

Créer une application avec Laravel 5.5 – Le profil

Maintenant notre galerie est opérationnelle il ne nous reste plus qu'à un peu peaufiner tout ça. Dans ce chapitre nous allons mettre en place une page de profil pour les utilisateurs pour leur permettre de modifier leur adresse courriel et la pagination.

Les données

Il nous faut compléter la table **users** pour ajouter la valeur de la pagination. On reprend donc la migration :

```
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        ...

        $table->json('settings');
        ...

    });
}
```

On se réserve la possibilité d'ajouter d'autres éléments dans le futur on opte pour un format JSON.

Si on avait une table déjà remplie d'informations sur un site existant on ferait une migration complémentaire pour juste ajouter la nouvelle colonne.

On met à jour le seeder pour la table **users** pour affecter une valeur à la colonne **settings** :

```
public function run()
{
    User::create([
        'name' => 'Durand',
        'email' => 'durand@chezlui.fr',
        'role' => 'admin',
```

```

        'password' => bcrypt('admin'),
        'settings' => '{"pagination": 8}',
    ]);

    User::create([
        'name' => 'Dupont',
        'email' => 'dupont@chezlui.fr',
        'password' => bcrypt('user'),
        'settings' => '{"pagination": 8}',
    ]);
}

```

On complète aussi **RegisterController** :

```

protected function create(array $data)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => bcrypt($data['password']),
        'settings' => '{"pagination": 8}',
    ]);
}

```

On relance les migrations pour rafraîchir notre base :

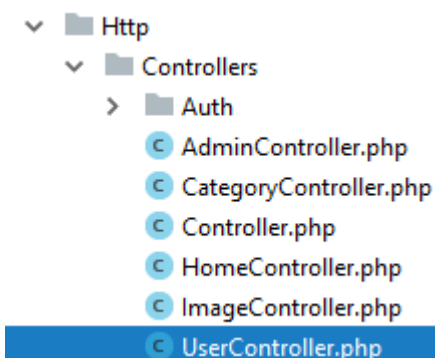
```
php artisan migrate:fresh --seed
```

Et on vérifie que la colonne a bien été créée dans la table.

Le contrôleur et les routes

On crée un nouveau contrôleur :

```
php artisan make:controller UserController --resource
```



On va conserver seulement les fonctions **edit** et **update** et ajouter la référence du modèle :

```
<?php

namespace App\Http\Controllers;

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

class UserController extends Controller
{
    public function edit($id)
    {
        //
    }

    public function update(Request $request, $id)
    {
        //
    }
}
```

Et on complète les routes :

```
Route::middleware('auth')->group(function () {

    Route::resource('profile', 'UserController', [
        'only' => ['edit', 'update'],
        'parameters' => ['profile' => 'user']
    ]);

    ...

});
```

PUT	PATCH	profile/{user}	profile.update	App\Http\Controllers\UserController@update	web,auth
GET	HEAD	profile/{user}/edit	profile.edit	App\Http\Controllers\UserController@edit	web,auth

Le menu

Il nous faut encore ajouter un item dans le menu (**views/layouts/app**) réservé aux utilisateurs connectés :

```

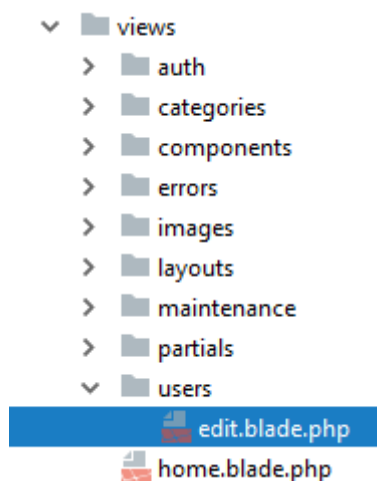
@guest
    ...
@else
    <li class="nav-item{{ currentRoute(route('profile.edit',
auth()->id())) }}"><a class="nav-link" href="{{
route('profile.edit', auth()->id()) }}">@lang('Profil')</a></li>
    ...
@endguest

```

Profil Déconnexion

La vue

On crée un dossier et une vue pour le formulaire :



Avec ce code pour la vue :

```

@extends('layouts.form')

@section('css')

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-slider/10.0
.0/css/bootstrap-slider.min.css">

@endsection

@section('card')

    @component('components.card')

