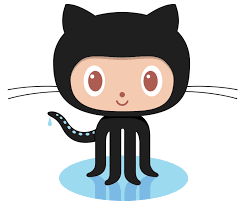
04/04/2022

Réalisé par :

FOUAD el metioui

Application web 1 &2

Atelier2



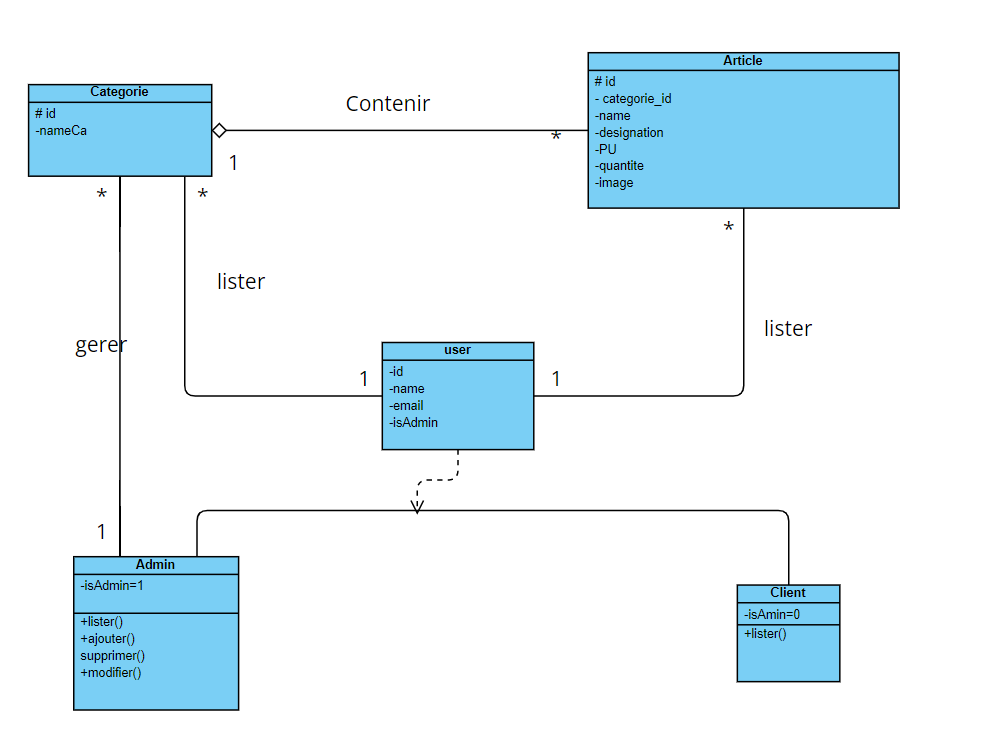
<https://github.com/FouadElMetioui/application-web>

## Introduction

l’objectif principal de cet atelier est la mise en place d’une application web sous frome d’une vitrine virtuelle online, en utilisant le Framework Laravel, l’application composée de deux parties, une première « back end » qui va servir pour la gestion interne du site, et une deuxième partie « Front end » sous forme d’une vitrine pour proposer les articles online.

## Réalisation

1. Développer le digramme de classe équivalant à cette application web.

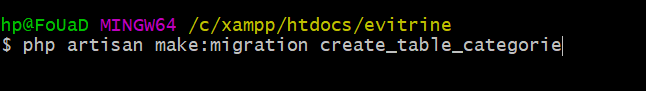


1. Créer un projet Laravel « evitrine » en utilisant la ligne de commande équivalente

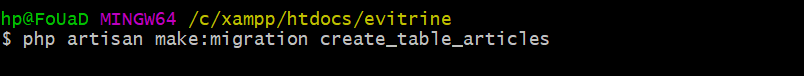


1. Créer les fichiers migrations selon les classes du diagramme de classe déjà établie.

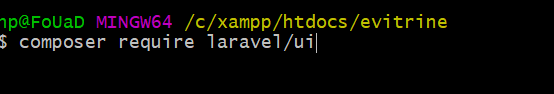
* **La table Catégorie**

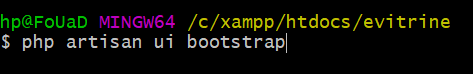
****

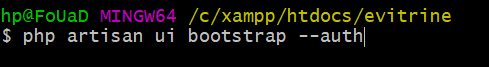
* **La table Article**

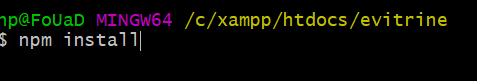
****

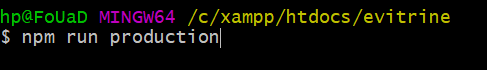
* **Système d’authentification**

****

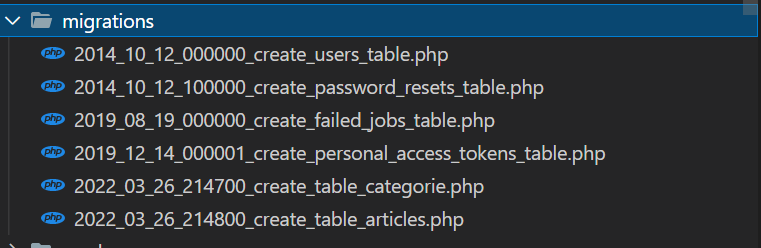
****

****

****

****

**Résultat**

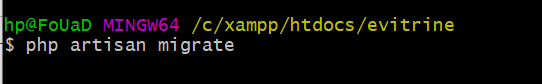


1. Taper la commande capable de lancer une migration vers une BDD Mysql , la commande est

basée sur l’outil Artisan, n’oublier pas de paramétrer le fichier .env ainsi de créer une BDD

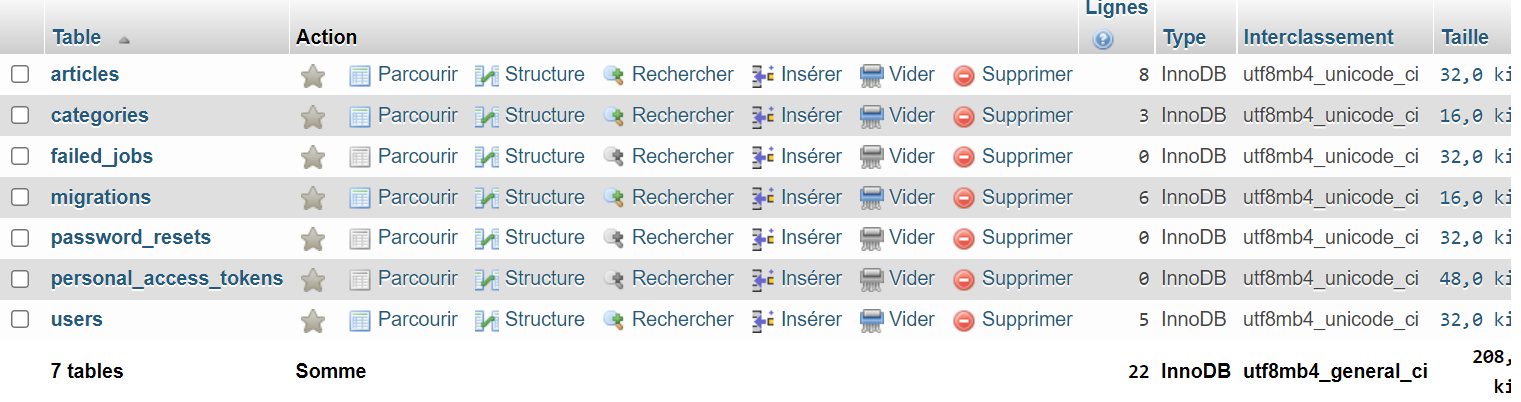
« evitrine » au niveau de la SGBD Mysql.

