

# INTRODUCTION API

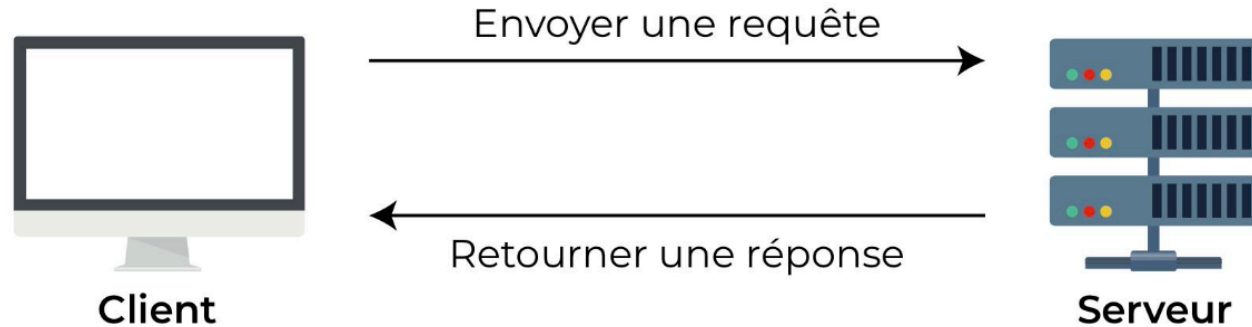
C&DI – 2023–2024

Attention le cours contient des mises en pratique qui serviront à votre rendu.

Pensez bien à conserver les différents exercices et mises en pratique.

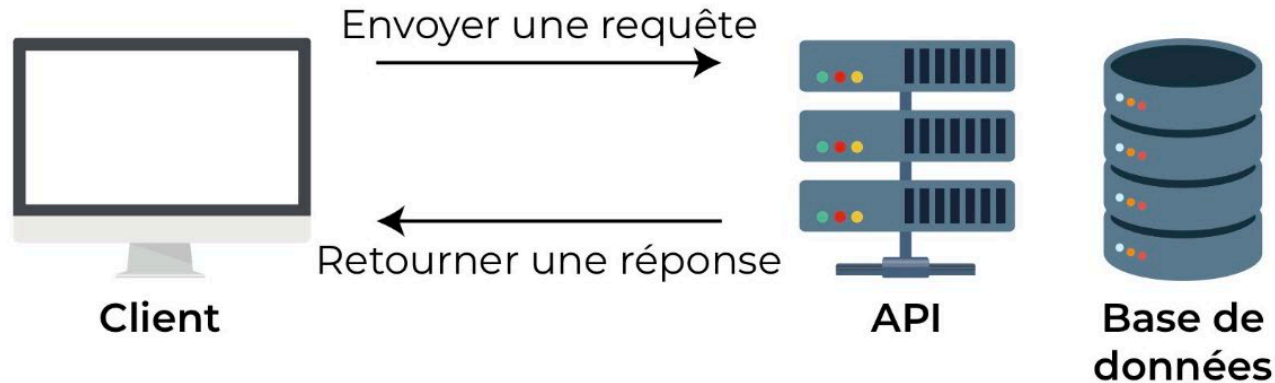
# API : késako ?

Avant de commencer, il est important de revoir les bases d'une simple communication client/serveur.



Le client effectue une requête afin d'obtenir une réponse du serveur. Le serveur renvoie ensuite l'information correspondante à la demande

Une API (application programming interface) est un intermédiaire qui va faciliter la communication entre deux entités.



Les API permettent la communication entre de nombreux composants différents de votre application, mais aussi entre autres développeurs.

Vous utilisez des API tous les jours !

Par exemple lorsque vous vous connectez sur votre l'application préférée, on vous propose ceci :



Les développeurs d'API créent des méthodes standardisées, réutilisables et documentées pour permettre à d'autres développeur d'accéder à des données spécifiques lors de la construction d'applications.

# Les différents types d'API

Il existe des API dites publiques et privées.

Les API publiques sont utilisables par n'importe qui, sans restriction d'accès. Par exemple, une API du gouvernement pour récupérer des données géographiques.

D'autres API ont une restriction en fonction du niveau d'accès que vous avez. Par exemple, l'API de OpenAI pour faire des prompts a une restriction d'utilisation/mois.

