

# Application avec Laravel 10

*Par*  
*Trésor KABISAYI*

## 1. Les prérequis

Laravel avec sa version 10, a besoin de quelques éléments côté serveur :

- PHP >= 8.1
- des extensions PHP (que du classique) :
  - Extension PDO,
  - Extension Mbstring,
  - Extension OpenSSL,
  - Extension Session,
  - Extension Tokenizer,
  - Extension XML.,
  - Extension Ctype,
  - Extension Fileinfo,
  - Extension DOM,
  - Extension PCRE,
  - Extension cUrl,
  - Extension Filter,
  - Extension Hash

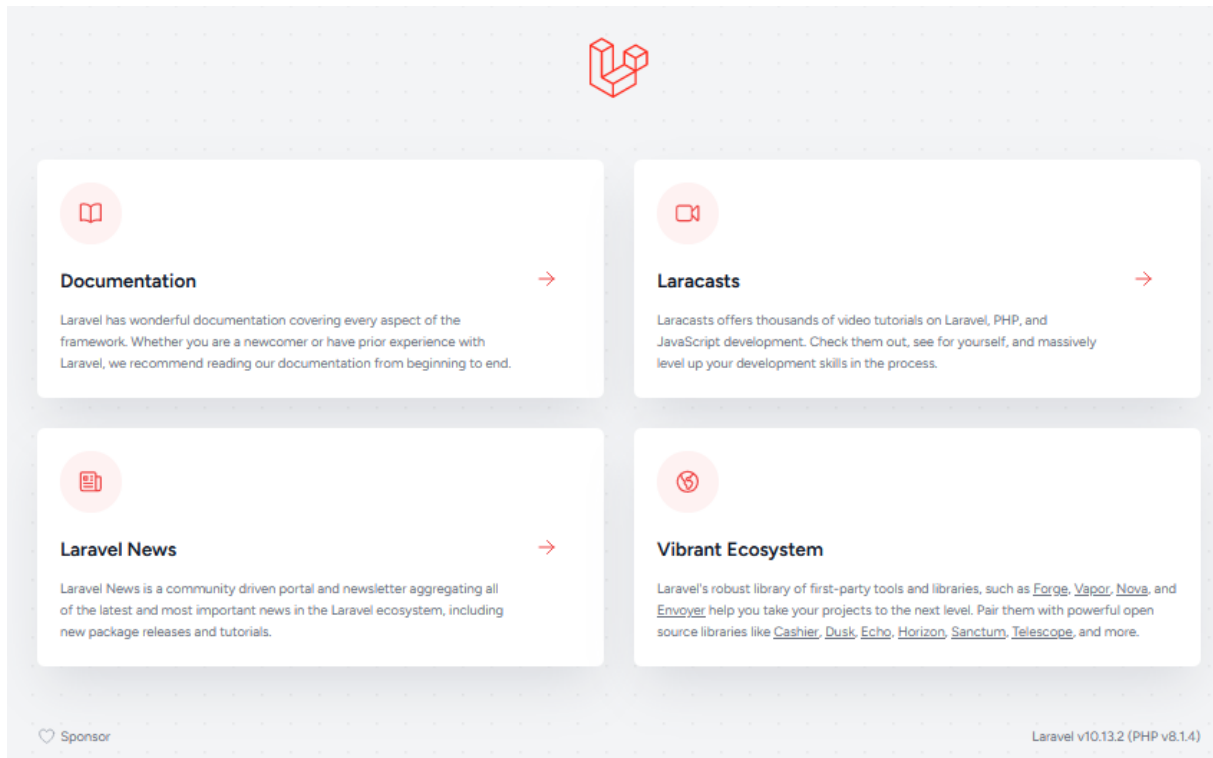
Vous pouvez trouver tout ça facilement sur un serveur local comme WAMPP ou XAMPP. On va aussi utiliser MySQL comme serveur de données. Vous aurez aussi besoin de Composer pour gérer les librairies PHP. Enfin pour le frontend il vous faudra node.js.

## 2. Création du projet

```
composer create-project laravel/laravel:10 stock
```

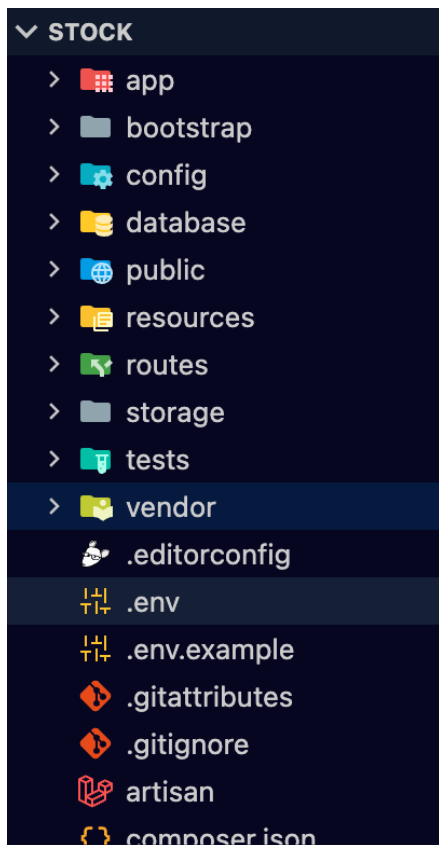
Il faudrait ensuite créer la base des données dans MySQL. Laravel dispose d'un serveur PHP local léger qu'on démarre en utilisant l'outil Artisan de Laravel :

```
php artisan serve
```

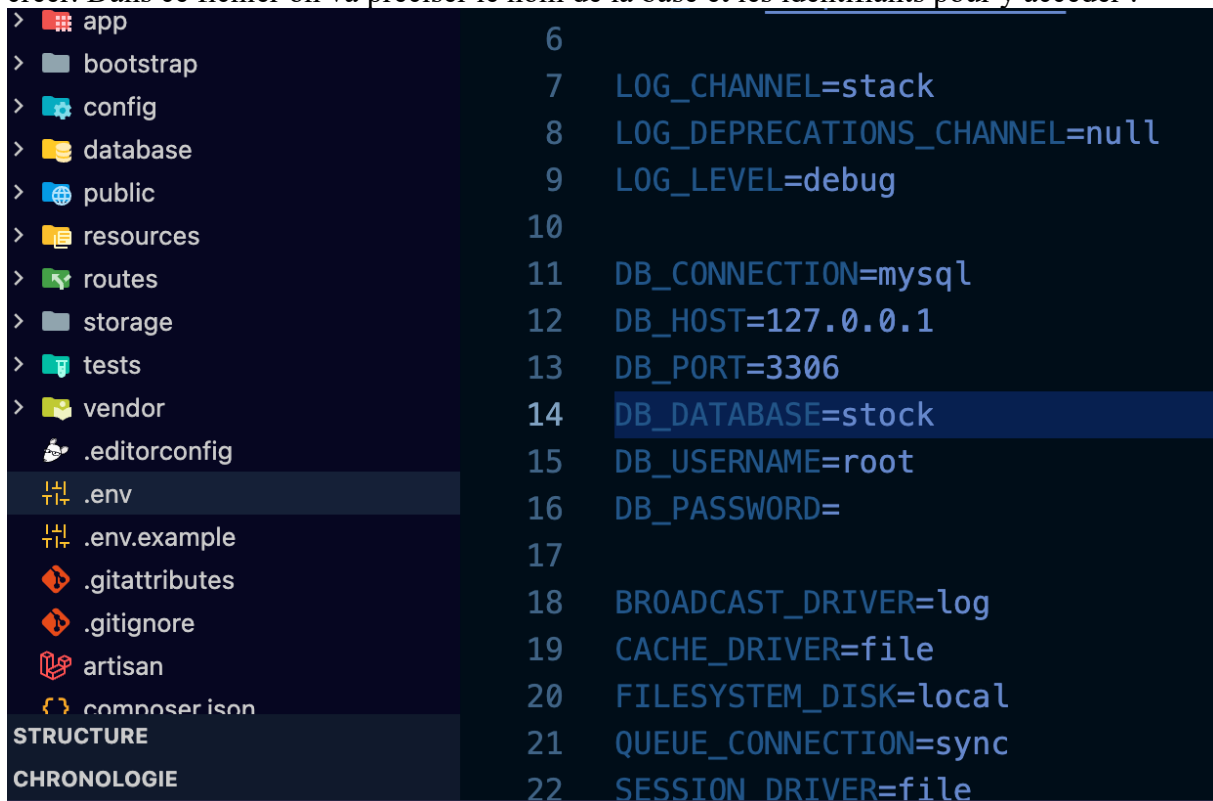


### 3. Base de données

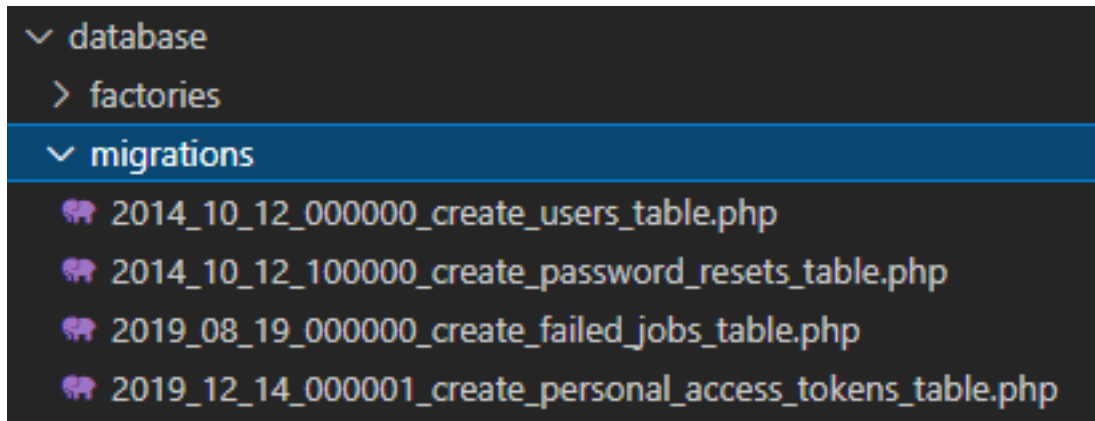
Vous avez créé la base de données mais il faut renseigner Laravel pour qu'il s'en serve. Pour ça on va utiliser le fichier de configuration `.env` qui se situe à la racine :



Si pour une raison quelconque ce fichier n'existe pas faite une copie de `.env.example` pour le créer. Dans ce fichier on va préciser le nom de la base et les identifiants pour y accéder :



Pour l'instant la base est vide. Laravel dispose d'un outil de migration pour créer des tables, les modifier, créer des index, etc. Vous le trouvez dans le dossier `database` :



On trouve 4 fichiers de migration :

- Pour la table **users**
- Pour la table **password\_resets**
- Pour la table **failed\_jobs**
- Pour la table **personnal\_access\_tokens**

Comme on ne servira pas des deux dernières on va supprimer ces fichiers. Il ne va donc plus rester que la migration pour la table **users** et **password\_resets**. Si on regrade un peu le code pour users on a une fonction **up** :

