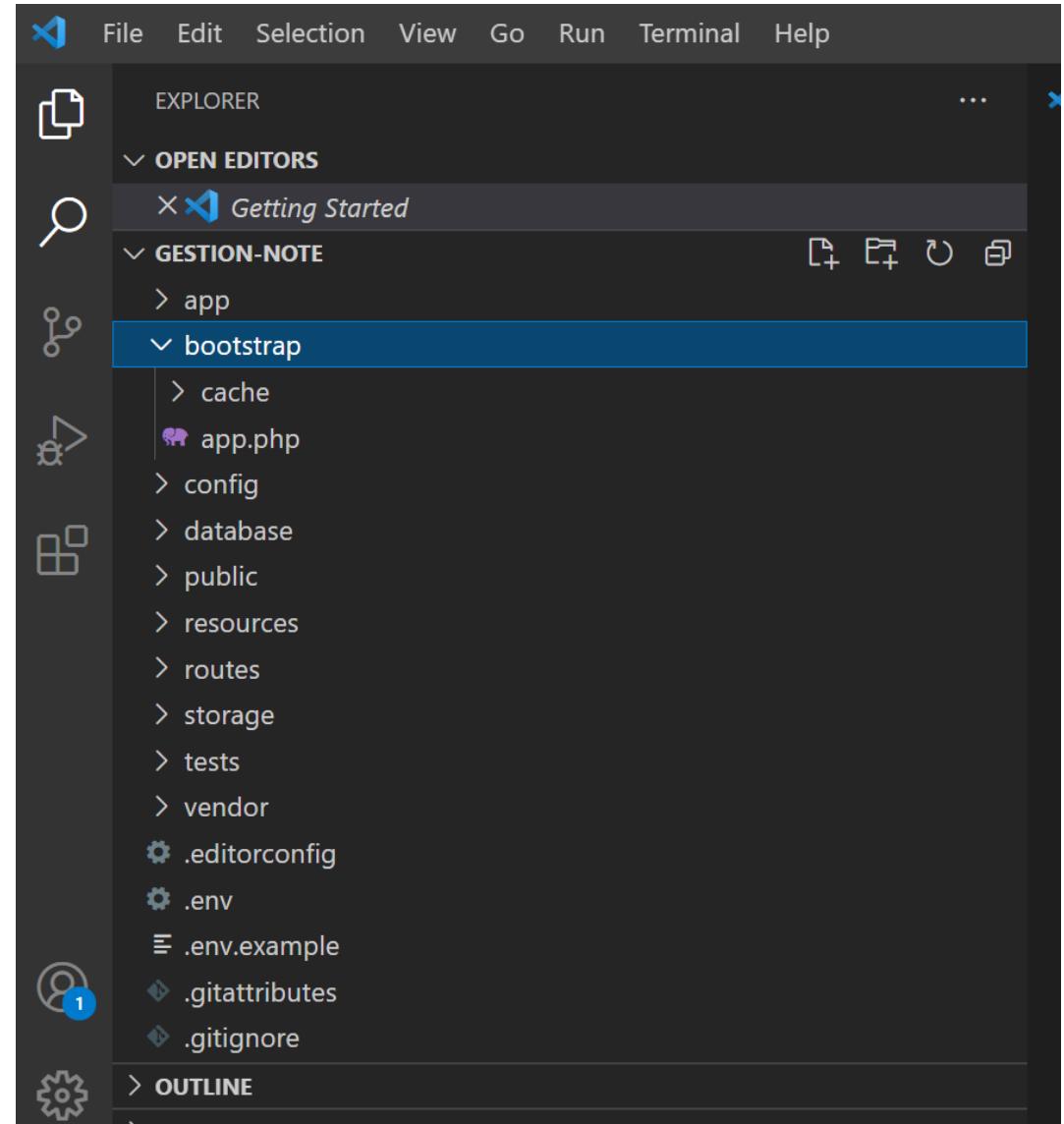
- Dossier app
  - Le répertoire « *app* » contient le code principal de l'application Laravel.
  - Presque toutes les classes de l'application se trouveront dans ce répertoire.



# Structure du projet

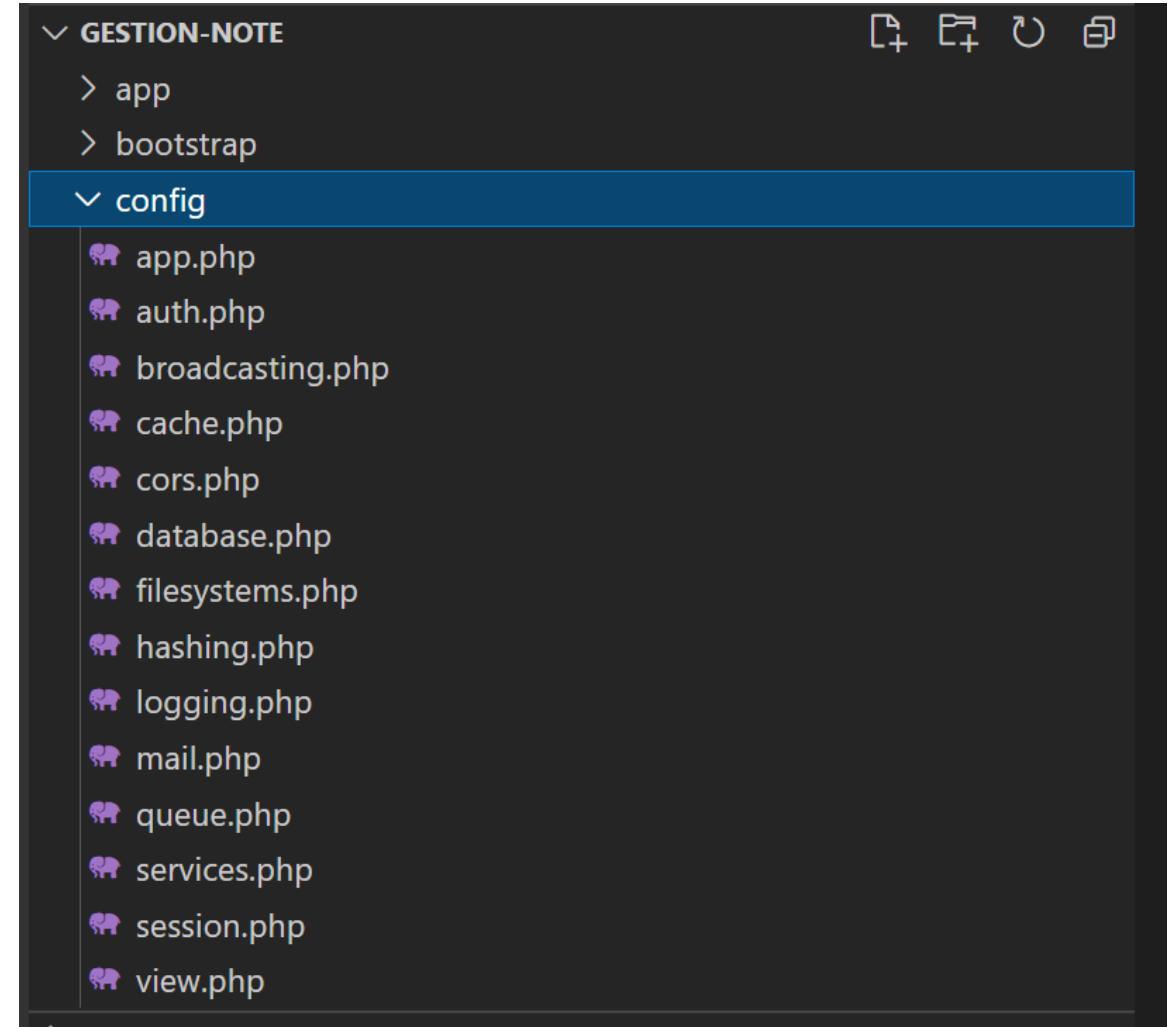
- Dossier Bootstrap

- Le répertoire « *bootstrap* » contient le fichier *app.php* qui amorce le framework. Ce répertoire contient également un répertoire de cache qui contient des fichiers générés par le Framework pour l'optimisation des performances, tels que les fichiers de cache d'itinéraire et de services. Vous ne devriez généralement pas avoir besoin de modifier les fichiers de ce répertoire.



# Structure du projet

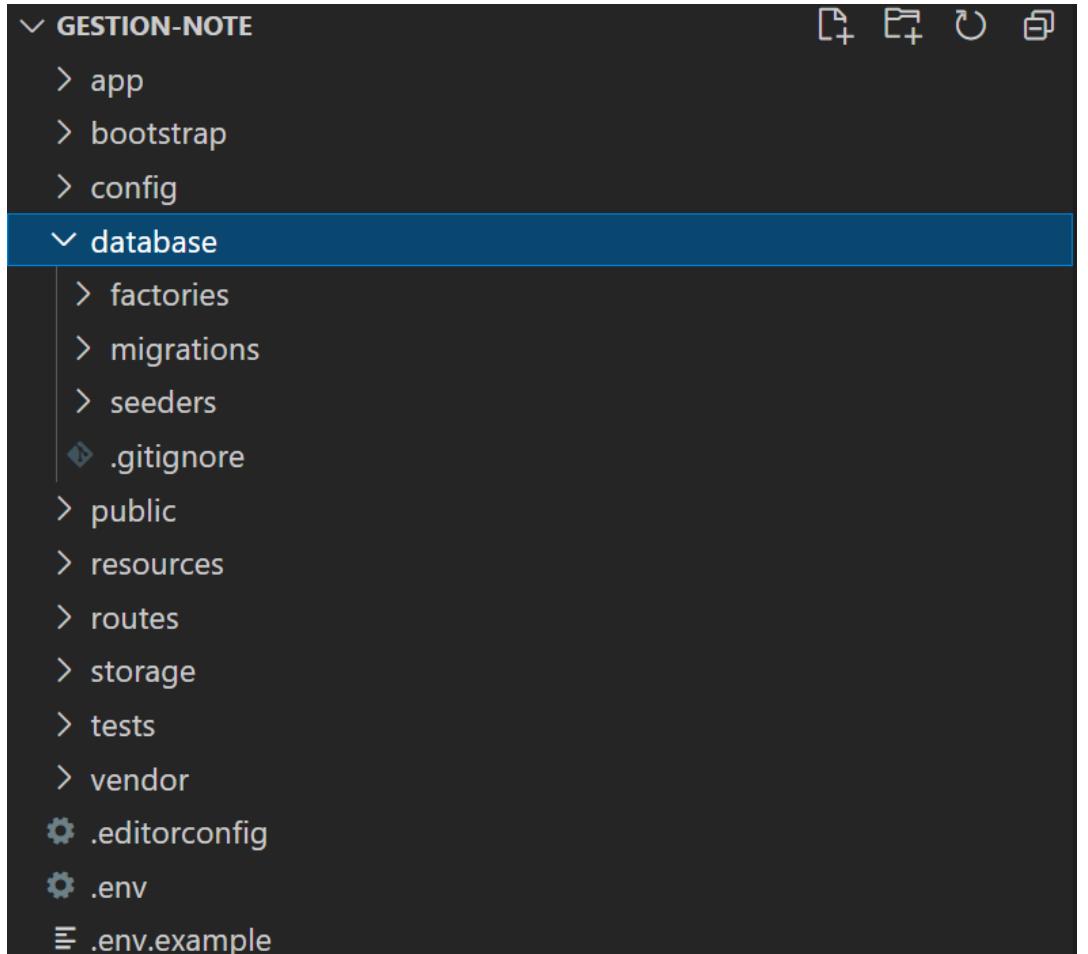
- Dossier Config
  - Le répertoire de configuration « **config** », comme son nom l'indique, contient tous les fichiers de configuration. C'est une excellente idée de lire tous ces fichiers et de vous familiariser avec toutes les options qui s'offrent à vous.



# Structure du projet

- Dossier Database

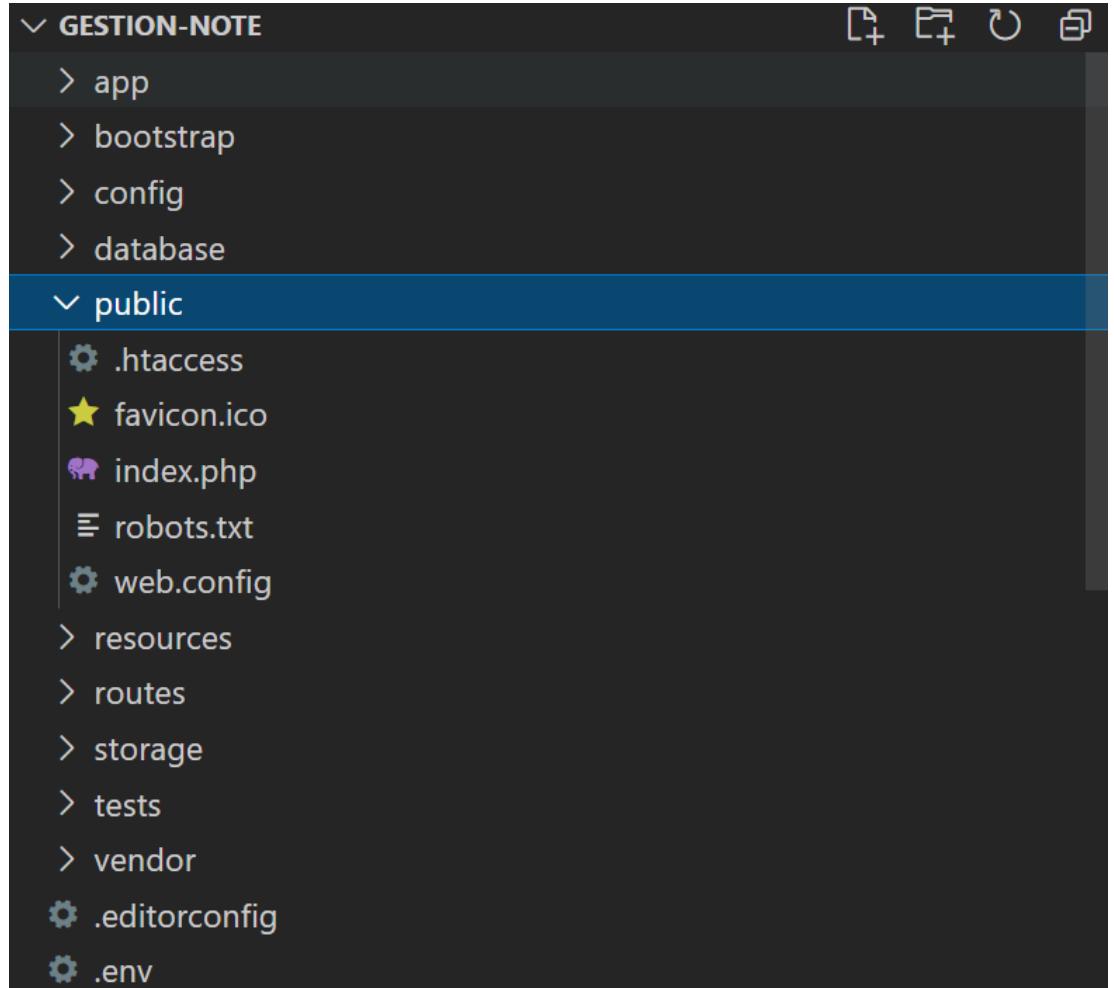
- Le répertoire de base de données « **database** » contient vos migrations de base de données, vos fabriques de modèles et vos semences. Si vous le souhaitez, vous pouvez également utiliser ce répertoire pour contenir une base de données SQLite.



# Structure du projet

- Dossier Public

- Le répertoire « **public** » contient le fichier **index.php**, qui est le point d'entrée de toutes les requêtes entrant dans votre application et configure le chargement automatique. Ce répertoire héberge également vos ressources telles que les images, JavaScript et CSS.



# Structure du projet

- Dossier Resources

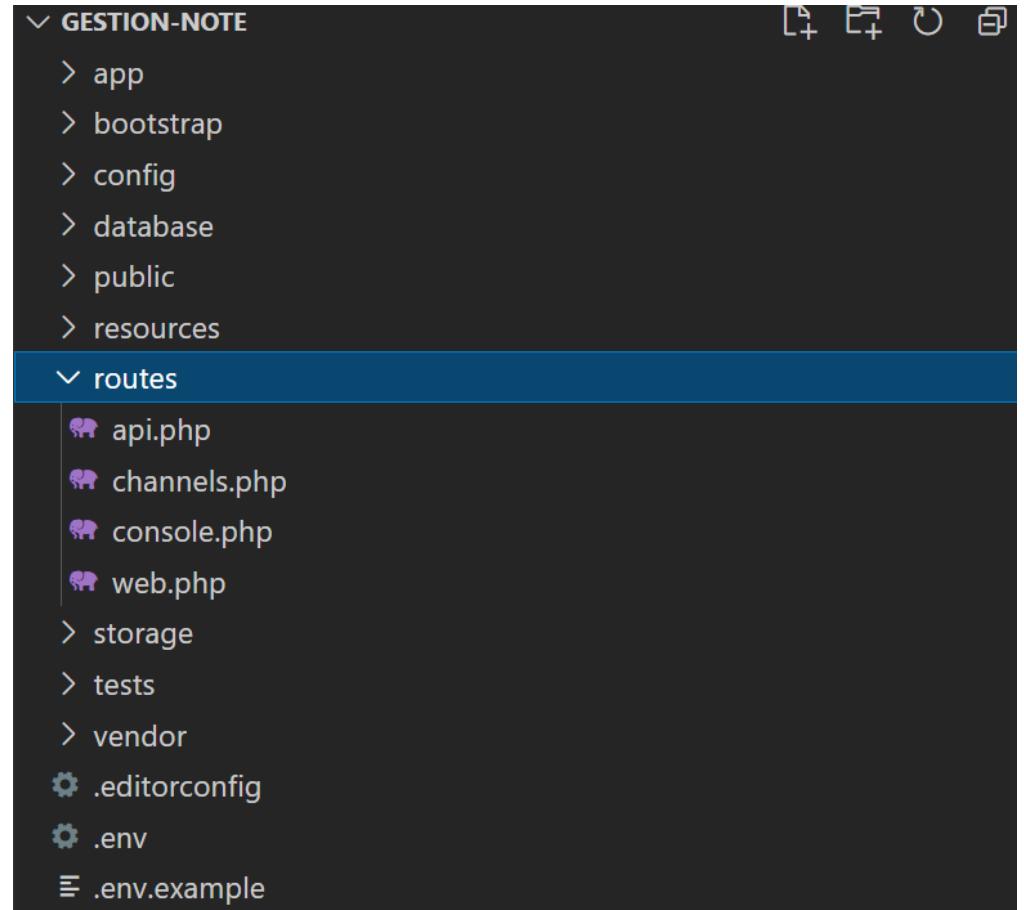
- Le répertoire « *resources* »  
contient vos vues ainsi que  
vos actifs bruts non compilés  
tels que CSS ou JavaScript. Ce  
répertoire contient également  
tous vos fichiers de langue.

```
▽ GESTION-NOTE
  > app
  > bootstrap
  > config
  > database
  > public
  ▽ resources
    > css
    > js
    > lang
    > views
    > routes
    > storage
    > tests
    > vendor
    ⚙ .editorconfig
    ⚙ .env
    ⌂ .env.example
```

