

Augmentez la sécurité et la simplicité de votre Système d'Information avec OAuth 2.0, OpenID Connect et @axa-fr/react-oidc

12 min read

Apr 1, 2022

Cet article a pour objectif d'introduire ce qu'est OAuth 2.0 puis OpenID Connect et finalement la librairie [@axa-fr/react-oidc disponible sur GitHub](#) (dont je suis l'un des repository-maintainer). Cet article se base sur un cours sur les [API REST disponible sur GitHub](#) que je donne à l'IG2I Lens (école du Groupe Centrale Lille).

Vous pouvez dès maintenant jouer avec la [démonstration de @axa-fr/react-oidc](#), cela vous aidera probablement à mieux comprendre la suite de l'article.

Introduction à la demo @axa-fr/react-oidc

Introduction

Dans cet article, je commencerai par expliquer quelques notions pré-requises, puis le protocole OAuth2, ensuite je présenterai les avantages d'OIDC qui s'appuient sur OAuth2 et le complètent. Finalement, j'introduirai pourquoi [@axa-fr/react-oidc](#) est une réelle innovation en matière de sécurité et simplicité.

Préambule

L'authentification, l'identification et la gestion des autorisations sont des choses centrales et très techniques; souvent négligées, elles engendrent des surcoûts énormes pour chaque application et dans la gestion globale des systèmes d'information.

[OAuth 2.0](#) est un protocole qui permet de gérer les autorisations. Malheureusement, il n'y a pas de détail sur comment l'implémenter techniquement. Du coup, chaque implémentation est différente.

OpenID Connect (OIDC), c'est l'étape suivante. C'est une couche d'identification basée sur le protocole OAuth 2.0. Elle normalise la partie technique via une API REST et standardise la récupération des informations d'identification.

Le résultat est que le consommateur peut changer de fournisseur OIDC sans écrire le moindre bout de code. Il vous permet aussi de rendre OAuth2.0 compatible avec vos vieux systèmes d'authentification. Il peut faire office d'interface.

Bien comprendre les notions d'identification, autorisation et authentification

Il faut maîtriser la distinction, ce vocabulaire est important pour bien comprendre la suite.

- **Identification** : Qui êtes-vous ? Exemple : Login
Qui peut être authentifié ? C'est souvent soit une personne, soit une machine.
- **Authentification** Êtes-vous réellement cette personne/machine ? Exemple : Mot de passe.
- **Autorisation** : Est-ce que cette personne/machine a le droit d'accéder à cette ressource ?

Côté client HTTP, cela se transcrit comme ci-dessous

- **Authentication**

Code de réponse **HTTP 401**, si je ne suis pas authentifié et essaye d'accéder à une ressource privée

- **Autorisation**

Code de réponse **HTTP 403**, je suis authentifié mais je n'ai pas le droit d'accéder à la ressource

- **Identification**

Elle peut être réalisée de plein de manières différentes (formulaire, certificat, validation par une application, etc.).

Si vous faite une identification via un formulaire web :

Code de réponse **HTTP 400**, les informations que j'envoie par formulaire via HTTP POST ne sont pas valides.

OAuth 2 définit 4 rôles distincts

- **Resource Owner**

Un humain ou une machine

- **Resource Server**

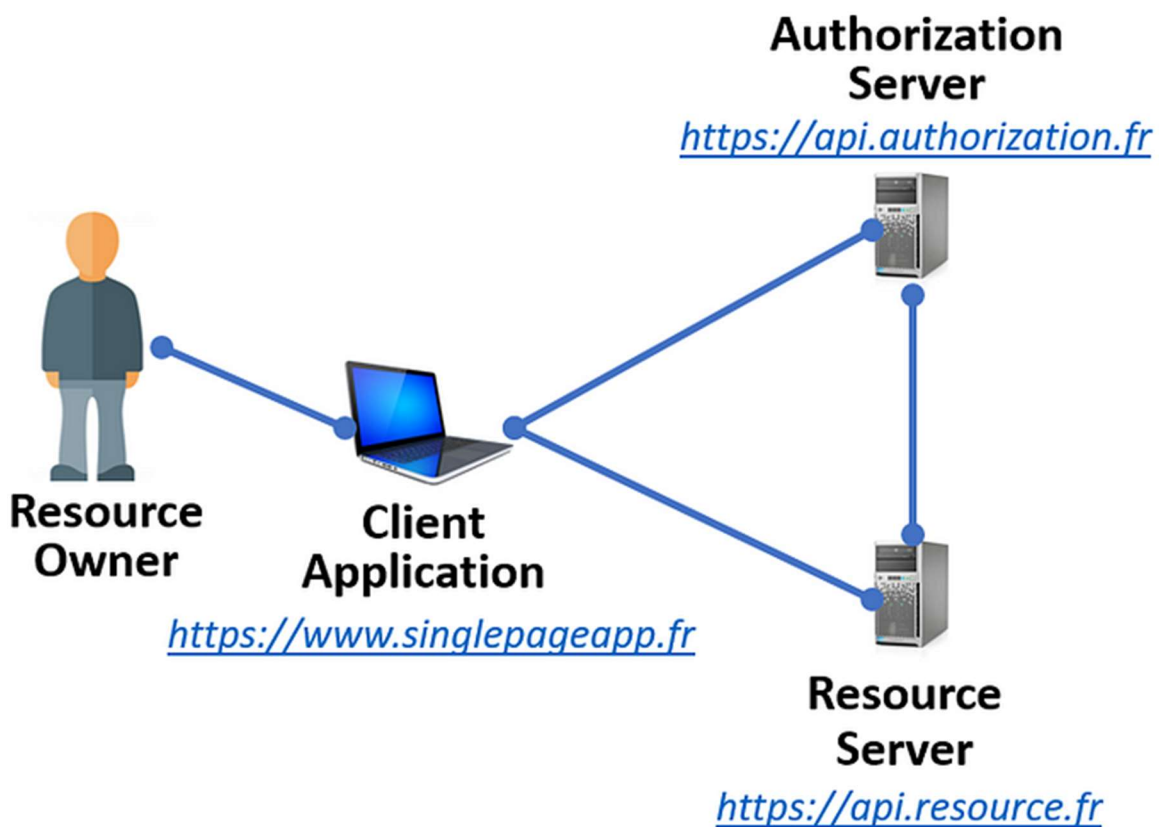
Héberge les données dont l'accès est protégé

