Chapitre 7
Authentification

Plan

- Une petite note sur le TLS
- Authentification HTTP
- Cookies et sessions
- OAuth

Authentification…

L'**authentification** et l'**autorisation** sont deux choses différentes.

Voir http://stackoverflow.com/questions/6556522/authentication-versus-authorization

Important de bien faire la distinction entre les deux… le cours et les laboratoires porteront principalement sur l'**authentification**.

Authentification - pourquoi

Vous aurez probablement à développer un jour une application nécessitant une certaine **gestion des usagers**. Certaines parties de l'application pourraient être restreintes à un usager connu.

NE PAS TENTER DE RÉINVENTER LA ROUE.

L'internet est une jungle, les failles de sécurité sont nombreuses, et vous n'êtes pas des experts.

Authentification - pourquoi

À retenir: **Ne stockez pas d'informations sensibles si ce n'est pas nécessaire**.

Par information sensible, on parle au plus simple de **mot de passe**. Toute information de vie privée également.

Si vous **devez** stocker ce genre d'informations, assurez vous de les **hasher**.

Les noms d'usagers/emails et les tokens ayant une durée de vie limitée ne sont pas des informations sensibles.

Authentification - pourquoi

- Utilisez des *third-party* pour la gestion des utilisateurs (nous allons voir dans le chapitre sur l'OAuth)
- Facile de stocker des tokens, durée de vie limitée et facile à *revoker* (vous entendrez aussi parler d'**API Key**)
- Le frontend devrait manipuler le moins de données possibles (tokens only)
- Intégrez vous avec Stripe ou Paypal si vous devez manipuler de l'argent.

TLS

Toutes les notions que nous verrons impliquent que les communications HTTP transitent par un canal **sécurisé**.

→ **TLS (Transport Layer Security)** et son prédécesseur SSL est le protocole permettant d'encrypter les communications lors d'un échange HTTP.

- SSL 1, 2 et 3 sont *deprecated*
- TLS 1.0 et 1.1 sont pratiquement *deprecated*
- TLS 1.2 est la norme courante
- TLS 1.3 arrive

TLS

Basé principalement sur un échange de **clés** entre le serveur et le client. L'échange se fait par la technique de cryptographie asymétrique.

- Les clés sont contenues dans des **certificats**, et le résultat est la **clé de session** partagée par les deux interlocuteurs.

- Importance de protéger les **certificats**!

RFC: https://www.ietf.org/rfc/rfc5246.txt

TLS

À retenir:

- Votre application **devrait** être exposée en HTTPS publiquement. Ça demande un peu plus de complexité, mais c'est un *quick win*.

- Si vous exposez un API qui demande un quelconque échange de données **sensibles** (ou même de simples emails), pas d'excuses! Vous **devez** être en HTTPS. Sinon, c'est votre faute...

TLS

Une dernière petite note:

- **Utiliser le *least privilege principle* : assurez vous que les seules parties exposées de votre application sont celles qui sont nécessaires.**

Votre API devrait être le seul *endpoint* accessible de l'extérieur… encore aujourd'hui, des millions de bases de données sont accessibles publiquement!

Ouvrez seulement les ports donc vous avez besoin (443)

