



Les API

Les fondamentaux



1/ Présentation

Qu'est-ce qu'un web services

{🎓} Un Web Services c'est quoi ?

{ API }

Un **web service** est une interface de programmation ouverte qui permet d'effectuer une tâche ou une action standard sur un serveur grâce à des requêtes basé sur une architecture SOA (service oriented architecture).

Ils s'appuient sur des protocoles très utilisés (XML, HTTP, WebSocket, ...) pour communiquer. Cette communication est intégralement réalisée sur le principe d'émission et de réception d'une requête qui donne une réponse sur un schéma d'échange basé en général sous le format **XML** ou **JSON**.

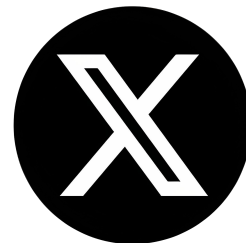
Les web services servent aujourd'hui les systèmes d'informations dans une logique **d'interopérabilité** entre les différentes couches logiciels et briques technologiques pour respecter et rendre cohérent la manipulation des données ou des actions.

Les web services sont aujourd'hui très répandu dans l'industrie du S.I (système d'information) pour faire la jonction entre l'interface utilisateur et l'interface métier.

{🌐} Ils utilisent tous les web services

{ API }

De nombreux acteurs utilisent les Web Services (En somme ils l'utilisent tous c'est devenu un standard dans l'industrie !)



WORDPRESS



Instagram



INTERCOM

YAHOO!

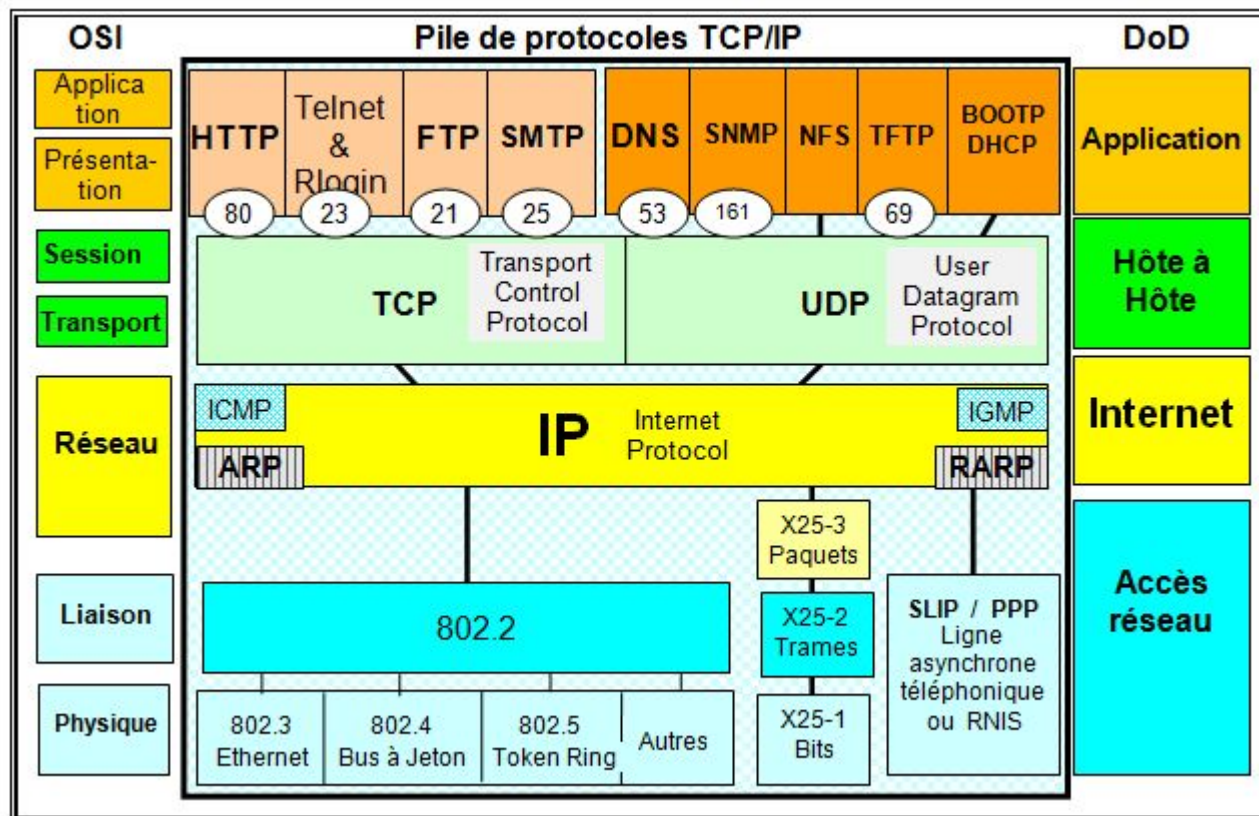
facebook®

Pour permettre à deux ordinateurs d'échanger des informations entre eux, il faut un lien physique entre ces deux ordinateurs (éventuellement sans fil). Si on relie non pas deux, mais plusieurs ordinateurs qui vont pouvoir s'échanger des informations, on construit un réseau informatique.

