|  |  |
| --- | --- |
|  | Année Universitaire : 2023-2024  **Applications Web Avancées** |
| **Atelier 5 : Authentification avec Laravel Breeze** | |

**Objectifs**

Le but de cet atelier est d’installer, configurer, personnaliser le package Breeze pour créer des fonctionnalités d'authentification dans Laravel.

**Prérequis logiciel**

* Projet Laravel créé.
* Création de la base de données
* Gestion du CRUD d’un modèle
  1. **Installation par défaut**

1. Créer un nouveau projet Laravel.
2. Créer la base de données « authentification »

**Remarque :** Laravel Breeze donne des fonctionnalités backend pour gérer l'authentification et aussi des vues. Ainsi, nous aurons besoin d'utiliser nos 2 gestionnaires de paquets backend et frontend favoris : composer et npm.

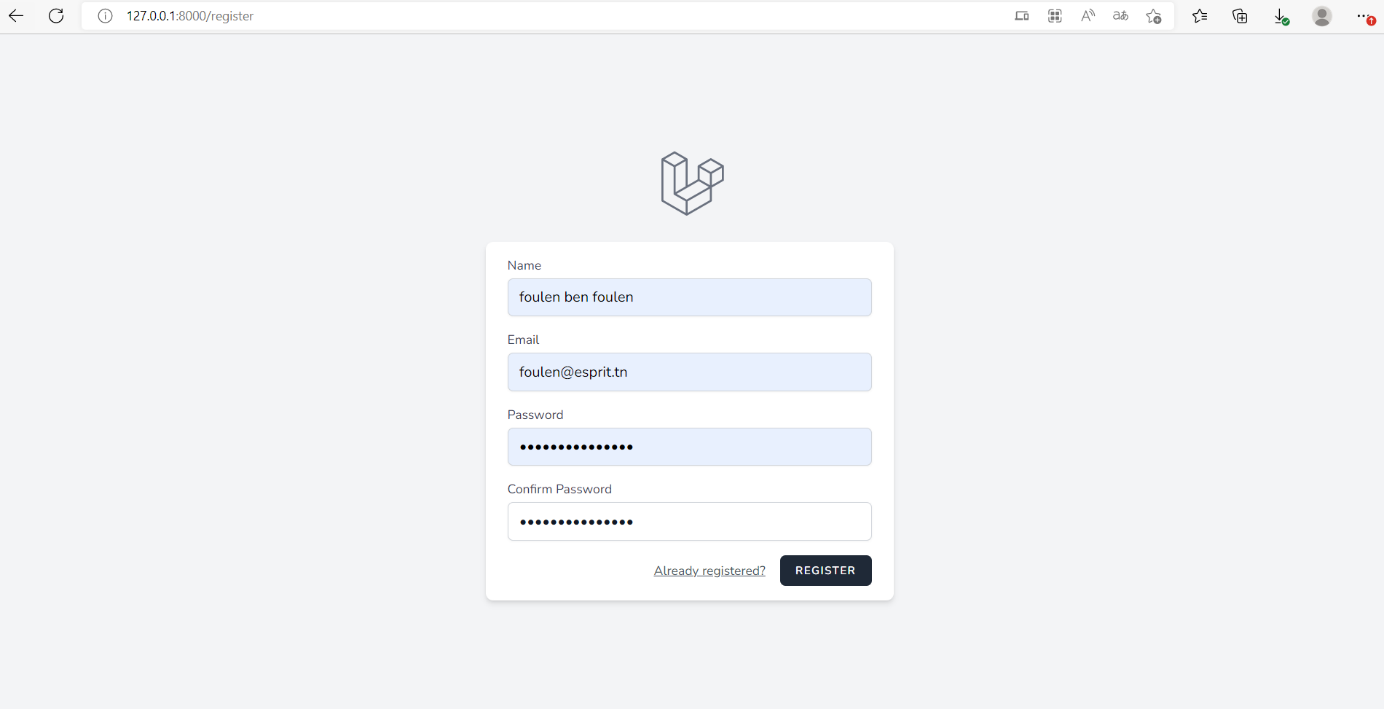
1. Lancer les migrations.
2. Installer le Laravel Breeze dans un projet laravel en tapant la commande suivante :

