AuteurEnHerbe

Zeste de savoir

How to : Mettre en place oAuth dans une Webapp avec Laravel Passport et Vue.js

4 février 2023

Table des matières

Introduction . . . 1
1. Présentation d'OAuth, protocole d'autorisation (permissions) et non de connexion 1
1.1. Le problème . . . 1
1.2. La (bonne) solution . . . 2
1.3. Pour aller plus loin . . . 4
2. Code back-end . . . 4
2.1. Routes . . . 4
2.2. Gestion des clients oAuth . . . 7
3. Code front end . . . 16
3.1. Présentation globale . . . 16
3.2. App.vue et l'intercepteur de requêtes Vue.js . . . 16
3.3. LaravelOAuthProjectTester.vue . . . 19
3.4. Ou, dit d'une autre façon, plus résumée… . . . 20
3.5. Note sur les deux URL . . . 21
Conclusion . . . 21

Introduction

Dans ce post, je vous propose un exemple concret d'implémentation du protocole d'autorisations oAuth avec Laravel Passport mettant en scène une application Vue.js côté front et une application Laravel côté back. Préalablement, je partirai du principe que vous découvrez oAuth et définirai donc ce protocole. Ce *post* présentera ensuite la partie back et la partie front, en mettant en lumière les opérations relatives aux access tokens et refresh tokens d'oAuth.

1. Présentation d'OAuth, protocole d'autorisation (permissions) et non de connexion

1.1. Le problème

Imaginons que vous souhaitiez importer les publications que vous avez sur votre compte Instagram sur votre compte Tinder, dans le cadre du remplissage de votre profil Tinder  .

Cette action nécessite de vous connecter sur l'application Tinder pour pouvoir accéder à votre compte, mais aussi sur le système d'Instagram, pour pouvoir accéder à vos publications. Il faut aussi que vous autorisiez le système de Tinder à recevoir les images de votre compte Instagram.

Remarque importante: je distingue donc la connexion (= ici, authentification pour accéder à son compte) de l'autorisation (= ici, donner des permissions à tel ou tel système).

1.1.1. La (mauvaise) solution

Une première solution serait de renseigner directement dans Tinder vos identifiants de connexion à Instagram. Tinder afficherait un champ "Votre Identifiant Instagram" et un autre "Votre Mot de Passe Instagram" par exemple. Vous rempliriez cela et on pourrait imaginer que cela suffise à établir une authentification entre Tinder et Instagram. L'autorisation de récupérer les images d'Instagram et les transmettre à Tinder pourrait également être envisagée.

1.1.2. Pourquoi cette solution est-elle mauvaise ?

On ne veut bien sûr pas donner à Tinder les identifiants dont on dispose sur un autre système (ici, Instagram). C'est l'une des raisons d'être du protocole OAuth: ce dernier permet d'éviter cette situation peu confidentielle. Ce n'est pas pour autant qu'OAuth est un protocole d'authentification, c'est bien un protocole d'autorisation, à base d'un système de *scopes* permettant de définir les permissions et de les donner.

1.2. La (bonne) solution

Je vais vous expliquer ce qu'est OAuth dans cette partie, en continuant avec l'exemple de Tinder et d'Instagram. Pour rappel, le contexte est: vous êtes connecté sur l'application de Tinder. Vous êtes en train de mettre à jour votre profil et dans ce cadre-là, vous souhaitez importer vos publications Instagram pour maximiser vos chances de match. Mais il serait peu confidentiel, et donc inenvisageable pour Tinder, que l'application de rencontres vous invite à lui communiquer vos identifiants Instagram. Au lieu de cela, Tinder utilise le protocole OAuth. Et comme indiqué au début de cette partie, je vais vous présenter OAuth. La boucle est bouclée.

Imaginons le système composé des systèmes qui suivent:

— L'utilisateur de l'application Tinder (c'est-à-dire "vous")
— Le serveur Tinder
— L'API Instagram (il s'agit du système Instagram précédemment mentionné)
— Serveur HTTP d'autorisation OAuth Instagram

Voici ci-dessous les différentes grandes étapes qui vont s'exécuter, dans cet ordre, et qui ensemble implémentent OAuth.

1. Vous tentez de récupérer vos images Instagram depuis Tinder. Donc, vous envoyez au serveur Tinder la requête de récupérer les images Instagram.
2. Le serveur Tinder transmet cette requête à l'API Instagram.
3. L'API Instagram répond au serveur Tinder en lui envoyant l'URL du serveur HTTP d'autorisation Instagram, qui contient notamment un formulaire d'authentification pour que vous puissiez vous connecter à l'API Instagram.
4. Le serveur Tinder transmet à l'application Tinder cette URL Instagram (on peut dire que c'est une "URL Instagram" puisqu'elle désigne leur serveur d'autorisation).