* **Pour lancer une migration vers BDD**

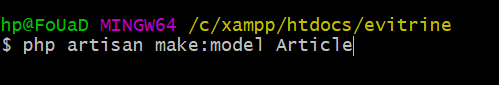
****

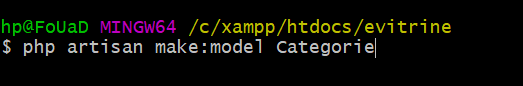
* **Parametrer le fichier .env**
  + DB\_CONNECTION=mysql
  + DB\_HOST=127.0.0.1
  + DB\_PORT=3306
  + DB\_DATABASE=evitrine
  + DB\_USERNAME=root
  + DB\_PASSWORD=

**Résultat**

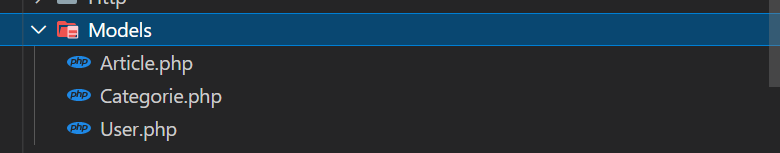
****

1. Créer les classes modèles en se basant sur le digramme de classe.





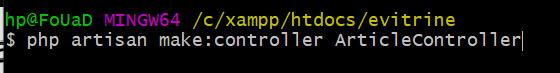
Résultat

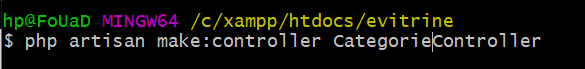


1. Créer les Contrôleurs de chaque sous gestions de l’application, pour les deux parties back end et

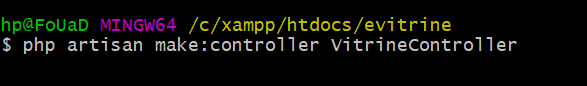
front end.

* Partie back end

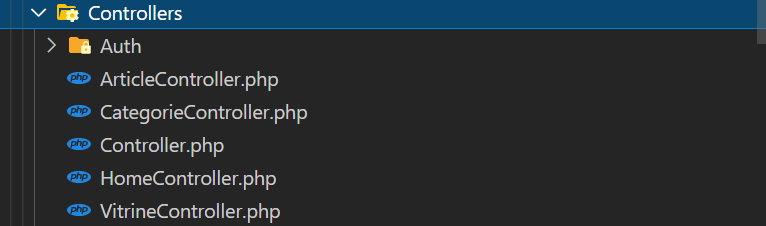




* Partie front end



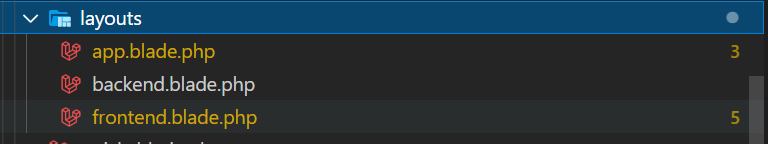
**Résultat**

****

1. Créer deux Templates « backend.blade.php, frontend.blade.php » basées sur le moteur Blade,

une pour la partie back end et autre pour front end.

On va les créer dans le fichier Views

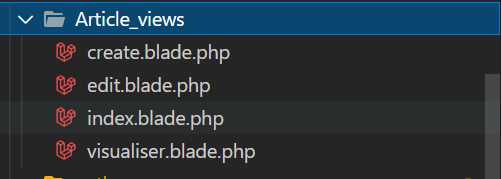


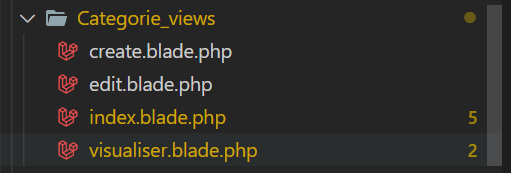
1. Créer les interfaces web qui héritent du Template « backend.blade.php », et qui représentent

l’opération CRUD liées à chaque sous gestion de la partie back end.

pour cette partie j’ai développé quatre views pour chaque sous gestion (article, catégorie)

