**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 3](#_Toc205119079)

[Tirez un maximum de ce cours 3](#_Toc205119080)

[Initiez-vous au fonctionnement des API 3](#_Toc205119081)

[Identifiez les avantages d’une API REST 6](#_Toc205119082)

[Utilisez les ressources et collections REST 9](#_Toc205119083)

[Appréhendez les données REST via l’utilisation des ressources 9](#_Toc205119084)

[Formulez des requêtes et envoyez des réponses avec une API REST 13](#_Toc205119085)

[Utilisez Postman pour formuler vos requêtes 13](#_Toc205119086)

[Réalisez vos premières requêtes sur une API 18](#_Toc205119087)

[Authentifiez une API pour plus de sécurité 23](#_Toc205119088)

[Comprenez l’importance de l’authentification pour une API 23](#_Toc205119089)

[Entraînez-vous avec l’API GitHub 29](#_Toc205119090)

[Concevez des API REST 37](#_Toc205119091)

[Définissez la structure de votre API REST 37](#_Toc205119092)

[Concevez les endpoints de votre API 42](#_Toc205119093)

[Utilisez les fonctionnalités avancées des endpoints 46](#_Toc205119094)

[Choisissez des frameworks pour construire votre API 50](#_Toc205119095)

[***En résumé*** 52](#_Toc205119096)

[Résumé du cours 53](#_Toc205119097)

**Références et outils.**

* Test et construction API : [Postman](https://www.postman.com/)
* EDI & Frameworks :
  + Liste [ici](#_Choisissez_des_frameworks) dans le doc
  + [Express](https://expressjs.com/)  JavaScript), ([Ruby on Rails](https://rubyonrails.org/) est basé sur Ruby), [Django](https://www.djangoproject.com/) (Python). [framework REST de Django](https://www.django-rest-framework.org/), [Flask](http://flask.pocoo.org/) (Python), [Spring](https://spring.io/projects/spring-boot) (framework web – Java), , [AWS API Gateway](https://aws.amazon.com/api-gateway/) et [AWS Lambda](https://aws.amazon.com/lambda/) (créer et d’utiliser des API REST en utilisant principalement une interface utilisateur (donc moins de code !)).

**Plus loin :**

* le cours [**Mettez en place une API avec Django REST framework**](https://openclassrooms.com/fr/courses/7192416-mettez-en-place-une-api-avec-django-rest-framework).
* suivre la [**partie 3**](https://openclassrooms.com/fr/courses/6900101-creez-une-application-java-avec-spring-boot/7078007-creez-lapi-avec-les-bons-starters) du cours Créez une application Java avec Spring Boot. (plein de prérequis)
* Sky is the limit, if you want to reach the sky f\*\*k a a duck and try to fly jusqu’à l’infini et au-delà.

# 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**

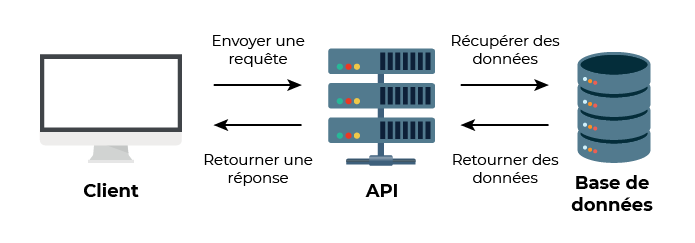
Vous trouvez plusieurs offres de trains pour Lisbonne en passant par l’Espagne. Et tout ça sans avoir à aller chercher l’information sur le site de chaque compagnie ferroviaire ou aérienne. Pas mal, non ?

 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 :

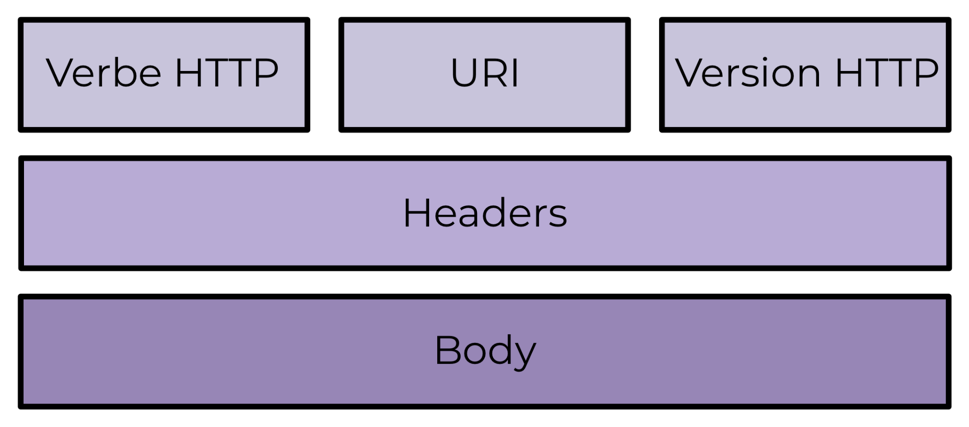
* Vous pouvez l’utiliser quel que soit le langage avec lequel vous programmez.
* Son interface utilisateur est simple : vous effectuez vos requêtes facilement.
* Vous n’avez pas besoin de savoir coder, ou d’utiliser une application.

*Formulez une requête sur Postman*

*La structure d’une requête*

Chaque requête a une structure spécifique qui a cette forme :

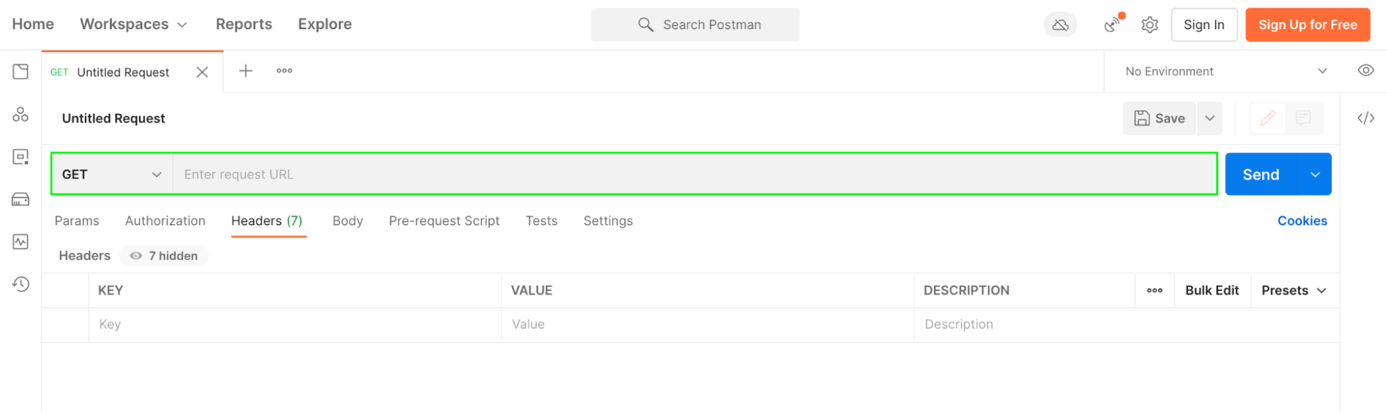
Verbe HTTP + URI + Version HTTP + Headers + Body (facultatif)

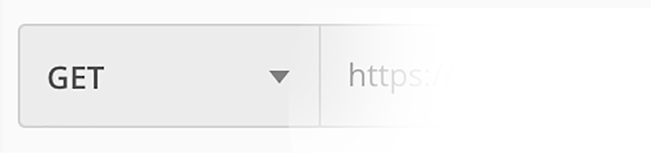


Une structure de requête typique

*Visualisez une requête sur Postman*

Vous devriez avoir cette vue :

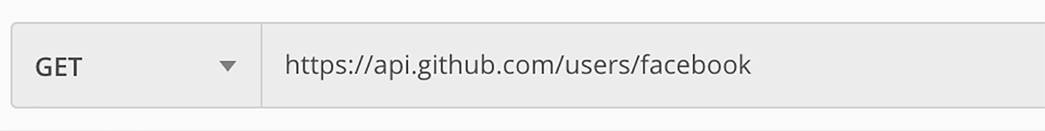




Exemple de verbe HTTP avec GET

*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

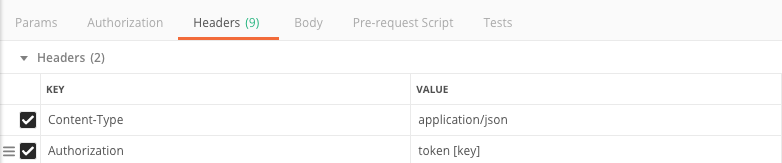
OK, mais imaginons que vous vouliez obtenir les informations d’un utilisateur spécifique. Dans ce cas-là, il vous faudrait préciser son ID. On obtiendrait quelque chose comme ceci :

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 ?

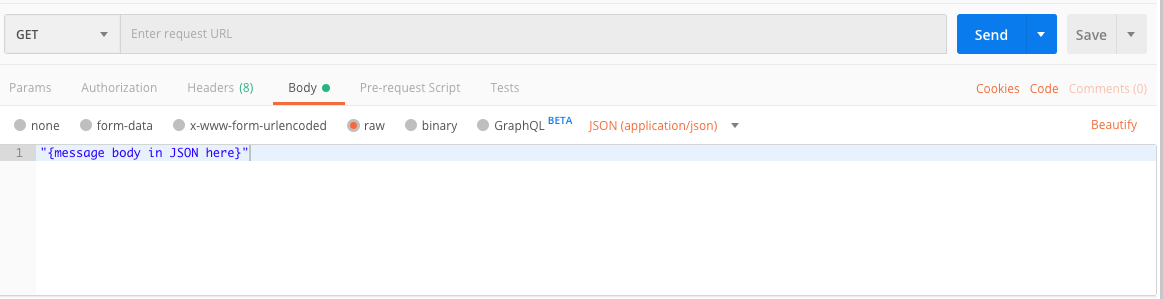
Les headers sont représentés par une paire **clé et valeur**, et il existe de nombreux types d’options différents pour eux. Par exemple :

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.

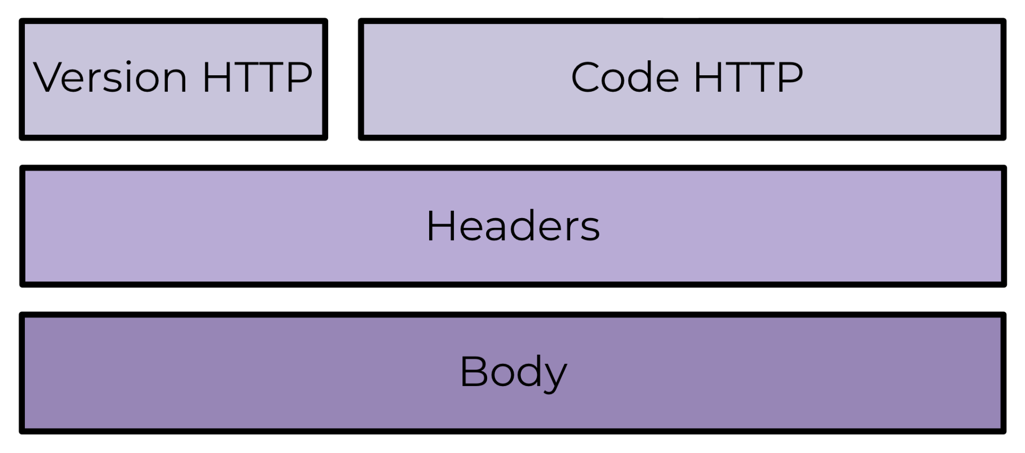
Petit rappel : le JSON est largement utilisé ; toutefois il se peut que certaines API n’acceptent que le XML. Vous trouverez cette information dans la documentation de l’API que vous utiliserez.

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*

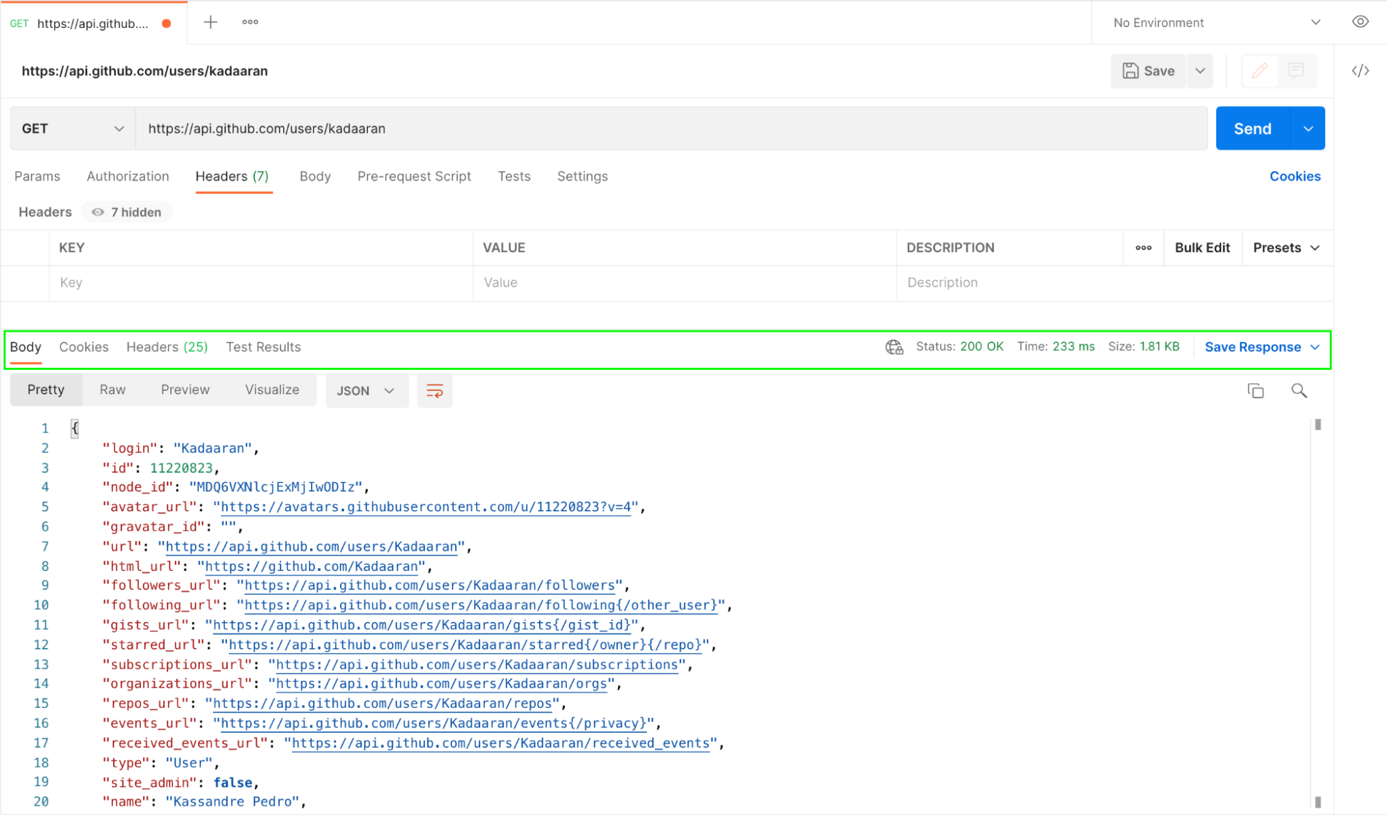
Le format du message de **réponse** est très similaire à celui de la requête :

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 :

