

# Développement Web et API

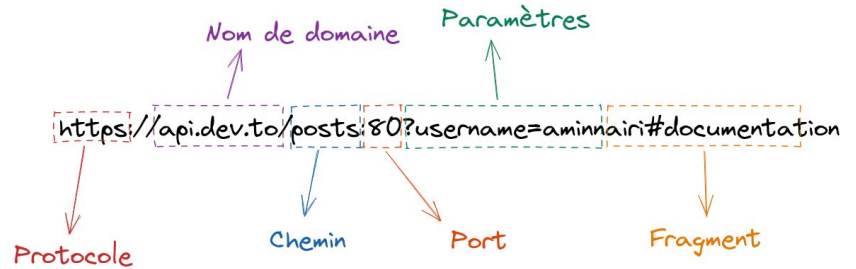
Amin NAIRI <[anairi@esgi.fr](mailto:anairi@esgi.fr)>

# 1 Comment fonctionne une URL

Une URL est composée de 3 parties : un protocole (HTTP, FTP, ...), un nom de domaine (esgi.fr, gitlab.com, ...), un chemin (/users/list, /project/new, ...).

Il est possible également d'avoir optionnellement des paramètres permettant d'envoyer des informations supplémentaires, un port afin de pouvoir atteindre un serveur exposé sur un port différent et un fragment qui est utilisé pour référencer des identifiants de sections dans un code HTML et dont nous ne nous servons pas.

`https://api.dev.to/posts:80?username=aminnairi#documentation`



# 1.1 Protocole

Le protocole est un moyen de savoir de quel manière communiquer avec un serveur. Chaque protocole à sa propre manière de transférer des données, ils ont néanmoins tous un objectif : transmettre une information.

Le protocole le plus rencontré sur internet est le protocole HTTPS qui permet de récupérer le contenu d'une page HTTP ou de transmettre des données d'un formulaire de manière sécurisée.

HyperText Transfer Protocol


http://google.fr

ws://debugger.com

WebSocket

https://gitlab.com

HTTP + TLS/SSL

ftp://csgo.com

File Transfer Protocol

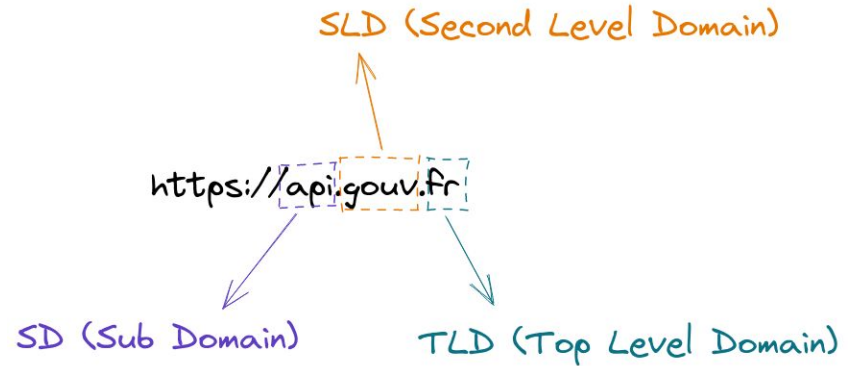
## 1.2 Nom de domaine

Un nom de domaine permet d'identifier un serveur sur internet de manière plus facile pour un être humain.

Les serveurs sont identifiés par une adresse IP, mais se souvenir d'une adresse IP de la forme 20.74.67.225 pour tous ses sites internet serait bien complexe.

Lors d'une requête HTTP, cette dernière va passer le nom de domaine à un serveur DNS qui s'occupe de réunir un listing des nom de domaine face à leur adresse IP. Lorsque l'IP est trouvée, elle est renvoyée pour nous permettre de faire la requête au bon serveur.

Le nom de domaine se compose d'un niveau supérieur (**TLD**, **Top Level Domain**), d'un niveau secondaire (**SLD**, **Second Level Domain**) et optionnellement d'un sous-domaine (**SD**, **Sub Domain**).

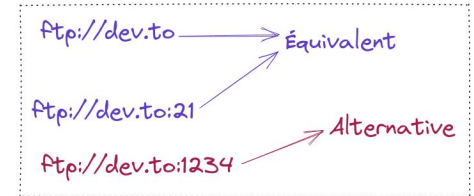
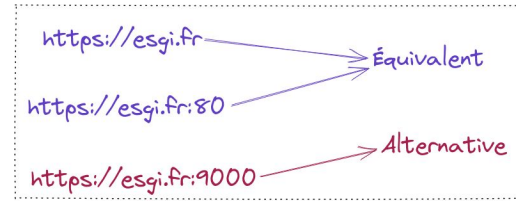


## 1.3 Port

Le port est un identifiant permettant d'accéder au service exposé par un serveur. Cela permet à un serveur d'avoir plusieurs services exposés sur des ports différents.

