**API & API Rest**

[**OpenClassRooms.com**](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web)

Table des matières

[Utilisez des API REST pour vos projets de code 2](#_Toc204964133)

[Tirez un maximum de ce cours 2](#_Toc204964134)

[Initiez-vous au fonctionnement des API 2](#_Toc204964135)

[Identifiez les avantages d’une API REST 2](#_Toc204964136)

[Utilisez les ressources et collections REST 2](#_Toc204964137)

[Quiz : Servez-vous des API REST pour vos projets de code 2](#_Toc204964138)

[Formulez des requêtes et envoyez des réponses avec une API REST 2](#_Toc204964139)

[Utilisez Postman pour formuler vos requêtes 2](#_Toc204964140)

[Réalisez vos premières requêtes sur une API 2](#_Toc204964141)

[Authentifiez une API pour plus de sécurité 2](#_Toc204964142)

[Entraînez-vous avec l’API GitHub 2](#_Toc204964143)

[Quiz : Servez-vous des API REST pour vos projets de code 2](#_Toc204964144)

[Concevez des API REST 2](#_Toc204964145)

[Définissez la structure de votre API REST 2](#_Toc204964146)

[Concevez les endpoints de votre API 2](#_Toc204964147)

[Utilisez les fonctionnalités avancées des endpoints 2](#_Toc204964148)

[Choisissez des frameworks pour construire votre API 2](#_Toc204964149)

[Résumé du cours 2](#_Toc204964150)

[Quiz : Concevez des API REST 2](#_Toc204964151)

**Références et outils.**

* Test et construction API : [Postman](https://www.postman.com/)

Plus loin :

* identifier

# Utilisez des API REST pour vos projets de code

## [Tirez un maximum de ce cours](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6573188-tirez-un-maximum-de-ce-cours)

Dans ce cours, vous allez dans un premier temps vous initier au fonctionnement des API : comprendre ce qu’est une API, son fonctionnement, ce qui caractérise une API et en particulier une API REST, car c’est ce type d’API que nous allons étudier dans ce cours !

Ensuite viendra le temps de la pratique ⚙️; vous verrez comment sont structurées les requêtes et les réponses et vous commencerez par formuler des requêtes sur une API afin d’obtenir vos premières données. Puis, vous utiliserez l’API de GitHub afin de créer, modifier et supprimer des ressources directement sur votre profil GitHub.

Enfin, vous utiliserez toutes les connaissances acquises durant les deux premières parties pour construire votre première API REST.

## [Initiez-vous au fonctionnement des API](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6816951-initiez-vous-au-fonctionnement-des-api)

**Découvrez ce qu’est une API**

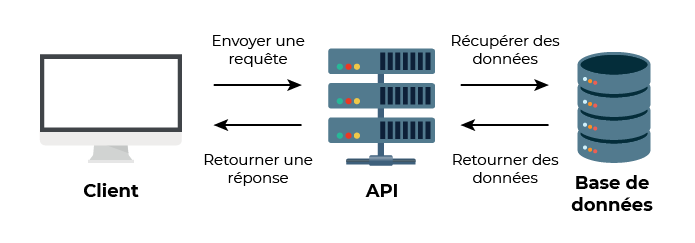
Vous trouvez plusieurs offres de trains pour Lisbonne en passant par l’Espagne. Et tout ça sans avoir à aller chercher l’information sur le site de chaque compagnie ferroviaire ou aérienne. Pas mal, non ?

 En effet, un gros travail est effectué pour rendre cela possible et pour que cela fonctionne, il nous faut l’un des outils les plus importants : **une API**.

API est une abbréviation et signifie ***Application Programming Interface*** (ou***interface de programmation d’application***, en français). Pour faire simple : c’est un moyen de communication entre deux logiciels, que ce soit entre différents composants d’une application ou entre deux applications différentes.

**Découvrez le fonctionnement des API**

En web, un service web et une API sont tous les deux des moyens de communication. Un service web standard facilite seulement la communication entre deux machines via un réseau. Une API facilite l’interaction entre deux applications différentes afin qu’elles puissent communiquer entre elles : elle sert d’intermédiaire. Le client va demander à l’API une information, celle-ci va aller chercher cette information dans la base de données puis la renvoyer au client dans un second temps.



Les API permettent la communication entre de nombreux **composants** différents de votre application, mais aussi entre des composants de votre application et d’autres **développeurs**. Elles agissent ainsi comme un intermédiaire qui transmet des messages à travers un système de requêtes et de réponses.

On crée une application de comparateur de vols que l’on va l’appeler VolScanner. Celle-ci ne peut pas accéder directement aux informations d'Air France ou de toute autre compagnie aérienne. En effet, l’application n’a pas accès à leurs bases de données... Mais si Air France a une API à qui on peut demander des informations et qui partage certaines données de la base de données avec d’autres applications, alors VolScanner peut demander des informations à l’API d'Air France. L’API lui renvoie alors des données que VolScanner peut partager !

Ainsi, VolScanner peut comparer les prix entre les différentes compagnies qui ont mis en place un vol le 10 décembre pour Lisbonne. À vous les délicieuses pasteis de nata ! ✨

Les API peuvent communiquer :

* d’un logiciel à un logiciel ;
* d’un client à un serveur ;
* ou d’un logiciel à des développeurs.

Je suis certaine que vous avez déjà vu un exemple d’utilisation d’une API pour communiquer entre logiciels et développeurs : sur certains sites, vous pouvez utiliser votre compte Google ou Facebook pour vous identifier sans avoir à créer un identifiant et un mot de passe.

Les API créent des méthodes **standardisées**et **réutilisables**qui permettent aux développeurs d’accéder à des données spécifiques lors de la construction d’applications.

**Observez comment utiliser les API en tant que développeur**

En tant que développeur, vous serez certainement amené à utiliser diverses API dans votre vie professionnelle ou pour vos projets personnels. Il existe deux types principaux : les API privées et les API publiques. Voyons ensemble de quoi il s'agit ! 🕵🏻‍♀️

*Les API privées*

Les API **privées** garantissent que les personnes en dehors de votre entreprise ou de votre application n’ont pas accès aux données disponibles de votre base de données.

Une API peut être utilisée comme un **tampon** ou une couche intermédiaire entre la base de données et la personne qui veut accéder ou manipuler les données. Une requête directe et non contrôlée sur une base de données pourrait engendrer le chaos !

Une API permet un niveau de sécurité supplémentaire pour mieux gérer l’accès et les modifications des données, en attribuant ce qu’on appelle des droits aux personnes qui en ont besoin. Ainsi, on s’assure de contrôler les utilisateurs qui auront ou non accès à la base de données.

*Les API publiques*

Contrairement aux API privées, les API que l’on appelle **publiques** sont utilisables par d’autres personnes, qu’elles soient sur votre application ou non. Elles permettent aux développeurs de récolter les données d’une autre. Il existe de nombreuses manières d’utiliser des données provenant d’API tierces (ou externes), mais en voici quelques-unes :

1. Imaginons que vous vouliez construire un site web qui répertorie les conditions météo des stations de ski. Plutôt que de collecter vos propres données météorologiques, vous pouvez utiliser une [API de météo](https://openweathermap.org/api) et y trouver vos données ! 🌤
2. Si vous êtes auteur-compositeur-interprète et que vous voulez créer un site web pour que vos fans puissent écouter votre musique, au lieu de construire votre propre lecteur de musique en streaming, vous pouvez utiliser[l’API de Spotify](https://developer.spotify.com/documentation/web-api/) et écouter votre musique directement sur votre site web ! 🎵
3. Vous voulez créer une page de fans pour votre série télé favorite (Kaamelot, bien sûr), en réunissant tous les comptes Instagram des différents acteurs sur un seul site web – devinez quoi, il existe une [API Instagram](https://developers.facebook.com/docs/instagram-api/?locale=fr_FR) pour vous aider à le faire ! 🌅

Il existe également certaines API à mi-chemin entre une API publique et privée. Cela peut se produire quand différentes**requêtes**de l’API sont possibles uniquement en fonction du niveau d’**accès** dont vous disposez. 🔐 Nous y reviendrons plus tard lorsque nous traiterons de l’authentification. 😉

Il existe des milliers d’API publiques que les développeurs peuvent utiliser de différentes façons pour améliorer leurs projets. Vous trouverez ici une liste de ces [API disponibles publiquement](https://github.com/toddmotto/public-apis)que vous pouvez utiliser !

