COURS REST

5ème année Génie Informatique, Ecole Polytechnique Yaoundé

Présenté par : Ing MASSAGA Aristide

Phd student in software ingineering

Master 2 in microservice architecture

Software engineer, devops, microservice architect

https://m-aristide.github.io

[Rappels sur le HTTP 2](#_Toc1756222201)

[Généralités sur le protocole 2](#_Toc178846392)

[Principaux aspects d'HTTP 2](#_Toc1523832889)

[Les messages HTTP 3](#_Toc1059927023)

[Evolutions du protocole 4](#_Toc492466218)

[Méthodes 5](#_Toc837053096)

[Codes réponses 5](#_Toc1638208007)

[Présentation des standards REST 6](#_Toc412102971)

[Contraintes architecturales 6](#_Toc1712023403)

[Appliqué aux services web 6](#_Toc85579102)

[Pratique : Consommation d’une api rest 8](#_Toc987409131)

[Postman 8](#_Toc1748413353)

[L’outil webhook.site 10](#_Toc1923462826)

[Références 12](#_Toc1193301584)

# Rappels sur le HTTP

## Généralités sur le protocole

Hypertext Transfer Protocol (HTTP) est un protocole de la couche application servant à transmettre des documents hypermédias, comme HTML. Il a été conçu pour la communication entre les navigateurs web et les serveurs web mais peut également être utilisé à d'autres fins.

* Il suit le modèle classique client-serveur : un client ouvre une connexion, effectue une requête et attend jusqu'à recevoir une réponse.
* Il s'agit aussi d'un protocole sans état, ce qui signifie que le serveur ne conserve aucune donnée (on parle d'état) entre deux requêtes.

En raison de son extensibilité, il n'est pas seulement utilisé pour récupérer des documents, mais aussi pour des images, des vidéos ou bien pour renvoyer des contenus vers des serveurs, comme des résultats de formulaires HTML. HTTP peut aussi être utilisé pour récupérer des parties de documents pour mettre à jour à la demande des pages web.

Chaque requête individuelle est envoyée au serveur, qui la traite et fournit une réponse. Entre cette requête et la réponse se trouve de nombreuses entités qu'on désignera de façon générique sous le terme proxies. Celles-ci exécutent différentes opérations et agissent comme passerelles ou comme caches par exemple.

## Principaux aspects d'HTTP

* **HTTP est simple**

Même s'il est devenu plus complexe avec l'arrivée d'HTTP/2 et l'encapsulation des messages HTTP dans des trames, HTTP est généralement conçu pour être simple et lisible par un humain. Les messages HTTP peuvent être lus et compris par des humains, ce qui facilite les tests des développeurs et réduit la complexité pour les débutants.

* **HTTP est extensible**

À partir de HTTP/1.0, les en-têtes HTTP permettent d'étendre facilement le protocole et de mener des expérimentations avec celui-ci. De nouvelles fonctionnalités peuvent même être introduites par un simple accord entre le client et le serveur à propos de la sémantique des nouveaux en-têtes.

* **HTTP est sans état, mais pas sans session**

HTTP est sans état : il n'y a pas de lien entre deux requêtes qui sont effectuées successivement sur la même connexion. Cela devient très rapidement problématique lorsque les utilisateurs veulent interagir avec une page de façon cohérente, par exemple avec un panier d'achat sur un site de commerce en ligne. Bien que le cœur d'HTTP soit sans état, les cookies HTTP permettent l'utilisation de sessions avec des états. En utilisant l'extensibilité par les en-têtes, des cookies HTTP sont ajoutés aux flux et permettent la création d'une session sur chaque requête HTTP pour partager un même contexte, ou un même état.