En général les API « privées » mettent à disposition des « clés API » afin de pouvoir s'authentifier à ces dernières (et prouver qu'on a bien le droit d'accéder à ce que l'on souhaite).



# Rappel de JavaScript

Reprenons rapidement les bases de JS.

Pour déclarer une variable on utilise :

- var
- let
- const

Une variable peut stocker une donnée ou plusieurs données de plusieurs types :

- string
- integer
- object
- array
- ...

```
let firstname = 'Nicolas'  
const lastname = 'de Garrigues'  
let age = 22
```

Les conditions peuvent s'écrire de différentes manières :

if / else

```
if(a == b) {  
    return true;  
} else {  
    return false;  
}
```

switch / case

```
switch(a + b) {  
    case 1:  
        return true;  
        break;  
    case 2:  
        return false;  
        break;  
    default:  
        return false;  
        break;  
}
```

ternaire (avancé)

```
a == b ? true : false
```

Une fonction peut s'écrire également de plusieurs manières :

- Classique

```
function addition(num1, num2) {  
  return num1 + num2;  
}
```

- Fléchée

```
let addition = (num1, num2) => {  
  return num1 + num2;  
};
```

# Les requêtes HTTP

Le protocole HTTP est un protocole de communication pour transférer des données sur le web.

Le client fait une demande au serveur, et il nous répond avec des données !

Ces requêtes sont guidées par une méthode :

Une méthode est un type d'action que le client souhaite effectuer sur la ressource.

- **GET** : récupérer des données
- **POST** : envoyer des données
- **PUT** : mettre à jour des données
- **DELETE** : supprimer des données

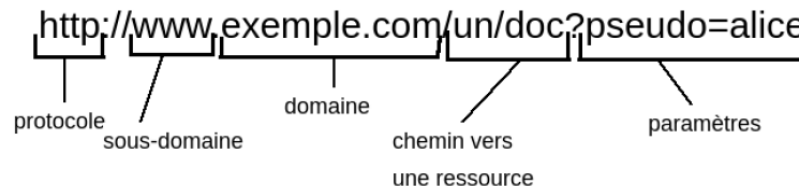
Par exemple, de quelle méthode s'agit-il ? :

- Pierrick souhaite s'inscrire sur mon site, il envoie ses informations pour qu'elles puissent être stockées
- Pierrick a tapé trop vite et s'est trompé dans son adresse, il envoie de nouvelles informations pour mettre à jour son lieu d'habitation
- Pierrick consulte sa page profil pour vérifier que tout est bon, il récupère ses informations pour les afficher sur le site
- Pierrick n'aime finalement pas ce site, il décide de supprimer son compte, et donc ses informations

# URLs



Comment fonctionne une URL ?



Vous connaissez probablement la base d'une URL.

- son protocole
- son sous-domaine
- son domaine
- son chemin

Mais saviez-vous qu'on peut envoyer également des informations complémentaires ?

Pour préciser certaines informations qui pourraient être variables, on peut placer des paramètres.

Vous utilisez ces paramètres tous les jours sans vous rendre compte lorsque vous allez sur n'importe quel site internet.

Lorsque vous effectuez une recherche sur X par exemple.

```
https://x.com/search?q=OnePiece
```

# FETCH

Fetch est une fonction JavaScript qui permet d'effectuer des requêtes réseau, généralement vers des serveurs web, et de gérer les réponses.

Elle offre une manière plus moderne et conviviale par rapport aux anciennes méthodes.

On va utiliser le fetch en 2 parties :

Syntaxe de base de **fetch()** : La fonction fetch prend l'URL de la ressource à récupérer comme argument et renvoie une promesse.

Pour gérer les réponses : La méthode **.then()** sur la promesse renvoyée par **fetch()**.

Syntaxe de base de **fetch()** :