# Structure du projet

- Dossier Routes

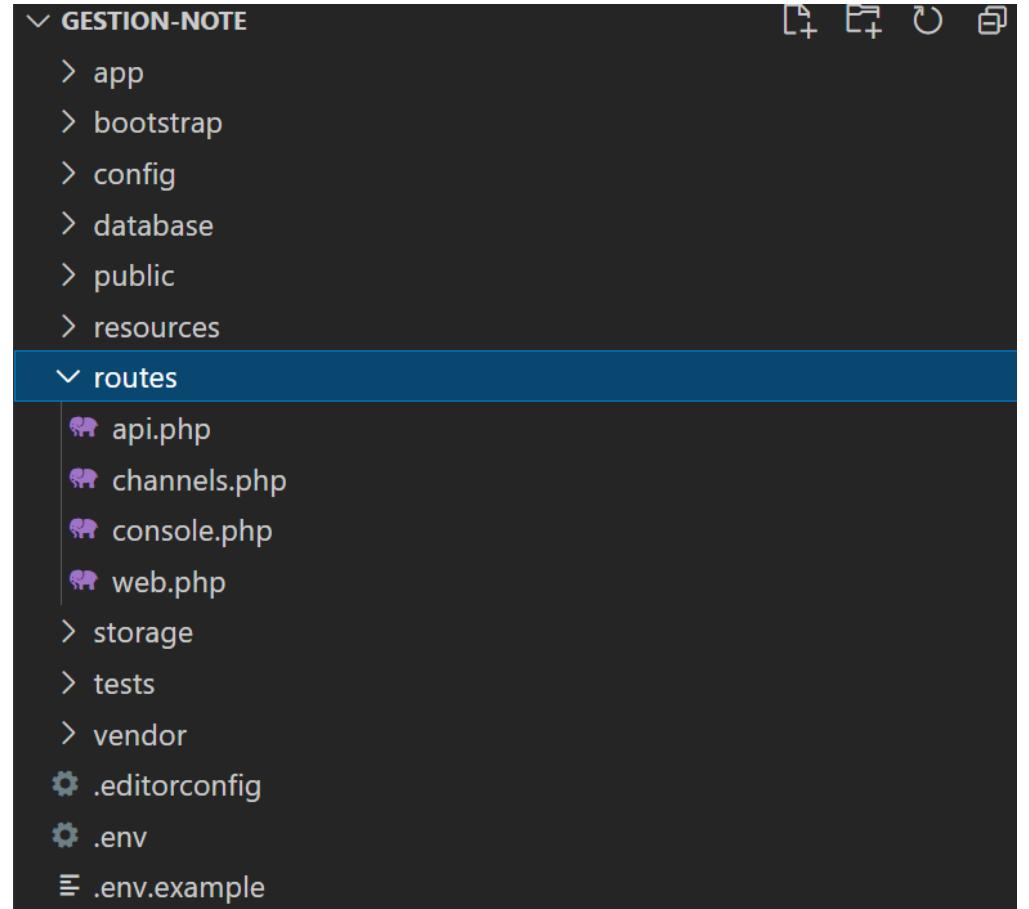
Le répertoire « *routes* » contient toutes les définitions d'itinéraire pour votre application. Par défaut, plusieurs fichiers de route sont inclus avec Laravel: *web.php*, *api.php*, *console.php* et *channels.php*.



# Structure du projet

- Dossier Routes : web.php

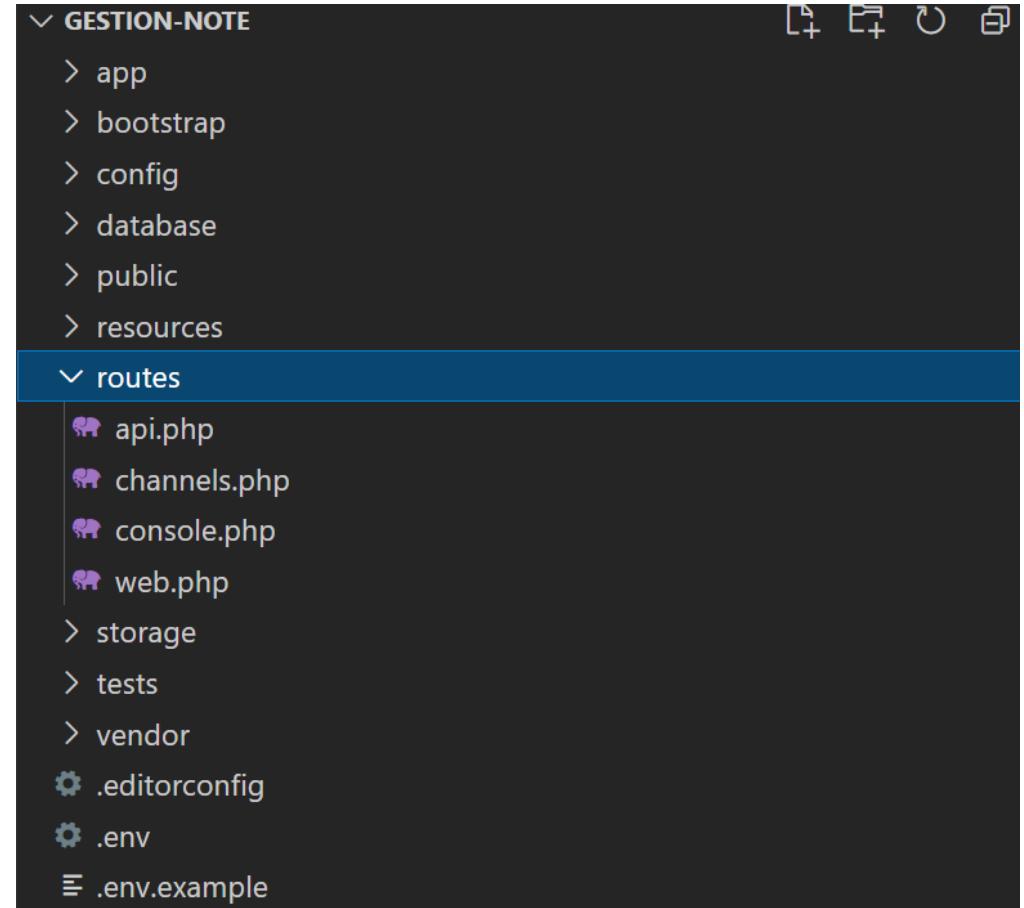
Le fichier « **web.php** » contient les routes que **RouteServiceProvider** place dans le groupe middleware Web, qui fournit l'état de la session, la protection CSRF et le cryptage des cookies.



# Structure du projet

- Dossier Routes : api.php

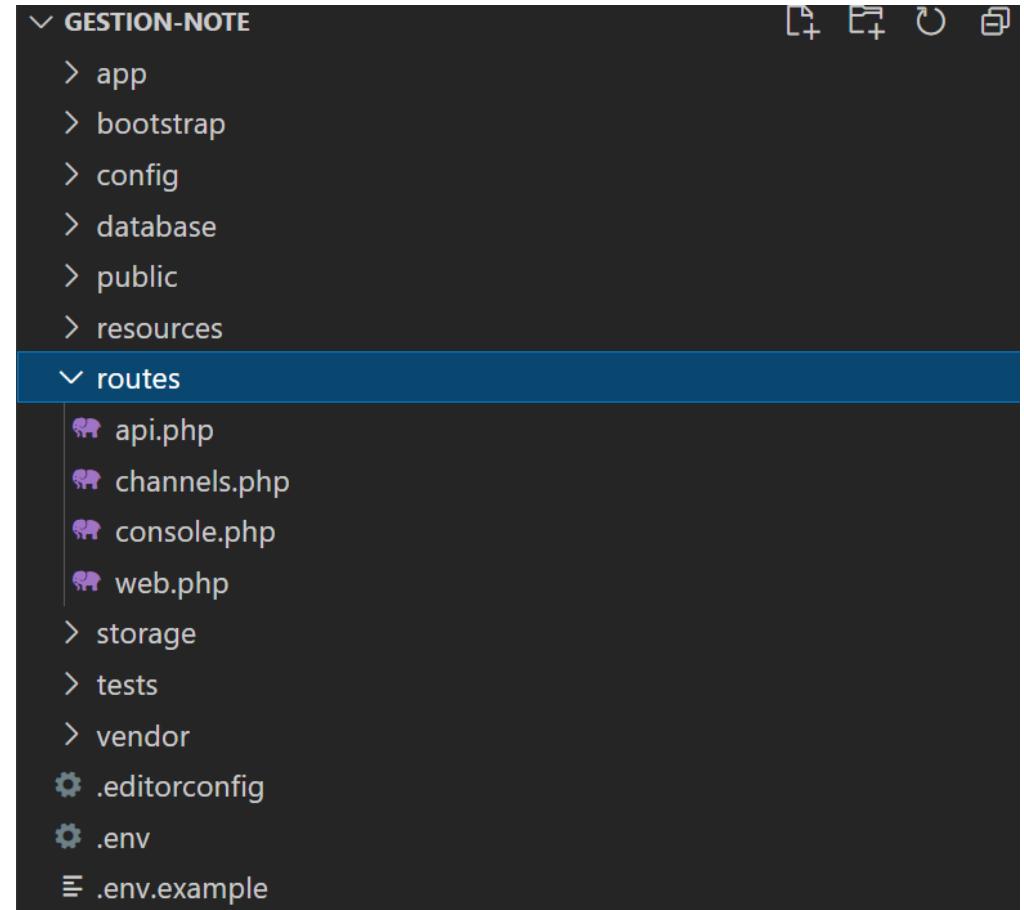
Le fichier « *api.php* » contient les routes que *RouteServiceProvider* place dans le groupe de middleware api. Ces routes sont destinées à être sans état, donc les demandes entrant dans l'application via ces routes sont destinées à être authentifiées via des jetons et n'auront pas accès à l'état de session.



# Structure du projet

- Dossier Routes : `console.php`

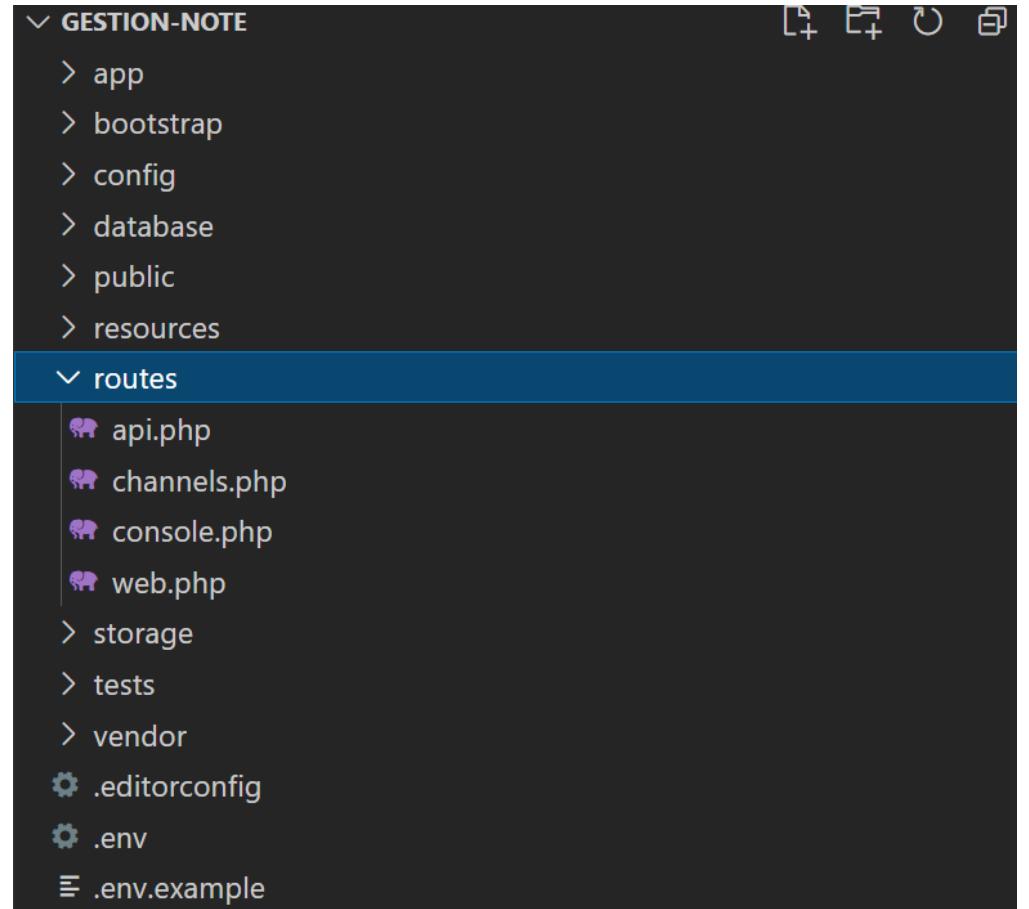
Le fichier « `console.php` » est l'endroit où vous pouvez définir toutes vos commandes de console basées sur la fermeture. Chaque fermeture est liée à une instance de commande permettant une approche simple pour interagir avec les méthodes IO de chaque commande. Même si ce fichier ne définit pas les routes HTTP, il définit des points d'entrée (routes) basés sur la console dans votre application.



# Structure du projet

- Dossier Routes : channels.php

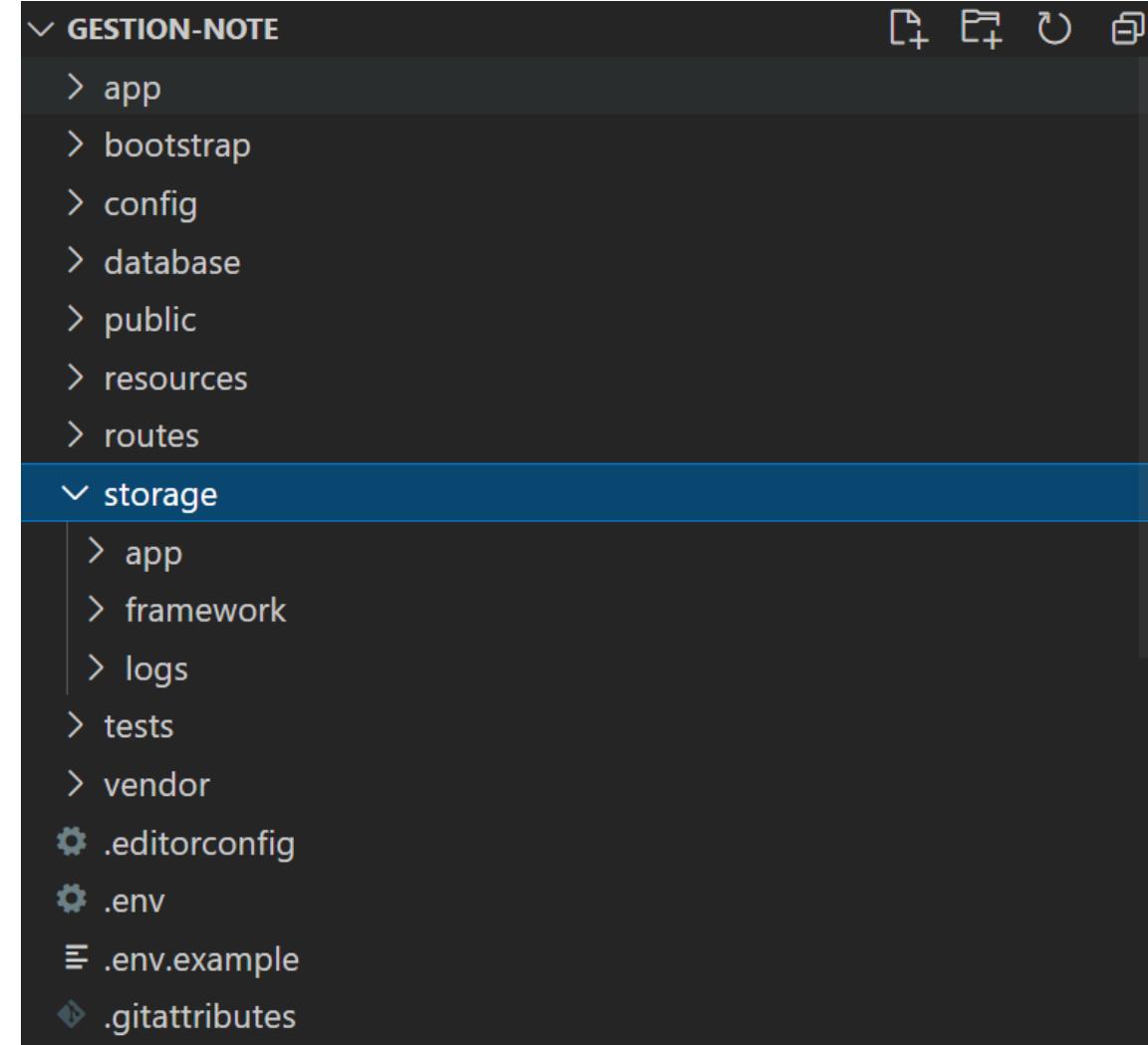
Le fichier « **channels.php** » est l'endroit où vous pouvez enregistrer tous les canaux de diffusion d'événements pris en charge par votre application.



# Structure du projet

- Dossier Storage

Le répertoire « *storage* » contient les journaux, les modèles Blade compilés, les sessions basées sur des fichiers, les caches de fichiers et d'autres fichiers générés par le framework. Le répertoire de l'application peut être utilisé pour stocker tous les fichiers générés par votre application. Le répertoire framework est utilisé pour stocker les fichiers et les caches générés par l'application. Enfin, le répertoire logs contient les fichiers journaux.