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

C'est ici qu'on a le code pour créer la table et ses colonnes. Pour créer les tables à partir de cette migration on utilise encore Artisan :

```
E:\laragon\www\todolist>php artisan migrate

INFO Preparing database.

Creating migration table ..... 8ms DONE

INFO Running migrations.

2014_10_12_000000_create_users_table ..... 10ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 14ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 12ms DONE
```

Là il peut vous sembler étrange que, bien qu'ayant supprimé la migration pour la table **personnal\_access\_tokens**, elle soit quand même créée. Cela est dû au fait que le package **Laravel\Sanctum** (qui sert à faciliter l'authentification pour certaines applications) est installé par défaut et comporte cette migration. Pour corriger ça vous allez intervenir dans la classe **App\Providers\AppServiceProvider** :

```
use Laravel\Sanctum\Sanctum;

...

public function register()
{
    Sanctum::ignoreMigrations();
}
```

On lance une commande pour rafraichir la base :

```
E:\laragon\www\todolist>php artisan migrate:fresh

Dropping all tables ..... 26ms DONE

INFO Preparing database.

Creating migration table ..... 7ms DONE



INFO Running migrations.

2014_10_12_000000_create_users_table ..... 10ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 14ms DONE
```

Si on regarde dans la base on se retrouve avec 3 tables :

Table
migrations
password_reset_tokens
users

On a bien nos tables **users** et **password\_reset\_tokens** mais aussi une table **migrations** qui mémorise les actions de migrations de Laravel. C'est une table d'intendance que vous n'avez pas à toucher. On a dans la table **users** les colonnes telles que définies dans la migration :

#	Nom
1	id 
2	name
3	email 
4	email_verified_at
5	password
6	remember_token
7	created_at
8	updated_at

## 4. L'authentification

Laravel 10 n'est pas équipé à la base d'un système d'authentification mais on peut l'ajouter avec un package complémentaire, c'est ce qu'on va faire en utilisant Composer :

```
composer require laravel/breeze --dev
```

```

E:\laragon\www\todolist>composer require laravel/breeze --dev
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^1.21 for laravel/breeze
./composer.json has been updated
Running composer update laravel/breeze
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/breeze (v1.21.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading laravel/breeze (v1.21.0)
  - Installing laravel/breeze (v1.21.0): Extracting archive
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

laravel/breeze ..... DONE
laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

82 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

 INFO  No publishable resources for tag [laravel-assets].

```

Composer a ajouté la librairie dans le fichier composer.json :

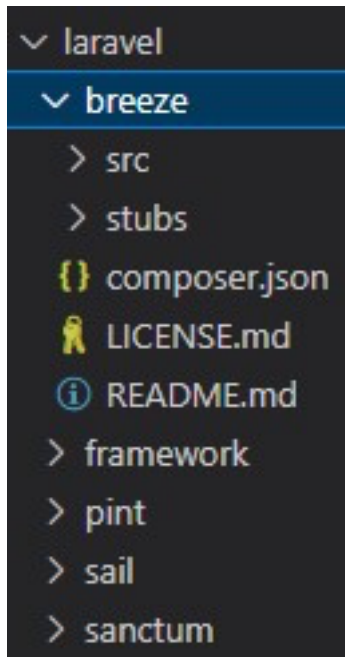
```

"require-dev": {
    ..
    "laravel/breeze": "^1.21",
    ...
},

```

Et l'a chargée dans le dossier **vendor** :





On va poursuivre l'installation de ce package :

```
E:\laragon\www\todolist>php artisan breeze:install

Which stack would you like to install?
blade ..... 0
react ..... 1
vue ..... 2
api ..... 3
```

Ici on a le choix, on va sélectionner la première option **blade**. Répondez non aux autres questions. On va enfin compiler les assets :

```
npm install
npm run dev
```

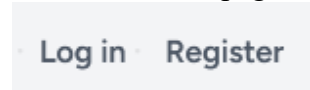
On va avoir la création d'un dossier **node\_modules** avec toutes les dépendances. Laravel utilise **Vite** pour la compilation à sa version 10.

```
> dev
> vite

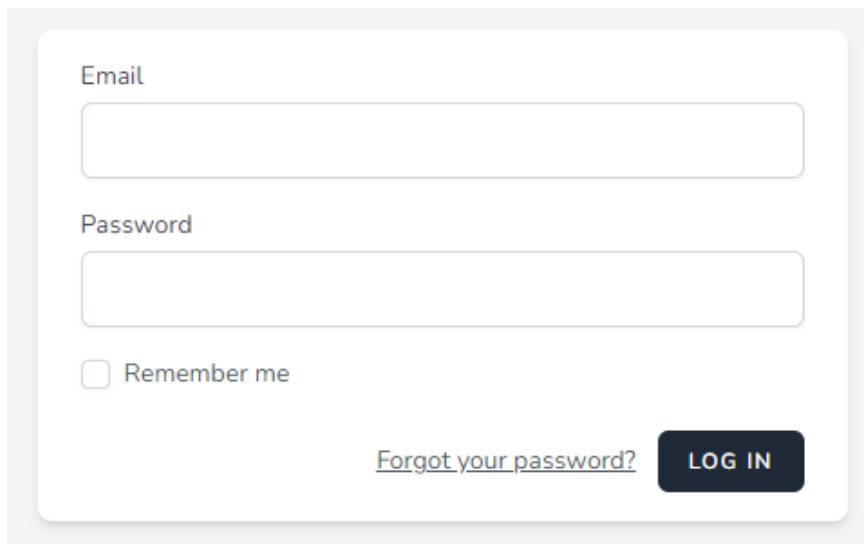
VITE v4.5.5 ready in 170 ms
  → Local:   http://localhost:5173/
  → Network: use --host to expose
  → press h to show help

LARAVEL v10.48.25 plugin v0.7.8
  → APP_URL: http://localhost
```

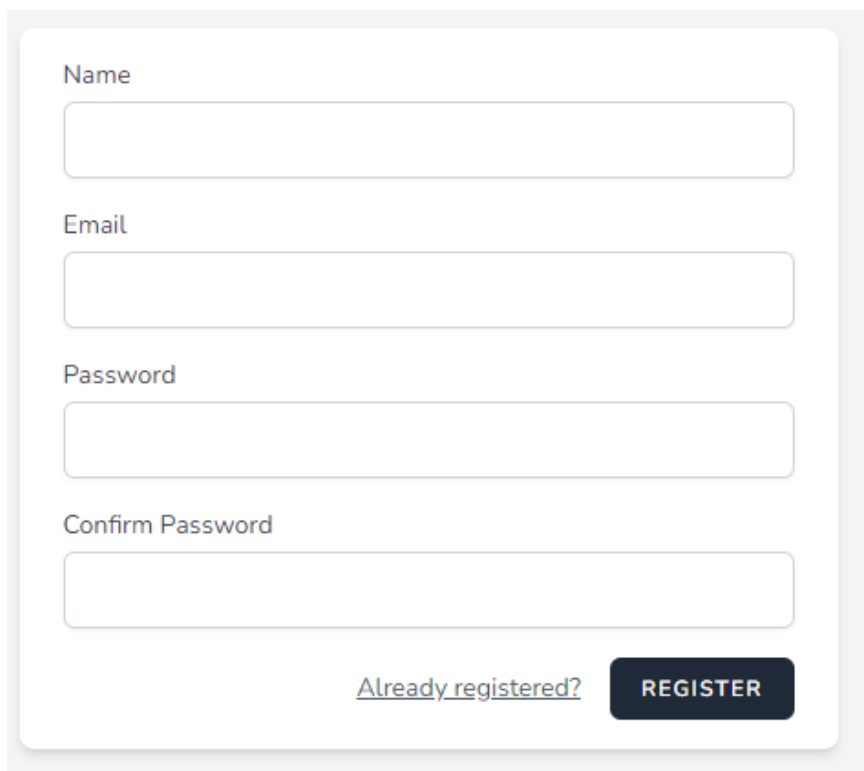
Au niveau de la page d'accueil on se retrouve avec deux liens :



On a ainsi un formulaire pour le login :

A login form with a light gray background. It contains two input fields: 'Email' and 'Password'. Below the 'Password' field is a checkbox labeled 'Remember me'. At the bottom right, there is a link 'Forgot your password?' and a dark blue button labeled 'LOG IN'.

Et un pour l'inscription :

A registration form with a light gray background. It contains four input fields: 'Name', 'Email', 'Password', and 'Confirm Password'. At the bottom right, there is a link 'Already registered?' and a dark blue button labeled 'REGISTER'.

Il est d'autre part possible de réinitialiser le mot de passe.

## 5. Les langues

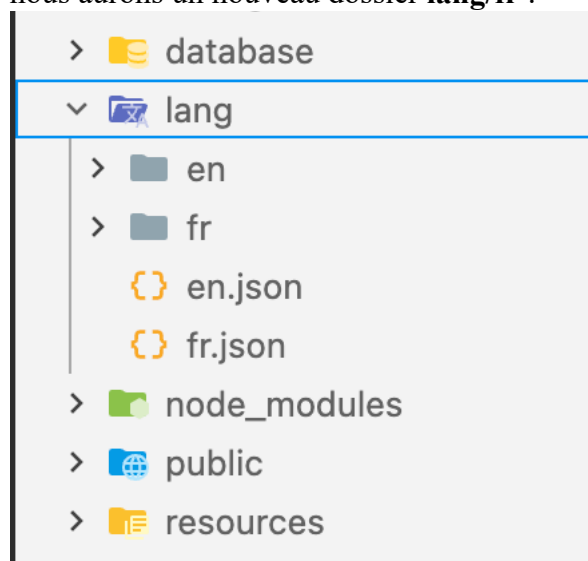
Normalement à la base tout est en anglais mais Laravel sait très bien gérer l'aspect linguistique. Voyons comment installer le français.

```
composer require laravel-lang/common --dev
```

```
php artisan lang:add fr
```

```
php artisan lang:update
```

nous aurons un nouveau dossier **lang/fr** :



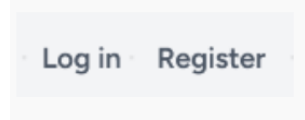
Ensuite vous changez la langue dans **config/app.php** :

```
'locale' => 'fr',
```

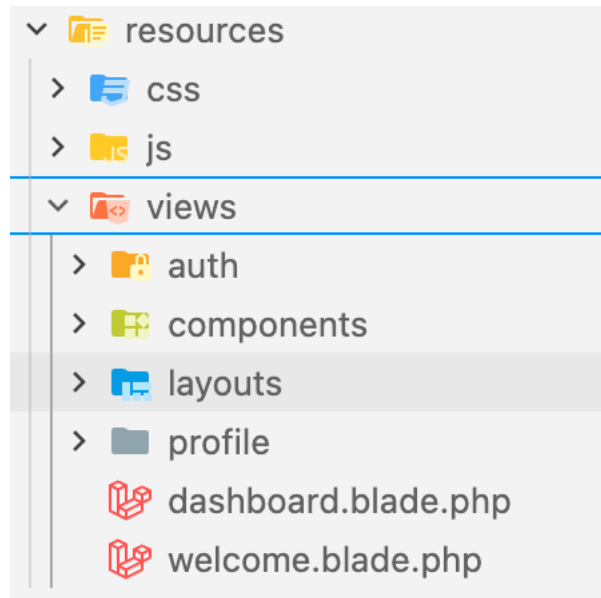
Vous aurez ainsi en français les messages pour l'authentification et la validation.