- **Client Application**

Application demandant des données au serveur de ressources.

- **Authorization Server**

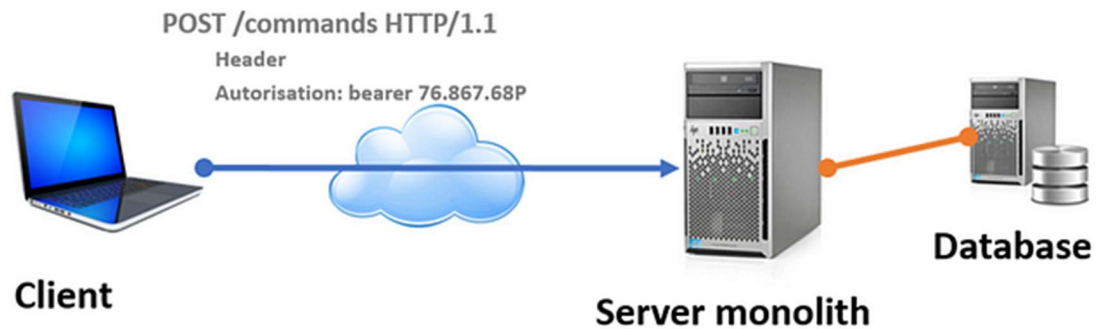
Délivre des jetons d'accès (tokens en anglais) au client.



OAuth 2 définit 4 rôles distincts

Pourquoi OAuth 2.0 ?

Imaginez que vous démarrez une startup et comme vous êtes pragmatique, vous réalisez un "Monolithe" bien découpé en domaines fonctionnels.



Exemple de domaines fonctionnels :

- Gestion des Commandes
- Identification, Authentification, Autorisation
- Envoi de SMS/mail
- Gestion des Stocks, Articles
- Gestion des Images

Super ! Votre application a du succès et est maintenant utilisée par des milliers de clients simultanément dans le monde. Les temps de réponse commencent à se dégrader car votre API est fortement sollicitée.

Vous êtes passé de 2 développeurs à 45, les développements et livraisons deviennent compliquées.