```

```

        @slot('title')
            @lang('Modifier le profil')
        @endslot

        <form method="POST" action="{{ route('profile.update',
$user->id) }}">
            {{ csrf_field() }}
            {{ method_field('PUT') }}

            @include('partials.form-group', [
                'title' => __('Adresse email'),
                'type' => 'email',
                'name' => 'email',
                'required' => true,
                'value' => $user->email,
            ])

            <div class="form-group">
                @lang('Pagination : ')<span id="nbr">{{
$settings->pagination }}</span> @lang('images par page')<br>
                <input id="pagination" name="pagination"
type="number" data-slider-min="3" data-slider-max="20" data-
slider-step="1" data-slider-value="{{ $settings->pagination
}}"/><br>
            </div>

            @component('components.button')
                @lang('Envoyer')
            @endcomponent

        </form>

    @endcomponent

@endsection

@section('script')

<script
src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-slider/9.9.0
/bootstrap-slider.min.js"></script>

<script>

```

```
$(function() {  
    $("#pagination")  
    .slider()  
    .on("slide", function(e) {  
        $("#nbr").text(e.value)  
    })  
    .on("change", function(e) {  
        $("#nbr").text(e.value.newValue)  
    })  
})  
</script>
```

@endsection

Bootstrap 4 n'est pas équipé d'un slider alors on utilise [celui-ci](#) :

Slider for Bootstrap bootstrap-slider.js

Il y a plusieurs exemples de mise en œuvre sur le site.

Comme on s'en sert que sur cette page on ne va pas l'ajouter dans les assets mais juste le charger par un CDN.

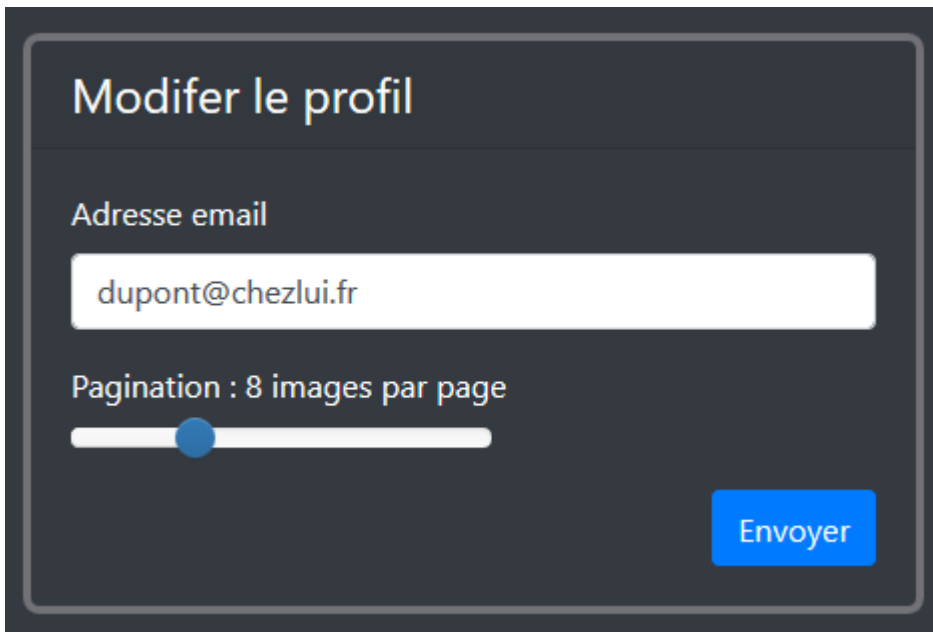
Le formulaire et sa soumission

L'affichage du formulaire

On complète **UserController** :

```
public function edit(User $user)  
{  
    $settings = json_decode($user->settings);  
  
    return view ('users.edit', compact('user', 'settings'));  
}
```

Et le formulaire peut maintenant s'afficher :



La soumission

On commence par compléter le modèle **User** pour l'assignation de masse avec la colonne **settings** :

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

On complète ensuite **UserContrôller** pour traiter la soumission :

```
public function update(Request $request, User $user)  
{  
    $request->validate([  
                                                                    'email' =>  
'required|string|email|max:255|unique:users,email,' . $user->id,  
        'pagination' => 'required',  
    ]);  
  
    $user->update([  
        'email' => $request->email,  
        'settings' => json_encode(['pagination' =>  
$request->pagination]),  
    ]);  
  
    return back()->with(['ok' => __('Le profil a bien été mis à  
jour')]);  
}
```

Et on vérifie que ça fonctionne !