*En résumé*

* Les API permettent de communiquer des données.
* Elles permettent la communication entre différents composants de votre application **et** entre votre application et d’autres développeurs, par l’utilisation de**requêtes** et de **réponses**.
* Elles donnent un moyen d’accès aux données de façon réutilisable et standardisée.
* Les développeurs peuvent utiliser des API publiques et privées.
* Les API publiques sont utilisables par tous sans restriction.
* Les API privées sont utilisables seulement par ceux qui ont un accès et y sont autorisés.

Pourquoi utiliser une API REST en particulier et pas une simple API ? REST possède de nombreux avantages. Regardons ensemble dans le chapitre suivant !

## [Identifiez les avantages d’une API REST](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6817216-identifiez-les-avantages-d'une-api-rest)

Bien ! Maintenant que vous savez ce qu’est une API, parlons de ce qui compose une **API REST**. Nous utiliserons REST dans ce cours, car c’est le plus populaire. C’est l’un des standards de création d’API les plus logiques, efficaces, et utilisés. Et, d’après le [rapport 2017 de l’état d’intégration des API de Cloud Elements](https://jaxenter.com/state-of-api-integration-report-136342.html) (en anglais), 83 % des API sont des API REST.

*Comprenez tous les avantages de REST*

REST signifie ***Re***presentational***S***tate***T***ransfer (ou transfert d’état de représentation, en français), et constitue un ensemble de **normes**, ou de lignes directrices **architecturales** qui structurent la façon de communiquer les données entre votre application et le reste du monde, ou entre différents composants de votre application.

Nous utilisons l’adjectif RESTful pour décrire les API REST. Toutes les API REST sont un type d’API – mais toutes les API ne sont pas RESTful !

Les API RESTful se basent sur le protocole **HTTP** pour transférer les informations – le même protocole sur lequel la communication web est fondée ! Donc, lorsque vous voyez **http** au début d’une URL, comme <http://twitter.com>– votre navigateur utilise HTTP pour faire une requête de ce site web au serveur. REST fonctionne de la même façon !

Si vous ne vous souvenez plus de ce qu’est le protocole HTTP, jetez un œil à ce chapitre du cours [**Comprendre le web**](https://openclassrooms.com/fr/courses/1946386-comprendre-le-web/6874807-decouvrez-les-protocoles).

Il y a **six** lignes directrices architecturales clés pour les API REST. Voyons ensemble de quoi il s’agit :

*1 : Client-serveur separation*

L’une des normes de REST est la **séparation du client et du serveur**.

Un **client** est celui qui va utiliser l’API. Cela peut être une application, un navigateur ou un logiciel.

Un **serveur**est un ordinateur distant capable de récupérer des données depuis la base de données, de les manipuler si besoin et de les renvoyer à l’API, comme ce gros ordinateur au milieu :

Cette séparation permet au client de s’occuper uniquement de la récupération et de l’affichage de l’information et permet au serveur de se concentrer sur le stockage et la manipulation des données.

Les API REST offrent un **moyen de communication standardisé entre le client et les données** (selon les lignes directrices architecturales REST, en utilisant le protocole http).

C’est particulièrement utile lorsque de grandes équipes de développeurs travaillent sur une même application. Vous pouvez avoir une équipe qui travaille indépendamment sur le backend tandis que l’autre travaille sur le frontend. Comme l’API REST communique entre les deux, cela permet aux développeurs de scaler plus facilement les applications et aux équipes de travailler de manière plus efficace. 🤝

*2 : Stateless*

L’un des autres aspects uniques des API REST est qu’elles sont **stateless** – sans état, en français – ce qui signifie que le serveur ne sauvegarde aucune des requêtes ou réponses précédentes.

OK, mais alors concrètement, qu’est-ce que cela signifie pour les API REST ? 🤔

Étant donné que chaque message est isolé et indépendant du reste, il vous faudra vous assurer d’envoyer avec la requête que vous formulez toutes les **données nécessaires** pour être sûr d’avoir la réponse la plus précise possible. Cela nous donnerait quelque chose comme : « Est-ce que je pourrais avoir du ketchup sur les **frites** que **j’ai** commandées à **ma** table ? » Avec toutes ces informations, votre serveuse pourra identifier à quelles frites il faut ajouter du ketchup !

**Le fait d’être stateless** rend chaque requête et chaque réponse très **déterminée** et **compréhensible**. Donc, si vous êtes développeur et que vous voyez la requête API de quelqu’un d’autre dans un code déjà existant, vous serez capable de comprendre l’objet de la requête sans contexte ! 👌

*3 : Cacheable (ou*sauvegardable*, en français)*

La réponse doit contenir l’information sur la capacité ou non du client de mettre les données **en cache**, ou de les sauvegarder. Si les données **peuvent être mises en cache**, la réponse doit être accompagnée d’un numéro de version. Ainsi, si votre utilisateur formule deux fois la même requête (c’est-à-dire s’il veut revoir une page) et que les informations n’ont pas changé, alors votre serveur n’a pas besoin de rechercher les informations une deuxième fois. À la place, le client peut simplement mettre en cache les données la première fois, puis charger à nouveau les mêmes données la seconde fois. 💪

Une mise en cache efficace peut réduire le nombre de fois où un client et un serveur doivent interagir, ce qui peut aider à accélérer le temps de chargement pour l’utilisateur ! 👏

*4 : Uniforme Interface (interface uniforme)*

Lors de la création d’une API REST, les développeurs acceptent d’utiliser les mêmes normes. Ainsi, chaque API a une **interface uniforme**. L’interface constitue un contrat entre le client et le service, que partagent toutes les API REST.

Une API REST d’une application peut communiquer de la même façon avec une autre application entièrement différente.

*5 : Layered system (système de couches)*

Chaque composant qui utilise REST n’a pas accès aux composants au-delà du composant précis avec lequel il interagit.

Cela signifie qu’un client qui se connecte à un composant intermédiaire n’a aucune idée de ce avec quoi ce composant interagit ensuite. Par exemple, si vous faites une requête à l’API Facebook pour récupérer les derniers posts : vous n’avez aucune idée des composants avec lesquels l’API Facebook communique.

Cela encourage les développeurs à créer des composants indépendants, facilitant le remplacement ou la mise à jour de chacun d’entre eux.

*6 : Code on demand (code à la demande)*

Le code à la demande signifie que le serveur peut étendre sa fonctionnalité en envoyant le code au client pour téléchargement. C’est facultatif, car tous les clients ne seront pas capables de télécharger et d’exécuter le même code – donc ce n’est pas utilisé habituellement, mais au moins, vous savez que ça existe !

*Découvrez les alternatives aux API REST*

REST n’est qu’**un** type d’API. Il existe des alternatives qui vous seront également utiles à connaître, notamment les API **SOAP**.

SOAP est l’acronyme de ***Simple Object Access Protocol***, ou protocole simple d’accès aux objets, en français. Contrairement à REST, il est considéré comme un protocole, et non comme un style d’architecture.

Les API SOAP étaient les API les plus courantes avant l’arrivée de REST. REST utilise le protocole HTTP pour communiquer, SOAP d’un autre côté peut utiliser de multiples moyens de communication. Le souci, c’est la complexité qui en ressort, car les développeurs doivent se coordonner pour s’assurer qu’ils communiquent de la même manière afin d’éviter les problèmes. De plus, le SOAP peut demander plus de bande passante, ce qui entraîne des temps de chargement beaucoup plus longs. REST a été créé pour résoudre certains de ces problèmes grâce à sa nature plus légère et plus flexible.

De nos jours, le SOAP est plus fréquemment utilisé dans les applications de grandes entreprises, puisqu’on peut y ajouter des couches de sécurité, de confidentialité des données, et d’intégrité supplémentaires. REST peut être tout aussi sécurisé, mais a besoin d’être implémenté, c’est-à-dire d'être développé au lieu d’être juste intégré comme avec le SOAP.

*En résumé*

* Toutes les API ne sont pas RESTful et les API REST ont des lignes directrices architecturales spécifiques.
* Les avantages clés des API REST sont les suivants :
  + la séparation du client et du serveur, qui aide à scaler plus facilement les applications ;
  + le fait d’être stateless, ce qui rend les requêtes API très spécifiques et orientées vers le détail ;
  + la possibilité de mise en cache, qui permet aux clients de sauvegarder les données, et donc de ne pas devoir constamment faire des requêtes aux serveurs.
* SOAP est un autre type d’API, mais est plus utilisé dans les grandes entreprises.

Vous venez de voir la structure d’une API REST et ses avantages ; il est temps de voir ce qui constitue une API REST : les ressources. Suivez-moi dans le prochain chapitre, et attaquons-nous aux ressources !

## [Utilisez les ressources et collections REST](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6817356-utilisez-les-ressources-et-collections-rest)

### Appréhendez les données REST via l’utilisation des ressources

Les données REST sont représentées dans ce qu’on appelle des ***ressources regroupées dans des collections***. (un peu comme des instances d’un objet).