Internet est un réseau de réseaux, c'est-à-dire un ensemble de technologies qui permettent à plusieurs réseaux de s'interconnecter de manière à permettre l'échange d'informations entre ordinateurs connectés non seulement au même réseau, mais aussi sur des réseaux différents.

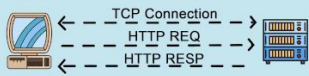

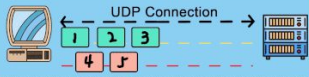

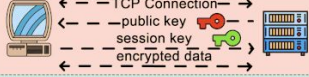

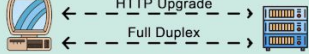

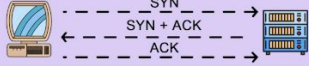

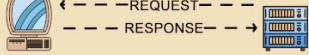

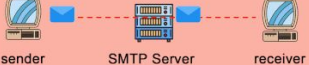

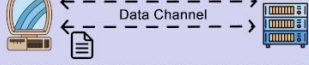

Par exemple, les ordinateurs dans une bibliothèque universitaire sont connectés, certainement par un câble Ethernet, au réseau interne de l'université. Chez un particulier, l'ordinateur familial sera plutôt connecté au réseau de son fournisseur d'accès à Internet, notamment par **ADSL**, **FIBRE**, **4G/5G**.

Le réseau de l'université comme le réseau du fournisseur d'accès à Internet sont connectés à d'autres réseaux, eux-mêmes connectés à d'autres réseaux. Pris ensemble, tous ces réseaux forment Internet. Pour gérer la transmission de données sur ce réseau de réseaux, deux protocoles sont utilisés et constituent le fondement d'Internet : **IP**, pour Internet Protocol, et **TCP**, pour Transfert Control Protocol.



8 Popular Network Protocols

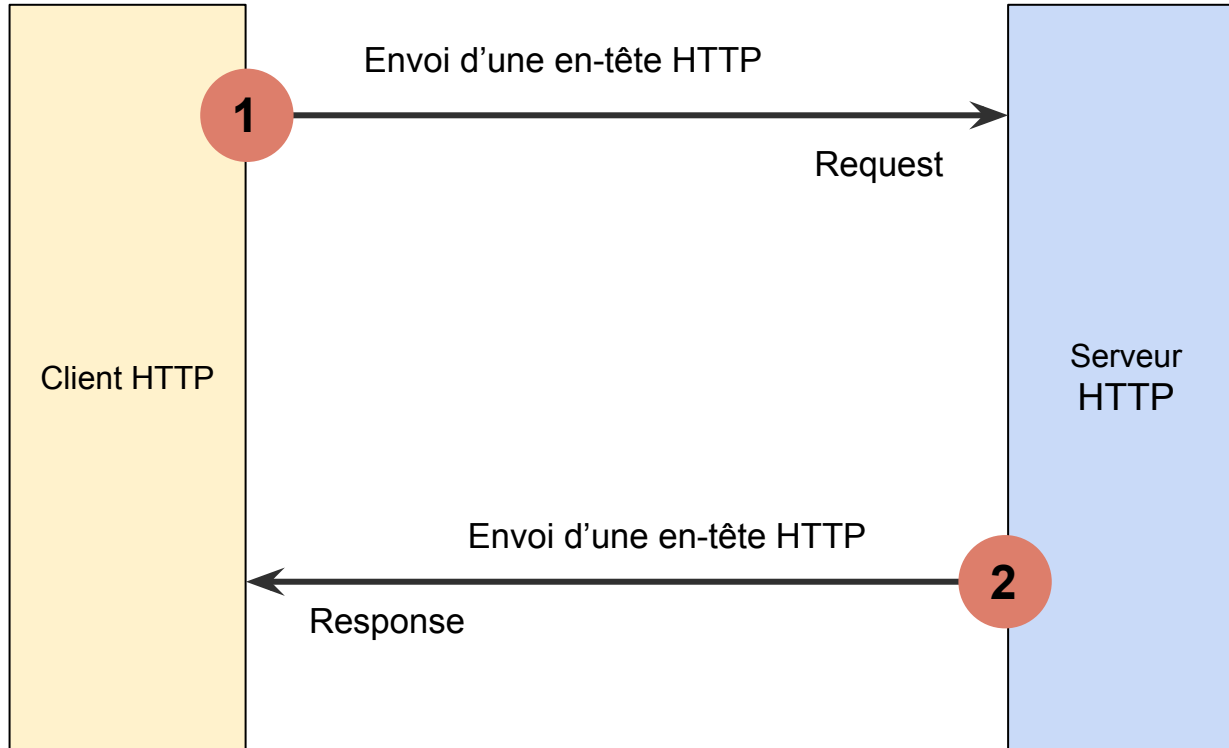
blog.bytebytego.com

Protocol	How does It Work?	Use Cases
HTTP		 Web Browsing
HTTP/3 (QUIC)		 IoT Virtual Reality
HTTPS		 Web Browsing
WebSocket		 Live Chat Real-Time Data Transmission
TCP		 Web Browsing Email Protocols
UDP		 Video Conferencing
SMTP		 Sending/Receiving Emails
FTP		 Upload/Download Files

L'**HyperText Transfer Protocol**, plus connu sous l'abréviation **HTTP** littéralement (protocole de transfert hypertexte) est un protocole de communication client-serveur développé pour le World Wide Web.