**composer require laravel/breeze –dev**

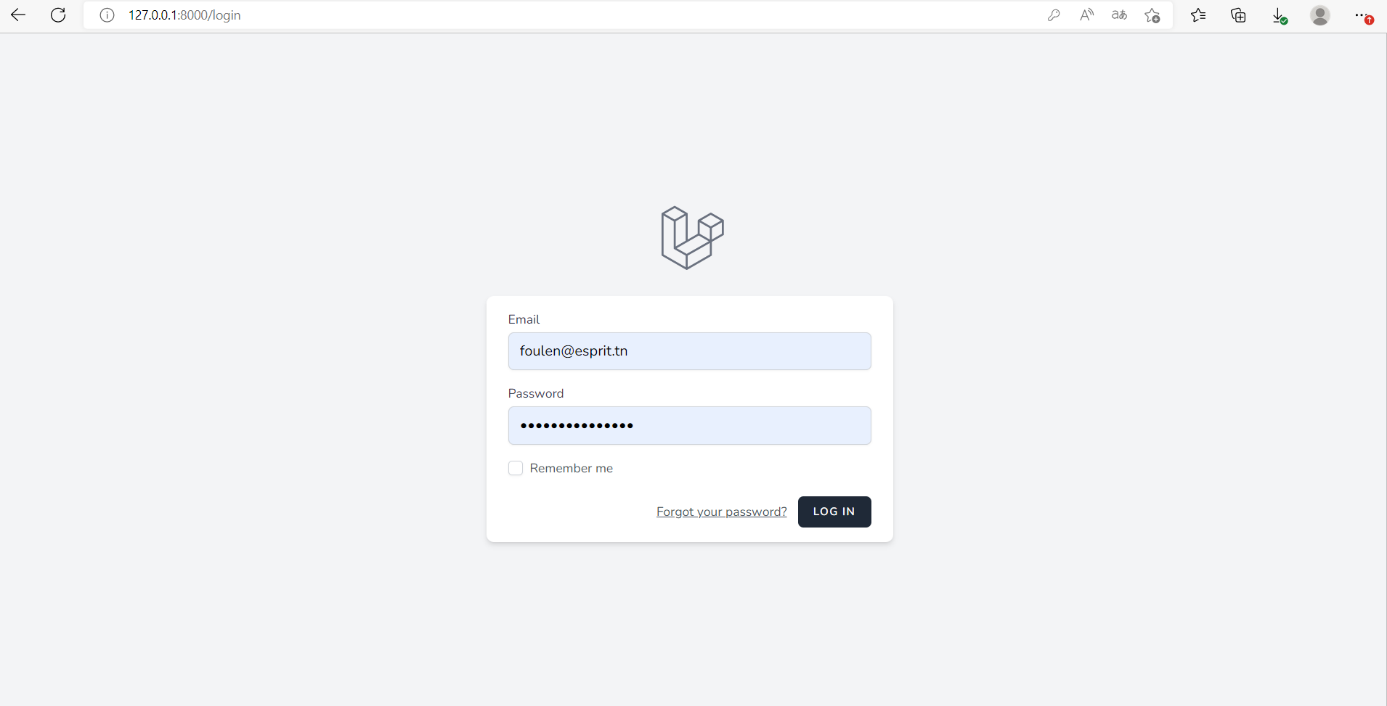
1. Une fois laravel/breeze installé avec composer, on l'implémente dans l'application en exécutant la commande artisan suivante :

