Quelques éléments à propos de l'authentification

- *capability* : token opaque
- Session
- token transparent : jwt
- OAuth

authentification par "capability"

- **principe** : un token opaque dans l'url ou dans un header permet l'accès à une ressource
- **intérêts** :
  - simple, pas de transport d'identifiant/mot de passe
- **risque** : partage accidentel de l'url/token
- **utilisation** :
  - Accès à des ressources sans identifiant dans 1 appli web ou 1 api, renouvellement de mots de passe
  - Clé d'API (ApiKey) pour identifier 1 client
  - *refresh* token pour authentification expirée
- **Conseil** : usage unique, re-génération après chaque utilisation

- Le token peut être transporté dans l'url ou dans un header

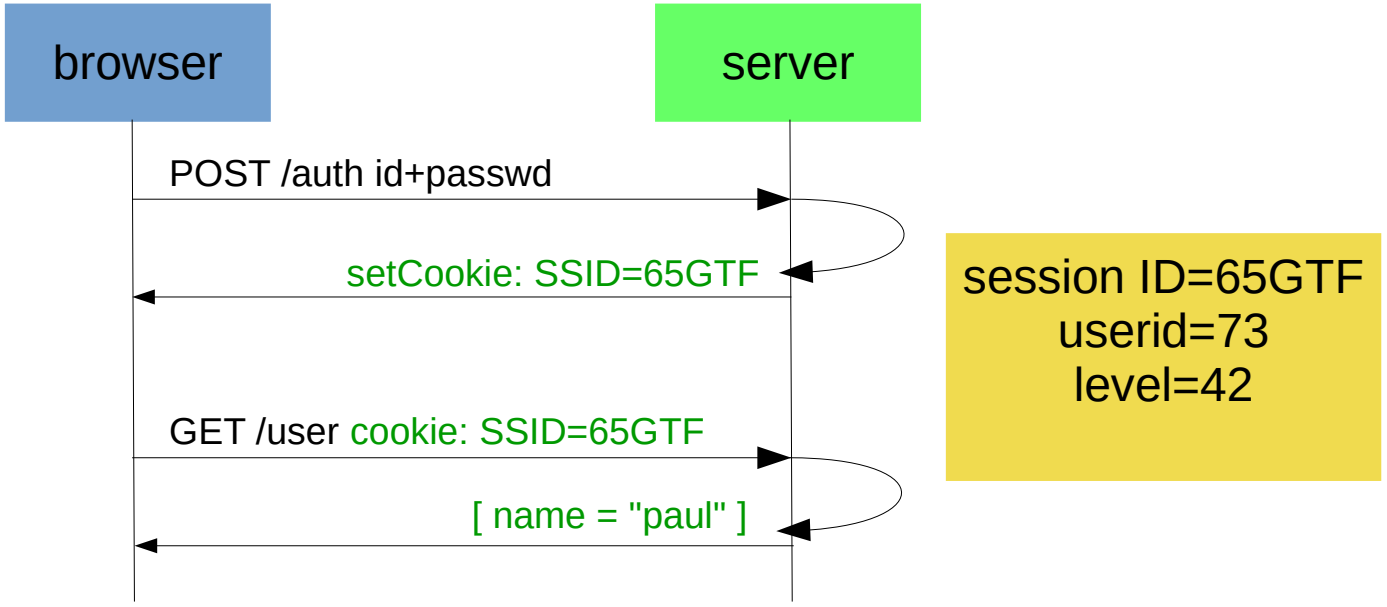
```
GET /commandes/345443876?token=FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