Une ressource peut être tout type d’objet **nominal** (on lui attribue un nom) que vous pouvez utiliser pour représenter les données dans votre application. Vous savez, une personne, un lieu, ou autre chose ! 😉 Pour faire simple, voyez les ressources comme des boîtes dans lesquelles vous rangerez des objets par catégorie et sur lesquelles vous collez une étiquette pour savoir quoi mettre dedans.

Vous trouvez que c’est abstrait ? C’est le but, afin que vous puissiez représenter n’importe quel élément de donnée sous la forme que vous souhaitez.

Chaque **ressource** comporte des informations supplémentaires sur les données contenues. Si on prend l’exemple d’une application qui liste les héros Marvel, une des ressources pourrait être Superhero et on pourrait avoir par exemple un nom, une description, etc., comme information supplémentaire.

Les ressources sont regroupées dans un groupe que l’on appelle une **collection**. On s’y réfère avec la forme au **pluriel**du nom de la ressource. Par exemple une ressource superhero donnerait superheroes.

Par convention, tous les champs d’une ressource et le nom d’une collection sont en anglais. Ils sont traduits ici en français pour une meilleure compréhension du cours, mais privilégiez toujours l’anglais !

Imaginons que vous créez une API pour qu’une boutique de skateboards ait un service de livraison en ligne. Ce que vous voulez, c’est que d’un côté vos clients puissent acheter des skateboards sur le site web, et de l’autre que vos salariés puissent ajouter des produits et mettre l’inventaire à jour.

Procédons étape par étape et déterminons ensemble les ressources, leurs informations supplémentaires et les collections.

Pour **une boutique de skateboards,**vos ressources pourraient être :

* Client ;
* Staff ;
* Basket (panier) ;
* Skateboard ;
* Inventory (inventaire).

Une ressource Skateboard pourrait comporter comme informations supplémentaires : nom,  marque, id,  prix.

Vos collections seraient donc :

|  |  |
| --- | --- |
| **Ressource** | **Collection** |
| skateboard | skateboards |
| client | clients |

Cette liste est un exemple et ne contient pas tous les exemples cités précédemment.

Bien ! Nous avons nos collections ainsi que les ressources correspondantes et leurs informations supplémentaires. Continuons avec notre exemple de boutique de skateboards et suivons ensemble le parcours de la requête d’un client via notre API.

Quand un client achète un skateboard en utilisant votre application web, cela donne :

* votre API envoie la requête du navigateur (le client) aux serveurs de l’application pour l’achat d’un skateboard ;
* la requête met à jour l’**inventaire** pour qu’il y ait un skateboard de moins ;
* la requête met à jour l’historique de commandes du **client** pour ajouter le skateboard à son historique d'achats.  😁

Super ! Vous savez maintenant comment et sous quelle forme stocker les données que vous voulez utiliser dans votre API via des ressources et des collections. Mais du coup, comment pouvez-vous y accéder ? Comment savoir où les récupérer, ces données ?

*URI et Endpoints*

Le **path (ou *chemin*)** que vous donnez à votre API lui permet de savoir exactement **où** se trouvent les données que vous voulez récupérer. Vous pouvez imaginer cela comme le fait de parcourir vos propres fichiers sur votre ordinateur. Vous devez aller de dossier en dossier pour trouver vos données, et chaque photo ou document que vous sauvegardez a son propre path, ou chemin de fichier, en français. Par exemple, votre photo de série préférée pourrait se trouver au bout de ce path :

MyComputer/Images/Series/gameofthrones.jpg

Les API REST stockent également les données de façon similaire, et un URI constitue le chemin pour y arriver.

Si une ressource est l’objet qui stocke vos données, pour les récupérer vous allez avoir besoin d’un identifiant de ressource uniforme, ou ***URI***pour ***U***niform***R***esource***I***dentifier. L’URI est le moyen d’identifier votre ressource, comme une étiquette.

Imaginons que vous créez une API pour un site web qui présenterait toutes les informations de Game of Thrones, que ce soit sur le livre ou la série. L’URI qui listerait tous les personnages pourrait être la suivante :

/characters

Si vous voulez voir les informations sur un seul personnage, qui porte l’ID 123, votre URI serait le suivant :

/characters/123

Tout comme les paths pour les fichiers, les URI peuvent avoir des ressources **imbriquées**. Si vous voulez obtenir uniquement le nomdu personnage qui vous intéresse, votre URI pourrait ressembler à ceci :

/characters/123/description

Wouhou ! Voilà du progrès ! Le souci, c’est que sans l’adresse réelle du site web, l’API ne saura pas du tout où chercher l’URI pour commencer ! C’est là que les **endpoints** (ou points de terminaison, en français) interviennent !

Un endpoint est une URL/URI qui fait partie d’une API. Si un URI est comme un chemin de fichier, alors un endpoint est comme l’adresse complète du fichier. Il vous suffit d’ajouter votre **nom de domaine** au début de votre URI, et vous avez un endpoint  ! Par exemple, si le nom de domaine de notre app est gameofthrones-informations.com, nous aurons :

https://gameofthrones-informations.com/characters

Houla, attends deux secondes, c’est quoi la différence entre URI et URL ?

Toutes les URL sont des URI, mais toutes les URI ne sont pas des URL 🤯. L’URI permet d’identifier une ressource tandis que l’URL permet de la localiser.

On confond souvent les deux. On va tout simplifier avec un exemple ! Reprenons notre site de Game of Thrones. Si le personnage de Jon Snow a pour ID 890, alors l’URI serait /characters/890. L’URL serait : https://gameofthrones-informations.com/characters/890

**L’URL de la requête** est l’endpoint complet que vous utilisez pour votre requête. Il associe le nom de domaine + le path de votre ressource. À présent, vous savez comment accéder aux données que vous souhaitez !

Au fait, pas besoin de créer votre propre API de GoT – il en existe déjà une et vous pouvez la découvrir ici : [**anapioficeandfire.com**](https://anapioficeandfire.com/)**.** ⚔️

*Distinguez XML et JSON*

Une fois que vous avez le bon endpoint sur lequel faire votre requête, il est temps pour vous d’obtenir vos données ! C’est là que vous obtenez les informations sur les ressources que vous avez créées.

Le terme données est un terme général qui décrit toute information envoyée ou reçue, tandis que que le terme ressource décrit plus précisément les**éléments** qui sont contenus dans cette information.

Les données des API REST peuvent utiliser deux langages : XML et JSON. Si une API renvoie un set de données en XML ou en JSON, le contenu restera le même, mais la forme change. Le format de données est différent.

*Le XML*

En**XML**, chaque élément de donnée a une balise ouvrante et une balise fermante qui peut également avoir des balises imbriquées :

<series>

<serie>

<titre>Game Of Thrones</titre>

<realisateur> Alan Taylor </realisateur>

</serie>

<serie>

<titre>Peaky Blinders</titre>

<realisateur> Otto Bathurst </realisateur> </serie>

</series>

Chaque **ressource** listée a la balise ouvrante  <serie>  et la balise fermante  </serie>  . Au sein de chaque ressource se trouvent davantage d’informations, comme "titre" et "réalisateur".

*Le JSON*

Le **JSON** stocke les données sous un format de clé-valeur avec comme clé le type de données, suivi de deux points **:**, suivi de la valeur de la donnée. Les données JSON sont entourées d'accolades { }, et chaque paire clé-valeur est envoyée comme chaîne de caractères avec des guillemets autour "".

Ce qui nous donne ceci :

{ "titre" :"Game of Thrones"}

Les tableaux, ou listes, en JSON sont entourées de crochets []. L’exemple ci-dessous montre comment une liste complète peut être considérée comme la valeur de la clé "series". Les mêmes données en XML ci-dessus seraient représentées ainsi en JSON :

{ "series":

[

{"titre": "Game Of Thrones",

"realisateur": "Alan Taylor" },

{ "titre": "Peaky Blinders",

"realisateur": "Otto Bathurst" }

]

}

Le JSON est généralement considéré comme :

1. Plus facile à analyser avec du code.
2. Plus court.
3. Plus rapide à lire et à écrire pour les machines.
4. Très "léger" et efficace grâce à sa structure en arborescence et sa syntaxe simple.

Voici quelques exemples d’API réelles qui renvoient du JSON et du XML :

* Penguin Random House : [XML](https://www.penguinrandomhouse.biz/webservices/rest/)
* Potter API : [JSON](https://github.com/Laboratoria/LIM011-data-lovers/blob/master/src/data/potter/potter.json)

Comme vous pouvez le constater, le JSON est le langage de données le plus utilisé ; c’est pour cette raison que nous l’utiliserons dans le reste de ce cours  !