5. L'application Tinder appelle cette URL Instagram.
6. Le serveur HTTP d'autorisation Instagram ainsi appelé répond à l'application Tinder en lui envoyant le formulaire d'authentification à Instagram. En d'autres termes: vous êtes donc redirigé, par l'application Tinder, sur le formulaire d'authentification à Instagram. Notez que ce formulaire appartient vraiment à Instagram et pas à Tinder! Ce qui fait que les identifiants d'authentification Instagram que vous y renseignerez ne seront pas connus de Tinder, ce qui résout le problème mentionné plus haut. Dans ce formulaire, en général si l'authentification réussit, vous trouverez aussi un sous-formulaire d'octroi des permissions: vous pourrez ainsi autoriser ou non Tinder à accéder puis à récupérer vos publications Instagram.
7. Vous allez vous authentifier et octroyer ces permissions en remplissant le formulaire d'Instagram. Vous allez cliquer sur le bouton d'authentification et sur celui d'octroi des permissions. Si les identifiants Instagram que vous indiquez sont valides, le serveur HTTP d'autorisation d'Instagram enverra un **code d'autorisation OAuth** ("**OAuth Authorization Code**").
8. L'application Tinder reçoit ce code d'autorisation OAuth. Elle l'envoi au serveur Tinder.
9. Le serveur Tinder envoi à l'API Instagram ce code d'autorisation OAuth.
10. Le serveur Tinder vérifie la validité de ce code d'autorisation OAuth pour être sûr que ce ne soit pas un faux et qu'il n'y ait pas d'autres problèmes de validité en termes de sécurité ou de date d'expiration.
11. Si la phase de vérification du code d'autorisation OAuth est passée avec succès, l'API Instagram répond au serveur Tinder en lui envoyant un **jeton d'accès OAuth** ("**OAuth Access Token**").
12. L'application Tinder va automatiquement, ou sous votre joug, relancer votre requête de récupération de vos publications Instagram vers Tinder depuis l'application Tinder en l'adressant, comme à la toute première étape, au serveur Tinder.
13. Le serveur Tinder, disposant du jeton d'accès OAuth, va enfin pouvoir sauter toutes les étapes au-dessus. Donc, il va transmettre une requête HTTP vers l'API Instagram pour y récupérer vos publications Instagram, tout en associant ce jeton d'accès OAuth à cette requête.
14. Bien sûr, Instagram va recevoir cette requête et ce jeton d'accès OAuth et va fournir au serveur Tinder vos publications Instagram.
15. Enfin, le serveur Tinder va les transmettre à l'application Tinder, donc il va vous les transmettre.

1.2.1. Notes sur le vocabulaire d'OAuth

Dans certaines documentations OAuth, le terme "Client OAuth" ne désigne pas l'application Tinder mais le serveur Tinder, puisqu'un client est un système qui communique avec un serveur, ce qui ne désigne pas nécessairement une application mobile.

L'API Instagram est appelée le serveur de ressources.

Les ressources correspondent ici aux publications Instagram.

1.2.2. Notes sur l'authentification

On remarque donc qu'OAuth ne gère pas l'authentification mais bien l'autorisation, à travers le sous-formulaire de permissions que j'ai brièvement mentionné plus haut.

L'authentification est quant à elle laissée aux deux systèmes Tinder et Instagram comme à leur habitude. On a d'ailleurs bien deux authentifications qui sont réalisées, de manière tout à fait distinctes, afin de ne pas porter à la connaissance de Tinder vos identifiants d'authentification Instagram comme précédemment indiqué.

1.3. Pour aller plus loin

Je vous invite à lire https://zestedesavoir.com/articles/1616/comprendre-oauth-2–0-par-lexemple/ , de @BestCoder.

2. Code back-end

Grâce à Laravel Passport, les choses à mettre en place côté serveur pour implémenter OAuth relèvent surtout de la configuration: la création et le retour du code d'autorisation, du token d'accès et du token de rafraîchissement (qui permet de requêter un nouveau token d'accès et un nouveau token de rafraîchissement, à utiliser si la date d'expiration du token d'accès a été atteinte ou dépassée) sont déjà gérés par Passport. Autrement dit, je n'ai pas eu besoin de les écrire.