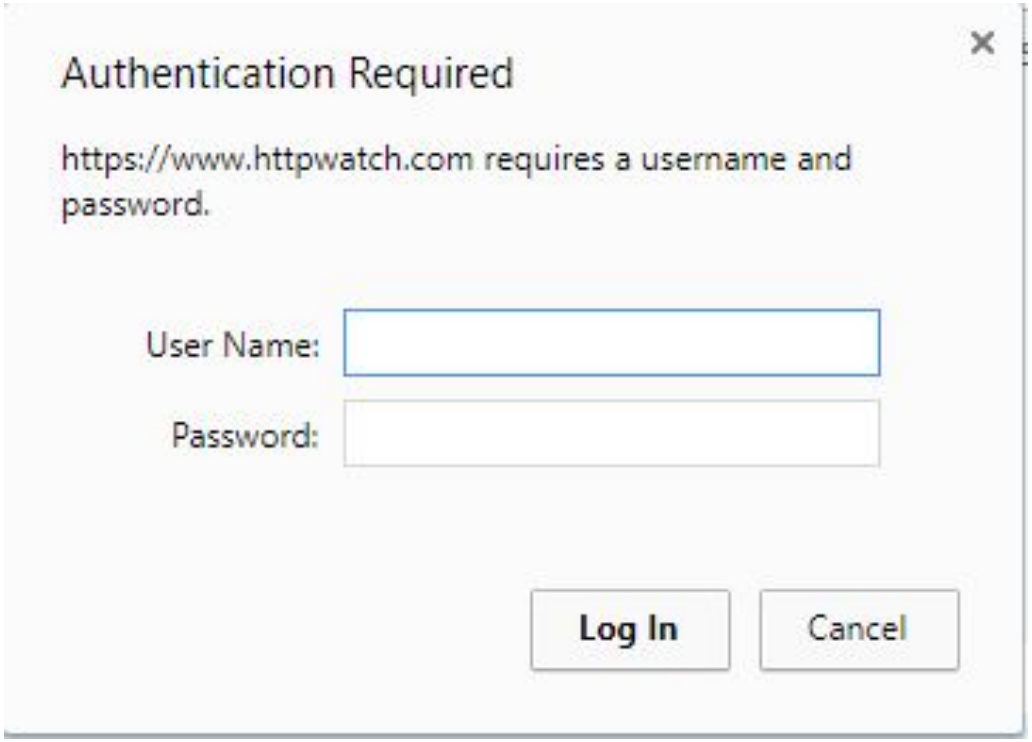
Authentification HTTP

Il existe deux modes d'authentification HTTP:

- **Basic** authentication
- **Digest** authentication

Le basic auth est le plus simple, mais donne de bons résultats en combinaison avec **TLS**. Le digest auth n'est pratiquement plus utilisé et nous n'en parlerons pas.

Basic Authentication



Basic Authentication

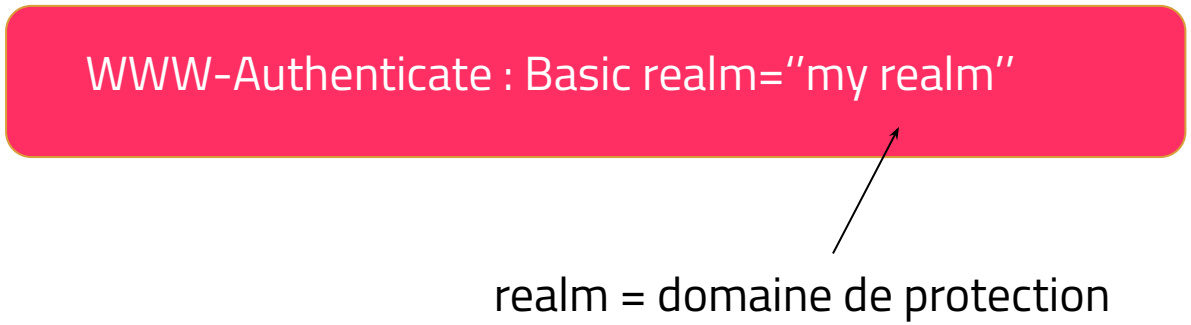
Forme d'authentification la plus simple: lors d'un accès à une ressource protégée, le serveur répond avec un **entête** spécifique, et le client n'a qu'à fournir ses **username**/**password** encodés.

- Doit absolument être utilisé en combinaison avec TLS.
- Le serveur est responsable de valider la combinaison username/password.
- Sera utilisé dans un des flots OAuth que nous verrons bientôt...