Le profil a bien été mis à jour

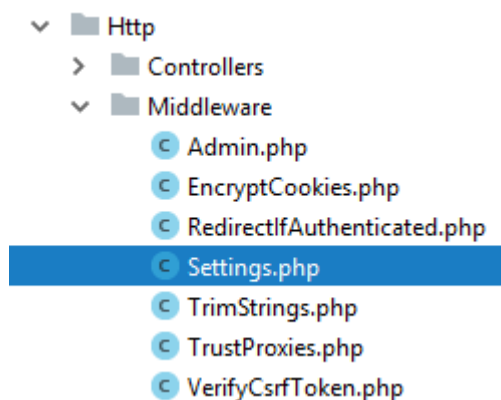


Un middleware

Ce n'est pas parce que maintenant la valeur de la pagination est fixée dans la table que ça va magiquement la changer dans la galerie !

On crée un nouveau middleware :

```
php artisan make:middleware Settings
```



Et on code ainsi :

```
<?php
```

```
namespace App\Http\Middleware;
```

```
use Closure;
```

```
class Settings
```

```
{
```

```
    public function handle($request, Closure $next)
```

```
    {
```

```
        if(auth()->check()) {
```

```
            $settings = json_decode(auth()->user()->settings);
```

```
            config(['app.pagination' => $settings->pagination]);
```



```

    }

    return $next($request);
}
}

```

Si le visiteur est authentifié on récupère son réglage de pagination et on actualise la configuration.

On déclare ce middleware dans **app/Http/Kernel** :

```

protected $middlewareGroups = [
    'web' => [

        ...

        \App\Http\Middleware\Settings::class,
    ],

    ...
];

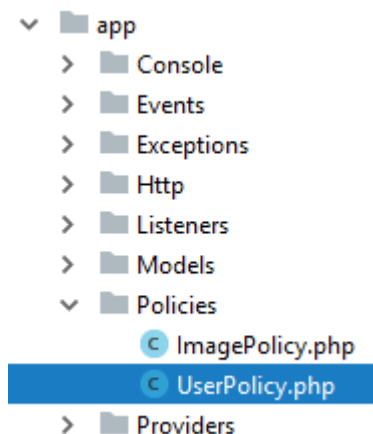
```

Et maintenant la pagination personnalisée doit fonctionner !

Une autorisation

Il nous reste à sécuriser la modification du profil. En effet il est pour le moment assez facile d'usurper une identité pour modifier le profil d'une autre utilisateur. On va donc créer un police :

```
php artisan make:policy UserPolicy --model=User
```



On va conserver que la fonction **update** et coder ainsi :

```
<?php

namespace App\Policies;

use App\Models\User;
use Illuminate\Auth\Access\HandlesAuthorization;

class UserPolicy
{
    use HandlesAuthorization;

    public function update(User $user, User $userprofile)
    {
        return $user->id === $userprofile->id;
    }
}
```

On renseigne **AuthServiceProvider** :

```
use App\Policies\ { ImagePolicy, UserPolicy };
use App\Models\ { Image, User };

...
```

```
protected $policies = [
    Image::class => ImagePolicy::class,
    User::class => UserPolicy::class,
];
```

Et on complète **UserController** :

```
public function update(Request $request, User $user)
{
    $this->authorize('update', $user);

    ...
}
```

Et maintenant on est tranquilles...

Conclusion

Dans ce chapitre on a :

- complété la migration de la table **users** pour la pagination personnalisée
- ajouté un lien dans le menu pour le profil
- créé un contrôleur et les routes pour le profil
- créé le formulaire de modification du profil
- codé le traitement du formulaire
- ajouté un middleware pour rendre effective la pagination personnalisée
- ajouté une autorisation pour la modification du profil

Pour vous simplifier la vie vous pouvez [charger le projet](#) dans son état à l'issue de ce chapitre.