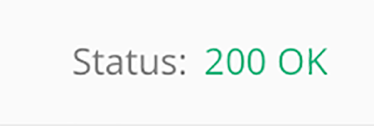


Une requête qui échoue

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*

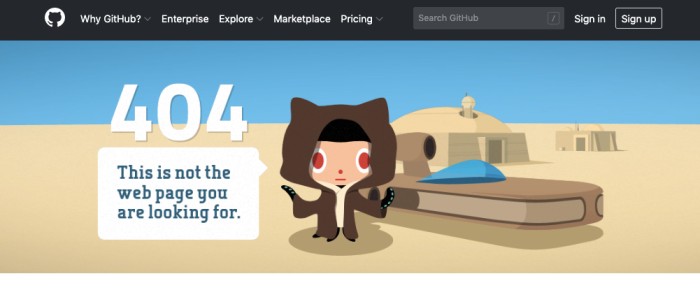
Le code de réponse HTTP aide le développeur et/ou le client à comprendre le **statut** de la réponse. Jetons un œil sur les exemples obtenus avec Postman :





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*

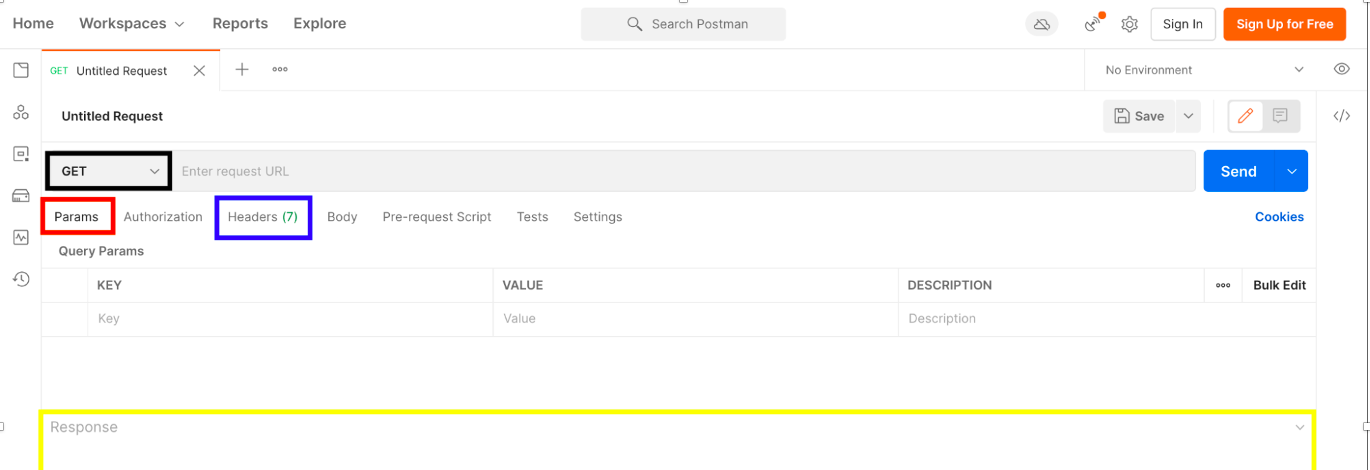
Le CRUD est la liste des actions de base que vous pouvez effectuer sur une ressource. C’est un acronyme qui signifie ***C***reate (créer), ***R***ead (lire), ***U***pdate (mettre à jour), et ***D***elete (supprimer). Bien que le CRUD ne constitue pas vraiment un mécanisme technique en soi, chaque **action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

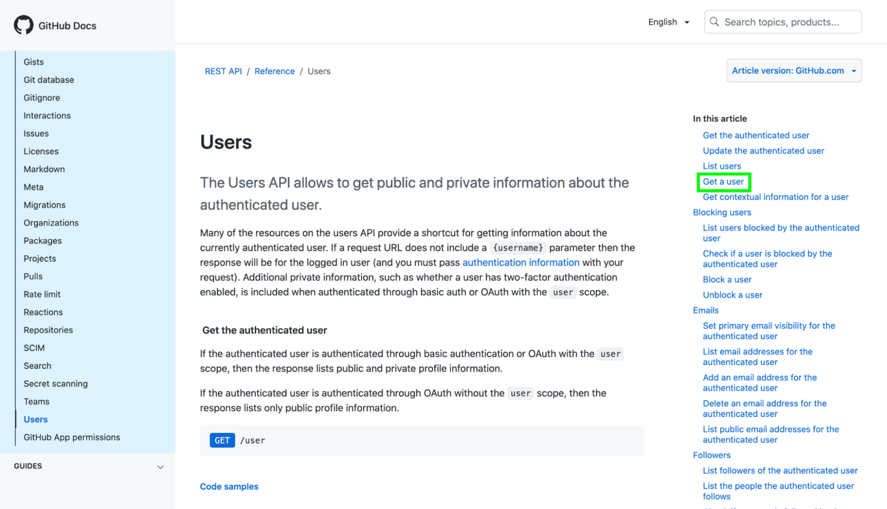
* La première ligne (encadrée en noir) vous permet de sélectionner votre type de requête dans le menu déroulant (dans notre cas, ce sera GET) 🤫.
* À côté, vous pouvez remplir la case avec l’URL complète de votre requête.
* Il y a un petit bouton (encadré en rouge) nommé Params. Si vous cliquez dessus, 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.

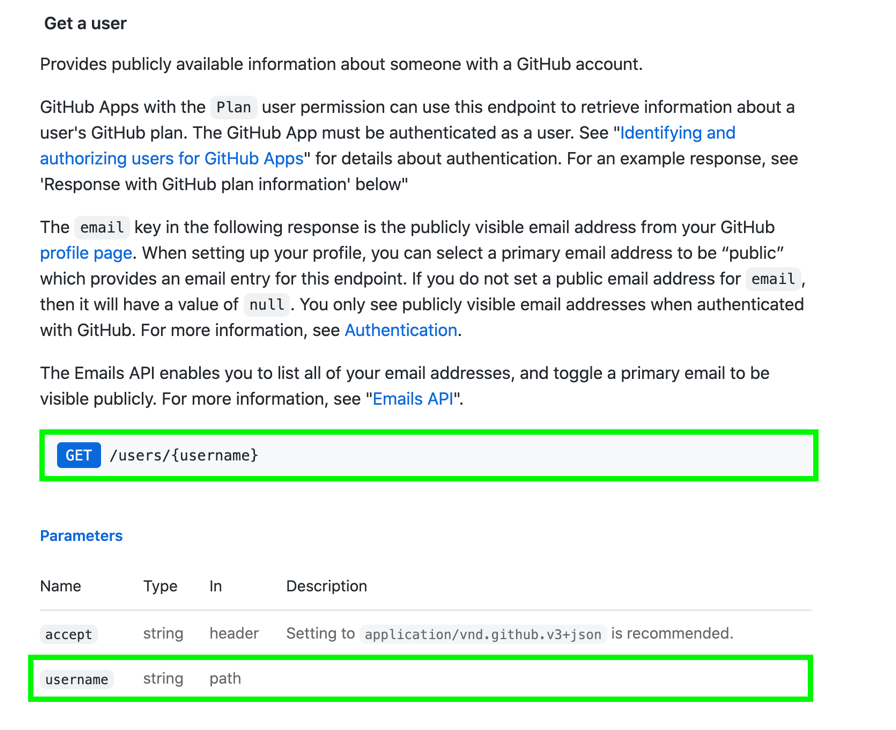
Nous voulons obtenir des informations sur un utilisateur. Mais comment faire ? Quelle URL utiliser ? 🤔

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.*

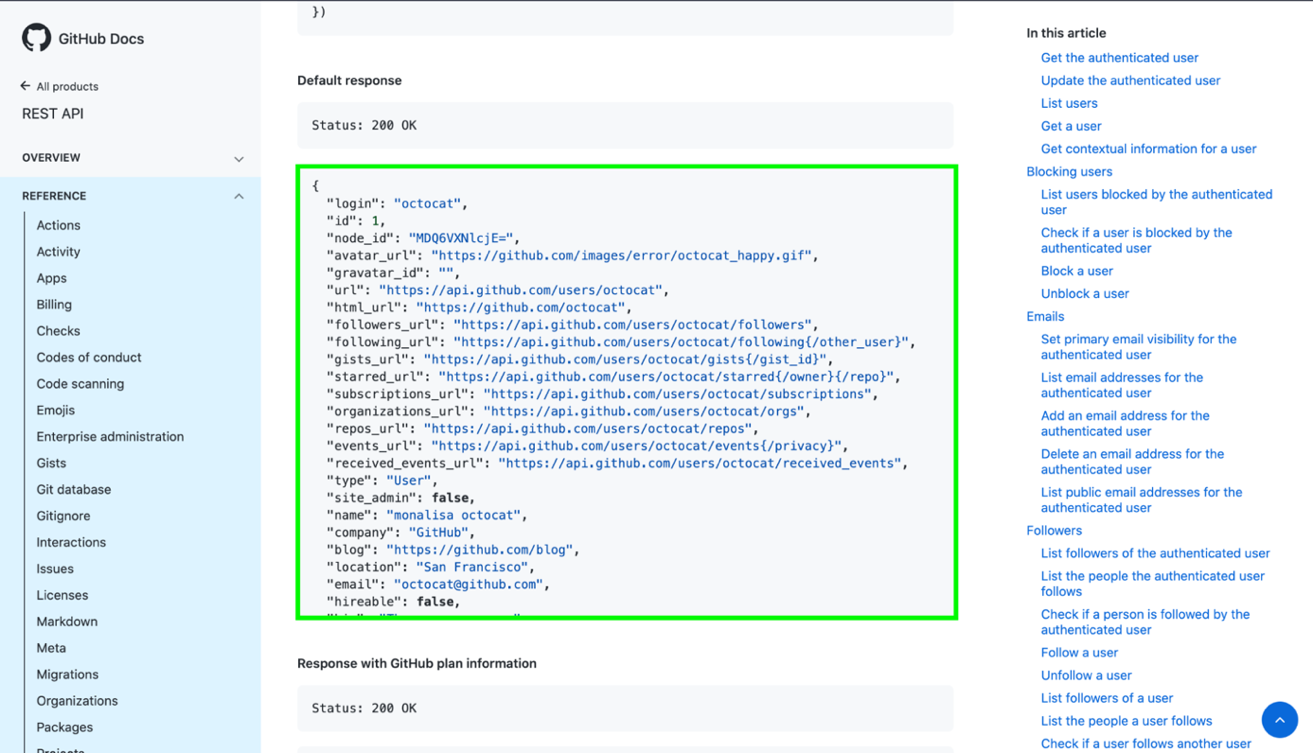
*Les applications GitHub avec la permission utilisateur Plan peuvent utiliser cet endpoint pour récupérer des informations sur le plan GitHub d’un utilisateur. L’application GitHub doit être authentifiée en tant qu’utilisateur. Voir «Identifier et autoriser les utilisateurs pour les applications GitHub» 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 :

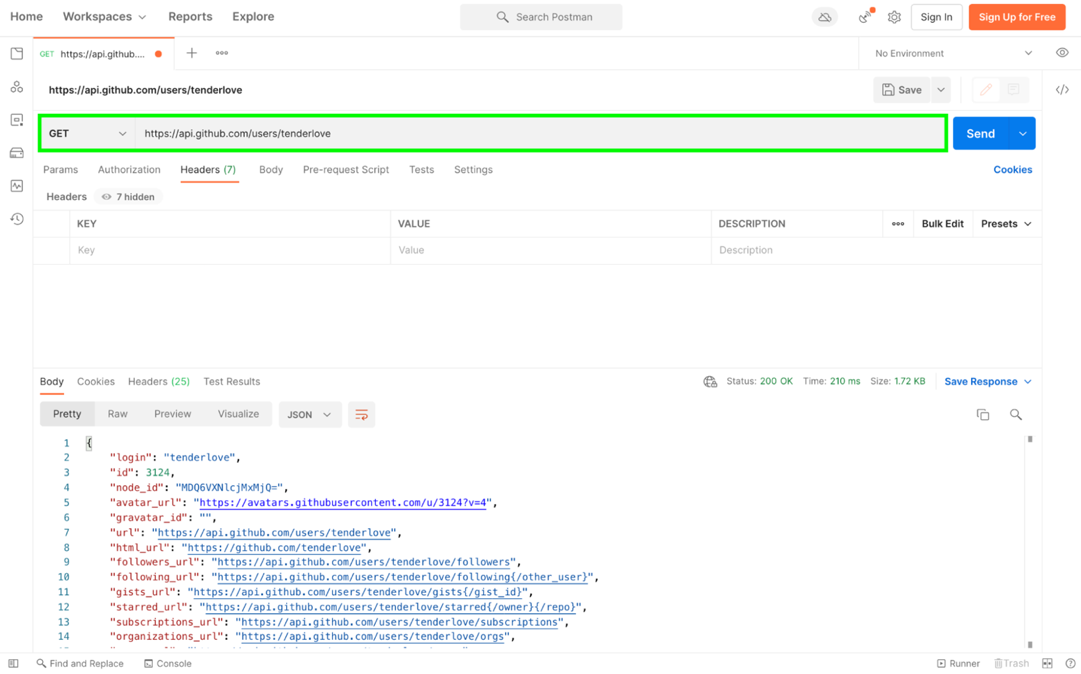
1. Cette section fournit les informations disponibles publiquement sur quelqu’un ayant un compte GitHub.
2. On peut accéder aux informations d’un utilisateur via **GET /users/:username**.
3. Un exemple de réponse :



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 !

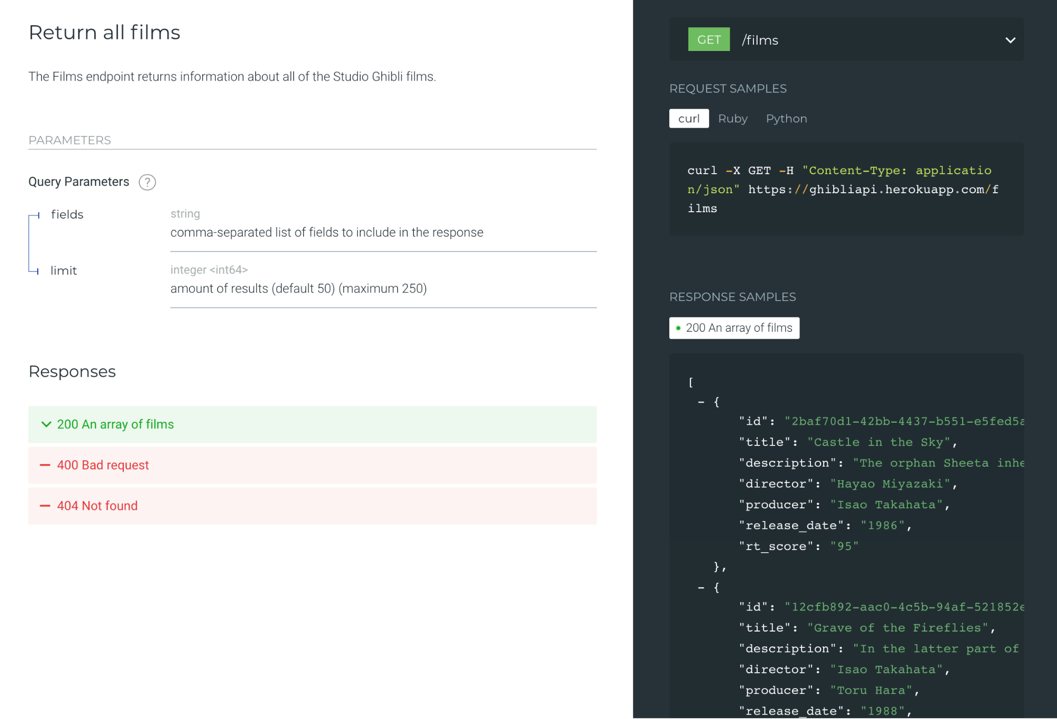
Vous pouvez même effectuer des requêtes GET directement depuis votre navigateur, car les endpoints REST utilisent le même protocole HTTP que le web. Essayez d’aller sur [**https://api.github.com/users/tenderlove**](https://api.github.com/users/tenderlove) maintenant, et vous verrez  !