Basic Authentication

Côté serveur :

→ Le serveur demande une authentification avec un **header** **WWW-Authenticate** :



Le **realm** est important → il définit ce à quoi l'usager aura accès.

Basic Authentication

Côté client :

→ Le client répond avec un **header** **Authorization**:

Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Le contenu est composé d'une **chaîne de caractères** sous forme **username:password**, encodée en **Base64**.

Sessions

Qu'est-ce qu'une **session** ?

Un **dialogue** semi-permanent établi entre un serveur et un client quelconque.

Mais encore?

- Une des parties de la communication doit sauvegarder des **informations** sur la session afin de la maintenir.
- Peut être soit le **client**, soit le **serveur**, dépendamment de l'implémentation.
- Le concept de **session** peut être implémenté de 1000 façons différentes, mais généralement basé sur un **token** quelconque.

Sessions

- La session repose sur un **identifiant** ou **token** négocié entre le client et le serveur lors de l'authentification.
- Ce token expire généralement après une période de temps X et doit être renégocié.
- Le token est généré par une fonction de hachage quelconque qui doit garantir **l'unicité** en tout temps.
- Stockés *server-side* et généralement mis en cache. Le token est validé sur chaque requête faite au serveur.

Sessions

À garder en tête :

1. **Confidentialité** : Seulement le serveur doit être capable d'interpréter toute information reliée à la session.
2. **Intégrité** : Seulement le serveur devrait manipuler de l'information reliée à la session.
3. **Authenticité** : Seulement le serveur doit être capable de valider la session.

Un petit exemple concret…

Cookies

Qu'est-ce qu'un **cookie** ?

→ **Information** stockée dans le **navigateur**, de manière persistante ou non.

Généralement un simple fichier texte!

Cookies

Principales utilisations :

1. **Sessions**
   Sauvegarde l'information sur **l'authentification** de l'utilisateur.

2. **Personnalisation (traditionnellement)**
   Sauvegarde des informations sur les **préférences** utilisées sur un site.

3. **Tracking**
   Sauvegarde des informations sur les **visites** d'un utilisateur, à des fins de statistiques.

Cookies ou local storage?

Très bien résumé ici → https://stackoverflow.com/questions/3220660/local-storage-vs-cookies

- Local storage est utilisé pour les informations purement *client-side* (donc… la **majorité**)
- Les informations de session (**token**) devraient être dans des cookies qui ont une date d'expiration - chose que le local storage n'a pas

Cookies

Exemple sur Twitter :

Name	Value	Domain	Path	Expires / Max-...	Size	HTTP	Secure
BA	ba=1197583&be=128758.05&l=109&le=0.52&ip=184.163.216.107&t=1375233010	.twitter.com	/	Wed, 07 Aug ...	71		
__utma	43838368.8243669.1325005651.1375831529.1375834169.967	.twitter.com	/	Fri, 07 Aug 20...	59		
__utmb	43838368.1.10.1375834169	.twitter.com	/	Wed, 07 Aug ...	30		
__utmc	43838368	.twitter.com	/	Session	14		
__utmv	43838368.lang%3A%20fr	.twitter.com	/	Fri, 07 Aug 20...	27		
__utmz	43838368.1374894452.935.20.utmcsr=google|utmccn=(organic)|utmcmd=organic|u...	.twitter.com	/	Wed, 05 Feb 2...	102		
_twitter_sess	BAh7CjoJdXNlcmkEyD3iEjoMY3NyZI9pZCIlYjJkMzQwNWRhZGUxYjU2Njg4%250ANzY5...	.twitter.com	/	Session	282	✓	✓
activity_modal_dismissed	true	twitter.com	/	Fri, 24 Dec 20...	28		
auth_token	66ae148609b09401b532c49c2befa0647f45300b	.twitter.com	/	Sat, 27 Dec 20...	50	✓	✓
guest_id	v1%3A132500497353442621	.twitter.com	/	Fri, 27 Dec 20...	31		
lang	fr	twitter.com	/	Session	6		
original_referer	padhuUp37zi4XoWogyFqcGgJdw+JPXpx	twitter.com	/	Session	48		
pid	v3:13750215677742394434465 60	.twitter.com	/	Tue, 27 Jan 20...	31		
secure_session	default	.twitter.com	/	Sat, 27 Dec 20...	21		
twid	u%3D316816840%7CFeWsO6DCWpt2%2BubPs0h0xVhSI0I%3D	.twitter.com	/	Session	52		✓
twll	l%3D1325005664	.twitter.com	/	Mon, 27 Dec 2...	18		