voici la gestion des fichier pour le CRUD

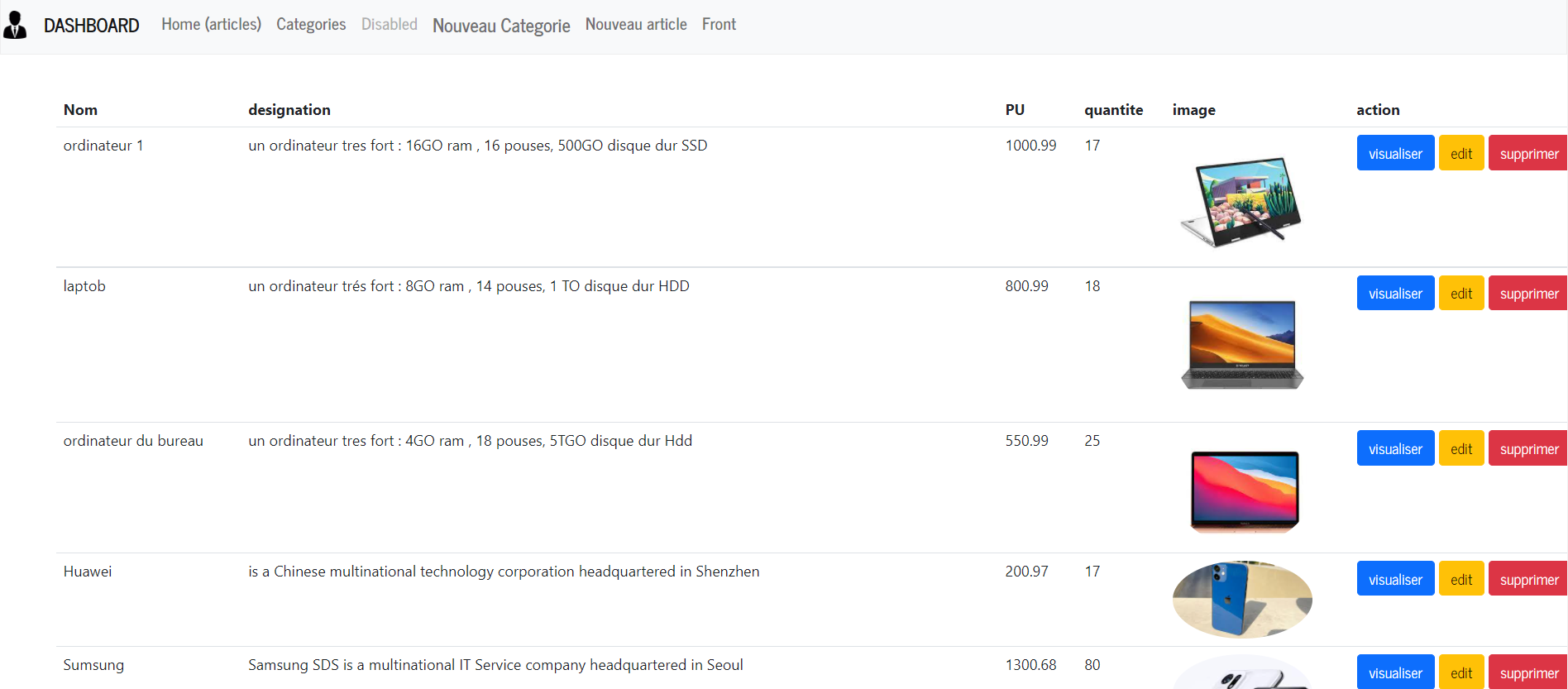




**La page principale du partie back end**

Pour afficher le vitrine

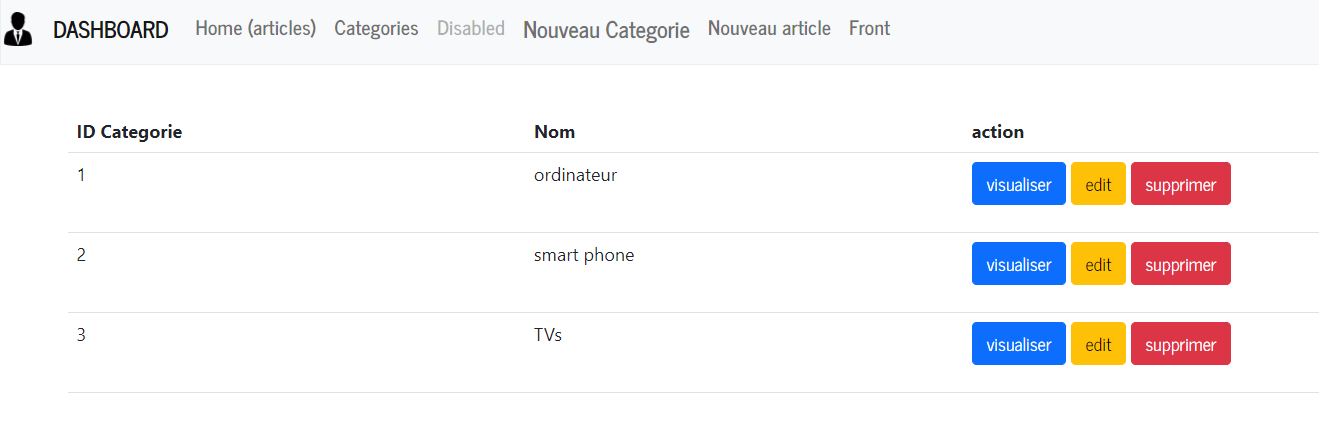
Lien vers mêmepageafficher la liste des catégories

****

Cette page contient un menu basé sur le layout backend.php

* **Gestion des categories**

Lorsque on clique sur le lien catégories on aura la page suivante

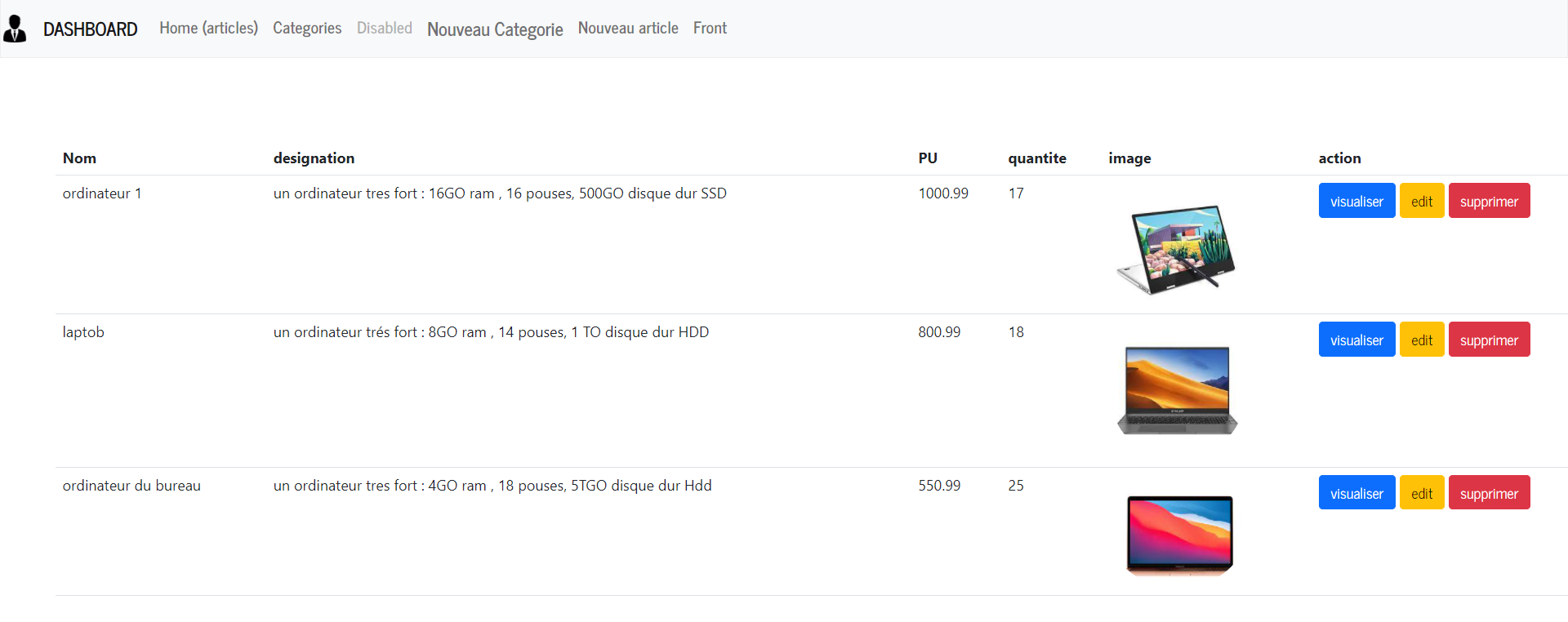


pour chaque catégorie on a les éléments principale du CRUD telque :

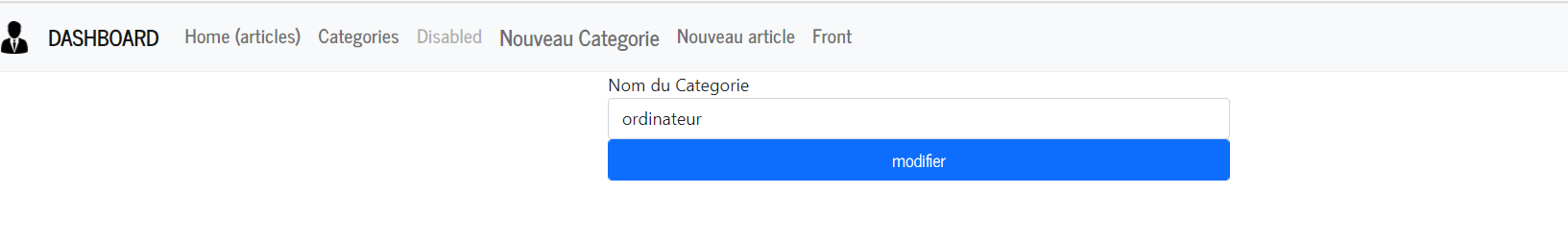
* Visualiser

Pour la premier visualiser s’intéressent la catégorie des ordinateurs

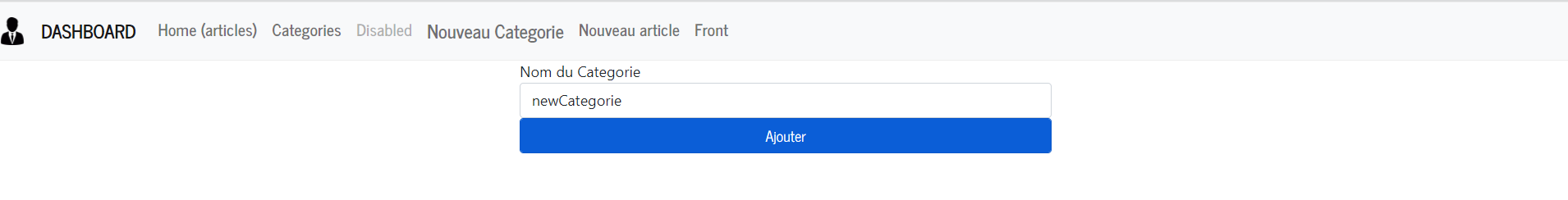
On voit que les article dont le clés étranger est égale primary key du catégorie