Il existe plusieurs ports par défaut correspondant aux différents protocoles et services utilisés sur internet comme le 80 (HTTP), le 443 (HTTPS) ou encore le 21 (FTP) et 22 (SSH).

Par défaut, si aucun port n'est renseigné dans une URL, c'est le port 80 par défaut du protocole qui est utilisé.

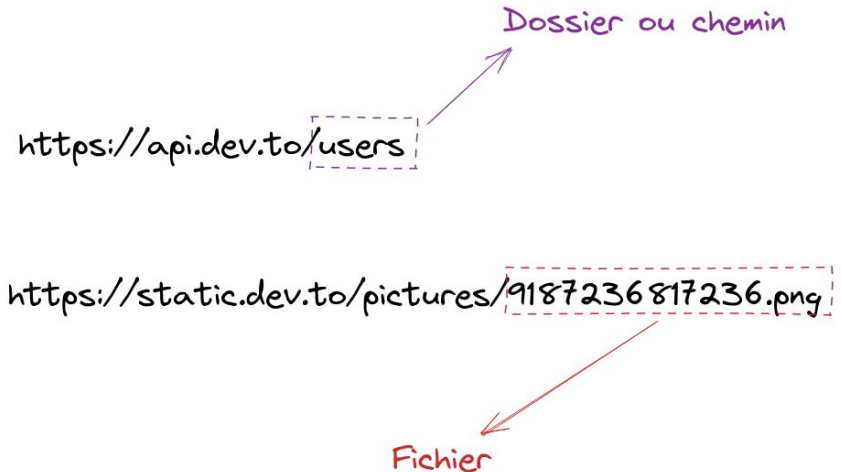


## 1.4 Chemin

Le chemin est un identifiant permettant d'accéder à une ressource particulière sur le serveur. C'est l'équivalent d'un chemin dans un dossier de votre système de fichier.

Ce chemin est interprété par le serveur Web pour répondre à la requête avec la réponse adéquate.

Le chemin peut contenir un nom de dossier mais également un nom de fichier directement.



## 1.5 Paramètres

Les paramètres permet de pouvoir préciser la nature de la requête faite à un serveur sur un chemin en particulier.

Par exemple, vous pourriez avoir envie de récupérer toute la liste des utilisateurs qui sont stockés en base de données.

Mais vous pourriez aussi avoir besoin de récupérer toute la liste des utilisateurs ayant été créé il y a deux semaines.

Ce paramètre nous évitera d'avoir à créer deux chemins : un pour lister tous les utilisateurs, et un autre pour lister tous les utilisateurs créés il y a deux semaines. En un seul chemin et avec un paramètres, nous pouvons arriver à répondre à ces deux requêtes.

Mauvaise pratique

```
https://api.dev.to/users
```

```
https://api.dev.to/users-two-weeks
```

Bonne pratique

```
https://api.dev.to/users
```

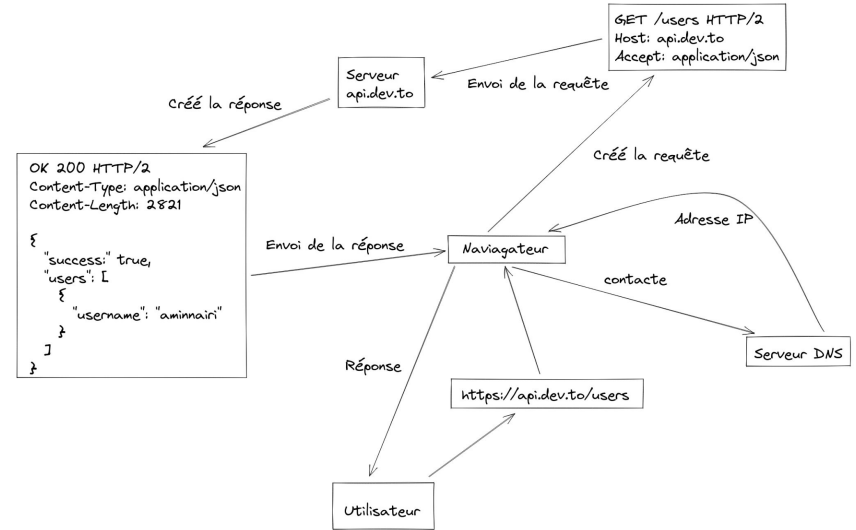
```
https://api.dev.to/users?from=two-weeks
```

## 2 Comment fonctionne le protocole HTTP

Il existe beaucoup de protocole de communication sur internet et le protocole sur lequel nous nous concentrerons sera le protocole HTTP.

