Laravel Passport Servisi Nedir?

Laravel framework ile geleneksel login formları ile authentication işlemlerini sağlamaktadır. API authentication işlemleri için ise OAuth2 protokolünün tüm özelliklerini Laravel **5.4** ile birlikte gelen **Passport** servisi ile sağlamaktadır.

Not: Bu makaleyi doğru bir şekilde anlamak için temel düzeyde OAuth2 protokolünün bilinmesi gerekmektedir.

Kurulum

```
composer require laravel/passport
```

komutu ile passport servisini kurabiliriz. Daha sonra config/app.php dosyasında yer alan providers array'ine aşağıdaki sınıf eklenir.

```
Laravel\Passport\PassportServiceProvider::class,
```

PassportServiceProvider sınıfı, Passport servisine ait olan migration'ları kayıt eder. Bundan dolayı `php artisan migrate` komutunu çalıştırarak yeni tabloların eklenmesi sağlanmalıdır.

Daha sonra `php artisan passport:install` komutu çalıştırılmalıdır. Bu komut çalıştırıldığı zaman encryption anahtarları ve personel access ile password grant client'ları veritabanına kaydedilir. Burada client users tablosunda tutulan bir kullanıcının client'ı olarak ifade edilmektedir. Yani Passport servisinde bir user'ın bir veya birden fazla client'ı olabilir. Örnek vermek gerekirse, Facebook kullanıcısı masaüstü bir uygulamadan giriş yaptığı zaman masaüstü client'ına, mobilden giriş yaptığında mobil client'a vs. gibi token tabanlı giriş işlemleri yaptığında client'lara sahip olmuş olur.

passport:install komutundan sonra `App\User` sınıfına Laravel\Passport\HasApiTokens trait eklenmelidir. Bu trait giriş yapan kullanıcının token ve scope değerlerini kontrol eden bazı yardımcı metodları ekler:

```
namespace App;

use Laravel\Passport\HasApiTokens;
use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{
    use HasApiTokens, Notifiable;
}
```

Bir sonraki adımda **Passport**::routes metodu **AuthServiceProvider** sınıfının **boot()** metodu içerisinde çağrılması gerekmektedir.

Passport::routes() metodu ile gerekli access token oluşturmayı, revoke etmeyi sağlayan rotaların eklenmesi sağlanır.

```
namespace App\Providers;

use Laravel\Passport\Passport;
use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;

class AuthServiceProvider extends ServiceProvider
{
    /**
     * The policy mappings for the application.
     *
     * @var array
```

```
         */
        protected $policies = [
            'App\Model' => 'App\Policies\ModelPolicy',
        ];

        /**
         * Register any authentication / authorization services.
         *
         * @return void
         */
        public function boot()
        {
            $this->registerPolicies();

            Passport::routes();
        }
    }
```

Son adımda, **config/auth.php** dosyasında api authentication guard'ın driver seçeneğini passport olarak güncellemek gerekmektedir. Bu güncelleme işleminden sonra uygulama Passport servisinin **TokenGuard** sınıfını kullanarak gelen API isteklerini karşılar.

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

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

Bu adımları tamamladıktan sonra Passport servisi başarılı bir şekilde uygulamamıza eklenmiş olacaktır.

Arayüz Hazırlanması

Passport servisi kullanıcıların client ve personal access token oluşturma yapabilmesi için Vue kütüphanesi ile oluşturulmuş componentleri hazır olarak sunar. Aşağıdaki komutu çalıştırdıktan sonra `resources/assets/js/components/passport` klasöründeki vue dosyalarında gerekli düzenlemeleri yapabiliriz.

```
    php artisan vendor:publish --tag=passport-components
```

Yayınlanan componentler `resources/assets/js/components` dizinine eklenir. Bu componentleri vue kütüphanesi ile aşağıdaki gibi `resources/assets/js/app.js` dosyasına eklemek gerekmektedir:

```
    Vue.component(
        'passport-clients',
        require('./components/passport/Clients.vue')
    );

    Vue.component(
        'passport-authorized-clients',
        require('./components/passport/AuthorizedClients.vue')
    );

    Vue.component(
        'passport-personal-access-tokens',
        require('./components/passport/PersonalAccessTokens.vue')
    );
```