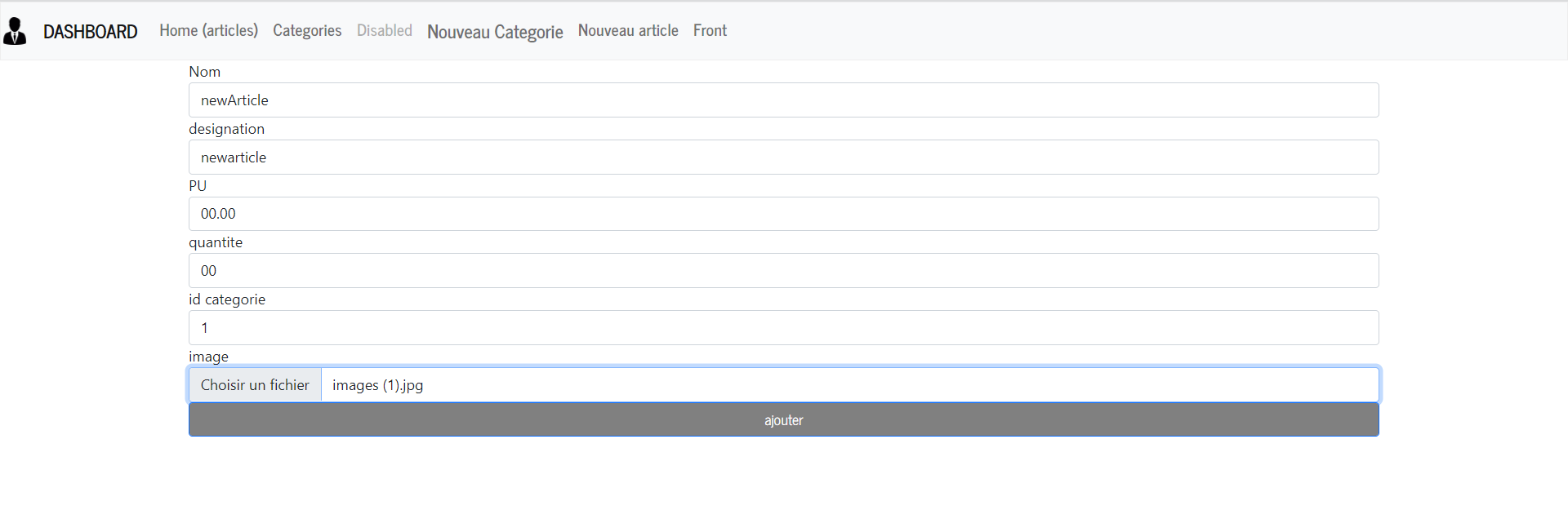
* Edit



* Pour supprimer permet la suppression d’une catégorie de la base de données
* Pour ajouter une catégorie Nouveau catégorie dans le menu de tableau de bord



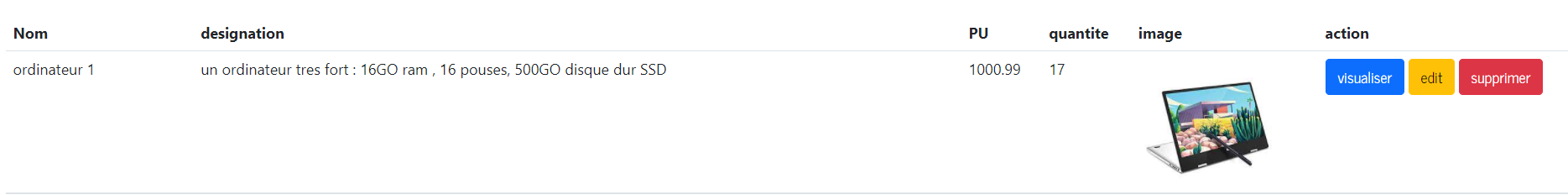
* **Gestion des articles**
* Pour ajouter une article *Nouveau article*

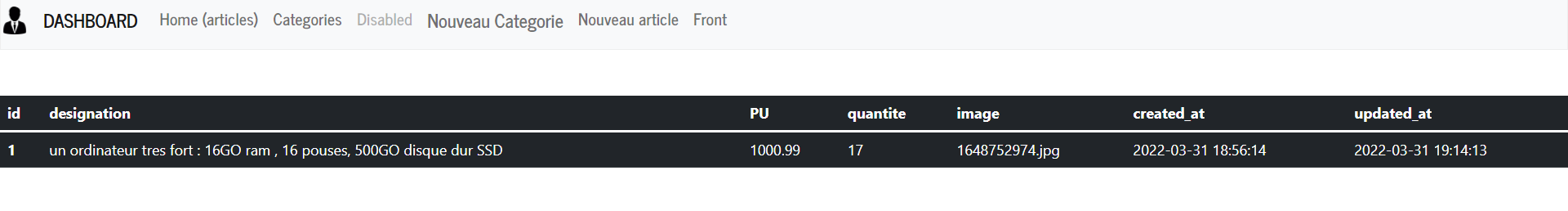


id du catégorie dont lequel appartenir cette article

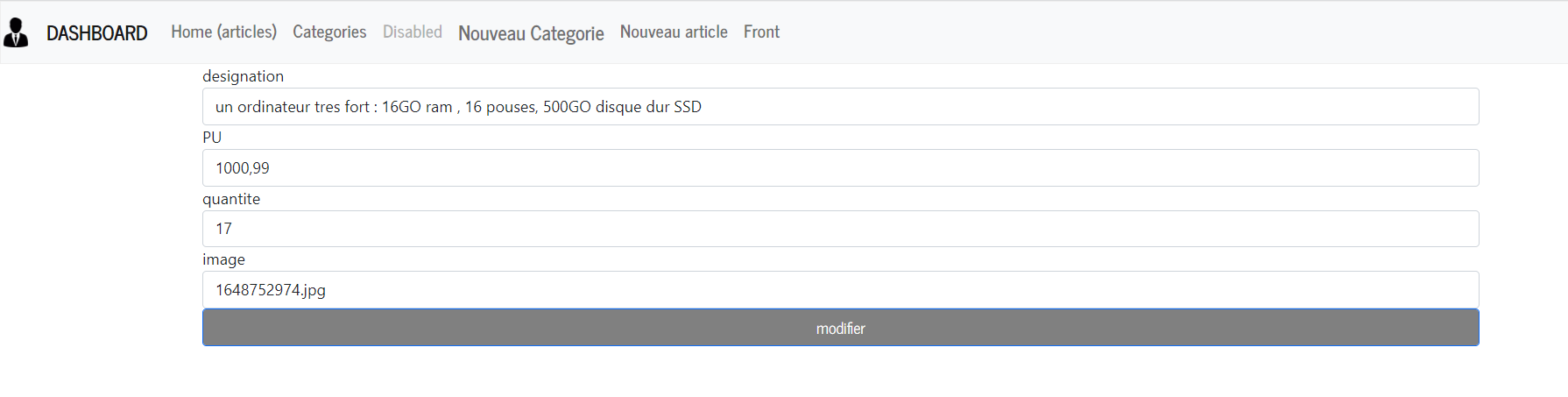
Visualiser catégorie permet d’afficher les détails concernant l’article courant