Par contre on voit que ça ne change rien sur la page d'accueil :



Toutes les vues de Laravel sont dans le dossier **resources/views** :



Le package, qu'on a installé, a ajouté les vues du dossier **auth**. La vue de l'accueil est **welcome.blade.php**. On a les deux textes dans ce code :

```
<a href="{{ route('login') }}" class="font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">
```

Log in

```
</a>
```

```
@if (Route::has('register'))
```

```
<a href="{{ route('register') }}" class="ml-4 font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">
```

Register

```
</a>
```

```
@endif
```

Nous pouvons faire un changement brutal et mettre les textes en français dans la vue, mais alors notre application ne sera plus multilingue. Après tout, si nous voulons notre site

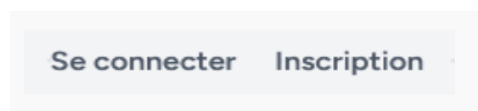
exclusivement en français ça ne pose aucun problème. Mais si nous voulons conserver la possibilité d'avoir plusieurs langues nous devons trouver une méthode plus élégante...

Nous utilisons un helper de Blade :

```
<a href="{{ route('login') }}" class="font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">
  {{ __('Log in') }}
</a>
```

```
@if (Route::has('register'))
  <a href="{{ route('register') }}" class="ml-4 font-semibold text-gray-600 hover:text-gray-900 dark:text-gray-400 dark:hover:text-white focus:outline focus:outline-2 focus:rounded-sm focus:outline-red-500">
    {{ __('Register') }}
  </a>
@endif
```

Nous obtenons le même résultat mais maintenant le site est multilingues.



Pour voir les traductions correspondantes, il faut ouvrir **lang/fr.json**.

```
"Login": "Connexion",
"Logout": "Déconnexion",
```

## 6. Les données

Il est temps de passer à la modélisation. Nous partons sur cette base :

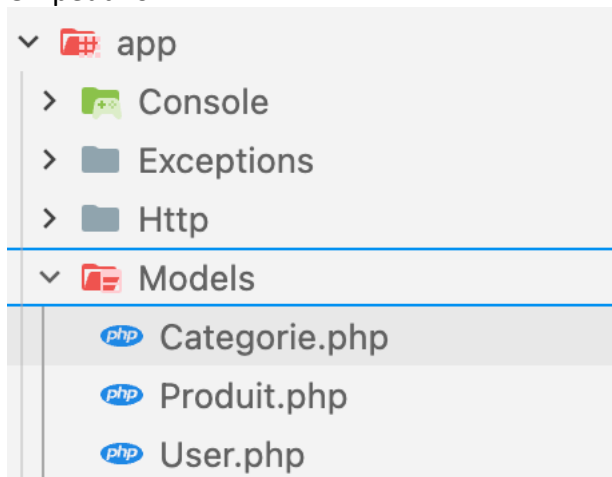
- L'entité Catégorie
  - code 10T unique
  - titre 100T unique
- L'entité Produit
  - reference 10T unique
  - libelle 100T

- quantité N
- prix N
- categorie\_id FK

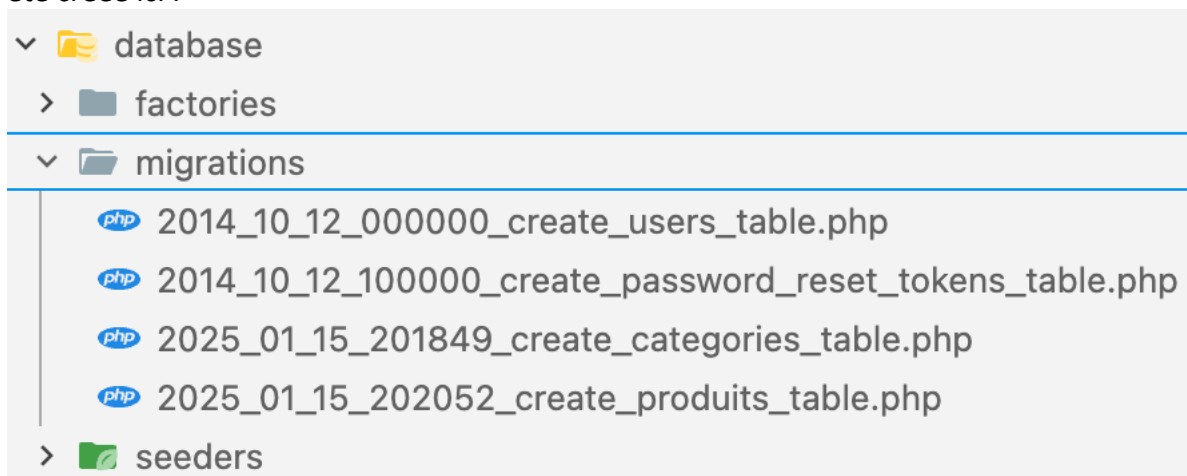
Nous allons donc créer les tables pour mémoriser toutes les données. On a vu qu'avec Laravel, on utilise des migrations. D'autre part Laravel est équipé d'un ORM efficace : **Eloquent**. Chaque table est représentée par une classe qui permet de manipuler les données. On dispose ainsi d'un modèle pour chaque table à manipuler. On va demander à Artisan de créer à la fois une table et son modèle Eloquent associé :

- pour Categorie : `php artisan make:model Categorie -m`  
 INFO Model [app/Models/Categorie.php] created successfully.  
 INFO Migration [database/migrations/2025\_01\_15\_201849\_create\_categories\_table.php] created successfully.
- pour Produit : `php artisan make:model Produit -m`  
 INFO Model [app/Models/Produit.php] created successfully.  
 INFO Migration [database/migrations/2025\_01\_15\_202052\_create produits\_table.php] created successfully.

On peut voir :



On a déjà le modèle par défaut **User** pour gérer les données de la tables **users**. La migration a été créée ici :



Pour categories, nous avons par défaut les codes suivants :

```
public function up(): void
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

On a :

- une clé **id**
- deux colonnes (**created\_at** et **updated\_at**) créée par la méthode **timestamps**.

Nous ajoutons les colonnes nécessaires :

Pour categories :

```
public function up(): void
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('code', 10)->unique();
        $table->string('titre', 100)->unique();
        $table->timestamps();
    });
}
```

Pour produits :

```
public function up(): void
{
    Schema::disableForeignKeyConstraints();
    Schema::create('produits', function (Blueprint $table) {
        $table->id();
        $table->string('reference', 10)->unique();
        $table->string('libelle', 100);
        $table->integer('quantite');
        $table->integer('prix');
        $table->timestamps();
        $table->foreignId('categorie_id')
            ->constrained()
            ->onUpdate('cascade')
            ->onDelete('cascade');
    });
}
```

Dans la table **produits** on déclare une clé étrangère nommée **category\_id** qui référence la colonne **id** dans la table **categories**. La méthode **foreignId** crée une colonne de type UNSIGNED BIGINT. La méthode **constrained** utilise les conventions de Laravel pour déterminer le nom de la table référencée, ici c'est **categories**. En cas de suppression (**onDelete**) ou de modification (**onUpdate**) on peut appliquer le mode cascade (**cascade**).

*Que signifient ces deux dernières conditions ?*

Imaginez que vous ayez une catégorie avec l'id 5 qui a deux produits, donc dans la table **produits** on a deux enregistrements avec **category\_id** qui a la valeur 5. Si on supprime la catégorie que va-t-il se passer ? On risque de se retrouver avec nos deux enregistrements dans la table **produits** avec une clé étrangère qui ne correspond à aucun enregistrement dans la table **categories**. En mettant **restrict** on empêche la suppression d'une catégorie qui a des produits. On doit donc commencer par supprimer ses films avant de le supprimer lui-même. On dit que la base assure l'intégrité référentielle. Elle n'acceptera pas non plus qu'on utilise pour **category\_id** une valeur qui n'existe pas dans la table **categories**. Une autre possibilité est **cascade** à la place de **restrict**. Dans ce cas si vous supprimez une catégorie ça supprimera en cascade les films de cette catégorie.

C'est une option qui est moins utilisée parce qu'elle peut s'avérer dangereuse, surtout dans une base comportant de multiples tables en relation. Mais c'est aussi une stratégie très efficace parce que c'est le moteur de la base de données qui se charge de gérer les enregistrements en relation, vous n'avez ainsi pas à vous en soucier au niveau du code.

On pourrait aussi ne pas signaler à la base qu'il existe une relation et la gérer seulement dans notre code. Mais c'est encore plus dangereux parce que la moindre erreur de gestion des enregistrements dans votre code risque d'avoir des conséquences importantes dans votre base avec de multiples incohérences.

On va lancer les migrations :

```
php artisan migrate
```

Les migrations sont effectuées dans l'ordre alphabétique, ce qui peut générer un problème avec les clés étrangères. Si la table référencée est créée après on va tomber sur une erreur du genre :

```
Illuminate\Database\QueryException : SQLSTATE[HY000]: General error: 1215 Cannot add foreign key constraint (SQL: alter table `produits` add constraint `produits_category_id_foreign` foreign key (`category_id`) references `categories` (`id`) on delete restrict on update restrict)
```

C'est pour cette raison que nous avons ajouté cette ligne dans la migration :

```
Schema::disableForeignKeyConstraints();
```

On désactive temporairement le contrôle référentiel le temps de créer les tables.



## 7. Le contrôleur et les routes

Laravel est basé sur le modèle MVC :

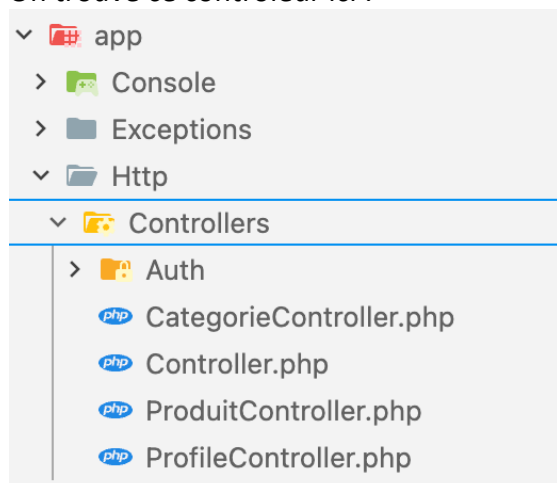
- **Modèle** : c'est Eloquent qui se charge de cet aspect et on a créé deux modèles (Categorie et Produit)
- **Vue** : on a déjà des vues pour la partie authentification, on va devoir en créer aussi pour nos tables
- **Contrôleur** : c'est le chef d'orchestre de l'application, on va créer à présent un contrôleur pour chaque modèle afin de gérer toutes les actions nécessaires pour les catégories et les produits.