2.1. Routes

J'ai mis en place cette route d'API dans le fichier `routes/api.php`:

```
Route::get('/hello', function (Request $request) {
        return 'hello';
});
```

L'idée est de n'autoriser son accès que si l'utilisateur s'est connecté à son compte Laravel et qu'il a autorisé le front Vue.js à requêter cette route en lui donnant ses permissions OAuth. Pour ce faire, Laravel Passport est à définir en tant que *guard* Laravel de cette route:

```
Route::middleware('auth:api')->group(function() {
        Route::get('/user', function (Request $request) {
                return $request->user();
        });

        Route::get('/hello', function (Request $request) {
```

```
                return 'hello';
        });
});
```

Le *guard* `auth:api` est défini dans le fichier *config/auth.php* d'après la documentation de Laravel Passport https://laravel.com/docs/9.x/passport#installation :

```
'guards' => [
        'web' => [
            'driver' => 'session',
            'provider' => 'users',
        ],

    'api' => [
      'driver' => 'passport',
      'provider' => 'users'
    ]
  ],
```

Les autres routes que j'ai définies le sont quant à elles dans le fichier *routes/web.php*. Les voici:

```
Route::get('/login', [AuthController::class,
    'show'])->name('login');
Route::post('/login', [AuthController::class, 'login']);
Route::post('/logout', function() {
        auth()->logout();
        return redirect('/');
})->name('logout');
```

Ces trois routes définies gèrent respectivement l'affichage du formulaire d'authentification Laravel, que Vue.js devra afficher à un moment donné (plus de détails dans la section dédiée au front end), le traitement de l'authentification tentée à travers le remplissage de ce formulaire par l'utilisateur final de l'application Vue.js, et la déconnexion de ce dernier.

2.1.1. Contrôleur *AuthController*

Voici le contenu du contrôleur correspondant à ces deux premières routes:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
6
7  class AuthController extends Controller
8  {
9          public function show()
10     {
11             return view('user.login');
12     }
13
14         public function login(Request $request)
15     {
16         $credentials = $request->validate([
17             'email' => ['required', 'email'],
18             'password' => ['required'],
19         ]);
20
21         if (!auth()->attempt($credentials, true)) {
22             return redirect(route('login'))->withErrors([
23                 'email' => 'Invalid email or password',
24             ])->onlyInput('email');
25         }
26
27         $request->session()->regenerate();
28         return redirect()->intended();
29     }
30
31         public function clientsIndex()
32     {
33         return view('oauth_clients.index');
34     }
35 }
```

Petite note concernant `$request->session()->regenerate();`: la documentation Laravel donne l'indication d'utiliser cet appel pour prévenir la faille de sécurité "Fixation de session ".

2.1.2. Appel des routes depuis l'application frond end

Je vous conseille de consulter la section qui présente le front end réalisé avec Vue.js pour mieux identifier le rôle des composants graphiques que je mentionne ci-dessous.

La route qui affiche le formulaire, GET `/login`, sera appelée si l'utilisateur initie le process OAuth, en cliquant sur le lien. Elle pointe la méthode `show` du contrôleur `AuthController`, qui retourne la vue suivante:

```
1  <form method="POST" action="/login">
2      @csrf
3
```

```
    @if ($errors->any())
        <div class="alert alert-danger">
            {{ $errors->first('email') }}
        </div>
    @endif

    <div>
        <label for="email">Email:</label>
        <input type="email" name="email" id="email">
    </div>

    <div>
        <label for="password">Password:</label>
        <input type="password" name="password" id="password">
    </div>