C'est un protocole basé sur le protocole TCP et qui permet des échanges de données entre un client et un serveur.

C'est le protocole utilisé par les navigateurs internet afin de récupérer le contenu des sites que nous visitons tous les jours.



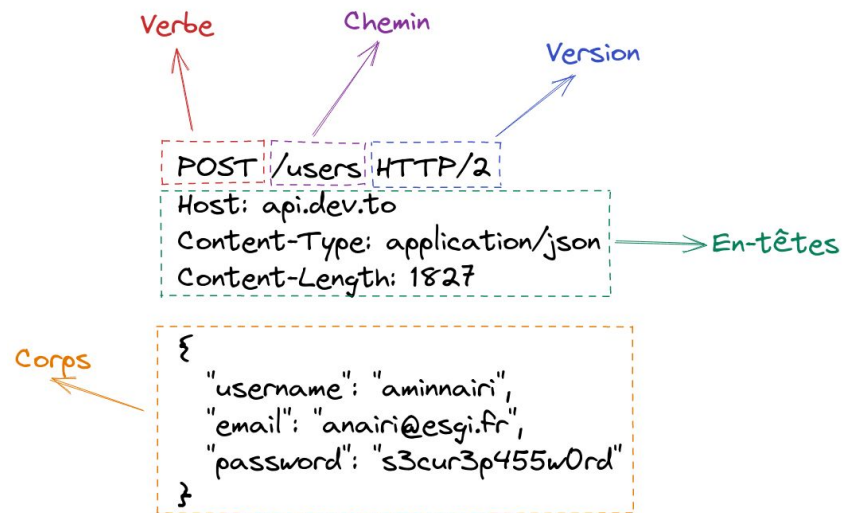


## 2.1 Requête

Une requête permet de demander une information à un serveur.

Cette requête comporte plusieurs parties dont un verbe HTTP, un chemin, une version du protocole, un ensemble d'en-têtes et un corps de requête.

Le corps de requête est optionnel et certains verbe n'en utilisent pas, d'autres l'utilisent pour attacher des données à la requête.

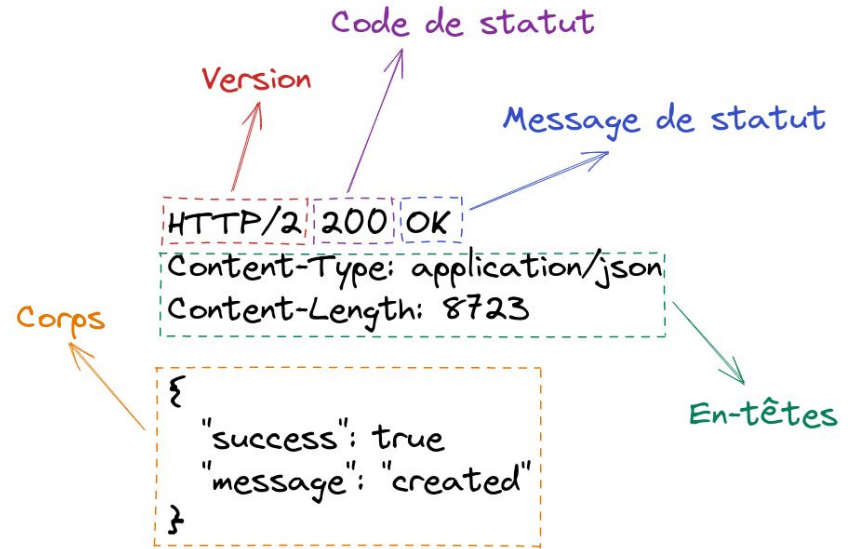


## 2.2 Réponse

Une réponse est en ensemble d'informations qui viennent suivent une requête.

La réponse contient la version du protocole HTTP supportée par le serveur, un code de statut, un message associé au code de statut, en ensemble d'en-têtes et un corps.

Tout comme pour la requête, la réponse n'a pas forcément de corps de réponse.



## 2.3 Verbe

Le verbe HTTP permet d'identifier l'intention derrière la requête faite par l'utilisateur.