# Structure du projet

- Dossier Tests

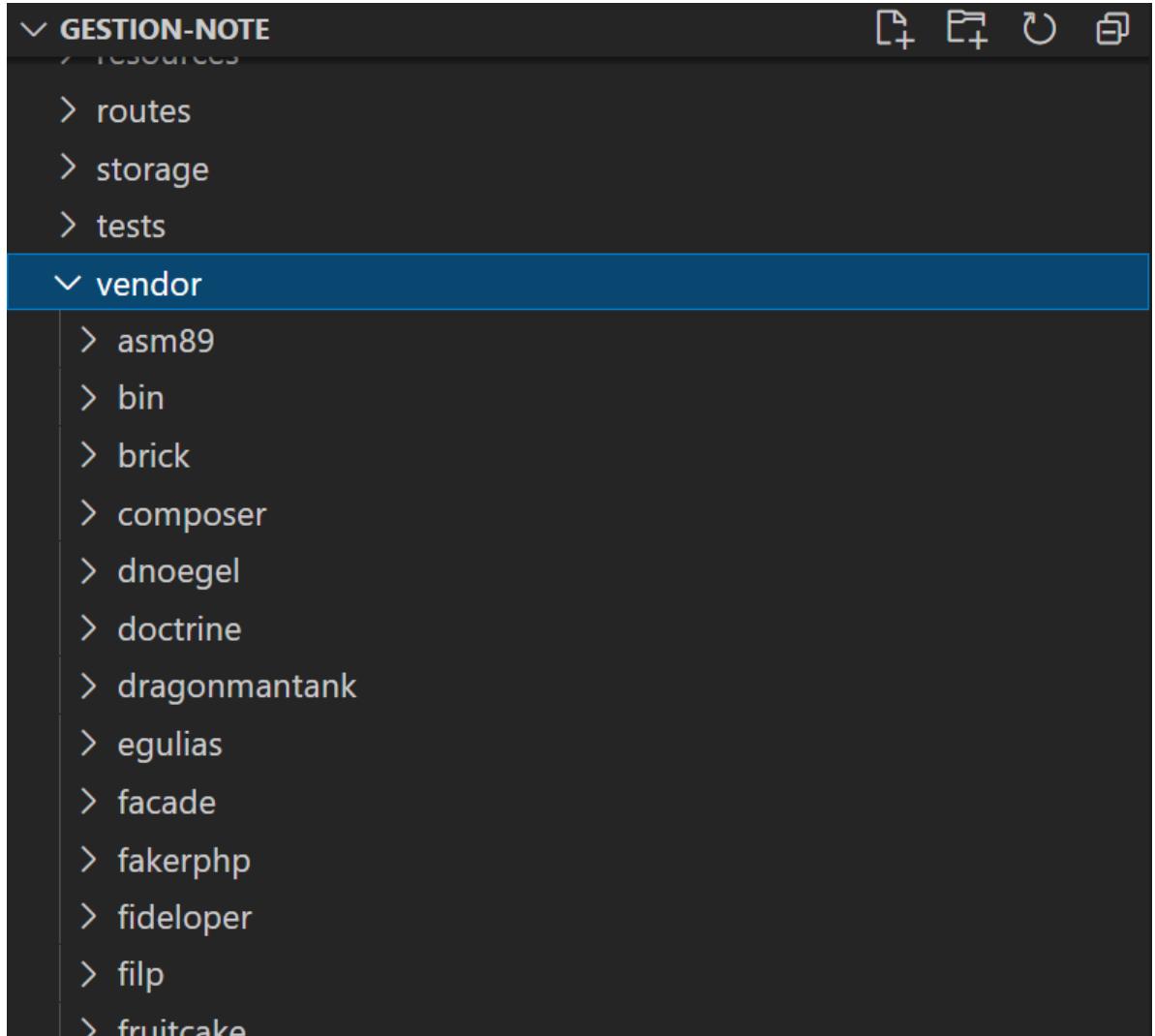
Le répertoire des tests contient vos tests automatisés. Des exemples de tests unitaires et de tests de fonctionnalités PHPUnit sont fournis prêts à l'emploi. Chaque classe de test doit être suffixée du mot Test.

```
└── GESTION-NOTE
    ├── app
    ├── bootstrap
    ├── config
    ├── database
    ├── public
    ├── resources
    ├── routes
    ├── storage
    └── tests
        ├── Feature
        ├── Unit
        ├── CreatesApplication.php
        ├── TestCase.php
        ├── vendor
        ├── .editorconfig
        ├── .env
        └── .env.example
```

# Structure du projet

- Dossier Vendor

Le répertoire du « **vendor** » contient les dépendances de Composer.



# Les packages

- Les packages sont le principal moyen d'ajouter des fonctionnalités à Laravel.
- Les packages peuvent être un excellent moyen de travailler avec des dates comme « Carbon » ou un package qui vous permet d'associer des fichiers à des modèles Eloquent tels que « Laravel Media Library » de Spatie.

# Créer sa première page

- Ouvrir le fichier routes/web.php
- Nous y trouvons par défaut la page qui se charge sur le <http://127.0.0.1:8000/>
- Cette fonction explique qu'on a une nouvelle route accessible grâce à la méthode « http get » et que la fonction doit retourner la page '**welcome**' qui s'appelle en réalité **welcome.blade.php** mais l'extension **blade.php** n'est jamais précisée dans le view.

```
routes > 🐄 web.php > ...
1  |?php
2
3  use Illuminate\Support\Facades\Route;
4
5  /*
6  |-----
7  | Web Routes
8  |-----
9  |
10 | Here is where you can register web routes for your application. These
11 | routes are loaded by the RouteServiceProvider within a group which
12 | contains the "web" middleware group. Now create something great!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
```

# Créer sa première page

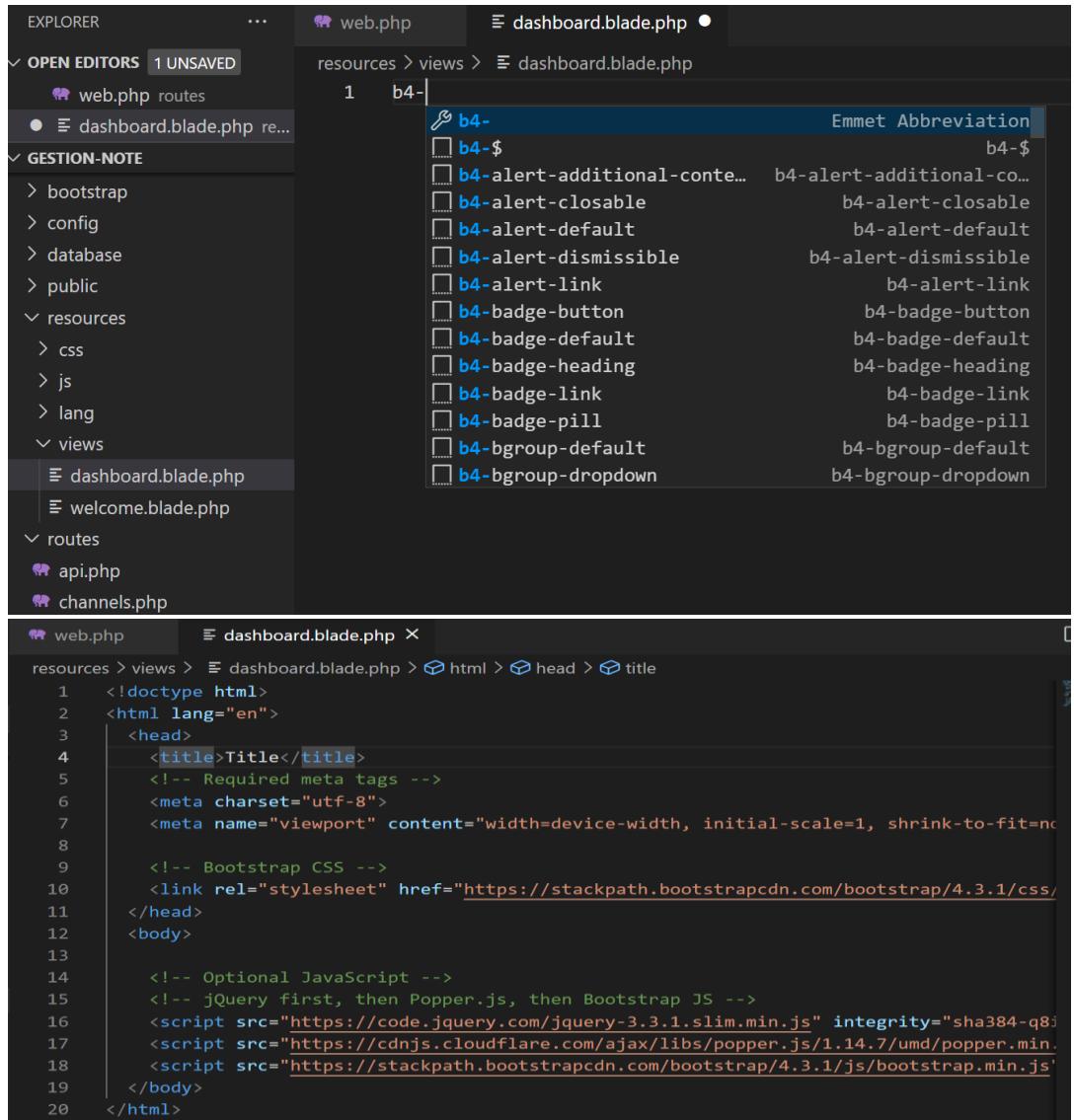
- Maintenant, nous allons copier et coller la route du **welcome**
- Nous allons l'appeler **dashboard** et pour la route, nous allons préciser « **/dashboard** »



```
routes > web.php > ...
11 | routes are loaded by the RouteServiceProvider when your application is published.
12 | contains the "web" middleware group. Now create
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19
20
21 Route::get('/dashboard', function () {
22     return view('dashboard');
23 });
24
```

# Créer sa première page

- Nous allons ensuite ouvrir le répertoire **resources/views** et créer notre fichier **dashboard.blade.php**
- Générer le contenu html avec bootstrap 4 en commençant par saisir b4- et sélectionner l'option b4-\$ et taper sur la touche « Enter »
- Ecrivez du code html dans le fichier **dashboard.blade.php** par exemple:  
« <h1>Bienvenue dans le tableau de bord</h1> »



The screenshot shows a code editor interface with two tabs: 'web.php' and 'dashboard.blade.php'. The 'dashboard.blade.php' tab is active, displaying the following code:

```
resources > views > dashboard.blade.php
1 <!doctype html>
2 <html lang="en">
3   <head>
4     <title>Title</title>
5     <!-- Required meta tags -->
6     <meta charset="utf-8">
7     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
8
9     <!-- Bootstrap CSS -->
10    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
11
12   <body>
13
14     <!-- Optional JavaScript -->
15     <!-- jQuery first, then Popper.js, then Bootstrap JS -->
16     <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8&gt;
17     <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/umd/popper.min.js" integrity="sha384-U9xgqKF9dIkhEW11Ej5&gt;
18     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-7&gt;
19   </body>
20 </html>
```

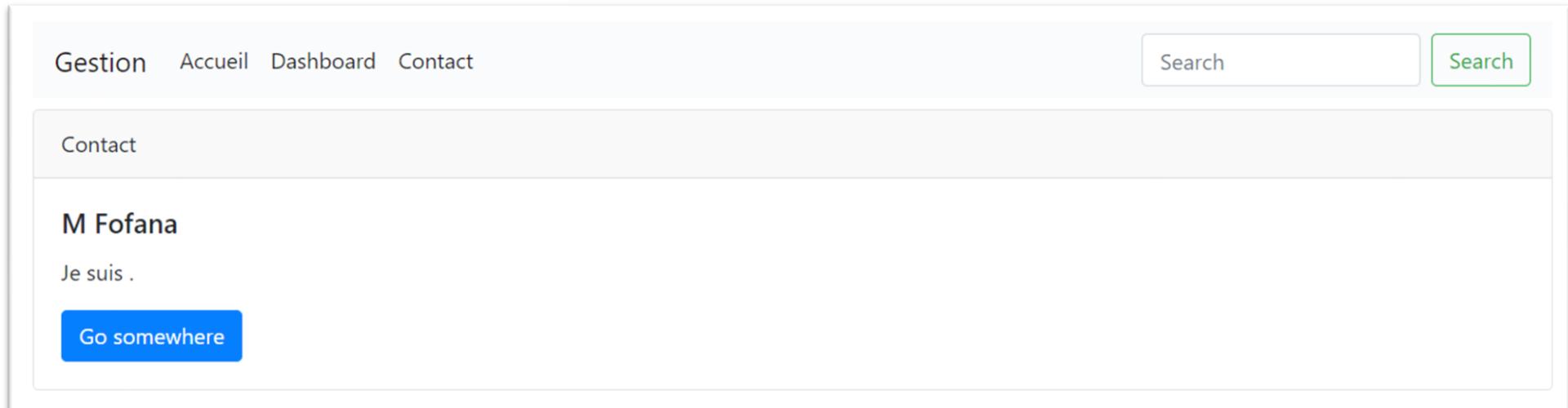
The Emmet Abbreviations panel on the right shows suggestions starting with 'b4-'. The first suggestion, 'b4-\$', is highlighted. Other suggestions include 'b4-alert-additional-content', 'b4-alert-closable', 'b4-alert-default', 'b4-alert-dismissible', 'b4-alert-link', 'b4-badge-button', 'b4-badge-default', 'b4-badge-heading', 'b4-badge-link', 'b4-badge-pill', 'b4-bgroup-default', and 'b4-bgroup-dropdown'. The 'Emmet Abbreviation' column lists the full Bootstrap class names corresponding to each abbreviation.

# Créer sa première page

- Félicitations !
- Vous venez de créer votre première page
- Maintenant pour la tester, il faut aller sur le navigateur et rajouter la route **/accueil** créé précédemment.

# Défi – Créer une seconde page et personnaliser la présentation

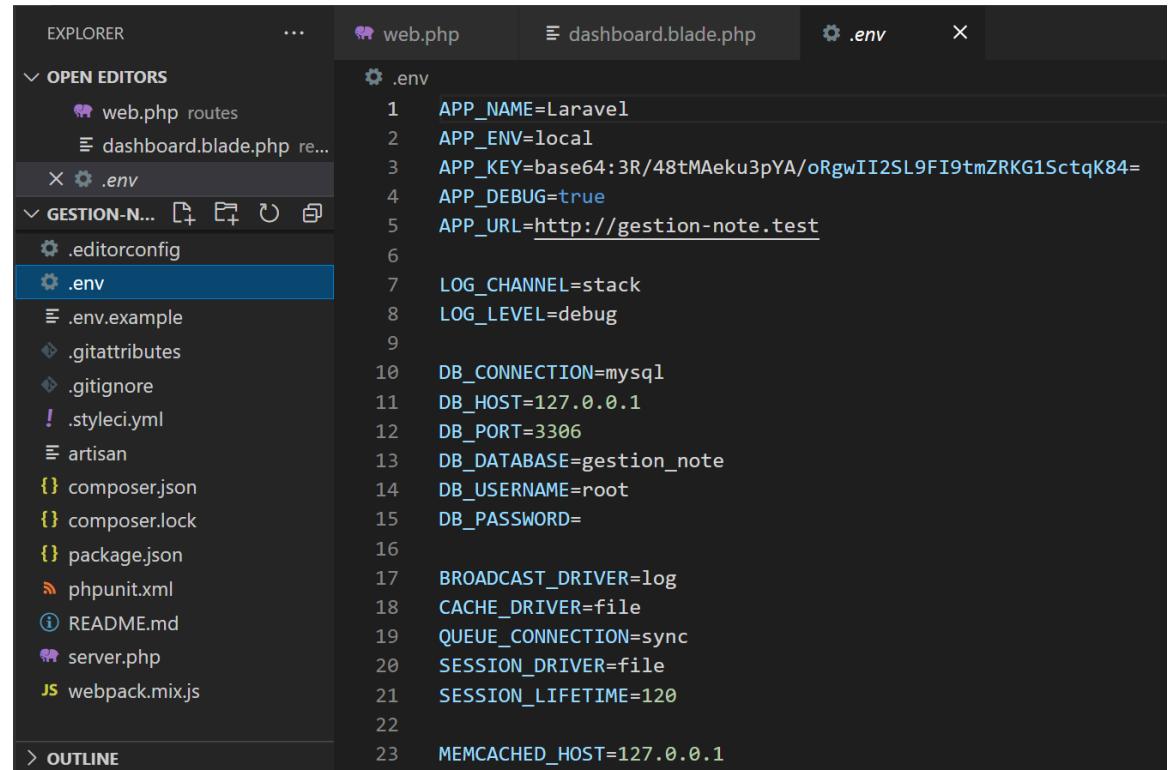
- Personnaliser la page Dashboard en ajoutant un menu, un pied de page
- Créer une seconde page qui doit se charger avec la route `/contact` et doit afficher une page dans la navigateur affichant les informations de contact de la personne concernée avec la même structuration que la page `dashboard.blade.php`



# Le fichier .env

Ce fichier comporte toutes les configurations de notre application comme :

- les détails de l'application,
- les identifiants de connexion à la base de données,
- les paramètres d'envoie d'emails
- Etc.