Daha sonra `npm run dev` komutu ile asset'ler yeniden compile edilir. Compile edilen asset'ler public klasöründeki app.js ve app.css dosyalarına import edilmiş olur. Yukarıda belirtilen templateler ile bir kullanıcının client oluşturma, silme, erişim yapma gibi arayüzleri

otomatik oluşturulmuş olur. Bu arayüzleri template yani view içerisinde kullanabilmek için aşağıdaki custom html elementleri eklenmelidir:

```
<passport-clients></passport-clients>
<passport-authorized-clients></passport-authorized-clients>
<passport-personal-access-tokens></passport-personal-access-tokens>
```

Üretim Ortamına Deploy Edilmesi

`php artisan passport:keys` komutu ile access token üretmek için gerekli olan encryption key'ler üretilir. Bu key değerleri versiyon sistemlerine (git, svn, tfs) eklenmemesine dikkat edilmelidir. Üretilen private ve public key'ler storage klasöründe tutulmaktadır. **.gitignore** dosyasında `storage/*.key` şeklinde bir satır eklenerek ignore işlemi sağlanmış olur.

Configuration

Token Lifetimes

Varsayılan olarak Passport yenileme istemeyen uzun süreli erişim tokenları oluşturur. Eğer kısa süreli token oluşturmak istiyorsak **AuthServiceProvider** sınıfının **boot()** metoduna tokensExpireIn() ve **refreshTokensExpireIn()** metodları eklenmelidir:

```
use Carbon\Carbon;

/**
  * Register any authentication / authorization services.
  *
  * @return void
  */
public function boot()
{
     $this->registerPolicies();

     Passport::routes();

     Passport::tokensExpireIn(Carbon::now()->addDays(15));

     Passport::refreshTokensExpireIn(Carbon::now()->addDays(30));
}
```

Access Token İşlemleri

Client İşlemleri (Oluşturma, Düzenleme, Silme)

Bir client oluşturmanın en basit yolu `php artisan passport:client` komutu çalıştırmaktır. Bu komutu çalıştırdıktan sonra Passport client ile iligli ek sorular soracaktır.

Kullanıcılar client oluşturma işlemini yukarıdaki komut ile sağlayamayacağı için Passport servisi JSON API sağlamıştır. Yukarıda tanımlanan vue component'leri ile arayüz hızlı bir şekilde yapılmış olur.

Bazı JSON API Metodları

`/oauth/clients` linki GET isteği ile **sisteme giriş yapmış kullanıcının** client'larının listesi json olarak döner. Bu linke aşağıdaki gibi bir

istek gönderildiği zaman yeni client eklenir:

```
const data = {
    name: 'Client Name',
    redirect: 'http://example.com/callback'
};

axios.post('/oauth/clients', data)
    .then(response => {
        console.log(response.data);
    })
    .catch (response => {
        // List errors on response...
    });
```

PUT requesti ile `PUT /oauth/clients/{client-id}` client bilgileri güncellenir:

```
const data = {
    name: 'New Client Name',
    redirect: 'http://example.com/callback'
};

axios.put('/oauth/clients/' + clientId, data)
    .then(response => {
        console.log(response.data);
    })
    .catch (response => {
        // List errors on response...
    });
```

DELETE requesti ile `DELETE /oauth/clients/{client-id}` client silinir:

```
axios.delete('/oauth/clients/' + clientId)
    .then(response => {
        //
    });
```

Client oluşturulduktan sonra, authorization code ve access token için **Passport**::routes() metodu ile tanımlanmış olan /oauth/authorize rotasına GET istek gönderir:

```
Route::get('/redirect', function () {
    $query = http_build_query([
        'client_id' => 'client-id',
        'redirect_uri' => 'http://example.com/callback',
        'response_type' => 'code',
        'scope' => '',
    ]);

    return redirect('http://your-app.com/oauth/authorize?'.$query);
});
```

Burada dikkat edilmesi gereken nokta şudur: Yukarıdaki kodlar consume yapan Laravel projesinde kullanılmalıdır, authentication sağlayan projede değil!