Par exemple, si je souhaite récupérer une ressource, j'utiliserais le verbe GET. Si je souhaite créer une ressource, c'est le verbe POST que j'utiliserais. Si je souhaite modifier une ressource, j'utiliserais le verbe PATCH ou PUT. Et si je souhaite supprimer une ressource, j'utiliserais le verbe DELETE.

```
GET /users HTTP/2  
Host: api.dev.to
```

→ Récupération

```
POST /users HTTP/2  
Host: api.dev.to
```

→ Création

```
DELETE /users/29 HTTP/2  
Host: api.dev.to
```

→ Suppression

## 2.4 Statut

Le statut permet de savoir rapidement quel a été le résultat de la réponse.

Par exemple, si le serveur a effectivement créé la ressource que j'ai demandé avec succès j'utiliserais le code de statut 201. Ce code de statut correspond à une création de ressource.

Si la requête est incorrectement créée, par exemple lorsque j'oublie de renseigner mon adresse email lors d'une connexion, j'utiliserais le code de statut 400. Ce code de statut correspond à une requête mal formée.

Il existe beaucoup de code de statut correspondant à des scénarios de réponses divers qui sont standardisés par le protocole HTTP.

### Succès

200 : La requête est un succès  
201 : La ressource a été créée  
204 : La réponse n'a aucun contenu

### Redirection

301 : Redirection permanente avec changement de verbe  
307 : Redirection temporaire de la ressource  
308 : Redirection permanente de la ressource

### Erreur

400 : Requête mal formée  
401 : Client non authentifié  
403 : Accès interdit  
404 : Ressource introuvable

## 2.5 Version

Le navigateur est un programme qui implémente le protocole de communication HTTP. Son implémentation respecte le standard HTTP qui liste les fonctionnalités du protocole.

Il existe plusieurs version, qui ont permis d'améliorer le protocole avec le temps, avec notamment l'ajout de fonctionnalités qui permettent de répondre aux besoins des utilisateurs en constante augmentation sur les navigateurs internet.

HTTP/0.9 :

- Une méthode HTTP
- Un chemin
- Rien d'autre

HTTP/1.0 :

- Information sur la version HTTP
- En-têtes HTTP
- Code de statut dans la réponse
- Documents autres que du plain/text

HTTP/1.1 :

- Réutilisation des connexions TCP
- Plusieurs requêtes avant réponses possibles
- Réponses morcelées
- Négociation de type de contenu
- Mécanisme de contrôle du cache
- Hôtes

HTTP/2

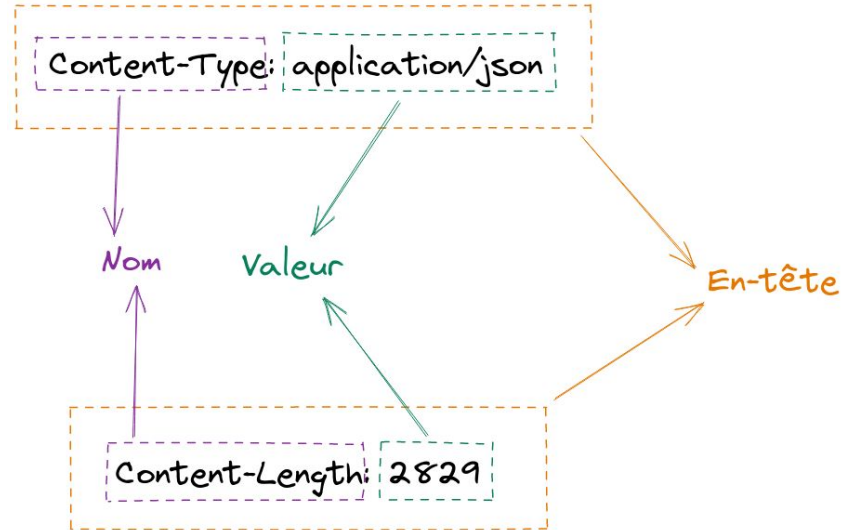
- Protocole binaire et non plus texte
- Multiplexage de requêtes (parallelisme)
- En-têtes compressés
- Server push

## 2.6 En-tête

Un en-tête est un mécanisme permettant d'ajouter des informations annexes à une requête.

Par exemple, vous pourriez souhaiter ne recevoir qu'un et un seul type de réponse, comme du HTML par exemple et pas du JSON. Ou vous pourriez souhaiter mettre en cache une requête de sorte à ne pas avoir à demander au serveur la même réponse si besoin.

Pour cela, nous utiliserons un en-tête. Il est composé d'un identifiant qui est son nom, ainsi que d'une valeur qui permet de décrire son contenu.