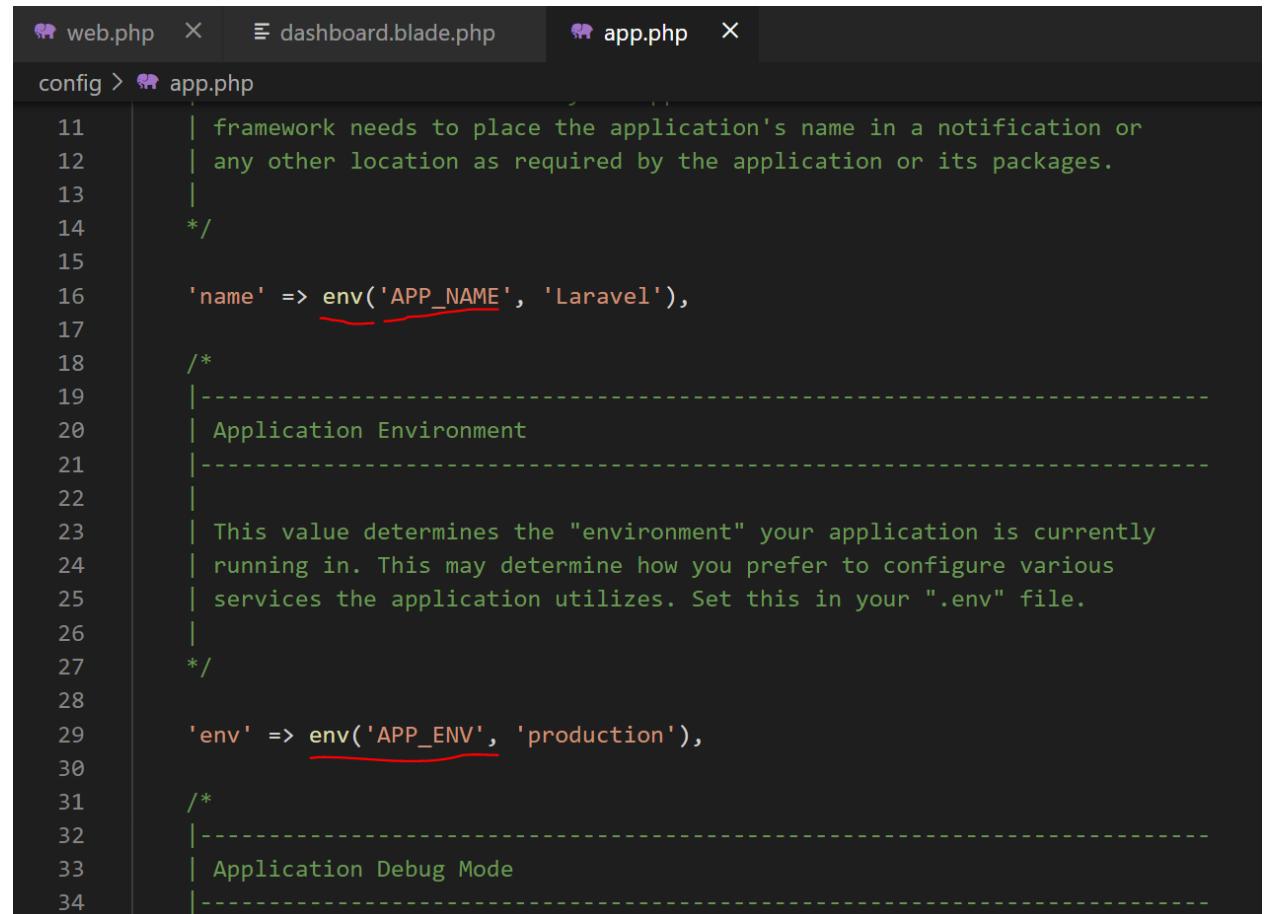


The screenshot shows a code editor interface with the ".env" file open. The left sidebar lists various files and folders, including "web.php", "routes", "dashboard.blade.php", ".env", ".editorconfig", ".env", ".env.example", ".gitattributes", ".gitignore", ".styleci.yml", "artisan", "composer.json", "composer.lock", "package.json", "phpunit.xml", "README.md", "server.php", and "webpack.mix.js". The right pane displays the contents of the ".env" file, which contains environment variables for a Laravel application. The variables include APP\_NAME, APP\_ENV, APP\_KEY, APP\_DEBUG, APP\_URL, LOG\_CHANNEL, LOG\_LEVEL, DB\_CONNECTION, DB\_HOST, DB\_PORT, DB\_DATABASE, DB\_USERNAME, DB\_PASSWORD, BROADCAST\_DRIVER, CACHE\_DRIVER, QUEUE\_CONNECTION, SESSION\_DRIVER, SESSION\_LIFETIME, and MEMCACHED\_HOST.

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=base64:3R/48tMAeku3pYA/oRgwII2SL9FI9tmZRKG1SctqK84=
APP_DEBUG=true
APP_URL=http://gestion-note.test
LOG_CHANNEL=stack
LOG_LEVEL=debug
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gestion_note
DB_USERNAME=root
DB_PASSWORD=
BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120
MEMCACHED_HOST=127.0.0.1
```

# Le fichiers .env et config/app.php

Ces informations de configuration du fichier `.env` sont utilisées un peu partout dans toute l'application pour éviter de dupliquer les données de paramétrage.

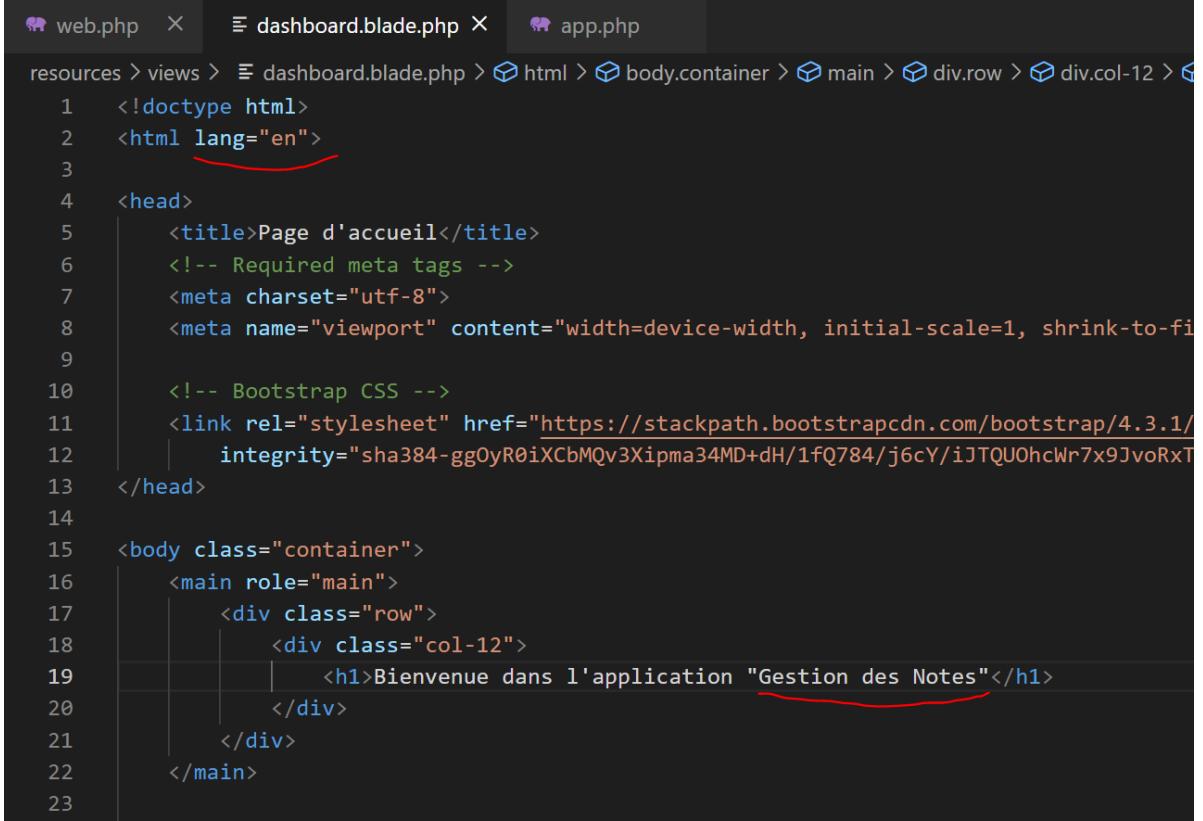


```
config > app.php
11 |     framework needs to place the application's name in a notification or
12 |     any other location as required by the application or its packages.
13 |
14 */
15
16 'name' => env('APP_NAME', 'Laravel'),
17
18 /*
19 | -----
20 | Application Environment
21 | -----
22 |
23 | This value determines the "environment" your application is currently
24 | running in. This may determine how you prefer to configure various
25 | services the application utilizes. Set this in your ".env" file.
26 |
27 */
28
29 'env' => env('APP_ENV', 'production'),
30
31 /*
32 | -----
33 | Application Debug Mode
34 | -----
```

# Les fichier .env, config/\* et les pages blade.php

Par exemple, l'attribut `lang` de html doit prendre la valeur dans les fichiers de configuration de même que le nom de l'application.

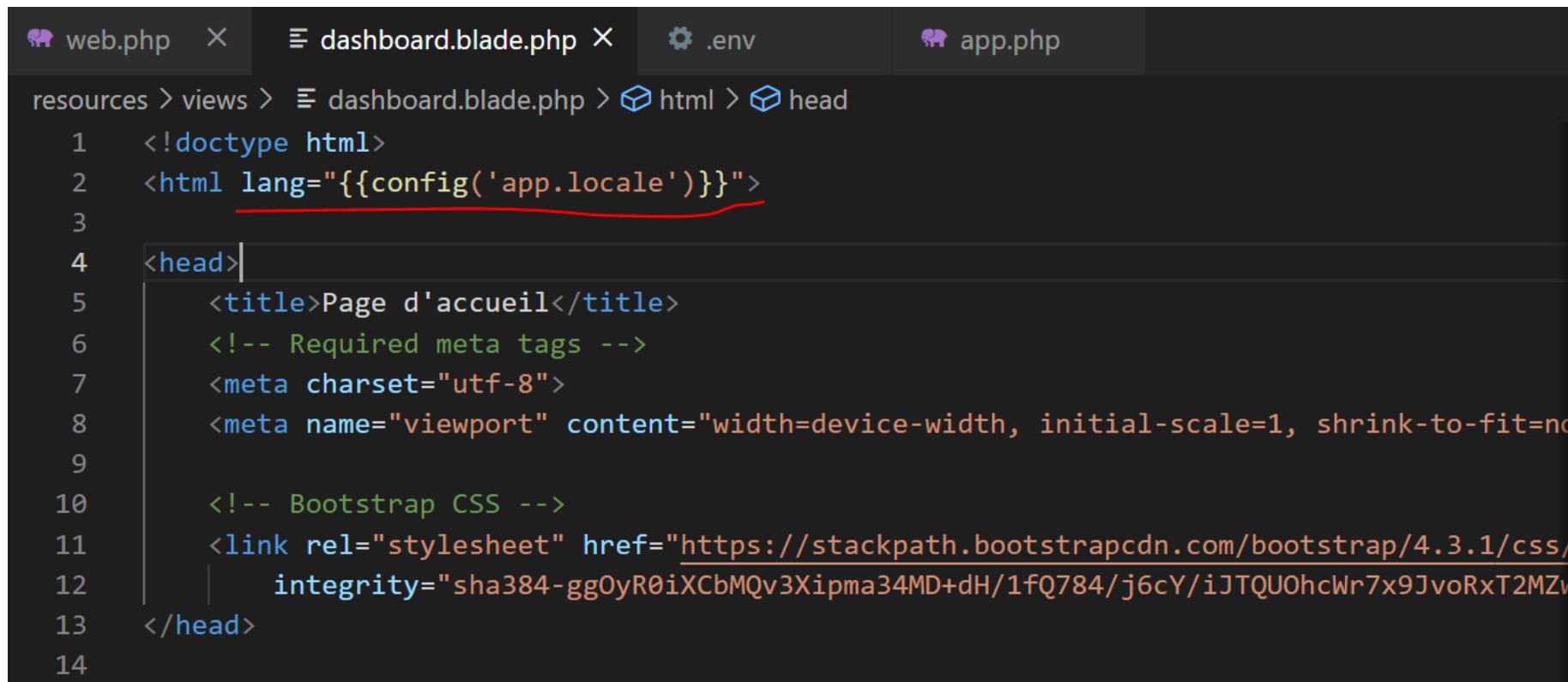
Il est possible d'accéder aux données de `.env` en utilisant la fonction `env('cle_config')` mais ceci n'est pas conseillé car en production, celle-ci ne marchera pas. Alors, il est conseillé de récupérer plutôt les informations de configuration depuis le fichier de configuration depuis le répertoire config.



```
1  <!doctype html>
2  <html lang="en"> -----^
3
4  <head>
5      <title>Page d'accueil</title>
6      <!-- Required meta tags -->
7      <meta charset="utf-8">
8      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no, viewport-fit=cover">
9
10     <!-- Bootstrap CSS -->
11     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxTzE" crossorigin="anonymous">
12
13 </head>
14
15 <body class="container">
16     <main role="main">
17         <div class="row">
18             <div class="col-12">
19                 <h1>Bienvenue dans l'application "Gestion des Notes"!</h1> -----^
20             </div>
21         </div>
22     </main>
23
24 
```

# Accéder aux paramétrages de configuration depuis la page blade.php

- On peut accéder à n'importe quel paramètre défini dans un fichier dans le répertoire config depuis notre page **blade.php** en utilisant la fonction ***config('nom\_fichier.paramètre\_à\_récupérer')***
- NB: Dans blade, pour afficher une valeur dynamique, on utilise ***{{ }}*** et entre les deux accolades, on ajoute la contenu à afficher.



The screenshot shows a code editor with several tabs at the top: 'web.php', 'dashboard.blade.php', '.env', and 'app.php'. The 'dashboard.blade.php' tab is active. The file path in the sidebar is 'resources > views > dashboard.blade.php > html > head'. The code in the editor is:

```
1  <!doctype html>
2  <html lang="{{config('app.locale')}}">
3
4  <head>
5      <title>Page d'accueil</title>
6      <!-- Required meta tags -->
7      <meta charset="utf-8">
8      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
9
10     <!-- Bootstrap CSS -->
11     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-gg0yR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZv" data-bbox="218 848 828 918"/>
12
13 </head>
14
```

The line 'lang="{{config('app.locale')}}"' is highlighted with a red underline, indicating it is being explained or demonstrated.

# Référencer les routes

Nous avons présentement 3 routes dans l'application :

- `/` associé à la page `welcome.blade.php`
- `/dashboard` associé à la page `dashboard.blade.php`
- `/contact` associé à la page `contact.blade.php`

Maintenant il faut ajouter ces 3 liens sur le menu respectivement comme indiqué sur la capture suivante:

```
<ul class="navbar-nav mr-auto mt-2 mt-lg-0">
  <li class="nav-item active">
    |   <a class="nav-link" href="/">Accueil <span class="sr-only">(current)</span></a>
  </li>
  <li class="nav-item active">
    |   <a class="nav-link" href="/dashboard">Dashboard <span class="sr-only">(current)</span></a>
  </li>
  <li class="nav-item active">
    |   <a class="nav-link" href="/contact">Contact <span class="sr-only">(current)</span></a>
  </li>
```

# TP2: Problème de changement de route

Précédemment, il a fallu ajouter le lien dans les 3 pages.

Changer la route `/` à `/home` et mettez à jour tous les liens de l'application

# Problème de changement de route

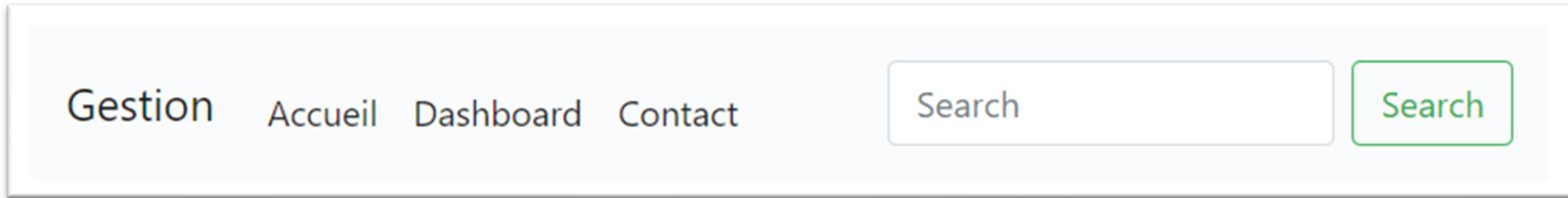
Vous vous rendez vite compte qu'avec 10 pages ou plus, le changement devient insupportable.

C'est pourquoi vous pouvez donner un nom à votre route et utiliser ce nom dans vos href avec la méthode **route(nom\_route)**. Cela vous permettra de changer de lien mais le nom sera en mesure de récupérer la route associée sans problème.

```
✓ Route::get('/', function () {
    return view('welcome');
})->name('home');
```

```
<li class="nav-item active">
    <a class="nav-link" href="{{route('home')}}">Accueil <span class="
```

# TP: Nommer toutes les autres routes et appliquer les dans le menu



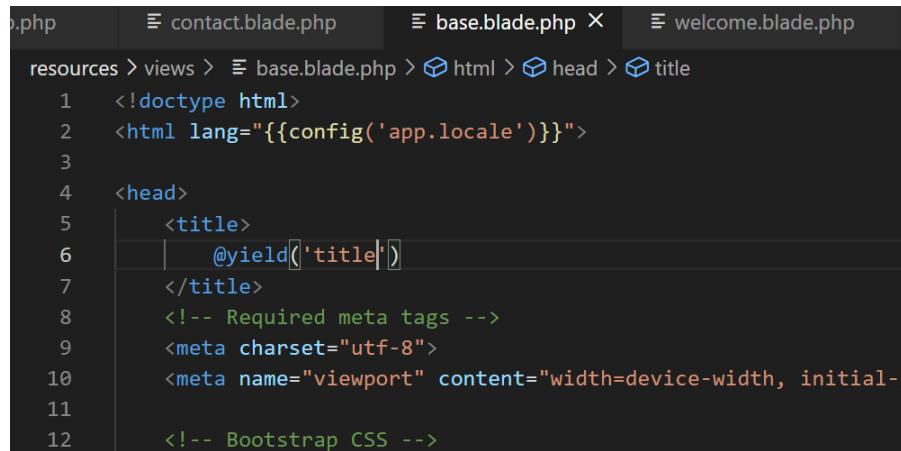
- Ne trouvez-vous pas difficile de devoir mettre le menu sur toutes les pages ?
- A chaque modification, allez-vous parcourir toutes les pages et changer le menu ?

# Templating et réduction de code

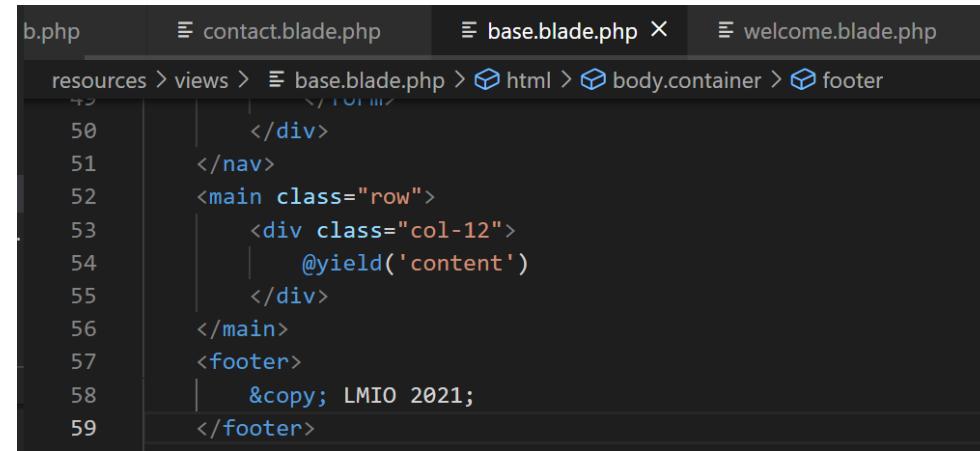
- La réponse à la dernière question est NON !
- Laravel, grâce à son moteur de template blade, permet de définir les parties d'un layout qui ne changent pas, cela permet de gérer plus facilement les mises à jour et réduire le code écrit.
- Pour ce faire, il faut créer dans *resources/view* un nouveau fichier blade qu'on va appeler *base.blade.php*
- On va coller le contenu de la page *contact.blade.php* et identifier les partie de la page qui change sur chaque page

# Définition du layout principal

- Pour chaque partie de la page susceptible de changer sur une autre page, on le mettra dans une section avec la fonction  
`@yield('nom_de_la_section')`