Champs principaux :

- Nom
- Valeur
- Domaine
- Chemin
- Date d'expiration / Âge maximum
- Taille

HttpOnly
(non-Javascript)

Secure
(HTTPS)

Cookies

Exemples **JavaScript**: (avec le plugin **js-cookie**)

Voir : https://github.com/js-cookie/js-cookie

- Création d'un cookie

```
Cookies.set('name', 'value');
```

- Création d'un cookie avec paramètres

```
Cookies.set('name', 'value', { expires: 7 });
```

Cookies

Exemples **JavaScript**: (avec le plugin **js-cookie**)

Voir : https://github.com/js-cookie/js-cookie

- Récupérer la valeur d'un cookie

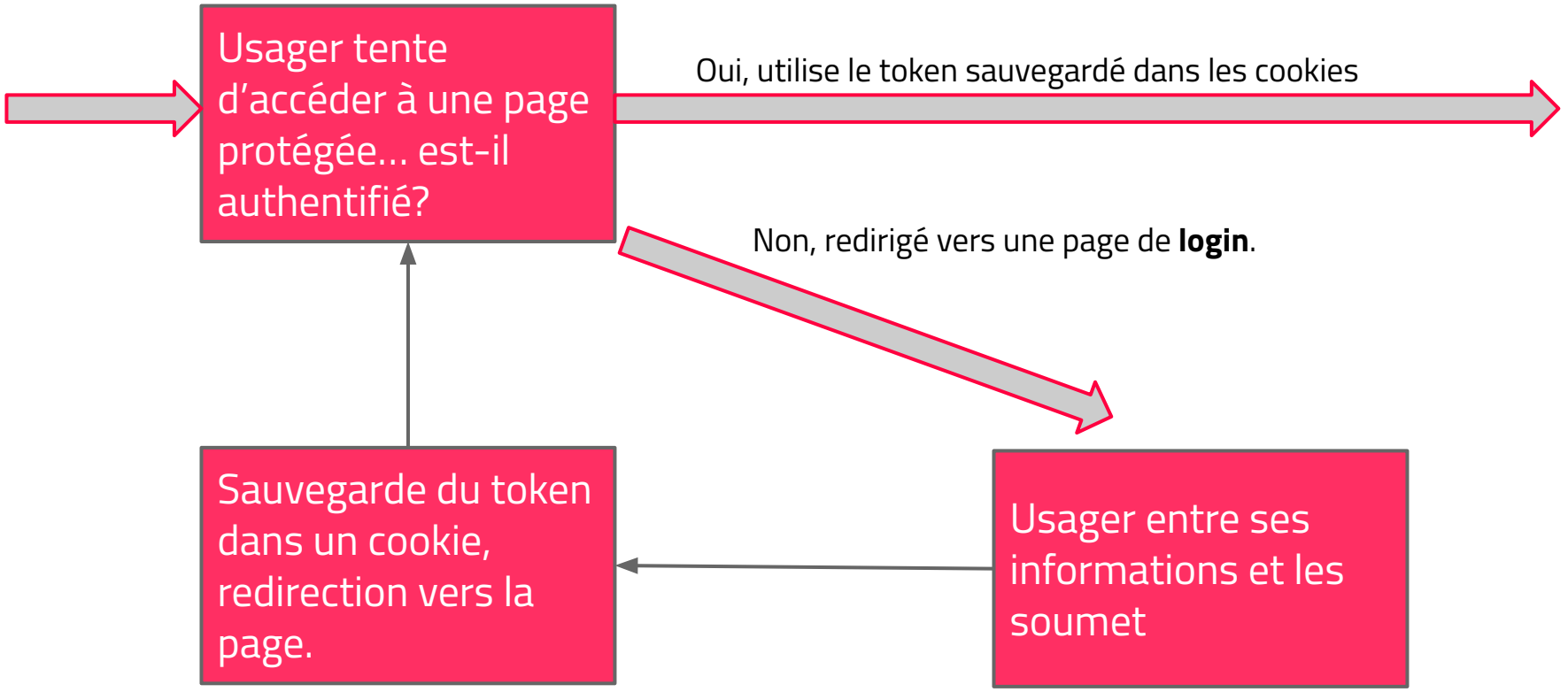
  Cookies.get('name');