**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)

### Comprenez l’importance de l’authentification pour une API

Avant d’aborder d’autres verbes HTTP, il est important que nous parlions davantage de l’**authentification**. L’authentification constitue simplement un moyen pour les API de garantir que le client a les **autorisations** nécessaires pour accéder aux données et les manipuler. 🔐 Si je crée une API, je ne veux pas que n’importe qui puisse changer le mot de passe de n’importe quel utilisateur, n’est-ce pas ? 🔥

Lorsque nous avions fait notre requête **GET** avec l’API GitHub, nous n’avions pas besoin d’authentification pour obtenir des données utilisateurs. Pourquoi ? Rappelez-vous : il existe des API privées... et… des API publiques !

Ah, GitHub est une API publique !

Oui et non. GitHub est une API qui possède **une partie publique** qui vous permet de faire des requêtes sans autorisation, comme nous l’avons fait dans le chapitre précédent. Cependant, **toute l’application n’est pas publique**. Une partie des endpoints nécessite une authentification, afin d’avoir les autorisations nécessaires pour mettre à jour des données.

Si vous voulez modifier, ajouter ou supprimer des données, GitHub doit vous donner l’autorisation de le faire. Vous ne voulez pas que n’importe qui puisse remplacer votre photo de profil par celle de Homer Simpson, duh ! 🍩 Donc, pour que ce soit possible, GitHub doit disposer d’un processus d’**authentification**.

Une des méthodes d’authentification les plus utilisées consiste à exiger qu’un développeur s’inscrive par le site web de l’API pour obtenir **un token ou *clé***. Une fois le token obtenu, le développeur l’utilise dans sa requête pour s’identifier, et voilà. 🎉

C’est quoi un token, au juste ? Une sorte de code ? 🤔

Eh bien, en quelque sorte ! Un token est généralement une chaîne longue et unique de lettres et de chiffres aléatoires que l’on assigne à un utilisateur. Le token est un peu comme un numéro de passeport : il est unique et permet de vous identifier. L’API peut donc l’utiliser pour savoir **qui** effectue la requête, et surtout de quel niveau d’**autorisation** cette personne dispose.

Les autorisations peuvent décrire des accès spécifiques à certaines fonctionnalités, comme le nombre de requêtes que l’on peut envoyer mais aussi quelles actions on peut effectuer (si on est un administrateur ou non, par exemple) ... La documentation de l’API se doit de fournir toutes les informations sur les fonctionnalités accessibles à travers un token d’authentification.

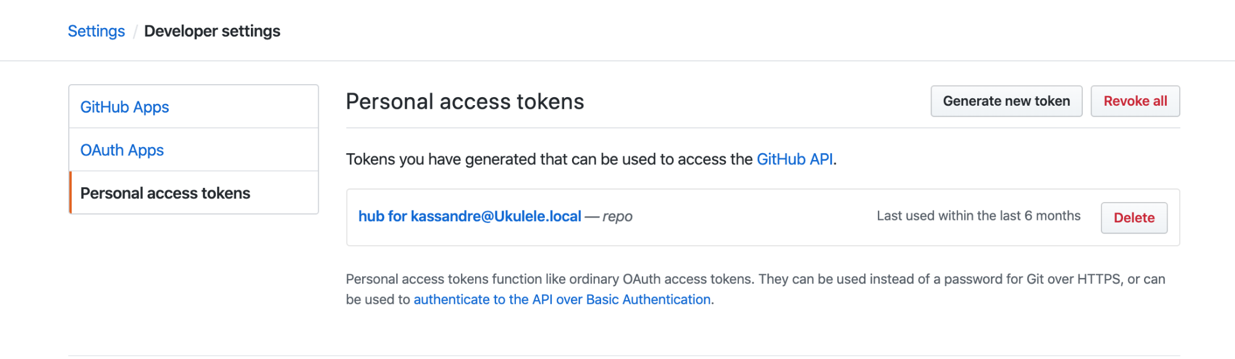
Comment ajouter le token à la requête ? 🤔

On l’envoie soit dans les **paramètres du header**, soit dans l’endpoint lui-même.

Pour illustrer ce qu’on vient de voir, nous allons regarder ensemble comment obtenir un token GitHub pas à pas !

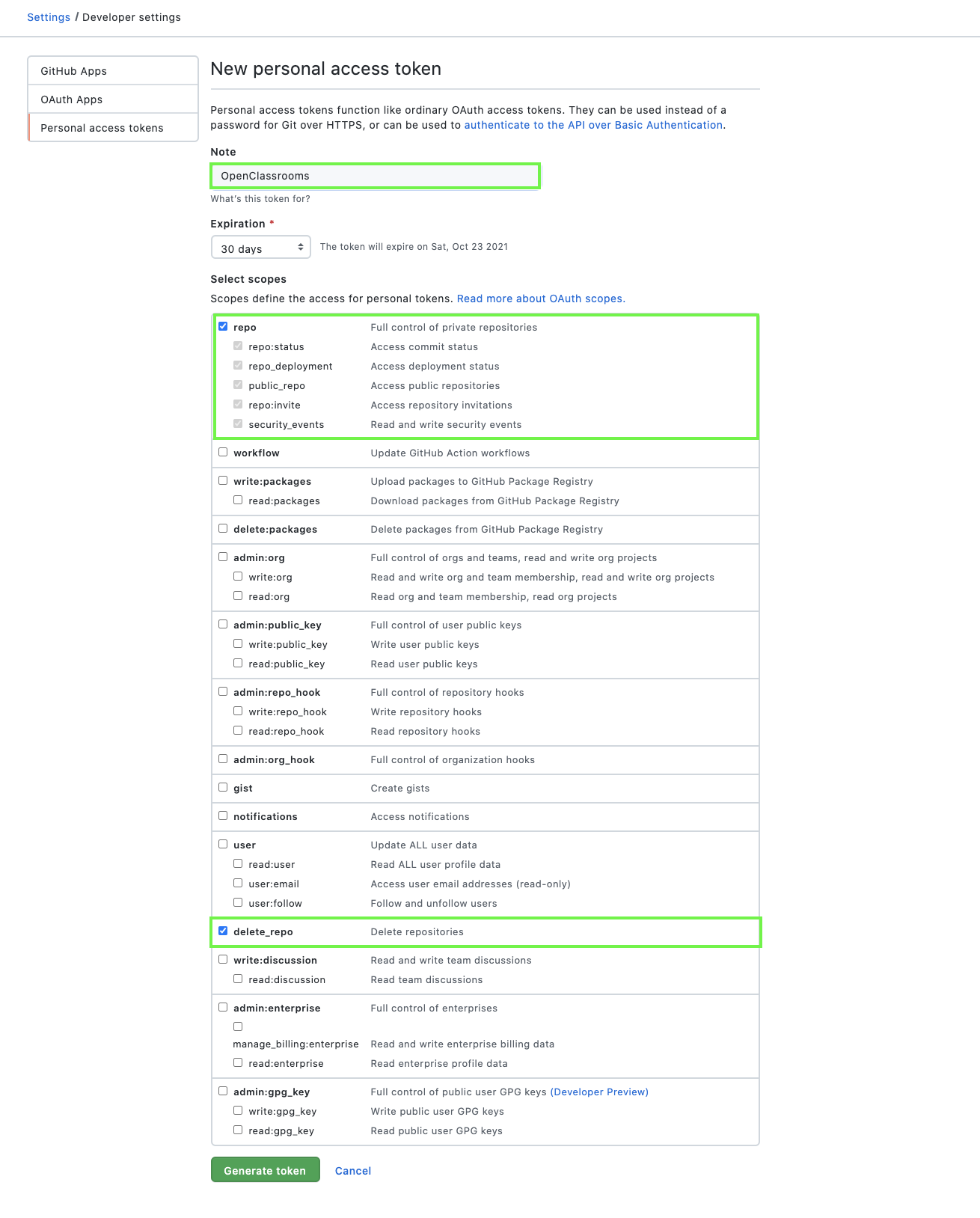
Pour commencer, allez sur ce lien : [https://github.com/settings/tokens.](https://github.com/settings/tokens)

Vous devriez obtenir cette page :

Page GitHub qui vous permet d'obtenir un token

Vous êtes à présent sur la partie qui vous permet de demander à GitHub de vous donner un token afin de pouvoir effectuer des opérations via l’API GitHub.

Cliquez sur **Generate new token**.

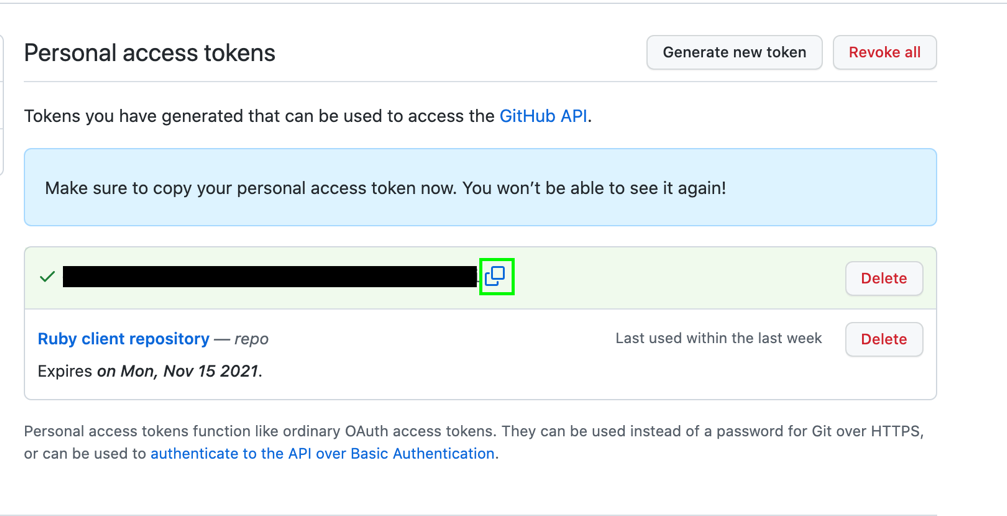
Génération d'un token

Dans notre cas, nous voulons que GitHub nous donne l’autorisation pour deux choses :

1. Effectuer des actions sur les repositories GitHub.
2. Supprimer des repositories GitHub

Un repo (**repository**, ou répertoire) est un espace de stockage pour le code dans GitHub. Si vous voulez en savoir plus à ce sujet, consultez ce [**chapitre**](https://openclassrooms.com/fr/courses/7162856-gerez-du-code-avec-git-et-github/7165707-saisissez-lutilite-des-depots-distants-sur-github) du cours **Gérez du code avec Git et GitHub**.

Suivez l’exemple ci-dessus et n’oubliez pas d’inscrire dans la section **Note**une information qui vous permettra plus tard de vous souvenir à quoi servait ce token. Cela peut être le nom d’une application, de votre API, etc. Dans mon cas, j’ai indiqué OpenClassrooms.😄 Cliquez ensuite sur **Generate token**(générer un token), et tada ! 🎉



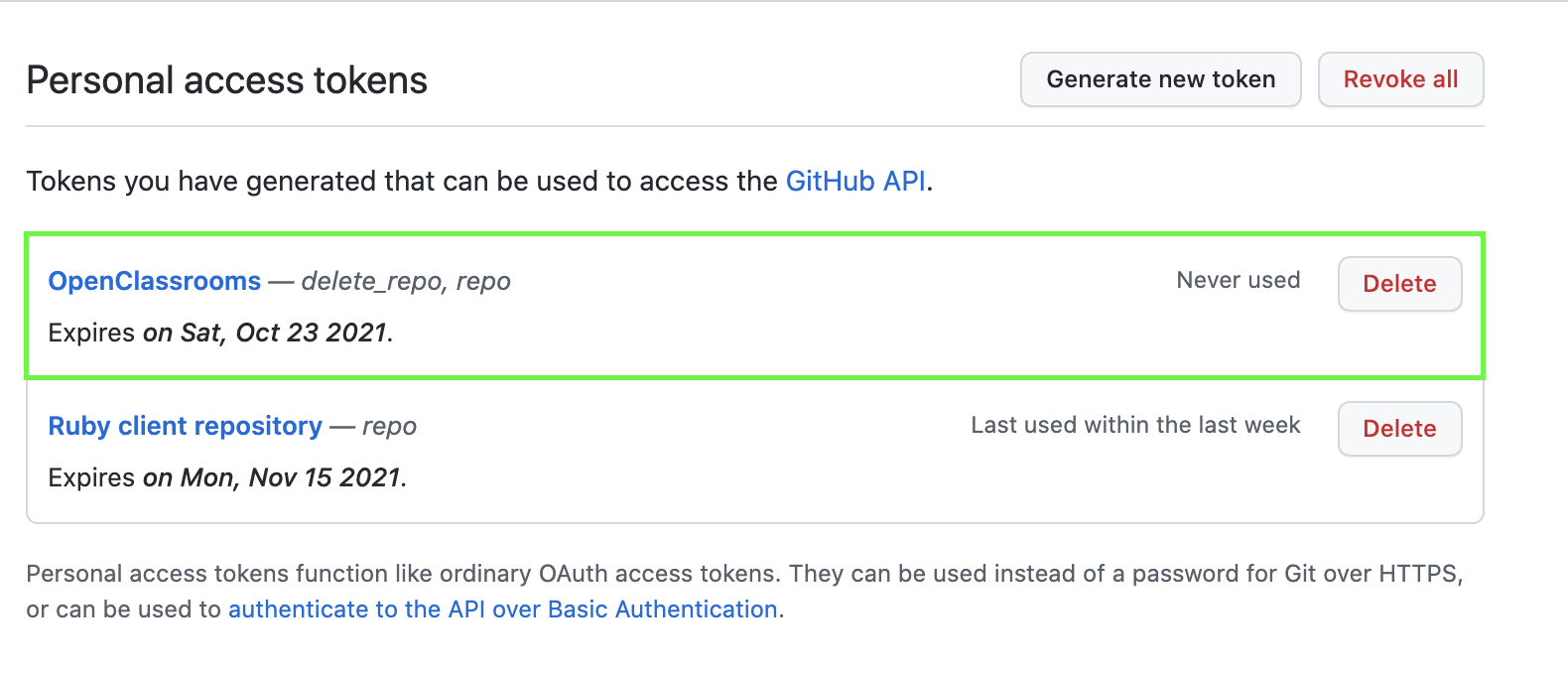
Génération de token

Votre token sera visible à l’emplacement du marqueur noir. J’ai choisi de masquer le mien. 🙊. Le mien : [ici](../1.5%20API%20KEY.rtf)

Un token est personnel et privé autant que votre code de carte bleue, alors gardez-le pour vous ! L’encadré bleu explique qu'il faut copier votre token personnel car vous ne serez plus en mesure de le voir.

Gardez-le précieusement et **copiez-le**, ou faites une capture d’écran afin de ne pas le perdre. Nous allons l’utiliser dans les chapitres suivants ! 🙋🏻‍♀️

Après avoir copié soigneusement votre token, rechargez la page. Vous devriez voir apparaître comme ci-dessus votre nouveau token par le nom que vous lui avez attribué ; dans mon cas : Open Classrooms. Vous verrez aussi les actions qu’il vous permet de faire sur l’API GitHub ; dans notre cas, les deux que nous avons séléctionnés : delete\_repo et repo.



Actions possibles avec le token

Nous avons vu ensemble comment utiliser des API. Sauf que les API peuvent être créées par des entreprises, des services, ou des développeurs indépendants. Comment être certain qu’elles sont fiables ?