Soit cette article





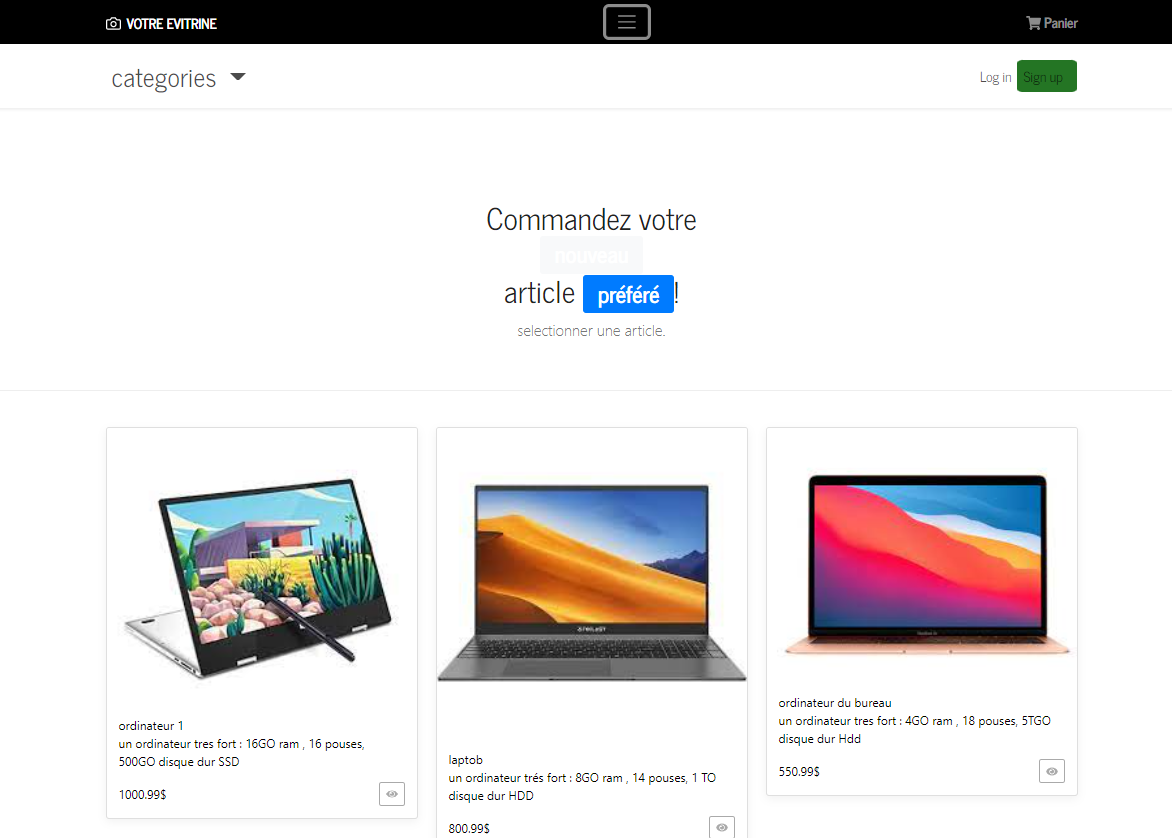
Edit



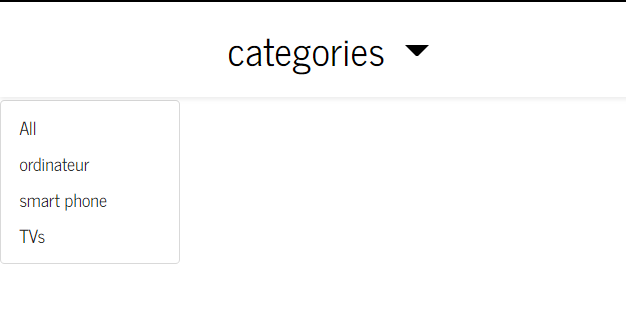
1. Développer une interface principale de la vitrine basée sur le fichier Template « frontend.blade.php » , pour afficher les articles selon les catégories et avec un système de

Pagination.

**La page principale du partie back end**



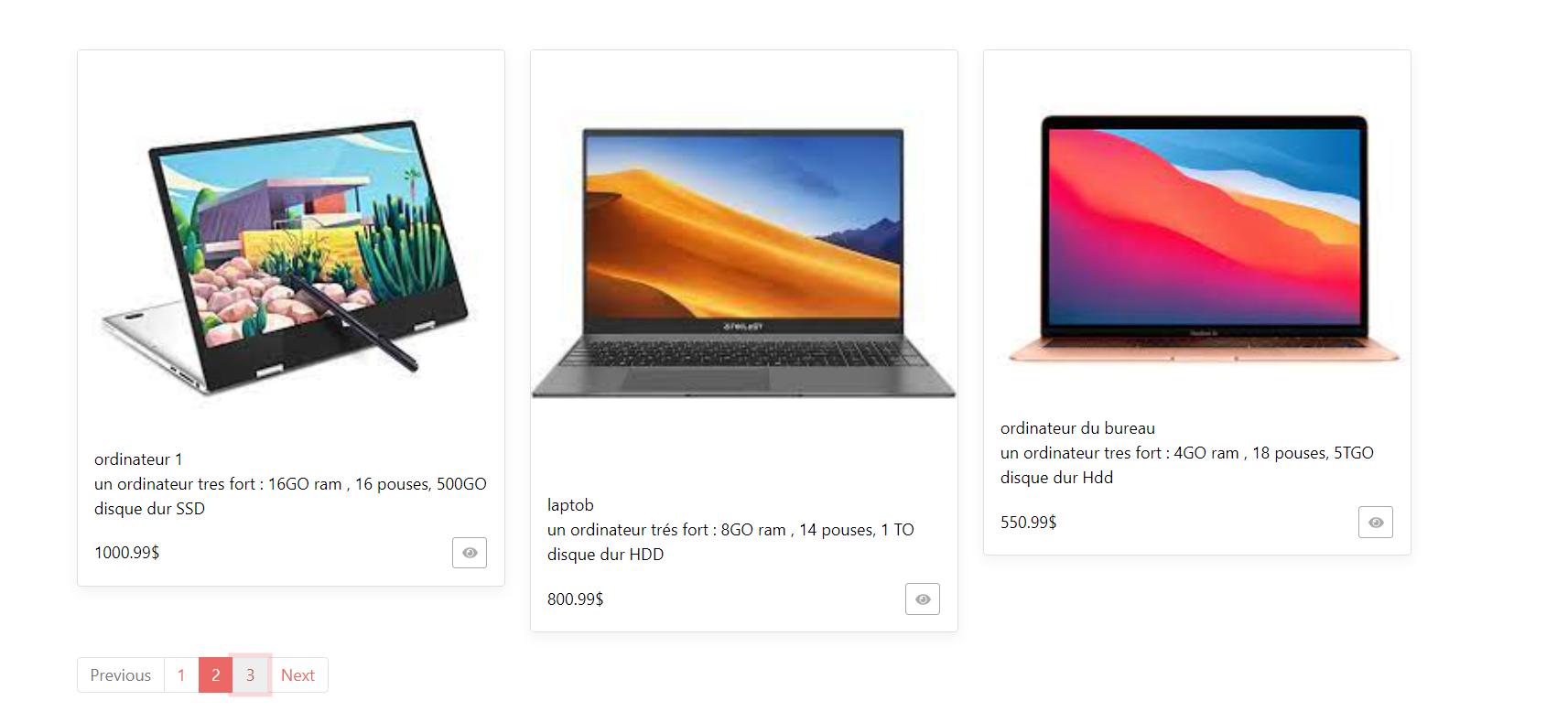
Consulter les catégories



La premier lien qu’on vois c’est All qui va nous diriger vers tous les articles (peu import la catégorie)