    <div>
        <button type="submit">Login</button>
    </div>
</form>
```

Ce formulaire, qui notons-le au passage implémente CSRF , envoie une requête HTTP POST à la route POST `/login`, qui correspond à la méthode `login` du contrôleur.

Une fois que l'utilisateur final aura cliqué sur son bouton "Connect with OAuth" et qu'il se sera connecté dans l'onglet contenant le formulaire Laravel et qui sera affiché par l'application Vue.js, Passport affichera le formulaire d'autorisation des permissions. Tout ce système est transparent pour le développeur back-end Laravel: en d'autres termes, il n'a pas besoin de gérer cela lui-même.

Le système oAuth offert par Passport peut toutefois être configuré. Par exemple, d'après https://laravel.com/docs/9.x/passport#token-lifetimes , il est possible gérer la date d'expiration des tokens oAuth (access_token et refresh_token).

Je vous invite à consulter la documentation de Laravel https://laravel.com/docs/9.x/passport pour en savoir plus sur ce qui est configurable et sur comment mettre en place Passport en back. A l'exception de la gestion des clients oAuth, que je vais vous présenter ci-dessous (j'ai généré une interface graphique prête à l'emploi qui pourrait peut-être vous intéresser).

2.2. Gestion des clients oAuth

Par "client oAuth", j'entends ici l'application Vue.js par exemple. Avec oAuth, on peut définir un couple ("client_id; client_secret") qui seront envoyés par Vue.js à Passport pour que Vue.js soit identifiée et rendue apte à communiquer en oAuth avec l'application Laravel (d'où le terme de "client").

Il faut donc que Passport génère ces client_id et client_secret et que ces derniers soient communiqués d'une façon ou d'une autre (pas nécessairement informatique d'ailleurs) au développeur front-end Vue.js.

J'ai écrit le code qui les génère en Javascipt. Passport propose en effet une interface API JSON qui peut être appelée pour ce faire: https://laravel.com/docs/9.x/passport#clients-json-api . Côté application Laravel, j'ai donc écrit une page de gestion des clients oAuth: on peut les CRUD. La création d'un client implique bien sûr la génération du couple ("client_id; client_secret") pour ce client. Cela se présente sous la forme d'un tableau avec des boutons. Une fois cela fait, il faut dans l'application Vue.js renseigner quelque part ce couple ("client_id; client_secret") et l'utiliser dans les *headers* de chaque requête envoyée à Passport (voir la partie front-end).

Voici le code de cette interface graphique, écrite en Blade (un moteur de *templating* Laravel). La partie en Javascript est celle qui communique avec l'API JSON de Passport, consultez par exemple le listener d'événement `$('#createClientBtn').on('click', function() {`.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">

<head>
    <meta charset="utf-8">
        <meta name="csrf-token" content="{{ csrf_token() }}">
    <meta name="viewport"
        content="width=device-width, initial-scale=1">
    <script
        src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
    <link rel="stylesheet" href=
        "https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.cs
        integrity=
            "sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqK

            crossorigin="anonymous">
    <script src=
        "https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bundle
        integrity=
            "sha384-ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7

            crossorigin="anonymous">
    </script>

    <title>Laravel oAuth Clients Handling</title>

    <!-- Fonts -->
    <link href=
        "https://fonts.bunny.net/css2?family=Nunito:wght@400;600;700&display=sw

        rel="stylesheet">
</head>

<body class="antialiased">
    @if (auth()->check())
        <div class="card bg-light mb-3">
            <div class="card-body">
```

```
25                  <h5 class="card-title">Welcome, {{
                        auth()->user()->name }}</h5>
26                  <p class="card-text">Your email is: {{
                        auth()->user()->email }}</p>
27
28                                  <form method="POST"
                                        action="{{ route('logout') }}">
29                                      <button type="submit" clas
                                            s="btn btn-primary">Lo
                                            gout</button>
30                                  </form>
31              </div>
32          </div>
33      @else
34          <p>You are not logged in</p>
35      @endif
36
37      <table class="table table-striped">
38          <thead>
39              <tr>
40                  <th>Client ID</th>
41                  <th>Name</th>
42                  <th>Redirect URL</th>
43                  <th>User ID</th>
44                  <th>Client Secret</th>
45                  <th>Actions</th>
46              </tr>
47          </thead>
48          <tbody>
49              <!-- Filled in JS -->
50          </tbody>
51      </table>
52
53      <!-- Modal for creating a new client -->
54      <div class="modal" tabindex="-1" role="dialog"
            id="createClientModal">
55          <div class="modal-dialog" role="document">
56              <div class="modal-content">
57                  <div class="modal-header">
58                      <h5 class="modal-title">Create a new
                            client</h5>
59                      <button type="button" class="close"
                            data-dismiss="modal" aria-label="Close">
60                          <span aria-hidden="true">&times;</span>
61                      </button>
62                  </div>
63                  <div class="modal-body">
64                      <form>
65                          <div class="form-group">
66                              <label for="name">Name</label>
```

```
67                              <input type="text"
                                    class="form-control" id="name"
68                                  placeholder=
                                        "Enter the name of the client">
69                          </div>
70                          <div class="form-group">
71                              <label for="redirect_url">Redirect
                                    URL</label>
72                              <input type="text"
                                    class="form-control"
                                    id="redirect_url"
73                                  placeholder=
                                        "Enter the redirect URL for the client"
                                        >
74                          </div>
75                      </form>
76                  </div>
77                  <div class="modal-footer">
78                      <button type="button"
                            class="btn btn-secondary"
                            data-dismiss="modal">Close</button>
79                      <button type="button" class="btn btn-primary"
                            id="createClientBtn">Create</button>
80                  </div>
81              </div>
82          </div>
83      </div>
84
85      <!-- Modal for editing a client -->
86      <div class="modal" tabindex="-1" role="dialog"
            id="editClientModal">
87          <div class="modal-dialog" role="document">
88              <div class="modal-content">
89                  <div class="modal-header">
90                      <h5 class="modal-title">Edit client</h5>
91                      <button type="button" class="close"
                            data-dismiss="modal" aria-label="Close">
92                          <span aria-hidden="true">&times;</span>
93                      </button>
94                  </div>
95                  <div class="modal-body">
96                      <form>
97                          <input type="hidden" id="clientId">
98                          <div class="form-group">
99                              <label for="name">Name</label>
100                             <input type="text"
                                    class="form-control" id="editName"
101                                 placeholder=
                                        "Enter the name of the client">
102                         </div>
```

```
103                         <div class="form-group">
104                             <label for="redirect_url">Redirect
                                    URL</label>
105                             <input type="text"
                                    class="form-control"
                                    id="editRedirectUrl"
106                                 placeholder=
                                        "Enter the redirect URL for the client"
                                        >
107                         </div>
108                     </form>
109                 </div>
110                 <div class="modal-footer">
111                     <button type="button"
                            class="btn btn-secondary"