```

tresorkabis@MacBookPro-Tresor stock % php artisan make:controller CategorieController --resource
INFO Controller [app/Http/Controllers/CategorieController.php] created successfully.
tresorkabis@MacBookPro-Tresor stock % php artisan make:controller ProduitController --resource
INFO Controller [app/Http/Controllers/ProduitController.php] created successfully.

```

On trouve ce contrôleur ici :



Par exemple pour catégorie, nous avons un contrôleur de ressource. Cela signifie qu'il est déjà équipé de 7 fonctions pour les actions suivantes :

Verbe	URI	Action	Route
GET	/categories	index	categories.index
GET	/categories/create	create	categories.create
POST	/categories	store	categories.store
GET	/categories/{categorie}	show	categories.show
GET	/categories/{categorie}/edit	edit	categories.edit
PUT/PATCH	/categories/{categorie}	update	categories.update
DELETE	/categories/{categorie}	destroy	categories.destroy

On va créer aussi les routes pour accéder à ces actions dans le fichier **routes/web.php** :

```
use App\Http\Controllers\CategorieController;
```

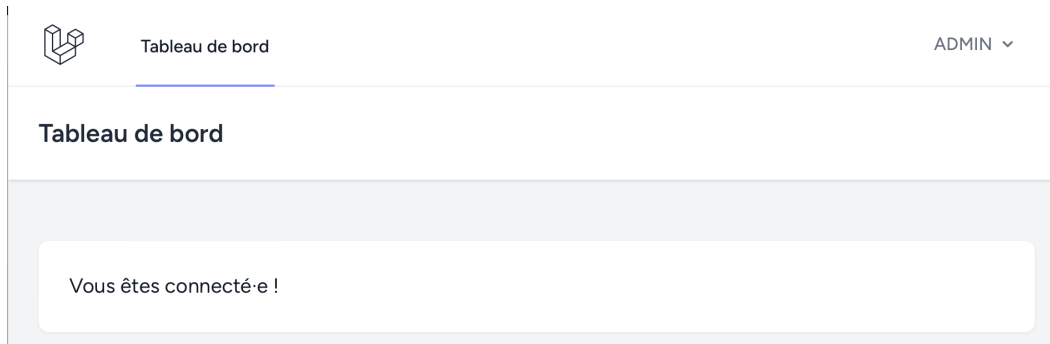
```
Route::resource('categories', CategorieController::class)->middleware('auth');
```

Si vous utilisez la commande **php artisan route:list** vous obtenez toutes les routes de l'application et en particulier les 7 pour le contrôleur :

```
GET|HEAD      categories ..... categories.index > CategorieController@index
POST          categories ..... categories.store > CategorieController@store
GET|HEAD      categories/create ..... categories.create > CategorieController@create
GET|HEAD      categories/{category} ..... categories.show > CategorieController@show
PUT|PATCH    categories/{category} ..... categories.update > CategorieController@update
DELETE        categories/{category} ..... categories.destroy > CategorieController@destroy
GET|HEAD      categories/{category}/edit ..... categories.edit > CategorieController@edit
```

Maintenant que tout ça est en place on va coder ces actions et créer les vues correspondantes. L'utilité du middleware **auth** que nous avons ajouté pour ces routes. Un middleware est une étape pour la requête http qui arrive, ça permet d'accomplir des vérifications, ici on demande que l'accès à ces routes soit limitées aux utilisateurs authentifiés. Ceux qui ne le sont pas seront automatiquement renvoyés à la page de connexion. Pour la suite, nous devons créer un utilisateur avec les formulaires déjà en place.

Remarquez que lorsqu'un utilisateur est connecté avec **Breeze** il se retrouve avec un Dashboard :



Vous avez aussi accès à une page de profil.

## 8. Organisation des vues

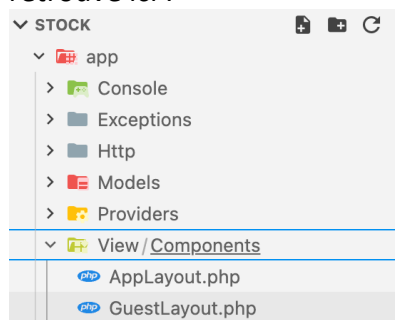
Par défaut Breeze utilise Tailwind avec un thème spécifique et toutes les vues de l'authentification ainsi que le tableau de bord et le profil l'utilise, on peut évidemment changer tout ça si on n'aime pas Tailwind, ce qui est d'ailleurs mon cas. Pour ce cours, nous allons nous contenter de prendre la situation telle qu'elle est proposée.

Laravel propose plusieurs façon d'organiser les vues, on peut les combiner avec un simple héritage ou utiliser des composants. Breeze met en place de nombreux composants et en fait une utilisation intensive, alors voyons un peu de quoi il s'agit. Si vous regardez la vue du dashboard (**views/dashboard.blade.php**) on a ce code :

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Dashboard') }}
    </h2>
  </x-slot>

  <div class="py-12">
    <div class="max-w-7xl mx-auto sm:px-6 lg:px-8">
      <div class="bg-white overflow-hidden shadow-sm sm:rounded-lg">
        <div class="p-6 text-gray-900">
          {{ __('You're logged in!') }}
        </div>
      </div>
    </div>
  </div>
</x-app-layout>
```

La syntaxe un peu particulière **x-app-layout** indique qu'on utilise un composant qu'on retrouve ici :



Un composant possède classiquement une vue et une classe associée. Ici on s'intéresse à la classe **AppLayout**. Dans cette classe on a en particulier ce code :

```
public function render(): View
{
    return view('layouts.app');
}
```

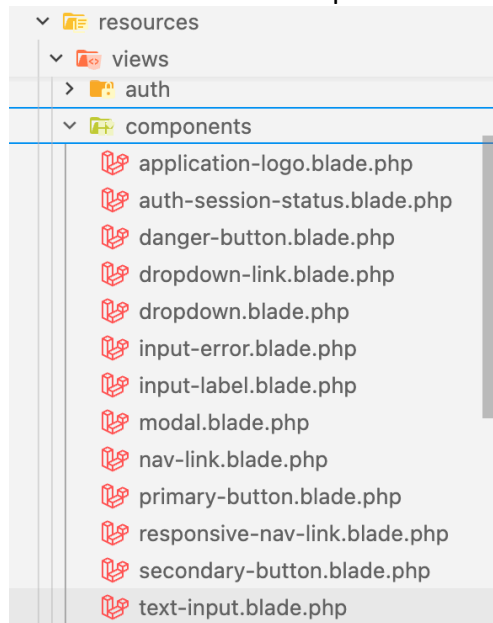
On voit qu'on retourne la vue **views/layouts/app.blade.php**. Donc quand nous utilisons le composant **<x-app-layout>** nous faisons en fait appel à cette vue. Si vous ouvrez ce fichier, vous allez trouver le code HTML de base de la page. On trouve dans cette vue ce code :

```
@include('layouts.navigation')
```

Qui veut dire que l'on inclut ici la vue **views/layouts/navigation.blade.php** qui correspond à cette partie de la page :



Parmi les nombreux composants on en trouve un pour un contrôle simple de formulaire :

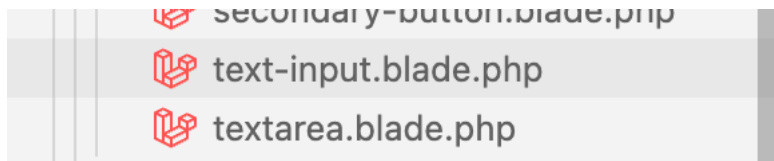


Le code est simple :

```
@props(['disabled' => false])
```

```
<input {{ $disabled ? 'disabled' : '' }} {!! $attributes->merge(['class' =>
'border-gray-300 focus:border-indigo-500 focus:ring-indigo-500 rounded-md shadow-
sm']) !!} >
```

Mais l'avantage est de mutualiser ce code et de l'utiliser pour tous les formulaires. Pour notre usage on va créer un composant équivalent pour les **textarea** :

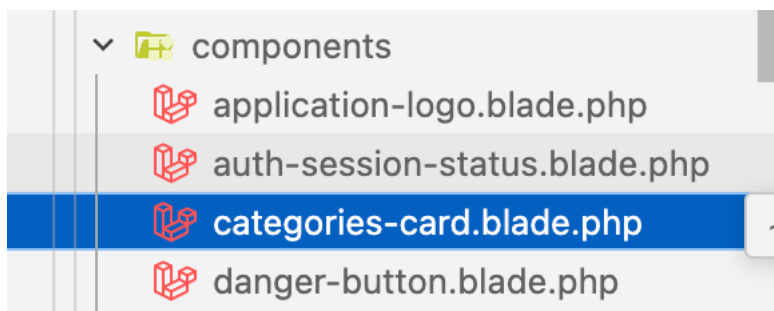


Avec ce code :

```
@props(['disabled' => false])
```

```
<textarea {{ $disabled ? 'disabled' : '' }} {!! $attributes->merge(['class' =>
'border-gray-300 focus:border-indigo-500 focus:ring-indigo-500 rounded-md shadow-
sm']) !!} >{{ $slot }}</textarea>
```

On va aussi ajouter le composant **categories-card** :



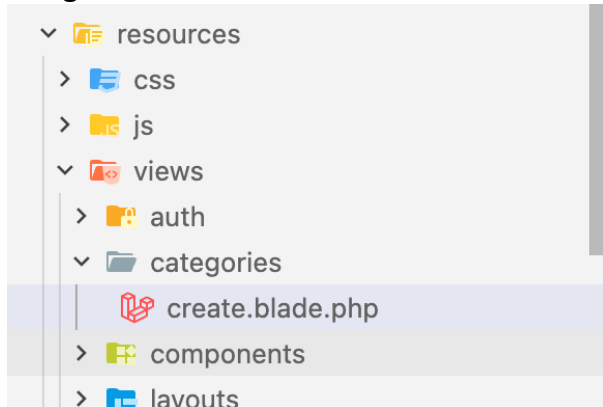
Avec ce code :

```
<div class="mt-8 flex flex-col sm:justify-center items-center pt-6 sm:pt-0 bg-
gray-100">
  <div class="w-full sm:max-w-md mt-6 px-6 py-4 bg-white shadow-md overflow-
hidden sm:rounded-lg">
    {{ $slot }}
  </div>
</div>
```

## 9. Création d'une catégorie

### 9.1. Le formulaire

Pour la création d'une catégorie, nous avons besoins d'un formulaire. Créons un dossier **categories** et la vue **create** :



Avec ce code :

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Create a category') }}
    </h2>
  </x-slot>

  <x-categories-card>

    <!-- Message de réussite -->
    @if (session()->has('message'))
      <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
        {{ session('message') }}
      </div>
    @endif

    <form action="{{ route('categories.store') }}" method="post">
      @csrf

      <!-- Code -->
      <div>
        <x-input-label for="code" :value="__('Code')" />

        <x-text-input id="code" class="block mt-1 w-full" type="text"
name="code" :value="old('code')" required autofocus />

        <x-input-error :messages="$errors->get('code')" class="mt-2" />
      </div>

      <!-- Titre -->
      <div class="mt-4">
        <x-input-label for="titre" :value="__('Titre')" />
```

```

        <x-textarea class="block mt-1 w-full" id="titre" name="titre">{{
old('titre') }}</x-textarea>

        <x-input-error :messages="$errors->get('titre')" class="mt-2" />
    </div>

    <div class="flex items-center justify-end mt-4">
        <x-primary-button class="ml-3">
            {{ __('Send') }}
        </x-primary-button>
    </div>
</form>