Bir diğer dikkat edilmesi gereken durum şudur: **redirect_uri** değeri yukarıda client oluştururken kullanılan redirect değeri ile **AYNI** olmalıdır.

Consume yapan Laravel projesinde /redirect routasına GET isteğinde bulunduğunda your-app.com sitesinde bir template gösterilir. Bu template güncellemek istiyorsak `php artisan vendor:publish --tag=passport-views` komutu çalıştırılıp, bu komut ile ortaya çıkan `resources/views/vendor/passport` klasörü içerisindeki authorize.blade.php dosyası güncellenir. Kullanıcı bu template yer alan Approve butonuna tıkladığı zamana uthorization code redirect_uri'de belirtilen adrese GET isteği ile otomatik iletilir. Bu adreste aşağıdaki gibi bir kod ile authorization code access token'lara dönüştürülmesi sağlanmalıdır:

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

```
    $http = new GuzzleHttp\Client;

    $response = $http->post('http://your-app.com/oauth/token', [
        'form_params' => [
            'grant_type' => 'authorization_code',
            'client_id' => 'client-id',
            'client_secret' => 'client-secret',
            'redirect_uri' => 'http://example.com/callback',
            'code' => $request->code,
        ],
    ]);

    return json_decode((string) $response->getBody(), true);
});
```

Bu işlemden sonra your-app.com sitesi `access_token`, `refresh_token` ve `expries_in` değerlerine sahip JSON sonucu dönderir.

Not: `/oauth/authorize` rotasında olduğu gibi, `/oauth/token` rotası da `Passport::routes()` metodu ile otomatik olarak kayıt edilmiştir.

Not: `'grant_type' => 'authorization_code'` yerine `'grant_type' => 'refresh_token',` denildiği zaman refresh token ve access token yenilenmiş olur.

Not: **AuthServiceProvider** sınıfının **boot()** metodu içerisine **Passport**::enableImplicitGrant() metodu eklenirse, 'code' parametresini kullanmaya gerek kalmaz. Buna **implicit grant tokens** denir. Bu tür kullanım en çok Javascript veya mobile uygulamalar gibi client credentials güvenli bir şekilde kayıt edilemeyen platformlarda yaygındır.

Password Grant Token

Mobil uygulama gibi API client'lara kullanıcı adı / e-mail adresi ve şifre ile access token, refresh token ve expires_in değerlerinin dönderilmesini sağlar.
`php artisan passport:client --password` komutu çağrıldığı zaman oauth_clients tablosuna aşağıdaki gibi bir satır eklenir:

id	user_id	name	secret	redirect	personel_access_client	pas
1	null	mobil	PsgprbVQ12UPmjLhxEde23iChJA9CglneuhMpRB8	localhost	0	1

Bu işlemden sonra kullanıcı API aracılığı ile giriş yapmak istediği zaman önce **access_token** ve **refresh_token** bilgilerini aşağıdaki gibi sunucuya istek gönderek alması gerekmektedir:

```
$http = new GuzzleHttp\Client;

$response = $http->post('http://your-app.com/oauth/token', [
    'form_params' => [
        'grant_type' => 'password',
        'client_id' => 'client-id',
        'client_secret' => 'client-secret',
        'username' => 'taylor@laravel.com',
        'password' => 'my-password',
        'scope' => '',
    ],
]);

return json_decode((string) $response->getBody(), true);
```

Önemli Notlar:

- **client_id** parametresine yukarıdaki tabloda yer alan ilk sütun yani 1 değeri yazılmalıdır.
- **client_secret** parametresine secret sütunundaki değer yazılmalıdır.
- **username** ve **password** parametrelerine ise normal giriş yaparken kullanılan bilgiler yazılmalıdır.