*Privilégiez la sécurité : choisissez vos API avec discernement*

Comme vous l’avez appris précédemment, il existe des milliers d’API différentes que vous pouvez utiliser dans vos projets. Comme toujours, il est important de garder en tête la sécurité de vos données et de votre application. En tant que développeur, vous êtes responsable de la sécurité des données de vos utilisateurs ! Vous devez vous assurer que vos API proviennent d’une source fiable.

Mais comment faire ? Comment puis-je savoir si une API est fiable ou non ? 🤔

Il existe quelques méthodes simples et rapides pour vérifier si une API est fiable ou non. Les API de qualité auront plusieurs mesures de sécurité comme **l’authentification, l’autorisation et le cryptage**. Elles auront aussi été mises à jour récemment ; vous saurez donc qu’elles sont mises à jour en fonction des derniers standards de sécurité.

Prenons un exemple ! Sur GitHub, vous trouverez l’API Pokémon nommée [PokeApi](https://github.com/PokeAPI/pokeapi/) qui a été mise à jour récemment. 👇



Un exemple d’une API mise à jour récemment #PokeApi

Vous pouvez constater sur leur GitHub que la date de leur dernier commit est assez récente ; vous savez donc que cette API est maintenue et mise à jour. 😉

Avant chaque utilisation d’une API externe, vérifiez la date de la dernière mise à jour sur GitHub ou bien sur lson site Internet, lisez la documentation et si vous avez des doutes (ou non), regardez en ligne des avis ou posez simplement la question à un autre développeur.

*En résumé*

* Pour obtenir un token d’authentification GitHub, vous pouvez :
  + aller sur <https://github.com/settings/developers> et cliquer sur **Personal Access Tokens**(tokens d’accès personnel) ;
  + cliquer sur **Generate new token** (générer nouveau token) et saisir « Open Classrooms » dans les notes ;
  + vous verrez alors toutes les différentes options d’autorisation dont vous voulez doter votre API ;
  + choisissez toutes celles que vous voulez tester. Nous avons utilisé Repo et Delete (supprimer) ;
  + cliquez sur **Generate token**(générer token) et obtenez votre token API personnel ; 👏
  + assurez-vous de copier et sauvegarder ce token, car nous l’utiliserons par la suite.
* L’authentification est nécessaire pour garantir que seules les personnes avec les autorisations adéquates peuvent accéder à votre API.
* Les clés ou tokens API sont couramment utilisés dans une requête pour authentifier un utilisateur.
* Assurez-vous de bien vérifier la fiabilité d’une API avant de l’utiliser.
* Vous pouvez trouver les indications de mise à jour d’une API sur GitHub ou sur son site Internet.

GET n’a plus de secret pour vous, vous avez créé votre compte GitHub et obtenu votre premier token. Il est temps de passer à la vitesse supérieure et de jouer avec l’API GitHub ! 🚀

## [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)

**Manipulez des données avec l’API GitHub**

On avance, on avance ! Maintenant que vous avez un token d’authentification, vous pouvez utiliser l’API pour mettre à jour votre profil GitHub  !

Dans ce chapitre, nous allons pratiquer et utiliser le reste des opérations CRUD – Create (créer), Update (mettre à jour) et Delete (supprimer) – et leurs verbes HTTP équivalents – POST  (publier),  PUT (mettre) et DELETE (supprimer).

***Créez un repo GitHub avec la méthode POST***

Pour **créer** quelque chose de nouveau, ou une nouvelle ressource, on utilise le verbe http **POST**(publier). Qu’il s’agisse d’un nouveau tweet, d’une nouvelle photo ou d’une nouvelle publication (vous comprenez ? 😉). Par exemple, dès que vous remplissez un formulaire en ligne ou que vous en utilisez un pour vous inscrire et vous créer un nouveau compte, le verbe associé par défaut est POST (cf [cours PHP](../1.14.1%20PhP%20&%20MySQL/1.14.1%20PHP%20&%20MySQL.docx)).

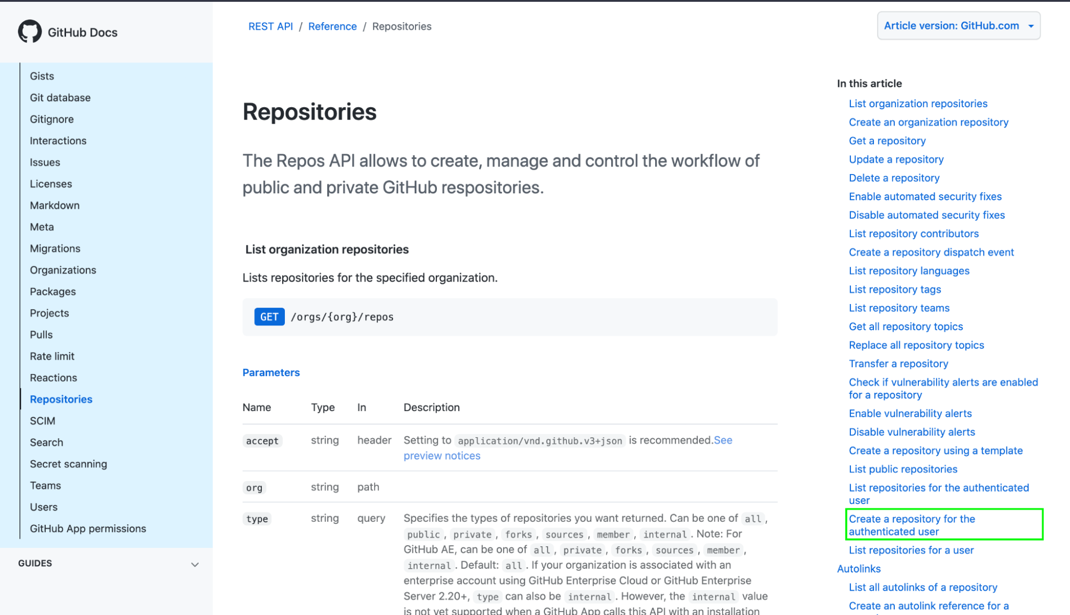
Comment l’API sait-elle ce qu’on essaie de **créer**quelque chose ? 🤔

C’est là qu’intervient le **body** ! Vous vous souvenez ? On l’a vu dans les chapitres précédents. Le body accompagne les requêtes POST et PUT pour contenir des informations supplémentaires. Vous pouvez intégrer les données que vous voulez créer dans le body de votre requête en utilisant du **JSON**.

Mettons tout ça en pratique ; je vous assure que ça fera sens ! 💪

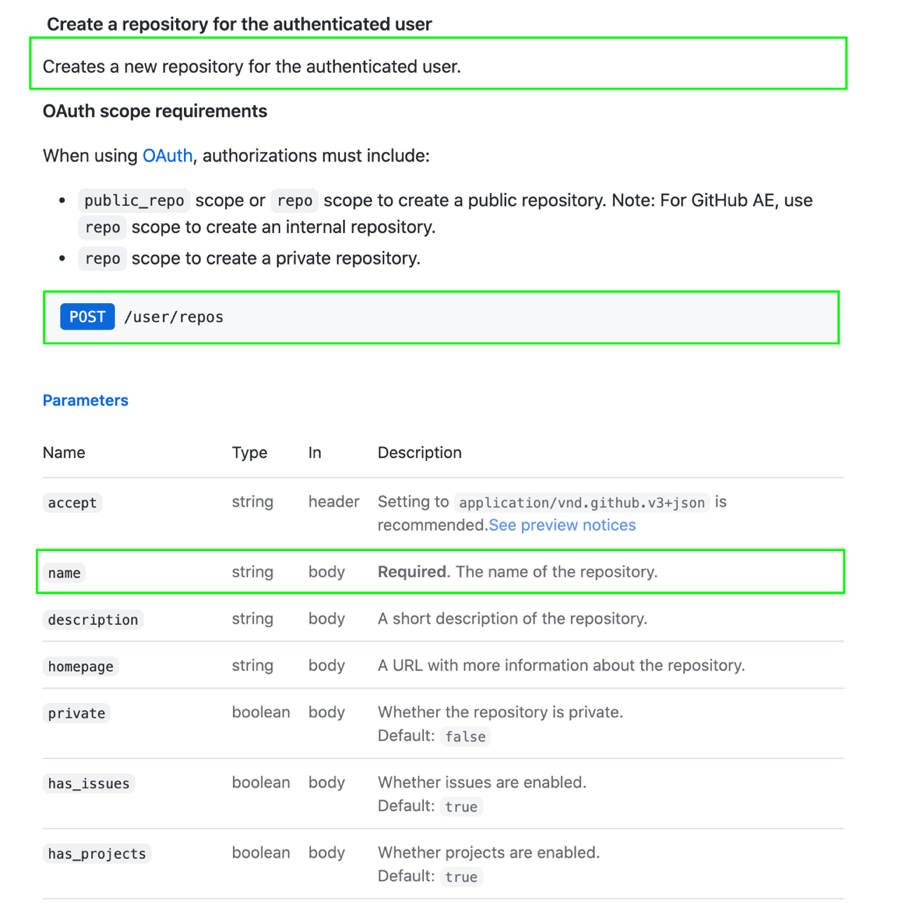
On veut créer un nouveau repo sur GitHub pour notre API. Super, mais comment faire ? 1re étape : la documentation !

Regardons ce que nous dit la documentation GitHub au sujet des repositories : [https://developer.github.com/v3/repos](https://developer.github.com/v3/repos/#create)



Documentation GitHub concernant les repositories

Houlà, on a beaucoup de choix ! Nous, ce qu’on veut, c’est créer un nouveau repo. Cliquez en bas à droite sur **[Create a repository for the authenticated user](https://docs.github.com/en/rest/reference/repos" \l "create-a-repository-for-the-authenticated-user)**.



Documentation pour créer un nouveau repo

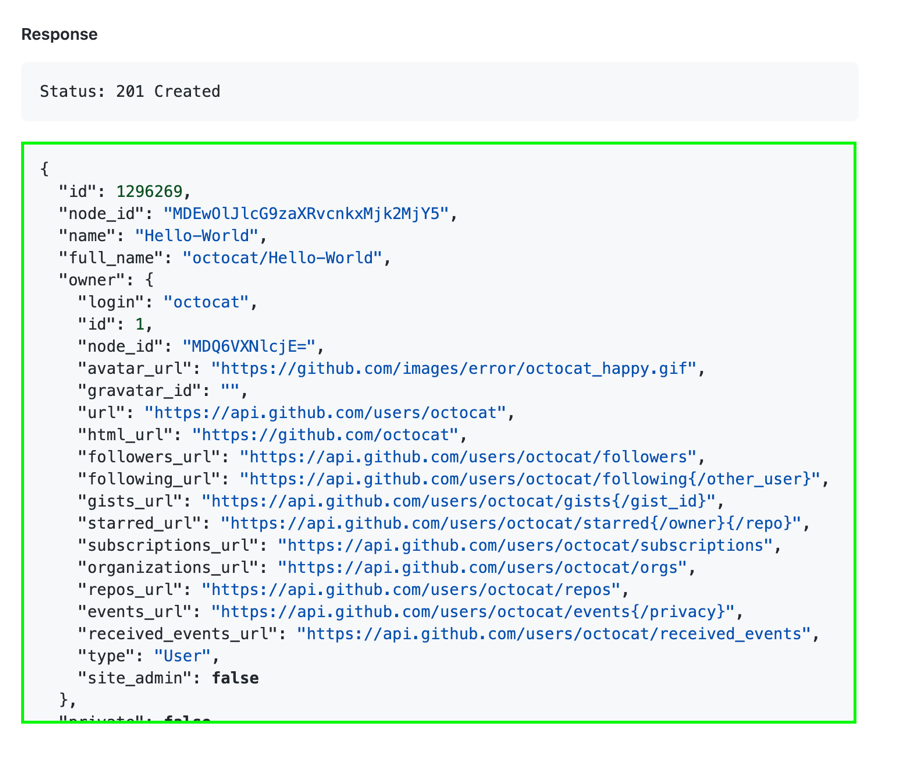
Dans le premier encadré en vert, on peut lire“Crée un nouveau repository pour un utilisateur authentifié”.

Pour créer un nouveau repository, il faut faire une requête **POST** vers **/user/repos**.

Parfait, on avance ! Maintenant, scrollez un peu et vous tomberez sur les paramètres.

L’encadré vert vous montre que le paramètre **name** est obligatoire.

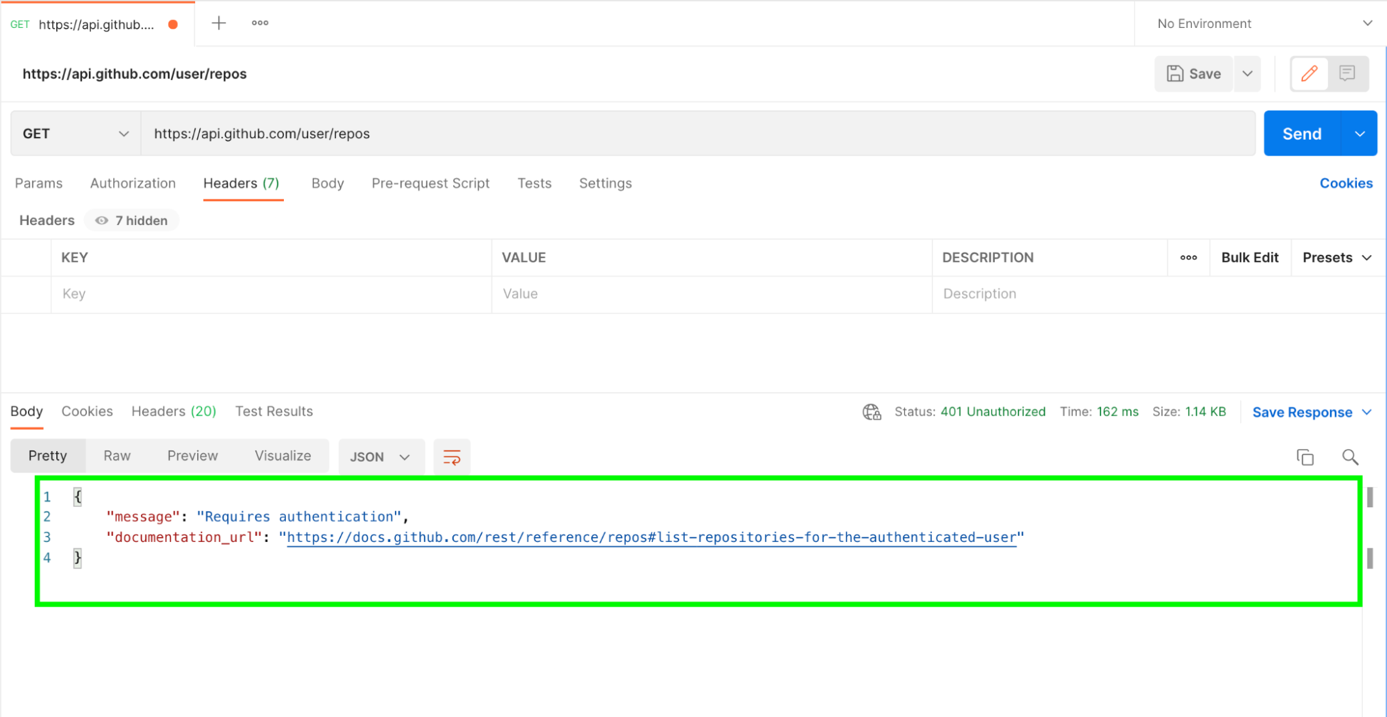
Mais ce n’est pas tout ! La documentation nous donne un exemple de requête et de réponse réussies.



Une réponse réussie

Nous avons notre endpoint, notre verbe http, nos paramètres et même un exemple ! Allons faire cette requête. 🙌Lancez Postman !

Sélectionnez POST dans le menu déroulant puis tapez l’endpoint pour créer un nouveau repo via l’API GitHub: <https://api.github.com/user/repos> ; et appuyez sur Send.