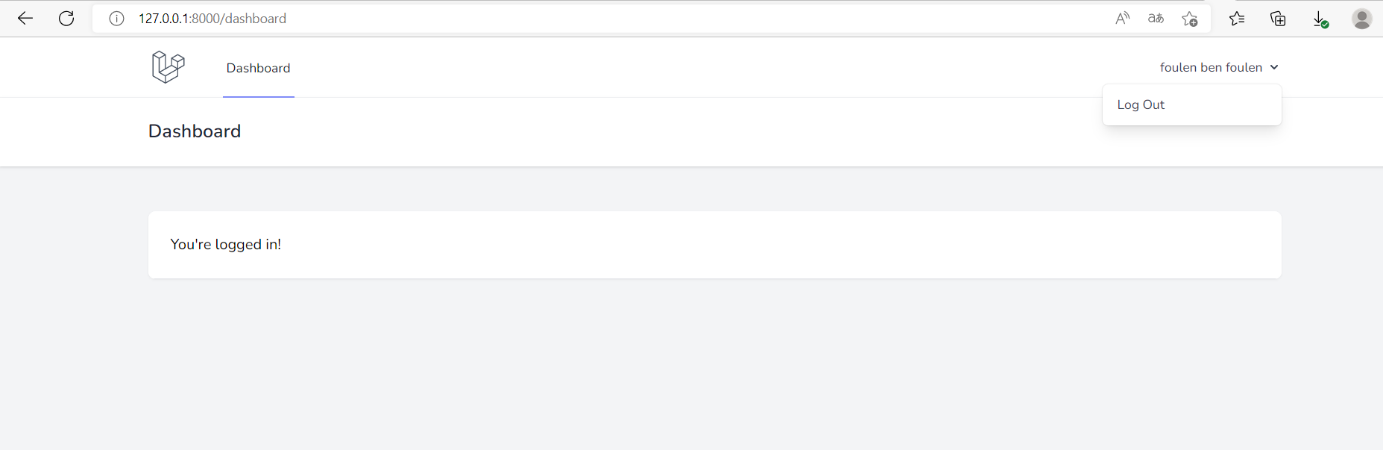
**php artisan breeze:install**

Cette commande installe les contrôleurs, les vues, les routes et d'autres ressources de Breeze pour l'authentification des utilisateurs comme illustré dans la figure ci-dessous :

1. Installer les dépendances : **npm install & npm run dev**
2. Lister les routes
3. Afficher la page d’accueil.
4. Créer un nouvel utilisateur. Puis connectez-vous et consulter la partie profile et dashboard comme illustré dans la figure ci-dessous :

**Figure 1: Page d’inscription d’un nouvel utilisateur**

**Figure 2: Page de connexion**

**Figure 3: Page dashboard d'un utilisateur connecté**

* 1. **Personnalisation de l’inscription :**

On va maintenant ajouter deux champs dans le formulaire d’inscription : username et phone.

1. Mettre à jour le fichier de migration de la table User

**public function** up()  
{…

$table->string(**'username'**)->**unique**();  
$table->string(**'phone'**)->**unique**();

..}

1. Mettre à jour la base de données
2. Mettre à jour le modèle User

**protected $fillable** = [  
 **'name'**,  
 **'username'**,  
 **'phone'**,  
 **'email'**,  
 **'password'**,  
];

1. Mettre à jour la vue **ressources/views/auth/register.blade.php**

<*<!-- UserName -->*<div>  
 <x-input-label for="username" :value="\_\_('Username')" />  
  
 <x-text-input id="username" class="block mt-1 w-full" type="text" name="username" :value="old('username')" required autofocus />  
  
 <x-input-error :messages="$errors->get('username')" class="mt-2" />  
</div>  
*<!-- phone Number -->*<div>  
 <x-input-label for="phone" :value="\_\_('Phonenumber')" />  
  
 <x-text-input id="phone" class="block mt-1 w-full" type="text" name="phone" :value="old('phonenumber')" required autofocus />  
  
 <x-input-error :messages="$errors->get('phone')" class="mt-2" />  
</div>

1. Mettre à jour le fichier **App/http/Controller/RegisteredUserController.php**

public function store(Request $request)  
{  
 $request->validate([  
 'name' => ['required', 'string', 'max:255'],  
 'username' => ['required', 'string', 'max:255','unique:users'],  
 'phone' => ['required','string','max:255'],  
 'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],  
 'password' => ['required', 'confirmed', Rules\Password::*defaults*()],  
 ]);  
  
 $user = User::*create*([  
 'name' => $request->name,  
 'email' => $request->email,  
 'password' => Hash::make($request->password),  
 'username' => $request->username,  
 'phone' => $request->phone,  
 ]);  
  
 ……………….