```
GET /commandes/345443876
X-app-token: FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

```
GET /commandes/345443876
Authorization: Bearer FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

authentification basée sur la session

- **principe** : un profil spécifiant un niveau d'accès est conservé en session sur le serveur après authentification
  - l'identifiant de session est échangé dans un cookie
- **intérêt** : le cookie de session est positionné automatiquement par le navigateur



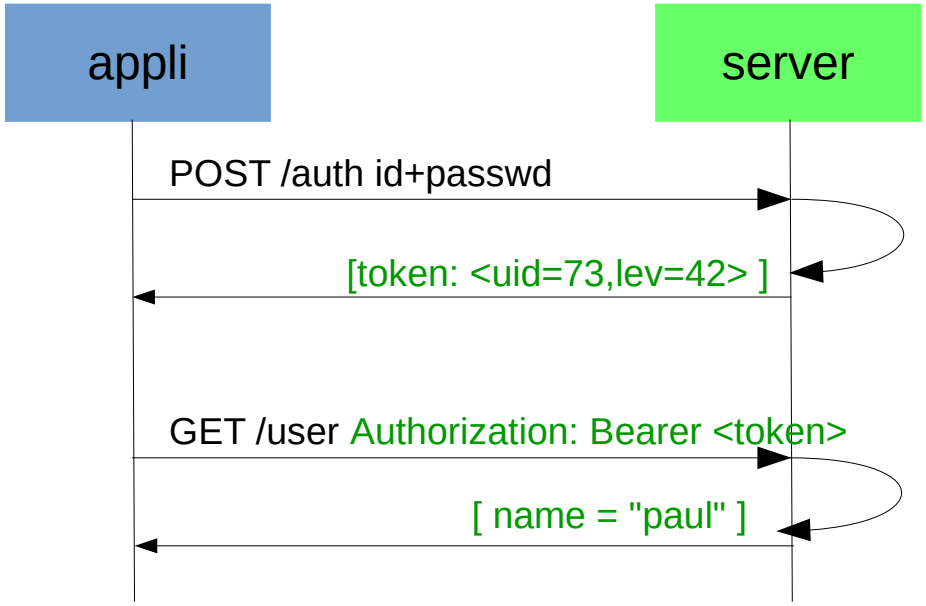
session

- **risques** :
  - vol de la session : interception du cookie de session
  - CSRF : l'exécution non choisie d'une requête malveillante se fait au sein de la session en raison du positionnement automatique du cookie
- **utilisation :**
  - authentification et contrôle des droits d'accès dans une application web classique pouvant fonctionner sans javascript

```
GET /commandes/345443876
Cookie:PHPSSID=FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

authentification basée sur un token type JWT

- **principe** : un token JWT est un token transparent (=lisible) transportant des informations (identité, droits ...)
- les informations d'autorisation sont placées dans le token et n'ont plus besoin d'être enregistrées en session



Token JWT

- le token est construit et signé par le serveur, il est le seul à pouvoir vérifier la signature grâce à un secret
- **intérêts :**
  - sans état : session inutile
  - bien adapté aux api et micro-services
  - pas de problème de CSRF
- **risques :** vol du token

```
GET /commandes/345443876
Authorization: Bearer u78hskkx9Nqfr345h7GGFDE21h2598Jendt
```