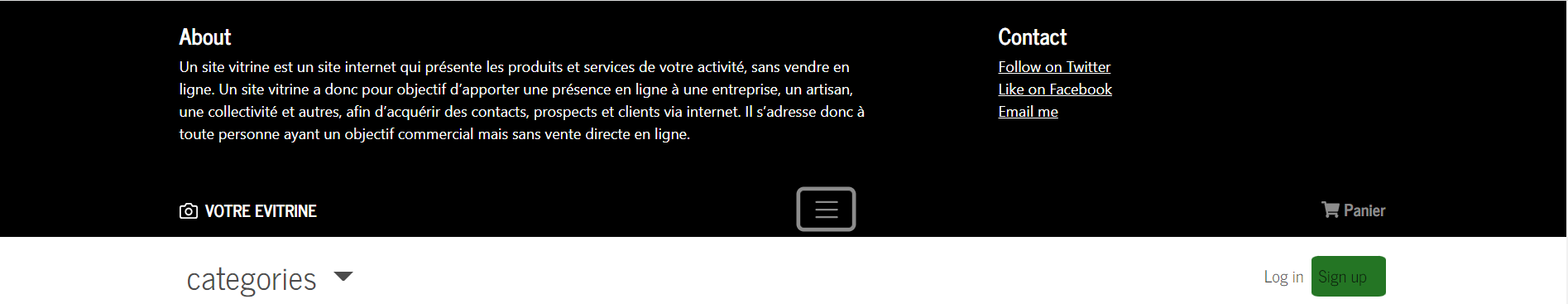
🡪 pour les autres lien ce sont des catégories qu’on a créer au niveau de tableau du bord

* Par exemple ordinateur permet de filtrer les articles pour laisser que les articles de la catégorie ordinateur, voici le résultat du clic sur ordinateur

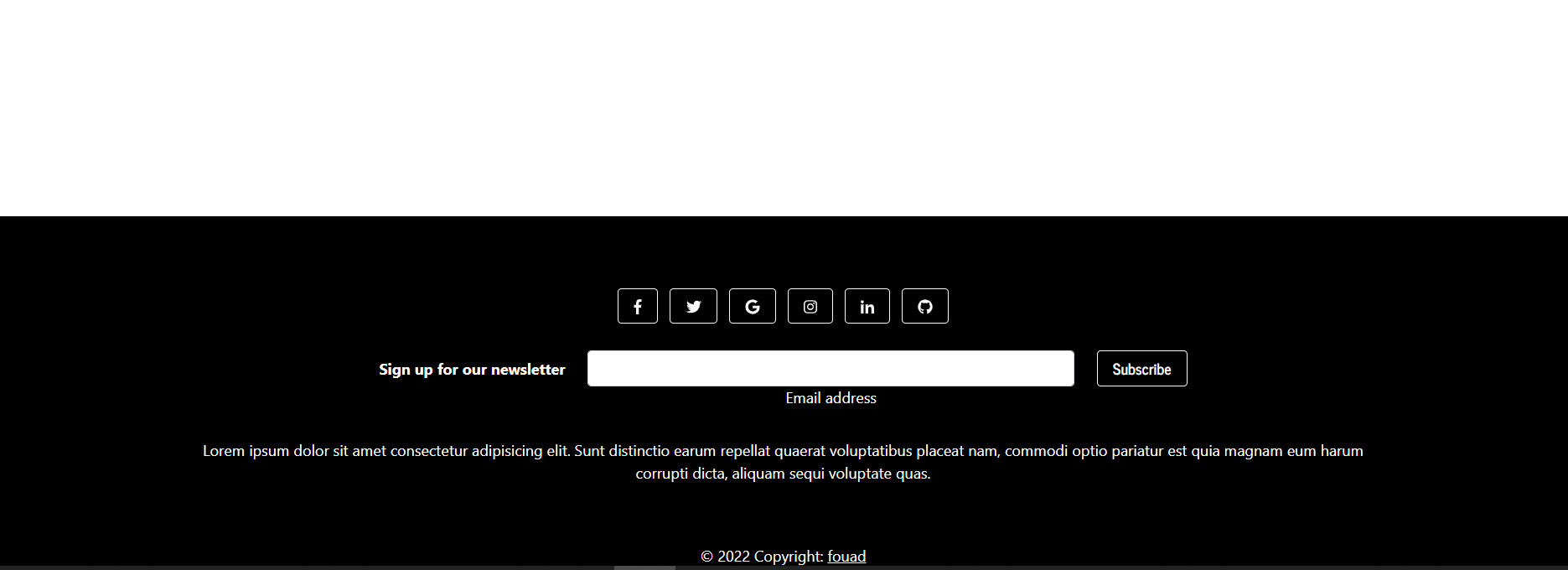


Système du pagination

# Header

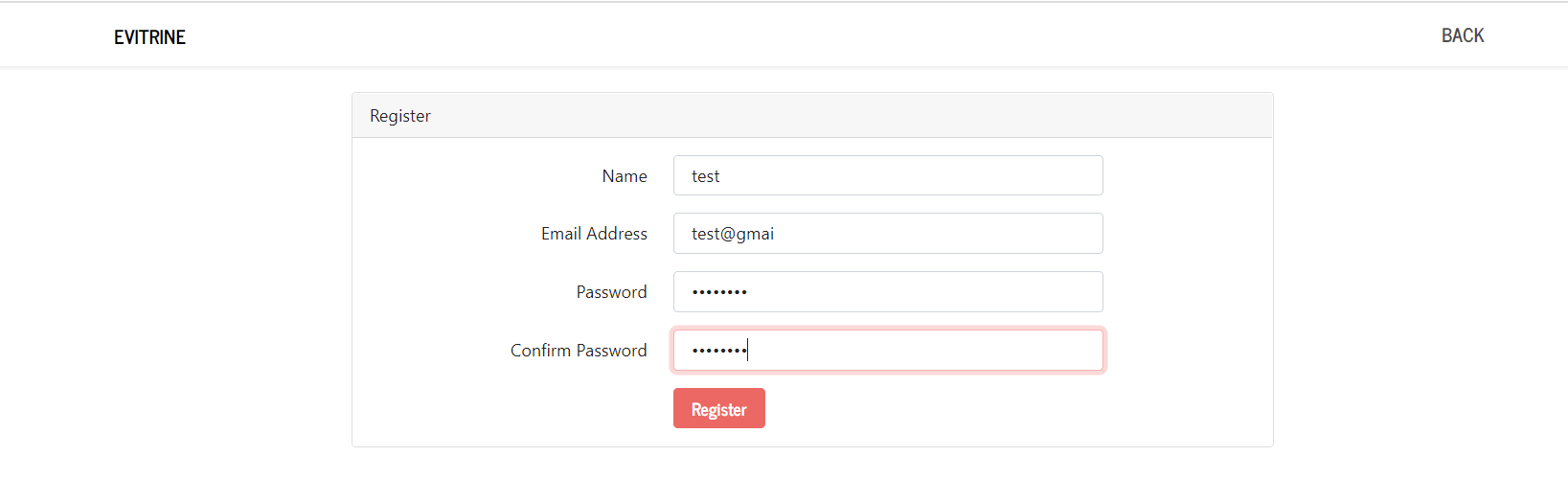


# Footer



Comment s’identifier ?