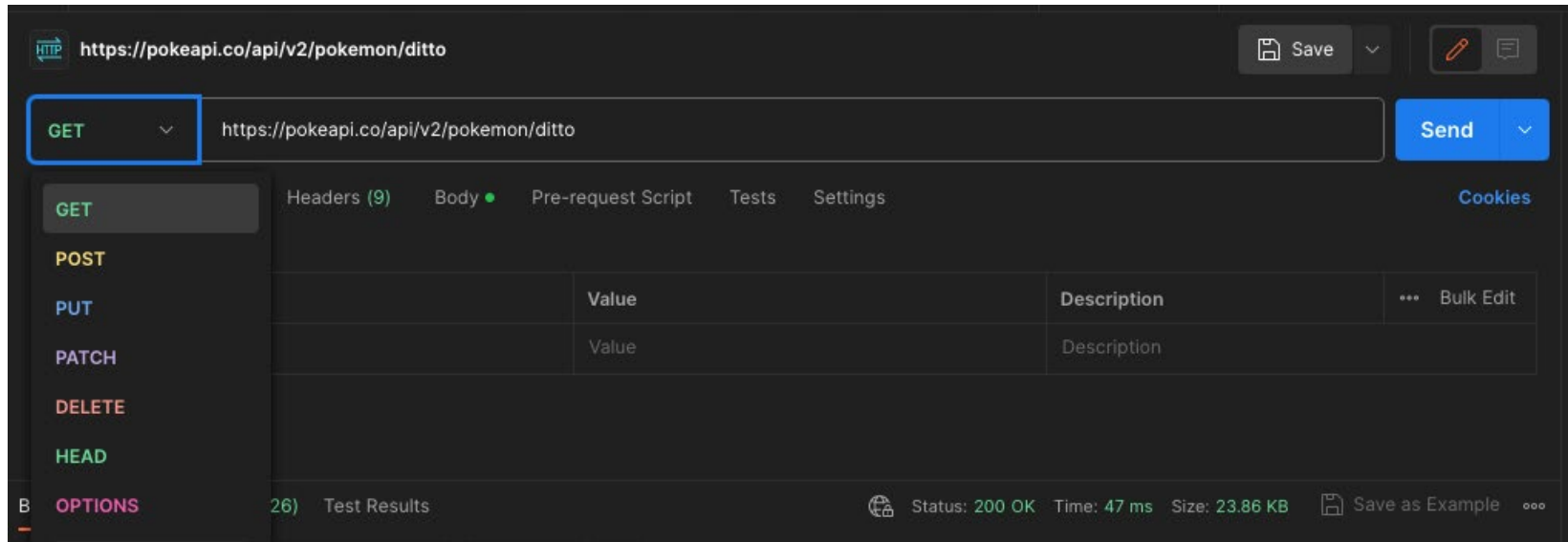
```
fetch('https://pokeapi.co/api/v2/pokemon/ditto')
```

On utilise ici l'API **Pokeapi**, qui regroupe l'intégralité des Pokemons, des lieux, des jeux, des baies... allant de la première à la dernière génération.

Pour vérifier si l'URL fonctionne, on peut utiliser l'application **Postman**.

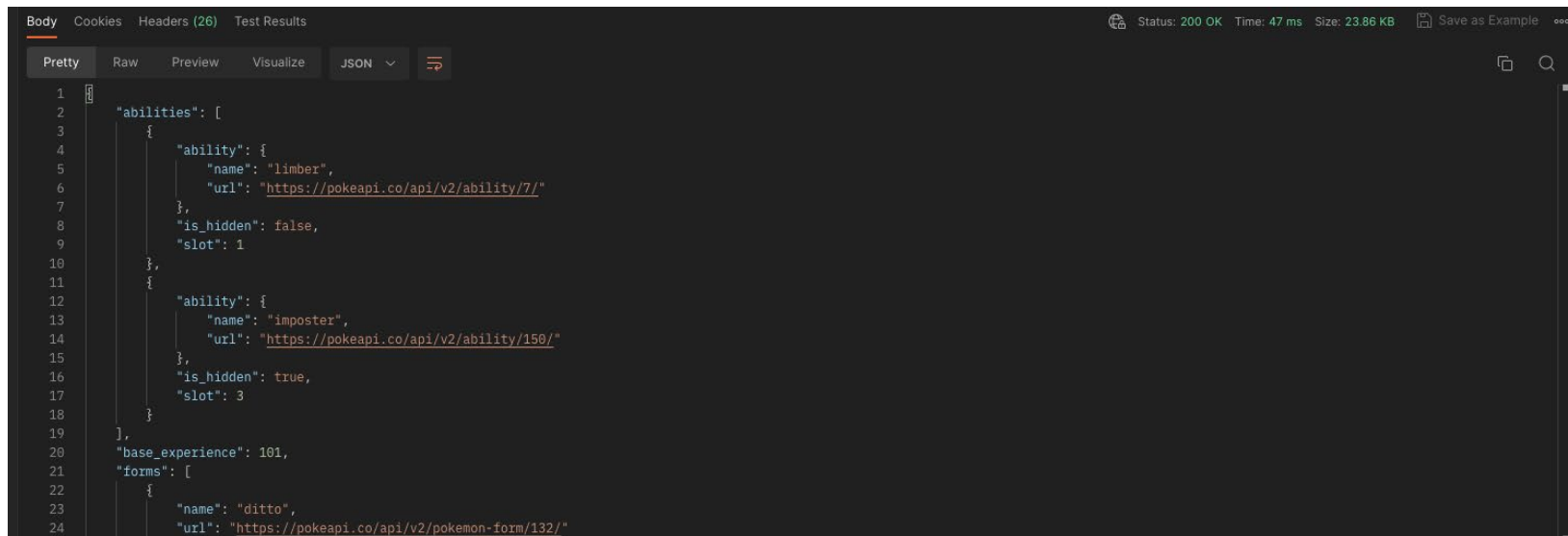
Pour vérifier cette URL, on va l'inscrire dans l'input principal et préciser sa méthode.