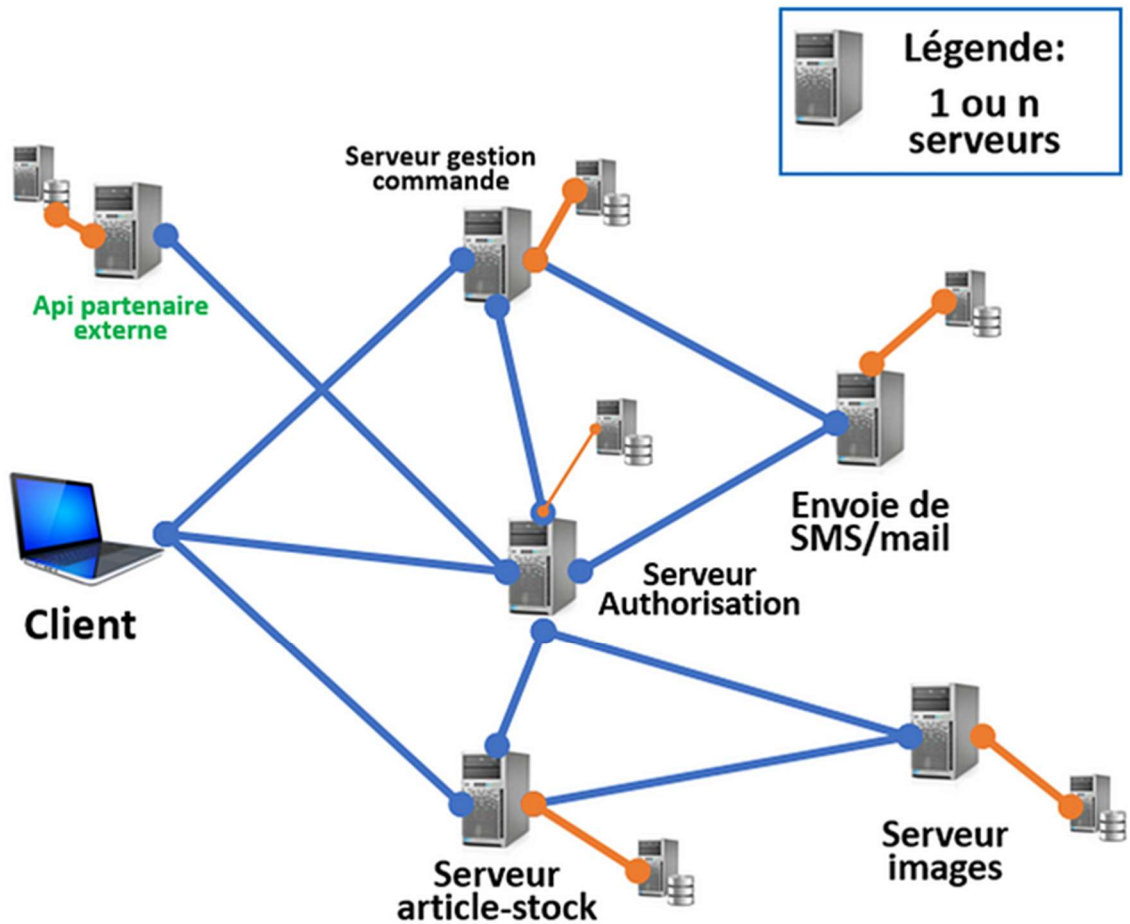
Vous avez un nouveau besoin qui est d'ouvrir votre API à des partenaires tiers !

La solution

Pour pouvoir garder des temps de réponses acceptables et que l'application reste maintenable, on réarchitecture l'application en microservices.

Afin de pouvoir gérer l'authentification sur tous les « services » fonctionnels, les premières briques à extraire sont celles qui gèrent l'authentification, l'indentification et l'autorisation.

Vous devez être capable de tout tracer qui/quoi accède à quoi à un instant T, de contrôler et de limiter les accès.



Microservice architecture

Ce protocole permet à des applications tierces d'obtenir un accès limité à un service exposé via HTTP par le biais d'une autorisation préalable du détenteur des ressources.



Château fort

C'est le principe **inverse** à celui des châteaux forts. Il est dur de rentrer dans un château fort mais, une fois que l'on y est on fait ce que l'on veut. Et ça, ce n'est pas bien ! C'est d'ailleurs pour cela que l'on ne fait plus de château fort.

Avec OAuth 2.0 on part donc du principe inverse, c'est à dire que l'on sait que « **oui** », on va se faire pirater un jour. On se fera pirater MAIS sur une ressource très limitée, et sur une période très courte.

De plus, en cas d'intrusion, les entités et « tokens » mis en jeu sont révocables par le serveur d'autorisation.



Pour cela OAuth 2.0 s'appuie sur un jeu de deux tokens (clefs ou jetons en français)

Access Token, le jeton d'accès

Permet au serveur de ressources d'autoriser la mise à disposition des données d'un utilisateur. Ce jeton est envoyé par le client (l'application) dans la requête vers le serveur de ressources. Il a une durée de vie limitée qui est définie par le serveur d'autorisation : *par exemple 20 minutes*.

Refresh Token, le jeton de renouvellement

Ce jeton est délivré au même moment que le jeton d'accès. Il sert à renouveler le jeton d'accès quand celui-ci est expiré. Il a une durée de vie limitée, mais plus longue que le jeton d'accès : *par exemple une semaine*.