</x-categories-card>
</x-app-layout>

```

Il y a beaucoup à dire sur ce code. Le formulaire est défini avec ce code :

```
<form action="{{ route('categories.store') }}" method="post">
```

On utilise l'helper **route** qui, à partir du nom de la route, génère l'url. Pour insérer des éléments avec PHP on utilise des accolades.

On a ensuite cette commande Blade :

```
@csrf
```

On demande ici à Blade d'insérer un token dans un champs caché qui va permettre à Laravel de contrer les attaques CSRF, le code généré ressemble à ça :

```
<input type="hidden" name="_token" value="sUTuW5dYrt2l2iPgAJ5EVXn5UwMjEGFLAZaH4jHz">
```

C'est automatique et on n'a pas besoin de s'en préoccuper ensuite. Un middleware se charge à l'arrivée de ce contrôle de sécurité.

On va compléter le code de la fonction **create** du contrôleur **CategorieController** pour appeler la vue :

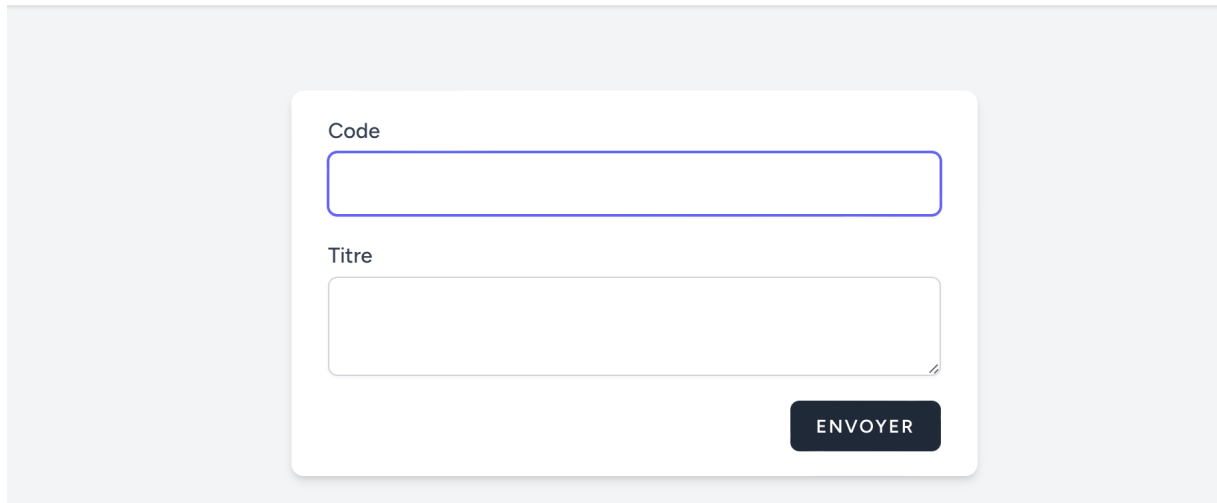
```

public function create()
{
    return view('categories.create');
}

```

Maintenant avec l'url <http://127.0.0.1:8000/categories/create> on obtient la page avec le formulaire (si on est connecté) :

### Create a category



Code

Titre

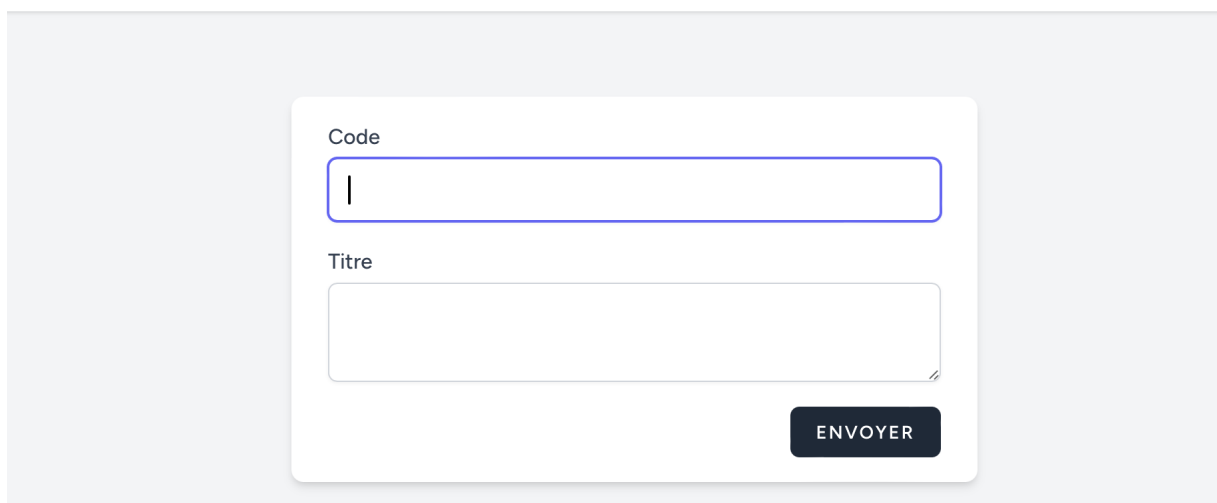
ENVOYER

On a avancé mais ça serait quand même mieux en français, pour cela il faut compléter le fichier **lang/fr.json** :

```
{  
  ...  
  "Create a category" : "Créer une catégorie"  
}
```

Et voyons le résultat après avoir actualiser.

### Créer une catégorie



Code

Titre

ENVOYER



## 9.2. La soumission

Pour créer effectivement la catégorie, on va devoir gérer la soumission du formulaire dans la méthode **store** du contrôleur :

```
public function store(Request $request)
{
    $data = $request->validate([
        'code' => 'required|max:10',
        'titre' => 'required|max:100',
    ]);

    $categorie = new Categorie();
    $categorie->code = $request->code;
    $categorie->titre = $request->titre;
    $categorie->save();

    return back()->with('message', "La catégorie a bien été créée !");
}
```

On commence par vérifier les entrées avec la validation. Les deux champs sont requis et on vérifie une longueur maximale.

Ensuite on utilise Eloquent pour créer la catégorie. Pour finir on renvoie dans la vue. Vous pouvez vérifier que la validation fonctionne :

Code

AZERTYUIOPPPPPPPPP

Le texte de code ne peut pas contenir plus de 10 caractères.

Si la validation est bonne, la catégorie est créée et on retourne la même vue (**back**) mais cette fois on flashe une information en session avec **with** :

```
return back()->with('message', "La catégorie a bien été créée !");
```

Dans la vue on vérifie s'il y a une information en session et si c'est le cas on l'affiche :

```
@if (session()->has('message'))
    <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
        {{ session('message') }}
    </div>
@endif
```

La catégorie a bien été créée !

Code

On trouve les données dans la table :

id	code	titre	created_at	updated_at
1	INFO	Informatique	2025-01-16 11:38:38	2025-01-16 11:38:38

Remarquez le renseignement automatique des deux dates. Pour accéder à ce formulaire de création on va ajouter un lien dans le menu dans la vue **views/layouts/navigation.blade.php** :

```
<!-- Navigation Links -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
  <x-nav-link :href="route('dashboard')" :active="request()->routeIs('dashboard')">
    {{ __('Dashboard') }}
  </x-nav-link>
</div>
```

```
<!-- Lien pour la création d'une catégorie -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
<x-nav-link
  :href="route('categories.create')"
  :active="request()->routeIs('categories.create')">
  {{ __('Create a category') }}
</x-nav-link>
</div>
```



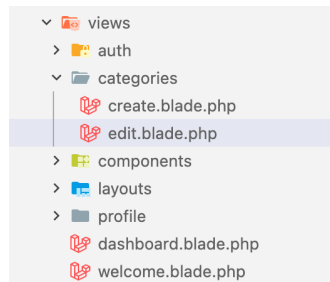
Tableau de bord

Créer une catégorie

## 10. Modification d'une catégorie

### 10.1. Le formulaire

Pour la modification d'une tâche, nous avons besoin d'un formulaire. Créons la vue **edit** :



Évidemment cette vue ressemble à celle de la création mais il va falloir renseigner les champs. Il faut aussi ajouter la possibilité de modifier l'état :

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Edit a category') }}
    </h2>
  </x-slot>

  <x-categories-card>
    <!-- Message de réussite -->
    @if (session()-has('message'))
      <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
        {{ session('message') }}
      </div>
    @endif

    <form action="{{ route('categories.update', $categorie->id) }}"
method="post">
      @csrf
      @method('put')

      <!-- Code -->
      <div>
        <x-input-label for="code" :value="__('Code')" />
        <x-text-input id="code" class="block mt-1 w-full" type="text"
name="code" :value="old('code')" value="{{ $categorie->code }}" required
autofocus />
        <x-input-error :messages="$errors->get('code')" class="mt-2" />
      </div>

      <!-- Titre -->
      <div class="mt-4">
        <x-input-label for="titre" :value="__('Titre')" />
        <x-textarea class="block mt-1 w-full" id="titre" name="titre">{{
old('titre') }} {{ $categorie->titre }}</x-textarea>
        <x-input-error :messages="$errors->get('titre')" class="mt-2" />
      </div>

      <div class="flex items-center justify-end mt-4">
        <x-primary-button class="ml-3">
          {{ __('Send') }}
        </x-primary-button>
      </div>
    </form>
  </x-categories-card>
</x-app-layout>
```

Dans la déclaration du formulaire, nous changeons l'url :

```
<form action="{{ route('categories.update', $categorie->id) }}" method="post">
```

On précise la route et il faut ajouter l'identifiant de la catégorie à modifier (**\$categorie->id**). Remarquez que la route attend une méthode **PUT** mais que là on déclare une méthode **POST**. C'est parce que les navigateurs gèrent encore très mal ce genre de méthode, alors on déclare un **POST** mais après on précise qu'on veut un **PUT** :

```
@method('put')
```

Cela crée un champ caché qui va servir à Laravel pour savoir de quelle méthode il s'agit :

```
<input type="hidden" name="_method" value="put" />
```

Pour le reste ça ne change pas beaucoup de la création. Dans le contrôleur on appelle la vue :

```
public function edit(Categorie $categorie)
{
    return view('categories.edit', compact('categorie'));
}
```

Remarquez le paramètre qui est du type du modèle **Categorie**. C'est une façon de dire à Laravel : le paramètre transmis est un nombre mais en fait c'est l'identifiant d'une instance de **Categorie**. Du coup Eloquent peu aller chercher dans la table la catégorie correspondante. Il suffit ensuite de transmettre à la vue.

Maintenant avec une url de la forme **/categories/1/edit** on atteint le formulaire.