Ici on veut récupérer des informations, on va donc utiliser la méthode **GET**.



Lorsque l'on appuie sur le bouton « Send », on envoie une requête.

La réponse s'affiche en bas de cette manière :



The screenshot shows a web browser's developer console with the 'Body' tab selected. The JSON response is displayed in a 'Pretty' format. The response contains an array of abilities, a base experience value, and a list of forms. The first ability is 'limber' and the second is 'imposter'. The 'forms' array contains a single entry for 'ditto'.

```
1  {
2    "abilities": [
3      {
4        "ability": {
5          "name": "limber",
6          "url": "https://pokeapi.co/api/v2/ability/7/"
7        },
8        "is_hidden": false,
9        "slot": 1
10     },
11     {
12       "ability": {
13         "name": "imposter",
14         "url": "https://pokeapi.co/api/v2/ability/150/"
15       },
16       "is_hidden": true,
17       "slot": 3
18     }
19   ],
20   "base_experience": 101,
21   "forms": [
22     {
23       "name": "ditto",
24       "url": "https://pokeapi.co/api/v2/pokemon-form/132/"
25     }
26   ]
27 }
```

# JSON



Le JSON est un format simple et léger utilisé pour échanger des données entre différents programmes.

Il ressemble beaucoup à une liste d'informations organisées.

Imaginons que l'on souhaite partager facilement des informations d'utilisateur, on va l'écrire de cette façon :

C'est sous ce format que nous recevons les réponses d'API.

```
[
  {
    "name": "Nicolas",
    "age": "23",
    "city": "Nanterre"
  },
  {
    "name": "Alexis",
    "age": "27",
    "city": "Montreuil"
  }
]
```

Maintenant que l'on sait que l'URL fonctionne correctement, nous pouvons l'utiliser dans notre code.

On va rajouter le **.then** pour traiter la donnée.

```
fetch('https://pokeapi.co/api/v2/pokemon/1002')  
  .then((response) => response.json())  
  .then((data) => {  
    console.log(data)  
  })
```

Dans un premier temps, on va transformer la réponse en format JSON pour l'exploiter.

Ensuite, pour vérifier, on va faire un **console.log()** de data.

Si vous voyez des informations en format JSON dans la console de votre navigateur, tout est bon !

Ici on retrouve donc plusieurs informations liées au Pokémon Ditto (Métamorph) : ses capacités, sa taille, son poids, des photos de lui...

```
index.html:16
▼ {abilities: Array(2), base_experience: 101, forms: Array(1), game_indices: Array(20), height: 3, ...} 8
  ► abilities: (2) [{...}, {...}]
    base_experience: 101
  ► forms: [{...}]
  ► game_indices: (20) [{...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}]
    height: 3
  ► held_items: (2) [{...}, {...}]
    id: 132
    is_default: true
    location_area_encounters: "https://pokeapi.co/api/v2/pokemon/132/encounters"
  ► moves: [{...}]
    name: "ditto"
    order: 214
  ► past_types: []
  ► species: {name: 'ditto', url: 'https://pokeapi.co/api/v2/pokemon-species/132'}
  ► sprites: {back_default: 'https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/default.png', ...}
  ► stats: (6) [{...}, {...}, {...}, {...}, {...}, {...}]
  ► types: [{...}]
    weight: 40
  ► [[Prototype]]: Object
```

On veut maintenant afficher les informations de notre pokemon directement sur notre page.