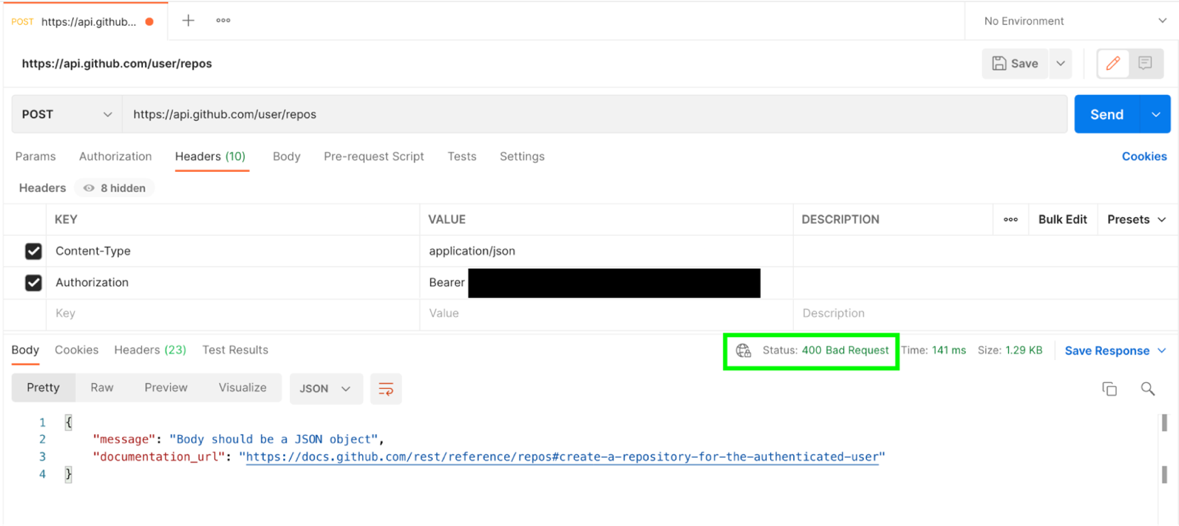
[](https://user.oc-static.com/upload/2021/10/15/1634296038817_image22.png)Message d'erreur

Arf, mais on a une erreur. Qu’est-ce que c’est ? 🤔

Le message nous indique qu’une authentification est nécessaire. Le code de réponse HTTP est 401 : Unauthorized (401 Non autorisé). Eh oui, notre token ! 😄

Reprenez votre token d’authentification API GitHub, celui que vous avez précieusement gardé. Ajoutez-le, comme sur l’image suivante, dans l’onglet Headers sous forme de clé-valeur où la clé sera Authorization et la valeur sera Token, votre token. Si mon token est abcde, alors la valeur sera : token abcde. (j’avais oublié la 1ère fois).

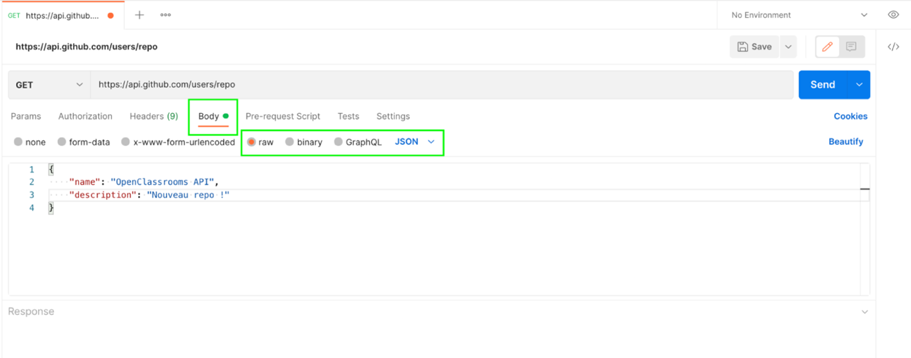
Ensuite, appuyez sur Send et regardez la réponse. 👇🏼



Statut de la requête avec le token

Encore une erreur ?! Mmmh... on a le bon endpoint, le token. Cette fois-ci, le code de réponse HTTP est : 400 Bad Request. Mais oui ! Il nous manque les paramètres 🙄. Eh oui, on demande à GitHub de nous créer un repo, mais on ne lui donne pas les informations.

Cliquez sur la section Body, puis sur Raw, binary et sélectionnez JSON.



Sélectionnez JSON

Maintenant, on peut ajouter nos paramètres sous forme de clé-valeur. Il nous faut un nom de repo qui est obligatoire, et ajoutons une description.

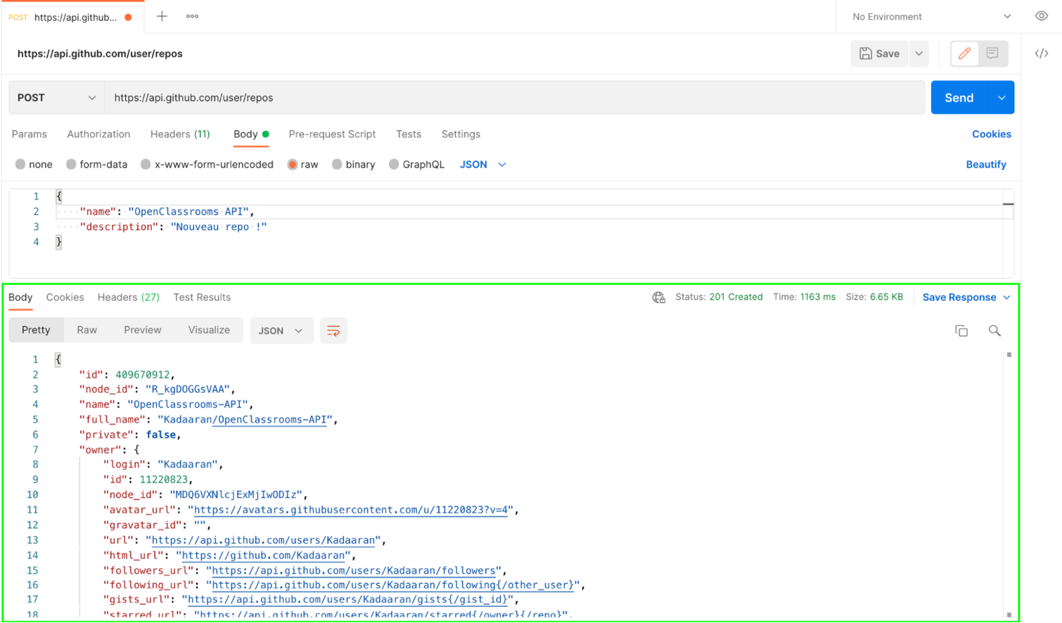
{

"name": "OpenClassrooms API",

"description": "Nouveau repo !"

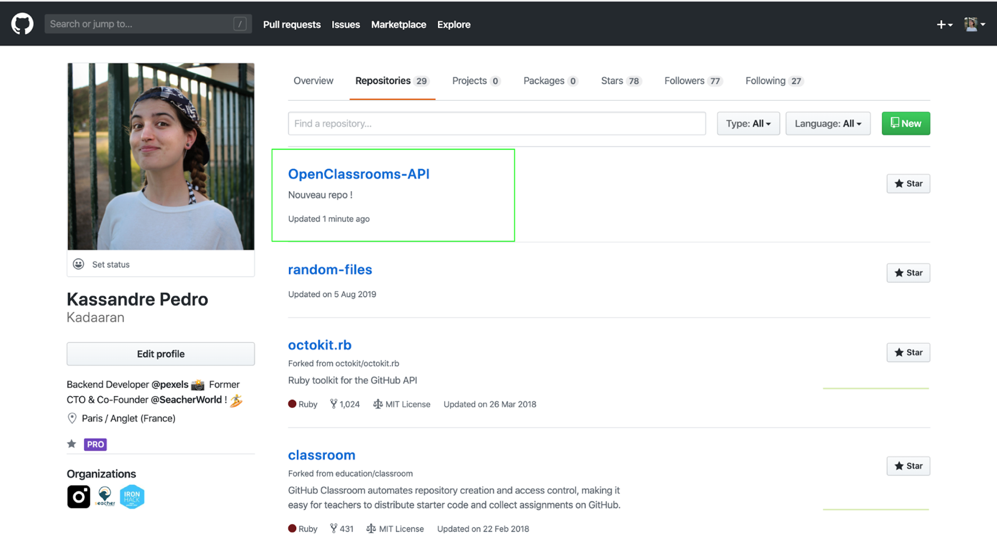
}

Appuyez sur Send et observez la réponse. 👇🏼



Le repo a été créé

Wooohooo, notre repository a été créé 🎉. Le code de réponse HTTP est bien 201 Created. L’API nous renvoie les informations du nouveau repo crée. On peut aussi vérifier sur GitHub si le repo apparaît bien dans notre liste, et c’est le cas ! 👇🏼🎉



Vérification de la création du repo

Pour **créer** un repo GitHub en utilisant l’API GitHub, nous avons :

* consulté la [documentation](https://api.github.com/user/repos) GitHub pour trouver l'endpoint adapté : POST user/repos ;
* ajouté notre token d’authentification à nos paramètres dans le header avec Postman ;
* ajouté les détails de notre repo à notre body en JSON avec Postman ;
* vu qu’une fois que nous avons effectué une requête réussie avec notre API, le repo est apparu sur l’UI de notre profil GitHub !

(cf vidéo, si nécessaire, mais c’est clair ici. ☺ ).

***Mettez à jour un repo GitHub avec les méthodes PUT/PATCH***

Vous avez **créé** votre repo GitHub, mais la description ne vous plaît plus trop, vous changez d’avis et vous voulez la modifier ! Vous pouvez utiliser PUT pour **mettre à jour** une ressource déjà existante dans votre API.

Sur Internet, on croise souvent PUT avec PATCH ? Les deux signifient la même chose ou est-ce qu’ils sont différents ? 🤔

* PUT : met à jour la ressource complète (c’est-à-dire, remplace tout).
* PATCH : met à jour uniquement la partie de la ressource qui a été envoyée.

On ne va pas s’attarder ici sur la différence entre les deux. Mais si vous ne savez pas lequel utiliser et pour quelle situation : consultez la documentation de l’API !

Reprenons la documentation des repos GitHub mais cette fois-ci cliquons sur [Update a repository](https://docs.github.com/en/rest/reference/repos#update-a-repository).

Pour mettre à jour un repo, il nous faut utiliser la méthode PATCH sur l’endpoint /repos/:Owner/:repo.

Les deux points ( : ) devant *owner*et *repo* signifient qu’il nous faut l’identifiant unique du repository que vous voulez modifier, et celui du owner (propriétaire) du repo : vous, en l'occurrence.

Changez le verbe par PATCH. Puis entrez l’endpoint correspondant. Mon username est Kadaaran et mon repo se nomme OpenClassrooms-API. Changez Kadaaran par votre username et OpenClassrooms-API par votre nom de repo s’il est différent. Dans la section Body, tapez une nouvelle description. Dans mon cas, j’ai mis : Paragraphe ci-dessus à revoir.

{

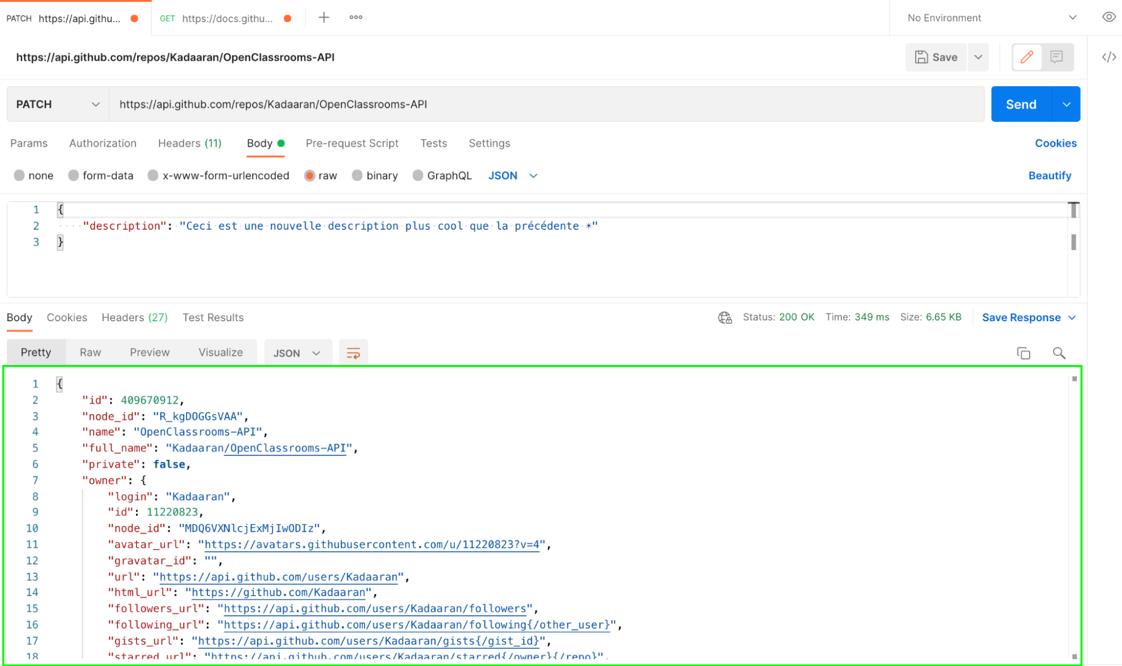
"description": "Ceci est une nouvelle descr. plus cool que la précédente ☀️"

}



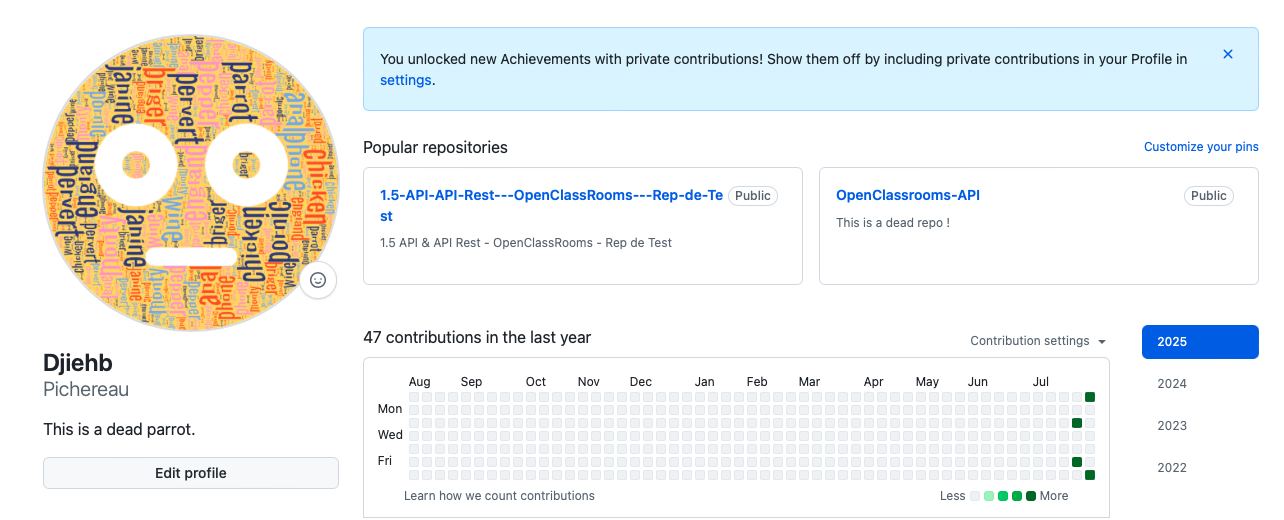
Changement de la description

Appuyez sur Send et regardez la réponse. 👇🏼



Description mise à jour

Notre repo a été mis à jour ! 🎉 Nous obtenons la réponse avec le body : le code de réponse HTTP est bien 200 OK.



