

# L'authentification

## Authentification ou autorisation ?

- **Authentification** : *qui es-tu ?* L'échec donne un `401 Unauthorized` (mal nommé : il devrait s'appeler « Unauthenticated »).
- **Autorisation** : *as-tu le droit ?* L'échec donne un `403 Forbidden`.

Au bar : montrer sa carte de membre, c'est l'authentification. Le videur qui vérifie que la carte donne accès au carré VIP, c'est l'autorisation.

HTTP étant sans état, le client doit prouver son identité **à chaque requête**, en général dans le header `Authorization`.

## La clé d'API

Une longue chaîne aléatoire, propre à chaque client, envoyée dans un header.

```
curl http://api-ritif.portlatch.com/v1/me -H "X-API-Key: ak_7f3c9e1b2d4a"
```

- ☐ Très simple, idéale pour les échanges entre serveurs et les scripts.
- ☐ Elle ne périmé pas toute seule : si elle fuit, elle reste valable jusqu'à sa révocation.
- ☐ Elle identifie une application plus qu'un utilisateur.

Une clé d'API est un mot de passe : jamais dans le code, jamais dans git, jamais dans une URL ( `?api_key=...` finit dans les logs et l'historique du navigateur). Côté serveur, on la stocke **hachée**, comme un mot de passe.

## Basic Auth

L'identifiant et le mot de passe, encodés en Base64, dans le header `Authorization`.

```
curl -u neo:redpill https://api.example.com/v1/me
# envoie : Authorization: Basic bmVv0nJlZHBpbGw=
```

Base64 est un **encodage**, pas un chiffrement : `echo bmVv0nJlZHBpbGw= | base64 -d` affiche `neo:redpill`. Sans HTTPS, c'est un mot de passe en clair.

## Sessions ou tokens ?

|                   | Session (cookie)                                                                                 | Token (JWT...)                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Principe          | Le serveur garde la session en mémoire ou en base, le client garde un identifiant dans un cookie | Le client garde un jeton qui contient lui-même les informations, signé par le serveur |
| Envoi             | Automatique par le navigateur (Cookie)                                                           | Manuel ( <code>Authorization: Bearer ...</code> )                                     |
| État côté serveur | Oui                                                                                              | Non : le serveur vérifie juste la signature                                           |
| Révocation        | Facile : on supprime la session                                                                  | Difficile : le token reste valable jusqu'à son expiration                             |
| Adapté à...       | Un site web classique                                                                            | Une API, une application mobile, des microservices                                    |

## JWT

Un **JSON Web Token** est composé de trois parties en Base64url, séparées par des points :

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM5MjcwNjQ0MDAwfQ.7xQ...
```

\_\_\_\_\_ header \_\_\_\_\_ payload  
\_\_\_\_\_ signature

```
// header
{"alg": "HS256", "typ": "JWT"}
```

```
// payload (les « claims »)
{"sub": "12", "nickname": "neo", "role": "customer", "exp": 1794644000}
```

- Le **payload** est lisible par n'importe qui : **aucun secret dedans**.
- La **signature** garantit qu'il n'a pas été modifié : changer `"role": "customer"` en `"bartender"` invalide la signature.
- `exp` fixe la date d'expiration (en secondes depuis 1970). Un token expiré donne un `401`.

Pour lire un payload :

```
import base64, json
json.loads(base64.urlsafe_b64decode(token.split(".")[1] + "=="))
```

Le site [jwt.io](https://jwt.io) décode les JWT visuellement. Ne collez jamais un vrai token de production dans un site tiers.

## Avec API-ritif

```
# Échanger sa clé contre un JWT valable 15 minutes
curl -s -X POST http://api-ritif.portlatch.com/v1/auth/token -H "X-API-Key: $KEY"
# {"access_token": "eyJ...", "token_type": "Bearer", "expires_in": 900}