Dönen örnek sonuç:


```
{
     "token_type": "Bearer",
     "expires_in": 1296000,
     "access_token":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6IjUwZmQ3NmI0ZGVjYmY1YjBlYWYzNWE5MWNjMWE3NTdjYWJjMzMzNjE
wYTM3ZWZiYmMyNTgxNTQyMGU3YzI4OWVkMDM0ODMyM2FkNGE3MTk0In0.eyJhdWQiOiI3IiwianRpIjoiNTBmZDc2YjRkZWNiZjV
iMGVhZjM1YTkxY2MxYTc1N2NhYmMzMzM2MTBhMzdlZmJiYzI1ODE1NDIwZTdjMjg5ZWQwMzQ4MzIzYWQ0YTcxOTQiLCJpYXQiOjE
0OTk1MTM3ODEsIm5iZiI6MTQ5OTUxMzc4MSwiZXhwIjoxNTAwODA5NzgxLCJzdWIiOiIyIiwic2NvcGVzIjpbXX0.v3Vk6pd-
tri4gBfWYN56xaDeBAJnJUYqwHFetdSeRNcoM9-F1jcjrbNdcaqo5cSyDlt9uDfeUu3VdYyyKXNs9FV0er90AF4VGiQwjaU-
YpYaFyDpJz-ETwoiF77VEvkuZuEdolrb9xwySbe_-
sljojsAiY9QX5yuXD2dXBNS57LpwlxID93Il29scmjBG_dr9xfrQddIXMpfoSI3Nl2mVscadNFhcibnSZXp81GWiXzZQsgaivRQM
aTBO34dW3ND1nswN8CLVWVi5rc0YF9gQeLkSwkvw7wWTgDqorp9XGvkojU6HhhlSn1yvNJtB0ewFFj0pG5nAXs5BOq5g4MqzbNWM
zWsFK7UCiLCS63tUU4l_2CBWMEmT-MEj-
1cA1JM8XKrejL1I8yn6dQaNe_3MNe5JjssW6rVtfCEn1tgjtY6akCjHymzcKV04TQmVnVHm6_WNUTe9HlZEcb3TA7GwRvq5SWxRr
YQlALLu1spIOnVfEd8mrGcVyGfmp0frWoluxKNkeVqcjLguV1KTd6jSZ3WJQVQUqZNHRWcWXSdQ2PvBQkodtp5AE1tR3_Qe3iw24
_FqZ2lAI32clnRS7rEO53PXcFqZubbrTCxm33T8MIzztcxVBdoEPyvCf6-Espyhno1BS_pktQFdtApJasuV4GV-
rq2zCQ7DhzFT8kce8g",
     "refresh_token":
"LhbuM/FQLKeEDWywI2V8r9RTINPSek86TnneDNu9E/M/fMz7VSqXKsL8AaiYJTa48/K/m0hog1+CbgNAqGYQ/ALaHXpv/sZ/YYO
Bwx/4k8rC8JC67FbxdJYriPtRFxOLCK2wj0P4fZhUvNDx6rYjOyR7lxe3hTj/u14IDGq3MSPpcITj9WssNRxTOD3CjK20za2uost
cOAr6bsphh0Yj6k56DZ+oWuEluFajzVQ3lcIYgWuRpCr4mLH5xXdkTkNP6sgvIDwfuX1M3rD3EqAW2Uk+0IH6YQsePSx1c3gmB3L
AyljqM2fXul4aLDer2qQ3oieTNQJJdP4qNrg6lTp8D8PoHD6QJV04G+qaJAxe4QFknFLHCO9zrtsPxOdrPt3sg/zscFY5TDHC5fe
5CZC3DsmkGFwIV3LWgeZS+xDPcD/G+9Uk79cEwyEBDQt8L30epI18C5+C8g/rLddOTlBbzuc3qHoYwc4VhA/RM9sM45dku5Jquhe
zqx7zWsoa/ZuAB/aaNfkTzj5jIpg5nlX1Putb8Fy527UCtRsQb8nO6p0ppukTq1oh3gVTJDzyTD0hcXLGh3K9PQgzaeobRxn6Vaj
gHRewCFVt3Q6LSiFlK8DSpkjJpndpSKfAMmBdYNXGf6cHEiU+b6J8u/tMiJkPUVBe21rvF9+rDdb8ZOM="
}
```


Not: `'scope' => '*'` şeklinde kullanıldığı zaman tüm scope'lara erişim izni verilmiş olur.