Pour ça, on va procéder d'une manière un peu différente.

On va séparer la requête à l'API et l'affichage dans 2 fonctions différentes.

- Une fonction **fetchPokemon()** qui va return la réponse de la requête API.
- Une fonction **displayPokemon()** qui va afficher sur notre page.

```
function fetchPokemon(pokemon) {  
    return fetch('https://pokeapi.co/api/v2/pokemon/' + pokemon)  
        .then((response) => response.json())  
}
```

On n'oublie pas d'appeler la fonction **displayPokemon('ditto')** à la fin pour la lancer avec un Ditto comme recherche ^^

```
function displayPokemon(pokemon) {  
    const data = fetchPokemon(pokemon)  
    document.getElementById("pokemon").innerHTML = `  
        <h1>${data.name}</h1>  
          
    `;  
}
```

Et là, c'est le drame !  
ÇA NE FONCTIONNE PLUS !

Mais pourquoi ?

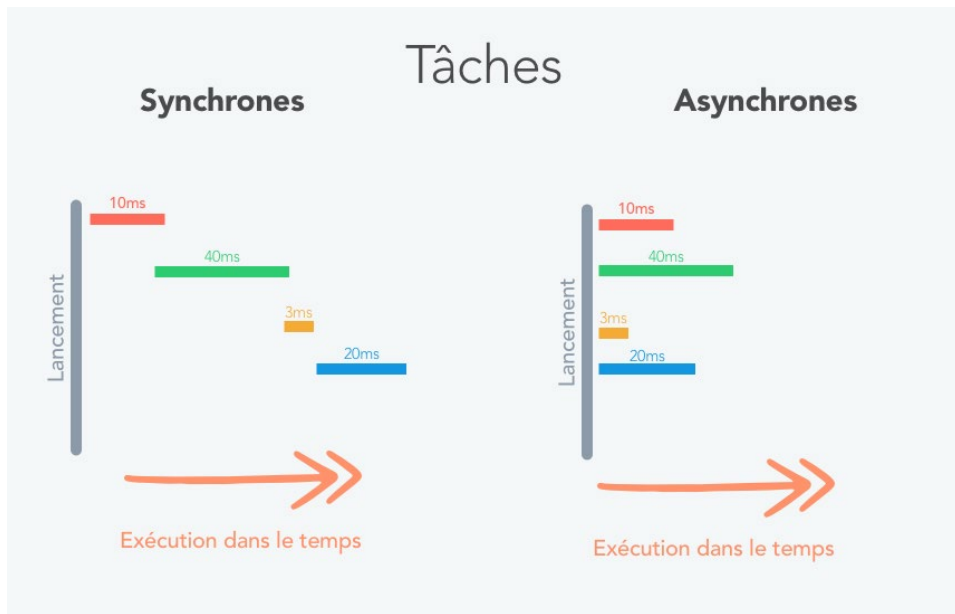
# Synchrone / Asynchrone

Le javascript fonctionne de manière dite « **synchrone** ».

Cela signifie que le code est exécuté de manière séquentielle, avec une instruction à la fois en suivant l'ordre d'écriture.

Chaque instruction doit se terminer avant que la suivante puisse commencer.

Mais il est possible de faire de **l'asynchrone**. Cela permet aux instructions lentes ou bloquantes de s'exécuter en arrière-plan tout en permettant au reste du code de s'exécuter.



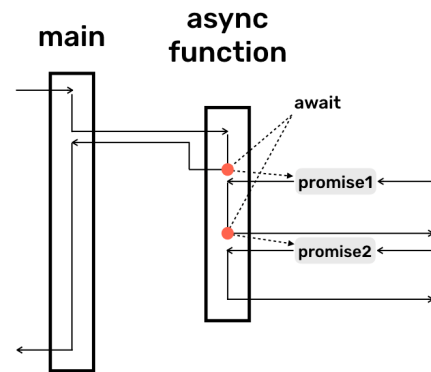
## D'accord mais pourquoi nous avons besoin de ça dans notre cas ?

Et bien puisque la fonction **fetch()** est une fonction asynchrone, ce qui veut dire qu'elle n'attendra pas la réponse pour passer à la suite.