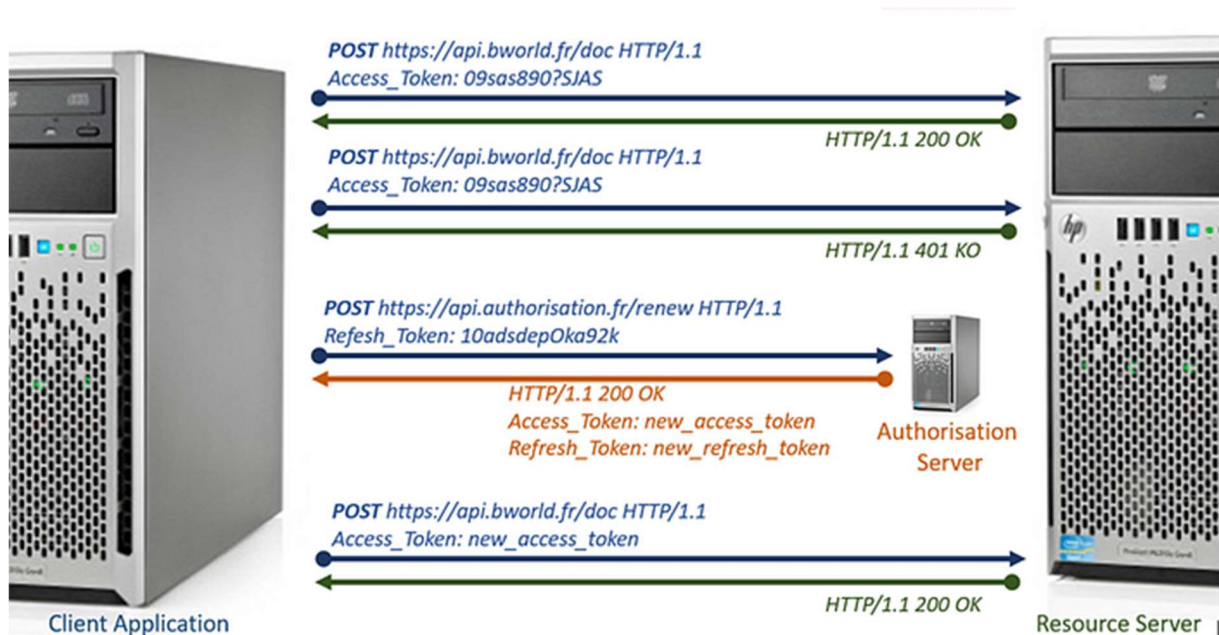


Schéma renouvellement des tokens

Concrètement l'« access_token » est celui que vous envoyez tout le temps à votre API. C'est la plupart du temps un [JSON Web Token](#) (JWT). C'est un format standard d'échange d'information entre clients et serveurs qui permet d'être sûr que l'information n'a pas été modifiée en cours de route.

Le « refresh token » n'est pas forcément un JWT. Il transite moins sur le réseau mais est plus sensible d'un point de vue sécurité car il permet le renouvellement.

Scopes et Audiences

Dans le contenu du jeton d'accès et si c'est un JSON Web Token (JWT), vous entendrez parler de « Scope » et d'« Audience ». Ce sont 2 informations très importantes et assez complexes à comprendre.


L'« Audience » et « Scope » ne sont pas des rôles. La notion de rôle est encore autre chose très spécifique à chaque API.

Dans votre API, vous vérifiez :


- 1) que le jeton porte bien l'« Audience » qui vous correspond,
- 2) pour chaque pan fonctionnel ou « ressources » en terme REST (car votre API peut en proposer plusieurs) vous vérifiez que le « Scope » associé au pan fonctionnel de la requête HTTP en cours est bien présent ; [Une API Restful est normalement architecturée en domaines fonctionnel bien découplés les uns des autres.](#)
- 3) Finalement, si vous avez besoin d'une gestion plus fine des droits d'accès vous vérifiez si l'utilisateur a le droit de réaliser cette action à l'aide d'un rôle (exemple: utilisateur, validateur, relecteur, administrateur). La notion de rôle est beaucoup plus fine et ne fait pas partie du protocole OAuth 2.0. Elle est vraiment liée a votre application uniquement. C'est à vous d'injecter les informations nécessaires par exemple dans les champs clef/valeur du jeton JWT depuis votre serveur OIDC.

Par exemple, ajout d'un "claim" pour donner les rôles de l'utilisateur connecté :

- **member_of:** user, administrator



DebuggerLibrariesIntroductionAsk

Crafted by 

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJHdWlsbG
F1bWUgQ2hlcjZldCIiImNlaSI6ImU4Y2I0YmZmI
iwiaXNzIjoiaHR0cHM6Ly9pZGVudGl0eS5pbyIs
ImNsaWVudF9pZCI6InNhbmhXb3ZSI6ImF1ZCI6WyJ
pZGVudGl0eSIsImV0ZyJdLCJhY3IiOiIxIiwicm
xtIjojUmVjZXR0ZSI6ImNjb3B1Ijoib3B1bm1kI
HByb2ZpbGUgZW1haWw1LCJheGFfdHlwZSI6IjEi
LCJleHAiOiJlNDg3MjMyODAsIm1lbWJlc19vZiI
6IkrBVEFfU0NJRlU5USVNULEFOTk9UQVRVFIiLC
JpYXQiOiJlNDg3MjM0ZDAsIm1lbWJlc19vZiI6I
MCJ9.~5eAY74nheCstH005KH5-
FYqoWC_8iabV5PdZq8P8EU
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