HTTPS (avec S pour secured, soit « sécurisé ») est la variante du HTTP sécurisée par l'usage des protocoles **SSL** ou **TLS**. Dans le protocole HTTP, une méthode est une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action.

SSL (**Secure Sockets Layer**) est un protocole de sécurisation des échanges sur Internet, devenu **TLS** (**Transport Layer Security**) en 2001.



Dans le monde de l'informatique les codes HTTP permettent de définir le résultat d'une requête ou une erreur du navigateur. Les codes sont principalement adressés pour permettre le traitement automatisés par les clients HTTP. Effectivement le client qui utilise les URI et le protocole HTTP suis la la norme **RFC7231** qui régit les codes HTTP pour tout les navigateur utilisant le protocole.

Les principales catégories de code permettent de déterminer : succès, l'informations, redirection, erreur client et erreur serveur.

Les codes les plus utilisés :

- **200** : succès de la requête
- **301** et **302** : redirection, respectivement permanente et temporaire
- **401** : utilisateur non authentifié
- **403** : accès refusé
- **404** : page non trouvée
- **500** : Erreur interne du serveur
- **504** : Gateway Time-out

Liste des codes disponible avec **RFC7231**. [ici](#)

HTTPS (avec S pour secured, soit « sécurisé ») est la variante du HTTP sécurisée par l'usage des protocoles **SSL** ou **TLS**. Dans le protocole HTTP, une méthode est une commande spécifiant un type de requête, c'est-à-dire qu'elle demande au serveur d'effectuer une action.

- **SSL** : (**Secure Sockets Layer**), un protocole de sécurisation des échanges sur Internet, devenu TLS (**Transport Layer Security**) en 2001.

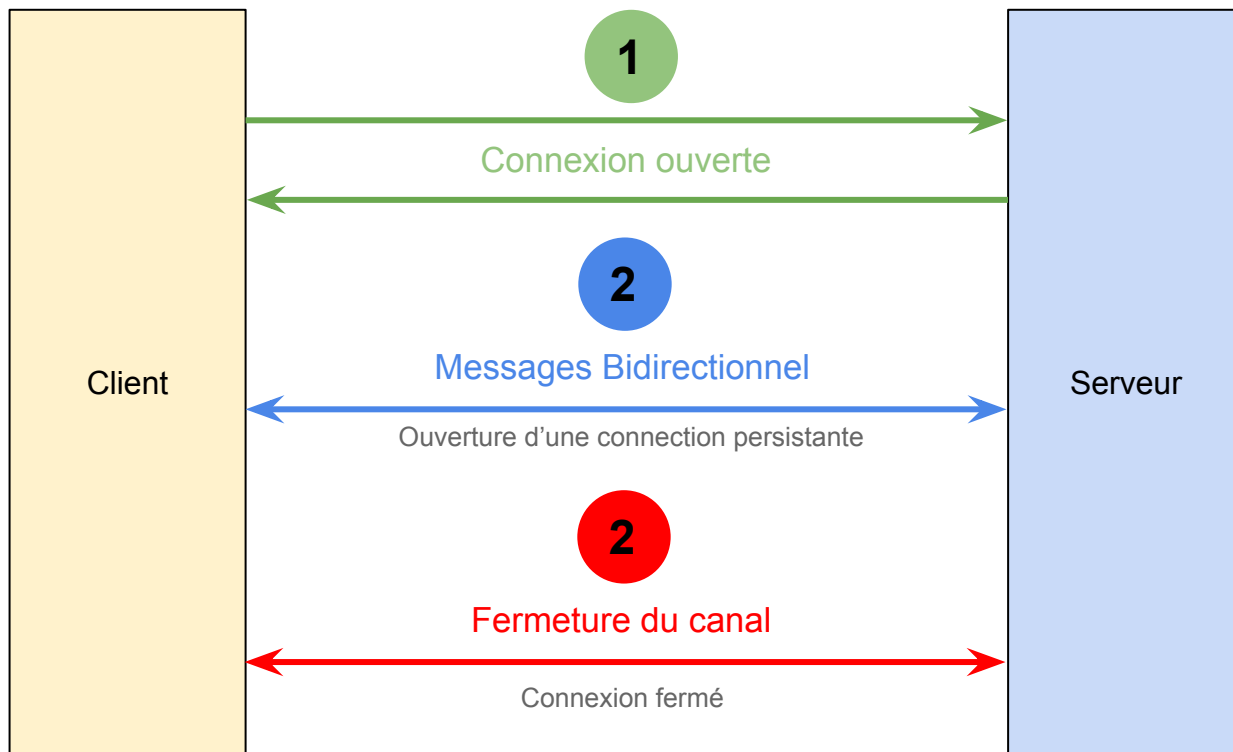
La technologie **HTTPS** repose uniquement sur une technique de chiffrement des données grâce à un certificat fournis par une organisation étant reconnu comme étant “certifié”.

Le protocole **WebSocket** permet une communication bidirectionnelle entre un agent utilisateur exécutant du code non approuvé s'exécutant dans un environnement contrôlé et un hôte distant ayant activé les communications à partir de ce code.

Le modèle de sécurité utilisé à cet effet est le modèle de sécurité basé sur l'origine couramment utilisé par les navigateurs Web.