On utilise donc **async** / **await** qui sont des directives qui spécifient le fonctionnement des instructions :

**async** force la fonction à retourner une promesse\*.

**await** capture une promesse en cours de traitement et bloque l'exécution en attendant la résolution de cette dernière.





Adaptons notre code avec ces directives :

```
async function displayPokemon(pokemon) {  
  const data = await fetchPokemon(pokemon)  
  document.getElementById("pokemon").innerHTML = `  
    <h1>${data.name}</h1>  
      
  `;  
}
```

La fonction **displayPokemon** est passée en asynchrone.

Ensuite on demande à la fonction **fetchPokemon** d'attendre la réponse avant de l'afficher.

Et c'est bon ! **ditto**



# Les API privées

On va voir maintenant comment utiliser une API privée.

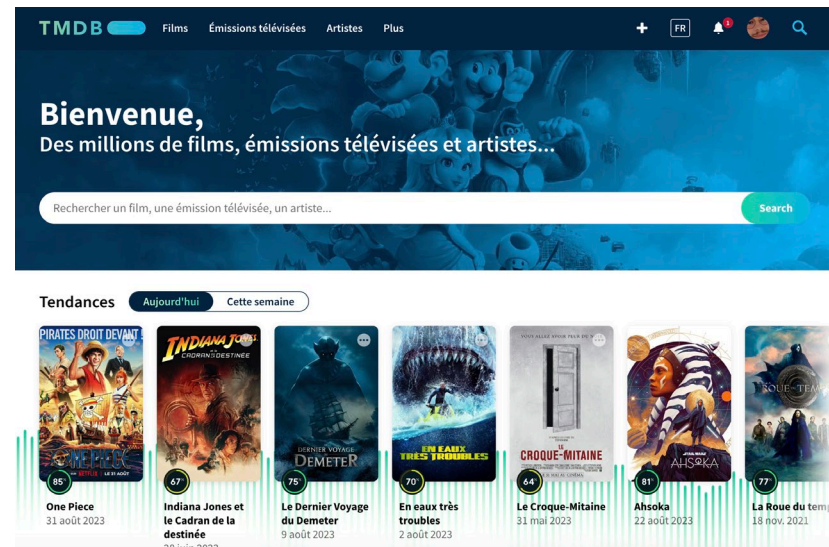
Une API privée utilise donc un token, qui est une sorte de clé d'accès qui ressemble à un code wifi beaucoup trop long.

ex: f28lia5fe6aod0mnl28o51c22178f344

Ce token permet de s'authentifier.

Le besoin d'authentification est souvent lié au nombre limité de requêtes autorisées

On va utiliser l'API de TheMovieDatabase pour notre exemple.



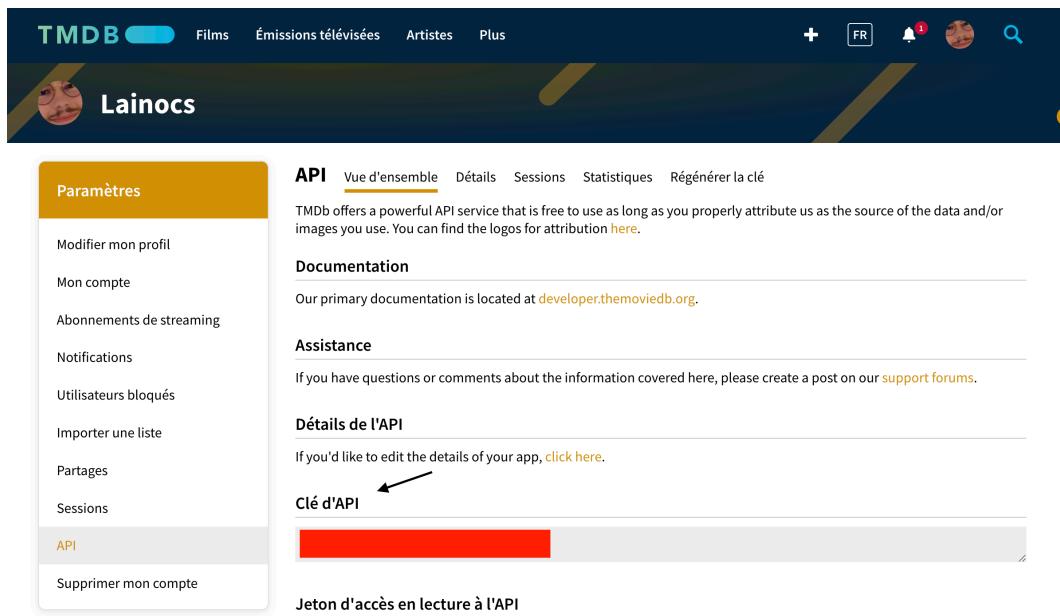
Pour cela, on va créer un compte TMDb.