Bu işlemden sonra sunucuya istek gönderileceği zaman Authorization-Header parametresi aşağıdaki şekilde kullanılmalıdır:

```
$response = $client->request('GET', '/api/user', [
     'headers' => [
         'Accept' => 'application/json',
         'Authorization' => 'Bearer '.$accessToken,
     ],
]);
```

Client Credentials Grant Tokens

Makineden makineye authentication için uygundur. Cron job gibi işlemlerde tercih edilmektedir.

`grant_type='client_credentials'` değerine sahip olur:

```
$guzzle = new GuzzleHttp\Client;

$response = $guzzle->post('http://your-app.com/oauth/token', [
     'form_params' => [
         'grant_type' => 'client_credentials',
         'client_id' => 'client-id',
         'client_secret' => 'client-secret',
         'scope' => 'your-scope',
     ],
```

```
]);

echo json_decode((string) $response->getBody(), true);
```

Personal Access Tokens

Bazı durumlarda, kolaylık açısından kullanıcılar**a oauth redirect flow yapısını kullanmadan** access token verilmesi gerekebilir. Yukarıda `passport:install` komutu ile hem Personal Access Client isminde hem de Password Grant Client isminde olmak üzere iki tane client oluşturmuştuk. Sadece Personal Access Client oluşturmak için `php artisan passport:client --personal` komutu kullanılmalıdır. Bu komut oluşturulacak olan client'a hangi ismin verileceğini soracaktır. Bu isme göre access token oluşturma aşağıdaki gibi yapılmalıdır:

```
$user = App\User::find(1);

// Creating a token without scopes...
$token = $user->createToken('Token İsmi')->accessToken;

// Creating a token with scopes...
$token = $user->createToken('Token İsmi', ['place-orders'])->accessToken;
```

Passport servisi aynı zamanda yukarıdaki işlemler için JSON API de sağlamaktadır. Bu api ile ilgili metodlar aşağıdaki gibidir:

Method	Route	Tanım
GET	/outh/scopes	Uygulamada tanımlı tüm scope'ları dönderir
GET	/oauth/personal-access-tokens	Giriş yapan kullanıcının access token'larını dönderir. Bu sayede düzenlenecek veya silinecek tokenlara ulaşmış oluruz
POST	/oauth/personal-access-tokens	Yeni bir personel access token oluşturmak için kullanılır. Aldığı parametreler name ve scopes arrayidir
DELETE	/oauth/personal-access-tokens/{token-id}	Personel access token'ı silmek için kullanılır

Protecting Routes

```
Route::get('/user', function () {
    //
})->middleware('auth:api');
```

middleware metodu ile bir rotaya erişimin kontrolü sağlanır.

```
$response = $client->request('GET', '/api/user', [
    'headers' => [
        'Accept' => 'application/json',
        'Authorization' => 'Bearer '.$accessToken,
    ],
]);
```

Yukarıdaki kodlarda görüldüğü gibi Bearer ile başlayan Authorization header ile access token alındıktan sonra istekler yapılır.

Nedir Bu Scope?

Yukarıda bir çok örnekte scopes parametresi kullanılmıştır. Bu parametre aslında permission gruplama için faydalı olmaktadır. Örneğin bazı client'ların bir e-ticaret sitesinde sipariş vermesini engellemek için kullanılabilir. Özetle scope değerleri klasik kullanıcı yetkilendirme işleminlerinde kullanılan ROL tanımlaması ile aynı mantıkta kullanılmaktadır. Bu sayede third-party uygulama sunucuda her isteği yapamayacaktır.