## 2.7 Corps

Une requête ou une réponse peut contenir un corps.

Dans une requête, un corps correspond aux données à envoyer au serveur. Par exemple, si je souhaite créer un utilisateur, ce seront les propriétés qui décrivent l'utilisateur.

Dans une réponse, cela correspond à des données qui répondent à une demande. Par exemple, la liste des utilisateurs déjà existants.

POST /users HTTP/2

Host: api.dev.to

Content-Type: application/json

```
{  
  "username": "aminnaïri"  
}
```

HTTP/2 200 OK

Content-Type: application/xml

Content-Length: 2810

```
<username>  
  aminnaïri  
</username>
```

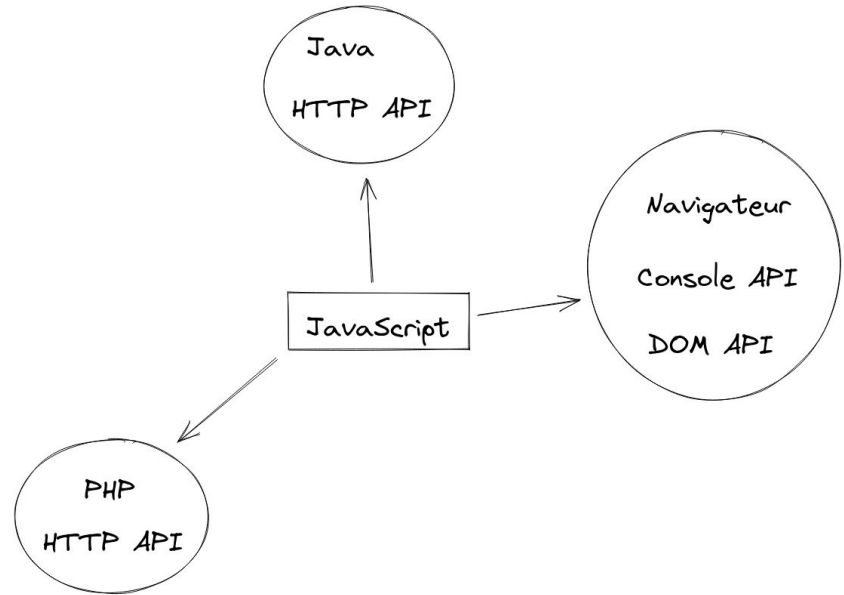
# 3 Qu'est-ce qu'une API

Une API au sens large est une interface de programmation applicative. Derrière ce terme complexe se cache en réalité un concept simple.

Un langage de programmation en tant que tel n'a rien d'intéressant puisqu'il ne permet en réalité aucune communication avec l'extérieur.

Ce sont les API disponibles qui permettent de rendre le langage intéressant. Comme par exemple l'API du DOM qui permet grâce au langage JavaScript de modifier un document HTML. Ou encore l'API du SDK Android qui permet de manipuler le module Bluetooth d'un appareil mobile en utilisant le langage Java ou Kotlin.

Cela peut également se matérialiser sous la forme d'un point d'entrée HTTP permettant d'interagir avec une base de données par exemple.



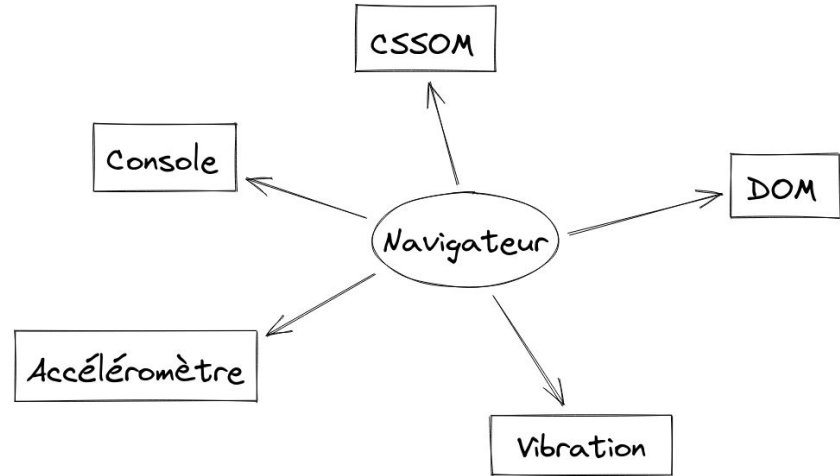


## 3.1 API Web

Lorsque nous parlons du Web, et plus particulièrement du navigateur, on parle bien souvent d'API Web.