L'objectif de cette technologie est de fournir un mécanisme pour les applications basées sur un navigateur qui nécessitent une communication bidirectionnelle avec des serveurs qui ne reposent pas sur l'ouverture de plusieurs connexions **HTTP** (par exemple, en utilisant **XMLHttpRequest**).

<https://tools.ietf.org/html/draft-abarth-websocket-handshake-01>



Le certificat **SSL** permet de sécuriser les transferts de données entre le client et son serveur. Effectivement il permet de lier une clé cryptographique à l'ensemble des informations pour une organisation ou un individu.

Le navigateur se connecte alors au port 443 en HTTPS pour assurer une connexion sécurisée entre le serveur web et lui même. L'utilisation du SSL c'est généralisé durant les 5 dernières années grâce notamment aux moteurs de recherche comme Google qui applique une lois d'un ranking sur les sites disposant d'un protocole HTTPS. Le SSL est donc devenu la norme pour sécurisé les données sensibles pouvant être transmissent comme par exemple les paiements bancaires.

Pour obtenir un certificat SSL il faut au préalable disposer :

- Un nom de domaine et d'une adresse d'hôte.
- Un certificat SSL délivré par une organisation garantissant et étant reconnu par tous comme sur !

Il est bien entendu possible de générer votre propre certificat SSL cependant vous ne serez pas reconnu par les navigateurs comme étant une organisation 100% sécurisé. Car effectivement si vous générez vous même votre propre certificat vous détenez également le clé qui permet de décrypter les informations depuis le client !

{🎓} Reconnaître une url qui utilise un certificat SSL certifié !

{ API }

Certificat généré par un tier certifié :

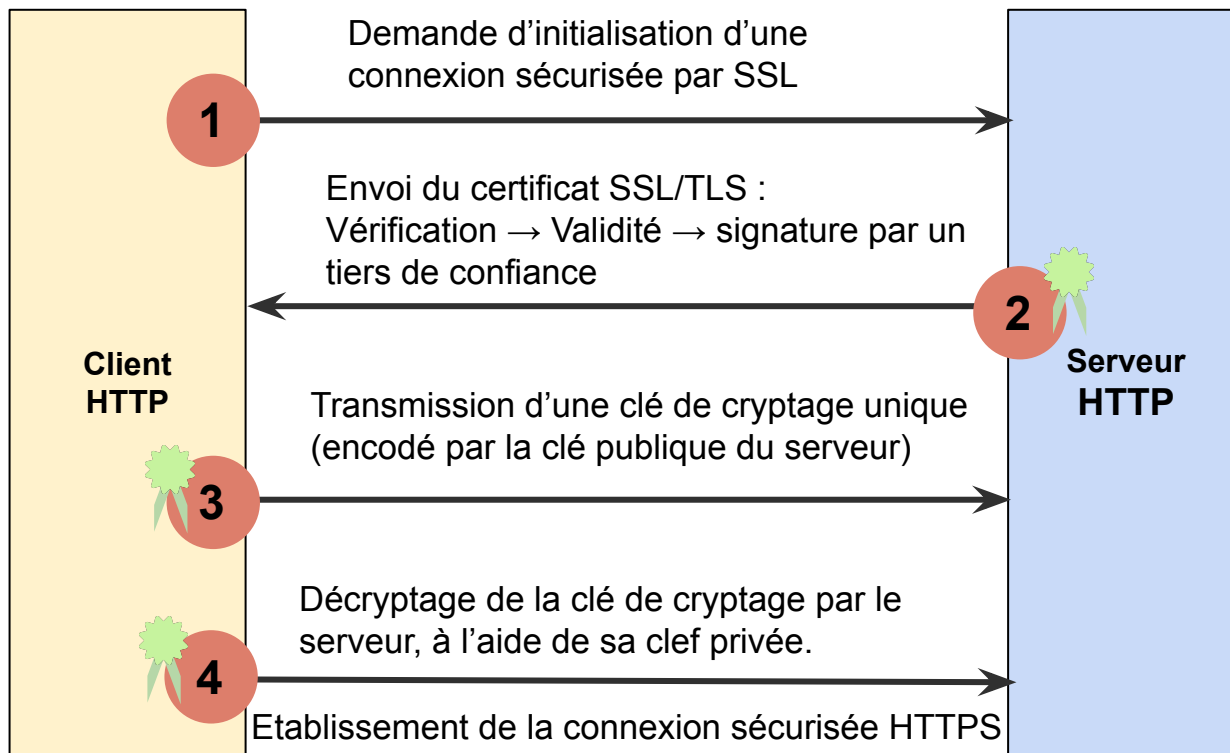
 Sécurisé | <https://www.socialobjects.ai>



Certificat généré par un tier non reconnu : ATTENTION !

 Non sécurisé | <https://socialobjects>







2/ Architecture SOAP

L'âge du XML

SOAP ancien acronyme de (Simple Oriented Application Protocol) est un protocole de **RPC**(Remote Procedure Call) orienté objet bâti sur **XML**.

Le XML (**Extensible Markup Language**) : est un métalangage informatique avec des balises dérivé du SGML. La syntaxe du XML est dite “extensible” pour permettre de définir différent “name space” qui permet de personnaliser le nom des balises.