La nouvelle description

Pour **mettre à jour** un repo GitHub, nous avons :

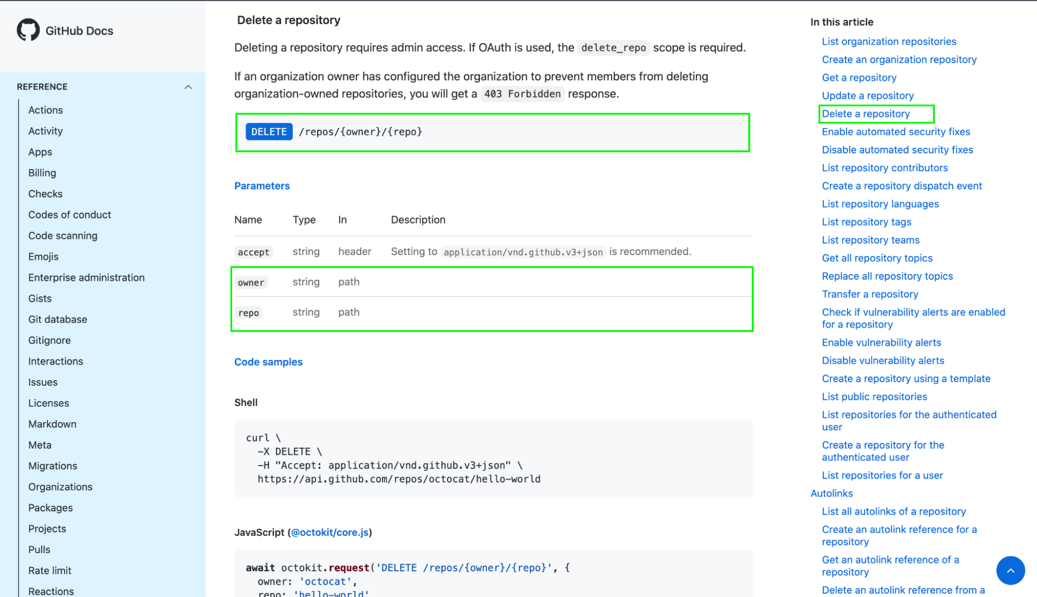
* vérifié la documentation GitHub pour l’URI approprié : PATCH /repos/wner/:repo ;
* utilisé Postman, changé notre verbe HTTP en PATCH, et saisi le propriétaire et le nom de repository que nous voulons modifier, ainsi que changé la description dans le body ;
* vu la nouvelle description de notre repository dans notre UI GitHub

***Supprimez votre repo GitHub avec la méthode DELETE***

Vous décidez à présent que vous n’avez pas réellement besoin de votre repo GitHub – alors allez-y et supprimez-le avec DELETE  !

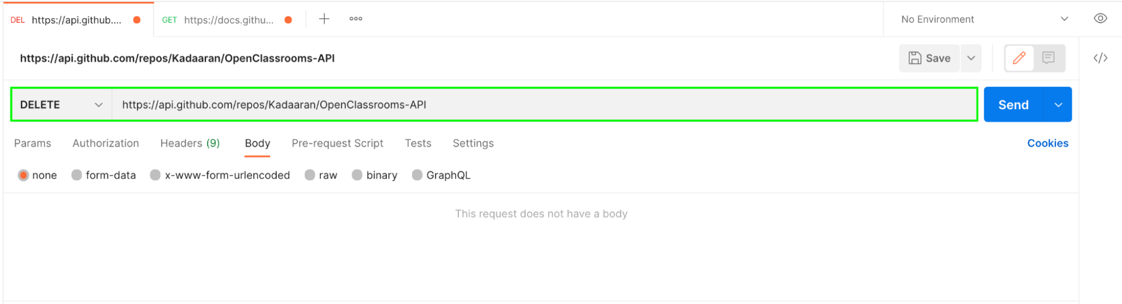
Allez hop ! On reprend la même gymnastique que pour les requêtes précédentes. Tout d’abord : la documentation ! Comment supprime-t-on un repository GitHub ?

Dans la liste des méthodes disponibles, nous trouvons : [Delete a repository](https://docs.github.com/en/rest/reference/repos" \l "delete-a-repository). Cliquez dessus !



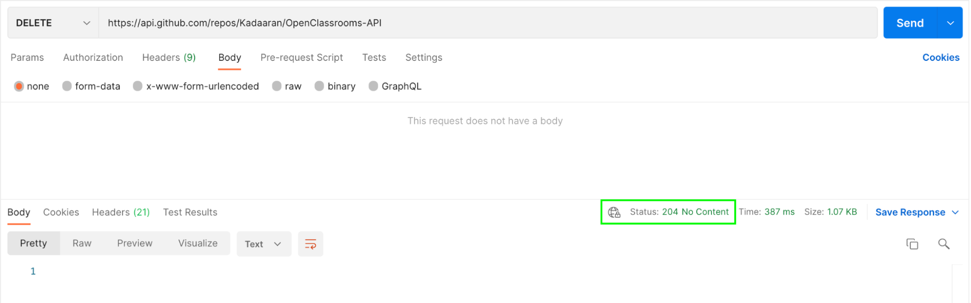
Documentation pour supprimer un repo

Selon la documentation, pour supprimer un repository GitHub, il nous faut utiliser le verbe DELETE sur l’endpoint /repos/Owner/:repo. Si la requête est un succès, alors nous devrions avoir 204 No Content (204 Aucun contenu) comme réponse. Essayons avec Postman.



Suppression d'un repo

Tapez les informations nécessaires : votre username et le nom du repo que vous souhaitez supprimer. Pas besoin de body pour DELETE, cliquez sur none. Puis cliquez sur Send et regardez la réponse obtenue. 👇🏼



Le repo est supprimé

Et voilà ! Votre repository est supprimé à tout jamais ! 🎉Nous obtenons un body vide et un statut 204 No Content comme l’a indiqué la documentation.

Vérifiez sur GitHub que votre repo a bien été supprimé. 👇🏼

Vous avez vu comment fonctionnent POST, PUT et DELETE et vous les avez vous-même utilisés avec l’API GitHub et Postman. Félicitations ! 👏

Vous les avez utilisés dans le contexte d’un repo GitHub, mais vous pouvez utiliser ces verbes HTTP pour tout type de fonctionnalité, en fonction de l’API.

* Vous avez vu dans la documentation GitHub que l’URI correct est le même que pour Update, mais avec un nouveau verbe HTTP : DELETE /repos/:Owner/:repo.
* Vous avez vu que notre repo est supprimé et ne se trouve plus dans notre UI GitHub !

***En résumé***

* Vous pouvez utiliser POST pour**créer** des ressources.
* PUT ou PATCH vous permet de **mettre à jour**des ressources.
* DELETE vous permet de **supprimer**des ressources !

Dans la prochaine partie, nous allons étudier les meilleures pratiques lorsque vous concevez vous-même une API  ! Mais avant, il est temps de réviser ce que vous venez d’apprendre avec le quiz de fin de partie !

# 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)

***Préparez-vous à concevoir votre propre API REST***

Maintenant que vous avez vu comment **utiliser** une API externe, parlons davantage de quand et comment **construire** concrètement une API REST vous-même.

Il existe deux raisons **principales**pour lesquelles vous pourriez vouloir développer votre propre API :

1. Si vous construisez votre propre application web ou mobile, par exemple pour des recettes de cuisine (ici, cf mon php project ([ici](../1.14.1%20PhP%20&%20MySQL/1.14.1%20PHP%20&%20MySQL.docx)), où vous savez que vous devrez **sauvegarder et manipuler beaucoup de données**. 🍳 Vous auriez absolument besoin d’une base de données, mais il n’est pas souhaitable que les développeurs front-end aient besoin de connaître SQL pour pouvoir afficher les informations sur votre UI. Vous voulez aussi respecter les normes pour les applications web pour qu’il y ait une couche API entre votre base de données et l’accès à vos données. Vous ne voulez pas que n’importe quel développeur soit en capacité d’accéder directement à votre base de données, pour des questions de sécurité et de confidentialité. Et vous voulez une manière rapide et efficace de rechercher et requêter les données existantes. Pour ce faire, vous aurez besoin d’une API interne.
2. Si vous construisez une application web ou mobile où **vous voulez que d’autres développeurs soient en capacité d’interagir avec vos données**. Par exemple, une application où vous collectez des données sur les personnages de votre série télé favorite et où vous voulez que d’autres développeurs puissent ajouter et utiliser des données.

UI signifie User Interface ou interface utilisateur en français. LE terme UI fait référence à l’interface utilisateur par le biais de laquelle l’utilisateur interagit, que ce soit un site internet, un logiciel ou une application mobile.

***Lancez-vous dans la conception de votre API***

Comme pour toute chose un peu compliquée, un **design** réfléchi et structuré est essentiel pour obtenir un succès sur le long terme. Réfléchir aux concepts de design-clés qui rendront votre API plus **agréable d’utilisation** et plus**facile à scaler.**

Scaler vient de l’anglais to scale et signifie en quelques mots la capacité de votre application/logiciel/site internet à s’adapter et à fonctionner même en cas de forte augmentation soudaine du trafic, de demandes ou juste d’un grand nombre de fonctionnalités implémentées.

Allons-y étape par étape. Commencez d'abord par vous poser des questions sur les fonctionnalités importantes dont vous avez besoin pour votre API :

* De quel type d’endpoints avez-vous besoin ?
* Quelles ressources devez-vous créer ?
* De quelles ressources avez-vous besoin pour y effectuer les opérations **CRUD**?
* Avez-vous besoin des quatre opérations **CRUD** pour chaque ressource ? Ou seulement d’une ou deux ?

Une fois que vous aurez répondu à ces questions, vous serez prêt à commencer à préparer … votre documentation.

***Documentez toute la journée***

Lors des chapitres précédents nous avons dû consulter la documentation GitHub afin de pouvoir utiliser l’API comme il se doit. Sans toutes les informations de la documentation, nous aurions eu du mal à avoir le bon endpoint ou savoir quels paramètres préciser.. La documentation est une des parties **les plus importantes** des API ! Quand vous créez une API, vous devez également tenir compte des **autres développeurs** qui utilisent votre API. Cela implique d’avoir une bonne **documentation** pour qu’ils puissent facilement comprendre ce que votre API peut accomplir et comment l’utiliser.

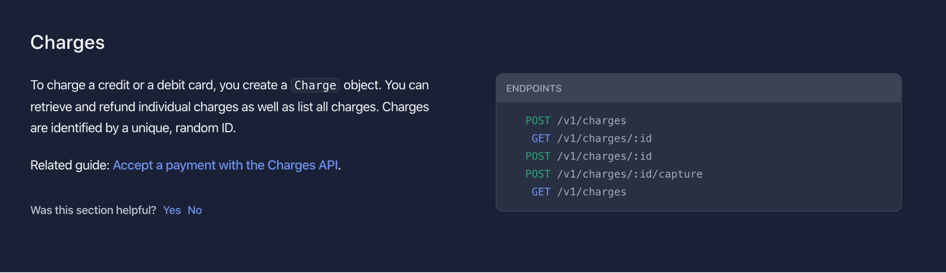
Comme vous l’avez vu avec la documentation GitHub, de nombreuses informations doivent être expliquées à votre utilisateur API. Elles incluent :

1. Les descriptions des ressources API.
2. Les URI et verbes HTTP disponibles ainsi que leur fonction.
3. Les paramètres (s’il y en a) qui doivent être donnés à l’endpoint.
4. Un exemple de requête.
5. Une réponse typique pour la requête donnée.

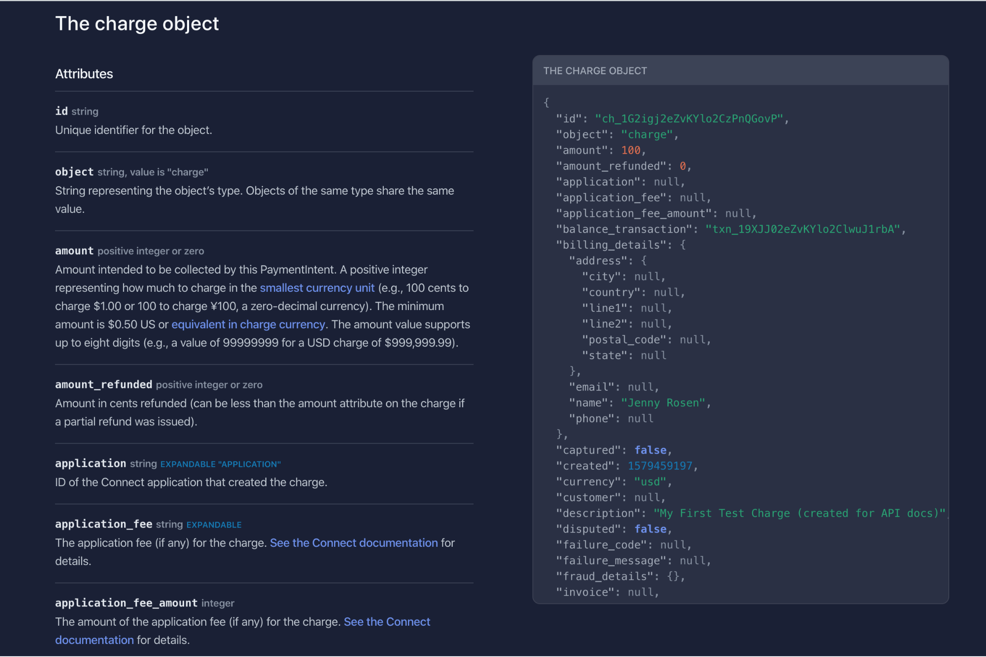
Chaque API organise la documentation de manière légèrement différente, mais voici quelques bons exemples pour vous aider.

*Documentation de l’API Stripe*

Dans [l’API de Stripe](https://stripe.com/docs/api) vous pouvez clairement voir que la documentation décrit l’URL de la ressource et les informations pertinentes que les développeurs doivent pouvoir comprendre.

