



▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ > ▶ o < + Ⓛ >

HTTP, API & Architectures web

- Parcours OCTO Skool

2022-10 F/R - Version 1.5 - 19 Octobre 2022



TODO

- **TP HTTP ?**
 - > construire une requête ?
 - > récupérer les informations intéressantes ?
- **Parler de WebSocket / SSE ?**
- **Ajouter TP design : Créer un design API sur la base d'un cas métier ?**
- **Slide Chronologie (34) et Twitter (35) : à mettre à jour / à vérifier**
- **(Slide sécu web ? (CORS, XRSF, Https, ...))**
- **Détailler PKCE ?**
- **Faire une liste des changements apportés par OIDC ?**



Sommaire

01 Accueil

02 HTTP

03 Les applications web

04 Design

05 Architecture en pratique

06 Sécurité et API Management

07 Questions ?



01

Accueil



Tour de table

Présentons nous en quelques mots

- ⦿ Qui suis-je ?
- ⦿ Mes attentes pour cette formation ?

Vos formateurs:

Jean-Sébastien Dupuis

Consultant Séniior @OCTO
j.dupuis@octo.com

Jérémy Bouhi

Consultant Confirmé @OCTO
jeremy.bouhi@octo.com





Vos attentes

Je serais content à l'issue de cette formation si :

- Léo : dev 10 ans
 - > Dev front principalement
 - > En apprendre plus sur le bas niveau
 - > Echanger avec pair
- Avel dev :
 - > Dev front principalement js
 - > Combler les "trous dans ma raquette" de connaissance
- Thomas : TL
 - > 7 ans de dev
 - > 2 ans de TL
 - > piqûre de rappel sur des choses que j'ai pratiqué en tant que dev
- Auguste : dev 1an 1/2
 - > Dev principalement Back
 - > apprendre des choses



Les outils

Installez sur vos machines au moins un des outils suivants :

Sur un terminal : à installer via brew/apt-get/...

- cURL ([cheat sheet](#))
- HTTPie ([cheat sheet](#))

Dans une application dédiée :

- Postman ([installation](#))
- Insomnia ([installation](#))

Sur votre navigateur :

- <https://hoppscotch.io/fr>



02

HTTP



HTTP dans le modèle OSI

v · m	Couches du modèle OSI	[masquer]
7. Application	BGP · DHCP · DNS · FTP · FTPS · FXP · Gopher · H.323 · HTTP · HTTPS · IMAP · IPP · IRC · LDAP · LMTP · MODBUS · NFS · NNTP · POP · RDP · RTSP · SILC · SIMPLE · SIP · SMB-CIFS · SMTP · SNMP · SOAP · SSH · TCAP · Telnet · TFTP · VoIP · Web · WebDAV · XMPP	
6. Présentation	AFP · ASCII · ASN.1 · HTML · MIME · NCP · TDI · TLS · TLV (en) · Unicode · UUCP · Vidéotex · XDR · XML	
5. Session	AppleTalk · DTLS · NetBIOS · RPC · RSerPool · SOCKS	
4. Transport	DCCP · RSVP · RTP · SCTP · SPX · TCP · UDP	
3. Réseau	ARP · Babel · BOOTP · CLNP · ICMP · IGMP · IPv4 · IPv6 · IPX · IS-IS · NetBEUI · OSPF · RARP · RIP · X.25	
2. Liaison	Anneau à jeton (token ring) · Anneau à jeton adressé (Token Bus) · ARINC 429 · AFDX · ATM · Bitnet · CAN · Ethernet · FDDI · Frame Relay · HDLC · I ² C · IEEE 802.3ad (LACP) · IEEE 802.1aq (SPB) · LLC · LocalTalk · MIL-STD-1553 · PPP · STP · Wi-Fi · X.21	
1. Physique	4B5B · ADSL · BHDn · Bluetooth · Câble coaxial · Codage bipolaire · CSMA/CA · CSMA/CD · DSSS · E-carrier · EIA-232 · EIA-422 · EIA-449 · EIA-485 · FHSS · HomeRF · IEEE 1394 (FireWire) · IrDA · ISDN · Manchester · Manchester différentiel · Miller · MLT-3 · NRZ · NRZI · NRZM · Paire torsadée · PDH · SDH · SDSL · SONET · T-carrier · USB · VDSL · V.21-V.23 · V.42-V.90 · Wireless USB · 10BASE-T · 10BASE2 · 10BASE5 · 100BASE-TX · 1000BASE-T	



HTTP

Proposé en 1991 par Tim Berners-Lee pour le CERN.

C'est le protocole qui sert aujourd'hui de colonne vertébrale pour le web, et c'est celui que l'on rencontrera le plus souvent en tant que développeurs d'applications web.

Les versions la plus utilisée reste HTTP/1.1 et la plupart des clients et serveurs sont aujourd'hui compatibles HTTP/2. La version 3 est en cours de déploiement.

C'était jusqu'à récemment un protocole "Human-Readable" où les données transitaient en texte clair. Aujourd'hui les données sont transmises en binaire par soucis de sobriété et d'efficacité.



HTTP

Les grandes évolutions

1991 HTTP/0.9 Les bases

1996 HTTP/1.0 Introduit le concept de Headers

1997 HTTP/1.1 Introduit l'authentification, le cache, ajoute le verbe OPTIONS

2015 HTTP/2 Introduit le multiplexing, le push, et envoie les données en binaire et non plus en texte

2021 HTTP/3 Changements techniques, passage à l'UDP pour le transport



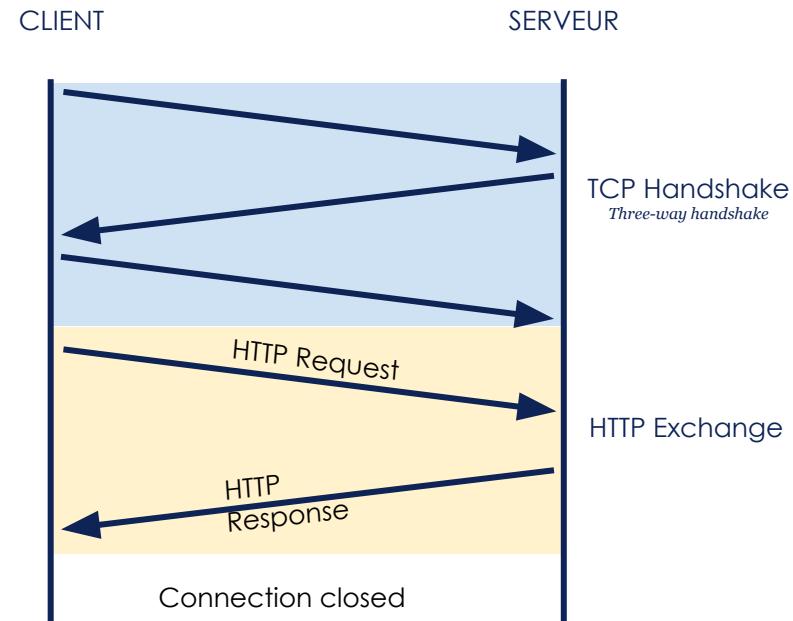
HTTP

Comment ça marche ?

HTTP repose sur le protocole TCP.
Une connexion TCP a un coût en temps non négligeable car elle nécessite beaucoup d'allers-retours.

Imaginez : le temps de transit entre le client et le serveur est de 10ms, le temps pour établir une connexion est de 40ms, et pour une connexion SSL/TLS de 60ms.

Et cela, avant même d'avoir commencé à envoyer notre requête HTTP !





HTTP

Structure d'une requête





HTTP

Structure d'une réponse

Status Code
Protocole
HTTP/1.1 200 OK

Date: Fri, 09 Oct 2020 16:13:16 GMT
Content-Type: application/json
Headers
Cache-Control: max-age=0, private, must-revalidate

```
{  
    "id": 2142664977,  
    "last_name": "Dupuis",  
    "first_name": "Jean-Sébastien",  
    "nickname": "DUJE",  
}
```

Body



HTTP

Les verbes

HTTP permet les opérations de type CRUD (Create, Read, Update, Delete) à travers plusieurs verbes d'action simples.

GET

Obtenir des ressources du serveur

HEAD

Idem que GET mais ne recevoir que les Headers

OPTIONS

Savoir quels sont les verbes disponibles sur un path donné

POST

Envoyer des données au serveur

PUT / PATCH

Modifier une donnée existante

DELETE

Supprimer une donnée du serveur



HTTP

Les headers

Les Headers permettent de délivrer tout un ensemble de métadonnées supplémentaires à l'envoi ou à la réception pour mieux gérer la requête.

Il existe des headers pour gérer l'authentification, le cache, définir les formats de données, la durée de vie de la connexion, la sécurité...

Liste exhaustive : <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>



HTTP

Les headers

Pourriez-vous citer quelques headers que vous avez déjà vu ?



HTTP

Des headers que l'on verra souvent

Accept, Content-Type, ... : Permettent de négocier / spécifier le type de contenu transmis

Authorization : Permet de gérer l'accès à une ressource (cas usuel : clef d'API, Token OAuth, ...)

Cache-Control, Etag, Expires, ... : Permet de définir le temps qu'un navigateur / proxy va garder une réponse en cache, ou d'identifier quand une ressource n'est plus fraîche

Content-Security-Policy : Définit quels domaines externes sont autorisés pour chaque type de contenu

Content-Length : Donne la taille du body, en octets

Ressource : <https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>



HTTP

Body

Que peut-on mettre dans le body d'une requête HTTP ?



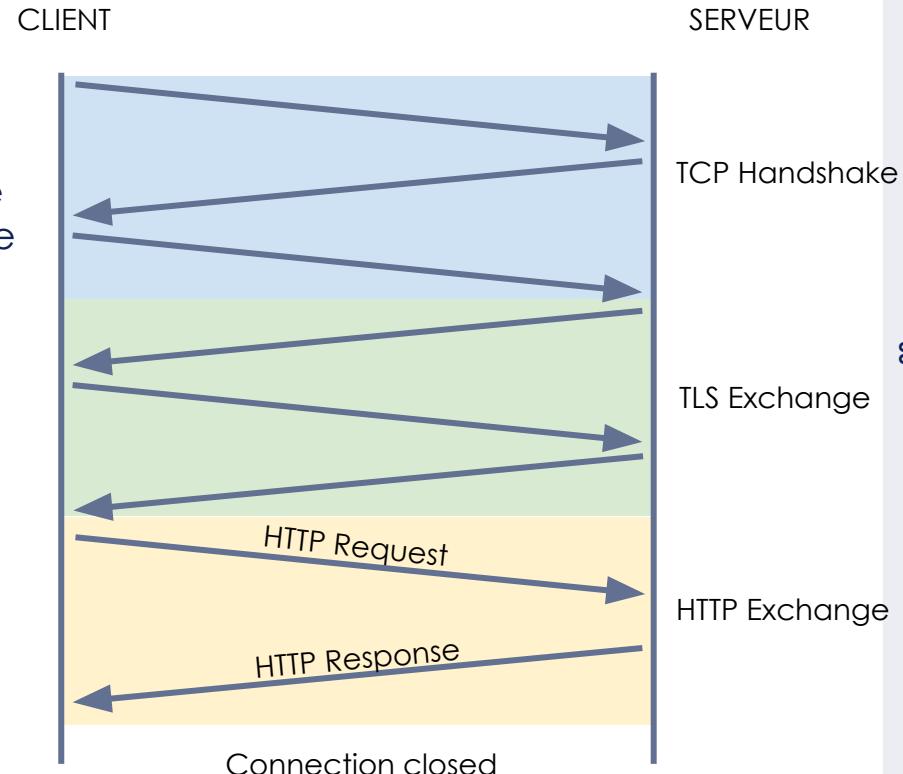
HTTP

HTTPS ?

HTTP Over TLS est une extension de HTTP qui permet l'échange de données chiffrées entre le client et le serveur. Il permet entre autres de se prémunir de l'écoute réseau, des attaques MiTM (Man in the middle), etc.

Cependant l'échange de secrets a un coût en terme de performance car nécessite des allers-retours client / serveur supplémentaires.

Quelle donnée importante ne cache pas HTTPS ?





HTTP

HTTP/2

HTTP/2 convertit le payload en binaire et introduit plusieurs concepts importants tels que le multiplexing et le server-push. HTTP/2 n'est supporté que via HTTPS dans la majorité des navigateurs du marché, le rendant obligatoire de fait.

A quoi sert le multiplexing ?

Quel est le changement de paradigme porté par le server-push ?



HTTP

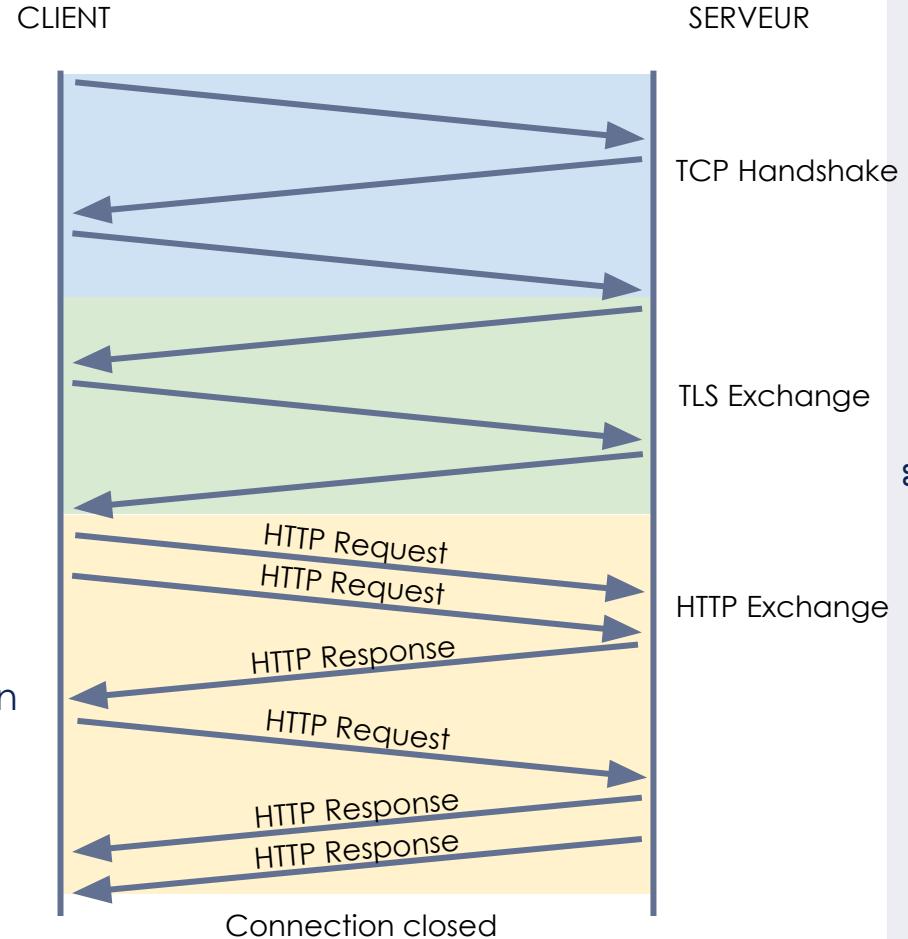
HTTP/2 : multiplexing & server push

Le multiplexing permet d'utiliser le tunnel de connexion déjà ouvert pour faire transiter plusieurs requêtes / réponses HTTP.

Cela permet d'économiser le temps de handshake TCP et d'échange TLS entre chacune des requêtes après la première.

Le serveur-push permet au serveur, **sans demande préalable** du client, d'envoyer des ressources supplémentaires sur une connexion déjà établie.