}

* 1. **Personnalisation de la page de connexion :**

1. Mettre à jour la vue de connexion **ressources/views/auth/login.blade.php**

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

1. Mettre à jour le fichier de validation **app/http/Requests/Auth/LoginRequest.php :**

* Remplacer le code suivant :

|  |
| --- |
| public function rules() {  return [  'email' => ['required', 'string', 'email'],  'password' => ['required', 'string'],  ]; } |

par

|  |
| --- |
| public function rules() {  return [  'username' => ['required', 'string'],  'password' => ['required', 'string'],  ]; } |

* Ensuite remplacer la ligne de code suivante :

|  |
| --- |
| public function authenticate() {  $this->ensureIsNotRateLimited();   if (! Auth::attempt($this->only('email', 'password'), $this->boolean('remember'))) {  RateLimiter::hit($this->throttleKey());   throw ValidationException::*withMessages*([  'email' => trans('auth.failed'),  ]);  }   RateLimiter::clear($this->throttleKey()); } |

Par :

|  |
| --- |
| public function authenticate() {  $this->ensureIsNotRateLimited();   if (! Auth::attempt($this->only('username', 'password'), $this->boolean('remember'))) {  RateLimiter::hit($this->throttleKey());   throw ValidationException::*withMessages*([  'username' => trans('auth.failed'),  ]);  }   RateLimiter::clear($this->throttleKey()); } |

* Puis remplacer le code suivant :

|  |
| --- |
| throw ValidationException::*withMessages*([  'email' => trans('auth.throttle', [  'seconds' => $seconds,  'minutes' => *ceil*($seconds / 60),  ]), ]); |

Par

|  |
| --- |
| throw ValidationException::*withMessages*([  'username' => trans('auth.throttle', [  'seconds' => $seconds,  'minutes' => *ceil*($seconds / 60),  ]), ]); |

1. Créer maintenant un nouvel utilisateur pour tester les nouveaux paramètres de connexion.
   1. **Vérification d’email**

**Remarque :** il fallait de générer le mot de passe d’application google pour réussir cette partie du l’atelier.

1. Configurer les détails SMTP dans le fichier**. env**, de la façon suivante :

MAIL\_MAILER=smtp  
MAIL\_HOST=smtp.gmail.com  
MAIL\_PORT=465  
MAIL\_USERNAME=……..

MAIL\_PASSWORD=…………….  
MAIL\_ENCRYPTION=null  
MAIL\_FROM\_ADDRESS="hello@example.com"  
MAIL\_FROM\_NAME="${APP\_NAME}"

Activer la vérification d’email au sein du fichier **config/fortify.php**, plus

1. Ajoutez que votre modèle **App\Models\User** implémente le contrat **Illuminate\Contracts\Auth\MustVerifyEmail**.
2. Ré-lister les routes.

Après avoir activé cette fonction, vous verrez toutes les routes de vérification des e-mails en action. Les itinéraires de vérification des e-mails sont de deux types :

* Route pour l’envoie un e-mail à l'utilisateur pour vérification.
* Route qui change le statut lorsque l'utilisateur clique sur l'email de vérification.

1. Créer un nouvel utilisateur
   1. **Gestion des rôles et permissions**

Le package de composition d'autorisation de rôle Spatie fournit un moyen de créer une liste de contrôle d'accès dans laravel 9.

On souhaite créer un compte administrateur, qui a le droit :

* Gérer les utilisateurs,
* Gérer les rôles,
* Attribuer un rôle à un utilisateur précis.

1. Installer le package Spatie en utilisant la commande suivante :

**composer require spatie/laravel-permission**

1. Installer le package de collecte de formulaires.

**composer require laravelcollective/html**

1. Taper la commande ci-dessous pour obtenir le fichier de configuration dans **config/permission.php** et les fichiers de migration.

