

# API

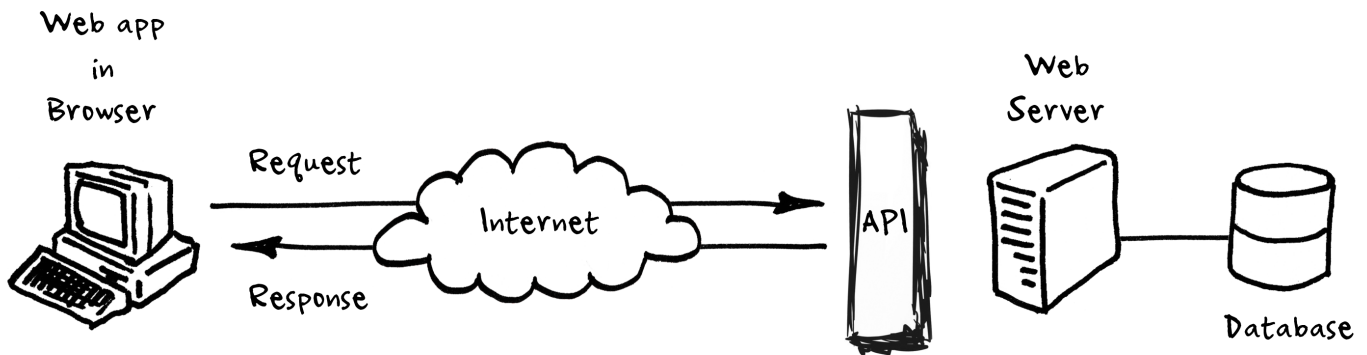
## What is an "API" ?

Le terme "API" est très générique et peut désigner bien des choses, mais dans le jargon on l'utilise souvent pour désigner un service web qui renvoie non pas:

des pages web au format HTML (destinées à être lues par un humain dans son navigateur)

mais:

des données au format JSON (destinées à être traitées par un programme)



Puisque les API sont dédiées à l'usage via des programmes, elles disposent en général d'une bonne documentation, et sont fiables et stables dans le temps. Tandis que sur des pages web HTML classiques, le design peut par exemple changer du jour au lendemain et rendre votre programme BeautifulSoup obsolète.

## Exemple: deezer (site web) VS deezer API

Récupérer le nombre de fans d'un artiste:

## Exemple: deezer (site web) VS deezer API

Récupérer le nombre de fans d'un artiste:

Entrée [ ]:

```
# Site web
response = requests.get("https://www.deezer.com/fr/artist/939")
soup = BeautifulSoup(response.text)
nb_fans = int(soup.find('div', id='naboo_artist_social_small').span.text)
```

Entrée [ ]:

nb\_fans

Entrée [ ]:

```
# API JSON
response = requests.get("https://api.deezer.com/artist/939")
data = json.loads(response.text)
nb_fans = data['nb_fan']
```

Entrée [ ]:

```
nb_fans
```

Encore mieux: il y a un module sur pypi pour accéder à l'api de deezer encore + facilement:

<https://pypi.org/project/deezer-python/> (<https://pypi.org/project/deezer-python/>)

Entrée [ ]:

```
import deezer
c = deezer.Client()
nb_fans = c.get_artist(939).nb_fan
```

Entrée [ ]:

```
nb_fans
```

Moralité: ne réinventez pas la roue, utilisez des modules déjà existant, ou utilisez l'API JSON officielle si ceux ci sont disponibles.

## Web scraping vs using the API vs using a python *client*

Entrée [ ]:

```
%%time
# Using web scraping from https://www.deezer.com

import requests
from bs4 import BeautifulSoup

response = requests.get("https://www.deezer.com/fr/artist/939")
soup = BeautifulSoup(response.text, 'html.parser')
nb_fans = int(soup.find('div', id='naboo_artist_social_small').span.text)

nb_fans
```

### Inconvénients du web scraping:

- plutôt lent (car on parse potentiellement beaucoup de HTML inutile)
- ne donne pas les résultats attendus si une partie du contenu est intégré dynamiquement à la page via javascript
- un changement dans l'architecture du html ou du css (e.g: refonte du design du site) oblige à réécrire le programme