- Suppression d'un cookie

  Cookies.remove('name');

Cookies

Flot typique d'authentification avec des **cookies** :



Cookies

Différence entre cookie **client-side** et cookie **server-side:**

Ce qu'on appelle traditionnellement cookie "**server-side**" est également un synonyme de **session**. L'information est utilisée à travers les requêtes HTTP… alors qu'un cookie "**client-side**" n'est typiquement qu'utilisé par le navigateur.

Voir
http://stackoverflow.com/questions/6922145/what-is-the-difference-between-server-side-cookie-and-client-side-cookie

OAuth

Qu'est-ce qu'OAuth?

Open Standard for **Authorization**.

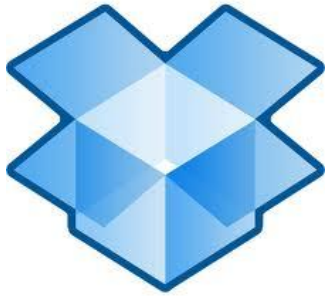
→Permet l'authentification à une API de manière **sécurisée**, sans que l'application n'aille à manipuler d'informations sensibles directement.

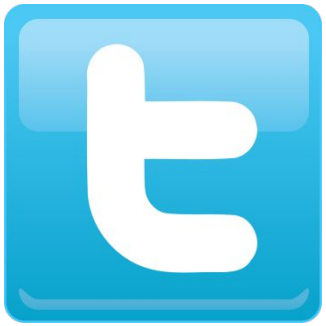
2010 : OAuth 1.0

2012 : OAuth 2.0 : critiqué et **abandonné** par son créateur… pour finalement devenir la norme.

OAuth

Fournisseurs OAuth communs:





PayPal™

OAuth

La base…

- Le standard OAuth permet de déléguer l'authentification à un **fournisseur** externe. Le contrat du fournisseur est de retourner à l'application si l'usager peut y accéder ou non.

- L'application n'a donc jamais à manipuler de nom d'usager ou de mot de passe…

- L'authentification nécessite de recueillir un **jeton d'accès** pour ensuite l'utiliser sur les requêtes d'API.

OAuth

Quelques notions

- **Client ID (Consumer key)**
  Le client id est ce qui identifie votre application auprès du fournisseur OAuth. C'est une information de nature publique (mais pas trop…)

- **Client Secret**
  Le client secret identifie également votre application auprès du fournisseur… par contre il s'agit de la partie **privée** (NE PAS PARTAGER)

OAuth

Différent **flots** d'authentification:

- **Authorization code**
- **Implicit**
- **Password**
- **Client credentials**

OAuth

Flot: Authorization code

Ce flot est typiquement utilisé par une application désirant s'authentifier avec une autre application côté serveur.

L'étape 0 est typiquement d'**enregistrer** votre application auprès du fournisseur OAuth, afin d'obtenir un **client id** et un **client secret**, et réserver un id unique.