Access Token / refresh Token

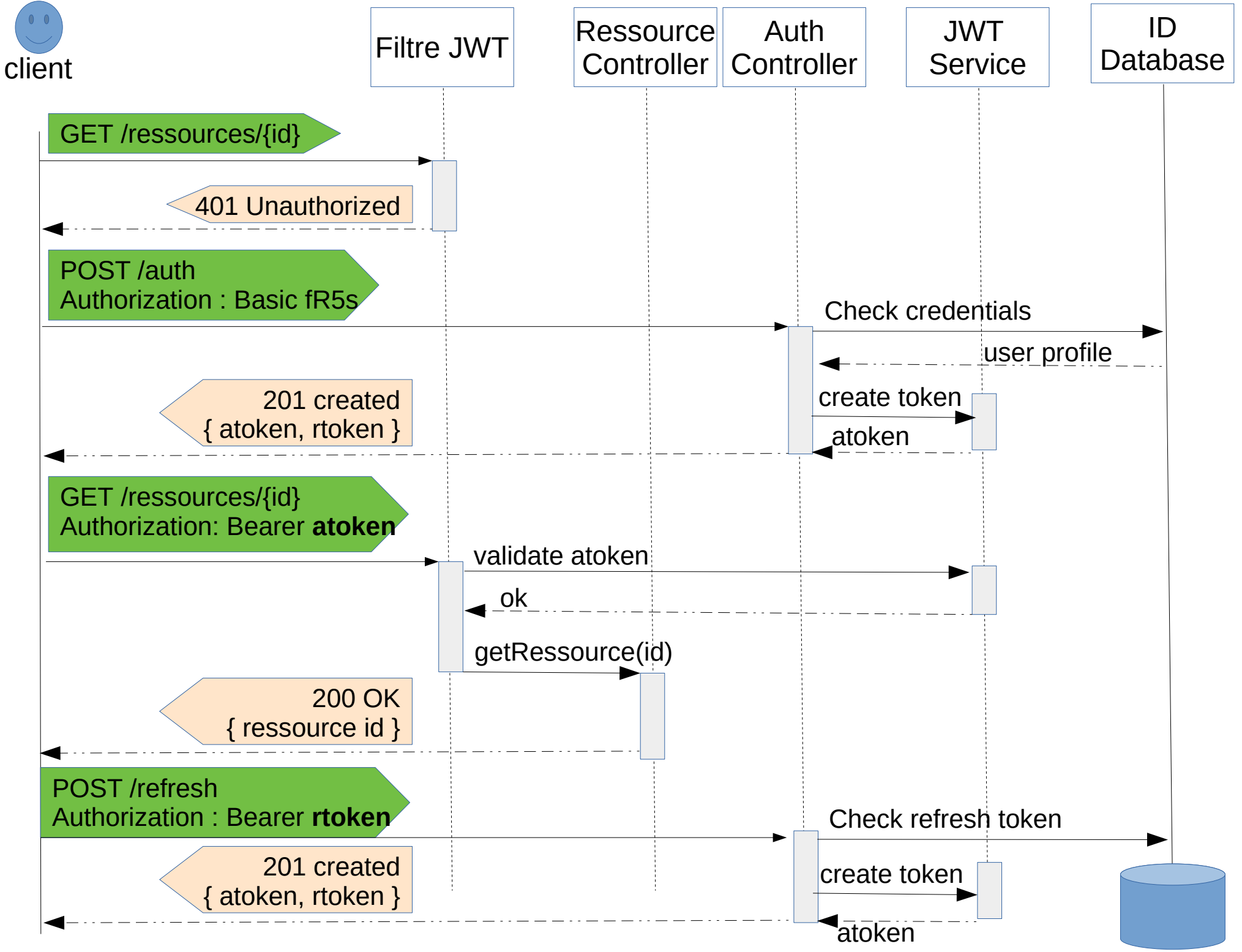
- Un token JWT contient des informations de contrôle d'accès → *access token*
- Il a une durée de vie limitée, parfois courte
- Pour éviter de demander au client de se ré-authentifier et de fournir et transporter à nouveau ses credentials, on peut fournir en plus un *refresh token* : token opaque à usage unique permettant de demander la re-génération d'un *access token*

OAuth

- protocole d'authentification distinguant :
  - le service nécessitant une autorisation
  - le serveur d'authentification délivrant les autorisations et détenant les credentials des utilisateurs
- Intérêt :
  - partager le service d'authentification entre plusieurs applications et API
  - Les applications et API n'ont pas besoin de connaître les credentials des utilisateurs