Une fois le compte créé, la clé est disponible dans les paramètres de l'utilisateur.

## The Movie Database

On va voir comment l'utiliser de manière sécurisée.

(On ne va pas la publier sur GitHub pour éviter que d'autres utilisent nos crédits)



The screenshot shows the TMDb website interface. At the top, the navigation bar includes 'TMDb', 'Films', 'Émissions télévisées', 'Artistes', and 'Plus'. The user's profile 'Lainocs' is visible. A sidebar on the left contains a 'Paramètres' (Settings) menu with options like 'Modifier mon profil', 'Mon compte', 'Abonnements de streaming', 'Notifications', 'Utilisateurs bloqués', 'Importer une liste', 'Partages', 'Sessions', 'API', and 'Supprimer mon compte'. The 'API' option is highlighted. The main content area shows the 'API' section with tabs for 'Vue d'ensemble', 'Détails', 'Sessions', 'Statistiques', and 'Régénérer la clé'. The 'Vue d'ensemble' tab is active, displaying information about the TMDb API service, documentation, assistance, and details about the API. The 'Clé d'API' (API Key) is shown in a red box, with an arrow pointing to it from the text 'Clé d'API'.

Pour utiliser cette API, on va devoir placer le token en paramètre d'URL.

```
https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mnl28o51c22178f344
```

On retrouve tout d'abord dans le chemin la version de l'API, le type de contenu qu'on souhaite rechercher, puis l'identifiant lié au film.

Ensuite on précise notre token avec le paramètre « **api\_key** » pour s'authentifier et accéder à l'API.

Pour le reste, c'est identique !

Star Wars: Episode III - Revenge of the Sith



```
function fetchStarWars3() {  
  return fetch(  
    'https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mnl28o51c22178f344'  
  )  
    .then((response) => response.json())  
  }  
  
async function displayStarWars3() {  
  const data = await fetchStarWars3()  
  document.getElementById("star-wars-3").innerHTML = `  
    <h1>${data.title}</h1>  
      
  `;  
  displayStarWars3()  
}
```

Cette API propose également un 2ème paramètre qui pourrait nous intéresser.

On peut changer la langue des informations récupérées !

Pour ça, il faut simplement rajouter le paramètre « **language** ».

On sépare 2 paramètres par un « **&** ».

```
https://api.themoviedb.org/3/movie/1895?api_key=f28lia5fe6aod0mnl28o51c22178f3448&language=fr-FR
```

Et on se retrouve maintenant avec toutes les informations traduites en français !

Plein d'autres possibilités sont disponibles.

Vous pouvez essayer librement tout ce qui est proposé sur l'API.

[API The Movie Database](#)

Star Wars, épisode III - La Revanche des Sith





# Rendu

### ■ Rendu des Mises en pratiques du cours 5 pts

- API Movie Data Base
- API Pokémon
- Mini Quizz Requêtes HTTP

### ■ Rendu de l'implémentation de l'API Harry Potter sur votre projet d'axe 10 pts

API Harry Potter : <https://hp-api.lainocs.fr>

Implémenter la récupération des données depuis l'API et l'affichage sur votre système de carte.

Continuez votre projet d'axe selon les modalités inscrit dans le PPT du projet d'axe.

Soyez curieux et chercher par vous-même de nouveaux moyens d'améliorer votre site.

### ■ Qualité Visuelle, Fonctionnelle et UX de votre site projet d'axe à ce stade 5 pts

Ne négligez pas l'aspect visuel, ainsi que l'expérience utilisateur ou les bugs.

#### Attention aux **MALUS** !!! :

- Code non ou trop peu commenté
- Nomenclatures fantaisistes des fichiers, variables, fonctions
- Dossier de rendu mal rangé
- Fonctionnalités ou fichiers demandés mal indiqués faisant perdre du temps lors des corrections
- GitHub en privé ou lien erroné