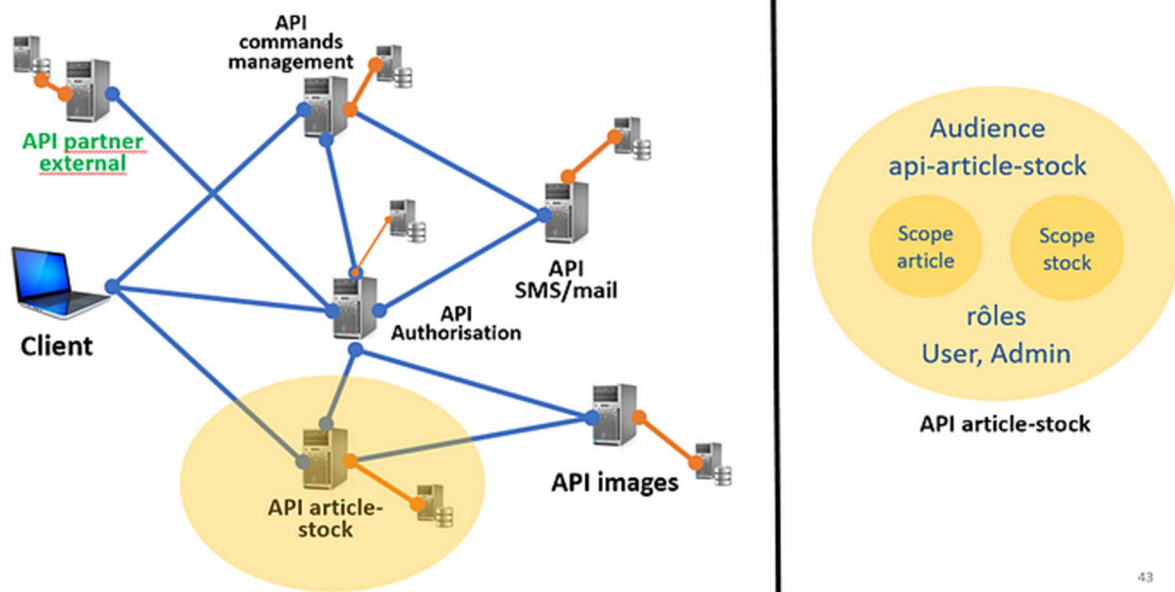
PAYLOAD: DATA

```
{
  "exp": "e8cb4bff",
  "iss": "https://identity.io",
  "client_id": "sample",
  "aud": [
    "identity",
    "etg"
  ],
  "acr": "1",
  "rlm": "Recette",
  "scope": "openid profile email",
  "axa_type": "1",
  "exp": 1648723280,
  "member_of": "DATA_SCIENTIST, ANNOTATEUR",
  "iat": 1648719680,
  "jti": "703fc11b-6013-4abf-8dfa-b4f614730760"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Exemple d'ajout d'un "claim" pour gérer les rôles propre a chaque API



43

Focus sur l'API article-stock de votre Système d'Information

Être connu du serveur d'autorisation

Un client ne peut utiliser le protocole OAuth 2.0 sans être connu du serveur d'autorisation. Il lui faut donc s'enregistrer auprès du serveur d'autorisation. Pour cela, il doit fournir un ensemble de données :

- Nom de l'application
- URL du site en HTTPS
- URL de retour
- Etc.

En échange, le serveur fournira un identifiant (« client id ») et, si nécessaire un code secret (« client secret ») sous forme de chaînes de caractères, qui permettront au client de s'identifier.

Types d'autorisation

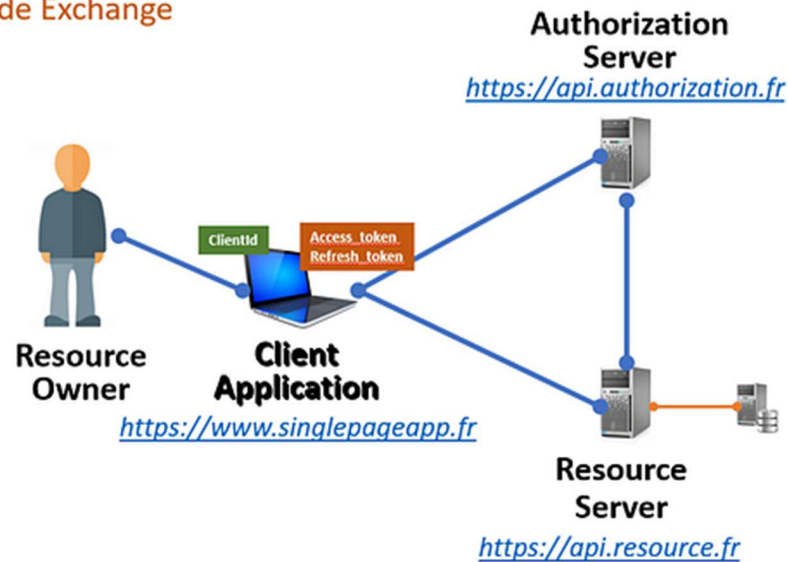
- **Authorization Code Grant, l'autorisation via un code [Déprécié]**
Le client est un serveur web. Permet d'obtenir un jeton d'accès de longue durée qui pourra être renouvelé via un jeton de renouvellement.
- **Implicit Grant, l'autorisation implicite [Déprécié]**
L'application se trouve côté client (Javascript, application mobile).
- **Authorization Code Flow with Proof Key for Code Exchange (PKCE)**
L'application se trouve côté client (Javascript, application mobile) ou serveur.
- **Client Credentials Grant, l'autorisation serveur à serveur**
Utilisé par les clients pour obtenir un jeton d'accès en dehors du contexte d'un utilisateur.
- **Resource Owner Password Credentials Grant, l'autorisation via mot de passe [Déprécié]**

Focus sur « Authorization Code Grant with PKCE »

Ce flux peut être utilisé côté client (un navigateur ou une application mobile, par exemple) et côté serveur.

Côté client

Proof Key for Code Exchange Client Side



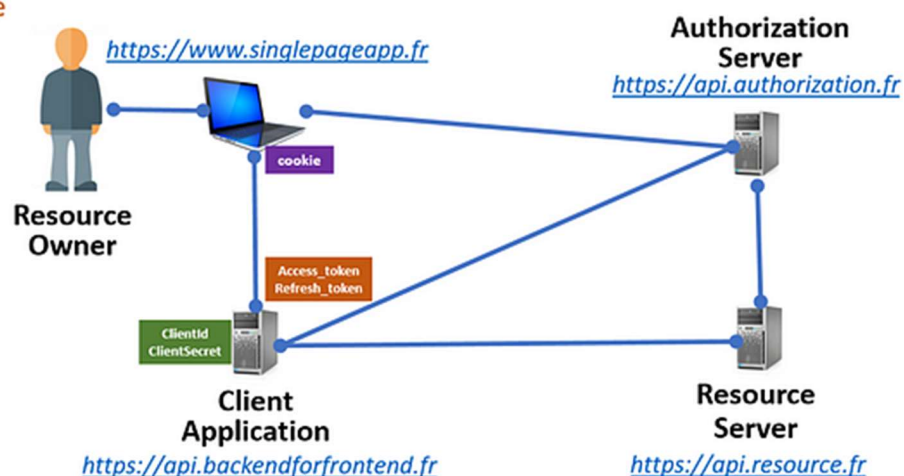
Proof Key for Code Exchange : Client Side