On clique sur le bouton Sign up pour un nouveau compte

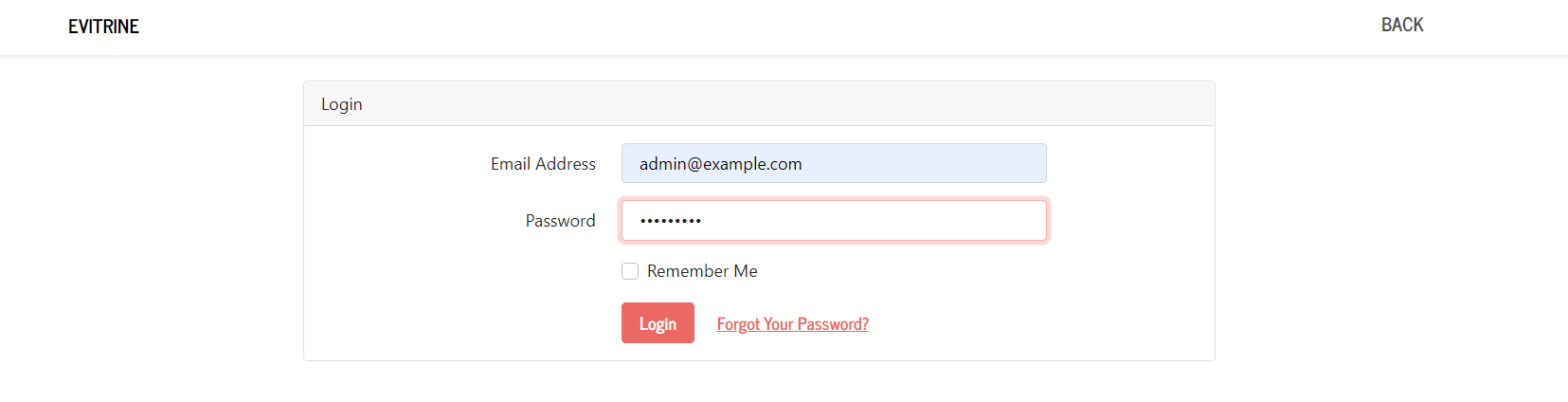




Nom du compte est test

On clique sur Log in si on a déjà un compte au niveau de BDD

Dans cette exemple on va essayer de s’identifier avec le compte admin dont lequel le champ isAdmin = 1 , on va remarquer qu’après l’identification qu’on va se diriger vers la page /admin( Dashboard)