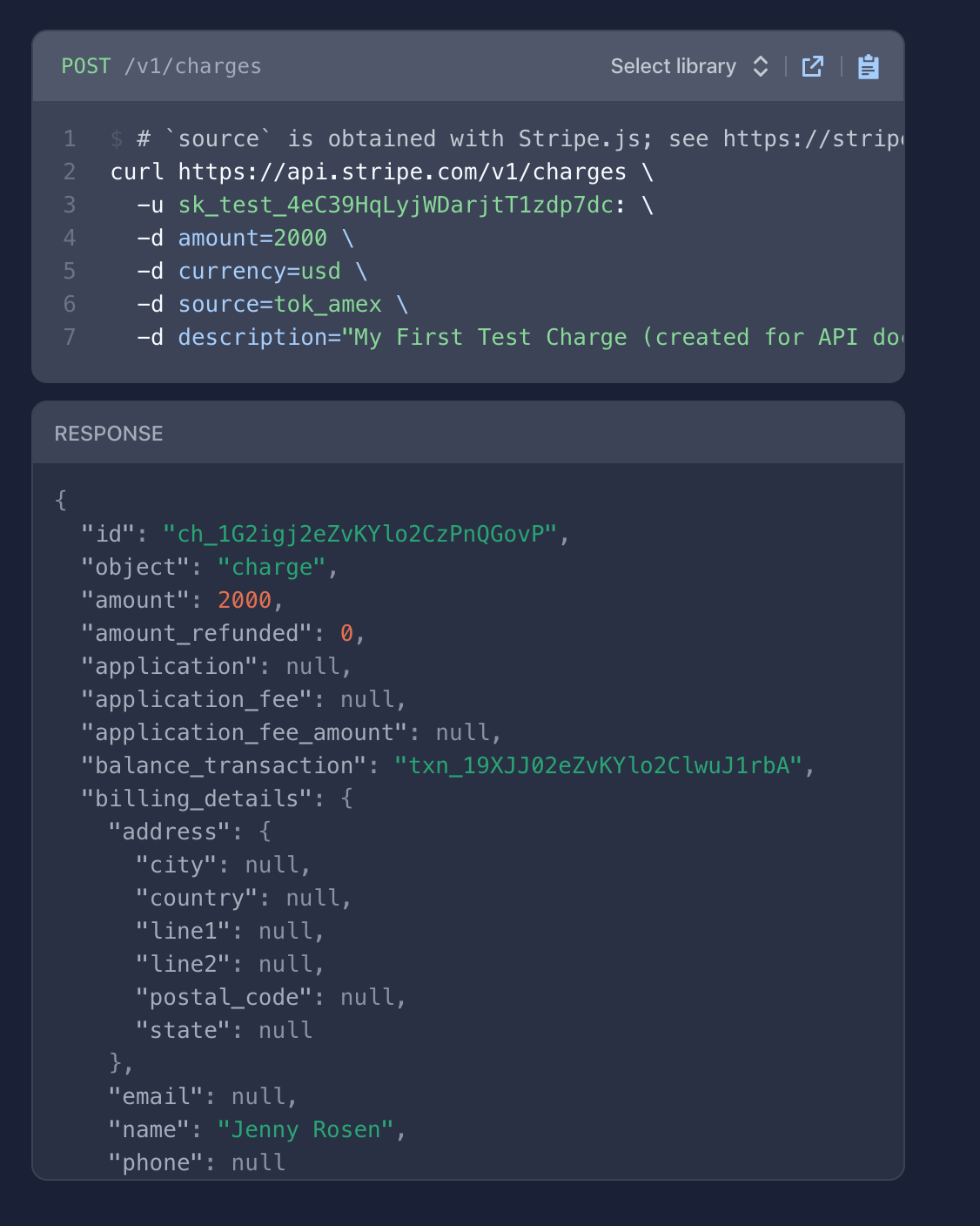


Un exemple dans la documentation de Stripe



Un exemple dans la documentation de Stripe

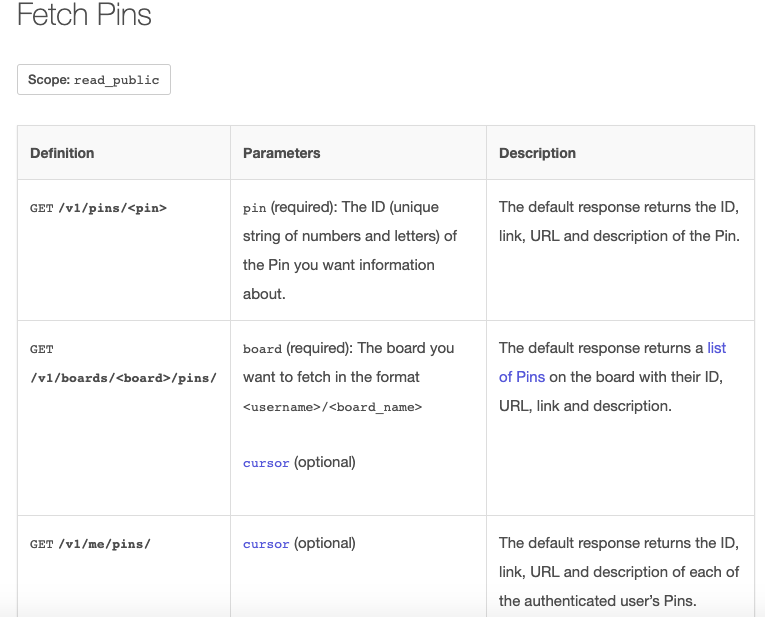
La [documentation](https://stripe.com/docs/api/charges/create) montre également un exemple de réponse aux requêtes API :



Un exemple de réponse dans la documentation Stripe

*Documentation de l’API Pinterest*

L’[API Pinterest](https://developers.pinterest.com/docs/api/pins/) contient un bon exemple d’affichage d’un verbe HTTP, d’un URI, de paramètres, et d’une description de la fonction de l’endpoint.



Documentation de l’API Pinterest

La documentation donne de nombreuses informations sur les endpoints et les requêtes, mais elle se doit aussi de documenter la gestion des erreurs.

*Gérez les erreurs 🙅‍♀️*

Comme nous l’avons vu dans les chapitres précédents, une requête peut être un succès mais aussi un échec, et ce pour diverses raisons : ressource introuvable, serveur indisponible… Votre API doit avoir une bonne **gestion des erreurs**. Si un développeur essaye accidentellement d’utiliser votre API pour un usage non souhaité (authentification incorrecte, lettres inutiles ajoutées à une requête), votre API doit être capable de dire à l’utilisateur quelle a été son erreur pour qu’il puisse la corriger. Et oui, personne n’est à l’abri d’une faute de frappe ! Le code HTTP donne une information sur le type d’erreur mais est généralement accompagné d’un message d’erreur dans le body de réponse.

Vous avez vu à quoi ressemblaient certaines erreurs dans l’API GitHub lorsque nous avons voulu créer un repository dans la dernière partie. Souvenez-vous, quand nous avons essayé d’effectuer une requête sans notre token d’authentification, nous avons reçu la réponse suivante :

{

"message": "Requires authentication",

"documentation\_url": "https://developer.github.com/v3/repos/#edit"

}

C’est ainsi que grâce au message d’erreur nous avons su que nous avions oublié notre code d’authentification ! Il existe de nombreux types d’erreurs différents pouvant se produire dans votre API qui, si elles sont gérées correctement, peuvent faciliter la vie des développeurs lorsqu’ils utilisent votre API. 💪

L’[API de Meetup](https://www.meetup.com/fr-FR/meetup_api/docs/?uri=%2Fmeetup_api%2Fdocs%2F) contient un bon exemple de toutes les erreurs différentes qui peuvent se produire quand vous utilisez leur API et ce qu’elles signifient :

Toutes les erreurs possibles de l’API Meetup

|  |  |
| --- | --- |
| announce\_error | Erreur dans l’annonce d’un événement |
| description\_error | Désolé, votre description Meetup est trop longue |
| duration\_error | La durée de cet événement n’est pas valide. |
| event\_error | L’ID fourni ne correspond pas à un évènement pouvant être modifié |
| event\_hosts\_error | Un ou plusieurs des membres donnés ne sont pas des membres actifs de ce groupe |
| featured\_photo\_id\_error | Il y a un problème avec la photo fournie. |
| fee.amount\_error | Si le groupe se trouve aux États-Unis, le montant à acquitter ne peut pas excéder 4999. Sinon, il ne peut pas excéder 1 000 000. |
| fee.refund\_policy\_error | La politique de remboursement de fonds ne peut pas excéder 250. |
| guest\_limit\_error | Le nombre d’invités maximum n’est pas valide. |
| how\_to\_find\_us\_error | Votre description « Comment nous retrouver » est trop longue. |

Une partie nécessaire et importante lors de la conception de votre API est la réflexion sur les possibilités de mauvaise utilisation par les utilisateurs ! Ainsi en fonction de l’erreur, vous pouvez renvoyer le message approprié dans la réponse.

***En résumé***

* Vous pouvez construire une API soit pour l’utiliser dans votre propre application, soit pour que d’autres développeurs puissent interagir avec vos données.
* Il est important en concevant une API de penser à toutes les fonctionnalités nécessaires avant de commencer à la développer.
* La documentation et une bonne gestion des erreurs sont importantes pour que d’autres développeurs puissent comprendre votre API.

Maintenant que vous avez les bases, concevez les endpoints de votre API. OK, mais comment faire ? Voyons cela ensemble dans le chapitre suivant.

## [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)

Lors de la création d’une API, il est important de ne pas se lancer tête baissée et de réfléchir à l’architecture de votre API, quitte à tout poser sur papier avant. Quel sera le rôle de votre API ? Quelles seront les ressources ? Quels seront les différents accès autorisés pour les utilisateurs ?

Bien structurer votre API dès le début de sa conception vous permettra d’anticiper les erreurs mais aussi d’avoir une API plus solide et mieux conçue.

Dans ce chapitre, vous ne coderez pas une API mais allez la concevoir. C’est un exercice de réflexion, afin de vous préparer au mieux à la création d’une API.

***Concevez une API de partage de photos***

Dans cette partie, nous allons réfléchir à la manière de concevoir une API pour une application de partage de photos nommée InstaPhoto. 📸  Nous voulons que les utilisateurs puissent publier et partager des photos avec leurs amis, commenter les photos des autres, créer des hashtags, rechercher des photos par localisation, la totale, le grand jeu !

Prenez une minute pour écrire ce à quoi vous avez besoin de réfléchir lors de la conception d’une API pour InstaPhoto. De quelles **ressources**avez-vous besoin ? Quels **URI** seraient compréhensibles par d’autres développeurs ? C’est aussi une belle occasion de revoir ce que vous avez appris dans les parties précédentes  !

Voici quelques exemples de **ressources** dont vous aurez absolument besoin :

* Photo ;
* User ;
* Location (localisation, en français) ;
* Post.

À partir de là, vous pouvez commencer à réfléchir à des questions importantes, comme celles-ci :

* Quels endpoints auront besoin d’autorisations ?
* Quelles ressources voulez-vous pouvoir mettre à jour ?
* Aurez-vous besoin de pouvoir modifier des publications après leur création ?
* Les commentaires doivent-ils pouvoir être supprimés ?
* Avez-vous besoin de toutes les opérations CRUD pour chaque ressource ? Ou seulement d’une ou deux ? Lesquelles ?

Il n’y a pas de bonne ou de mauvaise réponse ici – cela dépend simplement de l’application InstaPhoto que vous voulez développer ! À vous de choisir ! ✨

Exemple, moi :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | C | R | U | D |
| Photo | Oui, si user | Oui | Oui, si user | Non |
| User | Si Admin | Oui | Infos, si mme usr ou admin | Si Admin |
| Location | Oui | Oui |  | Si Admin |
| Post | Oui | Oui | Si propre ou admin | Oui |

***Créez vos endpoints***

Lors de la conception d’endpoints, le **naming** (ou nommage, en français) est un élément clé. Vous voulez que les développeurs qui l’utilisent comprennent naturellement ce que chaque endpoint est censé faire. Les conventions de naming actuelles demandent d’inclure uniquement la **ressource** que vous voulez mettre à jour, et non le **verbe** que vous voulez mettre en œuvre.

Pour faire simple, utilisez uniquement le **nom de la ressource**dans l’URI, car l’action se trouve déjà dans le verbe HTTP. Cela donnerait :

* POST /photo
* PUT /photo
* GET /photo

Puisque que le**verbe**de l’action est déjà inclus dans la requête HTTP, il n’est pas nécessaire de le préciser lors de la conception des endpoints.

Pour que vous puissiez mieux comprendre, des endpoints mal nommés ressembleraient à ceci :

* POST /createPhoto
* PUT /updatePhoto
* GET /getPhoto

***Simulez des endpoints supplémentaires***

Considérons les endpoints possibles pour notre InstaPhoto.

Ici aussi, il n’y a pas de bonne ou de mauvaise réponse, cela dépend simplement des besoins de votre application et de la façon dont vous voulez la concevoir.

Imaginons que vous vouliez que les utilisateurs soient en capacité de créer, voir et supprimer des photos – mais pas de les modifier une fois publiées. Tous ces éléments nécessitent également une authentification, car vous ne voudriez pas que les utilisateurs puissent publier, modifier ou supprimer des photos qui ne leur appartiennent pas ! 🔐

Maintenant, prenez une feuille de papier et écrivez quelques endpoints pour **créer, voir et supprimer des photos**. Rappelez-vous que vous voudrez peut-être spécifier une photo en particulier à visionner et à mettre à jour, non ? 😉 Une fois que vous aurez fini, regardez juste en-dessous si ce que vous avez écrit correspond à mes réponses.

Alors, curieux de connaître mes réponses :

* ●  POST /photos
* ●  GET /photos/{photo\_id}
* ●  DELETE /photos/{photoId}

Vu que l’on veut généralement voir ou supprimer des photos en particulier, le fait d’ajouter un ID pour les deux derniers endpoints est logique. Il permet à l’API de savoir de quelle photo il s’agit.

Et maintenant, vos utilisateurs ! De quels verbes HTTP aurez-vous besoin pour un compte utilisateur ? 🤔 Et puis, même si vous ne voulez pas que les utilisateurs puissent modifier leurs photos une fois qu’elles sont déjà créées, ils doivent au moins pouvoir **modifier leur profil utilisateur**! Par ailleurs, vous voulez qu’ils puissent **voir les informations disponibles publiquement sur un utilisateur** sans authentification (exactement comme l’API GitHub) ; donc pour cet endpoint, vous n’aurez pas besoin d’authentification. Quelles sortes d’endpoint pourriez-vous concevoir pour cela ?

Voici mes résultats :

* ●  POST /users
* ●  GET /users/{userId}
* ●  PUT /users/{userId}
* ●  DELETE /users/{userId}

Pour créer, modifier et supprimer un nouvel utilisateur, vous aurez besoin d’une authentification ainsi que d’un ID spécifique.

Et maintenant, les commentaires ! Comment les ajouter ? Si vous voulez que les utilisateurs puissent commenter les photos d’un autre utilisateur, il vous faudra imbriquer une ressource dans une autre. Et ce, parce qu’un commentaire doit être associé à une photo spécifique. Par exemple, imaginons que vous vouliez créer un nouveau commentaire pour une photo avec ID. Vous pourriez créer l’endpoint POST /photos/{photoId}/comments. Utilisez POST parce que vous créez quelque chose. Ensuite, travaillez en arrière : créez un commentaire (/comments) pour une photo spécifique (/{photoId}) parmi toutes les autres photos (/photos).

Et maintenant, voici une description de trois endpoints supplémentaires que vous devrez créer :

* obtenir tous les commentaires pour une photo spécifique avec l’ID {photoId} ;
* obtenir le commentaire avec l’ID {commentId} pour la photo avec l’ID {photoId} ;
* supprimer le commentaire avec l’ID {commentId} pour la photo avec l’ID {photoId}.

Vous ne voulez pas que les commentaires soient modifiables, n’incluez donc pas d’endpoint pour cela. Essayez d’écrire les endpoints pour ces trois éléments.

Voici ce que j’ai obtenu, le premier inclus :

|  |  |
| --- | --- |
| Endpoint | Description |
| POST /photos/{photoId}/comments | Créer un nouveau commentaire pour la photo avec l’ID {photoId} |
| GET /photos/{photoId}/comments | Obtenir tous les commentaires pour une photo spécifique avec l’ID {photoId} |
| GET /photos/{photoId}/comments/{commentId} | Obtenir le commentaire avec l’ID {commentId} de la photo avec l’ID {photoId} |
| DELETE /photos/{photoId}/comments/{commentId} | Supprimer le commentaire avec l’ID {commentId} de la photo avec l’ID {photoID} |

Tous ces endpoints nécessitent une authentification, car ils doivent seulement être modifiables par l’utilisateur qui les a créés, et non les autres. Et vous pourriez adopter le même mode de raisonnement pour toutes les autres ressources dont vous aurez besoin.

Rappel : ici, nous avons seulement conçu les endpoints. Les informations supplémentaires comme par exemple le contenu du commentaire, ou le nom de l’utilisateur que vous souhaitez modifier, seront contenus dans le body.

***En résumé***

* Lors de la planification, pensez à quelles opérations CRUD seront nécessaires aux ressources.
* Utilisez des noms de ressources et non des verbes en nommant les endpoints.
* Pensez aux ressources que vous devrez imbriquerdans d’autres ressources.
* Pensez aux ressources qui nécessiteront une authentification.