Entrée [2]:

```
%%time
# Using deezer's public JSON API https://api.deezer.com

import requests

response = requests.get("https://api.deezer.com/artist/939")
data = json.loads(response.text)
nb_fans = data['nb_fan']

nb_fans
```

CPU times: user 24 ms, sys: 4 ms, total: 28 ms  
Wall time: 411 ms

Out[2]:

183723

### Avantages d'une API

- renvoie du format JSON, facile et rapide à traiter
- renvoie un format stable et documenté (voire versionné): <https://developers.deezer.com/api>  
(<https://developers.deezer.com/api>)
  - la documentation indique comment interagir avec l'API:
    - quelle url
    - quelle méthode http (GET, POST, ...)
    - quels paramètres
    - ...

→ idéal pour les développeurs

Entrée [3]:

```
%%time
# Using "deezer-python" external package (which itself uses deezer API)

import deezer # needs to be installed (pip install deezer)

c = deezer.Client()
nb_fans = c.get_artist(939).nb_fan

nb_fans
```

CPU times: user 40 ms, sys: 8 ms, total: 48 ms  
Wall time: 393 ms

Out[3]:

183724

→ une fois qu'un service comme deezer expose une API, il devient "facile" pour un dev python de réaliser un tel module (aka: *api client*) qui abstrait complètement la partie requêtes http.

### Quel intérêt pour le fournisseur d'API ?

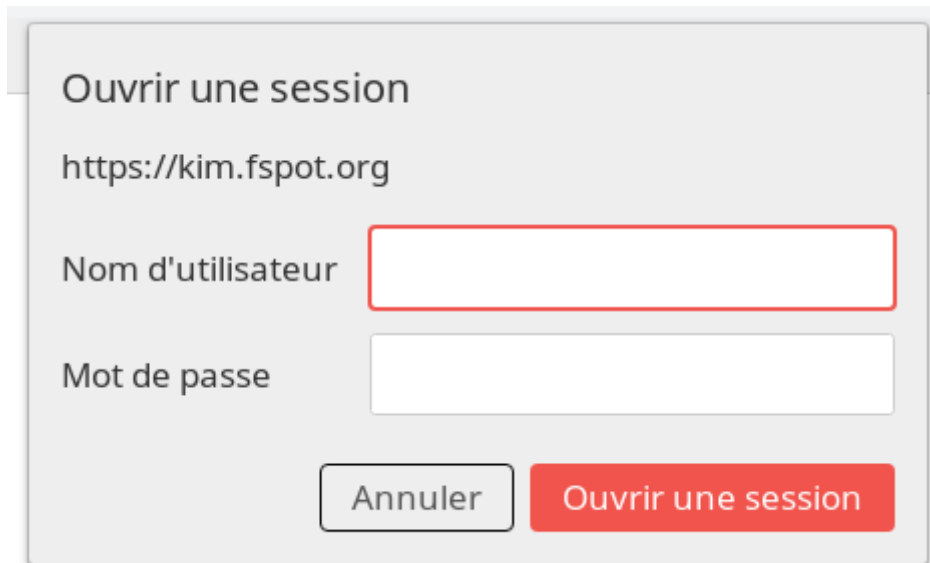
En général il met en place des quotas de requêtes ou d'autres limitations afin de proposer un service payant qui dispose de possibilités avancées / d'un meilleur support / etc.

C'est pourquoi de nombreux services nécessitent de se connecter avec son compte client pour utiliser une API (e.g <https://openweathermap.org> (<https://openweathermap.org>))

*(l'autre intérêt de se connecter est simplement de pouvoir accéder à ses données privées: e.g je veux faire un programme python qui me donne la liste de mes tweets mais mon compte twitter est protégé)*

## Basic Auth

Exemple: accéder à <https://kim.fspot.org/private/> (<https://kim.fspot.org/private/>) affiche:



Ouvrir une session

<https://kim.fspot.org>

Nom d'utilisateur

Mot de passe

Pour y accéder il est nécessaire d'utiliser les credentials suivant:

- login: admin
- password: secret

Si on ne les passe pas (ou si on ne passe pas les bons), on a une erreur 401 (= unauthorized).