AuthServiceProvider sınıfının **boot()** metodunda scope tanımlama işlemi yapılmaktadır:

```
use Laravel\Passport\Passport;

Passport::tokensCan([
     'place-orders' => 'Place orders',
     'check-status' => 'Check order status',
]);
```

Tanımlama işlemi bittikten sonra iki farklı yolla scope'lar tokenlara assign edilebilir:

İlk olarak klasik redirect flow işlemi ile scope parametresine bir veya birden fazla scope arada boşluk karakteri koyarak eklenir:

```
Route::get('/redirect', function () {
     $query = http_build_query([
         'client_id' => 'client-id',
         'redirect_uri' => 'http://example.com/callback',
         'response_type' => 'code',
         'scope' => 'place-orders check-status',
     ]);

     return redirect('http://your-app.com/oauth/authorize?'.$query);
});
```

İkinci yol ise Personel access token oluştururken assign işlemidir:

```
$token = $user->createToken('My Token', ['place-orders'])->accessToken;
```

Scope Kontrolü

`app/Http/Kernel.php` dosyasında yer alan $routeMiddleware property'e aşağıdaki **middleware** sınıfları eklenmelidir:

```
'scopes' => \Laravel\Passport\Http\Middleware\CheckScopes::class,
'scope' => \Laravel\Passport\Http\Middleware\CheckForAnyScope::class,
```

Birden fazla scope kontrol etmek için aşağıdaki gibi kullanılır:

```
Route::get('/orders', function () {
     // Access token has both "check-status" and "place-orders" scopes...
})->middleware('scopes:check-status,place-orders');
```

Herhangi bir scope kontrol etmek için ise aşağıdaki gibi kullanılmalıdır:

```
Route::get('/orders', function () {
     // Access token has either "check-status" or "place-orders" scope...
})->middleware('scope:check-status,place-orders');
```

Bu ikisi arasındaki tek fark 's' karakteridir. Dikkat edilirse **middleware()** metodunda scopes ve scope ile başlanmıştır.

Bir üçüncü yöntem ise aşağıdaki gibidir:

```
use Illuminate\Http\Request;

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

```
    if ($request->user()->tokenCan('place-orders')) {
        //
    }
});
```

Javascript ile API'yi Kullanmak

Bir API geliştirildiği zaman JavaScript uygulamalar tarafından kolay bir şekilde consume edilmesi çok büyük fayda sağlamaktadır. Bu sayede API'yi kolay bir şekilde dünya ile paylaşmış oluruz. Çünkü javascript hemen hemen her platformda kullanılmaktadır.

Normalde, API'yi JavaScript ile kullanmak için manuel olarak access token'ı uygulamaya göndermeli ve her istek yapılırken bu tokenı kullanmalıyız. Bu işlemi kolay bir şekilde yapılabilmesi için Passport CreateFreshToken **middleware** sağlamıştır. Bu **middleware**'i web **middleware** grubuna aşağıdaki gibi eklemeliyiz:

```
'web' => [
    // Other middleware...
    \Laravel\Passport\Http\Middleware\CreateFreshApiToken::class,
],
```

Bu işlemden sonra Passport, outgoing response'lara laravel_token cookie değeri ekler. Bu cookie encrypted JWT değeri içerir. Laravel authentication işlemlerini bu cookie değerine göre yapar. Bu sayede JavaScript tarafında extra olarak access token göndermeye gerek yoktur.

```
axios.get('/api/user')
    .then(response => {
        console.log(response.data);
    });
```

Yukarıda görüldüğü gibi Axios JavaScript framework ile GET isteği access token olmadan yapılmaktadır. Burada dikkat edilmesi gereken bir başka husus şudur: Axios framework otomatik olarak X-CSRF-TOKEN bilgisini otomatik olarak göndermektedir. Eğer Axios yerine farklı bir JavaScript framework kullanılacaksa X-CSRF-TOKEN gönderecek şekilde konfigüre edilmesi gerekir.

Events

Passport access token ve refresh token oluşturulurken event'lar üretir. **EventServiceProvider** sınıfında aşağıdaki gibi tanımlama yaparak bu eventları yakalayabiliriz. Bu sayede veritabanında tutulan tokenları silme vb gibi işlem yapılabilir.

```
/**
  * The event listener mappings for the application.
  *
  * @var array
  */
protected $listen = [
    'Laravel\Passport\Events\AccessTokenCreated' => [
        'App\Listeners\RevokeOldTokens',
    ],

    'Laravel\Passport\Events\RefreshTokenCreated' => [
        'App\Listeners\PruneOldTokens',
    ],
];
```