                            data-dismiss="modal">Close</button>
112                     <button type="button" class="btn btn-primary"
                            id="updateClientBtn">Update</button>
113                 </div>
114             </div>
115         </div>
116     </div>
117
118     <!-- Modal for deleting a client -->
119     <div class="modal" tabindex="-1" role="dialog"
            id="deleteClientModal">
120         <div class="modal-dialog" role="document">
121             <div class="modal-content">
122                 <div class="modal-header">
123                     <h5 class="modal-title">Delete client</h5>
124                     <button type="button" class="close"
                            data-dismiss="modal" aria-label="Close">
125                         <span aria-hidden="true">&times;</span>
126                     </button>
127                 </div>
128                 <div class="modal-body">
129                     <p>Are you sure you want to delete this
                            client?</p>
130                     <input type="hidden" id="clientIdToDelete">
131                 </div>
132                 <div class="modal-footer">
133                     <button type="button"
                            class="btn btn-secondary"
                            data-dismiss="modal">Close</button>
134                     <button type="button" class="btn btn-danger"
                            id="deleteClientBtn">Delete</button>
135                 </div>
136             </div>
137         </div>
138     </div>
```

```
139
140     <button class="btn btn-primary" data-toggle="modal"
            data-target="#createClientModal">Create a new
            client</button>
141
142     <script>
143         $(document).ready(function() {
144             // Retrieve the list of clients from the API
145             $.get('http://localhost:80/oauth/clients',
                    function(data) {
146                 data.forEach(function(client) {
147                     var row = '<tr>';
148                     row += '<td>' + client.id + '</td>';
149                     row += '<td>' + client.name + '</td>';
150                     row += '<td>' + client.redirect + '</td>';
151                     row += '<td>' + client.user_id + '</td>';
152                     row += '<td>';
153                     row += '<div class="input-group">';
154                     row +=
                            '<input type="password" class="form-control client-secr
                            +
155                         client
156                         .secret + '" readonly>';
157                     row += '<div class="input-group-append">';
158                     row +=
159                         '<button class="btn btn-secondary show-secret" type
                            ;
160                     row += '</div>';
161                     row += '</div>';
162                     row += '</td>';
163                     row += '<td>';
164                     row +=
                            '<a href="#" class="btn btn-primary edit-client"  data-
                            + client.id
                            +
165                         '" data-name="' + client.name +
                                '" data-redirect="' + client.redirect
                                + '">Edit</a>';
166                     row +=
                            '<a href="#" class="btn btn-danger delete-client" data-
                            +
                            client
167                         .id + '">Delete</a>';
168                     row += '</td>';
169                     row += '</tr>';
170                     $('tbody').append(row);
171                 });
172             });
```

```
173
174             // Toggle visibility of client secret
175             $('table').on('click', '.show-secret', function() {
176                 var $button = $(this);
177                 var $clientSecret = $(this).closest('.input-group'
                        ).find('.client-secret');
178                 if ($button.text() === 'Show') {
179                     $button.text('Hide');
180                     $clientSecret.attr('type', 'text');
181                 } else {
182                     $button.text('Show');
183                     $clientSecret.attr('type', 'password');
184                 }
185             });
186
187             // Open the edit client modal and populate the form
                    with the client's data
188             $('table').on('click', '.edit-client', function() {
189                                 $('#clientId').val($(this).data(
                                        'id'));
190                                 $('#editName').val($(this).data(
                                        'name'));
191                                 $('#editRedirectUrl').val($(this).
                                        data('redirect'));
192                                 $('#editClientModal').modal('show'
                                        );
193             });
194
195             // Update the client
196             $('#updateClientBtn').on('click', function() {
197                 var id = $('#clientId').val();
198                 var name = $('#editName').val();
199                 var redirectUrl = $('#editRedirectUrl').val();
200                 $.ajax({
201                                         method: 'PUT',
202                     url: '/oauth/clients/' + id,
203                     data: {
204                         name: name,
205                         redirect: redirectUrl
206                     },
207                     success: function(data) {
208                         $('#editClientModal').modal('hide');
209                         location.reload();
210                     },
211                     headers: {
212                         'X-CSRF-TOKEN':
                                $('meta[name="csrf-token"]').attr(
                                'content')
213                     }
214                 });
```

```
215             });
216
217             // Open the delete client modal
218             $('table').on('click', '.delete-client', function() {
219                 var id = $(this).data('id');
220                 $('#clientIdToDelete').val(id);
221                 $('#deleteClientModal').modal('show');
222             });
223
224             // Delete the client
225             $('#deleteClientBtn').on('click', function() {
226                 var id = $('#clientIdToDelete').val();
227                                 $.ajax({
228                     method: 'DELETE',
229                     url: '/oauth/clients/' + id,
230                     success: function(data) {
231                                                 $(
                                                        '#deleteClientModal'
                                                        ).modal('hide'
                                                        );
232                         location.reload();
233                     },
234                     headers: {
235                         'X-CSRF-TOKEN':
                                $('meta[name="csrf-token"]').attr(
                                'content')
236                     }
237                 });
238
239             });
240
241             // Create a new client
242             $('#createClientBtn').on('click', function() {
243                 var name = $('#name').val();
244                 var redirectUrl = $('#redirect_url').val();
245                 $.ajax({
246                     method: 'POST',
247                     url: '/oauth/clients',
248                     data: {
249                         name: name,
250                         redirect: redirectUrl
251                     },
252                     success: function(data) {
253                         $('#createClientModal').modal('hide');
254                         location.reload();
255                     },
256                     headers: {
257                         'X-CSRF-TOKEN':
                                $('meta[name="csrf-token"]').attr(
                                'content')
```

```
                    }
                });
            });
        });
    </script>

    <style>
        table {
            border-collapse: collapse;
            width: 100%;
        }

        th,
        td {
            text-align: left;
            padding: 8px;
        }

        tr:nth-child(even) {
            background-color: #f2f2f2;
        }

        th {
            background-color: #4caf50;
            color: white;
        }

        .client-secret {
            width: 60%;
        }

        .show-secret {
            cursor: pointer;
        }

        .modal {
            display: none;
        }

        .modal-dialog {
            max-width: 600px;
            margin: 1.75rem auto;
        }

        .btn {
            cursor: pointer;
        }
    </style>
</body>