Entrée [1]:

```
res = requests.get('https://kim.fspot.org/private')
res
```

Out[1]:

<Response [401]>

Entrée [58]:

```
# Dans les headers (metadata) de la réponse on peut voir
# qu'il faut faire une auth "Basic":
res.headers
```

Out[58]:

```
{'Server': 'nginx/1.10.3', 'Date': 'Wed, 02 Oct 2019 21:50:28 GMT', 'Content-Type': 'text/html', 'Content-Length': '195', 'Connection': 'keep-alive', 'WWW-Authenticate': 'Basic realm="Restricted"}
```

[Basic Auth \(wikipedia\)](#)

([https://fr.wikipedia.org/wiki/Authentification\\_HTTP#M%C3%A9thode\\_%C2%AB\\_Basic\\_%C2%BB](https://fr.wikipedia.org/wiki/Authentification_HTTP#M%C3%A9thode_%C2%AB_Basic_%C2%BB)) : il faut passer un header `Authorization` avec la valeur `Basic XXX` en remplaçant `XXX` par les credentials `username:password` encodés en base64:

Entrée [12]:

```
from base64 import b64encode

credentials = 'admin:secret'
encoded = b64encode(credentials.encode())

print(encoded.decode())

YWRtaW46c2VjcmV0
```

→ On réessaye la requête avec le bon header:

Entrée [74]:

```
headers = {'Authorization': 'Basic YWRtaW46c2VjcmV0'}
res = requests.get('https://kim.fspot.org/private', headers=headers)
res
```

Out[74]:

<Response [200]>

Entrée [76]:

```
# En réalité on s'épargne de faire ça à la main, requests peut le faire pour nous:
res = requests.get('https://kim.fspot.org/private', auth=('admin', 'secret'))
res
```

Out[76]:

<Response [200]>

## Auth par token

Exemple sur openweathermap:

- documentation: <https://openweathermap.org/appid> (<https://openweathermap.org/appid>)
- mes tokens: [https://home.openweathermap.org/api\\_keys](https://home.openweathermap.org/api_keys) ([https://home.openweathermap.org/api\\_keys](https://home.openweathermap.org/api_keys))

Avantage des tokens:

- évite que les requêtes HTTP contiennent le mot de passe - à la place elles contiennent un token
- si je me fais "voler" un token, je peux le supprimer de mon compte
- certains services fournissent des token plus ou moins limités: ainsi je peux accepter de prêter un token à quelqu'un d'autre si je sais qu'il ne pourra en faire qu'un usage restreint (e.g app facebook: voir mes infos de profil, pas publier des posts à ma place)

Entrée [13]:

```
url = "http://api.openweathermap.org/data/2.5/weather?APPID=515b9c16560819dfe610251"
res = requests.get(url)

res.json()['weather']
```

Out[13]:

```
[{'id': 800, 'main': 'Clear', 'description': 'clear sky', 'icon': '01d'}]
```

## OAuth

Il y a 15 ans:

### Are your friends already on Yelp?

Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service



Your Email Address

ima.testguy@gmail.com (e.g. bob@gmail.com)