## 10.2. La soumission

Pour modifier effectivement la catégorie, on va devoir gérer la soumission du formulaire dans la méthode **update** du contrôleur :

```
public function update(Request $request, Categorie $categorie)
{
    $data = $request->validate([
        'code' => 'required|max:10',
        'titre' => 'required|max:100',
    ]);
    $categorie->code = $request->code;
    $categorie->titre = $request->titre;
    $categorie->save();
}
```

```

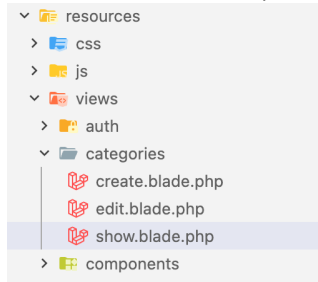
    }
    return back()->with('message', "La catégorie a bien été modifié !");
}

```

On a la même validation que pour la création. D'ailleurs, il faudrait pour bien faire mutualiser ce code éviter cette redondance qui n'est jamais une bonne pratique (principe DRY).

## 11. Voir une catégorie

On va partir du principe qu'on aura un tableau avec juste le titre des catégories et pas les autres informations, il faut donc prévoir de pouvoir afficher chaque catégorie. On crée la vue :



```

<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      @lang('Show a category')
    </h2>
  </x-slot>

  <x-categories-card>
    <h3 class="font-semibold text-xl text-gray-800">@lang('Code')</h3>
    <p> {{ $categorie->code }} </p>

    <h3 class="font-semibold text-xl text-gray-800">@lang('Titre')</h3>
    <p> {{ $categorie->titre }} </p>

    <h3 class="font-semibold text-xl text-gray-800 pt-2">
      @lang('Date creation')
    </h3>
    <p> {{ $categorie->created_at->format('d/m/Y') }} </p>
    @if($categorie->created_at != $categorie->updated_at)
      <h3 class="font-semibold text-xl text-gray-800 pt-2">
        @lang('Last update')
      </h3>
      <p> {{ $categorie->updated_at->format('d/m/Y') }} </p>
    @endif
  </x-categories-card>
</x-app-layout>

```

On complete le contrôleur:

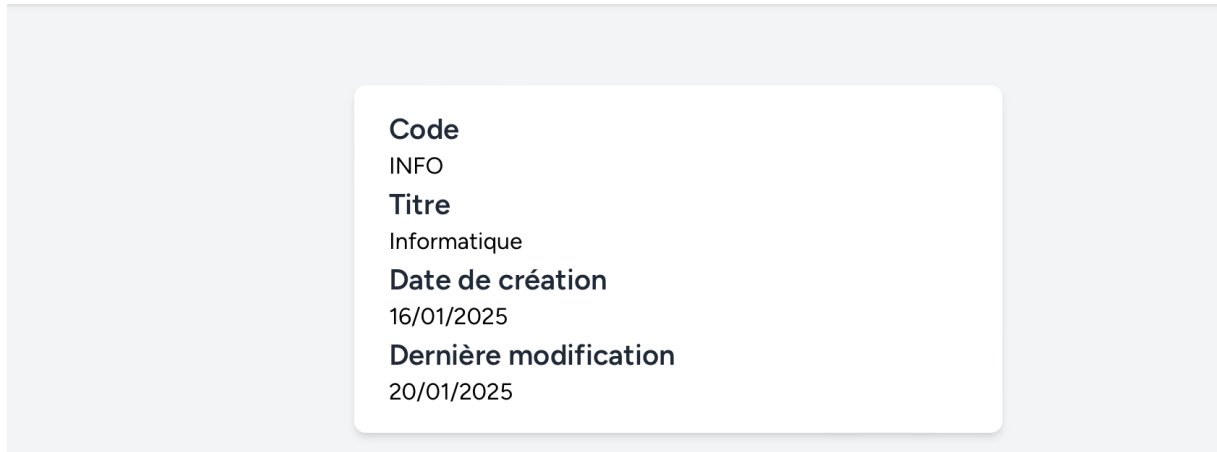
```

public function show(string $id)
{
    $categorie = Catégorie::find($id);
    return view('categories.show', compact('categorie'));
}

```

Et avec une url de la forme **/categories/1** on affiche les éléments d'une catégorie sans oublié les traductions :

### Voir une catégorie



On n'affiche la date de mise à jour que si elle est différente de celle de la création.

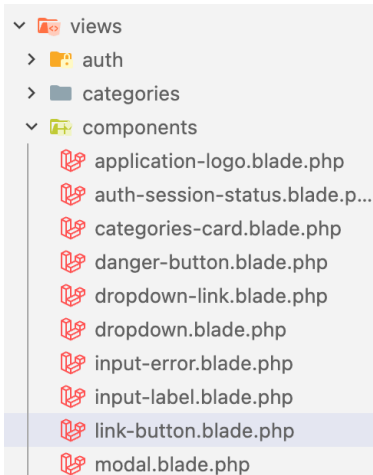
## 12. Liste des catégories

Maintenant qu'on sait créer, modifier et afficher des catégories on va voir comment en afficher la liste en prévoyant des boutons pour les différentes actions.

Au niveau du contrôleur, on va récupérer toutes les tâches et les envoyer dans une vue :

```
public function index()
{
    $categories = Category::paginate(5);
    return view("categories.index", compact('categories'));
}
```

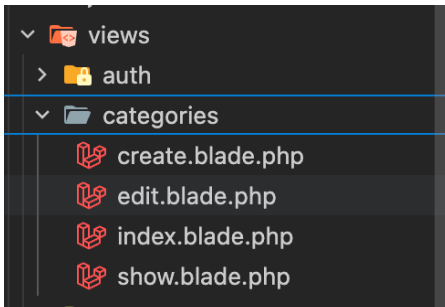
Nous ajoutons un composant pour les boutons dans la liste :



Avec ce code :

```
<td class="px-2 py-2">
  <a role="button" {{ $attributes->merge(['class' => 'inline-flex items-center
px-2 py-2 bg-gray-800 border border-transparent rounded-md font-semibold text-xs
text-white uppercase tracking-widest hover:bg-gray-700 focus:bg-gray-700
active:bg-gray-900 focus:outline-none focus:ring-2 focus:ring-indigo-500
focus:ring-offset-2 transition ease-in-out duration-150' ]) }} >
    {{ $slot }}
  </a>
</td>
```

On crée la vue :



```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      @lang('Categories List')
    </h2>
  </x-slot>
  <div class="container flex justify-center mx-auto">
    <div class="flex flex-col">

      <div class="w-full">
        @if (session()->has('message'))
          <div class="mt-3 mb-4 list-disc list-inside text-sm text-
green-600">
            {{ session('message') }}
          </div>
        @endif
        <div class="border-b border-gray-200 shadow pt-6">
          <table>
```





## Listes des catégories

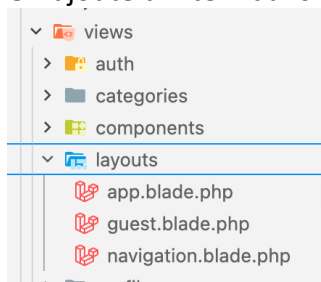
#	Code	Titre			
1	INFO	Informatique	AFFICHER	ÉDITER	SUPPRIMER
2	AGR	Agricole	AFFICHER	ÉDITER	SUPPRIMER
3	MUS	Musique	AFFICHER	ÉDITER	SUPPRIMER
4	COS	Cosmétique	AFFICHER	ÉDITER	SUPPRIMER
5	ELE	Electronique	AFFICHER	ÉDITER	SUPPRIMER

Montrant 1 à 5 de 6 résultats

< 1 2 >

Sans oublier les traductions.

On ajoute un item dans le menu (**navigation.blade.php**) :



```
...
<!-- Navigation Links -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
  <x-nav-link :href="route('dashboard')" :active="request()-
>routeIs('dashboard')">
    {{ __('Dashboard') }}
  </x-nav-link>
</div>

<!-- Lien pour la liste des catégories -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
  <x-nav-link :href="route('categories.index')" :active="request()-
>routeIs('categories.index')">
    {{ __('Categories List') }}
  </x-nav-link>
</div>

<!-- Lien pour la création d'une catégorie -->
<div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
  <x-nav-link :href="route('categories.create')" :active="request()-
>routeIs('categories.create')">
    {{ __('Create a category') }}
  </x-nav-link>
</div>
```

...

[Tableau de bord](#)[Listes des catégories](#)[Créer une catégorie](#)

Les boutons permettent d'accéder aux catégories qu'on a codées précédemment sauf la suppression. On va coder pour cela cette fonction dans le contrôleur :

```
public function destroy(string $id)
{
    Categorie::find($id)->delete();
    return redirect('categories')->with('message', "La catégorie a bien été
supprimée !");
}
```

## 13. La classe CategoryController

```

<?php

namespace App\Http\Controllers;

use App\Models\Categorie;
use Illuminate\Http\Request;

class CategoryController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $categories = Categorie::paginate(5);
        return view("categories.index", compact('categories'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        return view('categories.create');
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $data = $request->validate([
            'code' => 'required|max:10',
            'titre' => 'required|max:100',
        ]);

        $categorie = new Categorie();
        $categorie->code = $request->code;
        $categorie->titre = $request->titre;
        $categorie->save();

        //return back()->with('message', "La catégorie a bien été créée !");
        return redirect('categories')->with('message', "La catégorie a bien été
créée !");
    }

    /**
     * Display the specified resource.
     */
    public function show(string $id)
    {
        $categorie = Categorie::find($id);
        return view('categories.show', compact('categorie'));
    }

    /**
     * Show the form for editing the specified resource.
     */
    public function edit(string $id)
    {
        $categorie = Categorie::find($id);
    }
}

```

```

        return view('categories.edit', compact('categorie'));
    }

    /**
     * Update the specified resource in storage.
     */
    public function update(Request $request, string $id)
    {
        $data = $request->validate([
            'code' => 'required|max:10',
            'titre' => 'required|max:100',
        ]);
        $categorie = Categorie::find($id);

        $categorie->code = $request->code;
        $categorie->titre = $request->titre;
        $categorie->save();

        return redirect('categories')->with('message', "La catégorie a bien été
modifié !");
    }

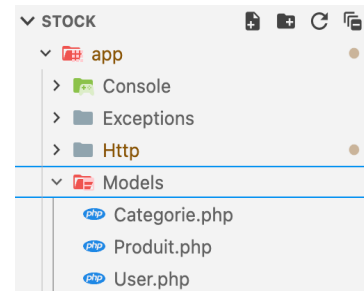
    /**
     * Remove the specified resource from storage.
     */
    public function destroy(string $id)
    {
        Categorie::find($id)->delete();
        return redirect('categories')->with('message', "La catégorie a bien été
supprimée !");
    }
}

```

## 14. Ajout de la classe fille : Produit

### 14.1. Les Models

Dans le dossier app/Models, on complète les classes comme suite pour déterminer la relation entre les deux :



#### a. Categorie.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Categorie extends Model
{
    use HasFactory;

    public function produits()
    {
        return $this->hasMany(Produit::class);
    }
}
```

#### b. Produit.php

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Produit extends Model
{
    use HasFactory;

    public function categorie()
    {
        return $this->belongsTo(Categorie::class);
    }
}
```