```

```
308 </html>
```

3. Code front end

3.1. Présentation globale

Visuellement, c'est une interface minimaliste affichant un lien initiant une requête OAuth vers mon serveur Laravel et un bouton initiant une requête nécessitant une autorisation OAuth depuis une application Vue.js. Attention: le client OAuth est ici bien l'application Vue.js. Autrement dit, l'élément "Serveur Laravel" situé entre "Vue.js" et "Instagram" de l'exemple que j'ai donné dans la section qui présente OAuth n'existe pas: à la place, les requêtes OAuth sont directement envoyées depuis l'application Vue.js vers le serveur Laravel. Ce dernier joue alors un rôle comparable à celui d'Instagram et du serveur d'autorisation OAuth Instagram.

Au clic sur le lien de cette interface minimaliste, le process OAuth démarre et, si tout se passe bien, c'est-à-dire que l'utilisateur (vous-même) s'authentifie bien sur Laravel à travers le formulaire que l'application Vue.js affiche, et qu'il donne bien les permissions OAuth demandées, alors une 200 s'affiche dans la console développeur (🍊) si l'utilisateur clique sur le bouton.

3.2. App.vue et l'intercepteur de requêtes Vue.js

Ce composant Vue.js affiche le composant enfant `LaravelOAuthProjectTester :accessToken="accessToken" @gotRefreshToken="setRefreshToken" @gotAccessToken="setAccessToken"/>` en lui passant le jeton d'accès OAuth. Le jeton de rafraîchissement - tout comme le token d'accès - est aussi géré à travers un événement qui serait déclenché par ce composant enfant et permet de demander un couple de nouveaux jetons d'accès et de rafraîchissement si le jeton d'accès est expiré (voir `LaravelOAuthProjectTester` et le paragraphe qui suit).