```
b.php contact.blade.php base.blade.php welcome.blade.php
resources > views > base.blade.php > html > head > title
1  <!doctype html>
2  <html lang="{{config('app.locale')}}">
3
4  <head>
5      <title>
6          @yield('title')
7      </title>
8      
9      <meta charset="utf-8">
10     <meta name="viewport" content="width=device-width, initial-s
11
12     <!-- Bootstrap CSS -->
```



```
b.php contact.blade.php base.blade.php welcome.blade.php
resources > views > base.blade.php > html > body.container > footer
+-->
50
51
52
53
54
55
56
57
58
59
```

```
</form>
</div>
</nav>
<main class="row">
    <div class="col-12">
        @yield('content')
    </div>
</main>
<footer>
    © LMIO 2021;
</footer>
```

- Nous avons remarqué que les titres des pages changent de même que le contenu central que nous avons nommé « **title** » et « **content** »

# Etendre le layout principal par les autres page

Sur les autres pages, il suffit d'étendre le layout principal avec la fonction **@extends('nom\_page\_layout')**, et de définir des section pour chaque partie du code qui change.

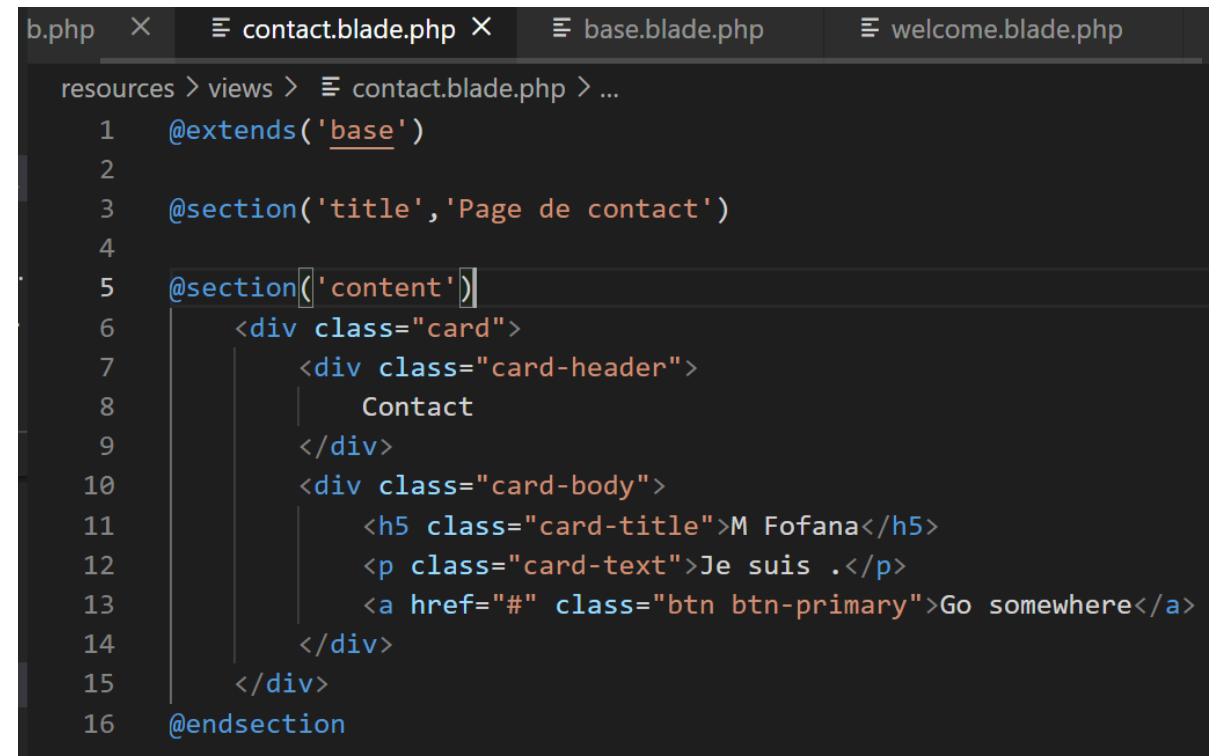
Exemple de la page contact après transformation :

La page contact est devenue plus courte et le contenu facile à lire.

La section **title** ne comporte qu'une chaîne de caractère, c'est pourquoi nous avons juste passé en paramètre le second argument qui est le titre et il n'y a pas de balise de fermeture de la section contrairement à la section **content** qui a du contenu complexe.

Après avoir rafraîchi la page, on voit que le menu est toujours présent et le design aussi.

NB: Les sections avec un second paramètre ne peuvent être fermé avec la balise **@endsection**



```
b.php × contact.blade.php × base.blade.php welcome.blade.php
resources > views > contact.blade.php > ...
1  @extends('base')
2
3  @section('title', 'Page de contact')
4
5  @section('content')
6      <div class="card">
7          <div class="card-header">
8              Contact
9          </div>
10         <div class="card-body">
11             <h5 class="card-title">M Fofana</h5>
12             <p class="card-text">Je suis .</p>
13             <a href="#" class="btn btn-primary">Go somewhere</a>
14         </div>
15     </div>
16 @endsection
```

# TP

- Appliquez cette extension du layout principal sur toutes les pages de l'application

# Les commandes Laravel

- Laravel propose une panoplie de commande qui permet d'automatiser certaines opération de développement.
- Pour exécuter une commande Laravel, il faut exécuter dans la racine du projet une commande commençant par ***php artisan***
- Exemple:
  - ***php artisan route:list*** liste toutes les routes du projet
  - ***php artisan -help*** permet d'afficher de l'aide
  - ***php artisan list*** permet d'afficher la liste des commandes disponibles
  - ***php artisan serve*** pour démarrer le serveur de développement
  - etc.

On en verra plus tout au long du cours

# Les bases de données

Presque toutes les applications Web modernes interagissent avec une base de données.

Laravel rend l'interaction avec les bases de données extrêmement simple sur une variété de bases de données prises en charge à l'aide de SQL brut, d'un générateur de requêtes fluide et de l'ORM Eloquent.

Actuellement, Laravel fournit un support de première partie pour quatre bases de données:

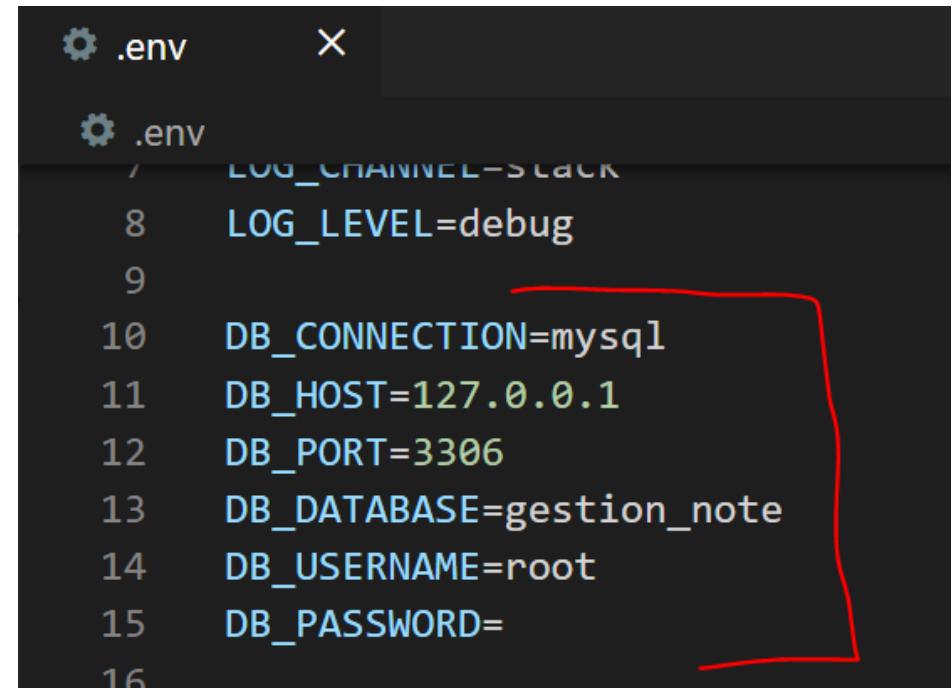
- MySQL 5.7+
- PostgreSQL 9.6+
- SQLite 3.8.8+
- SQL Server 2017+

# Cas pratique : SQLite

Dans notre cas, nous allons utiliser SQLite pour avoir une base de données portable.

Mais tout ce que nous allons faire côté programmation est valable pour tous les SGBD.

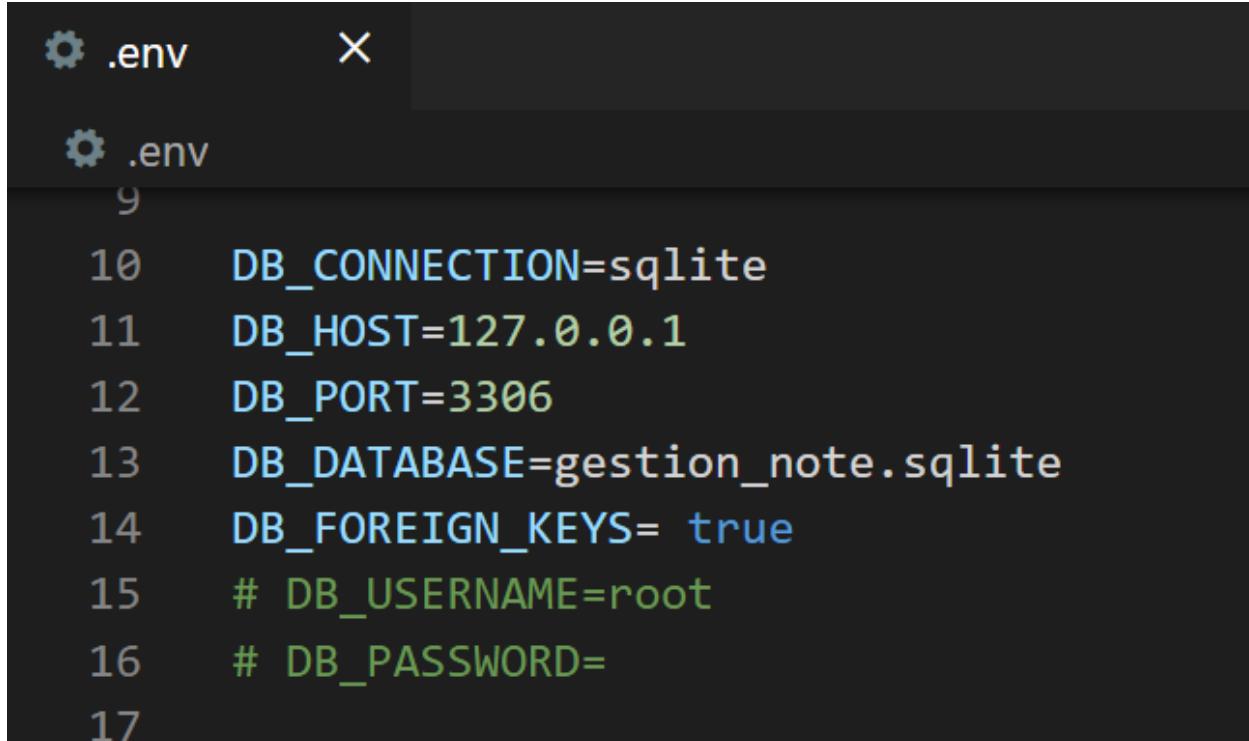
Nous allons mettre à jour ces configuration pour prendre en compte notre base de données SQLite



```
.env
LOG_CHANNEL=slack
LOG_LEVEL=debug
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=gestion_note
DB_USERNAME=root
DB_PASSWORD=
```

# Configuration de la bd sqlite

- Pour activer les contraintes de clé étrangère pour les connexions *SQLite*, vous devez définir la variable d'environnement *DB\_FOREIGN\_KEYS* sur *true* et créer le fichier *gestion\_note.sqlite* dans le répertoire *database*.
- Préciser le chemin absolu pour le fichier *SQLite* selon l'emplacement.

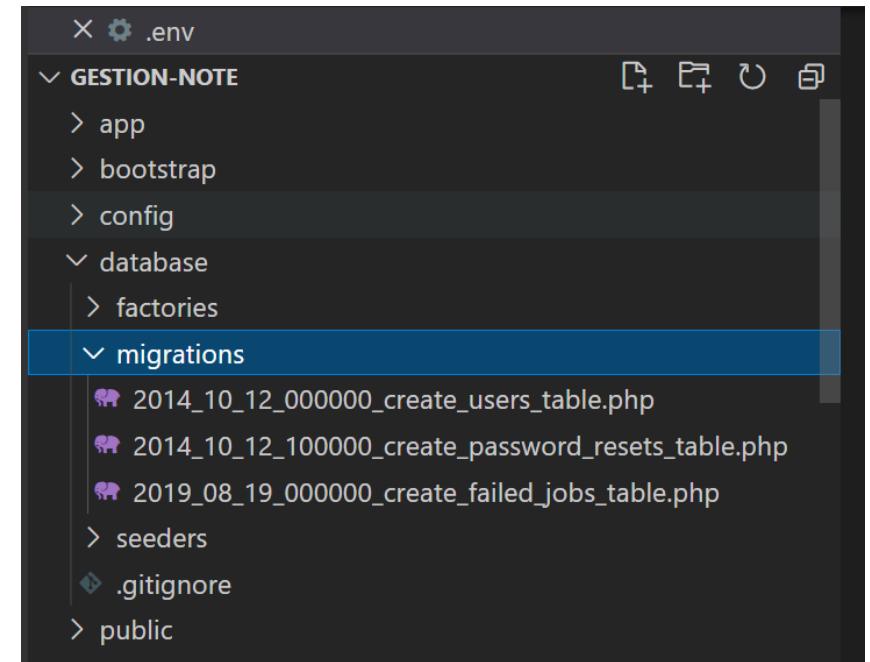


```
  .env      X
  .
  .
  9
  10 DB_CONNECTION=sqlite
  11 DB_HOST=127.0.0.1
  12 DB_PORT=3306
  13 DB_DATABASE=gestion_note.sqlite
  14 DB_FOREIGN_KEYS= true
  15 # DB_USERNAME=root
  16 # DB_PASSWORD=
  17
```

# Les migrations

Les migrations sont comme le contrôle de version de votre base de données, permettant à votre équipe de définir et de partager la définition du schéma de base de données de l'application. Si vous avez déjà dû dire à un coéquipier d'ajouter manuellement une colonne à son schéma de base de données local après avoir extrait vos modifications du contrôle de code source, vous avez été confronté au problème que les migrations de base de données résolvent.

La façade de schéma de Laravel fournit un support indépendant de la base de données pour la création et la manipulation de tables dans tous les systèmes de base de données pris en charge par Laravel. En règle générale, les migrations utiliseront cette façade pour créer et modifier des tables et des colonnes de base de données.



# Générer des migrations

Vous pouvez utiliser la commande `make:migration` de Artisan pour générer une migration de base de données. La nouvelle migration sera placée dans votre répertoire base de database/migrations/. Chaque nom de fichier de migration contient un horodatage qui permet à Laravel de déterminer l'ordre des migrations.

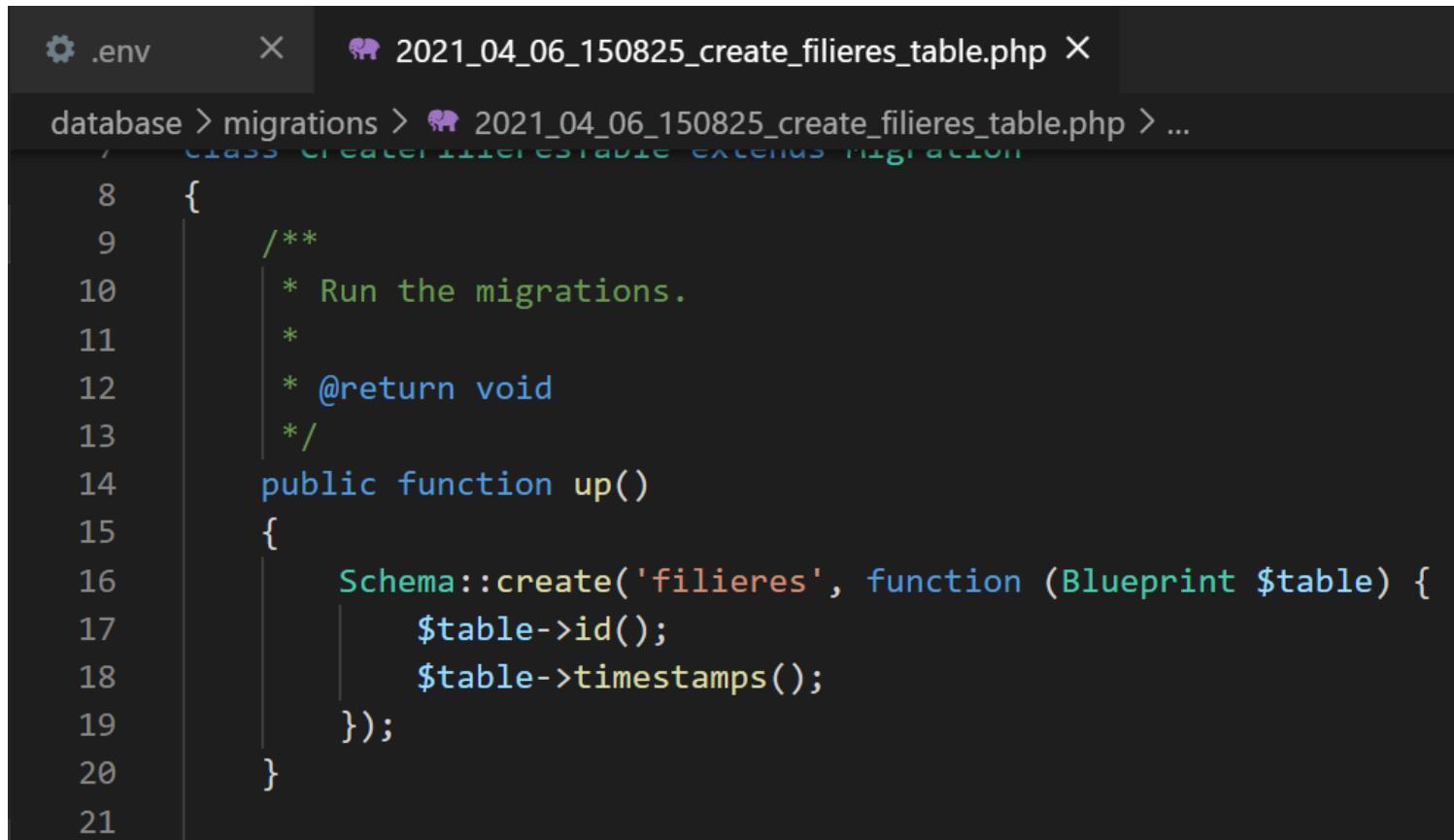
➤ `php artisan make:migration create_nom_table_avec_s_table --create=nom_table_avec_s`

Laravel utilisera le nom de la migration pour tenter de deviner le nom de la table et si la migration créera ou non une nouvelle table. Si Laravel est capable de déterminer le nom de la table à partir du nom de la migration, Laravel pré-remplira le fichier de migration généré avec la table spécifiée. Sinon, vous pouvez simplement spécifier manuellement la table dans le fichier de migration.

# Créons la migration pour la table filière

➤ ***php artisan make:migration create\_filières\_table --create=filières***

Ouvrir le fichier de migration dans le répertoire ***database/migrations***

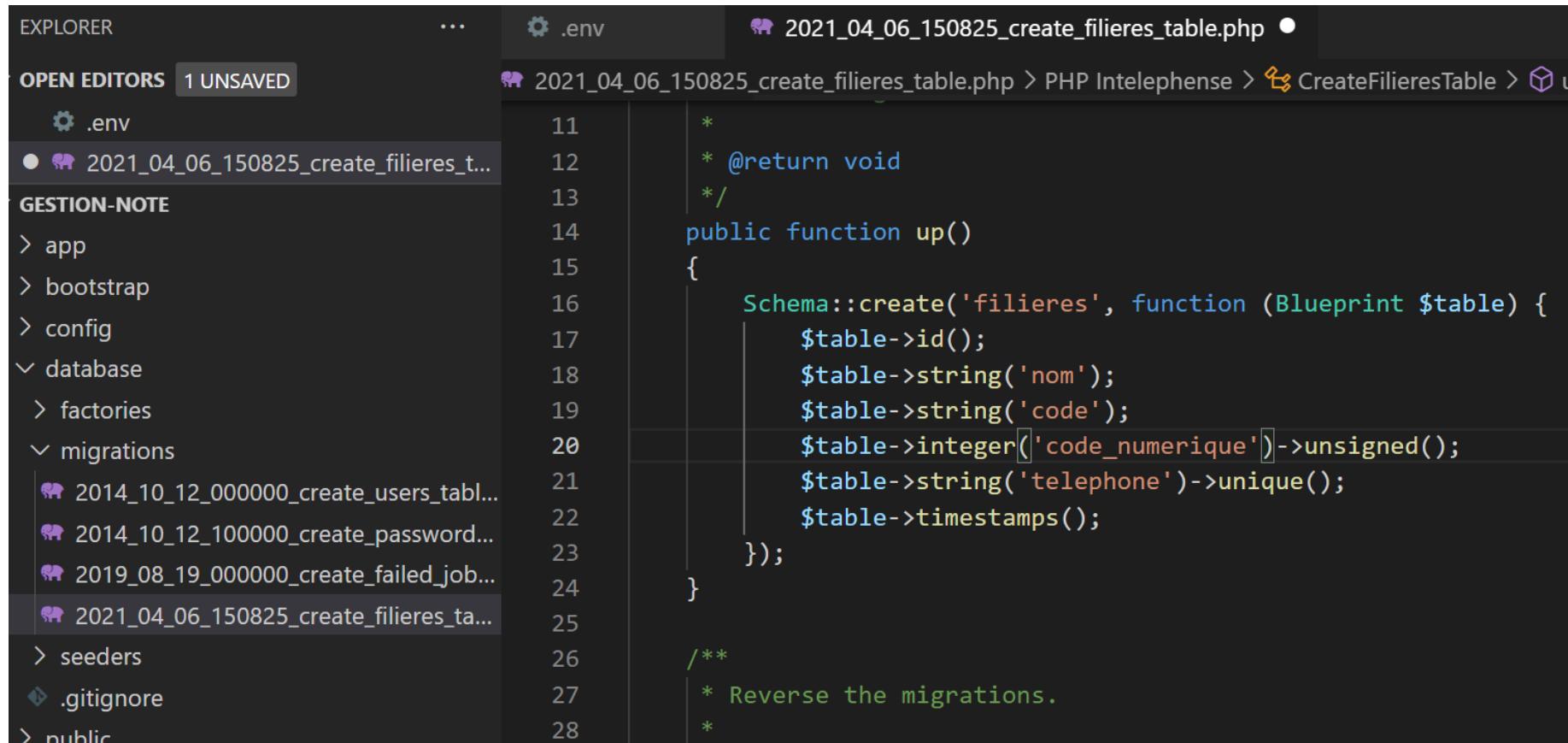


The screenshot shows a code editor with a dark theme. The file is named `2021_04_06_150825_create_filières_table.php`. The code defines a migration class with an `up()` method that creates a table named `filières` using the `Schema::create` method. The table has an `id` column and timestamp columns.