*En résumé*

* Une ressource est un objet de type nominal utilisé pour sauvegarder des données dans une API.
* Une ressource peut contenir des informations supplémentaires.
* Les ressources sont regroupées en collection et sont nommées au pluriel.
* Vous pouvez accéder aux ressources dans les API avec des URI.
* Les données REST peuvent être en langage JSON ou XML, mais le JSON est le plus courant.

Et maintenant, revenons sur tout ce que nous avons appris avec le quiz de cette première partie.

Une fois que vous aurez terminé ce quiz, je vous retrouve dans la seconde partie, dans laquelle nous allons sauter dans le grand bain et utiliser une API !

# Formulez des requêtes et envoyez des réponses avec une API REST

## [Utilisez Postman pour formuler vos requêtes](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/7498761-utilisez-postman-pour-formuler-vos-requetes)

**Identifiez les avantages de Postman**

Vous savez déjà qu’une API REST implique l’envoi de **requêtes** du client à l’API, qui passe la requête au serveur, l’API récupère la **réponse** et la renvoie enfin au client. Dans ce chapitre, nous allons voir comment formuler ces requêtes grâce à [Postman](https://www.postman.com/).

Cette interface graphique est utilisée par de nombreux développeurs. Elle facilite la construction de nos requêtes. C’est donc l’outil idéal pour tester des API sans devoir utiliser de code.

Parce que cette interface offre beaucoup d’avantages :

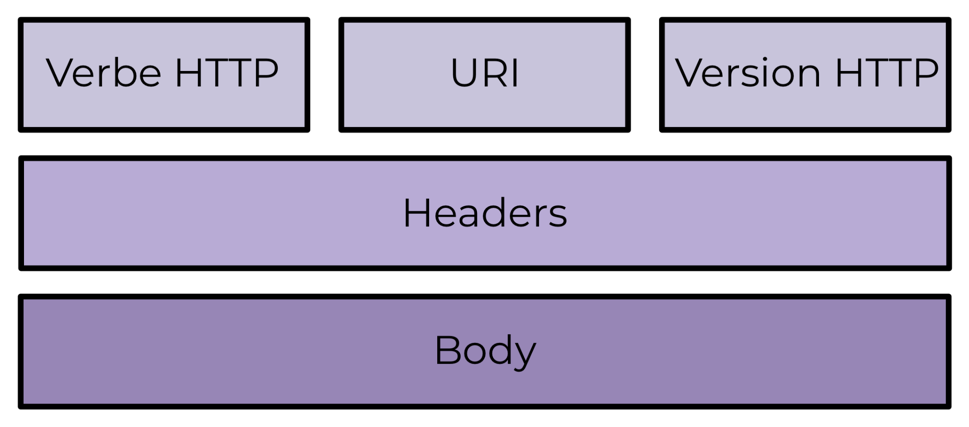
* Vous pouvez l’utiliser quel que soit le langage avec lequel vous programmez.
* Son interface utilisateur est simple : vous effectuez vos requêtes facilement.
* Vous n’avez pas besoin de savoir coder, ou d’utiliser une application.

*Formulez une requête sur Postman*

*La structure d’une requête*

Chaque requête a une structure spécifique qui a cette forme :

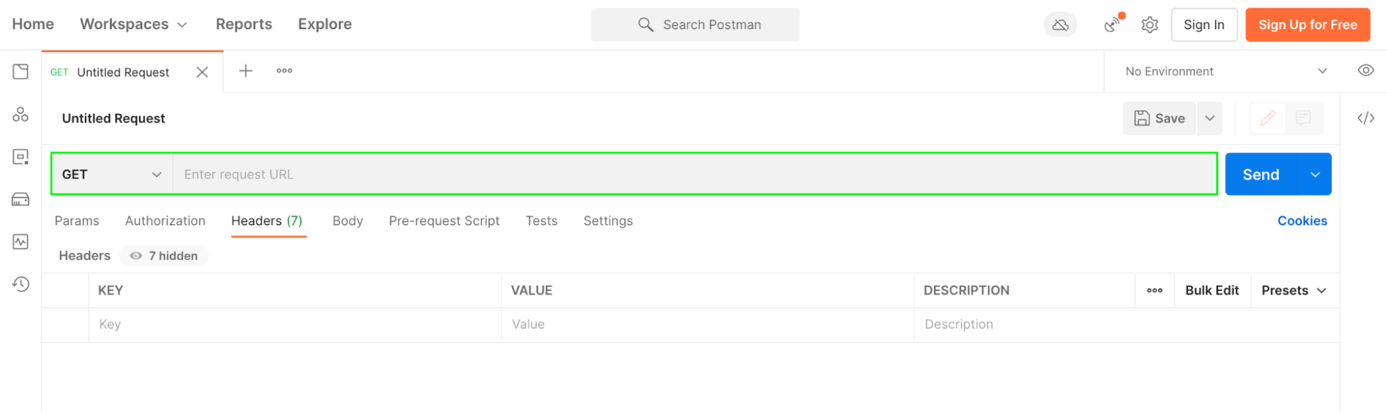
Verbe HTTP + URI + Version HTTP + Headers + Body (facultatif)

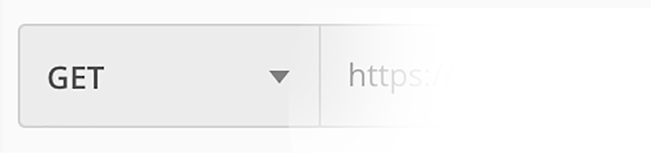


Une structure de requête typique

*Visualisez une requête sur Postman*

Vous devriez avoir cette vue :

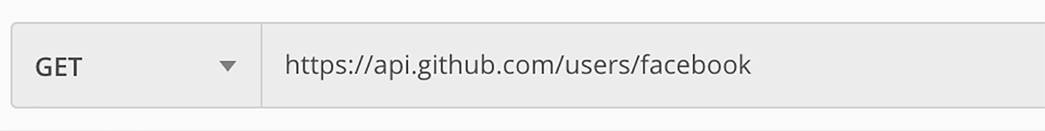




Exemple de verbe HTTP avec GET

*Le verbe*

Commençons avec le verbe. Les **verbes http** correspondent à différents types d’actions que vous pouvez accomplir avec votre requête. Ceux que vous rencontrerez le plus couramment sont **GET** (obtenir), **PUT** (mettre), **POST** (publier), et **DELETE** (supprimer). Ne vous y attardez pas trop pour le moment, nous les étudierons tous plus en détail plus tard !



*L’URL d’une requête complète comprend le nom de domaine : api.github.com, et l'URI (le chemin de la ressource) : /users/faceboook*

*L’URI*

Passons à l’**URI**. Un URI est le moyen d’identifier les ressources. Par exemple, si vous voulez voir tous les utilisateurs sur votre site web, le path serait le suivant :

/users

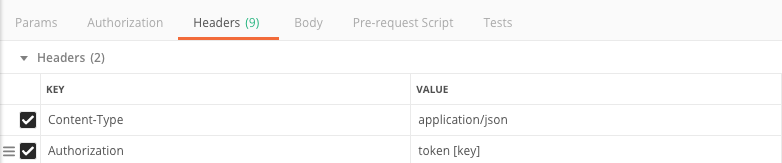
OK, mais imaginons que vous vouliez obtenir les informations d’un utilisateur spécifique. Dans ce cas-là, il vous faudrait préciser son ID. On obtiendrait quelque chose comme ceci :

users/:user\_id

Pourquoi on utilisait un nombre avant, et tout d’un coup tu nous mets  :user\_id ? 🤔

On utilise  :user\_id  pour matérialiser l’ID de l’utilisateur, c’est ce qu’on appelle un placeholder. En pratique, avec un ID réel, le path ressemblerait plutôt à ça :  users/145

*Le Header*



Headers dans une requête

Un **header** (ou ***en-tête***) vous permet de faire passer des informations supplémentaires sur le message. Par exemple :

* De quel langage s’agit-il ?
* À quelle date l’envoyez-vous ?
* Quel logiciel la personne utilise-t-elle ?
* Quelle est votre clé d’authentification ?