Tout d'abord, les deux parties (le client et le serveur) se partagent une notice d'utilisation : la **WSDL** (**Web Service Description Language**), qui est un document XML décrivant toutes les méthodes qui peuvent être appelées par le client sur le serveur, avec les structures de données disponibles pour ces échanges. Le client prépare donc sa demande et l'enveloppe dans une grosse structure XML contenant toutes les métadonnées (méthode appelée, signature numérique, etc.).

Une fois que le serveur a reçu cette demande, il la traite et répond aussi dans une grosse enveloppe XML le résultat en général depuis une base de données.

L'XML (Extensible Markup Language), est un méta langage informatique de balisage qui dérive du SGML.

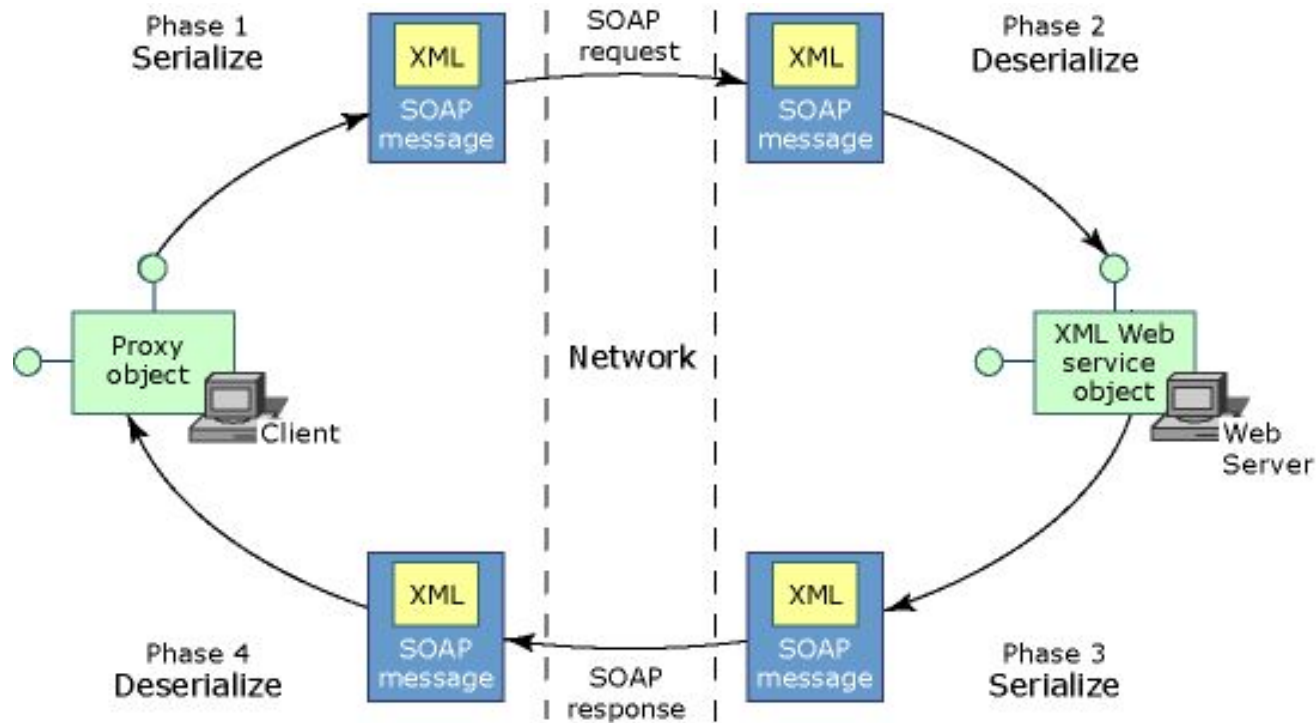
Cette syntaxe est dite “**extensible**” car elle permet de définir différents espaces de noms (name spaces). Le XML utilise les chevrons (<, >) qui permet d'encadrer les balises. Une balise peut contenir des attributs qui permettront de l'enrichir.

Un attribut peut être décrit comme une option. (<balise myattribut="my value"></balise>).

Le **XML** décrit, structure, échange des données tandis que le HTML ne fait qu'afficher des données. Il ont cependant un point commun, ils sont tous deux issus du **SGML**.