## 14.2. La classe ProduitController

Dans app/Http/Controllers, dans le fichier ProduitController.php :

```
<?php

namespace App\Http\Controllers;

use App\Models\Categorie;
use App\Models\Produit;
use Illuminate\Http\Request;

class ProduitController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        $produits = Produit::paginate(5);
        echo $produits;
        return view("produits.index", compact('produits'));
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        $categories = Categorie::all();
        return view('produits.create', compact('categories'));
    }

    /**
     * Store a newly created resource in storage.
     */
    public function store(Request $request)
    {
        $data = $request->validate([
            'reference' => 'required|max:10',
            'libelle' => 'required|max:100',
            'quantite' => 'required',
            'prix' => 'required',
            'categorie_id' => 'required'
        ]);

        $p = new Produit();
        $p->reference = $request->reference;
        $p->libelle = $request->libelle;
        $p->quantite = $request->quantite;
        $p->prix = $request->prix;
        $p->categorie_id = $request->categorie_id;
        $p->save();

        return redirect('produits')->with('message', "Le produit a bien été créé !");
    }

    /**
     * Display the specified resource.
     */
    public function show(string $id)
    {
        $produit = Produit::find($id);
        return view('produits.show', compact('produit'));
    }
}
```

```

/**
 * Show the form for editing the specified resource.
 */
public function edit(string $id)
{
    $produit = Produit::find($id);
    $categories = Categorie::all();
    return view('produits.edit', compact('produit', 'categories'));
}

/**
 * Update the specified resource in storage.
 */
public function update(Request $request, string $id)
{
    $data = $request->validate([
        'reference' => 'required|max:10',
        'libelle' => 'required|max:100',
        'quantite' => 'required',
        'prix' => 'required',
        'categorie_id' => 'required'
    ]);

    $p = Produit::find($id);
    $p->reference = $request->reference;
    $p->libelle = $request->libelle;
    $p->quantite = $request->quantite;
    $p->prix = $request->prix;
    $p->categorie_id = $request->categorie_id;
    $p->save();

    return redirect('produits')->with('message', "Le produit a bien été modifié !");
}

/**
 * Remove the specified resource from storage.
 */
public function destroy(string $id)
{
    Produit::find($id)->delete();

    return redirect('produits')->with('message', "Le produit a bien été supprimé !");
}
}

```

## 14.3. Les routes

Dans le dossier routes, dans le fichier web.php

```
<?php

use App\Http\Controllers\CategorieController;
use App\Http\Controllers\ProduitController;
use App\Http\Controllers\ProfileController;
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

require __DIR__.'/auth.php';

Route::resource('categories', CategorieController::class)->middleware('auth');
Route::resource('produits', ProduitController::class)->middleware('auth');
```

## 14.4. Le component produits-card

Dans le dossier resources/views/components, créer le fichier **produits-card.blade.php**

```
<div class="mt-8 flex flex-col sm:justify-center items-center pt-6 sm:pt-0 bg-gray-100">
    <div class="w-full sm:max-w-md mt-6 px-6 py-4 bg-white shadow-md overflow-hidden sm:rounded-lg">
        {{ $slot }}
    </div>
</div>
```

## 14.5. La vue produits/index

Dans le dossier resources/views, on crée le dossier produits. Dans ce dossier, on crée le fichier index.blade.php

```
<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            @lang('Products List')
        </h2>
    </x-slot>
    <div class="container flex justify-center mx-auto">
        <div class="flex flex-col">

            <div class="w-full">
```



```

@if (session()->has('message'))
  <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
    {{ session('message') }}
  </div>
@endif
<div class="border-b border-gray-200 shadow pt-6">
  <table>
    <thead class="bg-gray-50">
      <tr>
        <td colspan="6">&nbsp;</td>
        <td class="px-4 py-4">
          <x-link-button href="{{ route('produits.create') }}">
            @lang('Create')
          </x-link-button>
        </td>
      </tr>
      <tr>
        <th class="px-2 py-2 text-xs text-gray-500">#</th>
        <th class="px-2 py-2 text-xs text-gray-500">Référence</th>
        <th class="px-2 py-2 text-xs text-gray-500">Libelle</th>
        <th class="px-2 py-2 text-xs text-gray-500">Prix</th>
        <th class="px-2 py-2 text-xs text-gray-500">Catégorie</th>
        <th class="px-2 py-2 text-xs text-gray-500"></th>
        <th class="px-2 py-2 text-xs text-gray-500"></th>
        <th class="px-2 py-2 text-xs text-gray-500"></th>
      </tr>
    </thead>
    <tbody class="bg-white">
      @foreach($produits as $produit)
        <tr class="whitespace-nowrap">
          <td class="px-4 py-4 text-sm text-gray-500">
            {{ $produit->id }}
          </td>
          <td class="px-4 py-4">
            {{ $produit->reference }}
          </td>
          <td class="px-4 py-4">
            {{ $produit->libelle }}
          </td>
          <td class="px-4 py-4">
            {{ $produit->prix }}
          </td>
          <td class="px-4 py-4">
            {{ $produit->categorie->titre }}
          </td>
          <x-link-button href="{{ route('produits.show', $produit->id) }}">
            @lang('Show')
          </x-link-button>
          <x-link-button href="{{ route('produits.edit', $produit->id) }}">
            @lang('Edit')
          </x-link-button>
          <x-link-button onclick="event.preventDefault(); document.getElementById('destroy{{ $produit->id
}}').submit();">
            @lang('Delete')
          </x-link-button>
          <form id="destroy{{ $produit->id }}" action="{{ route('produits.destroy', $produit->id) }}"
method="POST" style="display: none;">
            @csrf
            @method('DELETE')
          </form>
        </tr>
      @endforeach
    </tbody>
  </table>

```

```

        </table>
    </div>
</div>
<div>
    <div class="card-footer is-centered">
        <br>
        {{ $produits->links() }}
    </div>
</div>
</div>
</x-app-layout>

```

## 14.6. Le fichier navigation.blade.php mise à jour

En fonds gris, les lignes à prêter attention.

```

<nav x-data="{ open: false }" class="bg-white border-b border-gray-100">
    <!-- Primary Navigation Menu -->
    <div class="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <div class="flex justify-between h-16">
            <div class="flex">
                <!-- Logo -->
                <div class="shrink-0 flex items-center">
                    <a href="{{ route('dashboard') }}">
                        <x-application-logo class="block h-9 w-auto fill-current text-
gray-800" />
                    </a>
                </div>

                <!-- Navigation Links -->
                <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
                    <x-nav-link :href="route('dashboard')" :active="request()-
>routeIs('dashboard')">
                        {{ __('Dashboard') }}
                    </x-nav-link>
                </div>

                <!-- Lien pour la liste des catégories -->
                <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
                    <x-nav-link :href="route('categories.index')" :active="request()-
>routeIs('categories.index', 'categories.edit', 'categories.show')">
                        {{ __('Categories List') }}
                    </x-nav-link>
                </div>

                <!-- Lien pour la liste des produits -->
                <div class="hidden space-x-8 sm:-my-px sm:ms-10 sm:flex">
                    <x-nav-link :href="route('produits.index')" :active="request()-
>routeIs('produits.index', 'produits.edit', 'produits.show')">
                        {{ __('Products List') }}
                    </x-nav-link>
                </div>
            </div>

            <!-- Settings Dropdown -->
            <div class="hidden sm:flex sm:items-center sm:ms-6">
                <x-dropdown align="right" width="48">
                    <x-slot name="trigger">
                        <button class="inline-flex items-center px-3 py-2 border border-
transparent text-sm leading-4 font-medium rounded-md text-gray-500 bg-white hover:text-
gray-700 focus:outline-none transition ease-in-out duration-150">
                            <div>{{ Auth::user()->name }}</div>

```

```

        <div class="ms-1">
            <svg class="fill-current h-4 w-4"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 20 20">
                <path fill-rule="evenodd" d="M5.293 7.293a1 1 0
011.414 0L10 10.586l3.293-3.293a1 1 0 11.414 1.414l-4 4a1 1 0 01-1.414
1.414z" clip-rule="evenodd" />
            </svg>
        </div>
    </button>
</x-slot>

    <x-slot name="content">
        <x-dropdown-link :href="route('profile.edit')">
            {{ __('Profile') }}
        </x-dropdown-link>

        <!-- Authentication -->
        <form method="POST" action="{{ route('logout') }}">
            @csrf

            <x-dropdown-link :href="route('logout')"
                onclick="event.preventDefault();
                this.closest('form').submit();">
                {{ __('Log Out') }}
            </x-dropdown-link>
        </form>
    </x-slot>
</x-dropdown>
</div>

<!-- Hamburger -->
<div class="me-2 flex items-center sm:hidden">
    <button @click="open = ! open" class="inline-flex items-center justify-
center p-2 rounded-md text-gray-400 hover:text-gray-500 hover:bg-gray-100 focus:outline-
none focus:bg-gray-100 focus:text-gray-500 transition duration-150 ease-in-out">
        <svg class="h-6 w-6" stroke="currentColor" fill="none" viewBox="0 0
24 24">
            <path :class="{ 'hidden': open, 'inline-flex': ! open }"
class="inline-flex" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M4
6h16M4 12h16M4 18h16" />
            <path :class="{ 'hidden': ! open, 'inline-flex': open }"
class="hidden" stroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6
18L18 6M6 6L12 12" />
        </svg>
    </button>
</div>
</div>
</div>

<!-- Responsive Navigation Menu -->
<div :class="{ 'block': open, 'hidden': ! open}" class="hidden sm:hidden">
    <div class="pt-2 pb-3 space-y-1">
        <x-responsive-nav-link :href="route('dashboard')" :active="request()-
>routeIs('dashboard')">
            {{ __('Dashboard') }}
        </x-responsive-nav-link>
    </div>

    <div class="pt-2 pb-3 space-y-1">
        <x-responsive-nav-link :href="route('categories.index')" :active="request()-
>routeIs('categories.index', 'categories.edit', 'categories.show')">
            {{ __('Categories') }}
        </x-responsive-nav-link>
    </div>

```

```

<div class="pt-2 pb-3 space-y-1">
  <x-responsive-nav-link :href="route('produits.index')" :active="request()-
>routeIs('produits.index', 'produits.edit', 'produits.show')">
    {{ __('Produits') }}
  </x-responsive-nav-link>
</div>

<!-- Responsive Settings Options -->
<div class="pt-4 pb-1 border-t border-gray-200">
  <div class="px-4">
    <div class="font-medium text-base text-gray-800">{{ Auth::user()->name
  }}</div>
    <div class="font-medium text-sm text-gray-500">{{ Auth::user()->email
  }}</div>
  </div>

  <div class="mt-3 space-y-1">
    <x-responsive-nav-link :href="route('profile.edit')">
      {{ __('Profile') }}
    </x-responsive-nav-link>

    <!-- Authentication -->
    <form method="POST" action="{{ route('logout') }}">
      @csrf

      <x-responsive-nav-link :href="route('logout')
      onclick="event.preventDefault();
      this.closest('form').submit();">
        {{ __('Log Out') }}
      </x-responsive-nav-link>
    </form>
  </div>
</div>
</div>
</nav>

```

## 14.7. La vue categories/index mise à jour

```

<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      @lang('Categories List')
    </h2>
  </x-slot>
  <div class="container flex justify-center mx-auto">
    <div class="flex flex-col">
      <div class="w-full">
        @if (session()->has('message'))
          <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
            {{ session('message') }}
          </div>
        @endif
        <div class="border-b border-gray-200 shadow pt-6">
          <table>
            <thead class="bg-gray-50">
              <tr>
                <td colspan="4">&nbsp;</td>
                <td class="px-4 py-4">
                  <x-link-button href="{{ route('categories.create') }}">
                    @lang('Create')
                  </x-link-button>
                </td>
              </tr>
              <tr>
                <th class="px-2 py-2 text-xs text-gray-500">#</th>
                <th class="px-2 py-2 text-xs text-gray-500">Code</th>