Ce mode est le plus facile à mettre en place (et celui qui coûte le moins cher), cependant les jetons d'accès se [retrouvent côté client](#). L'application client est donc très sensible aux attaques de type « [Cross Site Scripting](#) » (XSS).

De plus, si un pirate accède aux « refresh tokens », il pourra alors récupérer des jetons d'accès. Heureusement que vous n'offrez qu'un « scope » et une « audience » limités par application !

Côté serveur

Proof Key for Code Exchange Server Side



Proof Key for Code Exchange : Server Side

Ce mode est plus sécurisé mais plus complexe à mettre en place (coute plus cher). Aujourd'hui on entend parler de « Backend For Frontend » (BFF). Les jetons d'accès et les roulements se font côté serveur (il s'agit d'une délégation d'autorisation). Les jetons ne sont normalement pas accessibles du client (via javascript, notamment).

La session entre le client et le serveur doit se faire via un échange de cookie, qui doivent être fortement sécurisés car ils sont sensible eux aussi aux attaques de type XSS.

« Level up » avec OpenID Connect

Pour faire simple, OpenID Connect (OIDC) ajoute en standard tout ce qui manque à OAuth2.0. Avec OAuth 2.0, chaque implémentation est différente et nécessite des librairies spécifiques à chaque fournisseur. Avec OIDC, l'implémentation devient plus simple.

Open ID Connect = (Identity, Authentication) + OAuth 2.0

OpenID Connect standardise

- La récupération **informations utilisateurs**
- Une API (User Info endpoint)
- Utilisation du scope **ID Token**
- L'**authentification**
- Gestion de la **session SSO** (ex: le Single Logout)
- **Un Système de découverte du serveur OpenID, via l'utilisation d'une “/.well-known” URL;** une URL bien connue qui liste toute les autres URLs.

Tous cela vous permet de changer de fournisseur sans changer votre code

L'API REST est simple et standardisée :

- **authorization** : pour authentifier un utilisateur
- **token** : pour demander un jeton (access / refresh / ID)
- **user info** : pour récupérer des informations sur l'utilisateur (son identité, ses droits)
- **revocation** : pour supprimer un jeton (access / refresh)
- **introspection**: pour valider un jeton (access / refresh)

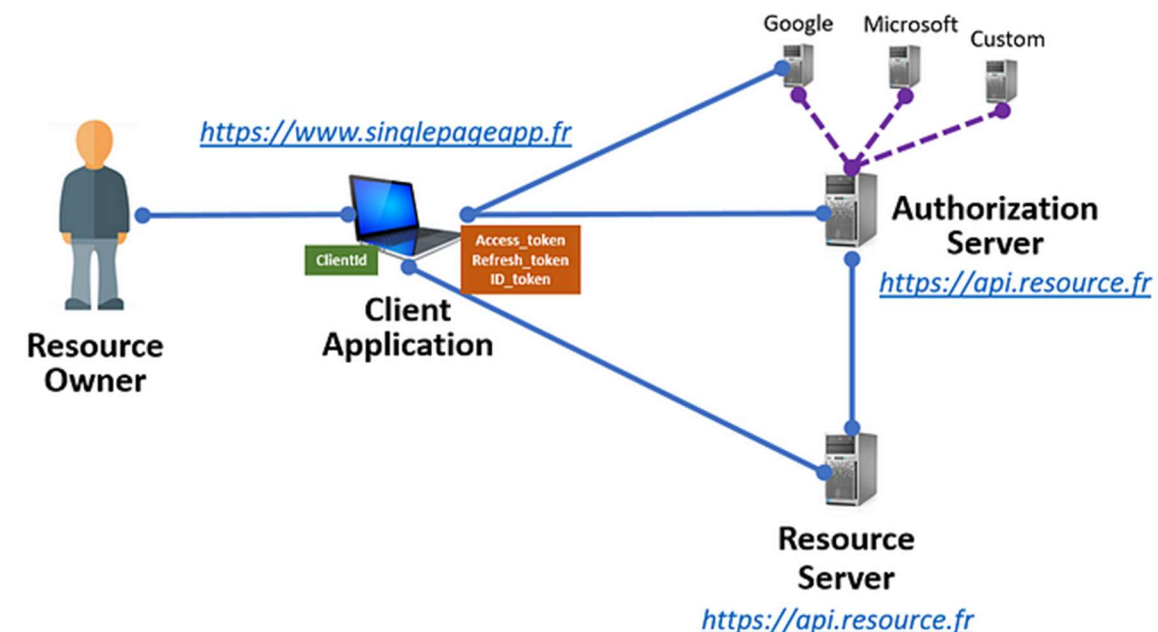
Ajout d'un troisième jeton : IDToken

L'« ID Token », c'est un JWT qui contient l'identité d'un utilisateur.