Standard Generalized Markup Language (« langage de balisage généralisé normalisé » - **SGML**) est un langage de description à balises, de norme ISO (ISO 8879:1986).

```
<Election>
  <Scrutin>
    <Type> Présidentielle </Type>
    <Annee> 2012 </Annee>
    <Date> 26-04-2012 </Date>
    <Heure> 11:23:43 </Heure>
    <ListeRegion>
      <Region>
        <CodReg> 42 </CodReg>
        <CodReg3Car> 042 </CodReg3Car>
        <LibReg> ALSACE </LibReg>
      </Region>
      <Region>
        <CodReg> 72 </CodReg>
        <CodReg3Car> 072 </CodReg3Car>
        <LibReg> AQUITAINE </LibReg>
      </Region>
    </ListeRegion>
  </Scrutin>
</Election>
```



```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnInteger xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">123</param1>
    </ns1:doubleAnInteger>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnIntegerResponse
      xmlns:ns1="urn:MySoapServices"
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:int">456</return>
    </ns1:doubleAnIntegerResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



2/ Architecture REST

Les microservices

REST (**r**epresentational **s**tate **t**ransfer) est un style d'architecture pour les systèmes hypermédia distribués. Le but du jeu est donc d'utiliser au maximum les possibilités du protocole HTTP, les verbes, les URL et les codes retours pour décrire des API de la manière la plus fidèle possible.

Pour ce faire on utilise le web lui même à savoir les interactions entre le client et un serveur qui sont les fondamentaux du HTTP à savoir le **CRUDS**. Le CRUDS est une norme que l'on applique en général à une base de données pour permettre d'en manipuler son contenu. Le CRUDS doit donc s'appliquer à chaque **ROUTES** de l'API pour les opération suivants : **C**reate, **R**ead, **U**ppdate, **D**eleate et **S**earch.

Opérations	SQL	HTTP
Create	INSERT	POST
Search	SELECT *	GET
Read	SELECT	GET
Update	UPDATE	PUT / PATCH
Delete	DELETE	DELETE

Restez simple ! Votre API doit être facilement compréhensible et accessible pour n'importe quel développeur. Donner des noms explicites à vos routes n'essayez pas de faire de longue phrase dans vos endpoints.

- N'utilisez pas de verbe mais des nom.
- Utilisez toujours le pluriels pour vos noms.
- Utilisez correctement le HTTP Header pour la serialization

Pensez également à mettre une version sur votre api. Il n'est pas rare de voir souvent ce qui suit :

<http://api.monservice.com/apiv1/users/:id>

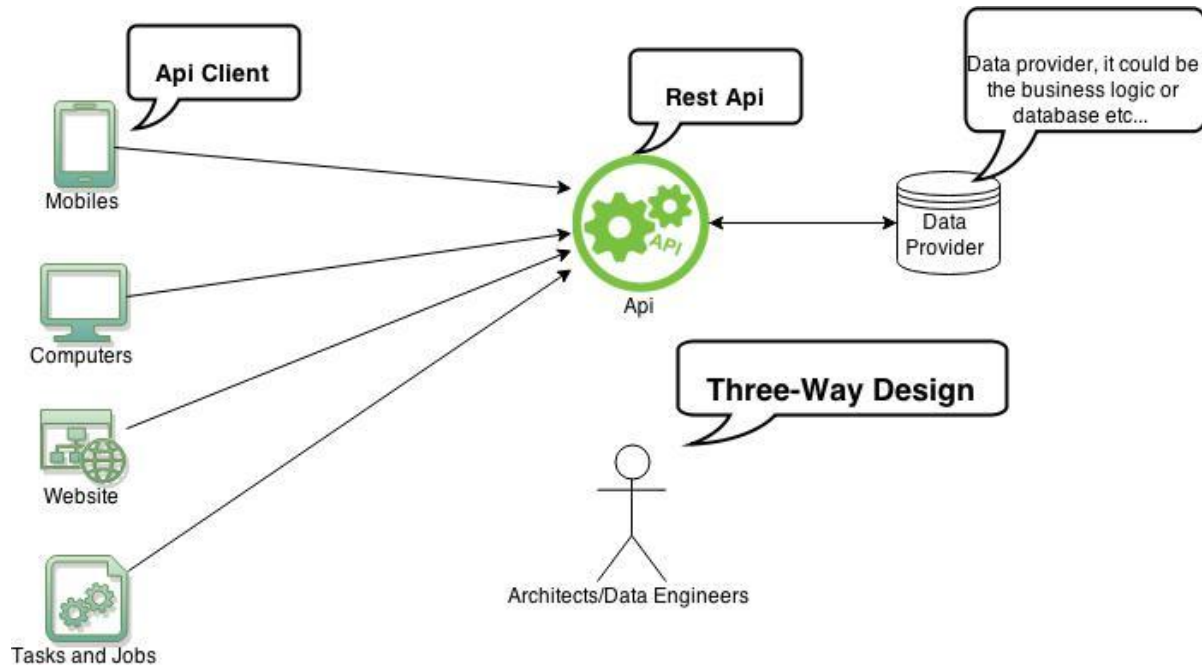
Documentez toujours votre API vous devez être clair et explicite en détaillant toutes les caractéristiques pour faire en sorte que personne ne vienne vous voir pour vous demander comment elle fonctionne. Utilisez la philosophie du Feedback et demandez à vos amis ou à des inconnus d'utiliser votre service pour améliorer votre documentation, vos endpoints, vos payload et vos messages d'erreurs.

En somme restez simple et garder à l'esprit la philosophie du KISS Keep. It Simple. Stupid !



{ 🎓 } Les différents cas d'utilisation d'une API

{ API }



Les **Endpoints** désignent les **URI** que l'utilisateur devra emprunter pour interagir avec votre interface selon les verbes **HTTP** et les règles **CRUDS** que vous aurez définie. On parle alors de microservices, les microservices permettent de concevoir les applications avec des composants élémentaires pour segmenter la gestion des données, de répartir la charge et de mieux maintenir son environnement applicatif.