Ce sont des interfaces qui nous permettent de communiquer avec du matériel ou avec d'autres langages.

On utilisera un langage de programmation qui est le JavaScript. On peut alors modifier un document HTML, faire vibrer un appareil, détecter les changements de direction de l'appareil, savoir lorsque l'utilisateur est à proximité de l'appareil etc...

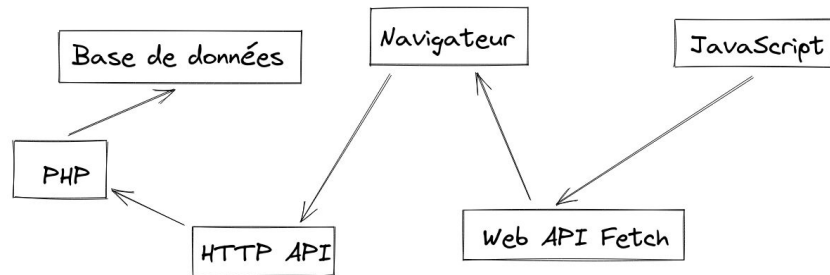


## 3.2 Service Web

Un service Web est un type d'API qui est exposé sur le réseau internet et qui communique en utilisant le protocole HTTP.

Bien souvent, cela correspond à un serveur qui va nous permettre d'interagir avec une base de données. Plutôt que d'écrire des requêtes SQL et de contacter le serveur de base de données nous-même, nous allons utiliser le protocole HTTP pour faire des requêtes qui seront interprétées par un serveur et qui les traduira en requêtes SQL.

Il est donc tout à fait possible d'utiliser un langage totalement différent du langage que l'on utilisera pour la présentation des données (JavaScript). On peut alors avoir des services Web écrit en Rust, en Haskell, en Java et continuer à développer une application Web en JavaScript.

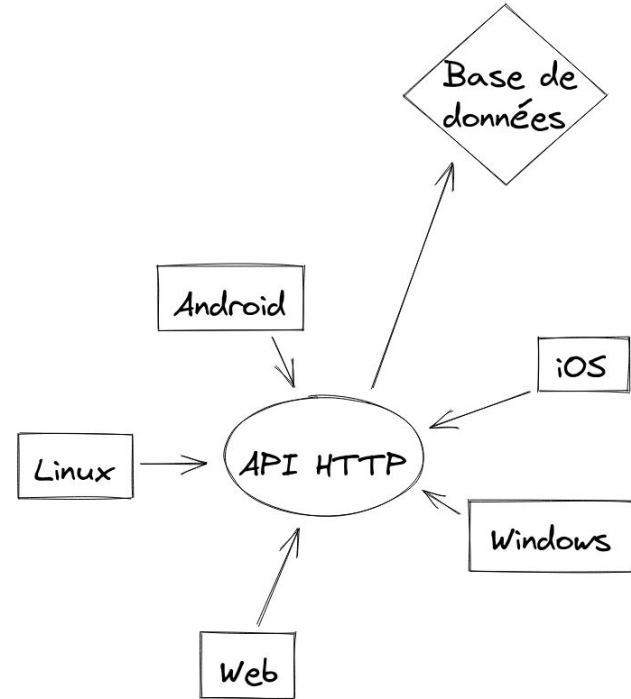


## 4 Dans quel cas créer une API

Créer une API n'est pas quelque chose d'anodin. Cela demande une toute autre organisation du code et de la consommation des données.

Par exemple, si la solution ne nécessite pas l'utilisation de plusieurs plateformes, alors il n'est pas nécessaire de créer une API. Un site internet suffit.

Lorsqu'il y a besoin de communiquer avec une même base de données dans le cadre d'une expérience utilisateur multi-applicative (mobile, Web, bureau, ...), dans ce cas là une API est nécessaire car il serait bien ennuyeux de devoir écrire plusieurs fois la même logique d'accès aux données pour plusieurs environnements différents.