- Il contient les paramètres de l'**authentification** : date d'expiration, date de création, date d'authentification, des moyens de contrôle permettant de valider l'« ID Token » et l'« Access Token ».
- **Les accréditations (rôles, habilitations) de l'utilisateur.**
- **Des attributs (claims) de l'utilisateur associés à des scope.**
attributs standards :
 - scope « **profile** » : nom, prénom, surnom, date de naissance,
 - scope « **email** » : email, email vérifié
 - scope « **address** » : adresse
 - scope « **phone** » : numéro de téléphone, numéro de téléphone vérifié
 - etc.**attributs privés :**
 - attributs proposés par le fournisseur d'identité. Il est nécessaire de les spécifier afin d'éviter toute collision avec des claims existants.

OpenID Connect vous permet de faire de la fédération

Votre serveur OpenID Connect vous permet de mettre en place une sorte d'interface avec tous vos systèmes d'identifications, d'authentifications et d'autorisations. Cela permet de rendre vos anciens systèmes compatibles OAuth 2.0 qui vous permet ainsi de faire des "Single Page Application" (SPA).



Confidential

Un exemple de Serveur OIDC qui fait l'interface entre Google, Microsoft et un système Custom, que vous vous connectiez à Google ou Microsoft ou Custom, le code de votre application cliente sera le même.

C'est le serveur OIDC qui gère la complexité et les mises à jour techniques avec les autres Systèmes. Les applications clientes elles, n'ont qu'à utiliser les librairies standards OIDC compatibles. Un gain énorme en simplicité et en coût pour votre système d'information !

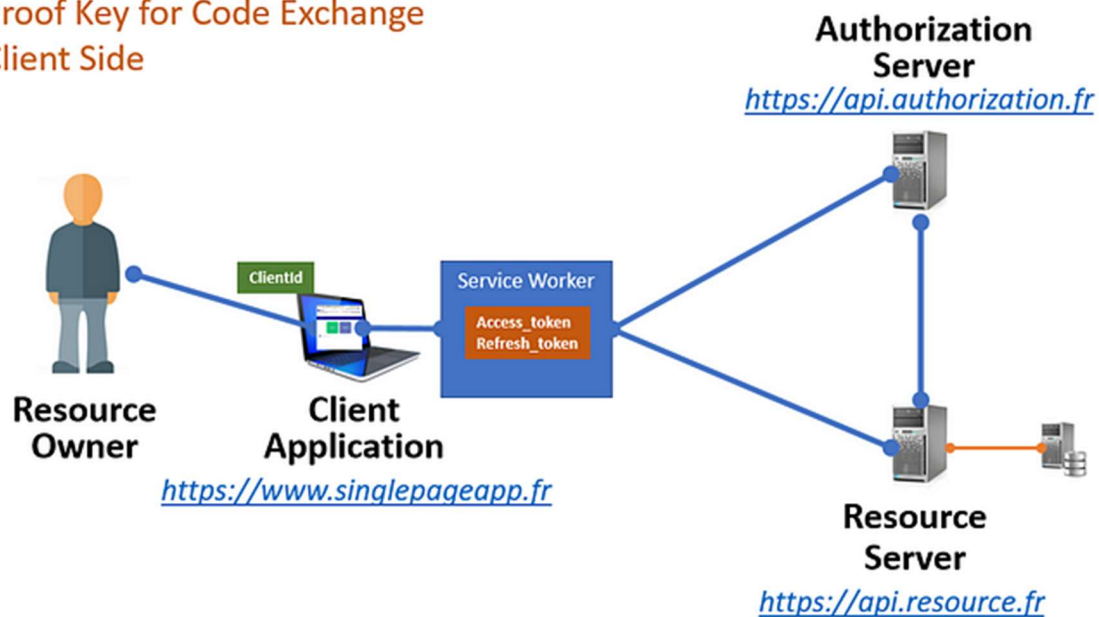
Finalement, parlons d'AXA React OIDC

[React OIDC V4](#) est une vraie innovation. Elle vous permet d'utiliser le flux « Authorization Code Grant with PKCE » côté client. [Ce flux est sensible aux attaques de type XSS](#).

Le petit truc en plus qu'offre [@axa-fr/react-oidc](#), c'est un mode qui utilise un « [ServiceWorker](#) » qui fait office de proxy entre le client et le serveur qui permet de **masquer complètement** au code client Javascript l'accès à l'« access_token » et au « refresh_token ». Le « [ServiceWorker](#) » est capable d'injecter automatiquement vos « access_token » dans vos requêtes vers votre API ainsi que le « refresh_token » dans les requêtes vers le serveur OIDC.