```
8  {
9      /**
10     * Run the migrations.
11     *
12     * @return void
13     */
14    public function up()
15    {
16        Schema::create('filières', function (Blueprint $table) {
17            $table->id();
18            $table->timestamps();
19        });
20    }
21}
```

# Ajoutons d'autre champs à filière

On va ajouter tous les autres champs de la table grâce aux méthodes prédéfinies de **Blueprint \$table**.



The screenshot shows a code editor with a dark theme. On the left, the file tree (EXPLORER) shows a folder structure under 'GESTION-NOTE' including 'app', 'bootstrap', 'config', 'database' (with 'factories' and 'migrations' subfolders), 'seeders', '.gitignore', and 'public'. In the 'migrations' folder, several files are listed: '2014\_10\_12\_000000\_create\_users\_table.php', '2014\_10\_12\_100000\_create\_password...', '2019\_08\_19\_000000\_create\_failed\_job...', and '2021\_04\_06\_150825\_create\_filières\_ta...'. The file '2021\_04\_06\_150825\_create\_filières\_table.php' is open in the editor. The code defines a migration class:

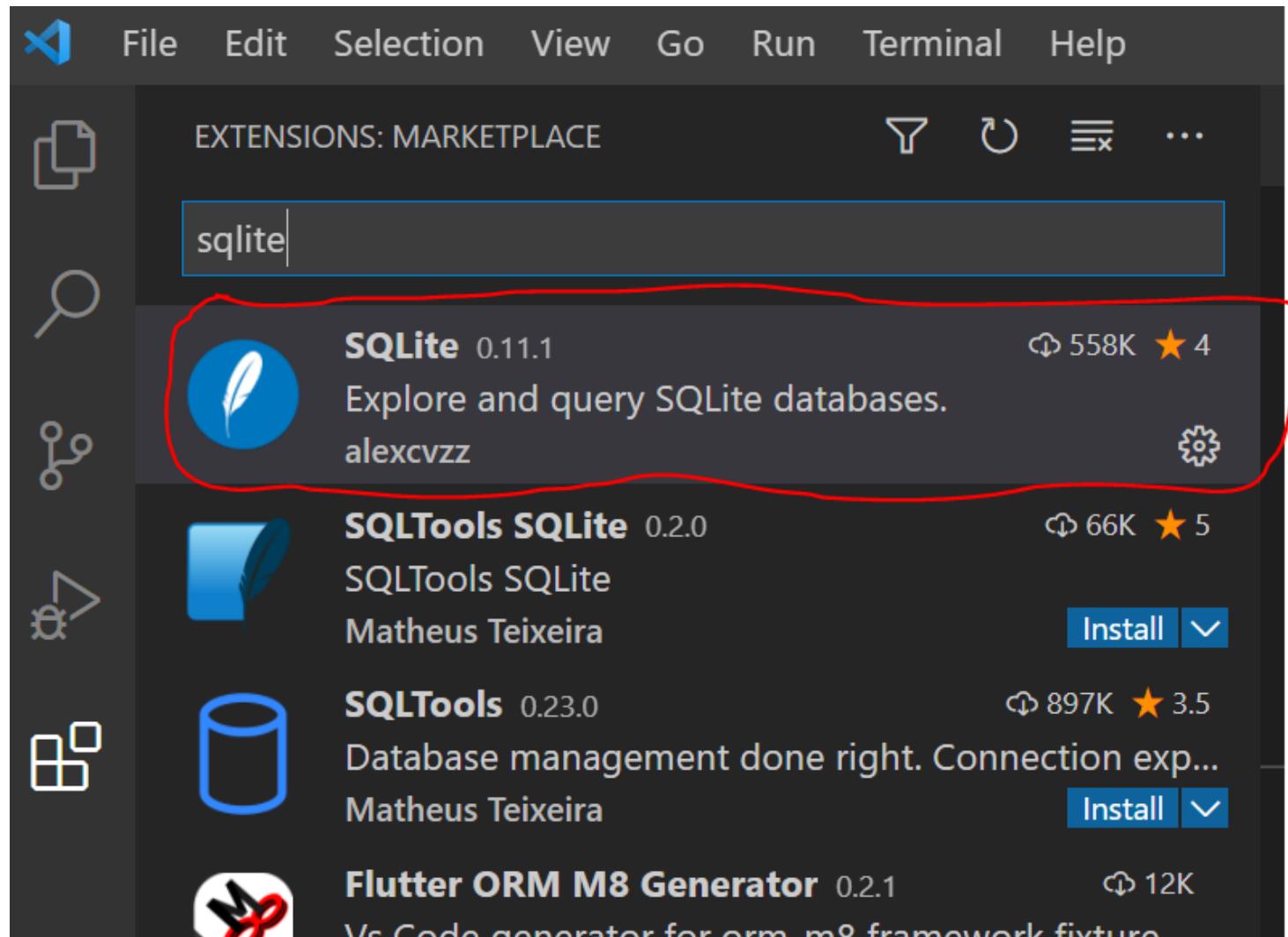
```
11 *  
12 * @return void  
13 */  
14 public function up()  
15 {  
16     Schema::create('filières', function (Blueprint $table) {  
17         $table->id();  
18         $table->string('nom');  
19         $table->string('code');  
20         $table->integer(['code_numerique'])->unsigned();  
21         $table->string('telephone')->unique();  
22         $table->timestamps();  
23     });  
24 }  
25  
26 /**  
27 * Reverse the migrations.  
28 */
```

# Appliquons les modifications dans la base de données

- *php artisan migrate* pour une migration directe
- *php artisan migrate --pretend* pour afficher la requête qui va être exécutée dans la base de donnée
- *php artisan migrate --help* pour voir toutes les options

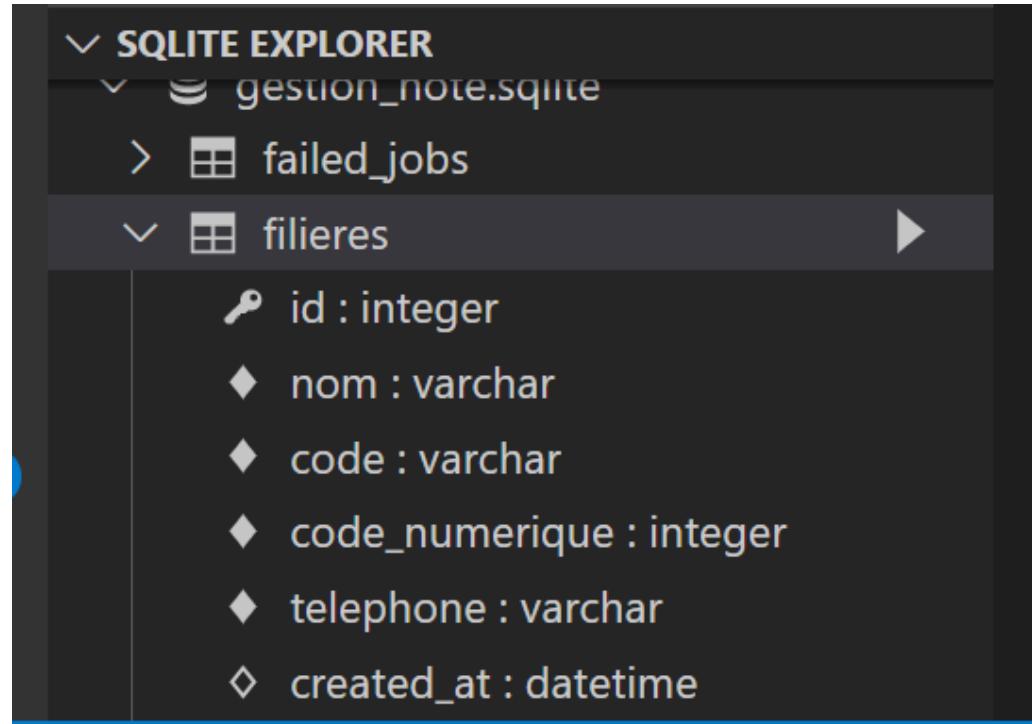
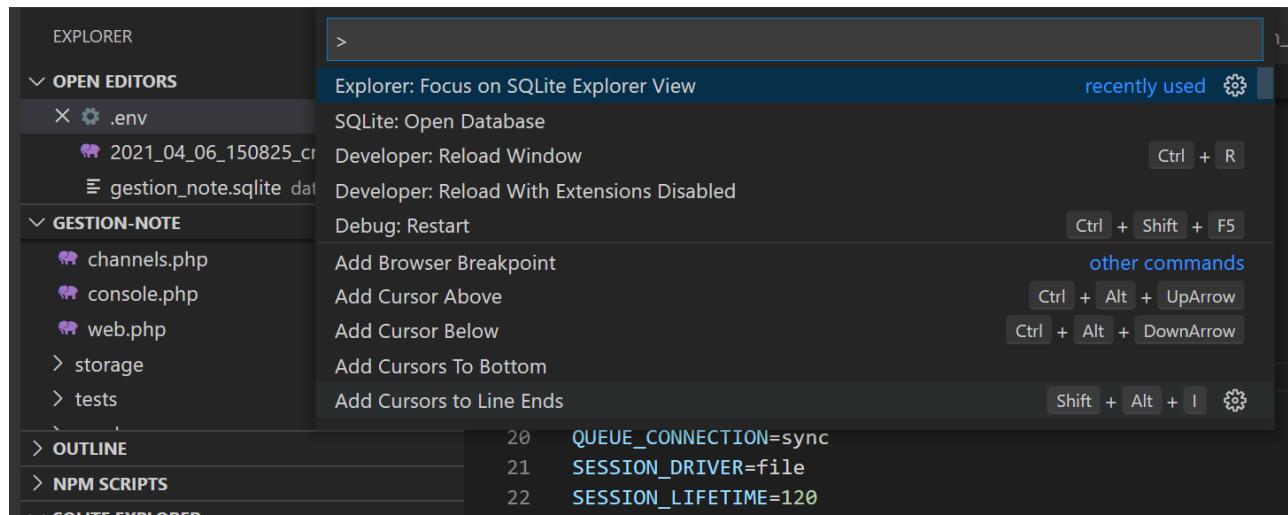
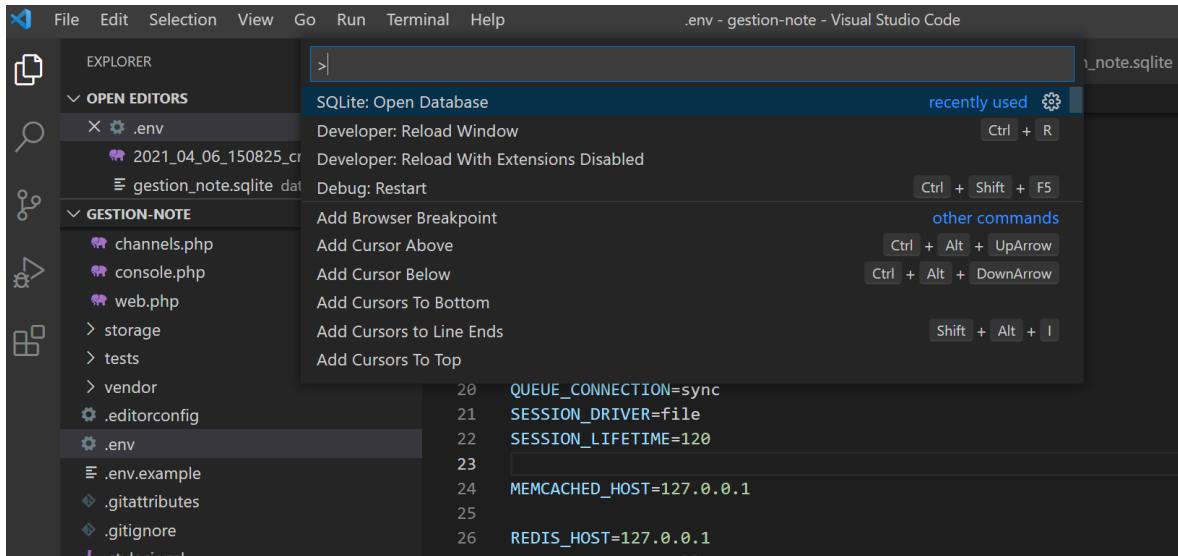
```
C:\wamp64\www\laravel\gestion-note>php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_000000_create_users_table (216.16ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table (222.13ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated: 2019_08_19_000000_create_failed_jobs_table (242.02ms)
Migrating: 2021_04_06_150825_create_filières_table
Migrated: 2021_04_06_150825_create_filières_table (254.04ms)
```

# Installer mysqli extension pour vscode



# Ouvrir la bd depuis vscode

- Dans VSCode, taper la touche Crtl+shift+p



# BD SQLite créée

- Notre base de données est bien créée et l'ensemble des tables créées dont d'autres par Laravel même pour faciliter la gestion des utilisateurs.

# Eloquent : l'ORM utilisé par Laravel

- Laravel inclut Eloquent, un mappeur objet-relationnel (ORM) qui facilite l'interaction avec votre base de données.
- Lors de l'utilisation d'Eloquent, chaque table de base de données a un «modèle» correspondant qui est utilisé pour interagir avec cette table. En plus de récupérer des enregistrements de la table de base de données, les modèles Eloquent vous permettent d'insérer, de mettre à jour et de supprimer également des enregistrements de la table.
- Pour commencer, créons un modèle Eloquent. Les modèles résident généralement dans le répertoire `app\Models` et étendent la classe `Illuminate\Database\Eloquent\Model`. Vous pouvez utiliser la commande `make:model`

# Les contrôleurs de ressources : CRUD

- Au lieu de définir toute votre logique de traitement des demandes comme des fermetures dans vos fichiers de routage, vous souhaiterez peut-être organiser ce comportement en utilisant des classes de «contrôleur». Les contrôleurs peuvent regrouper la logique de traitement des demandes associées en 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*.
- La commande de génération d'un contrôleur :
  - *php artisan make:controller nom\_controller* exemple *UserController*

# Générer le modèle Filiere et son controller

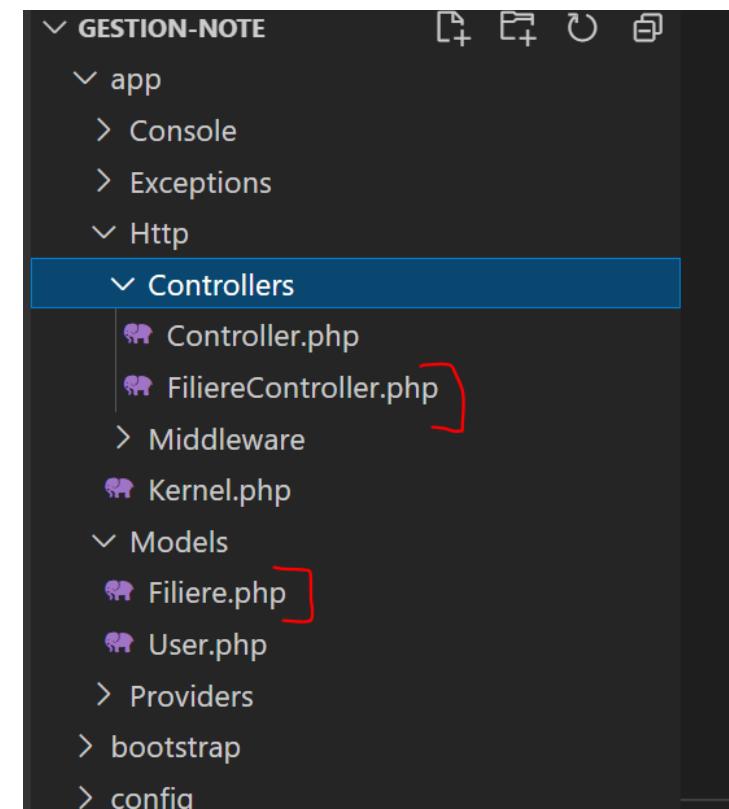
Il est possible de combiner les deux commandes précédentes en une seule en faisant ceci:

➤ ***php artisan make:model Filiere --controller***

Ajouter l'option **--resource** pour créer les méthodes du CRUD.

➤ ***php artisan make:model Filiere --controller --resource***