Vous avez la liste des opérations CRUD nécessaires, vos ressources et leur nom, les ressources complémentaires, celles qui nécessiteront une authentification, la liste de vos endpoints. Votre base est solide, mais vous pouvez approfondir le tout avec la recherche, par exemple. Passez au chapitre suivant, nous allons étoffer nos endpoints grâce à quelques fonctionnalités avancées !

## [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)

Jusqu’à présent, nous avons uniquement vu des endpoints basiques pour obtenir une liste de ressources ou une ressource unique par ID. Mais il existe tellement **plus** de possibilités avec les API  ! 💪 Selon la manière dont vous concevez vos URI, vous pouvez créer des requêtes plus complexes comme la recherche, le classement, et plus encore. Nous reprendrons l’application InstaPhoto 📸 pour décider de quels types d’URI complexes vous aurez besoin.

***Filtrez***

Et si vous vouliez que vos utilisateurs puissent rechercher des photos pour des lieux particuliers ? Faites entrer les filtres ! Vous pourriez concevoir un endpoint comme celui-ci :

GET /photos?location={locationId}

Ce  ”?” est nouveau ! Le point d’interrogation est un moyen pour l’API de savoir que vous passez un **paramètre**.

Attends – je pensais que ces valeurs étaient uniquement appelées endpoints quand elles contenaient le nom de domaine et l’URI  !

C’est une bonne remarque. Oui, les URI sont le chemin de la ressource, et un endpoint correspond au nom de domaine plus le chemin de la ressource.

***Recherchez***

La recherche vous permet de faire une requête sur une liste de ressources correspondant à votre requête de recherche. OK, alors admettons que vous vouliez avoir un endpoint où les utilisateurs peuvent rechercher toutes les photos qui ont un quelconque rapport avec le snowboard 🏂. Votre endpoint pourrait être celui-ci :

GET /photos?search=snowboard

Si vous avez déjà utilisé la fonctionnalité de recherche sur Twitter, vous avez pu constater que leur API de recherche fonctionne de façon très similaire  !

L’API de recherche de Twitter  !

Quelle est la différence entre le filtrage et la recherche, alors ? 🤔

C’est une question courante, qui dépend de la façon dont on trouve un élément. Pour filtrer, vous partez d’un élément plus petit et plus spécifique et vous collectez tous les items qui correspondent à cette valeur. Par exemple : vous allez chercher toutes les séries comiques qui commencent par la lettre C. Dans ce cas, vous filtrez.

La recherche constitue une approche plus large et élargit votre périmètre à toute chose associée à une certaine valeur.

***Triez***

Disons que vous voulez que vos utilisateurs voient tous leurs followers par ordre alphabétique ascendant et par nom de famille. Ce qui est assez complexe. Vous pouvez créer un endpoint pour cela !

GET /users/{userId}/followers?sort=lastName&order=asc

Le & est nouveau ici ! Il vous permet d’**enchaîner** différentes requêtes ensemble ! Par exemple, vous pouvez trier les utilisateurs par lastName (nom de famille) en **ordre ascendant**. Vous avez déjà vu un résultat trié quand vous naviguez sur le web. Mais si ! D’habitude, vous voulez voir les publications les plus **récentes**sur Twitter ou Instagram – ce qui constitue en fait un **tri par date**!

Vous pouvez enchaîner tous types de requêtes différentes entre elles. Par exemple, si vous voulez voir tous les utilisateurs vérifiés avec le prénom (firstName) Jamie, vous pouvez faire la requête suivante :

GET /users?verified=true&firstName=Jamie

Si vous voulez voir toutes les photos prises à New York qui ont été postées le soir du 31 décembre et qui ont plus de 5 000 likes, vous pouvez faire la requête suivante :

GET /photos?location=NYC&created\_at=2018-12-31&likes\_greater\_than=5000

Et vous pouvez encore en ajouter ! ✨

Pour que ces options fonctionnent, il vous faudra installer cette fonctionnalité en utilisant le langage de code que vous avez choisi pour l’application, car aucun d’entre eux n’est intégré à une API lorsque vous la créez. Ici, nous avons juste simplement illustré quelques-uns des moyens possibles dont vous pouvez utiliser les requêtes avancées dans votre API.

***Paginez vos résultats***

InstaPhoto 📸 a décollé et rencontre un immense succès ! 🙏 Mais vous avez maintenant des milliers de photos dans votre base de données, ce qui a rendu vos endpoints /photos très lents, au grand dam de vos utilisateurs. 😰 Que faire ? Le moyen pour corriger cela se trouve dans ce qu’on appelle la ***pagination***. Cela signifie que vos endpoints ne retournent qu’un **nombre limité** d’entrées par **numéro de page** dans votre réponse. Vous pouvez décider de combien d’items vous voulez sur chaque page, mais habituellement ce chiffre se situe entre 10 et 100.

Pour faire plus simple, /photos renvoie **toutes** les photos de votre application. Au lieu d’avoir /photos, vous devez donc préciser la page que vous voulez obtenir dans la requête :

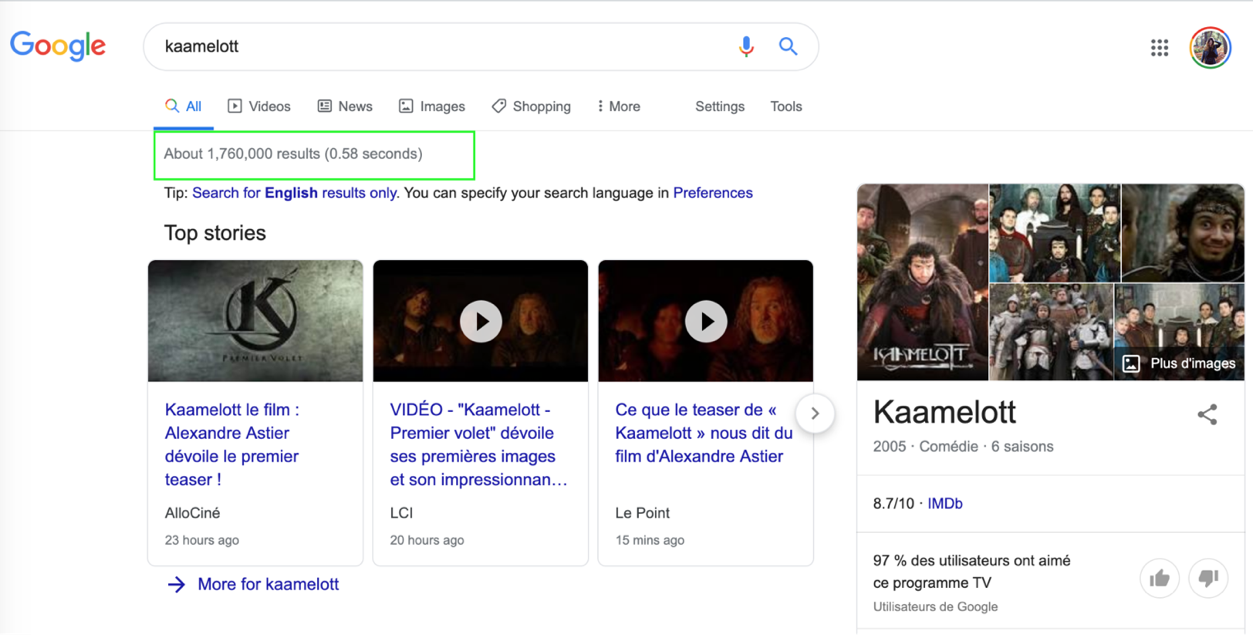
GET /photos?page=23

Ainsi, vous obtiendrez uniquement la**23e**page de photos qui en contient, disons 100 par exemple, et non la totalité d’entre elles. Le temps de chargement sera réduit et votre utilisateur sera ravi ! Pour voir les photos suivantes, on peut imaginer que votre utilisateur clique sur un bouton pour charger la suite. Et vous aurez donc :

GET /photos?page=24

Et ainsi de suite ! ✨

De la même façon, lorsque vous effectuez une recherche Google, il n’est pas rare d’obtenir des millions de résultats ; du coup, Google ne renvoie qu’une page des résultats les plus haut classés, et vous pouvez parcourir le reste pour en voir plus.



1 million de résultats lorsque l'on cherche Kaamelott sur Google  !

Google a besoin de paginer ses résultats pour ne pas surcharger votre UI.

La pagination permet aux utilisateurs de votre API de n’appeler qu’une page de résultats à la fois, pour que votre client ou votre UI ne soit pas débordé par les résultats  !

***Versionnez***

Le dernier outil de design important pour les API est le **versioning (ou *versionnage*)**. En donnant une version à vos API, vous pourrez facilement suivre les changements et vous assurer que votre application puisse être **compatible pour les versions antérieures**. Cela signifie que vous pouvez continuer à mettre à jour votre API, tout en garantissant que les utilisateurs dépendants d’une version plus ancienne puissent continuer à l’utiliser. Par exemple, si certains de vos clients utilisent déjà la version 1 de votre API, vous pouvez publier une version 2 de votre API. Alors, les utilisateurs de la version 1 de l’API peuvent continuer à l’utiliser jusqu’à ce qu’ils soient prêts à passer à la version 2. Les développeurs peuvent versionner leurs API de deux manières :

1. Ajoutez un champ **version** dans vos paramètres d’**en-tête** de requête : "accept-version": "1.3".
2. Ajoutez une**version** à votre URI : **/v1/**.

L’approche la plus courante est d’ajouter un **/v1** au début de votre URI :

GET /v1/photos

Les éléments ci-dessus sont des fonctionnalités basées sur le design et que vous devez prendre en compte pour le contenu de votre API, puisque vous devrez choisir votre propre langage ou framework. Nous aborderons ce sujet dans le dernier chapitre.

***En résumé***

* Le filtrage, la recherche et le tri sont des moyens d’ajouter de la complexité à vos requêtes API.
* La pagination aide vos clients et utilisateurs API à éviter d’être submergés par trop de données.
* Le versionnage vous permet de continuer à mettre à jour votre API sans casser le code des personnes qui en dépendent déjà.

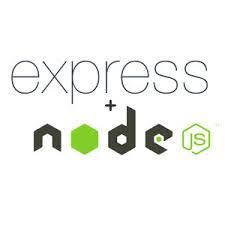
Vous venez de faire un sacré bout de chemin : vous avez construit votre première API et découvert les fonctionnalités avancées des endpoints. Mais qu’en est-il si vous souhaitez coder votre propre API ? Quels frameworks existent ? Quels langages utiliser ? Abordons cela rapidement dans le prochain et dernier chapitre de ce cours. 🎉

## [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)

***Découvrez les frameworks les plus connus pour construire une API***

Le processus de développement d’une API dépend du langage ou de l’outil de programmation utilisé. Nous avons vu les meilleures pratiques de développement pour que vous soyez préparé au mieux une fois que vous plongerez dans un langage. Voici quelques outils et frameworks populaires utilisés par les développeurs :

*Express.js (JavaScript)*



Express avec Node.js

[Express](https://expressjs.com/) utilise Node.js et le JavaScript. C’est un framework minimal et rapide. Il est très flexible et gère des applications complètes aussi bien que des API REST. Le plus gros inconvénient est qu’il n’y a pas de manière définie de faire les choses, ce qui peut être difficile pour les débutants.

#### Ruby on Rails (Ruby)



Ruby on Rails

[Ruby on Rails](https://rubyonrails.org/) est basé sur Ruby. C’est un framework populaire pour de nombreux développeurs. On le considère comme un framework magique, car derrière sa simplicité d’utilisation se cache une grande complexité. Cela aide les débutants à commencer dans le développement web plus facilement. De nombreuses librairies externes (appelées « Ruby gems ») sont disponibles, et la communauté de développeurs Rails est très vaste et met en ligne de très nombreux tutoriels. La courbe d’apprentissage de Rails devient très ardue une fois que vous plongez plus profondément dans le framework (pour comprendre la magie qui opère derrière).

*Django (Python)*

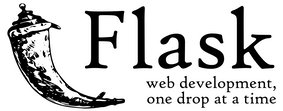


Django

[Django](https://www.djangoproject.com/) est basé sur Python. Il est utilisé par de grands noms comme Google, YouTube et Instagram. Le [framework REST de Django](https://www.django-rest-framework.org/) est facile à utiliser lorsque vous construisez vos API REST avec Django. Il demande un effort d’apprentissage aux débutants, mais possède d’excellentes fonctionnalités intégrées, comme l’authentification et la messagerie.

Vous voulez en savoir plus ? N’hésitez pas à suivre le cours [**Mettez en place une API avec Django REST framework**](https://openclassrooms.com/fr/courses/7192416-mettez-en-place-une-api-avec-django-rest-framework).

*Flask (Python)*



Flask

[Flask](http://flask.pocoo.org/) utilise Python pour le web et le développement des API REST. C’est un framework minimaliste, facile d’apprentissage et d’utilisation. Flask comprend moins de fonctionnalités intégrées que Django, mais permet aux développeurs d’avoir davantage de choix dans les outils additionnels qu’ils utilisent.

*Spring (Java)*



Spring

[Spring](https://spring.io/projects/spring-boot) est un framework web qui utilise Java, un langage très populaire. Il est utilisé par des sites web tels que Wix, TicketMaster et BillGuard. Il possède de nombreux outils liés qui boostent sa performance et vous permettent de mettre facilement votre business à l’échelle, mais il peut être difficile à prendre en main au début – surtout si vous ne maîtrisez pas Java.

Vous voulez en savoir plus ? N’hésitez pas à suivre la [**partie 3**](https://openclassrooms.com/fr/courses/6900101-creez-une-application-java-avec-spring-boot/7078007-creez-lapi-avec-les-bons-starters) du cours Créez une application Java avec Spring Boot.

*AWS API Gateway*



AWS API Gateway

[AWS API Gateway](https://aws.amazon.com/api-gateway/) et [AWS Lambda](https://aws.amazon.com/lambda/) sont des moyens de créer et d’utiliser des API REST en utilisant principalement une interface utilisateur (donc moins de code !). Ils vous permettent d’intégrer facilement votre site web avec tous les services AWS, et vous pouvez aisément scaler et gérer plus de requêtes avec leur infrastructure. Vous devez payer par requête effectuée par utilisateur à vos serveurs, mais le premier million est gratuit ! (Soyez prudent tout de même, le nombre de requêtes augmente rapidement).

Le choix de l’outil doit avant tout**dépendre des besoins de votre application**. Assurez-vous de faire les recherches nécessaires sur les frameworks et les librairies disponibles, pour choisir ce qui correspond à vos besoins. Une fois votre API construite et déployée sur le web, vous pouvez utiliser Postman comme nous l’avons fait dans l’ensemble de ce cours, pour la tester !

### ***En résumé***

* Il existe de nombreux outils et frameworks que vous pouvez utiliser pour mettre en pratique ces concepts.
* Assurez-vous de bien faire vos recherches avant de choisir quel outil API vous convient le mieux.

Bravo, vous êtes arrivé au bout de ce cours. Rendez-vous dans le chapitre suivant pour une rapide conclusion.

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

Fin de ce cours ! Vous avez appris ce qu’est une API, à utiliser une API et enfin à en concevoir une. ✨

Vous avez maintenant toutes les connaissances nécessaires pour :

* identifier la valeur des API REST pour vos projets de code ;
* formuler des requêtes et envoyer des réponses avec une API REST ;
* concevoir des endpoints pour les API REST.

Il ne vous reste plus qu’à mettre en pratique vos nouvelles compétences; tout d’abord en vous mettant au défi de passer le quiz final, puis en intégrant des API dans vos propres projets de code ! 🙌

Avant de vous laisser, voici un [lexique](Construisez+vos+projets+web+avec+les+APIs+REST_+le+lexique+.pdf) reprenant les principales notions abordées dans ce cours.