**php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"**

|  |
| --- |
| protected $routeMiddleware = [ ………………………………………..//  'role' => \Spatie\Permission\Middlewares\RoleMiddleware::class,  'permission' => \Spatie\Permission\Middlewares\PermissionMiddleware::class,  'role\_or\_permission' => \Spatie\Permission\Middlewares\RoleOrPermissionMiddleware::class, ]; |

1. Lancer les migrations.
2. Le package Spatie fournit son middleware intégré de sorte que nous pouvons l'utiliser. Pour ce faire, modifier le fichier « **app/http/kernet.php** » pour ajouter ce middleware comme suit :

8- Ajouter les contrôleurs de ressources de module utilisateur et rôle en tapant les commandes suivantes :

**php artisan make:controller UserController –resource**

**php artisan make:controller RoleController --resource**

1. Modifier les deux fichiers « **UserController.php** » et « **RoleController.php** » comme suit :

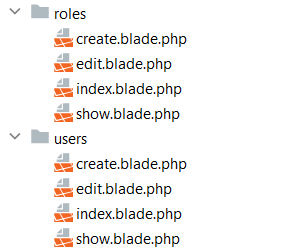
**UserController.php**

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use App\Models\User; use Illuminate\Http\Request; use Illuminate\Support\Arr; use Illuminate\Support\Facades\DB; use Illuminate\Support\Facades\Hash; use Spatie\Permission\Models\Role;    class UserController extends Controller {  */\*\*  \* Display a listing of the resource.  \*  \* @return \Illuminate\Http\Response  \*/* public function index(Request $request)  {  $data = User::*orderBy*('id','DESC')->paginate(5);  return view('users.index',*compact*('data'))  ->with('i', ($request->input('page', 1) - 1) \* 5);  }   */\*\*  \* Show the form for creating a new resource.  \*  \* @return \Illuminate\Http\Response  \*/* public function create()  {  $roles = Role::*pluck*('name','name')->all();  return view('users.create',*compact*('roles'));  }   */\*\*  \* Store a newly created resource in storage.  \*  \* @param \Illuminate\Http\Request $request  \* @return \Illuminate\Http\Response  \*/* public function store(Request $request)  {  $this->validate($request, [  'name' => 'required',  'email' => 'required|email|unique:users,email',  'password' => 'required|same:confirm-password',  'roles' => 'required'  ]);   $input = $request->all();  $input['password'] = Hash::make($input['password']);   $user = User::*create*($input);  $user->assignRole($request->input('roles'));   return redirect()->route('users.index')  ->with('success','User created successfully');  }   */\*\*  \* Display the specified resource.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function show($id)  {  $user = User::*find*($id);  return view('users.show',*compact*('user'));  }   */\*\*  \* Show the form for editing the specified resource.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function edit($id)  {  $user = User::*find*($id);  $roles = Role::*pluck*('name','name')->all();  $userRole = $user->roles->pluck('name','name')->all();   return view('users.edit',*compact*('user','roles','userRole'));  }   */\*\*  \* Update the specified resource in storage.  \*  \* @param \Illuminate\Http\Request $request  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function update(Request $request, $id)  {  $this->validate($request, [  'name' => 'required',  'email' => 'required|email|unique:users,email,'.$id,  'password' => 'same:confirm-password',  'roles' => 'required'  ]);   $input = $request->all();  if(!empty($input['password'])){  $input['password'] = Hash::make($input['password']);  }else{  $input = Arr::*except*($input,array('password'));  }   $user = User::*find*($id);  $user->update($input);  DB::table('model\_has\_roles')->where('model\_id',$id)->delete();   $user->assignRole($request->input('roles'));   return redirect()->route('users.index')  ->with('success','User updated successfully');  }   */\*\*  \* Remove the specified resource from storage.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function destroy($id)  {  User::*find*($id)->delete();  return redirect()->route('users.index')  ->with('success','User deleted successfully');  } } |