```

```

<th class="px-2 py-2 text-xs text-gray-500">Titre</th>
<th class="px-2 py-2 text-xs text-gray-500"></th>
<th class="px-2 py-2 text-xs text-gray-500"></th>
<th class="px-2 py-2 text-xs text-gray-500"></th>
</tr>
</thead>
<tbody class="bg-white">
  @foreach($categories as $categorie)
    <tr class="whitespace-nowrap">
      <td class="px-4 py-4 text-sm text-gray-500">
        {{ $categorie->id }}
      </td>
      <td class="px-4 py-4">
        {{ $categorie->code }}
      </td>
      <td class="px-4 py-4">
        {{ $categorie->titre }}
      </td>
      <x-link-button href="{{ route('categories.show', $categorie->id)
        @lang('Show')
      </x-link-button>
      <x-link-button-warning href="{{ route('categories.edit',
        @lang('Edit')
      </x-link-button-warning>
      <x-link-button-danger onclick="event.preventDefault();
document.getElementById('destroy{{ $categorie->id }}').submit();">
        @lang('Delete')
      </x-link-button-danger>
      <form id="destroy{{ $categorie->id }}" action="{{
route('categories.destroy', $categorie->id) }}" method="POST" style="display: none;">
        @csrf
        @method('DELETE')
      </form>
    </tr>
  @endforeach
</tbody>
</table>
</div>
</div>
<div>
  <div class="card-footer is-centered">
    <br>
    {{ $categories->links() }}
  </div>
</div>
</div>
</x-app-layout>

```

## 14.8. La vue produits/create

```
<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __( 'Create a product' ) }}
    </h2>
  </x-slot>

  <x-categories-card>

    <!-- Message de réussite -->
    @if (session()->has( 'message' ))
      <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
        {{ session( 'message' ) }}
      </div>
    @endif

    <form action="{{ route( 'produits.store' ) }}" method="post">
      @csrf
```

```

<!-- Référence -->
<div>
  <x-input-label for="reference" :value="__('Référence')" />
  <x-text-input id="reference" class="block mt-1 w-full" type="text" name="reference"
:value="old('reference')" required autofocus />
  <x-input-error :messages="$errors->get('reference')" class="mt-2" />
</div>

<!-- Libellé -->
<div class="mt-4">
  <x-input-label for="libelle" :value="__('Libellé')" />
  <x-textarea class="block mt-1 w-full" id="libelle" name="libelle">{{ old('libelle')
}}</x-textarea>
  <x-input-error :messages="$errors->get('libelle')" class="mt-2" />
</div>

<!-- Quantité -->
<div>
  <x-input-label for="quantite" :value="__('Quantité')" />

  <x-text-input id="quantite" class="block mt-1 w-full" type="text" name="quantite"
:value="old('quantite')" required autofocus />

  <x-input-error :messages="$errors->get('quantite')" class="mt-2" />
</div>

<!-- Prix -->
<div>
  <x-input-label for="prix" :value="__('Prix')" />

  <x-text-input id="prix" class="block mt-1 w-full" type="text" name="prix"
:value="old('prix')" required autofocus />

  <x-input-error :messages="$errors->get('prix')" class="mt-2" />
</div>

<!-- Catégorie -->
<div>
  <x-input-label for="categorie_id" :value="__('Catégorie')" />
  <x-select class="block mt-1 w-full" name="categorie_id"
:value="old('categorie_id')">
    <option value=""></option>
    @foreach($categories as $categorie)
      <option value="{{ $categorie->id }}"> {{ $categorie->titre }} </option>
    @endforeach
  </x-select>
  <x-input-error :messages="$errors->get('categorie_id')" class="mt-2" />
</div>

<div class="flex items-center justify-end mt-4">
  <x-primary-button class="ml-3">
    {{ __('Send') }}
  </x-primary-button>
</div>
</form>

</x-categories-card>
</x-app-layout>

```

## 14.9. La vue produits/edit

```

<x-app-layout>
  <x-slot name="header">
    <h2 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __( 'Edit a product' ) }}
    </h2>
  </x-slot>

  <x-produits-card>

    <!-- Message de réussite -->
    @if (session()->has('message'))
      <div class="mt-3 mb-4 list-disc list-inside text-sm text-green-600">
        {{ session('message') }}
      </div>
    @endif

    <form action="{{ route('produits.update', $produit->id) }}" method="post">
      @csrf
      @method('put')

      <!-- Référence -->
      <div>
        <x-input-label for="reference" :value="__('Référence')"/>
        <x-text-input id="reference" class="block mt-1 w-full" type="text" name="reference"
:value="old('reference')" value="{{ $produit->reference }}" required autofocus />

        <x-input-error :messages="$errors->get('reference')"/>
      </div>

      <!-- Libellé -->
      <div class="mt-4">
        <x-input-label for="libelle" :value="__('Libellé')"/>
        <x-textarea class="block mt-1 w-full" id="libelle" name="libelle">{{ old('libelle')
}} {{ $produit->libelle }}</x-textarea>

        <x-input-error :messages="$errors->get('libelle')"/>
      </div>

      <!-- Quantité -->
      <div>
        <x-input-label for="quantite" :value="__('Quantité')"/>
        <x-text-input id="quantite" class="block mt-1 w-full" type="text" name="quantite"
:value="old('quantite')" value="{{ $produit->quantite }}" required autofocus />

        <x-input-error :messages="$errors->get('quantite')"/>
      </div>

      <!-- Prix -->
      <div>
        <x-input-label for="prix" :value="__('Prix')"/>
        <x-text-input id="prix" class="block mt-1 w-full" type="text" name="prix"
:value="old('prix')" value="{{ $produit->prix }}" required autofocus />

        <x-input-error :messages="$errors->get('prix')"/>
      </div>

      <!-- Catégorie -->
      <div>
        <x-input-label for="categorie_id" :value="__('Catégorie')"/>
        <x-select class="block mt-1 w-full" name="categorie_id"
:value="old('categorie_id')">
          <option value=""></option>
          @foreach($categories as $categorie)
            <option
              value="{{ $categorie->id }}"
              @if ($produit->categorie_id == $categorie->id)
                selected
              @endif
            >
              {{ $categorie->titre }} </option>
          @endforeach
        </x-select>
      </div>
    </form>
  </x-produits-card>
</x-app-layout>

```

```

        </x-select>

        <x-input-error :messages="$errors->get('categorie_id')" class="mt-2" />
    </div>

    <div class="flex items-center justify-end mt-4">
        <x-primary-button class="ml-3">
            {{ __('Send') }}
        </x-primary-button>
    </div>
</form>

</x-produits-card>
</x-app-layout>

```

## 14.10. La vue produits/show

```

<x-app-layout>
    <x-slot name="header">
        <h2 class="font-semibold text-xl text-gray-800 leading-tight">
            {{ __('Show a product') }}
        </h2>
    </x-slot>

    <x-produits-card>
        <h3 class="font-semibold text-xl text-gray-800">@lang('Référence')</h3>
        <p> {{ $produit->reference }} </p>

        <h3 class="font-semibold text-xl text-gray-800">@lang('Libelle')</h3>
        <p> {{ $produit->libelle }} </p>

        <h3 class="font-semibold text-xl text-gray-800">@lang('Quantité')</h3>
        <p> {{ $produit->quantite }} </p>

        <h3 class="font-semibold text-xl text-gray-800">@lang('Prix')</h3>
        <p> {{ $produit->prix }} </p>

        <h3 class="font-semibold text-xl text-gray-800">@lang('Catégorie')</h3>
        <p> {{ $produit->categorie->titre }} </p>

        <h3 class="font-semibold text-xl text-gray-800" pt-2>
            @lang('Date creation')
        </h3>
        <p> {{ $produit->created_at->format('d/m/Y') }} </p>
        @if ($produit->created_at != $produit->updated_at)
            <h3 class="font-semibold text-xl text-gray-800" pt-2>
                @lang('Last update')
            </h3>
            <p> {{ $produit->updated_at->format('d/m/Y') }} </p>
        @endif

    </x-produits-card>
</x-app-layout>

```

## 14.11. Le fichier lang/fr.json

```

...
"Create a category": "Créer une catégorie",
"Edit a category": "Modifier une catégorie",
"Last update": "Dernière modification",
"Date creation": "Date de création",
"Show a category": "Voir les détails d'une catégorie",
"Categories List": "Liste des catégories",
"Products List": "Liste des produits",
"Edit a product": "Modifier un produit",
"Show a product": "Voir les détails d'un produit"
}

```



## 15. Dépôt git public

<https://github.com/tresorkabis/stock.git>

## Table des matières

<b>1. LES PRÉREQUIS.....</b>	<b>1</b>
<b>2. CRÉATION DU PROJET .....</b>	<b>1</b>
<b>3. BASE DE DONNÉES .....</b>	<b>2</b>
<b>4. L'AUTHENTIFICATION .....</b>	<b>6</b>
<b>5. LES LANGUES .....</b>	<b>10</b>
<b>6. LES DONNÉES .....</b>	<b>12</b>
<b>7. LE CONTRÔLEUR ET LES ROUTES.....</b>	<b>16</b>
<b>8. ORGANISATION DES VUES.....</b>	<b>18</b>
<b>9. CRÉATION D'UNE CATÉGORIE.....</b>	<b>21</b>
9.1. LE FORMULAIRE .....	21
9.2. LA SOUMISSION .....	24
<b>10. MODIFICATION D'UNE CATÉGORIE.....</b>	<b>26</b>
10.1. LE FORMULAIRE .....	26
10.2. LA SOUMISSION .....	27
<b>11. VOIR UNE CATÉGORIE .....</b>	<b>28</b>
<b>12. LISTE DES CATÉGORIES .....</b>	<b>29</b>
<b>13. LA CLASSE CATEGORIECONTROLLER .....</b>	<b>34</b>
<b>14. AJOUT DE LA CLASSE FILLE : PRODUIT.....</b>	<b>36</b>
14.1. LES MODELS.....	36
a. <i>Categorie.php</i> .....	36
b. <i>Produit.php</i> .....	36
14.2. LA CLASSE PRODUITCONTROLLER .....	37
14.3. LES ROUTES .....	39
14.4. LE COMPONENT PRODUITS-CARD .....	39
14.5. LA VUE PRODUITS/INDEX .....	39
14.6. LE FICHIER NAVIGATION.BLADE.PHP MISE À JOUR.....	41
14.7. LA VUE CATEGORIES/INDEX MISE À JOUR.....	43
14.8. LA VUE PRODUITS/CREATE .....	44
14.9. LA VUE PRODUITS/EDIT .....	46
14.10. LA VUE PRODUITS/SHOW.....	47
<b>15. DÉPÔT GIT PUBLIC.....</b>	<b>48</b>
<b>TABLE DES MATIÈRES.....</b>	<b>49</b>