(Par exemple une feuille CSS avec la page HTML qui va avec, ou les sous-ressources des résultats d'une requête sur une API)





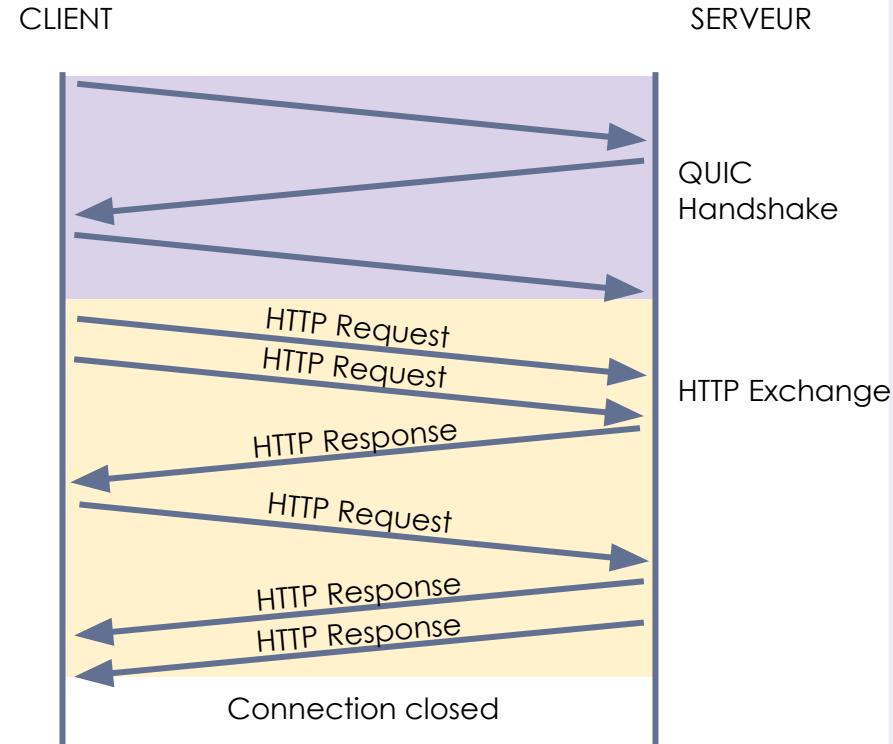
HTTP

HTTP/3 : HTTP over QUIC

Actuellement en draft mais déjà en production entre Chrome et les services Google.

QUIC permet les échanges sécurisés via le protocole UDP et promet une connexion sécurisée plus rapide que TCP + TLS.

Encore une fois dans le but de réduire la latence avant le TTFB (Time To First Byte).





Exercice

Requêtes & Headers HTTP

A l'aide d'un des outils installés précédemment,
requêter la homepage de Google :

- Quelle est la requête minimale que je peux envoyer pour obtenir une réponse ?
- Quel charset utilise la homepage Google ?
- Quelle est la valeur du Cache-Control ?



03

Les applications web



API - Définition



En informatique, une interface de programmation applicative (souvent désignée par le terme API pour Application Programming Interface) est un ensemble de classes, de méthodes, de fonctions qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.



L'API est l'industrialisation des processus de consommation des services de l'entreprise à travers le WEB

WEB	Industrialisation
+ HTTP + REST	+ Portail développeur + TTFAC* & DX** + X-device / X-channel + API Management

* "Time To First API Call" est le temps nécessaire pour qu'un développeur consomme une API après avoir lu la documentation sur le portail développeur ! Nous ciblons un TTFAC de 5 minutes.

** "Developer experience". Une API est utilisée par des humains. Elle doit avoir pour objectif une adoption massive et être conçue avec amour.



Les niveaux d'ouverture d'API

Ouverture du SI

Industrialisation de la consommation de l'API
design externalisable



Objectifs

- Rationaliser sur les ressources et services du cœur de métier
- « Embracer » les standards et la performance des architectures WEB

Enjeux

- **Design** : designer et concevoir des API RESTful selon l'état de l'Art
- **Architecture** : Proposer des ressources scalables, stateless, de granularité moyenne, asynchrone,...



Objectifs

- Proposer à des **partenaires** de consommer les API pour développer son business
- Répondre à la profusion de **terminaux émergeants** sur le WWW

Enjeux

- **Sécurisation** OAuth2 : AAA (Authentication, Authorization, Accounting)
- **Portail développeur** et Portail d'API management



Objectifs

- Source de revenu directe
- Attirer des utilisateurs pour rendre un service incontournable
- Outsourcer l'innovation / Faire émerger de nouveaux usages
- Se reconcentrer sur son cœur de métier

Enjeux

- Portail développeur et Portail d'API management
- Affordance de l'API (qualité du design)
- Aspects Communautaires : Doc, Events

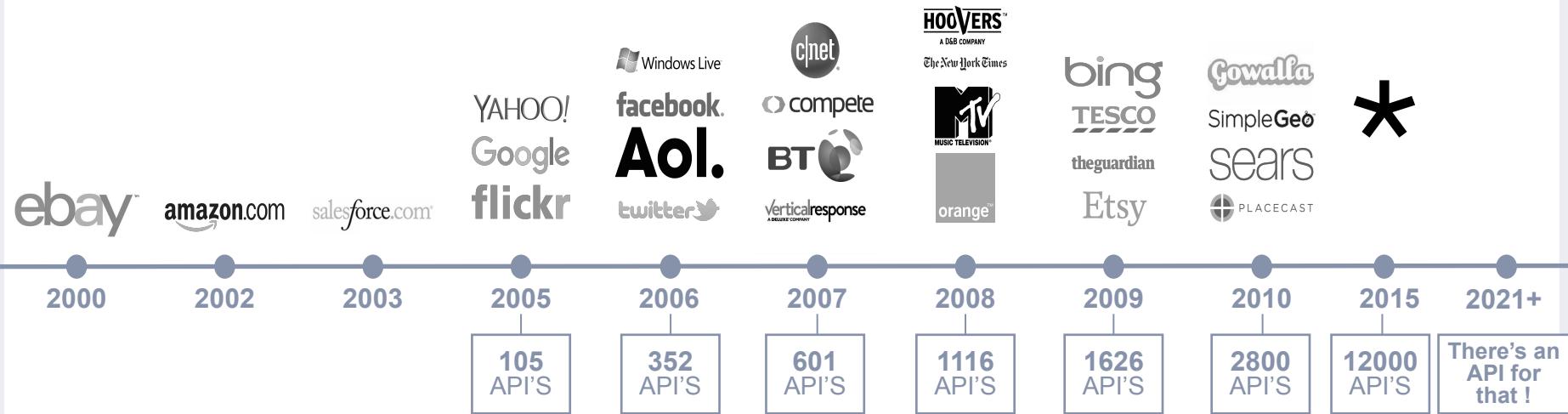




Chronologie des API

Initiée par les géants du Web

OCTO ACADEMY - Learn to change





Les usages numériques sont en constante évolution

ATAWAD

De nombreux nouveaux canaux de communication et de nombreux appareils sont apparus ;
La manière dont nous concevons les logiciels aujourd'hui a beaucoup changé ;
Si l'Internet des Objets rencontre la même adoption, de nouveaux usages numériques apparaîtront.



bradfrostweb.com



bradfrostweb.com



bradfrostweb.com

Ces différents canaux et différents appareils sont résumés à l'aide de l'acronyme **ATAWAD** :
> « Anytime, Anywhere on Any Device »



Les usages numériques en constante évolution

DIGITALIZATION

AnyTime, AnyWhere, Any Device



La même scène au Vatican, avec 8 ans d'écart

Le pape François utilise une API pour tweeter (avec TweetDeck)

The screenshot shows a Twitter interface with a search bar at the top. Below it, a tweet from Pope Francis (@Pontifex) is displayed. The tweet reads: "Prayer reconnects us to God, charity to our neighbor, fasting to ourselves. God, my brothers and sisters, my life: these are the realities that do not end in nothing, and in which we must invest. #Lent". The tweet was posted at 1:30 PM · 17 mars 2019 · TweetDeck. It has 5,7 k Retweets and 25,5 k J'aime. Below the tweet, three replies are shown:

- キヨンシー** @wg95945TGPjFgyb · 18 mars
En réponse à @Pontifex
神が教えてくれていると思います、チャリティーアイベントなどの大切さを感謝ですね。 🙏
- Terri King** @naturegurl4 · 18 mars
En réponse à @Pontifex
Amen.
- Regina Oyonwo** @oyonwo · 17 mars
En réponse à @Pontifex
Amen



Open API

Visionnaires



« All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions. Anyone who doesn't do this will be fired. Thank you; have a nice day! »



Jeff Bezos

CEO, Amazon

Communication interne – 2002



Création de
nouveaux business
models



« Externaliser »
l'innovation



La vision technologique d'Octo

L'état de l'art des architectures de SI aujourd'hui

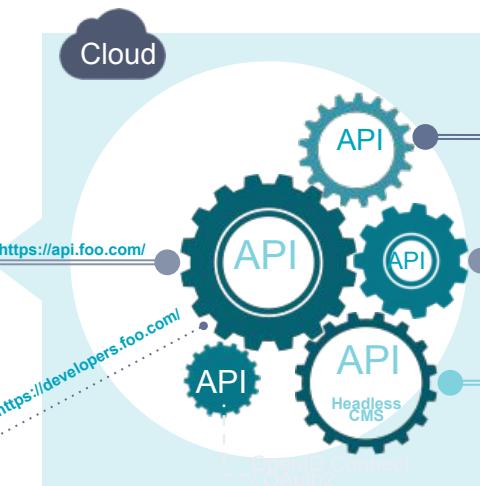
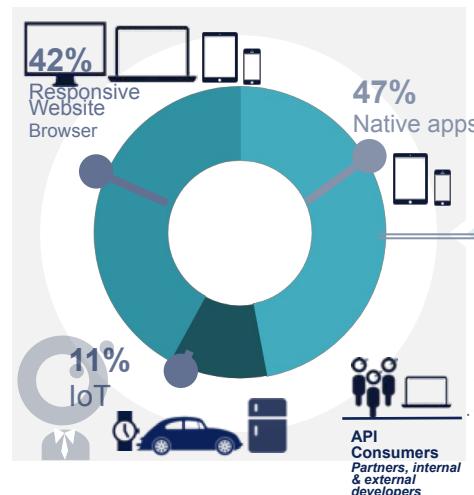
WOA

Web Oriented Architecture

Interfaces
multiples et remplaçables

Microservices
évolutifs et RESTful

Stockage
distribué





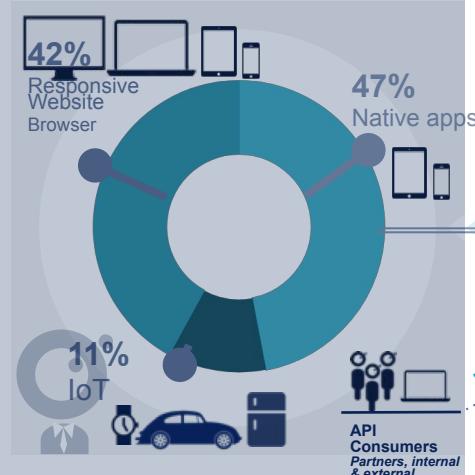
La vision technologique d'Octo

L'état de l'art des architectures de SI aujourd'hui

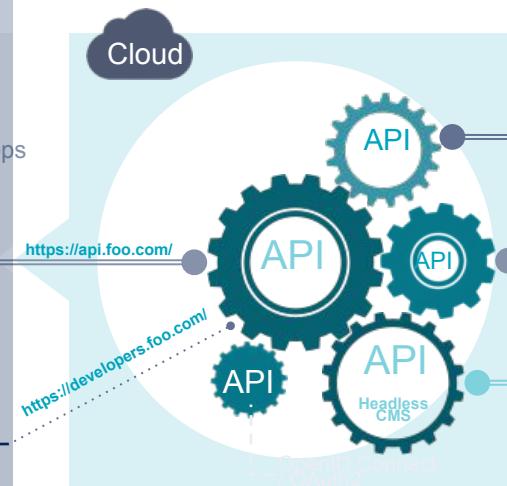
WOA

Web Oriented Architecture

Interfaces
multiples et remplaçables



Microservices
évolutifs et RESTful



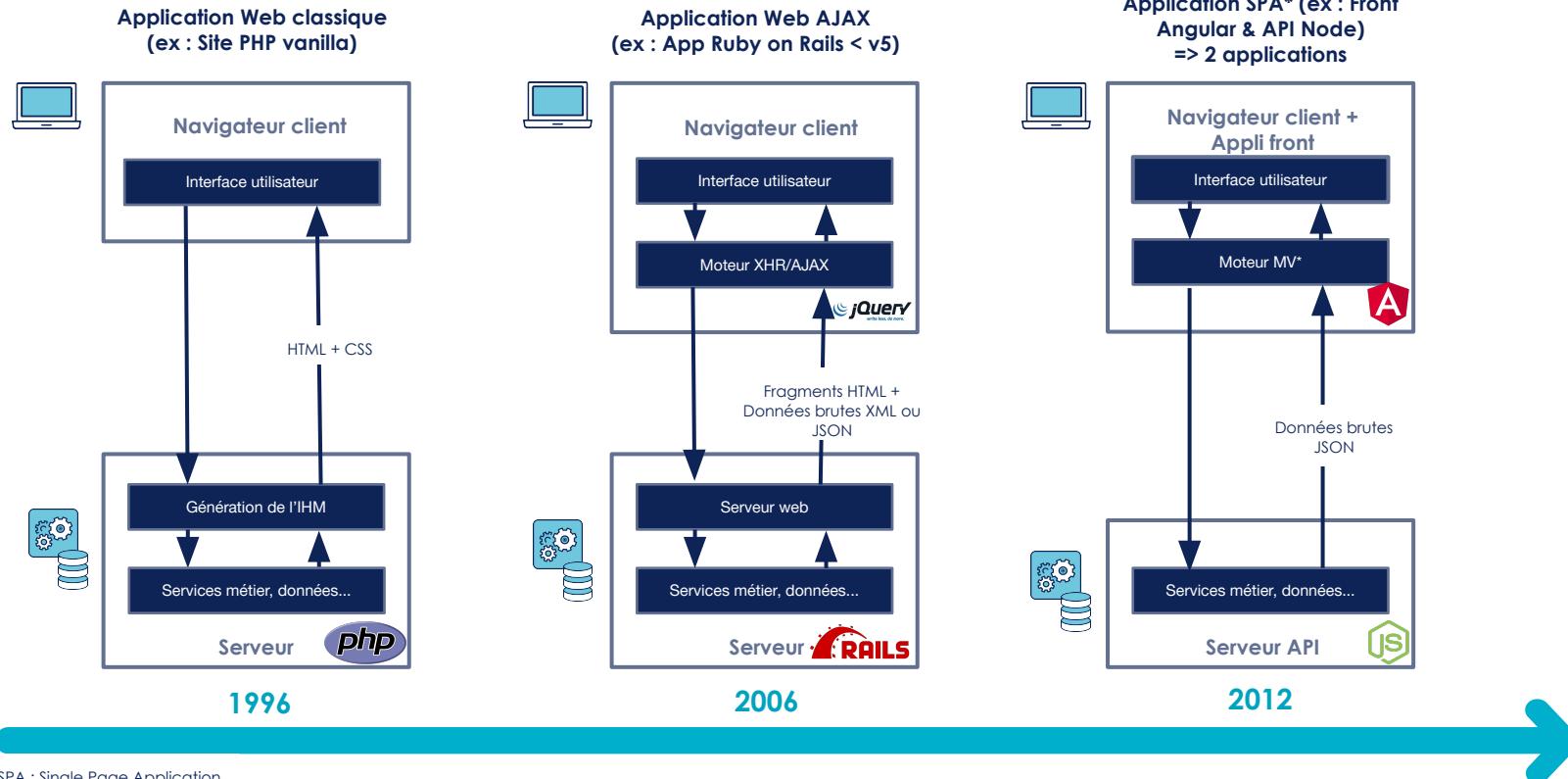
Stockage
distribué



Les API sont la colonne vertébrale des architectures WEB



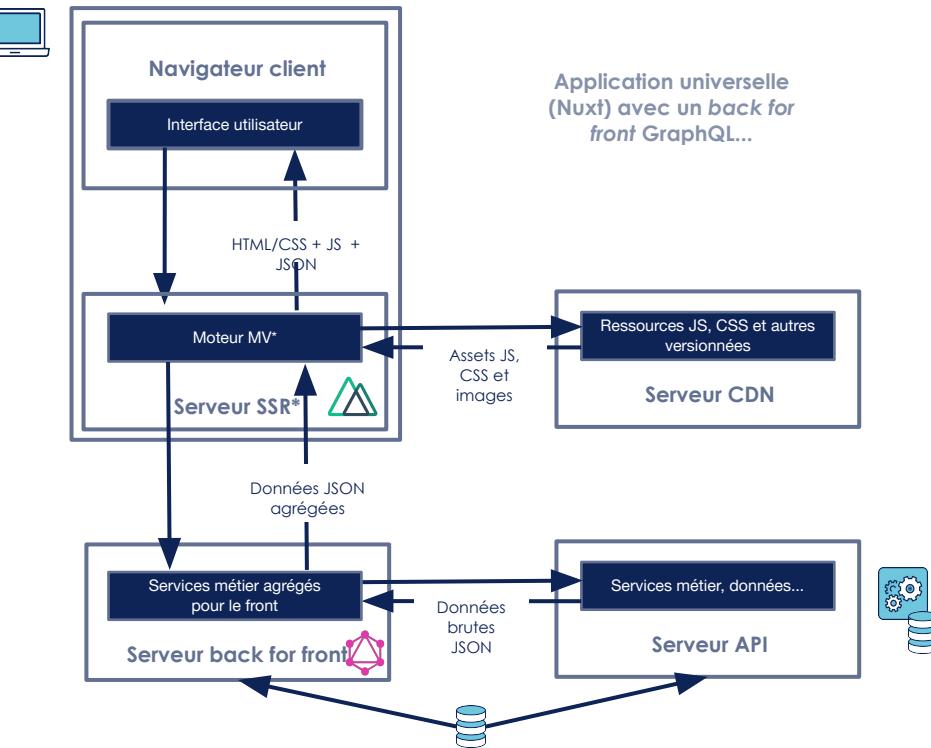
Evolution des applications web



SPA : Single Page Application
MV : Moteur Vue



Evolution des applications web



MV : Moteur Vue
SSR : Server-side Rendering

2022





04

Design



REST, c'est quoi ?