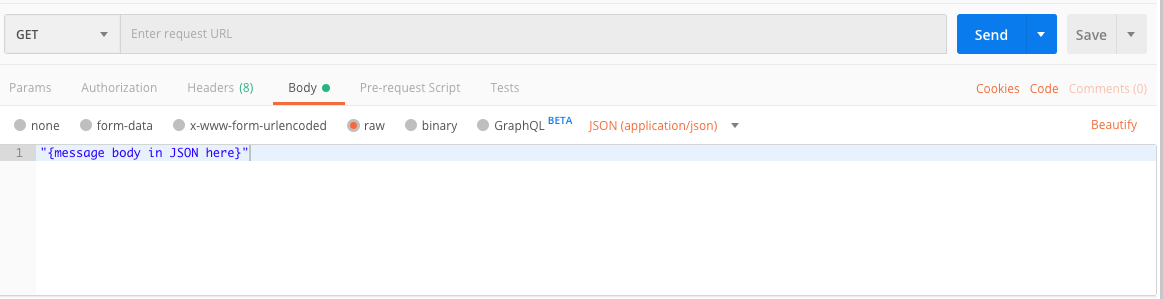
Les headers sont représentés par une paire **clé et valeur**, et il existe de nombreux types d’options différents pour eux. Par exemple :

Date : Mardi 19 Janvier 2019 18:15:41 GMT

Utilisateur-Agent : Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_5)

Vous pouvez consulter la [**liste complète**](https://developer.mozilla.org/fr/docs/Web/HTTP/Headers) des différentes options pour les  headers.

*Le body*



Le body (ou corps de message, en français)

Pour finir, parlons du **body** ! Pour formuler une requête, il n’est utilisé qu’avec **PUT** (mise à jour) ou **POST** (création). Il contient les données réelles de la ressource que vous essayez de créer ou de mettre à jour. Les données sont envoyées sous format JSON.

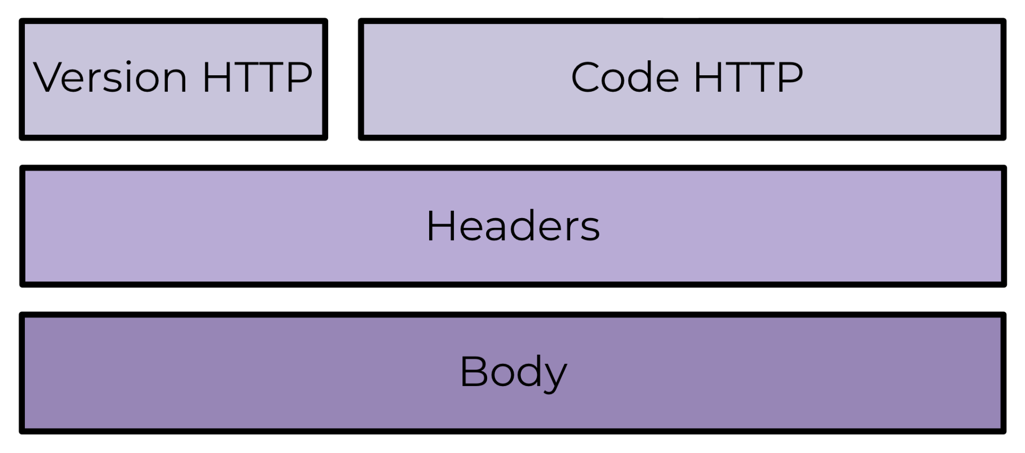
Petit rappel : le JSON est largement utilisé ; toutefois il se peut que certaines API n’acceptent que le XML. Vous trouverez cette information dans la documentation de l’API que vous utiliserez.

Notez que le body est facultatif dans ces deux cas. Cela signifie qu’il est tout à fait possible d’envoyer un body vide en fonction des actions de l’API visée.

*Obtenez une réponse avec Postman*

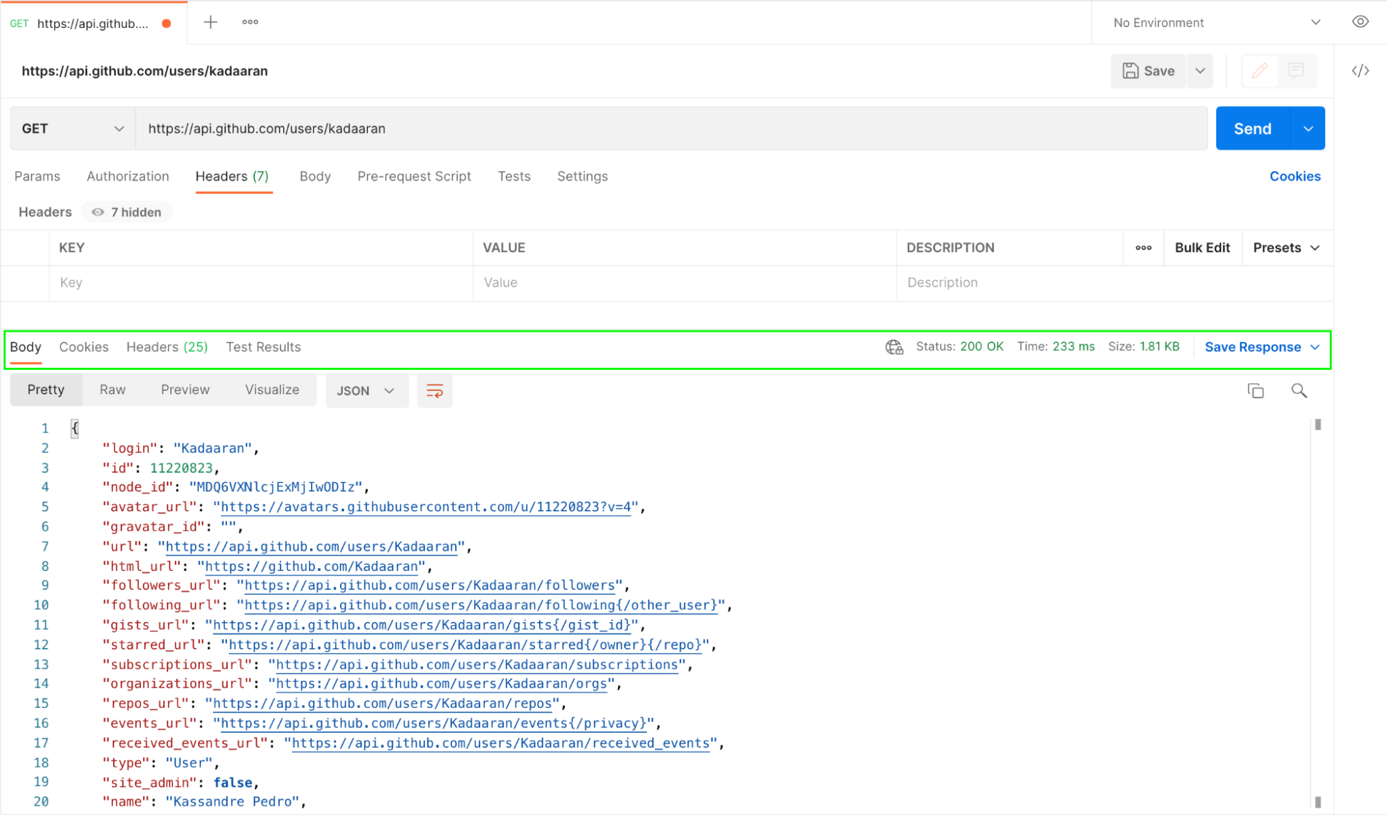
Le format du message de **réponse** est très similaire à celui de la requête :

Version HTTP + Code de réponse HTTP + Headers + Body



Une structure de réponse typique

Dans Postman, vous verrez un message de réponse comme celui-ci :



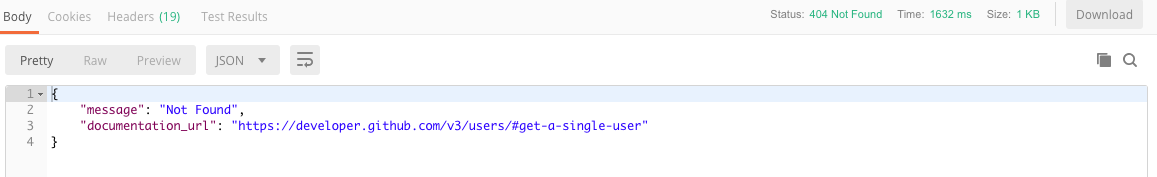
Un message de réponse typique

Attends deux secondes... Le body ? Mais il n'est pas censé être utilisé seulement pour faire une requête avec POST et PUT ? 🤔

Pour formuler une requête, oui ! Mais pour les réponses, le **body** contient l’information que vous avez demandée, et que l’API vous renvoie. Celle-ci est matérialisée au sein du body sous la forme d’un JSON ou en XML. L’image ci-dessus montre une réponse d’une requête réussie, faite à l’API GitHub.