```
C:\wamp64\www\laravel\gestion-note>php artisan make:model Filiere --controller
Model created successfully.
Controller created successfully.
```



# Créer une route pour la ressource

- Ajouter `Route::resource('filiere', FiliereController::class);` dans `routes/web.php`
- Cette déclaration de route unique crée plusieurs routes pour gérer diverses actions sur la ressource. Le contrôleur généré aura déjà des méthodes affectées pour chacune de ces actions.
- N'oubliez pas que vous pouvez toujours obtenir un aperçu rapide des itinéraires de votre application en exécutant la commande `php artisan route:list`

	GET HEAD	filiere	filiere.index	App\Http\Controllers\FiliereController@index	web
	POST	filiere	filiere.store	App\Http\Controllers\FiliereController@store	web
	GET HEAD	filiere/create	filiere.create	App\Http\Controllers\FiliereController@create	web
	GET HEAD	filiere/{filiere}	filiere.show	App\Http\Controllers\FiliereController@show	web
	PUT PATCH	filiere/{filiere}	filiere.update	App\Http\Controllers\FiliereController@update	web
	DELETE	filiere/{filiere}	filiere.destroy	App\Http\Controllers\FiliereController@destroy	web
	GET HEAD	filiere/{filiere}/edit	filiere.edit	App\Http\Controllers\FiliereController@edit	web

# Exécutons un petit code dans la méthode du Dashboard pour ajouter une filière

```
Route::get('/dashboard', function () {
    Filiere::create(
        ['nom'=>'Licence Management Informatisé des Organisations',
        'code'=>'LMIO',
        'code_numerique'=>'01',
        'telephone'=>'339511400']);
    return view('dashboard');
})->name('dashboard');
```

- Exéutez la route du **dashboard**, une exception sera lancée. Parce que avec eloquent, les attributs des champs sont protégés à l'écriture.
- Pour palier à ce problème, on doit ajouter un attribut **protected \$fillable** dans notre modèle Filière et mettre le tableau des champs qui seront autorisés pour écriture.

```
protected $fillable = [
    'nom',
    'code',
    'code_numerique',
    'telephone'
];
```

# Ouvrez l'explorateur SQLite

Notre enregistrement a bien été créé dans la table filière

The screenshot shows an SQLite database interface with a single row of data in a table named 'filiere'. The columns are 'nom', 'code', 'code\_numerique', and 'telephone'. The data is as follows:

nom	code	code_numerique	telephone
Licence Management Informatisé des Organisations	LMIO	1	339511400

- Changez les paramètres et créez plusieurs filières

```
Route::get('/dashboard', function () {
    Filiere::create(
        ['nom'=>'Licence Management Informatisé des Organisations',
        'code'=>'LMIO',
        'code_numerique'=>'01',
        'telephone'=>'339511400']);
    return view('dashboard');
})->name('dashboard');
```

```
[  
    'nom' => 'Langue étrangère appliquée',  
    'code' => 'LAC',  
    'code_numerique' => '02',  
    'telephone' => '339511402'  
]
```

```
[  
    'nom' => 'Master Management Touristique & Hôtellerie',  
    'code' => 'MMTH',  
    'code_numerique' => '03',  
    'telephone' => '339511403'  
]
```

```
[  
    'nom' => 'Licence Sciences Économiques & Gestion',  
    'code' => 'LSEG',  
    'code_numerique' => '04',  
    'telephone' => '339511404'  
]
```

# Requêtes avec Eloquent

- `Filiere::all();` retourne la liste de toutes les filières de la BD
- `Filiere::where('code','LMIO')->orderBy('nom')->get();` récupere la liste des filières dont le code='LMIO' et le résultat ordonné par nom
- `Filiere ::where('code_numerique', 02)->first();` récupère le premier élément du résultat
- `Filiere::get(1);` récupere l'element filiere don't la valeur de la clé primaire est égale à 1
- `Filiere::where('telephone','like','%33951%')->get();` retourne toutes les filières dont le numéro de téléphone contient "33951"

# Implémenter la fonction index() de Filière contrôleur pour afficher la liste des filières

- La méthode **dd()** appelée « die dump » permet d'afficher des valeurs brutes pour étudier leur contenu
- En exécutant le chemin **/filiere**, on voit bien que la liste des filières est récupérée

The screenshot shows a development setup with two main components:

- Code Editor:** On the left, the file `FiliereController.php` is open in a code editor. The code defines a controller with an `index()` method that retrieves all filières from the database and displays them using the `dd()` function.
- Browser:** On the right, a browser window is displaying the URL `127.0.0.1:8000/filiere`. The page content is the raw PHP array output from the `dd()` function, showing four items, each being an instance of the `Filiere` model.

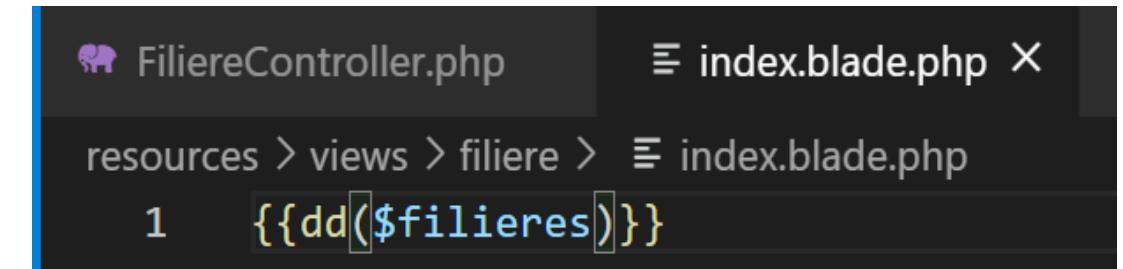
```
8 class FiliereController extends Controller
9 {
10     /**
11      * Display a listing of the resource.
12      *
13      * @return \Illuminate\Http\Response
14      */
15     public function index()
16     {
17         $filières = Filiere::all();
18         dd($filières);
19     }
20 }
```

```
Illuminate\Database\Eloquent\Collection {#1217 ▼
  #items: array:4 [▼
    0 => App\Mode...\\Filiere {#1218 ▶}
    1 => App\Mode...\\Filiere {#1219 ▶}
    2 => App\Mode...\\Filiere {#1220 ▶}
    3 => App\Mode...\\Filiere {#1221 ▶}
  ]
}
```

# Implémenter la méthode index de FiliereController

- Créez un répertoire *filiere* dans *resources/views* et ajouter un fichier *index.blade.php*
- Mettez à jour la fonction index en retournant une *view()* associée à la liste des filières
- Dans la page index, essayez de faire un « die dump » de la variable filière

```
| public function index()
{
    $filières = Filiere::all();
    return view('filiere.index',['filières'=>$filières]);
}
```



- Avec blade :

- *Hello, {{\$name}}* : Affiche le nom
- *@foreach (\$users as \$user)*  
*<p>This is user {{ \$user->id }}</p>*  
*@endforeach*

*permet de parcourir une liste et afficher des attributs de l'objet*

# TP : (résultat attendu sur la page suivante)

- Etendre la page *filiere/index.blade.php* avec le fichier layout
- Personnaliser la page avec les tableaux Bootstrap 4 et afficher tous les attributs de filière
- Ajouter au tableau une colonne actions et ajouter différents boutons (voir, supprimer) dans cette colonne pour chaque colonne
- Ajouter un bouton plus en haut du tableau
- Ajouter un nouveau menu filière avec le lien de index filière

PS: préférez les icônes aux textes dans les boutons

# Résultat attendu du TP

Gestion Accueil Dashboard Filière Contact  Search

## Liste des filières

<b>Id</b>	<b>Nom</b>	<b>Code</b>	<b>Code numérique</b>	<b>Téléphone</b>	<b>Actions</b>
1	Licence Management Informatisé des Organisations	LMIO	1	339511400	 
2	Langue étrangère appliquée	LAC	2	339511402	 
3	Master Management Touristique & Hôtellerie	MMTH	3	339511403	 
4	Licence Sciences Économiques & Gestion	LSEG	4	339511404	 

LMIO © 2021

# Implémenter le formulaire d'ajout d'une filière

Dans la méthode `create`, nous allons rediriger l'utilisateur vers la page du formulaire.

Pour cela, nous allons créer une page `views/filiere/new.html` et nous allons le lier à la méthode

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

# Implémenter le formulaire d'ajout d'une filière

Dans l'attribut action de la balise `form`, lui donner la route '`'filiere.store'`' qui référence la méthode `store` de `FiliereController` et l'attribut `name` de chaque balise input doit correspondre exactement au nom du champ pour que Laravel puisse le faire correspondre à l'objet.

```
@section('content')


#### Formulaire création Filière


<form action="{{route('filiere.store')}}" method="post">
    <div class="form-group">
        <label for="nom">Nom</label>
        <input type="text" name="nom" id="nom" class="form-control" placeholder="">
    </div>
    <div class="form-group">
        <label for="code">Code (Sigle)</label>
        <input type="text" name="code" id="code" class="form-control" placeholder="">
    </div>


```

The screenshot shows a web interface for creating a new filière. At the top, there is a navigation bar with links: Gestion, Accueil, Dashboard, Filière, Contact, and a search bar. Below the navigation, the title 'Formulaire création Filière' is displayed. The form contains four input fields: 'Nom' (Name), 'Code (Sigle)' (Code (Sigle)), 'Code numérique' (Numerical Code), and 'Téléphone' (Phone). A blue 'Enregistrer' (Register) button is located at the bottom right of the form area.

# Implémenter le formulaire d'ajout d'une filière

Récupérer et afficher le contenu du formulaire depuis notre méthode store

```
public function store(Request $request)
{
    dd($request->all());
}
```

Lorsque vous validez le formulaire, une exception est lancée.

Cela s'explique par le fait que tout formulaire Laravel doit comporter un token de sécurité appelé « **csrf** » pour s'assurer qu'il ne s'agit pas d'une attaque externe et malveillante.

# Implémenter le formulaire d'ajout d'une filière

Ajoutons le token csrf avec `@csrf` à l'intérieur de la balise `form`

```
<form action="{{route('filiere.store')}}" method="post">
    @csrf
    <div class="form-group">
        <label for="nom">Nom</label>
        <input type="text" name="nom" id="nom" class="form-control" value="Licence Informatique" required="required" />
    </div>
    <div class="form-group">
        <label for="code">Code (Sigle)</label>
        <input type="text" name="code" id="code" class="form-control" value="LI" required="required" />
    </div>
```

Le contenu du formulaire est bien récupéré avec `$request->all();`

```
array:5 [▼
  "_token" => "ZAQo10eKV2eBy6ppQ9MwOEohAIB9GF8FJn0vSqhh"
  "nom" => "Licence Informatique"
  "code" => "LI"
  "code_numerique" => "05"
  "telephone" => "339511844"
]
```

Lorsque vous validez le formulaire, l'exception disparaît

Gestion Accueil Dashboard Filière Contact Search

Formulaire création Filière

Nom

Code (Sigle)

Code numérique

Téléphone

Enregistrer

# Implémenter le formulaire d'ajout d'une filière

Maintenant, nous allons personnaliser la méthode store afin qu'il stocke notre filière et nous redirige vers la liste des filières.

```
public function store(Request $request)
{
    Filiere::create($request->all());
    return redirect()->route('filiere.index');
}
```

Le formulaire marche bien maintenant, les filières sont créées.

Mais qu'en est il des contraintes des champs non null, des champs uniques, etc. ?

Si on ne renseigne pas des champs obligatoires, l'ajout échoue et une exception est lancée. Laravel nous donne un moyen simple de gérer ces incidences.

# Gestion des erreurs avec le formulaire

- Dans la méthode store du **controller**, on ajoute la validation du formulaire avec les contraintes pour s'assurer de l'intégrité des données.
- Sur la vue **new.blade.php**, on parcours et affiche la liste des erreurs rencontrées

```
@section('content')
@foreach ($errors->all() as $message)
    <div class="alert alert-danger" role="alert">
        <strong>{{$message}}</strong>
    </div>
@endforeach
<div class="card">
    <div class="card-body">
        <h4 class="card-title">Formulaire création Filière</h4>
        <form action="{{route('filiere.store')}}" method="post">
            @csrf
            <div class="form-group">
                <label for="nom">Nom</label>
                <input type="text" name="nom" id="nom" class="form-control">
            </div>
        </form>
    </div>
</div>
```

```
public function store(Request $request)
{
    $this->validate($request, [
        'nom'=>'required',
        'code'=>'required',
        'code_filiere'=>'required|unique:filières,code_filière|unsigned',
        'telephone'=>'unique:filières,telephone'
    ]);
    Filière::create($request->all());
    return redirect()->route('filière.index');
}
```

The screenshot shows a user interface for creating a new filière. At the top, there's a navigation bar with links for 'Gestion', 'Accueil', 'Dashboard', 'Filière', and 'Contact', along with a search bar. Below the navigation, three error messages are displayed in red boxes: 'The nom field is required.', 'The code field is required.', and 'The code filiere field is required.'. Below these errors is a form titled 'Formulaire création Filière' with four input fields: 'Nom', 'Code (Sigle)', 'Code numérique', and 'Téléphone'. Each input field has a placeholder text corresponding to its label.

# Gestion des erreurs avec le formulaire

- On peut améliorer ces erreurs en affichant devant chaque champ l'erreur détectée.
- Grace à blade, **@error('name\_input')** permettra de stocker les erreurs d'un champ
- **{{\$errors->first('name\_input')}}**  affiche le premier message d'erreur du champ

```
@section('content')


<div class="card-body">
        <h4 class="card-title">Formulaire création Filière</h4>
        <form action="{{route('filiere.store')}}" method="post">
            @csrf
            <div class="form-group">
                <label for="nom">Nom</label>
                <input type="text" name="nom" id="nom" class="form-control" placeholder="" a
                    @error('nom')
                    <small id="nomHelp" class="form-text text-danger h6">{{$errors->first('nom')}}
                    @enderror
                </div>


```

Formulaire création Filière

Nom

The nom field is required.

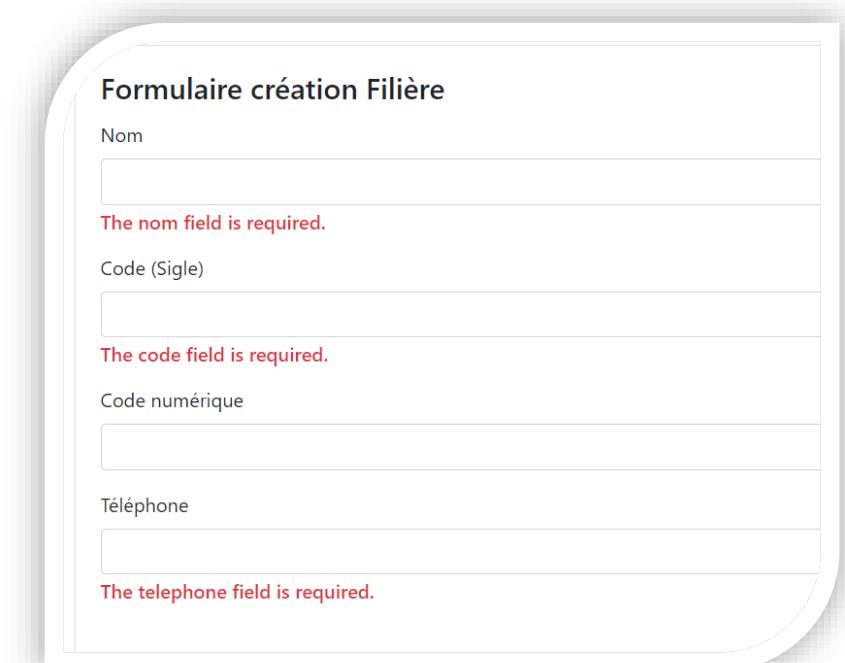
Code (Sigle)

The code field is required.

Code numérique

Téléphone

The telephone field is required.



# Gestion des erreurs avec le formulaire

- On peut aller plus loin en améliorant l'expérience.
- Bootstrap dispose d'une classe css **is-invalid** qui entoure l'input en rouge avec des indications d'erreur
- Nous utiliserons le test conditionnel @error('input\_name') pour ajouter cette classe à l'input

```
<div class="form-group">
    <label for="code">Code (Sigle)</label>
    <input type="text" name="code" id="code"
        class="form-control @error('code') is-invalid @enderror" placeholder=""
        aria-describedby="codeHelp">
    @error('code')
        <small id="codeHelp" class="form-text text-danger h6">$errors->first('code'
            @enderror
    </div>
```

L'autre problème est que nous constatons qu'après erreur, le formulaire se vide et l'utilisateur est obligé de ressaisir tout le formulaire, comment éviter cela ?

The screenshot shows a "Formulaire création Filière" (Form creation) page. It contains four input fields: "Nom", "Code (Sigle)", "Code numérique", and "Téléphone". Each field has a red border and a red 'X' icon at its right end, indicating it is required. Below each field, an error message is displayed in red: "The nom field is required.", "The code field is required.", "The code numerique field is required.", and "The telephone field is required.". A blue "Enregistrer Ø" button is located at the bottom right of the form area.

# Gestion des erreurs avec le formulaire

- Pour éviter au formulaire de se vider après une erreur de validation, il y'a une méthode **old('input\_name')** qui permet de stocker l'ancienne valeur d'un champ
- On le donne comme valeur à l'input