Le flot sera majoritairement effectué *server-side*.

OAuth

Flot: Authorization code

1. L'utilisateur désire s'authentifier. L'API s'occupe de rediriger vers le fournisseur OAuth. L'application doit passer son **client id** et un **redirect uri**. Également possible de passer des **scopes...**(privilèges demandés).

https://oauth2server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos

OAuth

Flot: Authorization code

2. L'usager est redirigé vers la page de login du fournisseur. Si le login fonctionne, le fournisseur redirige au **redirect URI** en lui passant un **code d'authorization**. Le redirect URI est typiquement une autre route de votre API.

https://monserveur.com/cb?code=AUTH_CODE_HERE

OAuth

Flot: Authorization code

3. Lors de la redirection, votre application utilise son **client id**, **client secret** et le code d'authorization afin de négocier un **jeton d'accès**.

POST https://api.oauth2server.com/token

grant_type=authorization_code&code=AUTH_CODE_HERE&edirect_uri=REDIRECT_URI&client_id=CLIENT_ID&

client_secret=CLIENT_SECRET

OAuth

Flot: Authorization code

4. Si l'authentification est réussie, le **jeton d'accès** est retourné et il est maintenant possible d'effectuer des requêtes sur l'API.

Le flot est assez transparent pour l'usager, il n'a qu'à entrer ses informations auprès du fournisseur. Le jeton est ensuite utilisé par le code de l'application *client-side*.

OAuth

Flot: Implicit

Ce flot est utilisé lorsque l'authentification doit être faite dans le navigateur

- Utilisé lors de possibilité d'exposition d'informations sensibles
- Le fournisseur doit supporter ce genre de flot…
- Une étape de moins: suffit de passer le **client id** et le **redirect URI**, votre application est directement redirigée avec son **jeton d'accès**.

https://oauth2client.com/cb?token=ACCES_TOKEN

OAuth

Flot: Password

Le fournisseur OAuth peut également supporter un flot permettant de s'authentifier avec un nom d'usager/mot de passe.

- Comme le flot **d'authorization code**, ne devrait qu'être utilisé côté serveur.
- Le **client ID** suffit afin d'obtenir un **jeton d'accès**.