Et si elle échoue ? Que se passe-t-il ? On obtient une information différente ? 🤔

Exactement ! Si la requête échoue, le **body**peut contenir un message d’erreur :

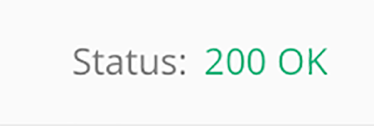


Une requête qui échoue

Mais le message ne suffit pas, et entre nous il peut arriver que des API n’envoient pas de messages du tout – que la requête soit un succès ou non. Cela peut arriver, même si c’est rare ! Dans ce cas, votre meilleur allié sera le **code de réponse HTTP** !

*Analysez le code de réponse HTTP*

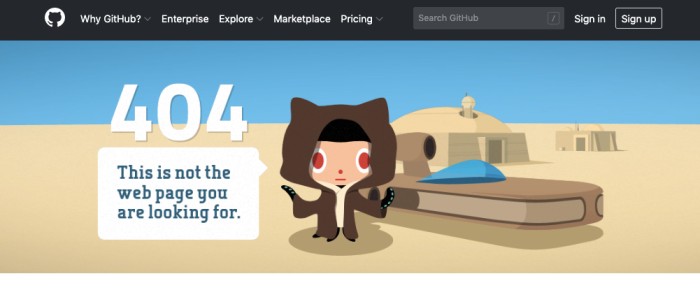
Le code de réponse HTTP aide le développeur et/ou le client à comprendre le **statut** de la réponse. Jetons un œil sur les exemples obtenus avec Postman :





Lorsqu’un client vous envoie une requête comme « Salut, pourriez-vous m’envoyer tous les tweets de cet utilisateur ? », vous pouvez vous représenter le code de réponse comme un feu de signalisation 🚥 qui vous dit par exemple :

Le **404 not found**(ou introuvable, en français) sur GitHub,:



Un autre code de réponse important à connaître est le**200 OK** – qui signifie que votre requête a réussi, et que votre réponse est prête ! En général, les règles de base pour les codes de réponse HTTP sont les suivantes :

* 100+ ➡ Information
* 200+ ➡ Succès
* 300+ ➡ Redirection
* 400+ ➡ Erreur client
* 500+ ➡ Erreur serveur

Si vous avez un doute sur un code HTTP, n’hésitez pas à consulter [**cette documentation**](https://developer.mozilla.org/fr/docs/Web/HTTP/Status) qui contient de plus amples informations et détails !

En résumé

* **Postman** est un logiciel gratuit qui vous permet d’effectuer des requêtes API sans coder.
* Les requêtes prennent la forme suivante :  
  Verbe HTTP + URI + Version HTTP + Headers + Body facultatif.
* Les verbes HTTP sont des types d’**actions** que l’on peut faire lors de la formulation d’une requête.
* Les réponses prennent la forme suivante :  
  Code de réponse HTTP + Version HTTP + Headers + Body.
* Les codes de réponse HTTP sont des sortes de**feux de signalisation** 🚦 avec des codes spécifiques, pour informer les clients si la requête est un succès ou un échec.
* Les codes HTTP sont codifiés en fonction du type de réponse ; vous trouverez la liste [ici](https://developer.mozilla.org/fr/docs/Web/HTTP/Status).

## [Réalisez vos premières requêtes sur une API](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6818136-realisez-vos-premieres-requetes-sur-une-api)

Dans le chapitre précédent, nous avons parlé des verbes HTTP et de la façon dont ils permettent de réaliser des **actions**spécifiques lors de la formulation d’une requête API. Rappelez-vous, j’avais mentionné GET, POST, PUT et DELETE

*Découvrez le CRUD*

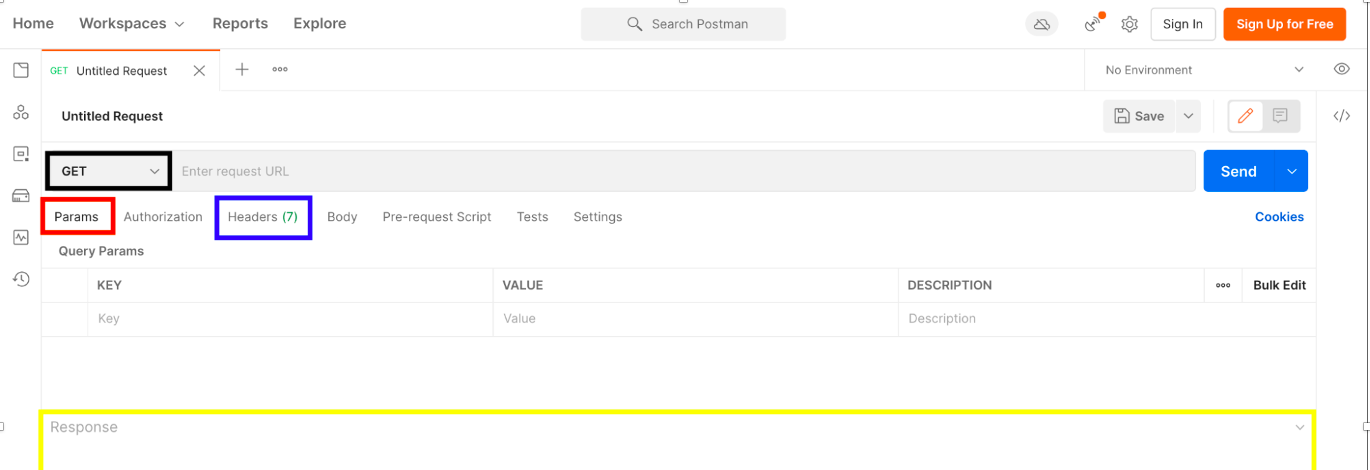
Le CRUD est la liste des actions de base que vous pouvez effectuer sur une ressource. C’est un acronyme qui signifie ***C***reate (créer), ***R***ead (lire), ***U***pdate (mettre à jour), et ***D***elete (supprimer). Bien que le CRUD ne constitue pas vraiment un mécanisme technique en soi, chaque **action CRUD** est associée à un **verbe HTTP**. Voici la cartographie :

|  |  |
| --- | --- |
| **Action CRUD** | **Verbe HTTP associé** |
| Create (Créer) | POST (Publier) |
| Read (Lire) | GET (Obtenir) |
| Update (Mettre à jour) | PUT (Mettre) |
| Delete (Supprimer) | DELETE (Supprimer) |

*Obtenez des résultats avec votre première requête GET*

Maintenant que nous avons vu tout le contexte qui se cache derrière une API, il est temps de pratiquer ! 🏋️‍♀️ Utilisons une API pour obtenir des données. Nous utiliserons le verbe HTTP **GET** et l’[API GitHub](https://developer.github.com/v3/)(de [GitHub](https://github.com/)) pour obtenir des données sur un utilisateur GitHub spécifique.

Nous l’avions survolé lors du dernier chapitre, mais il est toujours bon de revoir ce que nous avons appris. Détaillons un peu ce que nous voyons, de haut en bas ! 🕵️



Interface utilisateur Postman

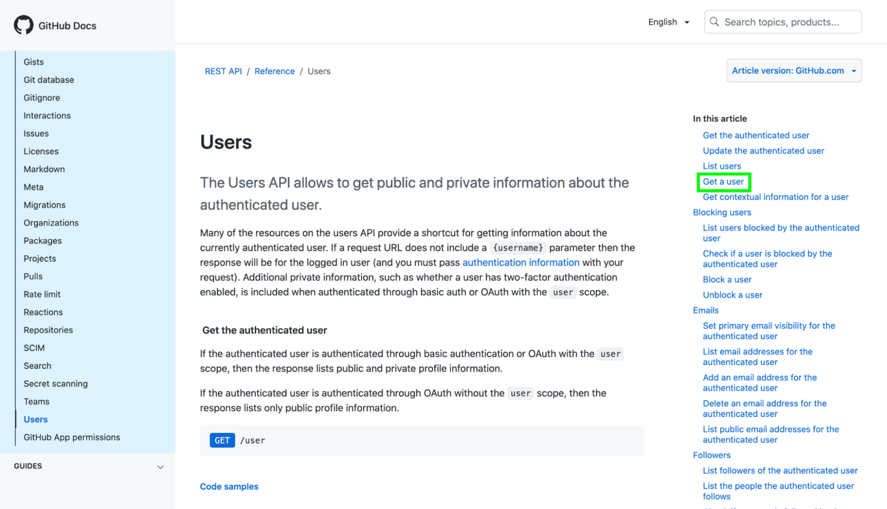
* La première ligne (encadrée en noir) vous permet de sélectionner votre type de requête dans le menu déroulant (dans notre cas, ce sera GET) 🤫.
* À côté, vous pouvez remplir la case avec l’URL complète de votre requête.
* Il y a un petit bouton (encadré en rouge) nommé Params. Si vous cliquez dessus, vous aurez un emplacement pour définir les valeurs clés de vos paramètres.
* À droite  (encadré en bleu), vous pouvez cliquer sur **Headers**. Cela vous permettra de définir vos headers de requête.
* Et pour finir, en dessous, en jaune , vous pouvez voir l’emplacement du body de votre réponse.