- **R**Epresentational State Transfer
- C'est un style architectural
- Défini par **Roy Fielding** en 2000^[1] :
 - > Core contributor de la spec HTTP
 - > Co-fondateur du serveur Apache
 - > Co-fondateur de l'Apache Foundation...
- 5 points
 - > Client-serveur
 - > Sans état (stateless)
 - > Informations de cache transmises par le serveur
 - > Découpe en ressources hypermedia
 - > Interface uniforme



Design API

- Qu'est-ce qu'un bon design ?
- HTTP : codes & verbes
- JSON
- Query strings
- Versioning
- Pagination
- Content Negotiation
- ...



OCTO Technology

RESTful API

General concepts

KISS

Anyone should be able to use your API without having to refer to the document.

- Use standard, concrete and shared terms, not your specific business terms or acronym.
- Never allow application developers to do things more than one way.
- Design your API for your clients (Application developers), not for your data.
- Target major use cases first, deal with exceptions later.

OCTO Technology

Query strings

Search

You may use the "Google way" to perform a global search:

```
GET /search?q=running+paid
```

Filters

You should use "?" to filter resources

```
GET /orders?state=paid
```

or (multiple URLs may refer to the same resource)

```
GET /users/007/orders?state=paid
```

Pagination

You may use a range query parameter. Pagination is not defined, for example : range=0-25.

The response should contains the following headers :

```
Link: <https://api.fakecompany.com/v1/orders;range=0-25>;rel="self"
```

Note that pagination may cause some unexpected behavior.

206 Partial Content

Content-Range: 48-55/971

Accept-Range: order 10

Link: <https://api.fakecompany.com/v1/orders;range=0-25>;rel="self"

<https://api.fakecompany.com/v1/orders;range=26-55>;rel="next"

<https://api.fakecompany.com/v1/orders;range=0-55>;rel="last"

Partial responses

You should use partial responses so developers can select (essential for mobile development).

```
GET /users/007?fields=firstname,name,address
```

200 OK

```
{ "id": "007", "name": "John Doe", "address": "123 Main Street", "email": "john.doe@example.com", "phone": "+1 555-1234", "status": "active", "created_at": "2023-01-01T00:00:00Z", "updated_at": "2023-01-01T00:00:00Z" }
```



Design API

Qu'est-ce qui fait un "bon" design ?

Une API bien designée :

- Suit les standards HTTP
- Suit les tendances des grands acteurs du Web
- Est simple et compréhensible par des acteurs extérieurs
- A une bonne affordance*

*Affordance : capacité d'un outil à indiquer, par son aspect et sa forme, comment il peut être utilisé





Design API

Trois approches

1. L'approche "puriste"

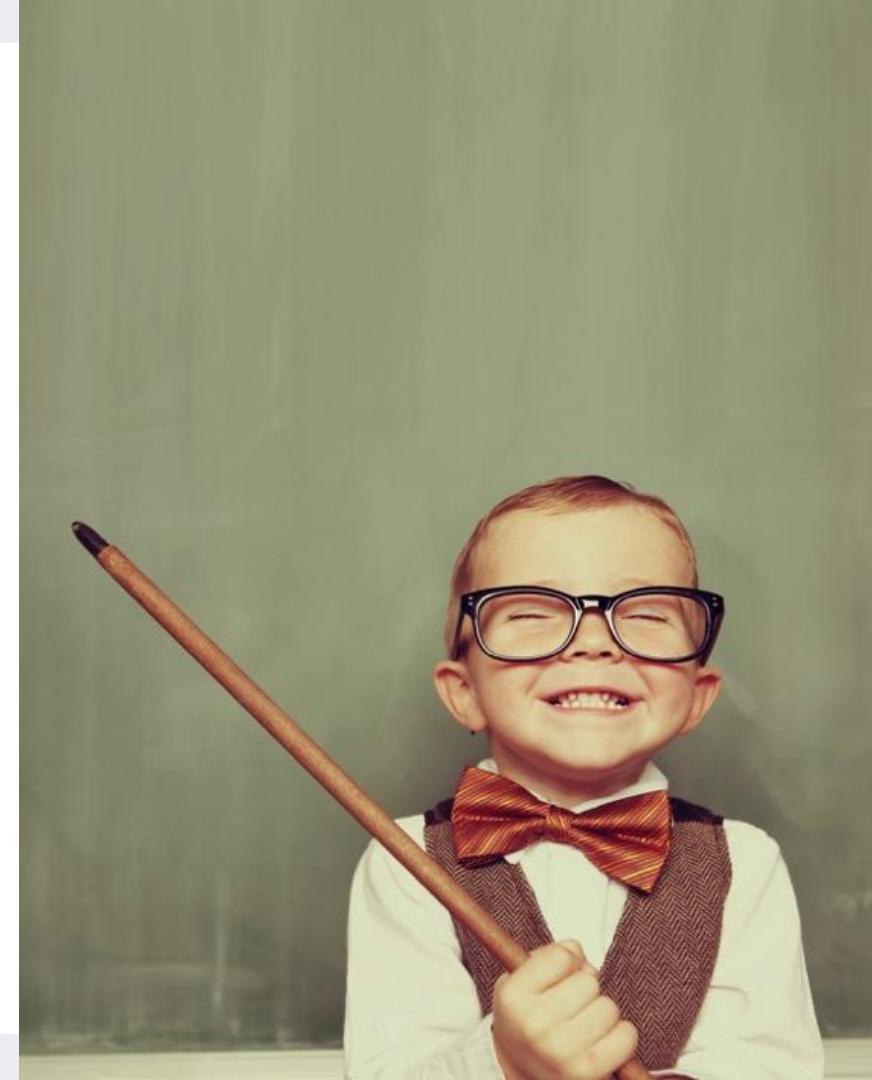
- > RESTful API telle que décrite dans les documents de référence (Roy Fielding, Leonard Richardson, Martin Fowler, spécification HTTP, etc.)

2. L'approche "vite fait"

- > Développer une API qui fonctionne mais loin des standards (mauvaise Developer Experience)

3. L'approche "pragmatique"

- > Compromis, tel que pratiqué par les Géants du Web (suit la plupart des standards, mais adapté aux besoins)





Design API

C'est en forgeant qu'on devient forgeron

- **Les spécifications HTTP ne fournissent pas toujours une réponse aux problèmes de conception**

- > Certains choix de conceptions doivent être faits en fonction de nos convictions
 - > S'inspirer des choix de conception des Géants du Web est plutôt une bonne idée
"If it's good enough for Google, it's good enough for me"

- **Une API n'est jamais parfaite dès sa V1**

- > Il faut décider d'un design
 - > Déetecter les douleurs à l'usage, et s'améliorer dans une V2 (et ainsi de suite)





Design API

Vocabulaire ?

- **Être cohérent au niveau de la langue**
 - > N'utiliser que des noms communs
 - > Si open API internationale, utiliser de **l'anglais** :
 - + customers, orders, products...
 - > Si service public, on peut utiliser du **français** (surtout si les traductions peuvent porter à confusion à propos des termes métiers) :
 - + villes, entreprises, évènements...
- **Ne pas utiliser de jargon interne ou des acronymes**



Kaltxì*

*hello



Design API

Pour les développeurs





Design API

Simplicité

Ne jamais donner deux possibilités de faire la même action.





Design API

Status codes

Quand **tout s'est déroulé sans accroc**, utiliser un des codes HTTP suivants :

- **200 “OK”** : Générique pour tout ce qui est un succès, par exemple pour GET/PUT/PATCH
- **201 “Created”** : Une ressource a été créée. Souvent en réponse d'un POST, parfois d'un PUT
- **202 “Accepted”** : La requête a été acceptée et commence son traitement (possiblement long)
- **204 “No Content”** : La requête a fonctionné, mais il n'y a pas de contenu de retour. Souvent utilisé suite à un DELETE.
- **206 “Partial Content”** : La ressource retournée est incomplète. Par exemple, en cas de pagination.



Design API

Status codes

Quand **le client a fait une erreur**, ou doit changer ses paramètres de requête :

- **400 “Bad Request”** : Les paramètres envoyés sont mauvais, ou une règle métier empêche le fonctionnement
- **401 “Unauthorized”** : On ne sait pas qui fait l'appel
- **403 “Forbidden”** : L'utilisateur n'a pas le droit d'effectuer cette action
- **404 “Not Found”** : La ressource demandée n'existe pas
- **406 “Not Acceptable”** : Erreur au niveau des Headers Accept-*. Par exemple, demander du XML au lieu du JSON
- **409 “Conflict”** : L'action rentre en conflit avec l'état actuel du serveur
- **410 “Gone”** : Cette ressource existait mais n'existe plus



Design API

Status codes

- **418 “I'm a Teapot”** : ...
- **422 “Unprocessable Entity”** : La requête a une syntaxe correcte mais une sémantique erronée (pour les erreurs métier ?)
- **429 “Too Many Requests”** : Le client a effectué trop de requêtes par rapport à ce qu'il est autorisé. Il doit réessayer plus tard.



Design API

Status codes

Quand l'API / le serveur a des **problèmes techniques** :

- **500 “Internal Server Error”** : Un cas inattendu a empêché la requête d'aboutir
- **502 “Bad Gateway”** : Le système sous-jacent ne répond pas
- **503 “Service Unavailable”** : Une surcharge ou une maintenance rend le service indisponible pour l'instant



Design API

Status codes

- Vous pouvez utiliser le type d'erreur suivant (défini par la spécification OAuth 2.0) :

```
{  
  "error": "USER_UNAVAILABLE", // short description  
  "error_description": "This user is not available for now", // description, Human-readable  
  "error_uri": "https://myservice.com/errors/USER_UNAVAILABLE" // link to detailed error  
description on the developer portal  
}
```

- Vous pouvez renvoyer un tableau de ce type d'erreur

> Pratique dans le cas d'une validation de formulaire par le serveur



Design API

Noms de domaine

Production

API – <https://api.fakecompany.com>

OAuth2 – <https://oauth.fakecompany.com>

Portal developer – <https://developers.fakecompany.com>

Sandbox

OAuth2 – <https://oauth.sandbox.fakecompany.com>

API – <https://api.sandbox.fakecompany.com>





Design API

Verbes

SOAP/EJB/RPC

```
getClient(1)  
creerClient()  
majSoldeCompte(1)  
ajouterProduitDansCommande(1)  
effacerAdresse(1)
```

RESTful

```
GET      /customers/1  
POST    /customers  
PATCH    /accounts/1  
PUT      /orders/1  
DELETE    /addresses/1
```

- Utiliser des noms, pas des verbes
- HTTP est un protocole applicatif
 - > Ne pas réinventer la roue à chaque fois et créer des "protocoles applicatifs spécifiques"



Design API

CRUD

- Obtenir une collection

```
GET https://api.company.com/v1/customers/007/orders  
200 OK  
[  
    {"id":"1234", "state":"paid"},  
    {"id":"5678", "state":"running"}  
]
```

- Obtenir une instance

```
GET https://api.company.com/v1/orders/1234  
200 OK  
{"id":"1234", "state":"paid"}
```

- Supprimer une instance

```
DELETE https://api.company.com/v1/orders/1234  
204 No Content
```

- Créer une instance

```
POST https://api.company.com/v1/orders  
{"state":"running", "id_customer":"007"}  
201 Created  
Location: https://api.company.com/v1/orders/1234
```

- Créer une instance, avec un ID défini par le client

```
PUT https://api.company.com/v1/orders/1234  
201 Created
```

- Mise à jour complète

```
PUT https://api.company.com/v1/orders/1234  
{"state":"paid","id_customer":"007"}  
200 OK
```

- Mise à jour partielle

```
PATCH https://api.company.com/v1/orders/1234  
{"state":"paid"}  
200 Ok
```

<https://blog.octo.com/should-i-put-or-should-i-patch/>



Design API

Granularité intermédiaire

- Utiliser une granularité moyenne, plutôt que fine ou grossière
- Les ressources ne devraient pas avoir plus de 3 niveaux de profondeur

```
GET https://api.fakecompagny.com/v1/customers/007
200 OK
{
  "customer_id": "007",
  "firstname": "James",
  "name": "Bond",
  "address": {
    "street": "H.Ferry Rd.",
    "city": {
      "city_id": 7
      "name": "London"
    }
  }
}
```





Design API

Pluriel vs singulier

- Utiliser des noms au **pluriel** plutôt qu'au singulier pour manipuler chaque ressource
 - > Récupérer une collection :
 - + GET /users
 - > Créer une instance :
 - + POST /users
 - > Récupérer une instance :
 - + GET /users/007
 - + et pas **GET /user/007**





Design API

Casse

- Vous pouvez choisir le `snake_case` ou le `camelCase` pour vos attributs et paramètres, mais restez **homogène**.
- Si il y a plus d'un seul mot dans l'URL, vous devriez utiliser le `spinal-case` ou le `snake_case`
 - > `POST /v1/specific-orders`
 - > `POST /v1/specific_orders`
 - > (pas de `camelCase` car certaines configurations serveurs ignorent la casse)
- Pour la casse du body, on recommande le `snake_case` ou le `camelCase`.
 - > `GET /orders?id_customer=007`
 - > `POST /orders {"id_customer": "007"}`





Exercice

Utilisation d'API

A l'aide d'un des outils installés précédemment,

requêter l'API suivante : [la documentation de la Punk API](#)

Obtenir les informations suivantes :

- l'id technique de la bière nommée "Ace Of Equinox"
- le nombre de bières qui se marient bien avec un crumble de pommes (Apple_crumble)
- la liste des bières brassées avant mai 2007
- la liste des bières confectionnées via un houblon de type "Columbus" et qui se marient bien avec les "taco"



Design API

Versioning (1/2)

- Le numéro de version devrait être obligatoire dans les URL, par API et pas par route (versions majeures).
- On peut assurer le support d'au plus deux versions en même temps (les applications natives ont besoin de plus de temps)
 - > Mettre en place une doc exhaustive des changements entre les deux versions