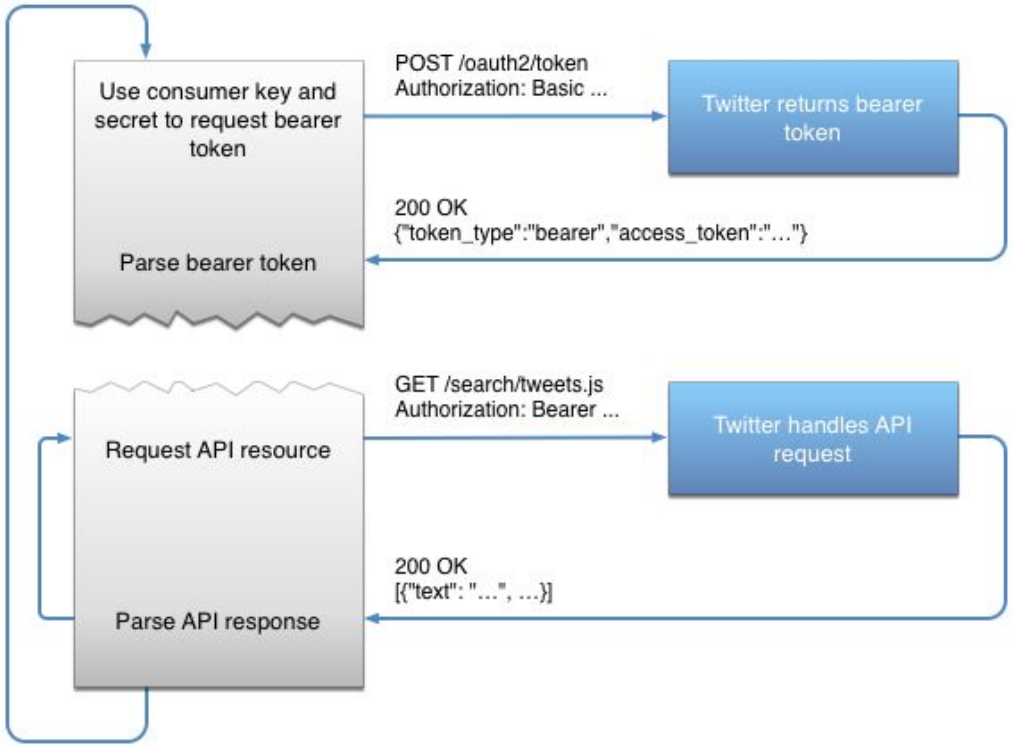
POST https://api.oauth2server.com/token

grant_type=password&username=USERNAME&password=PASSWORD&client_id=CLIENT_ID

OAuth

Exemple basé sur Twitter, flot **Client credentials** (100% *server-side*)

(voir https://developer.twitter.com/en/docs/basics/authentication/overview/application-only)

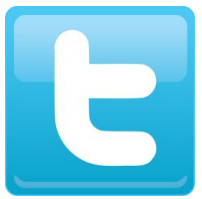


OAuth

1. **Encodage** en Base64 de la concaténation du '**consumer key**' et du '**consumer secret**'... fournis lors de l'enregistrement de l'application

Consumer key	xvz1evFS4wEEPTGEFPHBog
Consumer secret	L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg
Bearer token credentials	xvz1evFS4wEEPTGEFPHBog:L8qq9PZyRg6ieKGEKhZolGC0vJWLw8iEJ88DRdyOg
Base64 encoded bearer token credentials	eHZ6MWV2RlM0d0VFUFRHRUZQSEJvZzpMOHFxOVBaeVJnNmlIS0dFS2hab2xHQzB2SldMdzhpRUo4OERSZHlPZw==

Voir démonstration !



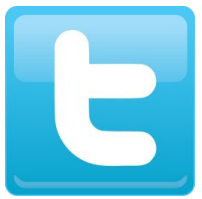
OAuth

2. Envoi d'une requête **POST** afin d'obtenir le **jeton d'authentification** (**bearer token**)

```
POST /oauth2/token HTTP/1.1
Host: api.twitter.com
User-Agent: My Twitter App v1.0.23
Authorization: Basic eHZ6MWV2RlM0d0VFUFRHRUZQSEJvZzpMOHFxOVBaeVJn
                     NmllS0dFS2hab2xHQzB2S1dMdzhpRUo4OERSZHlPZw==
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Content-Length: 29
Accept-Encoding: gzip

grant_type=client_credentials
```

Valeur reçue de l'étape 1



OAuth

2 (suite). Twitter retourne le **token** valide pour une période X...


```
{
"token_type":"bearer",
"access_token":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%2FAAAAAAA
AAAAAAAAAAAAA%3DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
}
```


OAuth

3. Les **requêtes** peuvent maintenant s'effectuer avec le **token**

```
GET /1.1/statuses/user_timeline.json?count=100&screen_name=twitterapi HTTP/1.1
Host: api.twitter.com
User-Agent: My Twitter App v1.0.23
Authorization: Bearer AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%2FAAAAAAAAAAAAA
                      AAAAAAAA%3DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Accept-Encoding: gzip
```

Valeur reçue de l'étape 2

OAuth

À considérer:

- Votre application peut devenir un fournisseur OAuth si vous le désirez. Elle devra respecter la **RFC**: https://tools.ietf.org/html/rfc6749

- Typiquement, votre application utilisera un fournisseur OAuth externe afin de s'authentifier. (Google, Twitter etc.)