```
<div class="form-group">
    <label for="code_numerique">Code numérique</label>
    <input type="text" name="code_numerique" id="code_numerique"
        value="{{old('code_numerique')}}"
        class="form-control @error('code_numerique') is-invalid @enderror"
        placeholder=""
        aria-describedby="code_numeriqueHelp">
    @error('code_numerique')
        <small id="code_numeriqueHelp"
            class="form-text text-danger h6">$errors->first('code_numerique')</small>
    @enderror
</div>
```

Formulaire création Filière

Nom  
Licence Informatique

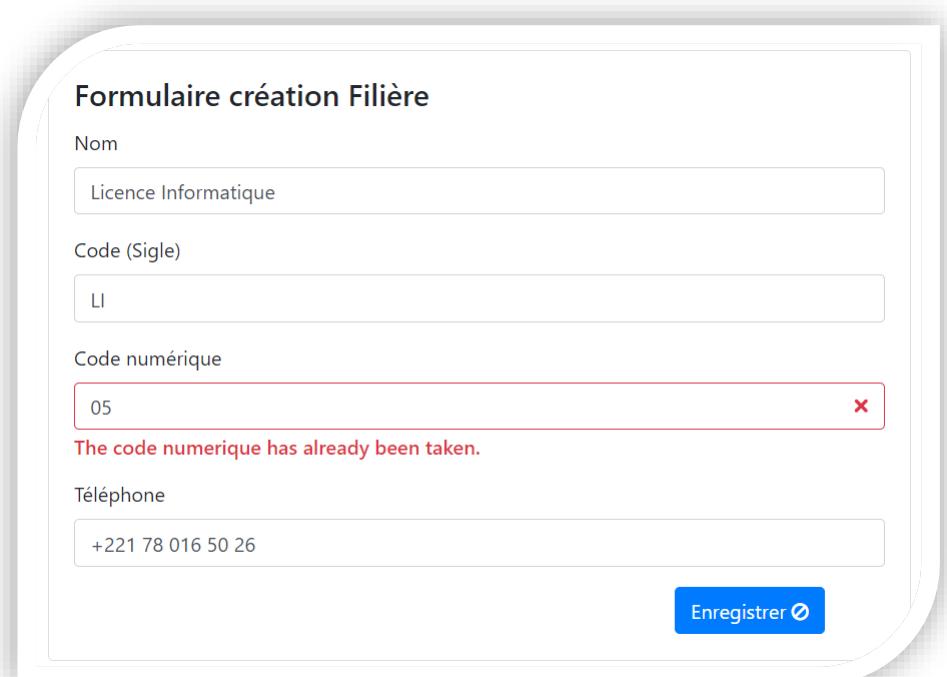
Code (Sigle)  
LI

Code numérique  
05 ×

The code numerique has already been taken.

Téléphone  
+221 78 016 50 26

**Enregistrer** Ø



# Implémentation de la suppression

- La méthode `destroy()` du controller Filière permet de supprimer une filière de la base de données.
- En exécutant la commande `php artisan route:list`, on remarque cette ligne.

```
|   | DELETE    | filiere/{filiere}      | filiere.destroy | App\Http\Controllers\FiliereController@destroy | web
```

- Donc il faut noter que les méthodes PUT, PATCH, HEAD, DELETE ne peuvent être appelé depuis le web qu'à partir d'un formulaire et il faut se rappeler que dans Laravel, tout formulaire doit comporter `@csrf` pour autoriser l'envoie du formulaire et vue que le `form` ne comprend pas la méthode `delete` dans Laravel, il faut le préciser grâce à `@method('nom_methode')` avec `nom_methode = DELETE` dans ce cas.

```
public function destroy(Filiere $filiere)
{
    $filiere->delete();
    return redirect()->route('filiere.index');
}
```

Après avoir ajouté ces codes, la suppression marchera correctement.

```
<td>
  <div>
    <form action="{{route('filiere.destroy',['filiere'=>$filiere])}}" method="post"
          class="inline">
      @csrf
      @method('delete')
      <button type="button" class="btn btn-primary btn-sm btn-rounded mr-1">
        <i class="fa fa-eyedropper" aria-hidden="true"></i>
      </button>
      <button type="submit" class="btn btn-danger btn-sm btn-rounded mr-1">
        <i class="fa fa-trash-o" aria-hidden="true"></i>
      </button>
    </form>
  </div>
</td>
```

# Implémenter la page d'affichage d'une filière

- Implémentez la méthode **show()** de filière Controller de sorte qu'elle retourne la page de visualisation des détails d'une filière
- Créez une nouvelle page show.blade.html qui étend le layout principal et qui affiche les détails d'une filière
- Ajouter un lien avec la route de filiere show sur la colonne nom du tableau filière

NB: Lorsqu'on retourne une vue, on peut préciser un tableau en paramètre avec la clé de la variable qui sera utilisée sur blade et la valeur, celle de la variable elle-même.

Exemple: **return view('filiere.index',['filieres'=>\$filieres]);**

Cependant, on peut simplifier cette écriture en utilisant directement la fonction compact() qui va identifier la variable et lui associer une clé du même nom. Cette fonction compact() peut prendre plusieurs valeurs en paramètre ou un tableau de valeurs.

Exemple: **return view('filiere.show',compact('filiere'));**

# Implémenter la page d'affichage d'une filière

- Pour retourner à la page précédente, blade nous donne la façade URL avec la fonction `previous()` comme ça [URL::previous\(\)](#) récupère la route précédente.

Liste des filières					
Id	Nom	Code	Code numérique	Téléphone	Actions
1	Licence Management Informatisé des Organisations	LMIO	1	339511400	 
2	Langue étrangère appliquée	LAC	2	339511402	 
3	Master Management Touristique & Hôtellerie	MMTH	3	339511403	 
4	Licence Sciences Économiques & Gestion	LSEG	4	339511404	 
5	Licence Informatique	LI	5	339511844	 
7	Licence Science de l'Eau et de l'Environnement	LSEE	6	339511846	 
8	Licence Mathématique Informatique	LMI	7	339511847	 

Gestion Accueil Dashboard Filière Contact



### Filière : Licence Management Informatisé des Organisations

Id	1
Nom	Licence Management Informatisé des Organisations
Code	LMIO
Code numérique	1
Téléphone	339511400

```
public function show(Filiere $filiere)
{
    return view('filiere.show',compact('filiere'));
}
```

```
<h4 class="card-title">Filière : {{$filiere->nom}}</h4>
<table class="table table-hover table-striped">
    <tbody>
        <tr>
            <th>Id</th>
            <td scope="row">{{$filiere->id}}</td>
        </tr>
        <tr>
            <th>Nom</th>
            <td>{{$filiere->nom}}</td>
        </tr>
    </tbody>
</table>
```

# Implémenter le formulaire de modification d'une filière

- Créer une page blade dans `views/filiere/edit.blade.php`, étendre le layout principal, y coller le contenu de la page `views/filiere/new.blade.php`
- Dans `FiliereController`, dans la méthode `edit`, retourner la page `edit`
- Dans `FiliereController`, dans la fonction `update()`, implémenter la logique de modification et gérer toutes les contraintes comme lors de la création et retourner l'utilisateur sur la page précédente
- Vu qu'il s'agit de modification, le formulaire doit être prérempli avec les informations de l'instance de filière qu'on est entrain de modifier, pour cela il faut remplacer la valeur de l'attribut `value` de chaque input en mettant `$filiere->nom_champ` à la place de `old('nom_champ')`. Mettre à jour le lien d'action pour la mise à jour du formulaire
- Identifier le nom de la route qui est associée à la fonction `edit()` et ajoute un lien au bouton de modification sur la liste (tableau) des filière

# Implémenter le formulaire de modification d'une filière

```
public function update(Request $request, Filiere $filiere)
{
    $this->validate($request,[
        'nom'=>'required|unique:filières,nom',
        'code'=>'required|unique:filières,code',
        'code_numerique'=>'required|unique:filières,code_numerique',
        'telephone'=>'required|unique:filières,telephone'
    ]);
    $filiere->update($request->all());
    return redirect()->route('filiere.index');
}
```

```
public function edit(Filiere $filiere)
{
    //
    return view('filiere.edit',compact('filiere'));
}
```

Mis à jour du bouton de modification dans la liste des filières

```
<a href="{{route('filiere.edit',compact('filiere'))}}"
    class="btn btn-primary btn-sm btn-rounded mr-1">
    <i class="fa fa-eyedropper" aria-hidden="true"></i>
</a>
```

Formulaire Modification Filière

Nom

Code (Sigle)

Code numérique

Téléphone

Mettre à jour

# Gestion des unicités dans les formulaire de modification

Lors de l'ajout d'un enregistrement dans la base de données, il suffit de vérifier que la valeur unique n'y existe pas déjà.

Alors que lors de la mise à jour, on ne peut pas garder cette même logique. Parce que une valeur non modifiée existe déjà dans la base de données même si elle appartient à l'objet qu'on est entrain de mettre à jour.

Exemple: filiere[code=>'LMIO',codenum='1'], si je met à jour cet objet en filiere[code=>'LMIO',codenum='2'], la mise à jour va échouer, il dira qu'il y'a duplication du code LMIO. Mais cela est faux parce que c'est le même objet qui a cette valeur.

Dans ce cas, il faut exclure l'objet courant de la vérification en remplaçant le premier par le second

```
$this->validate($request,[  
    'nom'=>'required|unique:filières,nom',
```

```
$this->validate($request,[  
    'nom'=>['required',Rule::unique('filières')->ignore($filiere->id)],
```

# Gestion des unicités dans les formulaires de modification

Le code final de gestion des unicités doit ressembler à ceci, vous pouvez aussi passer en paramètre à `ignore()` l'objet `$filiere`, Laravel va extraire l'id automatiquement.

```
public function update(Request $request, Filiere $filiere)
{
    $this->validate($request, [
        'nom' => [
            'required', Rule::unique('filieres')->ignore($filiere->id)
        ],
        'code' => [
            'required', Rule::unique('filieres')->ignore($filiere->id)
        ],
        'code_numerique' => [
            'required', Rule::unique('filieres')->ignore($filiere->id)
        ],
        'telephone' => [
            'required', Rule::unique('filieres')->ignore($filiere->id)
        ]
    ]);
    $filiere->update($request->all());
    return redirect()->route('filiere.index');
}
```

# Gestion des erreurs dans le formulaire de modification

- Essayer remplir le formulaire précédent mais qu'une erreur de validation survienne : exemple, on modifie le code de la filière en mettant le code d'une autre filière déjà enregistré dans la base de données. En même temps, on modifie le numéro de téléphone en mettant un nouveau numéro.
- Une erreur de validation d'unicité se produira mais les informations que l'utilisateur a modifié dans le formulaire seront perdues. Il sera obligé de ressaisir le numéro de téléphone ou autre.
- Pour éviter cela et améliorer l'expérience utilisateur, nous allons devoir demander à Laravel de retenir les informations saisies par l'utilisateur. A la différence du ***new.blade.php***, dans le edit, nous utiliserons ***value="{{old('telephone') ?? \$filiere->telephone}}"*** sur tous les inputs.
- Ce code dit au formulaire : si l'utilisateur a déjà saisi une valeur, ramène cette valeur, sinon ramène la valeur initial de l'objet.

# Félicitations !

Nous venons de voir comment créer un CRUD complet avec Laravel tout en prenant en compte toutes les contraintes et l'amélioration de l'expérience utilisateur.

# Comment ajouter des méthodes supplémentaires au Controller CRUD ?

On a vu que pour définir toutes les routes d'une ressource, il suffit d'ajouter

```
Route::resource('/filiere', FiliereController::class);
```

Cependant, seules les méthodes par défaut du CRUD sont concernées.

Pour créer par exemple une méthode qui recherche une filière par nom ou par code, on peut appeler la fonction `search()` et pour la route, on fera ainsi :

```
Route::post('/filiere/search/', [FiliereController::class, 'search']);|  
Route::resource('/filiere', FiliereController::class);
```

# Défi – Implémenter la méthode search pour rechercher des filières dont le nom ou code contient la valeur dans l'input

Astuces :

- Utiliser la méthode `where()` de Eloquent avec l'option like.
- Pour la valeur à rechercher, l'entourer de % pour récupérer les meilleures occurrences.
- Ajouter un champ recherche au tableau des filières au niveau de `index.blade.php`, si l'utilisateur remplit le formulaire et valide, le tableau s'affiche mais seulement avec les occurrences trouvées.

# Implémenter la méthode search

```
public function search(Request $request) {
    $request->validate(['motCle'=>'required|min:3']);
    $motCle = $request->motCle;
    $filières = Filiere::where('nom','like','%'.$motCle.'%')
        ->orWhere('code','like','%'.$motCle.'%')
        ->get();
    return view('filiere.index',compact('filières'));
}
```

```
<div class="col-6 offset-md-6">
    <form action="{{route('filiere.search')}}" method="post">
        @csrf
        <div class="form-group">
            <label for="motCle">Chercher par...</label>
            <input type="text"
                class="form-control" name="motCle" id="motCle" aria-describedby="motCleId"
                <small id="motCleId" class="form-text text-danger h6">{{ $erro
        </div>
    </form>
</div>
```

```
Route::post('/filiere/search/',
    [FiliereController::class, 'search'])
->name('filiere.search');
```

The screenshot shows a web application interface. At the top right is a search bar with placeholder text "Chercher par..." and a blue search button. Below it is a table titled "Liste des filières". The table has columns: Id, Nom, Code numérique, Téléphone, and Actions. It contains three rows of data:

Id	Nom	Code numérique	Téléphone	Actions
1	Management Informatisé des Organisations	MIO	1	
5	Licence Informatique	LI	5	
8	Licence Mathématique Informatique	LMI	7	

# Générer des données de test dans la BD

Lors du test, vous devrez peut-être insérer quelques enregistrements dans votre base de données avant d'exécuter votre test. Au lieu de spécifier manuellement la valeur de chaque colonne lorsque vous créez ces données de test, Laravel vous permet de définir un ensemble d'attributs par défaut pour chacun de vos modèles Eloquent à l'aide de fabriques (Factory) de modèles.

La commande ***make:factory*** pour créer un factory

➤ ***php artisan make:factory Filiere***

```
public function definition()
{
    return [
        'nom'=>$this->faker->unique()->name(),
        'code'=>$this->faker->unique()->name(),
        'code_numerique' => random_int(10,1000),
        'telephone'=>$this->faker->unique()->phoneNumber()
    ];
}
```

# Générer des données de Tests

La commande tinker pour executer des codes de l'application depuis le CMD

➤ `php artisan tinker`

Cette appelle spécifie le nombre d'enregistrements à créer et à appliquer dans la base de données dont **50** dans notre cas:

>> `App\Models\Filiere::factory()->count(50)->create()`

Après cela, vous remarquerez qu'il y'a pleins d'enregistrements dans notre liste et qu'on devrait trouver un moyen de réduire la quantité chargée, ce qui permettra de rendre l'application plus rapide et améliorer l'experience Utilisateur.

# Charger les données par pagination

Il existe plusieurs façons de paginer les éléments. Le plus simple consiste à utiliser la méthode `paginate()` sur le [query builder](#) ou une [Eloquent query](#).

La méthode `paginate()` prend automatiquement en charge la définition de la limite et du décalage appropriés en fonction de la page en cours de visualisation par l'utilisateur.

Par défaut, la page actuelle est détectée par la valeur de l'argument de chaîne de requête de page sur la requête HTTP.

Cette valeur est automatiquement détectée par Laravel, et est également automatiquement insérée dans les liens générés par le paginateur.

# Charger les données par pagination

## FiliereController

```
public function index()
{
    $filieres = Filiere::simplePaginate(10);
    return view('filiere.index', [
        'filieres' => $filieres
    ]);
}
```

## index.blade.php

```
</table>
{{$filieres->links()}}
```

ID	Nom	Code	Code numérique	Téléphone	Actions
13	Mekhi Witting	Dr. Bernardo Treutel	51	463.503.0732	 
14	Mrs. Aubree Konopelski I	Mrs. Birdie Gulgowski PhD	896	+1.912.698.5625	 
15	Prof. Betsy Pouros	Prof. Lonnie Frami Sr.	273	308.998.5882	 
16	Marietta Farrell	Jaquan Beer	135	+15153529704	 
17	Trinity Rath	Eldora Brown	598	757.325.4318	 
18	Else Hackett V	Dr. Jarret Kuhn III	163	706-835-6806	 
19	Flo Wuckert	Prof. Berta Rowe	51	360-692-2948	 
20	Cedrick Hintz	Mossie Lynch	364	272.725.5033	 
21	Mr. Brian Kub	Vincenzo Runolfsson	968	678.307.2098	 
22	Kameron Wisoky PhD	Tierra Deckow Jr.	807	1-458-757-0392	 
<a href="#">« Previous</a>		<a href="#">Next »</a>			

# TP

- Créer le CRUD complet de année avec le nom Anneeacad et les attributs suivants :
  - Nom : chaîne de caractère, unique
  - Code: chaîne de caractère avec une taille de 5 caractères net, unique
  - Etat : boolean, nullable
- Insérer 20 années académiques grâce aux factories et gérer la pagination sur la liste

# Gestion des utilisateurs

## Création des utilisateurs

- Générer un contrôleur pour user
  - `php artisan make:controller UserController`
  - Implémenter une méthode de création de compte avec le nom `signup()`

Créer une page de création de compte du nom de `signup.blade.php`

Gestion des utilisateurs

**Authentification des utilisateurs**

# Envoie d'emails

# Les types de relations (clés étrangères)

# Refs

- <https://www.futura-sciences.com/tech/actualites/internet-web-30-ans-quatre-grandes-phases-son-evolution-75312/>
- <https://home.cern/fr/science/computing/birth-web>
- <https://www.net-concept.fr/actualites/le-developpement-web-cest-quoi/#:~:text=Dans%20cet%20article%20nous%20examinerons,notamment%20Python%2C%20Ja~vaScript%20et%20HTML.>
- <https://www.net-concept.fr/actualites/le-developpement-web-cest-quoi/#:~:text=Dans%20cet%20article%20nous%20examinerons,notamment%20Python%2C%20Ja~vaScript%20et%20HTML.>
- <https://kinsta.com/fr/blog/frameworks-php/#:~:text=Un%20framework%20PHP%20est%20une,de%20code%20original%20%C3%A0%20%C3%A9crire.>
- <https://kinsta.com/fr/blog/tutoriels-laravel/>