~~GET /v1/orders~~

GET /v2/orders

GET /v3/orders





Design API

Versioning (2/2) - Gestion des clients

- Nécessite de prévenir suffisamment à l'avance (3 mois - 6 mois) les consommateurs d'API de la suppression d'une version
 - > Nécessite d'avoir la liste exhaustive des contacts pertinents chez ses clients
- Nécessite d'avoir un monitoring de l'utilisation de l'API au jour le jour (APIM)
 - > Si la version n'est plus utilisée, on peut la supprimer
 - > Si la version est encore utilisée par 50% des clients le jour de la suppression : ... on va décaler la date de suppression !
- Attention aux apps mobiles qui utilisent la vieille version si non mises à jour





Design API

Filtres

- Vous devriez utiliser les query parameters pour filtrer des ressources
 - > GET /orders?state=paid&id_user=007
 - > GET /users/007/orders?state=paid





Design API

Réponses partielles

- Utiliser les réponses partielles pour donner la possibilité aux développeur de ne sélectionner que les informations dont ils ont réellement besoin, pour optimiser la bande passante (crucial pour le développement mobile)

GET

```
https://api.fakecompany.com/v1/customers/007?fields=firstname,name,address(street)
```

206 Partial Content

```
{ "id":"007",
  "firstname":"James",
  "name":"Bond",
  "address":{"street":"Horsen Ferry Road"}
}
```





Design API

Tri

- Utiliser `?sort=attribut1,attributN` pour trier des ressources
- Par défaut, les ressources sont triées par ordre croissant.
- Préciser avec `:asc` ou `:desc` la direction du tri (croissant ou décroissant)
- Exemple :
`GET /restaurants ?sort=rating:asc,reviews:desc,name:asc`





Design API

Pagination (1/3)

- On peut utiliser le query parameter `range`. La pagination est obligatoire et une pagination par défaut doit être définie, par exemple : `range=0-25`
- La réponse doit contenir les headers suivants :
`Content-Range`, `Accept-Range`
- Exemple

```
GET https://api.fakecompany.com/v1/customers?range=60-72
```

```
206 Partial Content
```

```
Content-Range: 60-72/46518
```

```
Accept-Range: customer 50
```

```
...
```

=> Confus et compliqué à utiliser en pratique





Design API

Pagination (2/3)

- On peut aussi raisonner en termes de **pages**
?page=19&per_page=50
- Les informations de pagination peuvent être dans le body de la réponse plutôt que les headers.
- La liste de résultats se trouve dans data.

```
1  {
2      "pagination": {
3          "page": 19,
4          "resultsCountPerPage": 50,
5          "resultsCountOnThisPage": 39,
6          "totalResultsCount": 939,
7          "totalPagesCount": 19
8      },
9      "data": [...]
2029
2030 }
```





Design API

Pagination (3/3)

- La pagination peut créer des comportements inattendus si beaucoup de ressources sont créées par minute.
 - > Cas de services ultra-réactifs comme Twitter
- On peut alors demander :
 - > X contenus à partir d'une **heure** donnée :
?since=2020-11-23T12:29:02
 - > X contenus à partir d'un **ID** donné :
?after=473812





Design API

Recherche

- Utiliser le mot-clef /search pour effectuer une recherche sur une ressource spécifique

```
GET  
https://api.fakecompany.com/v1/restaurants/search?type=thai
```

```
GET  
https://api.fakecompany.com/v1/offers/search?destination=gade  
loupe,Ile*  
200 Ok  
{  
    "count" : 5,  
    "query" : "destination=Gadeloupe,Ile*",  
    "suggestions" : ["Guadeloupe", "ile Galápagos"],  
    "results" : [...]  
}
```

- Vous pouvez utiliser "the Google way" pour effectuer une recherche globale parmi plusieurs ressources

```
GET /search?q=running+paid
```





Design API

Négociation de contenu

- La négociation du format (*content negotiation*) est gérée uniquement de manière RESTful via des headers
- Le client demande le format souhaité, via le header `Accept`, par ordre de préférence. JSON est le format par défaut.

```
Accept: application/json, text/plain  
et non pas  
/orders.json
```

```
GET https://api.fakecompany.com/v1/offers  
Accept: application/xml; application/json  
< 200 OK  
< [XML]
```

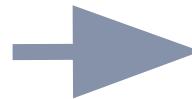




Design API

Hypermedia : exemple

```
{  
  "drink": "coffee",  
  "cost": 4.00  
}
```



```
{  
  "drink": "coffee",  
  "cost": 4.00,  
  "@controls": {  
    "self": {  
      "href": "http://api.coffee.com/orders/1234"  
    },  
    "update": {  
      "href": "http://api.coffee.com/orders/1234",  
      "method": "PUT",  
      "template": {  
        "drink": "coffee"  
      }  
    },  
    "pay": {  
      "href": "http://api.coffee.com/payment/order/1234",  
      "method": "PUT",  
      "template": {  
        "amount": 4.00  
      }  
    }  
  }  
}
```

Ressource : <https://blog.octo.com/transformez-votre-api-web-en-une-api-hypermedia/>



Design API

Scénarios sans ressources

- Dans quelques rares cas, nous pouvons considérer l'utilisation d'opérations ou de services, au lieu de ressources
- Vous pouvez dans ce cas utiliser une requête POST avec un verbe à la fin de l'URL

POST https://api.fakecompany.com/v1/calculator/sum

[1,2,3,5,8,13,21]

< 200 OK

< { "result" : "53" }

POST https://api.fakecompany.com/v1/customers/0007/carts/42/checkout

< 200 OK

< { "id_cart": "7", [...] }

Réfléchissez vraiment à
l'utilisation de ressources avant
de vous lancer dans cette
direction !



Exercice

Design d'API

A l'aide d'un des outils installés précédemment,

sur la base de documentation suivante : [Postman MyT&E](#)

Prenez 5 minutes pour parcourir la documentation.

- Qu'améliorerez-vous sur cette API, au regard des bonnes pratiques de design d'API vues durant ce chapitre ?



05

Architecture en pratique



Application ou architecture ?

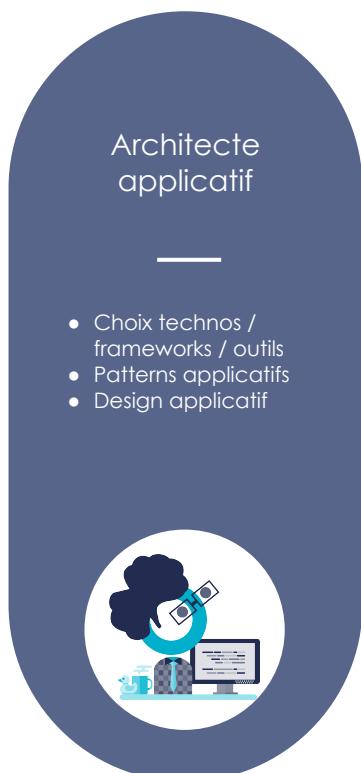
- Un projet qui grandit devient composé de plusieurs applications, services, et infrastructures qui sont en interaction
- Architecture = conception = design
Que ce soit au niveau d'une application, d'un projet, ou d'une entreprise.
- Dès qu'on réfléchit à l'agencement de plusieurs éléments techniques entre eux, on fait de l'architecture !



*La meilleure façon de manger un éléphant,
c'est bouchée par bouchée.*



Différents rôles d'architecte





Exercice

Site hébergement vidéos

- ➊ Le but est de rajouter une nouvelle fonctionnalité à un site qui héberge des contenus vidéos:
 - > Ajouter un commentaire à une vidéo
 - > Notifier par mail l'utilisateur propriétaire de la vidéo
 - > Récupérer les commentaires sur une vidéo
- ➋ Dessinez les communications entre le front, l'API et l'API tierce de mailing



Exercice

Quelques contextes mission

Les slides suivantes sont des cas d'architecture vus en mission (un peu adaptés).

Les cas sont découpés en 3 parties :

1. Présentation du cas (les grandes lignes)
2. Un schéma d'architecture sur lequel il faut réagir (**brainstorming**)
3. Une conclusion sur les remarques qu'on peut formuler



Quelques contextes mission

Sites de lecture de PDF



Sites de lecture de PDF

Brief

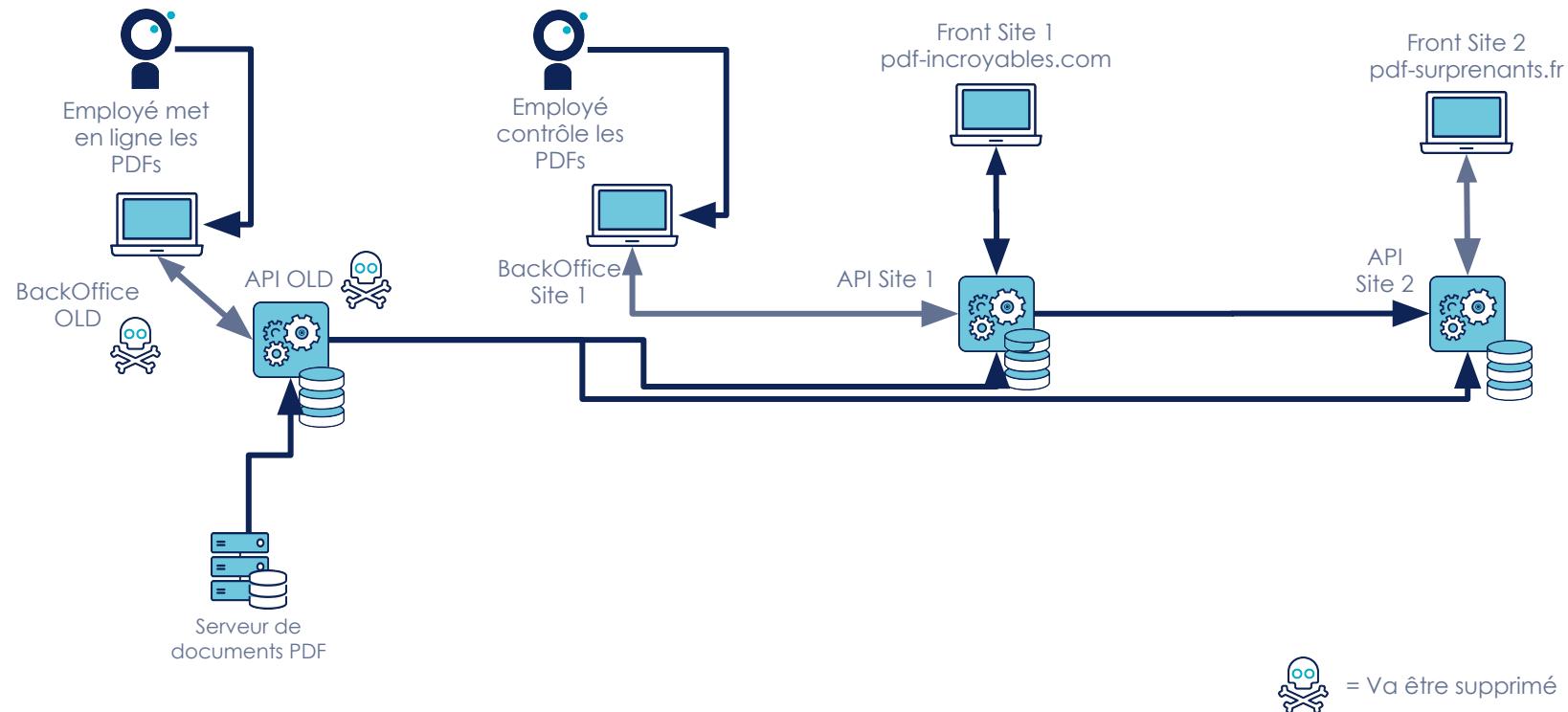
- Deux sites gérés par la même entreprise permettent à leurs visiteurs de consulter des fichiers PDFs très intéressants.
- Il n'y a pas forcément les mêmes PDFs affichés sur chaque site.
- Chaque site reçoit ses données d'une API qui lui est propre, mais le site 2 s'alimente de données du site 1.
- Un employé de l'entreprise met en ligne les PDF et un autre les contrôle après coup.
- Il y a un vieux back-office ("OLD") qui va bientôt être supprimé.

Que pensez-vous du schéma qui suit ?



Quelques contextes mission

Sites de lecture de PDF





Quelques contextes mission

Sites de lecture de PDF

Conclusion:

- Pourquoi les API des sites 1 et 2 s'alimentent de l'API OLD qui va être supprimée ? Pourquoi le serveur PDF qui est crucial est encore connecté à l'API OLD ? Vu les fortes dépendances, il semblerait que la migration / fermeture de ce vieux back-office ne soit pas encore pour tout de suite
- Pourquoi les employés qui mettent en ligne et qui contrôlent les PDF n'utilisent pas le back-office récent ? Cela crée encore beaucoup d'adhérence avec le back-office OLD.
- L'API 2 s'alimente de l'API 1 : il y a une hiérarchie/dépendance entre les deux applications ?
- Pas de brique API Management



Quelques contextes mission

Rédacteurs d'articles



Rédacteurs d'articles

Brief

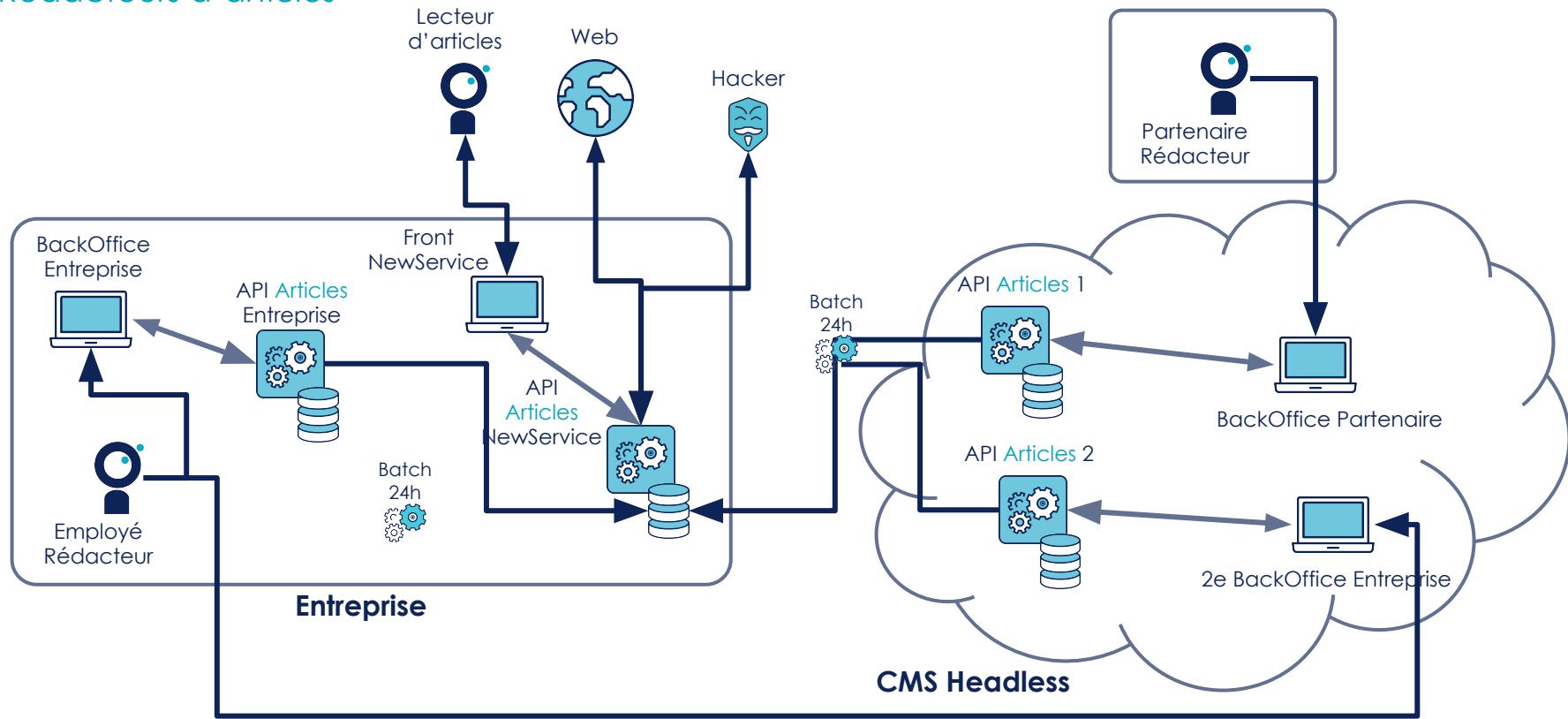
- Octo crée une application web de lectures d'articles ("NewService") pour une entreprise cliente
- Les employés de cette entreprise rédigent des articles sur un back-office interne
- Des partenaires à l'entreprise cliente doivent pouvoir aussi rédiger des articles (mais pas sur la même interface car pas la même politique de sécurité)
 - > Décision d'utiliser un "CMS Headless" qui fera office de back-office de rédaction

Que pensez-vous du schéma qui suit ?