Je mets aussi en place un intercepteur de requêtes Vue.js. Dès que le serveur Laravel renvoie une erreur HTTP suite à une requête initiée par le client Vue.js, et qu'il s'agit d'une 401, je considère (abusivement! Il faudrait en effet effectuer une analyse plus fine) qu'il s'agit d'un problème d'expiration du token d'accès OAuth. L'intercepteur requête alors le couple de nouveaux jetons d'accès et de rafraîchissement précédemment mentionné, puis il initie de nouveau la requête HTTP qui avait échoué avec une erreur 401, en fournissant cette fois le token d'accès OAuth issu du rafraîchissement: il est donc valide. Alors, la requête sera finalement exécutée avec succès par le serveur Laravel, c'est-à-dire qu'une réponse 200 sera retournée depuis Laravel vers le client Vue.js.

3.2.1. App.vue

```
<script>
import HelloWorld from './components/HelloWorld.vue'
import LaravelOAuthProjectTester from
    './components/LaravelOAuthProjectTester.vue'
import authService from './authService';

export default {
        data() {
                return {
                        refreshToken: null,
                        accessToken: null
                }
        },

        methods: {
                setRefreshToken(refresh_token) {
                        this.refreshToken = refresh_token.data
                },
                setAccessToken(access_token) {
                        this.accessToken = access_token.data
                        console.log('setAccessToken',
                            this.accessToken)
                },
        },

        components: {
                LaravelOAuthProjectTester,
                HelloWorld
        },

        created() {
                authService.setupInterceptor.bind(this)();
        }
}


</script>

<template>
  <header>
    <img alt="Vue logo" class="logo" src="./assets/logo.svg"
        width="125" height="125" />

    <div class="wrapper">
      <HelloWorld msg="You did it!" />
    </div>
  </header>
```

```
45
46    <main>
47      <LaravelOAuthProjectTester :accessToken="accessToken"
            @gotRefreshToken="setRefreshToken"
            @gotAccessToken="setAccessToken"/>
48    </main>
49  </template>
50
51  <style scoped>
52  header {
53    line-height: 1.5;
54  }
55
56  .logo {
57    display: block;
58    margin: 0 auto 2rem;
59  }
60
61  @media (min-width: 1024px) {
62    header {
63      display: flex;
64      place-items: center;
65      padding-right: calc(var(--section-gap) / 2);
66    }
67
68    .logo {
69      margin: 0 2rem 0 0;
70    }
71
72    header .wrapper {
73      display: flex;
74      place-items: flex-start;
75      flex-wrap: wrap;
76    }
77  }
78  </style>
```

3.2.2. Intercepteur de requêtes Vue.js

```
1  import axios from 'axios';
2
3  export default {
4          setupInterceptor() {
5                  axios.interceptors.response.use(response => {
6                          return response;
7                  }, error => {
```

```
                        const originalRequest = error.config;
                        if (error.response.status === 401 &&
                            !originalRequest._retry) {
                            const vm = this;

                            originalRequest._retry = true;
                            return axios.post(
                                'http://localhost:80/oauth/token'
                                ,
                                {
                                    'grant_type':
                                        'refresh_token',
                                    'client_id': '10',
                                    'client_secret':
                                        'JNoZHK3lUVvUd7V0xyqYyhcMmfX3hpdo9
                                        ,
                                    'refresh_token':
                                        vm.refreshToken,
                                    'scope': '',
                                }).then(response => {
                                    vm.refreshToken = response
                                        .data.refresh_token;
                                    vm.accessToken = response.
                                        data.access_token;

                                    axios.defaults.headers.com
                                        mon['Authorization'] =
                                        'Bearer ' + response.d
                                        ata.access_token;
                                    originalRequest.headers[
                                        'Authorization'] =
                                        'Bearer ' + response.d
                                        ata.access_token;
                                    return
                                        axios(originalRequest);
                                });
                        }
                        return Promise.reject(error);
                    });
            }
}
```

3.3. LaravelOAuthProjectTester.vue

Dans le composant enfant *LaravelOAuthProjectTester.vue*, on retrouve le bouton et le lien auxquels je faisais référence plus haut. Le clic sur le bouton déclenche l'écouteur d'événement `callApiHello` qui initie une requête vers le serveur Laravel qui, si celle-ci se déroule sans encombre (c'est-à-dire si le token d'accès qui est fourni est valide, notamment au niveau de sa

date d'expiration), retournera une 200. En cas de problème, l'intercepteur que j'ai précédemment montré entrera en scène (comportement défini au-dessus).

On retrouve aussi le lien `Connect with OAuth` qui permet de dérouler le process OAuth. Le lien que j'y fournis est celui prévu par Laravel Passport et est trouvable en jouant un peu avec l'affichage des routes Laravel `php artisan route:list` ainsi qu'en consultant la documentation de Laravel Passport: https://laravel.com/docs/9.x/passport#requesting-tokens-redirecting-for-authorization .