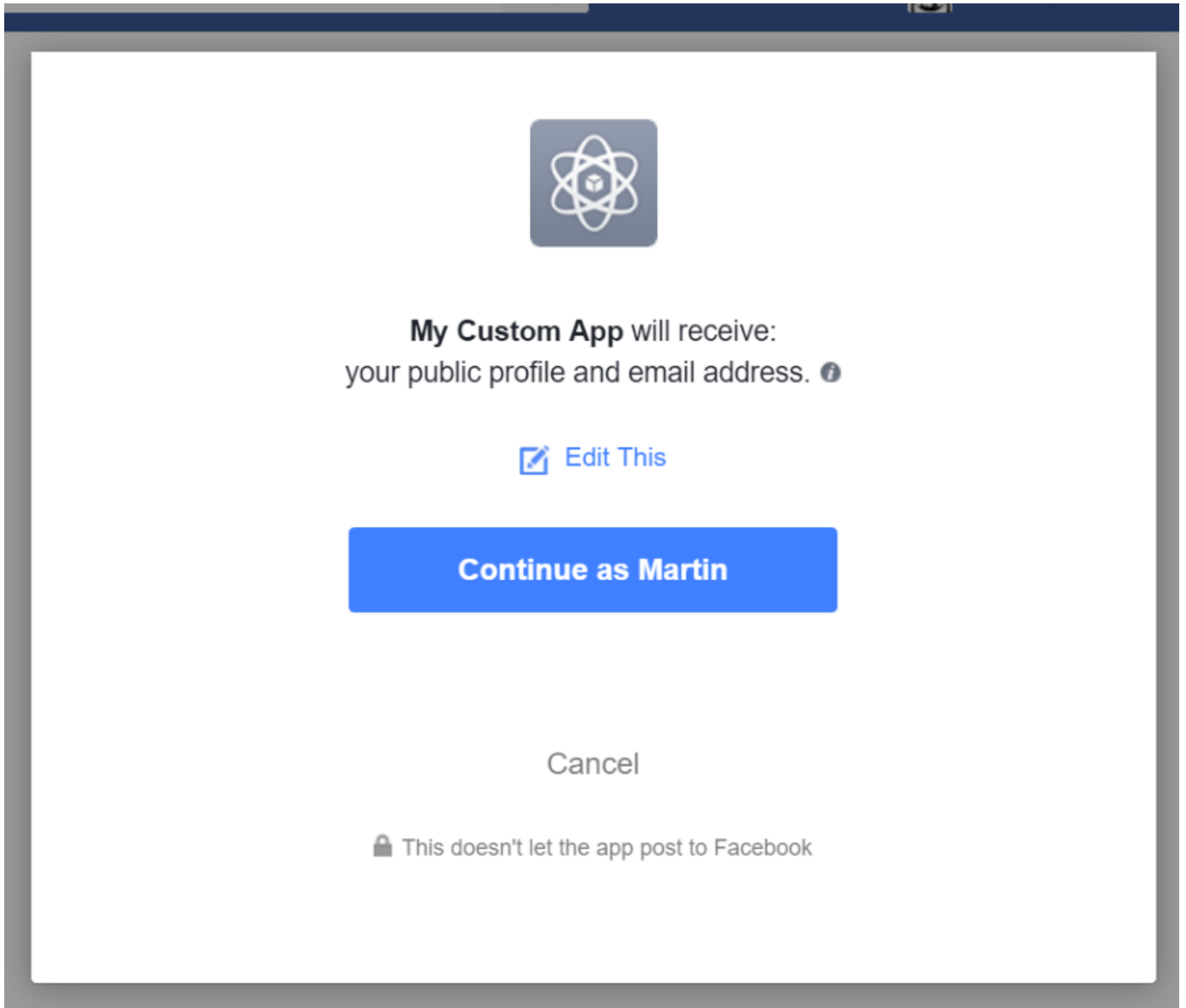
Your Gmail Password

•••••••••• (The password you use to log into your Gmail email)

[Skip this step](#)

**Check Contacts**

Maintenant:



OAuth est un standard très répandu pour gérer l'authentification car il permet le workflow ci-dessus (autoriser une app à accéder à une portion d'un service où vous êtes inscrit, e.g facebook) dans le browser. Par conséquent la plupart des API des gros services (twitter, facebook, google, etc.) ont une authentification basée sur OAuth.

Mais il est beaucoup moins simple que du Basic Auth ou bien qu'un simple token dans l'url: cf. cet exemple de requête http authentifiée sur l'api twitter:

<https://developer.twitter.com/en/docs/basics/authentication/guides/authorizing-a-request>  
(<https://developer.twitter.com/en/docs/basics/authentication/guides/authorizing-a-request>).

En général on utilise donc des modules python qui abstraient les requêtes http en charge de l'authentification via oauth.

Exemple avec le client d'api twitter en python:

Entrée [ ]:

```
# https://python-twitter.readthedocs.io/en/latest/getting\_started.html#your-keys

import twitter # pip install python-twitter

# fetch tokens from https://developer.twitter.com/en/apps/
api = twitter.Api(consumer_key="[consumer key]",
                  consumer_secret="[consumer secret]",
                  access_token_key="[access token]",
                  access_token_secret="[access token secret]")

api.GetUserTimeline(screen_name="username")
```

### Ex. github

- <https://github.com/settings/tokens> (<https://github.com/settings/tokens>)
- doc: <https://developer.github.com/v3/> (<https://developer.github.com/v3/>) /  
<https://developer.github.com/v3/repos/#list-your-repositories> (<https://developer.github.com/v3/repos/#list-your-repositories>)