Mise en œuvre de l'authentification à base de jetons JWT

1) Le client émet 1 requête d'accès vers une ressource
2) Le serveur répond avec un code 401 et une url de redirection vers l'authentification
3) Le client fait 1 demande vers le serveur d'auth. en utilisant l'url et en transmettant les credentials
4) Le serveur vérifie les credentials, génère un *access token* et un *refresh token* et les retourne au client
5) Le client place l'*access token* dans toutes ses requêtes vers le serveur
6) Pour chaque requête, le serveur contrôle la validité de l'*access token* et réalise le contrôle d'accès
7) Lorsque l'*access token* n'est plus valide, le client utilise le *refresh token* pour obtenir un nouvel *access token*


client
Filtre JWT
Ressource Controller
Auth Controller
JWT Service
ID Database
GET /ressources/{id}
401 Unauthorized
POST /auth
Authorization : Basic fR5s
Check credentials
user profile
201 created
{ atoken, rtoken }
create token
atoken
GET /ressources/{id}
Authorization: Bearer atoken
validate atoken
ok
getRessource(id)
200 OK
{ ressource id }
POST /refresh
Authorization : Bearer rtoken
Check refresh token
201 created
{ atoken, rtoken }
create token
atoken

token JWT (rfc 7519)

- un token JWT comprend 3 parties :
  - une **entête** : spécifie le type de token et de hash
  - un **contenu** (payload) : des données placées par le serveur
  - une **signature** : permet au serveur de vérifier la validité du jeton, généré par un algo utilisant une clé secrète (HMAC ou RSA)
- le contenu : un objet json
  - des propriétés **prédéfinies** standardisées
  - des propriétés liées à **l'application**

jwt

- propriétés prédéfinies (optionnelles) :
  - "**iss**" : "issuer", identifie l'émetteur du token
  - "**sub**" : "subject", le sujet du token
  - "**aud**" : "audience", destinataires du token – le serveur recevant un token doit vérifier qu'il lui est bien destiné
  - "**iat**" : "issued at", date d'émission du token
  - "**exp**" : "expires", date d'expiration du token
  - "**nbf**" : "not before", date de validité du token
  - "**jti**" : "jwt id", identificateur unique du token

exemple

base64Url(header)

```
{
  "alg" : "HS512",
  "typ" : "JWT"
}
```

```
{
  "iss" : "https://auth.myapp.net",
  "aud" : "https://api.myapp.net",
  "iat" : 1513330929,
  "exp" : 1513334529,
  "data": {
     "user" : 123,
     "levl" : 6
}
```

base64Url(payload)

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJodHRwOi8vYXV0aC5teWFwcC5uZXQiLCJhdWQiOiJodHRwOi8vYXBpLm15YXBwLm5ldCIsImlhdCI6MTUxMzMzMDkyOSwiZXhwIjoxNTEzMzM0NTI5LCJkYXRhIjp7InVzZXIiOjEsImxldmwiOjZ9fQ.
kg65cK2uP2fAStQPhm1klpvqnTVPM9uZWyBFLBTuv-4
```

HMACSHA512(base64Url(header) . base64Url(payload), SECRET_KEY)

en php

- utiliser une librairie
  - firebase/php-jwt , lcobucci/jwt
- exemple avec firebase/php-jwt :

```
{
  "require" : {
    "firebase/php-jwt": "^5.0.0"
  }
}
```

```
use Firebase\JWT\JWT;

$token = JWT::encode( [ 'iss'=>'http://auth.myapp.net',
                        'aud'=>'http://api.myapp.net',
                        'iat'=>time(), 'exp'=>time()+3600,
                        'uid' =>  $user->id,
                        'lvl' =>  $user->access ],
                       $secret, 'HS512' );
```

- décodage du token :

```php
use Firebase\JWT\JWT;
use Firebase\JWT\Key;
use Firebase\JWT\ExpiredException;
use Firebase\JWT\SignatureInvalidException ;
use Firebase\JWT\BeforeValidException;

try {
  $h = $rq->getHeader('Authorization')[0] ;
  $tokenstring = sscanf($h, "Bearer %s")[0] ;
  $token = JWT::decode($tokenstring, new Key($secret,'HS512' ) ;

} catch (ExpiredException $e) {

} catch (SignatureInvalidException $e) {

} catch (BeforeValidException $e) {

} catch (\UnexpectedValueException $e) { }
```