Quelques contextes mission

Rédacteurs d'articles





Quelques contextes mission

Sites de lecture de PDF

Conclusion:

- Pourquoi un employé a accès à deux back offices pour le même type de contenus ?
- Les batches créent une attente / désynchronisation. Pas l'idéal, mais cela permet de pallier rapidement les systèmes sous-jacents du client (Ici : rendre cohérent les articles de 2 sources différentes, et problèmes de perfs de l'API entreprise).
- Il y a un hacker ?! Est-ce que l'API est sécurisée ? Est-ce qu'il a le droit de lire les articles depuis l'API ? Le schéma ne dit pas s'il y a la sécurité à tous les étages, ni si les articles sont en accès payant ou public.
- Pas de brique d'API Management



Exercice

Site e-commerce

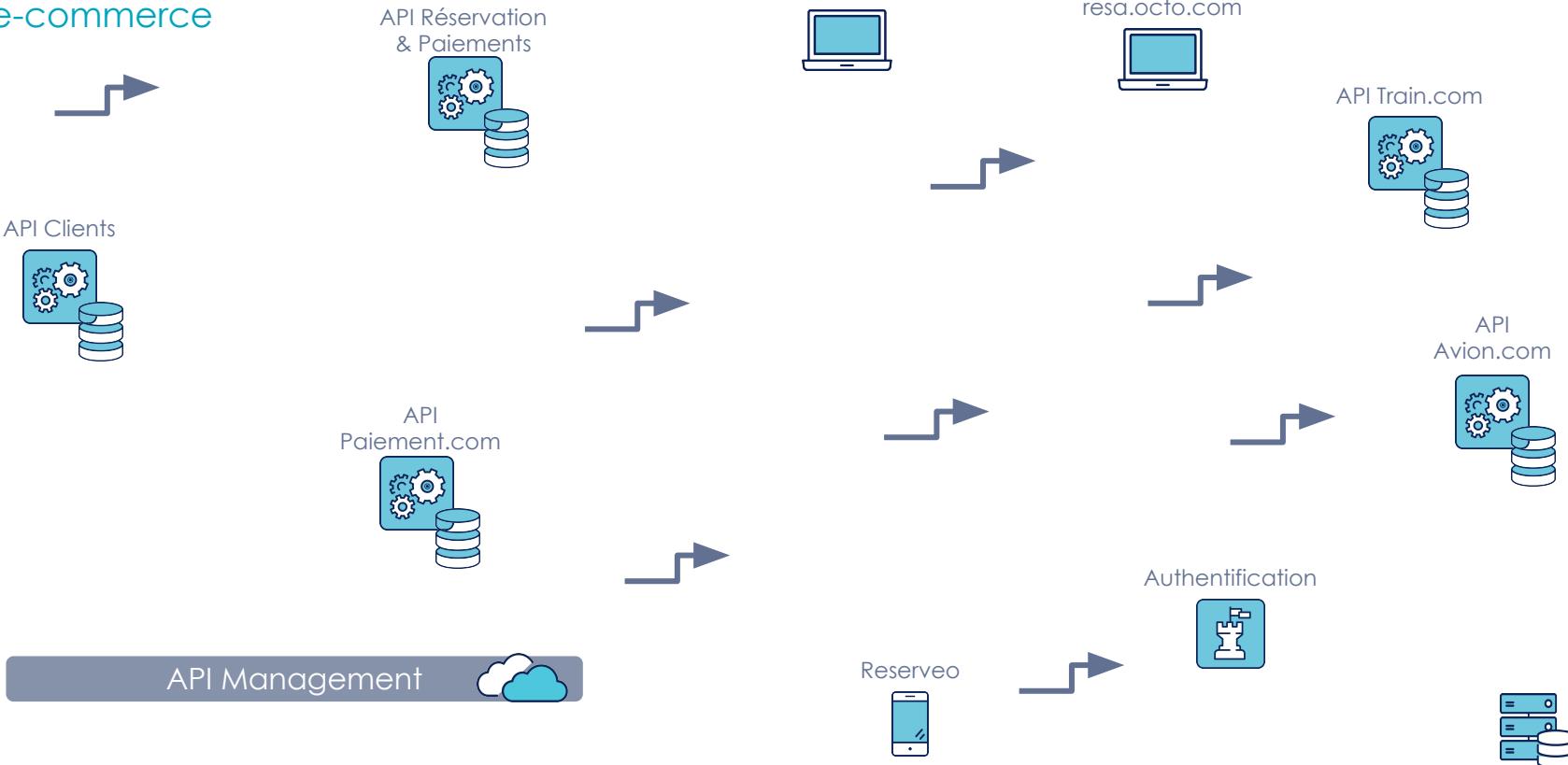
- Le but est d'établir le schéma d'un site e-commerce de réservation de trajets dans lequel est présent les briques suivantes :
 - > 2 apps web : resa.octo.com, resa.accenture.com
 - > 1 appli mobile : Reserveo
 - > 1 API commune de réservation et paiement
 - > 1 API commune pour les clients et leurs adresses
 - > Une solution d'API Management
 - > De l'authentification des utilisateurs
 - > Deux API tierces : une de réservation de billets de train (api.train.com) et une de billets d'avion (api.avion.com)
 - > Une API tierce de paiement (api.paiements.com)
- Séparez-vous en équipes de trois et dessinez le schéma d'archi correspondant !
Support à dupliquer :
<https://docs.google.com/presentation/d/1XBTdVu90nVoWpUCfbGySLEfC-MbTqHVFWbYm1q82GOI/edit?usp=sharing>



Exercice

Site e-commerce

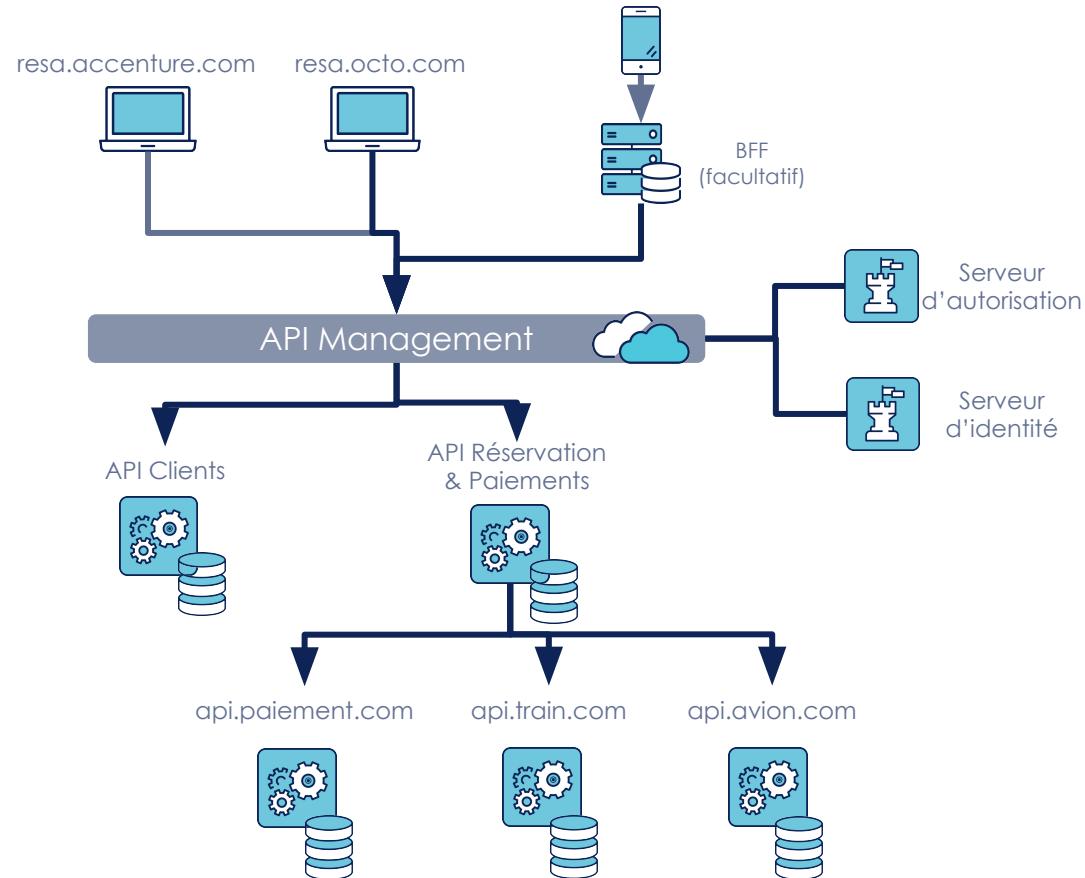
OCTO ACADEMY - Learn to change





Exercice

Site e-commerce





06

Sécurité et API Management



Introduction à la sécurité applicative

Brainstorming

A quoi vous fait penser la sécurité sur le web ?

Quelles sont les différentes façons de se connecter sur le web aujourd'hui ?



Introduction à la sécurité applicative

- Une stratégie API consiste à permettre à des développeurs de consommer des ressources
- Exposer ses API sur le web = faille de sécurité ?
Dur équilibre à trouver entre mener la vie dure aux attaquants potentiels et faciliter la vie des consommateurs légitimes
- Tout va bien si :
 - > on **authentifie** et on **autorise** les **utilisateurs**
 - > on **authentifie** et on **autorise** les **applications** qui consomment les API



Principes de sécurité applicative

Ressources publiques et privées

Toutes les ressources exposées par une API ne nécessitent pas le même niveau de vigilance.

On distingue deux types de ressources :

- **Ressources publiques**

- > Ex: /products, /catalogues, /offers, etc
- > Déjà publiées sur votre site Web sans mécanisme de sécurité
 - + Une **API_KEY** permet éventuellement de gérer la traçabilité

- **Ressources privées**

- > Ex: /customers, /carts, /payments, etc.
- > Nécessitent une connexion de la part de l'utilisateur
- > Une **API_KEY** permet de gérer la traçabilité
- > Mais il faut un mécanisme pour vérifier que l'utilisateur n'accède qu'à ses propres données : **OAuth2**



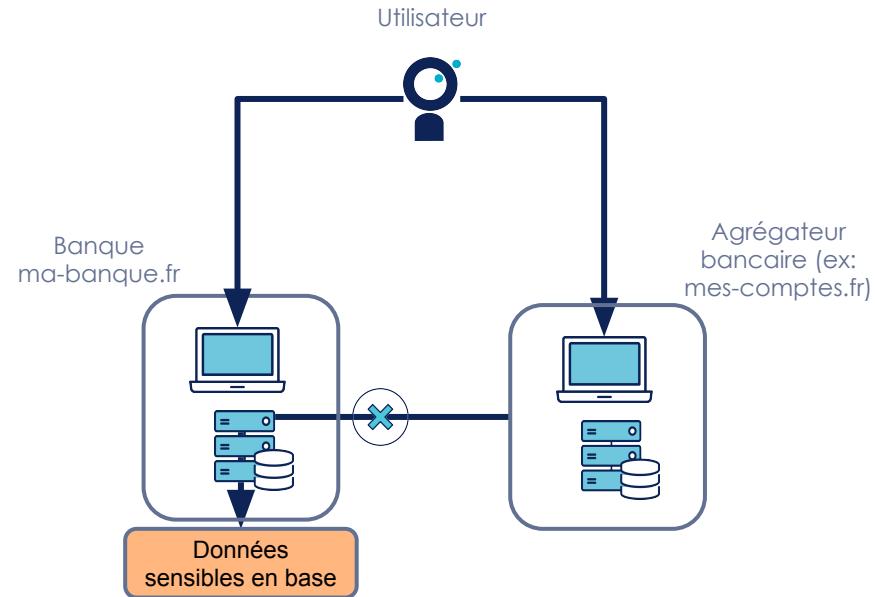
Principes de sécurité applicative

Ressources privées et clients externes

Une API peut être consommée par des clients externes (par ex: partenaires) !

Le propriétaire d'une ressource privée peut vouloir accéder à cette ressource via plusieurs clients d'API différents. Par exemple : un utilisateur qui utilise un agrégateur bancaire pour consulter tous ses comptes d'un coup d'oeil.

Il faut un moyen d'informer l'API de la banque que l'agrégateur bancaire est autorisé à accéder au compte.





Principes de sécurité applicative

3 piliers de la sécurité applicative

3 piliers, les “trois A” :

- Authentification (**Authentication**)

- > qui est l'utilisateur ?
 - > si l'authentification échoue : code d'erreur 401

- Autorisation (**Authorisation**)

- > qu'est-ce que l'utilisateur a le droit de faire ?
 - > si l'autorisation échoue : code d'erreur 403

- Traçabilité (**Accounting**)

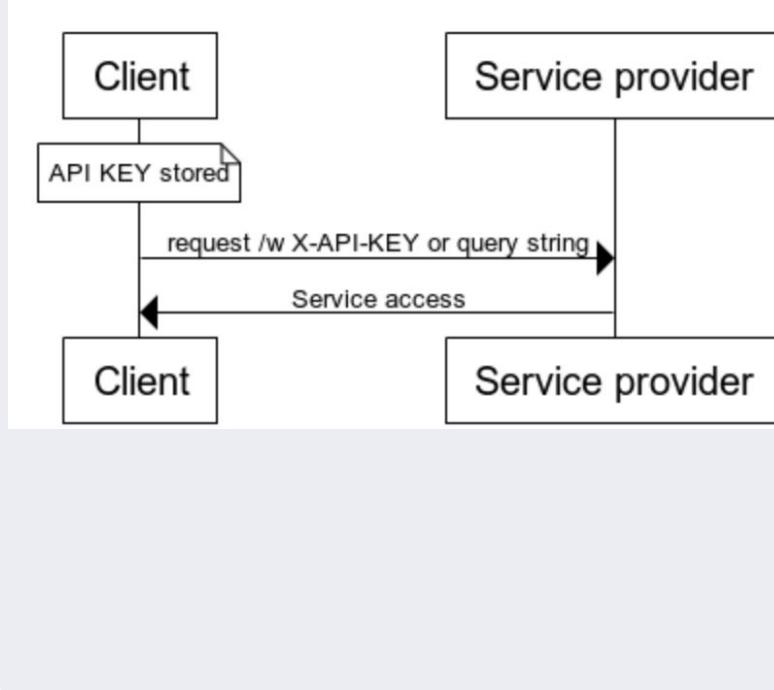
- > qui a fait quoi quand ?
 - > quel utilisateur, a effectué quelle action, sur quelle ressources, à quel moment ?