Les Endpoints sont automatiquement liés aux verbes **HTTP** et il ne saurait être question de déclarer une méthode **POST** pour récupérer la fiche d'un client alors que le **GET** permet d'accélérer le temps de réponse de la requête. Effectivement si vous envoyez une requête **POST** à un serveur la requête **POST** enverra une requête **OPTIONS** puis une requête **POST** CQFD.

Exemples :

GET `http://api.monapp.com/user/:id`
POST `http://api.monapp.com/user/`
POST `http://api.monapp.com/users/`
DELETE `http://api.monapp.com/user/:id`
DELETE `http://api.monapp.com/users/`
PUT `http://api.monapp.com/user/:id`

Vous devez utiliser le spinal-case avec les espaces dans les mots de vos endpoints d'api. Exemple :



`http://api.monapp.com/user-analytics/:id`

Les en-têtes HTTP permettent au client et au serveur de transmettre des informations supplémentaires avec la requête ou la réponse. Un en-tête de requête est constitué de son nom (insensible à la casse) suivi d'un deux-points ':', puis de sa valeur (sans saut de ligne). L'espace blanc avant la valeur est ignoré.

Les en-têtes peuvent être groupés selon leur contexte :

- En-tête général : en-têtes s'appliquant à la fois aux requêtes et aux réponses mais sans rapport avec les données éventuellement transmises dans le corps de la requête ou de la réponse.
- En-tête de requête : en-têtes contenant plus d'informations au sujet de la ressource à aller chercher ou à propos du client lui-même.
- En-tête de réponse : en-têtes contenant des informations additionnelles au sujet de la réponse comme son emplacement, ou au sujet du serveur lui-même (nom et version, etc.)
- En-tête d'entité : en-têtes contenant plus d'informations au sujet du corps de l'entité comme la longueur de son contenu ou son type MIME.

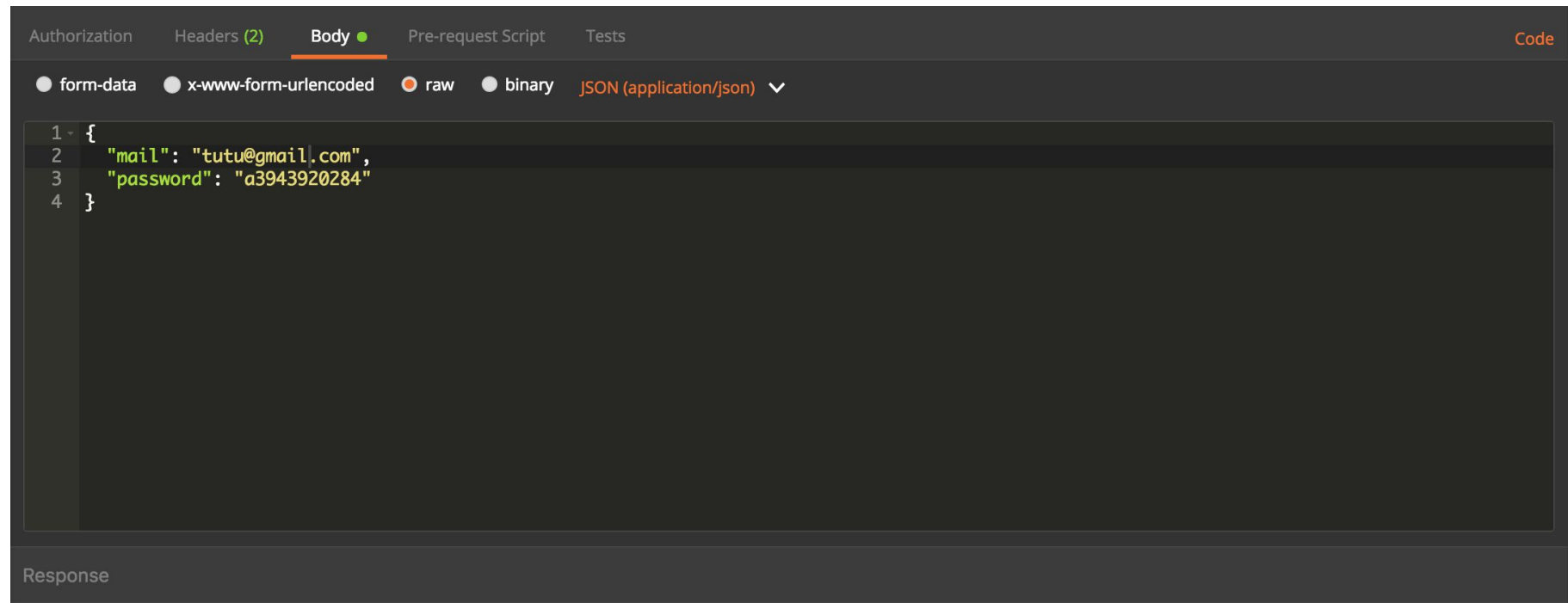
En-têtes de bout en bout ('End-to-end headers') : Ces en-têtes doivent être transmis au destinataire final du message ; c'est-à-dire le serveur dans le cas d'une requête ou le client dans le cas d'une réponse. Les serveurs mandataires intermédiaires doivent retransmettre les en-têtes de bout en bout sans modification et doivent les mettre en cache.

En-têtes de point à point ('Hop-by-hop headers') : Ces en-têtes n'ont de sens que pour une unique connexion de la couche transport et ne doivent pas être retransmis par des serveurs mandataires ou mis en cache. Il s'agit d'en-têtes tels que: Connection, Keep-Alive, Proxy-Authenticate, Proxy-Authorization, TE, Trailer, Transfer-Encoding et Upgrade. A noter que seuls les en-têtes de point à point peuvent être utilisés avec l'en-tête général Connection.

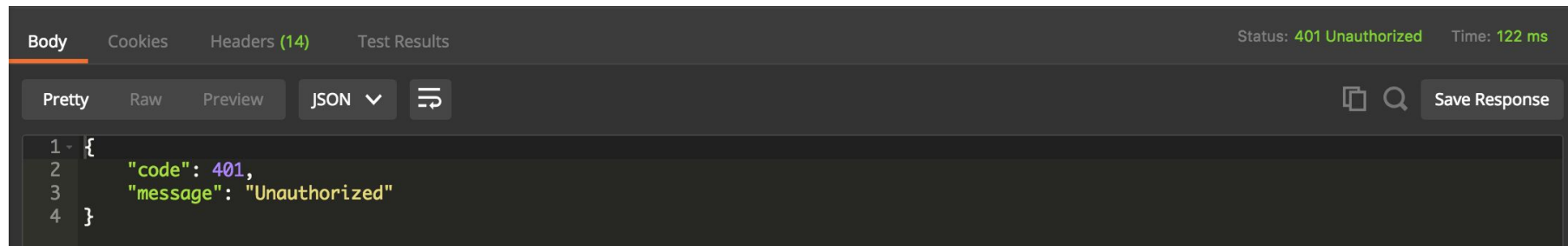
Pour en savoir plus sur toutes les en-têtes disponible voir ici :

<https://developer.mozilla.org/fr/docs/Web/HTTP/Headers>

Le **Payload** désigne les paramètres passés à une API pour obtenir un résultat en contre parti. En général le payload est écrit au format JSON dans un requête POST ou PUT par exemple. Il permet aux développeurs de soumettre n'importe quel paramètre. Il est également possible de passer des paramètres d'url. Exemple :



Le **Payload** doit être contrôlé, il est indispensable d'instaurer des règles sur le format des données soumis à l'API. Si un payload ne correspond pas il faudra alors retourner une erreur avec un code HTTP. Exemple :



The screenshot shows a web browser's developer tools interface. The 'Body' tab is selected, displaying a JSON response. The status bar at the top right indicates 'Status: 401 Unauthorized' and 'Time: 122 ms'. The JSON body is as follows:

```
1 {  
2   "code": 401,  
3   "message": "Unauthorized"  
4 }
```

Sur cet exemple on a soumis un mail et un password à une API d'authentification. Vu que l'utilisateur n'a pas entré les bonnes données l'API lui renvoi une erreur **401 : Unauthorized**. Dans le cas contraire l'api lui aurait retourné les informations qui correspondent à son compte.

Si par exemple un utilisateur dépasse la limite autorisé, il est recommandé de bloquer les appels à la base de données pour éviter les dérives. Il est également recommandé de limiter le nombre de requête par seconde et de bannir les utilisateurs avec leurs adresses IP pour se prémunir des attaques de type DDOS.

Les **BluePrints** permet de documenter votre API pour permettre aux développeurs de manipuler facilement les données provenant de vos micro-services.

Vous pouvez utiliser **MARKDOWN** ou des éditeurs dédiés. **APIARY** vous permet par exemple de documenter votre API mais également de tester les requêtes en live directement depuis votre documentation. Il existe également des générateurs de documentation automatique d'après les commentaires de votre code, ce qui peut être très pratique et vous éviter de gérer la documentation. Cependant vous devrez quand même vous occuper des commentaires de votre API pour mettre à jour la version de votre documentation.

Vous devez toujours mettre à jour votre documentation à chaque fois que vous modifié une api ! C'est un réflexe qui doit devenir automatique.



OAuth est un protocole en version 2 appelé également framework OAuth. Ce protocole permet à de nombreuses applications d'accéder pendant une durée limitée à un micro-service via HTTP ou HTTPS avec une autorisation préalable du détenteur des ressources d'accès.

Chaque accès demande d'avoir une clé d'authentification pour récupérer les ressources du service ou bien de produire ses propres identifiants via une route d'autorisation spécifique. OAuth n'est en aucun cas un protocole d'authentification mais de délégation d'autorisation. Ça veut dire que l'on vous autorise avec certaines limitations et de délai l'accès à une ressource.

La première version de OAuth a été finalisée le 3 octobre 2007.

OAuth est défini par 4 rôles bien distincts :

- Le détenteur des données (vous même)
- Le serveur de ressources (le serveur)
- Le client (navigateur)
- Le serveur d'autorisation (génération de token)

Le **Token** d'accès permet au serveur d'autoriser à consulter ses ressources il est en général généré par une api qui délivre une clé d'activation, c'est cette clé qui est envoyé dans le header d'une requête pour que le serveur autorise une connexion.

Le token à une durée de vie limité dans le temps qui est défini par le serveur d'autorisation, il est donc capital que chaque utilisateurs garde secret cette clé. En général dans les services de type SAS les clés de tokens sont lié au contrat passé avec une application tiers dont l'utilisateur à payé le service.

En général la durée du contrat est basé sur la date d'expiration d'une clé token, elle sera renouvelé automatiquement après le renouvellement par client de son abonnement.



DONE