* Si on fait maintenant la loupe sur une article on va voire



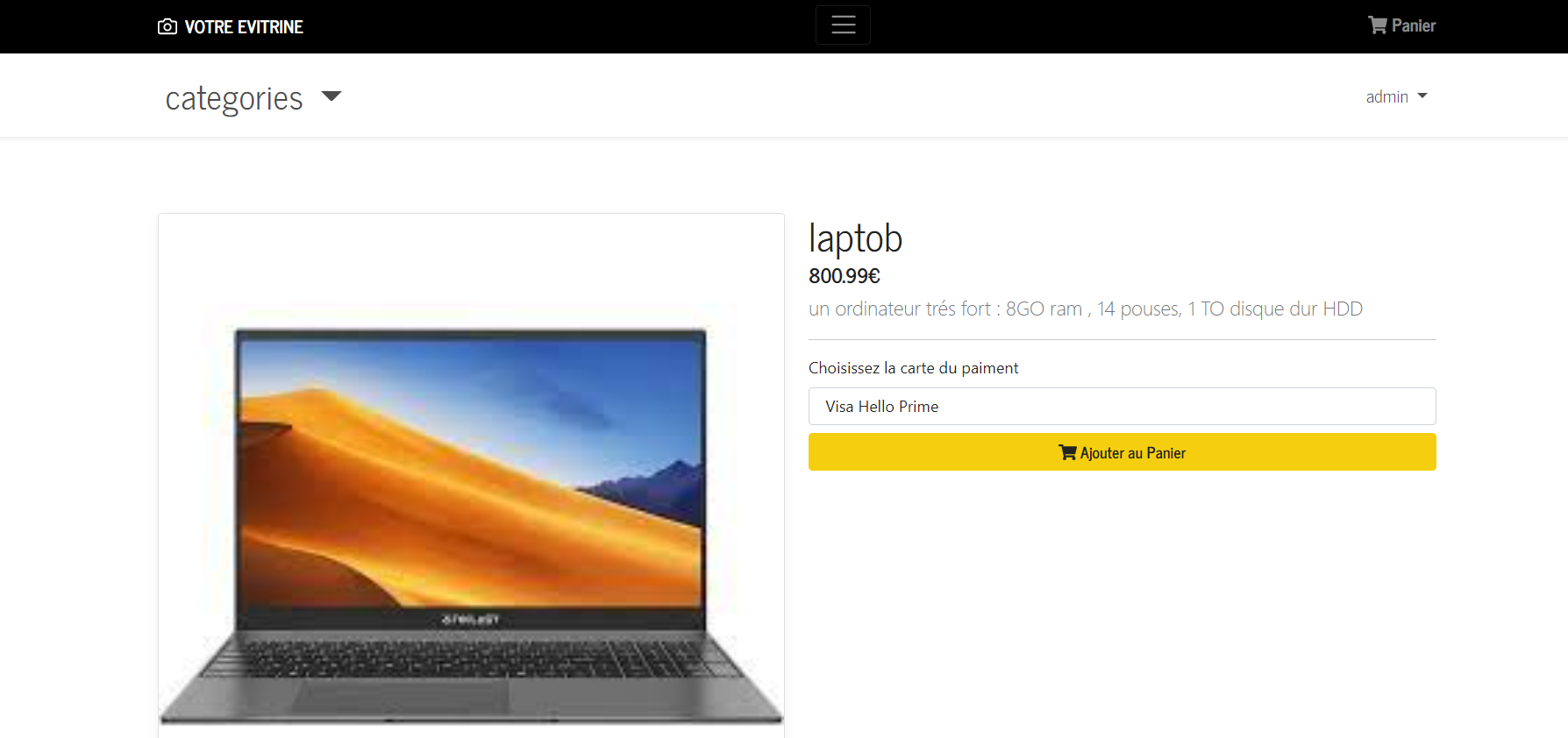
Le prix PU de l’article de l’article

Designation de l’article

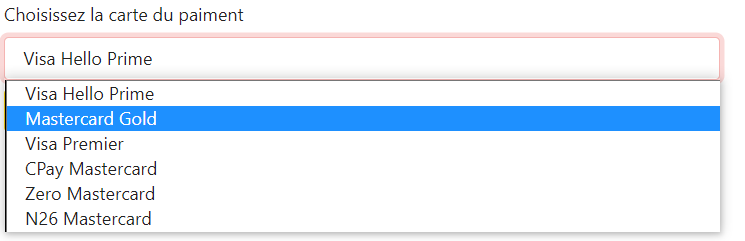
Nom de l’article

Pour voire les détails de l’article + le paiement en ligne

Voici le résultat de la clique sur une Yeu d’un article

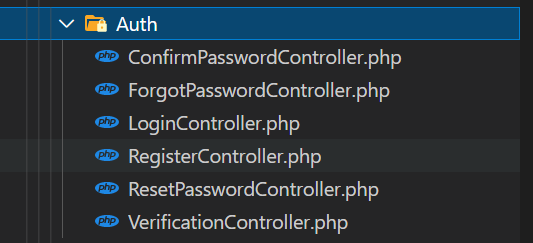


* Les types du carte de paiement

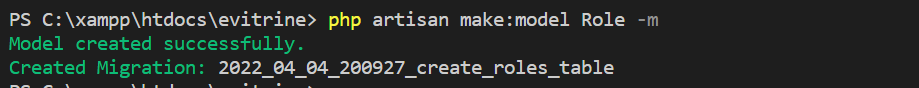


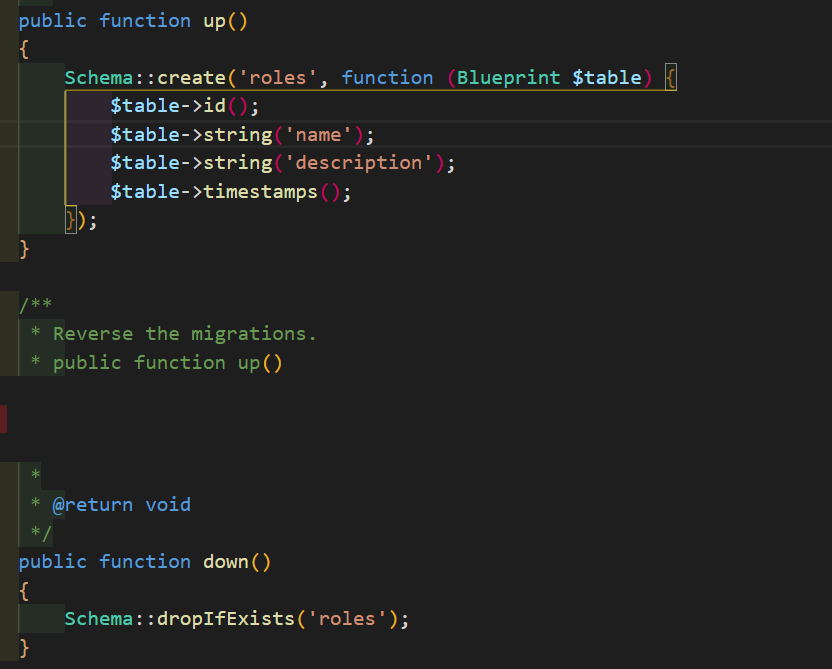
# Atelier 2 : Laravel et Système d’authentification

1. Création des ressources d’authentification via l’outil Artisan



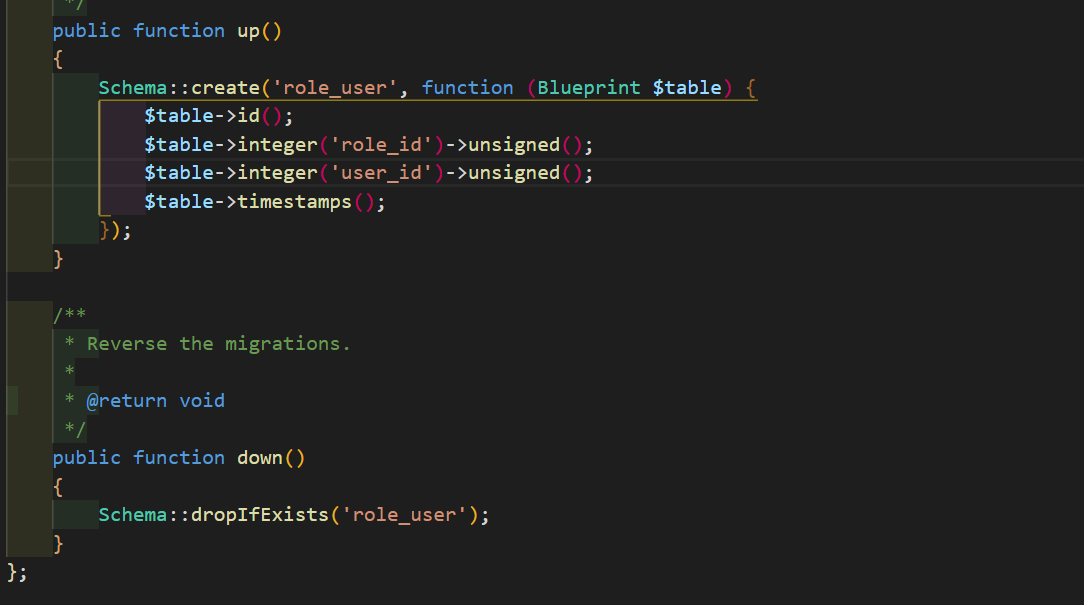
1. Création de la class model et de fichier migration Rôle :





1. Création d’une migration role\_user « une table intérimaire entre role et user »





1. Modification des classes du model « User et Role » en ajoutant une relation Many-To-Many .

User :

public function roles()

    {

    return $this->belongsToMany(Role::class);

    }

Role :

 public function users()

    {

    return $this->belongsToMany(User::class);

    }

1. Création des seeders “ jeu de teste pour les tables users et roles ”





public function run()

    {

        $role\_employee = new Role();

        $role\_employee->name = 'internaute';

        $role\_employee->description = 'A internaute User';

        $role\_employee->save();

        $role\_manager = new Role();

        $role\_manager->name = 'admin';

        $role\_manager->description = 'A Admin User';

        $role\_manager->save();

   }

 public function run()

    {

        $role\_agent = Role::where('name', 'internaute' )->first();

        $role\_admin = Role::where('name', 'admin')->first();

        $employee = new User();

        $employee->name = 'Internaute Name';

        $employee->email = 'employee@example.com';

        $employee->password = bcrypt('secret');

        $employee->save();

        $employee->roles()->attach($role\_agent);

        $manager = new User();

        $manager->name = 'Admin Name';

        $manager->email = 'manager@example.com';

        $manager->password = bcrypt('secret');

        $manager->save();

        $manager->roles()->attach($role\_admin);

    }

public function run()

    {

        // \App\Models\User::factory(10)->create();

        // Role comes before User seeder here.

        $this->call(RoleTableSeeder::class);

        // User seeder will use the roles above created.

        $this->call(UserTableSeeder::class);

    }

1. Modification de la class User model en ajoutant ce code de 3 méthode nécessaire pour

l’authentification

 public function roles()

    {

       return $this->belongsToMany(Role::class);

    }

    /\*\*

    \* @param string|array $roles

    \*/

    public function authorizeRoles($roles)

    {

    if (is\_array($roles)) {

        return $this->hasAnyRole($roles) ||

        abort(401, 'This action is unauthorized.');

    }

    return $this->hasRole($roles) ||

    abort(401, 'This action is unauthorized.');

    }

    /\*\*

    \* Check multiple roles

    \* @param array $roles

    \*/

    public function hasAnyRole($roles)

    {

      return null !== $this->roles()->whereIn('name', $roles)->first();

    }

    /\*\*

    \* Check one role

    \* @param string $role

    \*/

    public function hasRole($role)

    {

        return null !== $this->roles()->where('name', $role)->first();

    }

1. Modifier la méthode create() du contrôleur RegisterController.php dans le dossier

“app/Http/Controllers/Auth/ »

 $user = User::create([

        'name' => $data['name'],

        'email' => $data['email'],

        'password' => bcrypt($data['password']),

        ]);

        $user

        ->roles()

        ->attach(Role::where('name', ‘internaute’)->first());

        return $user;

1. Dans l’un des contrôleurs « Home par exemple », ajouter le code suivant

class HomeController extends Controller

        {

        public function \_\_construct()

        {

        $this->middleware('auth');

        }

        public function index(Request $request)

        {

        $request->user()->authorizeRoles([‘internaute ’, 'admin']);

        return view(‘home’);

        }

        /\* function qui didié pour les admin

        public function someAdminStuff(Request $request)

        {

        $request->user()->authorizeRoles('admin');

        // rmaplacer par une vue selon votre projet

        return view(‘some.view’);

        }

        \*/

    }