**RoleController.php**

|  |
| --- |
| <?php  namespace App\Http\Controllers;  use Illuminate\Http\Request; use App\Http\Controllers\Controller; use Spatie\Permission\Models\Role; use Spatie\Permission\Models\Permission; use Illuminate\Support\Facades\DB;  class RoleController extends Controller {  */\*\*  \* Display a listing of the resource.  \*  \* @return \Illuminate\Http\Response  \*/* function \_\_construct()  {  $this->middleware('permission:role-list|role-create|role-edit|role-delete', ['only' => ['index','store']]);  $this->middleware('permission:role-create', ['only' => ['create','store']]);  $this->middleware('permission:role-edit', ['only' => ['edit','update']]);  $this->middleware('permission:role-delete', ['only' => ['destroy']]);  }   */\*\*  \* Display a listing of the resource.  \*  \* @return \Illuminate\Http\Response  \*/* public function index(Request $request)  {  $roles = Role::*orderBy*('id','DESC')->paginate(5);  return view('roles.index',*compact*('roles'))  ->with('i', ($request->input('page', 1) - 1) \* 5);  }   */\*\*  \* Show the form for creating a new resource.  \*  \* @return \Illuminate\Http\Response  \*/* public function create()  {  $permission = Permission::*get*();  return view('roles.create',*compact*('permission'));  }   */\*\*  \* Store a newly created resource in storage.  \*  \* @param \Illuminate\Http\Request $request  \* @return \Illuminate\Http\Response  \*/* public function store(Request $request)  {  $this->validate($request, [  'name' => 'required|unique:roles,name',  'permission' => 'required',  ]);   $role = Role::*create*(['name' => $request->input('name')]);  $role->syncPermissions($request->input('permission'));   return redirect()->route('roles.index')  ->with('success','Role created successfully');  }  */\*\*  \* Display the specified resource.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function show($id)  {  $role = Role::*find*($id);  $rolePermissions = Permission::*join*("role\_has\_permissions","role\_has\_permissions.permission\_id","=","permissions.id")  ->where("role\_has\_permissions.role\_id",$id)  ->get();   return view('roles.show',*compact*('role','rolePermissions'));  }   */\*\*  \* Show the form for editing the specified resource.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function edit($id)  {  $role = Role::*find*($id);  $permission = Permission::*get*();  $rolePermissions = DB::table("role\_has\_permissions")->where("role\_has\_permissions.role\_id",$id)  ->pluck('role\_has\_permissions.permission\_id','role\_has\_permissions.permission\_id')  ->all();   return view('roles.edit',*compact*('role','permission','rolePermissions'));  }   */\*\*  \* Update the specified resource in storage.  \*  \* @param \Illuminate\Http\Request $request  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function update(Request $request, $id)  {  $this->validate($request, [  'name' => 'required',  'permission' => 'required',  ]);   $role = Role::*find*($id);  $role->name = $request->input('name');  $role->save();   $role->syncPermissions($request->input('permission'));   return redirect()->route('roles.index')  ->with('success','Role updated successfully');  }  */\*\*  \* Remove the specified resource from storage.  \*  \* @param int $id  \* @return \Illuminate\Http\Response  \*/* public function destroy($id)  {  DB::table("roles")->where('id',$id)->delete();  return redirect()->route('roles.index')  ->with('success','Role deleted successfully'); }} |

1. Ajouter les vues de chaque module comme indiqué dans la figure suivante :



1. Créer un seeder intitulé « **PermissionTableSeeder** » en tapant la commande suivante :

**php artisan make:seeder PermissionTableSeeder**

Puis modifier ce fichier en ajoutant le code suivant :

|  |
| --- |
| public function run() {  $permissions = [  'role-list',  'role-create',  'role-edit',  'role-delete',  ];   foreach ($permissions as $permission) {  Permission::*create*(['name' => $permission]);  } } |

1. Lancer la commande suivante :

**php artisan db:seed --class=PermissionTableSeeder**

1. Créer maintenant un nouveau seeder nommer « **CreateAdminUserSeeder** » pour créer un utilisateur administrateur :

**php artisan make:seeder CreateAdminUserSeeder**

Puis modifier ce fichier en ajoutant le code suivant :

|  |
| --- |
| public function run() {  $user = User::*create*([  'username'=>'admin\_test',  'phone'=>'12121',  'name' => 'Admin',  'email' => 'email@esprit.tn',  'password' => bcrypt('123456')  ]);   $role = Role::*create*(['name' => 'Admin']);   $permissions = Permission::*pluck*('id','id')->all();   $role->syncPermissions($permissions);   $user->assignRole([$role->id]); } |

Puis lancer la commande suivante :

**php artisan db:seed --class=CreateAdminUserSeeder**

1. Tester maintenant votre code avec les informations d'identification suivantes :

|  |
| --- |
| **Email:** email@esprit.tn  **Password:** 123456 |