Flow Authorization Code Grant **with PKCE**

Proof Key for Code Exchange
Client Side



react oidc masque les jetons du client javascript

Votre application devient par conséquent beaucoup moins sensible aux attaques de type XSS et augmente donc significativement son niveau de sécurité.

C'est une vraie alternative qui coute beaucoup moins chère à réaliser que de mettre en place un BFF.

La librairie vous permet aussi de faire de l'authentification multiple. Cela peut être pratique pour notamment sécuriser certaines opérations sensibles qui nécessitent un scope ou audience particulière. Vous pouvez même ne pas demander de jeton de renouvellement.

Par exemple lorsque vous vous connectez au site de votre banque vous pouvez voir le solde de votre compte en banque. Lorsque vous souhaitez réaliser un transfert d'argent, le site va vous demander de nouveau de vous authentifier (une sur-authentification). Dans ce cas vous recevez un nouveau scope et/ou audience plus sensible valide sur une durée de temps très courte et sans jeton de renouvellement.

Pour plus d'information, rendez-vous sur la page [Github de React OIDC](#).

Pour résumer, vous pouvez donner finement et en permanence des accès limités sur une période limitée.

La suite en vanillajs

La librairie derrière [@axa-fr/react-oidc](https://github.com/axa-fr/react-oidc) est native javascript, elle n'est donc pas liée à React, il y a juste un "binding" avec React. Il y a de fortes chances de voir très rapidement apparaître de nouveaux "binding". Par exemple : webcomponent, angular, vuejs, blazor, etc.

Conclusion

Vous avez maintenant un aperçu de ce qu'est OpenID Connect et les clés pour sécuriser simplement votre Système d'Information.

La sécurité n'est pas négociable elle doit être présente par défaut dans vos applications ET justement, c'est ce que vous permettent les outils présentés dans cet article ! Alors qu'attendez-vous ?

Un grand merci à AXA France qui m'accompagne et me pousse à donner des cours ainsi qu'à faire de l'[Open Source](#).

Un **très** grand merci à tous ceux qui ont aidé à la relecture de cet article :

- Gilles Cruchon
- Lilian Delouvy
- François Descamps
- Antoine Blancke
- Louise Ryckewaert
- Preclin Amaury