# 5 Conception d'une API

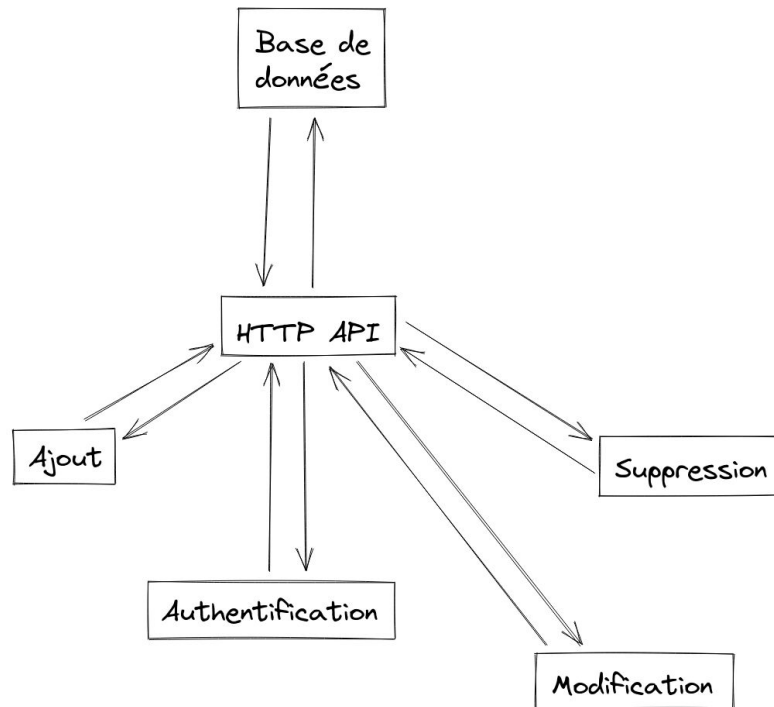
Dans le cadre de notre cours, nous aurons l'occasion de créer une API HTTP.

Cette API nous permettra notamment de pouvoir communiquer avec une base de données pour l'accès, la modification et la suppression aux données.

Elle nous permettra également de nous authentifier et de récupérer un jeton de connexion.

Nous verrons enfin comment tester notre API et les bonnes pratiques d'organisation.

Cette API sera conçue en PHP, mais nous aurions pu utiliser à peu près n'importe quel langage qui implémente dans son cœur une API pour communiquer en utilisant le protocole HTTP.



## 5.1 header

Lorsque nous souhaitons attacher un en-tête particulier à notre requête, nous utiliserons la fonction `header`.

Cette fonction prend en paramètre une chaîne de caractères qui contient le nom de l'en-tête et sa valeur. Les deux parties sont séparées par un double-point.

Le deuxième paramètre est un booléen qui permet de savoir si l'appel de cette fonction avec un même en-tête doit écraser un en-tête déjà défini auparavant.

Le troisième paramètre permet d'attacher un code de status HTTP. Nous n'utiliserons généralement pas ce dernier paramètre mais plutôt la fonction `http_response_code`.

```
<?php
```

```
header("Content-Type: application/json");  
header("Access-Control-Allow-Origin: http://127.0.0.1");  
header("Access-Control-Allow-Headers: GET,PATCH,POST,DELETE");
```

## 5.2 http\_response\_code

Cette fonction nous permet de paramétrer le code de statut HTTP liée à notre réponse.

Elle prend en paramètre un entier qui correspond au code de statut HTTP que nous souhaitons renvoyer dans notre réponse.

Elle renvoie l'ancien code de statut HTTP qui était auparavant paramétré.

```
<?php
```

```
http_response_code(201);
```

## 5.3 echo

Pour pouvoir paramétrer un corps à notre réponse HTTP, il suffit d'écrire sur la sortie standard.

Pour cela, on utilisera la fonction echo. Cette fonction prendra en paramètre une chaîne de caractères qui sera écrite dans la sortie standard.

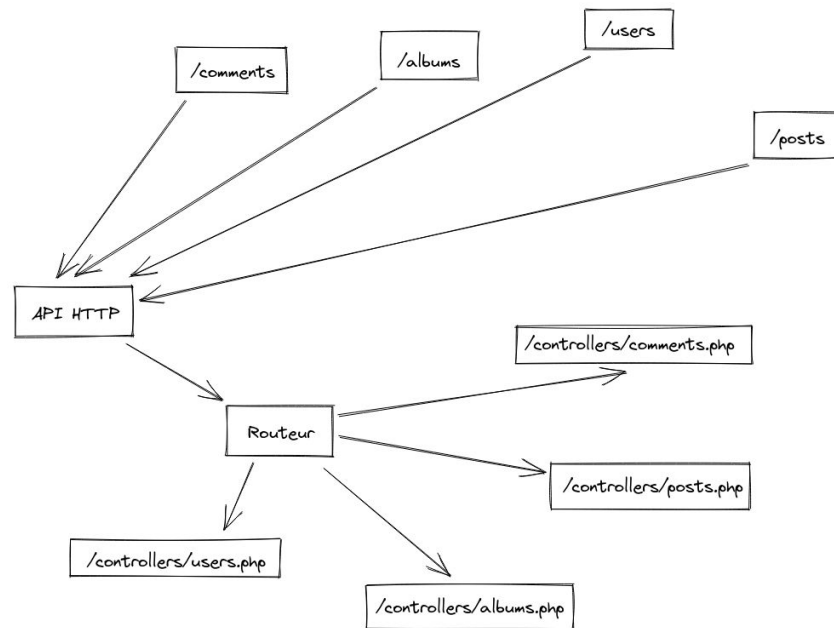
```
<?php  
echo '{"id": 1, "username": "aminnaïri"}';
```

## 5.4 Routeur

Un serveur est amené à recevoir beaucoup de requêtes, et il est essentiel de pouvoir les identifier correctement pour pouvoir s'organiser.