Ne pas confondre authentification et autorisation



API KEY



- Facile à mettre en place et à utiliser
- Pas de gestion fine de l'autorisation
- Pas de mécanisme d'expiration
- Secret partagé entre client et API => pas adapté aux SPA et apps mobiles
- Pas de standard partagé pour l'échange du secret



Principes de sécurité applicative

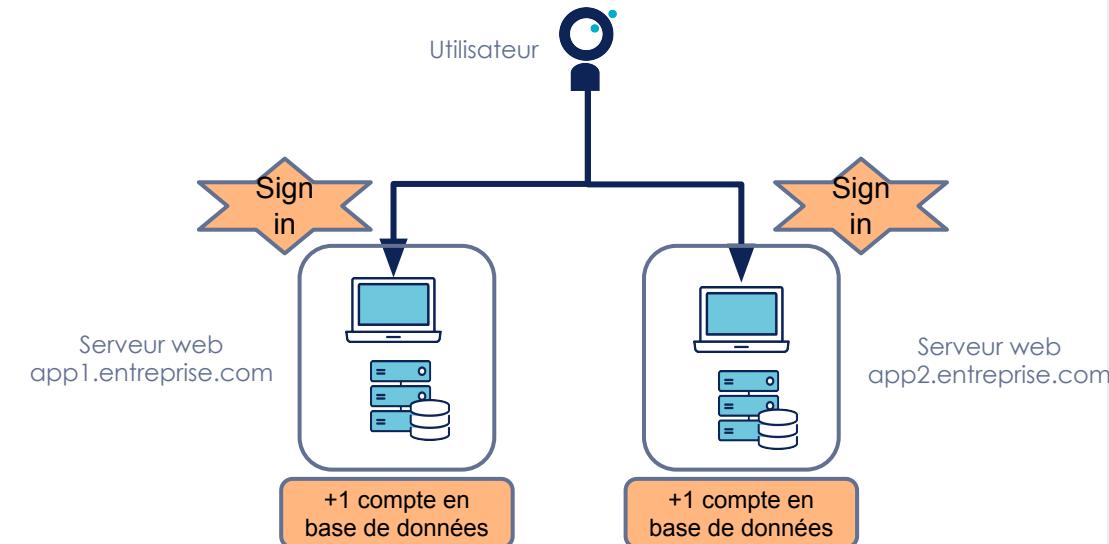
Single Sign On

Si chaque service d'une même entreprise embarque la logique de création de compte

- Compliqué et inutile pour les utilisateurs et employés
- Les informations sont éparpillées dans deux bases de données différentes

=> Besoin d'un seul point d'entrée :

Single Sign On (SSO)

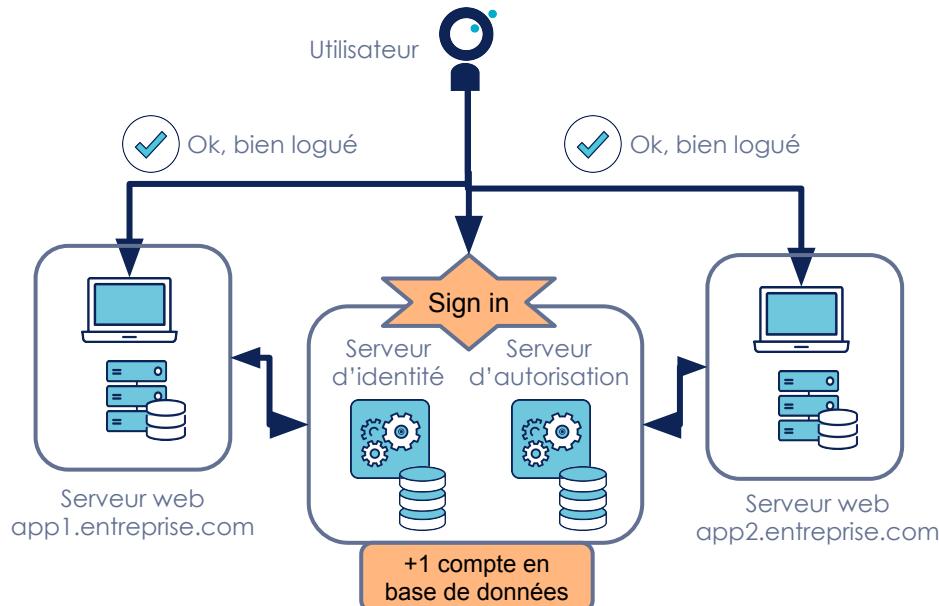




Principes de sécurité applicative

Serveur d'identité ?

- C'est à ce moment là qu'apparaissent les serveurs d'**identité** et d'**autorisation**.
- L'utilisateur ne se crée un compte qu'une fois auprès du serveur d'identité et sera considéré comme connecté sur toutes les applications de l'entreprise !
- Cela nécessite des **échanges sécurisés** entre les serveurs applicatifs, d'identité et d'autorisation.

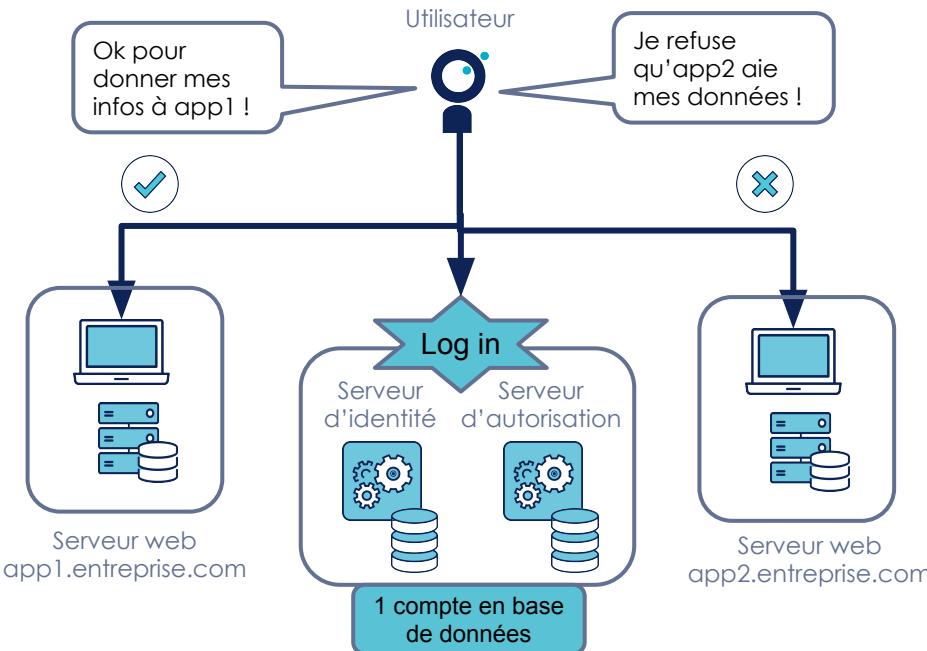




Principes de sécurité applicative

Autorisation ?

- L'utilisateur reste cependant maître de ses données. On lui demande son **consentement**, et il peut refuser le partage de tout ou partie des données à une autre application, au risque de ne pas pouvoir y accéder.
- Cette logique d'authentification et d'autorisation est tout l'objet du protocole **OAuth2** et de son extension **OpenID Connect (OIDC)**.





Sécurité API

OAuth/OAuth2

- OAuth (Open Authorization)
 - > Standard de gestion de l'autorisation
 - > Il offre deux fonctionnements possibles :
 - + 2-legged : autorisation du client
 - + 3-legged : autorisation du client ET de l'utilisateur
- OAuth a été poussé par les géants du Web
 - > Pour résoudre le problème suivant : permettre à des applications tierces d'appeler leurs API sans devoir partager les identifiants des utilisateurs
- OAuth et OAuth2 sont des standards IETF®
 - > OAuth (RFC5849) est déprécié par OAuth2 (RFC6749)... lui-même récemment remplacé par OAuth2.1





OAuth2

Glossaire OAuth

Concepts généraux

- **Client**

Une application qui effectue des requêtes sur des ressources privées au nom d'un Resource Owner avec son autorisation. Le terme "client" n'a aucune implication sur le type d'implémentation (serveur, desktop, mobile...)

- **Resource**

Server

Le serveur qui héberge les ressources, capable d'accepter et de répondre aux requêtes avec access token concernant des ressources privées.

- **Resource**

Owner/end-user

Entité capable de donner accès à une ressource privée. Appelé "utilisateur final/end-user" s'il s'agit d'une personne physique.

- **Authorization**

Server

Le serveur qui fournit les access token au Client après avoir authentifié le Resource Owner avec succès et obtenu son autorisation.

Concepts spécifiques au cas sans utilisateur final

- **Resource**

Owner/Client

Ici le terme "client" implique une implémentation côté serveur. Il s'agit du serveur qui effectue des requêtes vers des ressources privées... auxquelles il a accès car il s'agit ici également du Resource Owner.

- **Authorization**

Server

Le serveur fournit les access token au Client après l'avoir authentifié.

Optionnel

- **Identity**

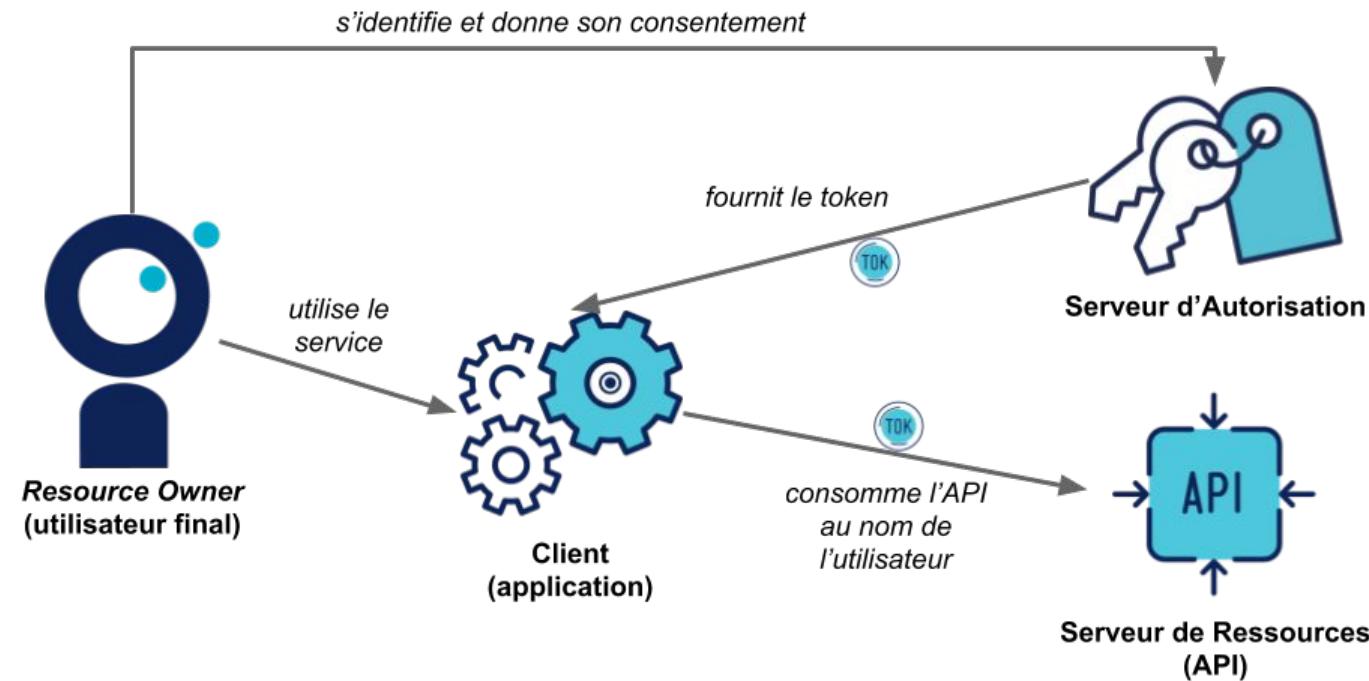
Server

Le serveur permettant d'authentifier l'utilisateur final



OAuth2

Les acteurs en présence





OAuth2

Les différents flow

3 flow à utiliser :

- **Client Credentials Grant** : cas où l'application cliente est aussi propriétaire des ressources (two-legged). De serveur à serveur, idéal pour les traitements par batch. Pas d'interaction avec un utilisateur final (seulement un "utilisateur technique").
- **Authorization Code Grant** : cas où l'application cliente a besoin d'accéder à des données d'un autre utilisateur (three-legged). (+ PKCE dans certains cas)
- **Device Grant** : cas où le client n'a ni navigateur ni clavier (Apple TV...)

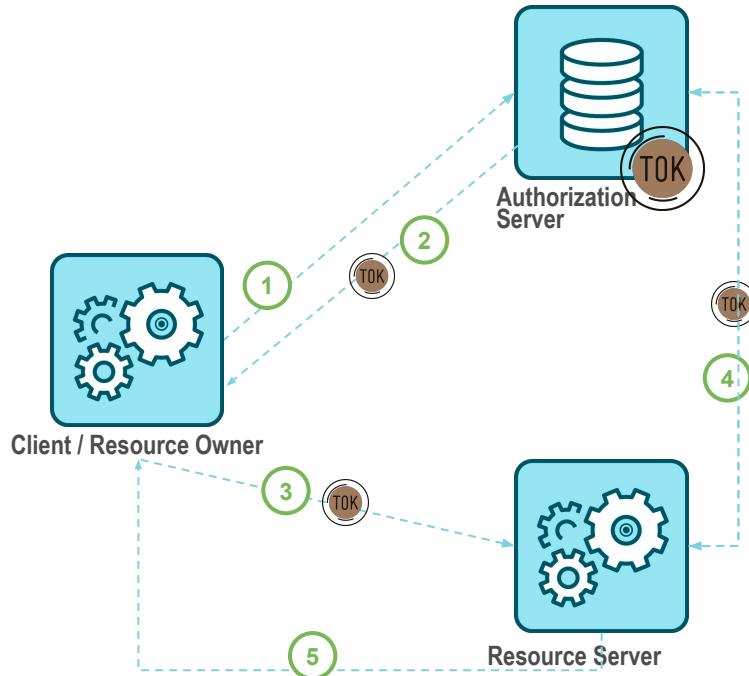
2 flows dépréciés pour des raisons de sécurité :

- **Implicit Grant** : pour les Single Page Applications (SPA)
- **Resource Owner Credentials Grant (ROPC)** : pour les applications legacy



OAuth2

Big picture - Sans utilisateur final



1. Le *client* demande un access token au serveur d'autorisation
2. Le serveur d'autorisation fournit le token
3. Le *client* fait une requête auprès du serveur de ressources, pour une ressource privée, en envoyant l'access token pour s'authentifier
4. Le serveur de ressources valide l'access token auprès du serveur d'autorisation
5. Si l'access token est valide, le serveur de ressources répond à la requête.