# L'utiliser
curl -i http://api-ritif.portlatch.com/v1/me -H "Authorization: Bearer $TOKEN"
```

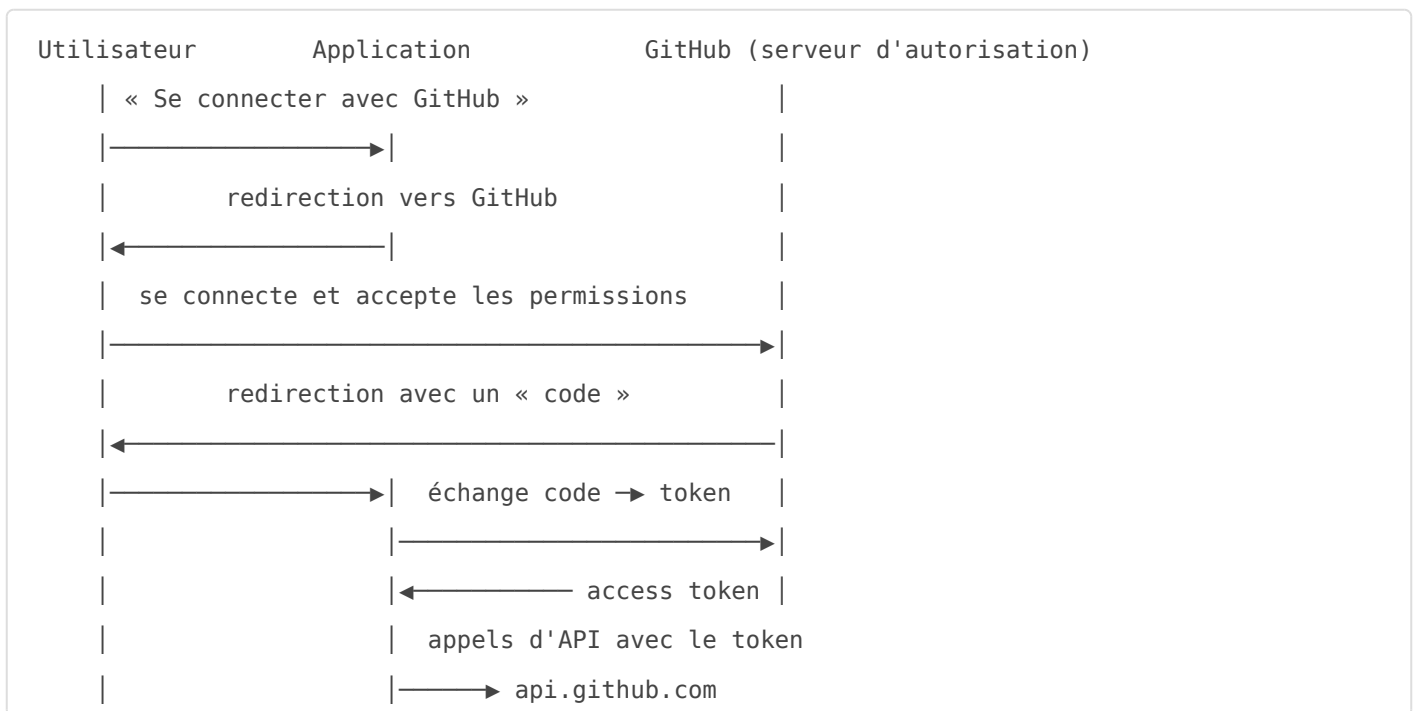
Revenez dans 15 minutes : même requête, `401`.

## Les pièges

- `alg: none` : certaines vieilles bibliothèques acceptaient un token « non signé ». Le serveur doit imposer l'algorithme attendu.
- **Durée de vie** : courte (quelques minutes), avec un *refresh token* pour en obtenir un nouveau sans se reconnecter.
- **Stockage dans le navigateur** : `localStorage` est lisible par n'importe quel script de la page (faille XSS). Un cookie `HttpOnly; Secure; SameSite` est plus sûr.

## OAuth2 : « Se connecter avec GitHub »

OAuth2 permet à une application d'agir **au nom d'un utilisateur** sur un autre service, sans jamais connaître son mot de passe.



- **Authorization Code + PKCE** : le flux ci-dessus, pour les applications web et mobiles.
- **Client Credentials** : pour un serveur qui appelle une API en son propre nom, sans utilisateur.
- **OpenID Connect** : une surcouche d'OAuth2 qui ajoute l'identité de l'utilisateur (un *ID token*, au format JWT).

On n'implémente jamais OAuth2 soi-même : on utilise un fournisseur (Keycloak, Auth0, Entra ID...) et une bibliothèque éprouvée.

Revision #4

Created 2 October 2026 20:00:55 by Thibaud FRICHET

Updated 13 October 2026 14:52:48 by Thibaud FRICHET