Il n'est pas question d'avoir un seul fichier sur notre API, nous allons être amené à répartir le développement de cette API dans plusieurs modules.

Le module principal (index.php) seront donc le point d'orgue de notre service Web. Il sera responsable de diriger les requêtes aux bons fichiers.





## 5.5 Base de données

Pour pouvoir se connecter à une base de données, on utilisera l'objet *PDO*, pour *PHP Data Object*.

Cet objet s'utilise avec le mot-clé `new`, qui permet de créer une nouvelle instance de ce dernier.

Il prend en paramètre trois chaînes de caractères et un tableau d'options.

Le premier est un DSN, ou Data Source Name. C'est une chaîne de caractères spéciale qui contient entre autre le pilote de base de données à utiliser, le nom de la base de données ainsi que le nom de domaine à utiliser pour atteindre la base de données.

Il contient également un deuxième paramètre qui est le nom d'utilisateur à utiliser pour accéder à cette base de données.

Le troisième paramètre est le mot de passe d'accès à la base de données.

Le dernier paramètre est un tableau d'options que peut prendre cet objet.

```
<?php
try {
    $connection = new PDO("mysql:dbname=webapi;host=localhost", "root", "root");
    echo "Connected.";
} catch (PDOException $error) {
    die($error->getMessage());
}
```

## 5.6 Patron de conception REST

Pour organiser ses routes, nous allons les séparer en entités logiques en suivant le patron de conception *REST*, ou ***RE***presentational ***State*** ***Transfer***.

L'idée est de créer des points d'entrées qui permettent à un client de pouvoir accéder à des données et de donner l'intention derrière la requête effectuée au travers de méthode HTTP.

C'est un patron de conception très répandue pour les API HTTP qui expose des données aux clients concernés.

```
<?php
```

```
// http://api.domain.com/users  
// http://api.domain.com/comments  
// http://api.domain.com/articles  
// http://api.domain.com/photos  
// http://api.domain.com/todos
```

## 5.7 Patron de conception CRUD

Un *CRUD*, ou **Create Read Update Delete** est un patron de conception très utilisé lorsque l'on utilise du REST.

Il nous permet notamment d'avoir des entités dont le contrôle est total, c'est-à-dire que les client ont la possibilité d'effectuer toutes les requêtes possible sur ces dernières qui permet de les manipuler sans avoir besoin d'accéder une seule fois à une base de données.

Un CRUD expose généralement les points d'entrées aux méthodes suivantes de manière exhaustives : GET, POST, PATCH ou PUT et DELETE.

```
<?php
```

```
// GET      http://api.domain.com/users  
// POST     http://api.domain.com/users  
// PUT      http://api.domain.com/users/1  
// PATCH    http://api.domain.com/users/1  
// DELETE   http://api.domain.com/users/1
```

```
// GET      http://api.domain.com/users/1/comments  
// POST     http://api.domain.com/users/1/comments  
// PUT      http://api.domain.com/users/1/comments/2  
// PATCH    http://api.domain.com/users/1/comments/2  
// DELETE   http://api.domain.com/users/1/comments/2
```

## 5.8 Patron de conception MVC

Une architecture MVC, ou Model View Controller est un modèle d'organisation hiérarchique de dossier qui permet d'avoir un meilleur contrôle sur son code-source, notamment lorsque ce dernier est amené à grandir avec l'apparition de nombreuses fonctionnalités.

Il permet de séparer en 3 parties logiques sont application.

La vue est la partie de représentation des données, le contrôleur est la partie de gestion du côté métier et le modèle est la représentation de nos entités et permet de faire le lien avec notre base de données.

Toutes ces trois parties sont intimement liées entre elle : Le modèle est passé au contrôleur. Le contrôleur effectue toutes les opérations permettant de récupérer ou modifier le modèle la vue permet d'afficher les retour du contrôleur aux clients qui le souhaitent et au format souhaité.