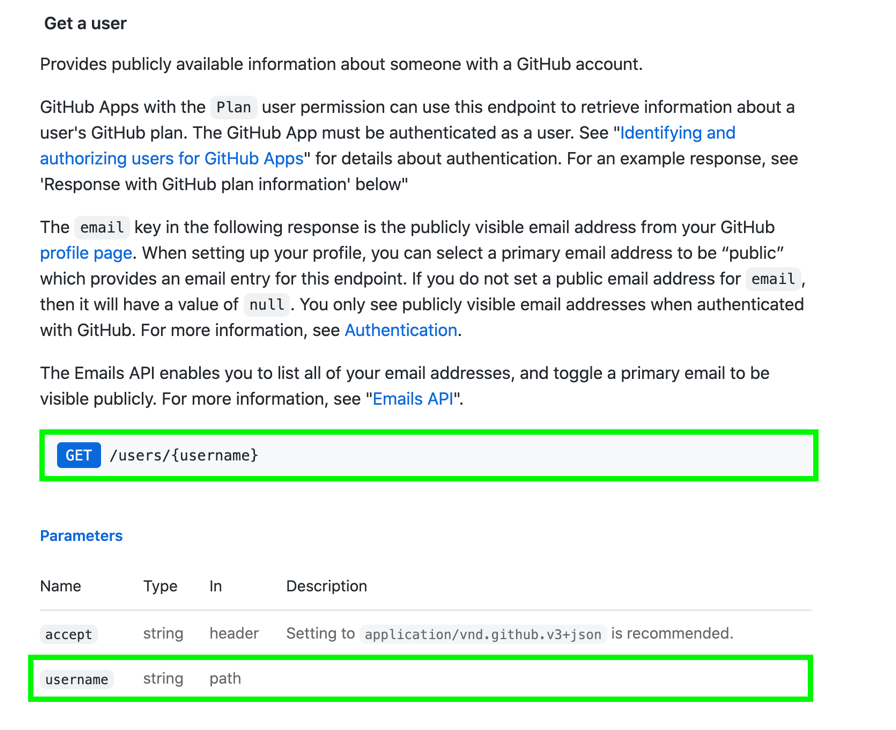
Nous voulons obtenir des informations sur un utilisateur. Mais comment faire ? Quelle URL utiliser ? 🤔

Avant de  faire une requête sur l’API GitHub et d'obtenir un utilisateur en particulier, vous devez avant tout faire une chose très importante : consulter la [documentation GitHub](https://developer.github.com/v3) ! Et plus précisément la section [Users](https://developer.github.com/v3/users/), car c’est celle-ci qui nous intéresse. La documentation, c’est le mode d’emploi d’une API. C’est ainsi que vous trouverez les ressources, URI et endpoints que vous pouvez utiliser pour récupérer des données.

Allez sur la section Users de l’API GitHub via cette URL : <https://developer.github.com/v3/users/>

La partie qui nous intéresse ici est celle qui nous permet d’obtenir **un seul utilisateur**(Get a user en anglais). Cliquez dessus !



Accédez à la documentation GitHub et cliquez sur Get a user

Le texte correspondant à la documentation sur l'utilisateur unique

Voici ce que dit la documentation en français :

***Obtenir un utilisateur unique***

*Fournit les informations disponibles publiquement sur quelqu’un ayant un compte GitHub.*

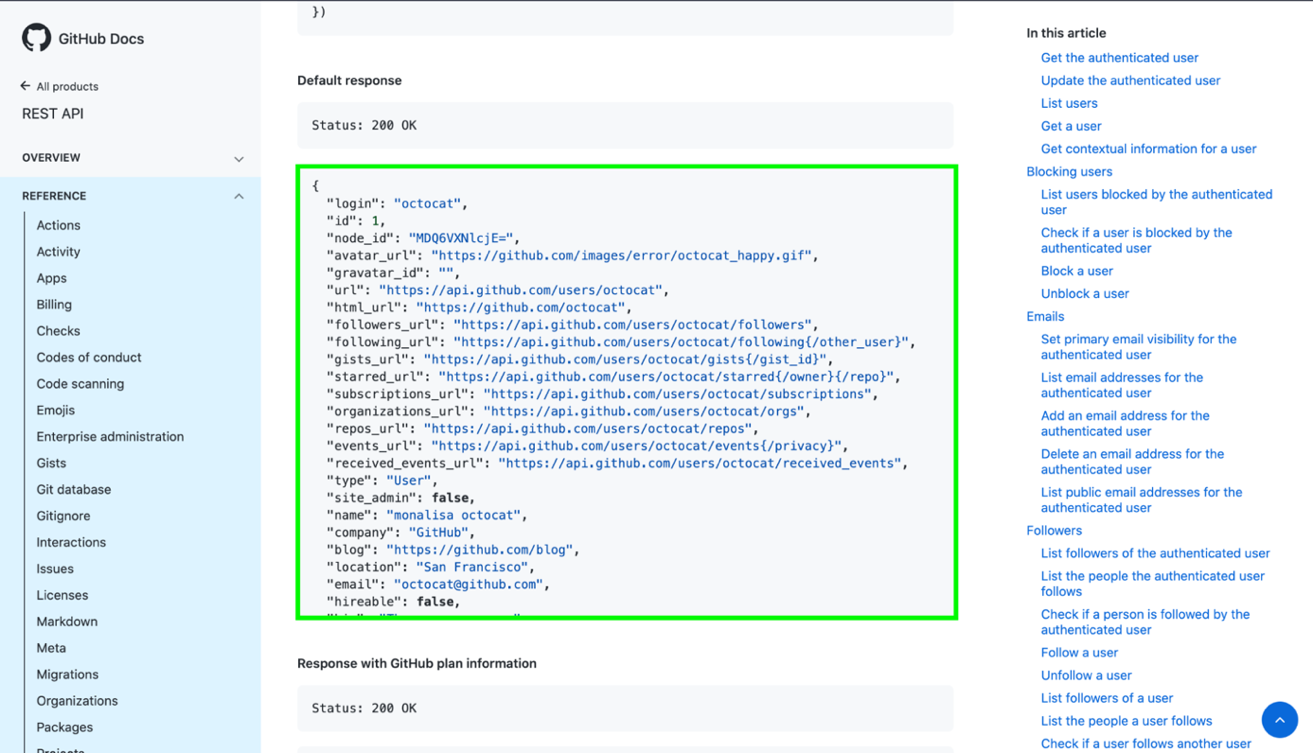
*Les applications GitHub avec la permission utilisateur Plan peuvent utiliser cet endpoint pour récupérer des informations sur le plan GitHub d’un utilisateur. L’application GitHub doit être authentifiée en tant qu’utilisateur. Voir «Identifier et autoriser les utilisateurs pour les applications GitHub» pour plus de détails sur l’authentification. Pour un exemple de réponse, voir « Réponse avec l’information du plan GitHub ».*

*La clé e-mail dans la réponse ci-dessous correspond à votre adresse e-mail visible publiquement depuis votre page de profil GitHub. Lors de la création de votre profil, vous pouvez sélectionner une adresse e-mail principale comme « publique », ce qui fournit une entrée e-mail pour cet endpoint. Si vous ne choisissez pas d’adresse e-mail publique pour e-mail, alors sa valeur sera nulle. Vous ne voyez que les adresses e-mail visibles publiquement lorsque vous êtes authentifié dans GitHub. Pour plus d’informations, voir Authentification.*

*L’API Emails vous permet de lister toutes vos adresses e-mail, et d’ajouter un toggle à une adresse e-mail principale pour qu’elle soit visible publiquement. Pour plus d’informations, voir « API Emails ».*

La documentation nous apprend plusieurs choses :