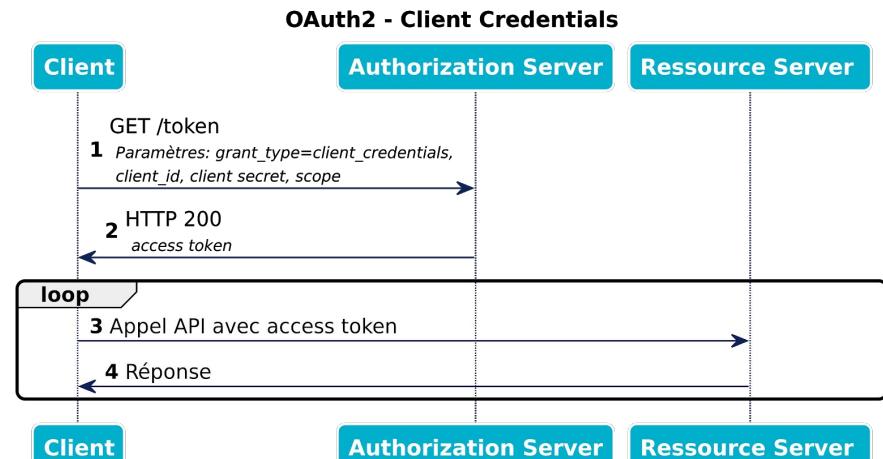


OAuth2

Client Credentials

Client Credentials Grant

- Le Client possède la donnée
- Pas d'interaction utilisateur : échange de serveur à serveur
- Spec: <http://tools.ietf.org/html/rfc6749#section-4.4>
- Exemple :
 - > cronjob interne de traitement de données à des fins statistiques





Exercice



Sécuriser une ressource privée avec Client Credentials

Objectif:

Sécuriser une API à l'aide du flow Client Credentials

- Utilisation d'Auth0 comme "Authorization server"
- Déclaration d'une API auprès de Auth0 qui aura le rôle de "resource server"
- Déclaration d'une application auprès de Auth0 qui aura le rôle de "client"

Résultat attendu:

Obtenir un access token avec scope autorisé





Exercice



Sécuriser une ressource privée avec Client Credentials

En tant que développeur d'API:

- Se créer un compte sur Auth0 <https://auth0.com/>
- Créer une API (Identifier https://test.fr) avec pour permissions/scope read:content et write:content
- Créer une Application type "Machine to Machine"
Avec le scope read:content uniquement

En tant que client de l'API:

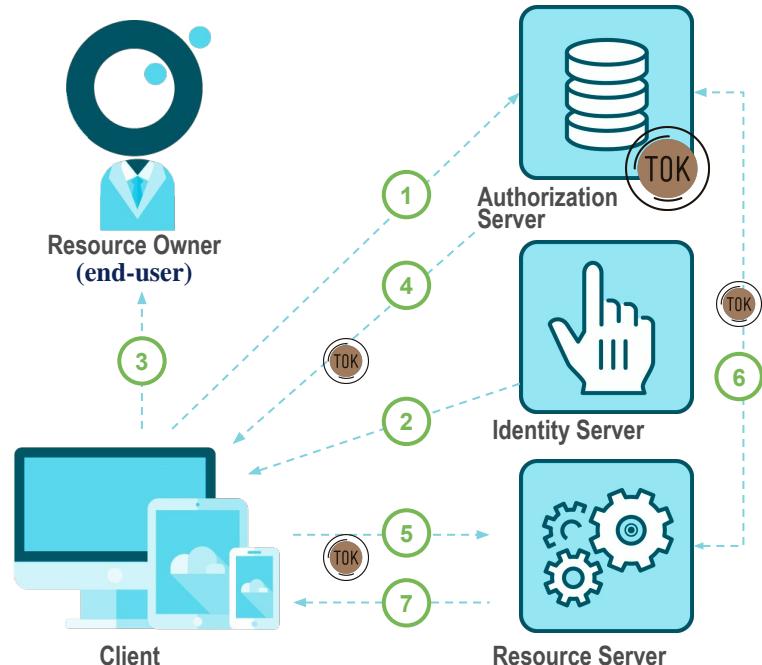
- Récupérer un token via **Client Credentials Flow**
(cf doc <https://auth0.com/docs/api/authentication>)
- Consulter le contenu de l'access token
sur <https://jwt.io>





OAuth2

Big picture - avec utilisateur final



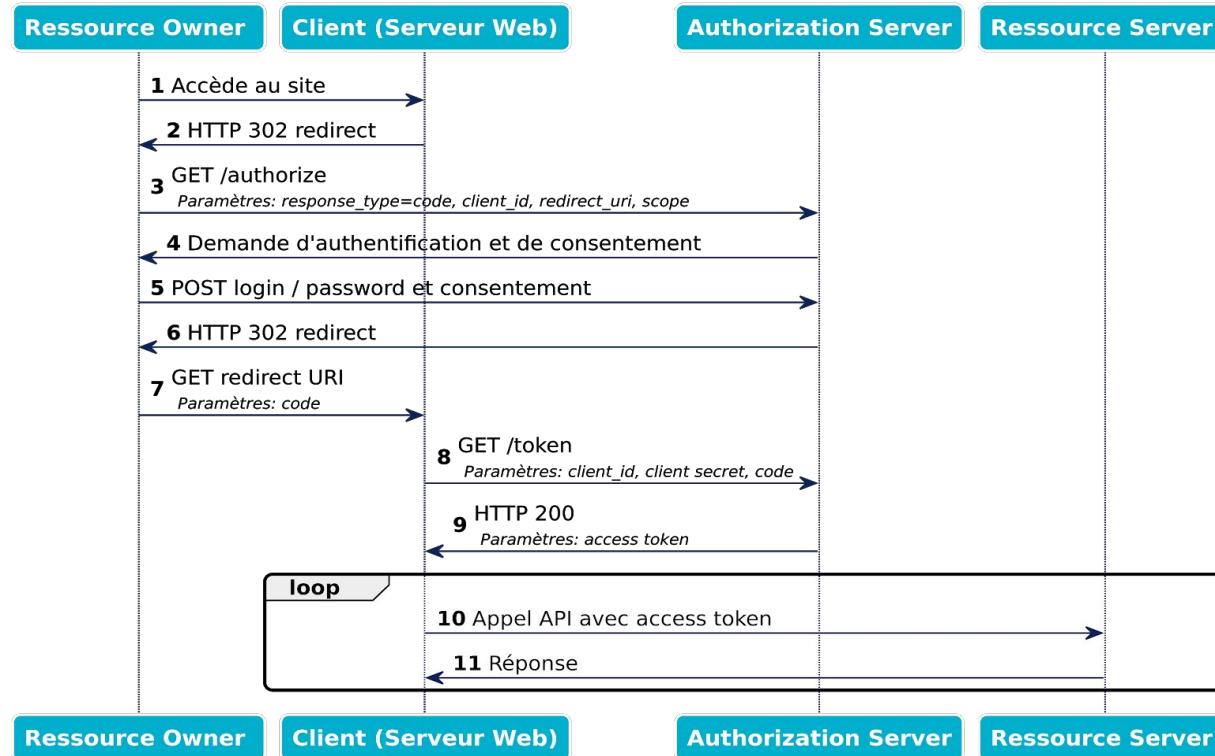
1. Le client demande un access token au serveur d'autorisation
2. Le serveur d'identités demande à l'utilisateur de s'identifier
3. Le client demande à l'utilisateur son autorisation pour accéder à ses ressources
4. L'utilisateur donne son **consentement**, puis Le serveur d'autorisation fournit le token
5. Le client fait une requête auprès du serveur de ressources, pour une ressource privée, en envoyant l'access token pour s'authentifier
6. Le serveur de ressources valide l'access token
7. S'il est valide, le serveur de ressources répond à la requête.



OAuth2

Authorization Code

OAuth2 - Authorization code





Exercice



Sécuriser une API avec Authorization Code Grant

Objectif:

Sécuriser une notre API à l'aide du flow Authorization Code Grant

- Utilisation d'Auth0 comme "Authorization server"
- Utilisation de notre API précédemment déclarée comme "ressource server"
- Déclaration d'une application auprès de Auth0 qui aura le rôle de "client"
- Nous jouerons le rôle du "resource owner"

Résultat attendu:

Obtenir un access token avec scope autorisé

Demander un access token avec scope non autorisé





Exercice



Sécuriser une ressource privée avec Client Credentials

En tant que développeur d'API:

- Créer une Application
 - > type "Regular Web Application"
 - > Renseigner une URL de callback autorisée (ex Postman : <https://oauth.pstmn.io/v1/callback>)

En tant qu'utilisateur final:

- S'authentifier auprès du "Authorization server" à l'aide du Authorization Code Flow ([doc](#)) pour obtenir le code

En tant que client de l'API:

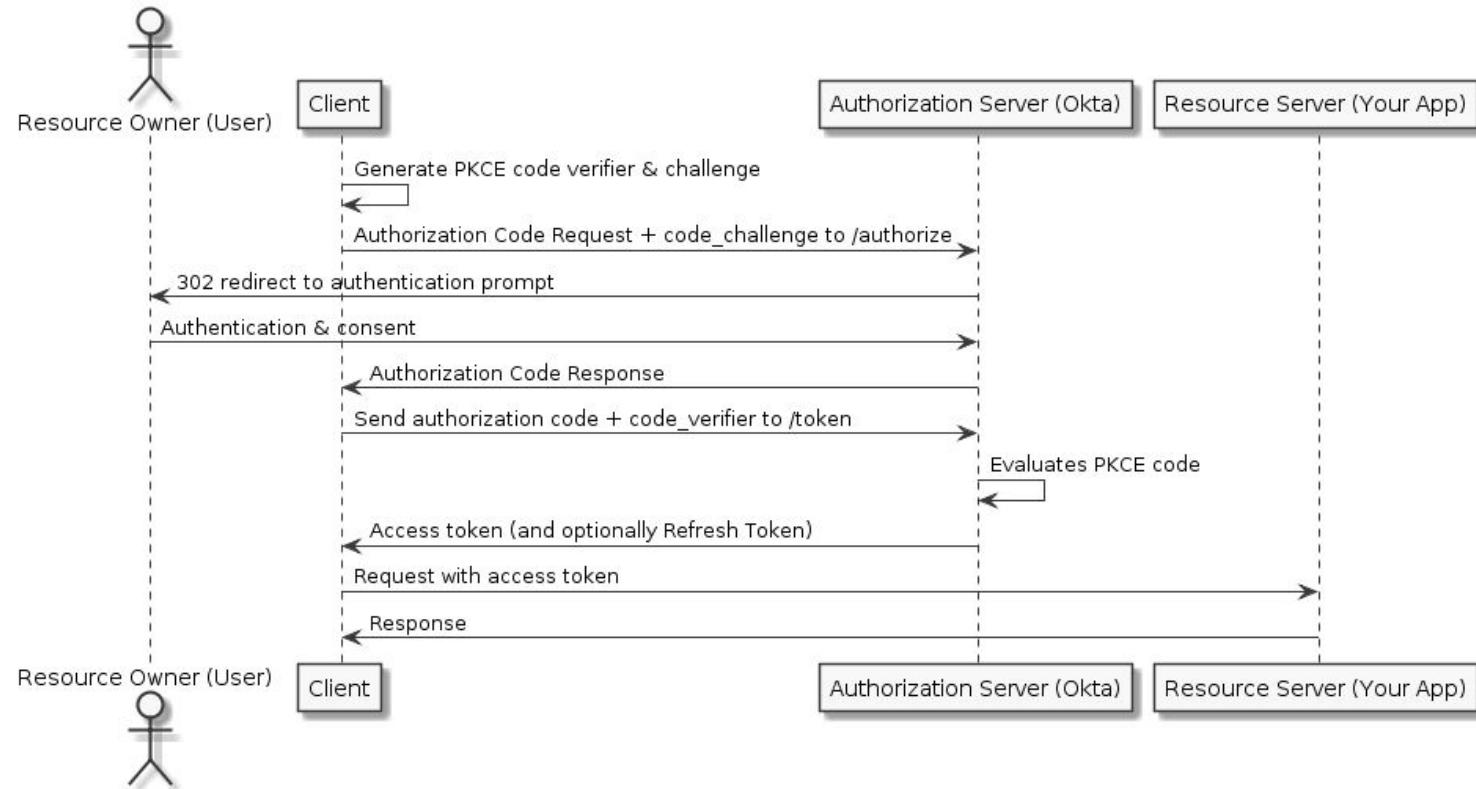
- Récupérer un token via **Authorization Code Flow** (cf doc <https://auth0.com/docs/api/authentication>)
- Consulter le contenu du token sur <https://jwt.io>





OAuth2

Authorization Code Grant + PKCE



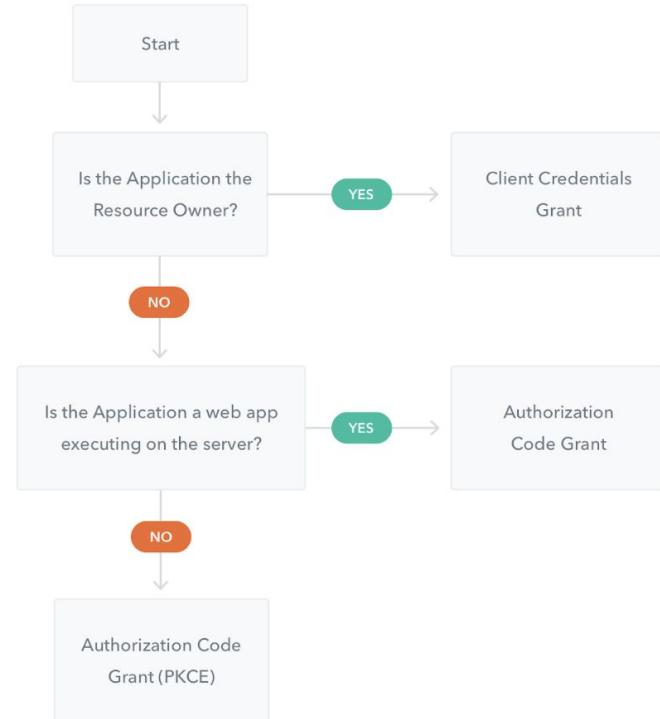


Quel flow utiliser ?

<https://auth0.com/docs/api-auth/which-oauth-flow-to-use>

Quick refresher - OAuth 2.0 terminology

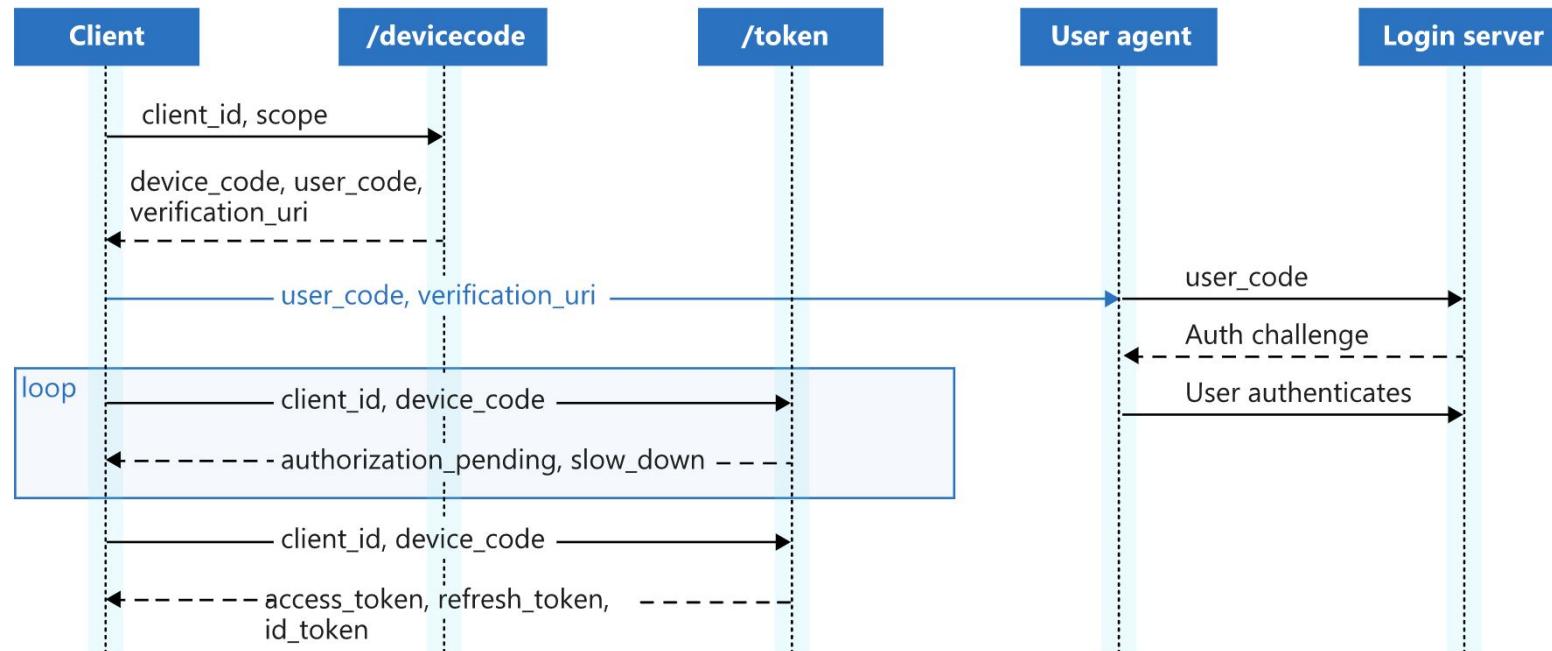
- **Resource Owner:** the entity that can grant access to a protected resource. Typically this is the end-user.
- **Application:** an application requesting access to a protected resource on behalf of the Resource Owner.
- **Resource Server:** the server hosting the protected resources. This is the API you want to access.
- **Authorization Server:** the server that authenticates the Resource Owner, and issues Access Tokens after getting proper authorization. In this case, Auth0.
- **User Agent:** the agent used by the Resource Owner to interact with the Application, for example a browser or a native application.