Le lien `Connect with OAuth` a pour URL `http://localhost:80/oauth/authorize?${button_oauth_url}&state=${state}`. C'est-à-dire qu'il fournit à Laravel l'URL Vue.js de redirection `http://localhost:5174/auth/callback` que Laravel devra appeler si le formulaire d'authentification Laravel et celui des permissions sont correctement remplis. Y sont également ajoutés quelques paramètres d'URL requis par Laravel Passport, par exemple `client_id` avec sa valeur `10`. La notion de Client ID et de Client Secret a été expliquée dans la partie précédente sur la mise en place du back end Laravel. Le `state` est une valeur aléatoire (ici, fixe, pour raison de simplicité de développement) qui sert simplement à des fins de sécurité, gérée par Laravel Passport.

Enfin, dernier point: à l'affichage de ce composant (plus précisément, à l'exécution du hook `handleOAuthURL` associé au cycle de vie `created` de Vue.js), j'essaie de récupérer le token d'accès OAuth si l'URL Vue.js de redirection du formulaire Laravel `http://localhost:5174/auth/callback` est appelée. Comme le formulaire a bien été rempli par l'utilisateur (la preuve en est que, précisément, cette URL de redirection a été appelée), alors Laravel aura fourni le code d'autorisation OAuth à la page ciblée par cette URL de redirection. Si cela n'est pas clair, vous pouvez vous rafraîchir la mémoire dans la section qui présente OAuth à travers 15 étapes clé. Je réceptionne donc ce code d'autorisation avec `const code = params.get('code');` et j'essaie de récupérer à l'aide de ce code d'autorisation le token d'accès en appelant en POST l'URL `http://localhost:80/oauth/token` qui est bien sûr une URL Laravel Passport: https://laravel.com/docs/9.x/passport#requesting-tokens-converting-authorization-codes-to-access-tokens + `php artisan route:list` si besoin. Enfin, j'en notifie le composant Vue.js parent.

3.4. Ou, dit d'une autre façon, plus résumée...

Nous avons donc un lien et un bouton. Le lien permet d'amorcer le process OAuth, c'est-à-dire d'afficher le formulaire Laravel d'authentification Larvel et le formulaire Laravel Passport OAuth des permissions.

Tant que l'utilisateur ne clique pas sur ce lien, le bouton ne sert à rien car il est censé appeler une route Laravel protégée par Passport, c'est-à-dire nécessitant les permissions (et la connexion du compte Laravel). Dès que l'utilisateur se sera connecté à Laravel et aura donné ses permissions OAuth Passport, et s'il clique sur ce bouton, alors la requête sera traitée avec succès par Laravel et cela se traduira par une 200 visible dans l'historique des requêtes HTTP de son inspecteur de développement Firefox ou Chromium, etc.

Pour réaliser cela: "Dès que l'utilisateur se sera connecté à Laravel et aura donné ses permissions OAuth Passport", il faut que l'utilisateur clique sur le lien et accomplisse avec succès le process OAuth: donc qu'il se connecte à Laravel grâce au formulaire d'authentification Laravel affiché par Vue.js car retourné par Laravel à Vue.js.

Dès que l'utilisateur sera redirigé vers l'URL de redirection suite à l'envoi du formulaire Laravel, `handleOAuthURL` sera appelée et le jeton d'accès sera demandé. Les jetons d'accès et de rafraîchissement sont alors stockés en RAM. Le jeton d'accès en RAM sera alors utilisé si l'utilisateur clique sur le bouton.

Là où l'intercepteur entre en jeu c'est quand une requête envoyée par Vue.js échoue: ça peut être le cas si l'utilisateur clique sur le bouton et que le tokan d'accès a expiré. Alors, l'intercepteur utilise le token de rafraîchissement en RAM et demande un couple de nouveaux token d'accès et token de rafraîchissement. Puis il met à jour les tokens d'accès et de rafraîchissement avec ceux obtenus et rafraîchis et relancer la requête du bouton, avec le bon token d'accès (celui qui est issu du rafraîchissement).

3.5. Note sur les deux URL

Je fais référence à:

— http://localhost:80/oauth/authorize?$button_oauth_urlstate$ ={state}
— http://localhost:80/oauth/token

Le traitement qui est effectué derrière chacune d'elles est implémenté par Laravel Passport. Autrement dit, je n'ai pas eu besoin de m'en préoccuper. Ces traitements sont bien entendus les implémentations d'OAuth.

Conclusion

Comme vous avez pu le constater, la mise en place d'oAuth dans une application Laravel est fortement facilitée par Passport. N'hésitez donc pas à utiliser cet outil si vous pensez avoir besoin de mettre en place un système de permissions pour une API REST Laravel par exemple.