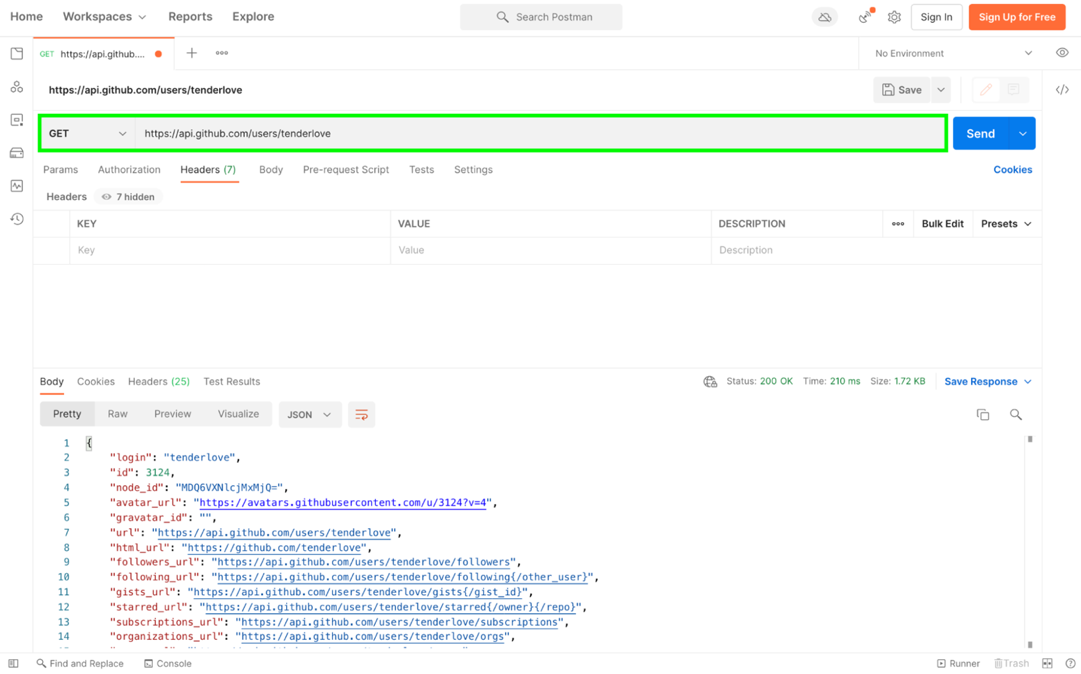
1. Cette section fournit les informations disponibles publiquement sur quelqu’un ayant un compte GitHub.
2. On peut accéder aux informations d’un utilisateur via **GET /users/:username**.
3. Un exemple de réponse :



Un exemple de réponse

Parfait ! 🙌

Si on résume, cela signifie que, pour obtenir la donnée user de l’utilisateur “tenderlove” (oui oui c’est un vrai login, cet homme existe et est d’ailleurs un des contributeurs de [Ruby on Rails](https://rubyonrails.org/) 👌), vous allez dans Postman et entrez <https://api.github.com/users/tenderlove> dans l’URL, puis appuyez sur Send (Envoyer). Sur GitHub : <https://github.com/tenderlove> .



Requête GET

Et voilà ! Vous avez effectué votre première requête GET avec succès !

Vous pouvez remplacer tenderlove par votre login GitHub et observer la réponse. 😉 N’hésitez pas à comparer les données récupérées par l’API et celles de GitHub : vous verrez, ce sont les mêmes !

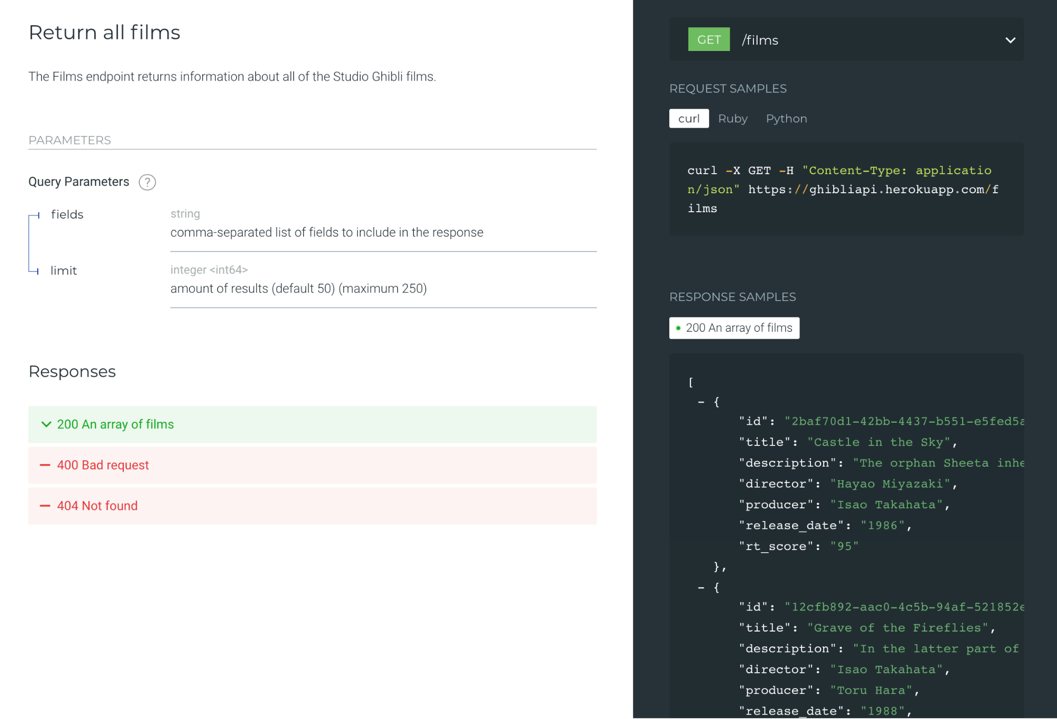
Vous pouvez même effectuer des requêtes GET directement depuis votre navigateur, car les endpoints REST utilisent le même protocole HTTP que le web. Essayez d’aller sur [**https://api.github.com/users/tenderlove**](https://api.github.com/users/tenderlove) maintenant, et vous verrez  !

**Utilisez la documentation pour connaître le mode d’emploi de l’API**

Comme vous pouvez le constater, la documentation d’une API ressemble à un manuel d’utilisation très très détaillé. Étant donné que chaque API est différente, vous ne sauriez pas les utiliser sans une documentation claire et précise.

Quelles informations nous donne cette documentation ? ✋

La documentation liste tous les appels API possibles, les requêtes et réponses typiques, mais surtout les verbes à utiliser pour chaque requête. Cette [API Ghibli](https://ghibliapi.vercel.app/) constitue un bon exemple (studio de mon voisin Toroto, Château dans le ciel, etc..)



Documentation de l’API Ghibli

Dans cet exemple, pour récupérer tous les films Ghibli, vous voyez qu’il vous suffit de faire une requête GET sur <https://ghibliapi.vercel.app/#tag/Films>. Vous avez aussi une indication sur les messages d’erreur que renvoie l’API en cas de mauvais format de la requête (400) ou d’absence de ressources trouvées (404). Vous voyez aussi qu’en cas de succès, vous obtiendrez une réponse sous forme d’un **array** (ou tableau) qui contiendra la liste des films Ghibli. Un exemple de cette donnée se trouve sur la droite dans l’encadré.

Chaque fois que vous voulez utiliser une API, commencez par consulter la documentation.

**En résumé**

* CRUD signifie *create (créer), read (lire), update (mettre à jour) et delete (supprimer)*.
* GET est le verbe HTTP pour obtenir des données, et il est généralement utilisé avec un ID pour obtenir une donnée spécifique.
* Les applications utilisent GET pour présenter des informations sur des pages web.
* Utilisez Postman pour tester les API.
* La documentation est le manuel d’utilisation d’une API.
* La documentation vous permet de trouver la liste des endpoints accompagnée du verbe HTTP correspondant.

*Que ce soit à des fins personnelles ou professionnelles, utiliser une API c’est utiliser des données. Ces données sont parfois sensibles et il est important de les sécuriser. Abordons à présent un élément que je considère comme****ultra important****: la sécurité des données et l’authentification.*

## [Authentifiez une API pour plus de sécurité](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6818386-authentifiez-une-api-pour-plus-de-securite)

## [Entraînez-vous avec l’API GitHub](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6820251-entrainez-vous-avec-l'api-github)

## [Quiz : Servez-vous des API REST pour vos projets de code](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/exercises/4278)

# Concevez des API REST

## [Définissez la structure de votre API REST](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6824631-definissez-la-structure-de-votre-api-rest)

## [Concevez les endpoints de votre API](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6824856-concevez-les-endpoints-de-votre-api)

## [Utilisez les fonctionnalités avancées des endpoints](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6825136-utilisez-les-fonctionnalites-avancees-des-endpoints)

## [Choisissez des frameworks pour construire votre API](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6825316-choisissez-des-frameworks-pour-construire-votre-api)

## [Résumé du cours](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/6825441-resume-du-cours)

## [Quiz : Concevez des API REST](https://openclassrooms.com/fr/courses/6573181-adoptez-les-api-rest-pour-vos-projets-web/exercises/4279)