OAuth2

Device Grant

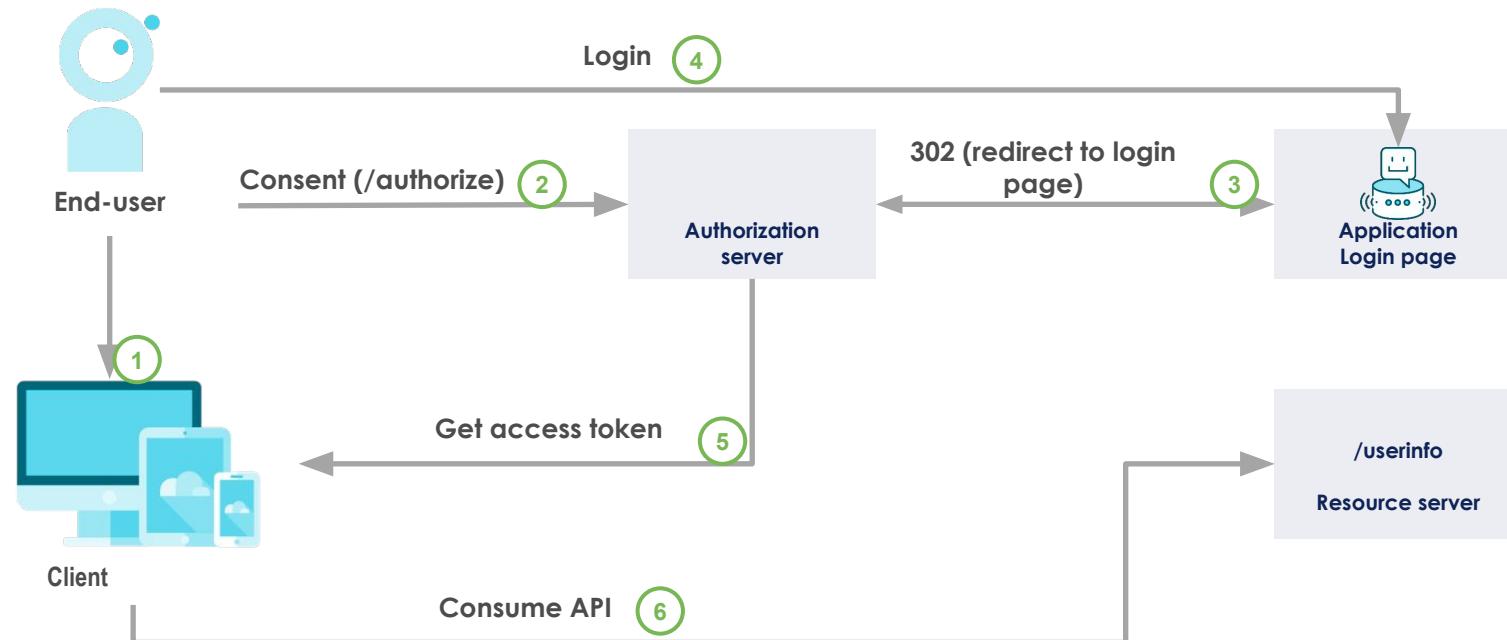




OpenID Connect

Pourquoi ?

=> OAuth2 ne permet pas au client de récupérer l'identité de l'utilisateur





OpenID Connect

Fédération d'identités



- OIDC permet aux développeurs d'applications d'authentifier leurs utilisateurs, sans devoir prendre la responsabilité de stocker et gérer les mots de passe et toute l'authentification.
- C'est le protocole derrière les Single Sign On
- C'est le standard pour gérer l'identité de l'utilisateur

Là où OAuth 2.0 concernait l'accès et le partage des ressources, OIDC s'occupe de **l'authentification des utilisateurs**



Sécurité API

Take away

La sécurité doit être gérée via **OAuth2** ou **OIDC**

SIMPLE

identification de
l'application appelante

- API keys
- OAuth2 Client Credentials

COMPLEXE

identification de
l'application appelante
et de **l'utilisateur connecté**

- OAuth2 / OIDC Authorization code avec PKCE

NE PAS UTILISER

- de protocoles « propriétaires »
- les protocoles : SAML2, WS-Security, OAuth1-a, etc.
- de mécanismes de cryptographie applicative (autre que JWT) : responsabilité de la couche de transport HTTPS
- les flows OAuth 2.0 dépréciés : Implicit, Resource Owner Password Credentials



Pour aller plus loin

L'article de référence pour creuser le sujet =>

<https://blog.octo.com/securiser-une-api-rest-tout-ce-qu'il-faut-savoir/>

Un article sur les changements de OAuth 2.1 =>

<https://fusionauth.io/blog/2020/04/15/whats-new-in-oauth-2-1>



API Management

Présentation

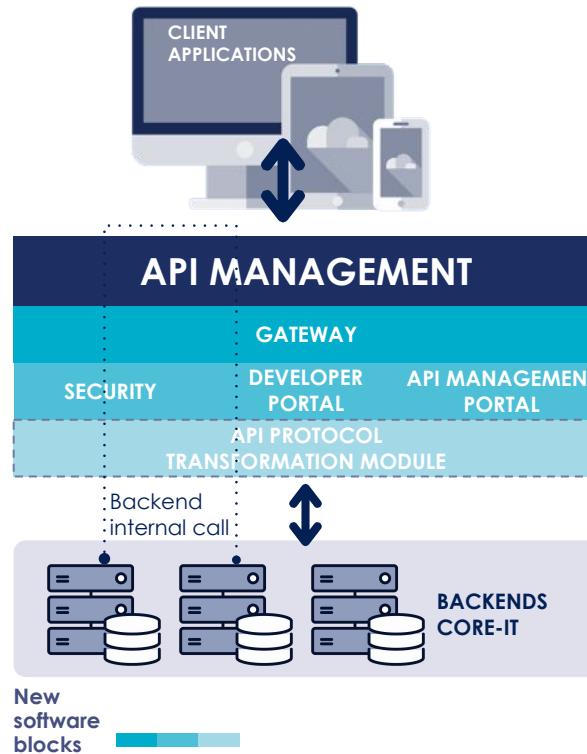
API Management

- L'API management regroupe les processus de
 - > publication
 - > documentation
 - > et supervision d'APIs
 - > dans un environnement sécurisé et scalable
- Le but de l'API management est de permettre à une organisation qui publie des API, de suivre le cycle de vie des API, et de s'assurer que les attentes des développeurs et des applications qui les consomment sont satisfaites.

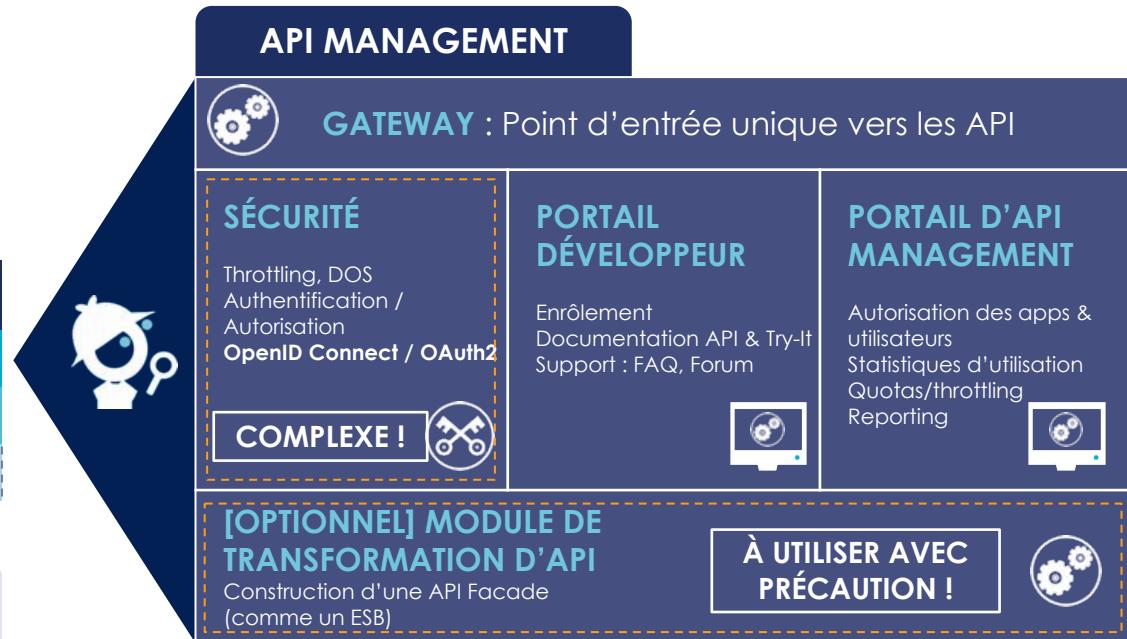


API Management

Blocs fonctionnels



Ces blocs sont souvent gérés par d'autres solutions dédiées



Un produit d'API Management **n'est pas une solution miracle**. Les vrais enjeux sont la **gouvernance des API, la sécurité, et une architecture découpée**.

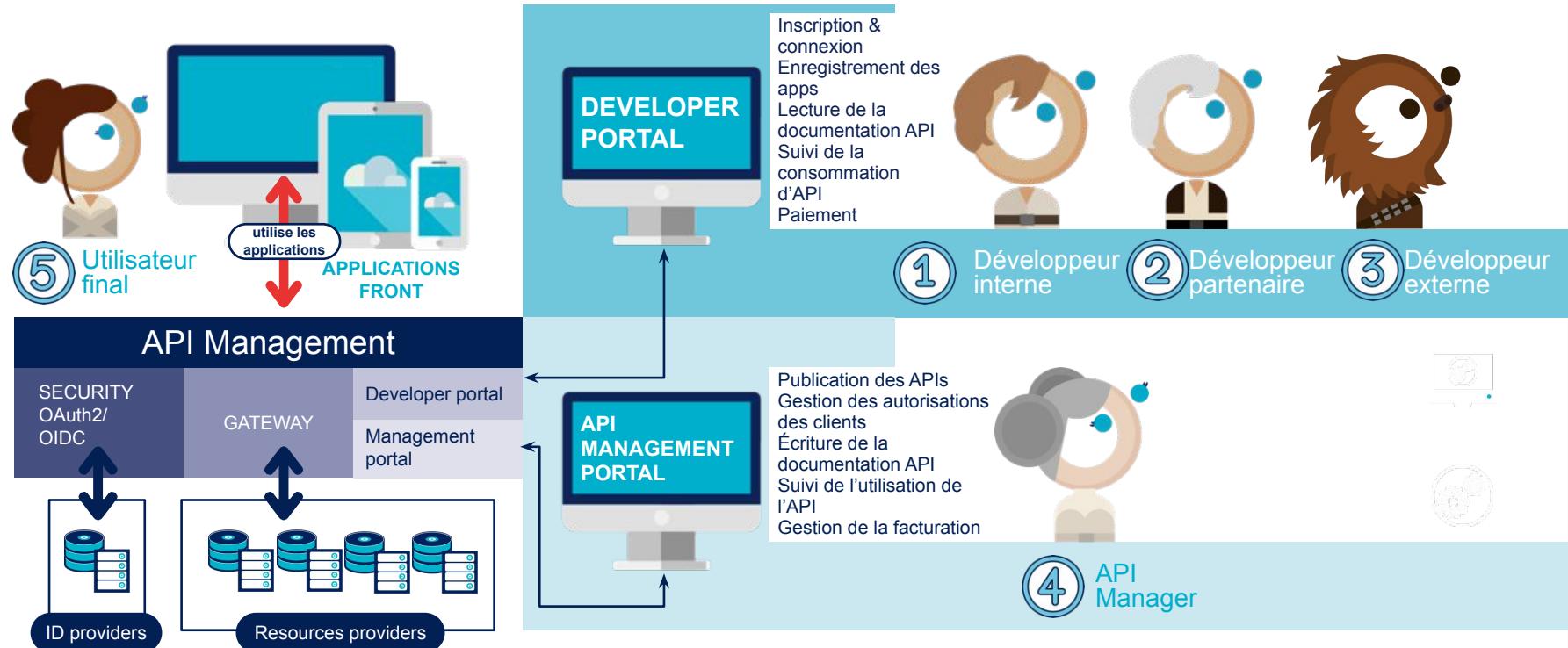


Utilisez une solution d'API Management pour **gérer** vos API, pas pour les construire.



API Management

Utilisateurs de l'APIM

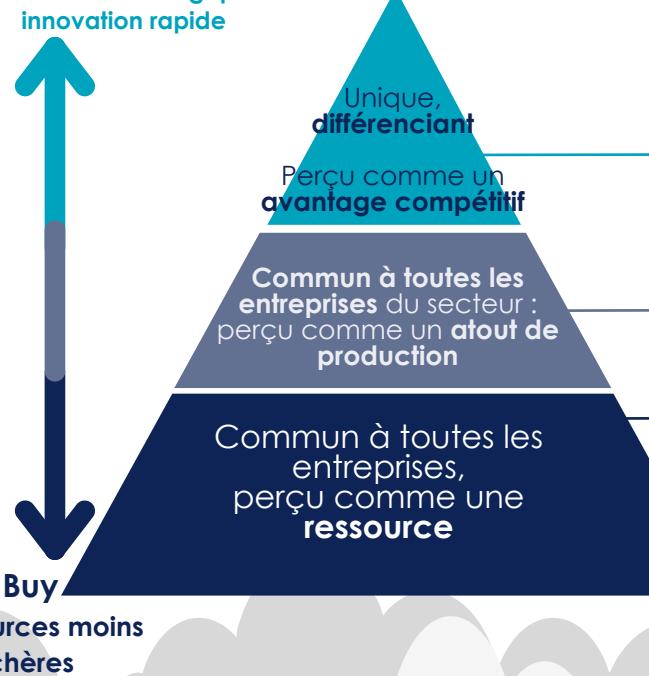




Build vs Buy

Les acteurs du numérique combinent les 2 approches

Build : Atouts stratégiques et innovation rapide



- Apps mobiles
- Sites web
- API
- WM banking CORE system
- PSD2 SCA features
- Authentification (OpenID Connect)
- API Management
- Headless CMS
- Cloud & Managed services
- etc...





07

Questions ?



*There
is
a Better
Way*

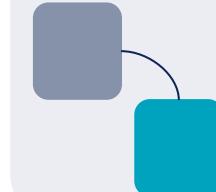


Check-list

Avant de publier la présentation :

- ✓ Vérifier qu'il n'y a qu'un seul thème appliqué
 - > Cliquer en dehors du slide
 - > Cliquez sur "Thème"
 - > "Dans cette présentation"
 - > Sélectionner le premier "2020-02 OCTO 4/3"
- ✓ Vérifier la référence, la date et la version du document
- ✓ Vérifier l'affichage des slides
 - > Clic-droit / Masquer la diapositive
- ✓ Supprimer les slides cachés / inutiles
- ✓ Générer le sommaire
- ✓ Vérifier l'utilisation **exclusive** des couleurs du thème
 - > Test avec les palettes Blanc et Noir
- ✓ Nommer la version actuelle
Fichier / Historique des versions / Nommer la version actuelle
- ✓ Télécharger une version PDF (avec ou sans notes) pour diffusion au client
Gdrive : Fichier / Paramètre d'impression et aperçu / 1 diapositive avec des notes
Powerpoint : Fichier / Enregistrer sous / PDF / Autres options / Options

La documentation est ici



Les schémas doivent être sans bordures, avec coins arrondies