* **HTTP et les connexions**

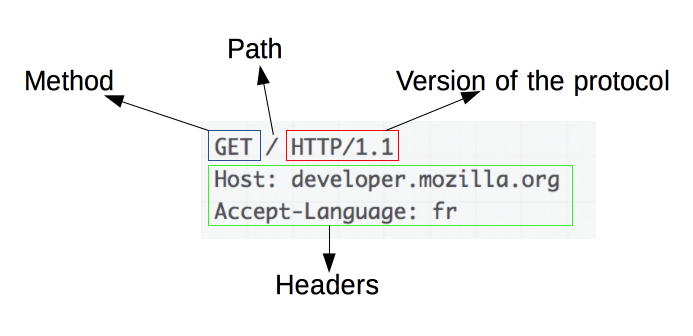
Une connexion est contrôlée au niveau de la couche transport et est donc fondamentalement hors de portée d'HTTP. Bien que HTTP ne nécessite pas un protocole de transport basé sur une connexion. Le protocole doit être fiable ou empêcher la perte de messages (donc gérer au minimum la remontée des erreurs). Parmi les deux protocoles de transport les plus courants sur Internet, TCP est fiable et UDP ne l'est pas. HTTP s'appuie sur le standard TCP, qui est basé sur la connexion, même si une connexion n'est pas toujours nécessaire.

## Les messages HTTP

Les messages HTTP/1.1 et ceux des versions précédentes d'HTTP sont lisibles par des humains. Avec HTTP/2, ces messages sont intégrés dans une nouvelle structure binaire, une trame, ce qui permet des optimisations telles que la compression des en-têtes et le multiplexage. Même si seule une partie du message HTTP d'origine est envoyée dans cette version d'HTTP, la sémantique de chaque message est inchangée et le client reconstitue (virtuellement) la requête HTTP/1.1 d'origine. Il est donc utile de comprendre les messages HTTP/2 au format HTTP/1.1.

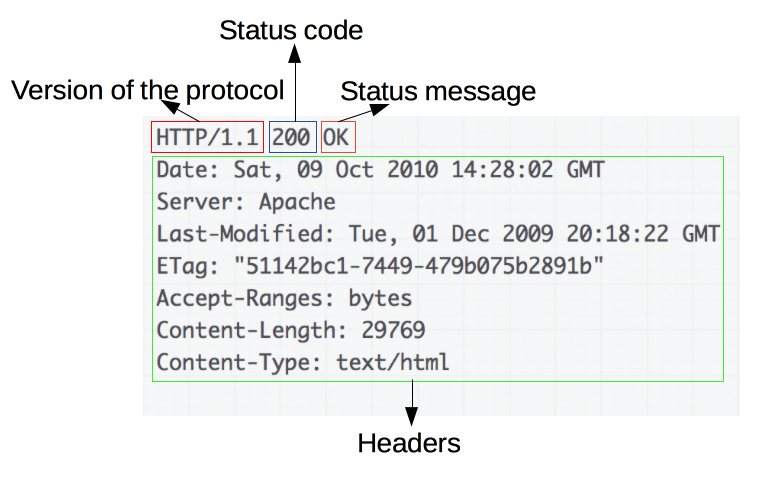
Il existe deux types de messages HTTP, les requêtes et les réponses, chacun ayant son propre format.

* **Requêtes**



Une requête comprend les éléments suivants :

* Une méthode HTTP : généralement un verbe tel que GET, POST ou un nom comme OPTIONS ou HEAD qui définit l'opération que le client souhaite effectuer. Par exemple, un client souhaite accéder à une ressource (en utilisant GET) ou téléverser le résultat d'un formulaire HTML (en utilisant POST), bien que d'autres opérations puissent être nécessaires dans d'autres cas.
* Le chemin de la ressource à extraire : l'URL de la ressource à laquelle on a retiré les éléments déductibles du contexte, par exemple le protocole (http://), le domaine (ici .mozilla.org), ou le port TCP (ici 80).
* La version du protocole HTTP.
* Les en-têtes optionnels qui transmettent des informations supplémentaires pour les serveurs.
* Ou un corps, pour certaines méthodes comme POST, semblable à ceux dans les réponses, qui contiennent la ressource envoyée.
* **Réponses**



Une réponse comprend les éléments suivants:

* La version du protocole HTTP qu'elle suit
* Un code de statut, qui indique si la requête a réussi ou non.
* Un message de statut qui est une description rapide, informelle, du code de statut
* Les en-têtes HTTP, comme pour les requêtes.
* Éventuellement un corps contenant la ressource récupérée.

## Evolutions du protocole

* **HTTP/0.9 – Le protocole une ligne**

Initialement nommé Mesh, il prit le nom de World Wide Web lors de sa mise en place en 1990. Bâti sur les protocoles existants TCP et IP, il consistait en quatre éléments de base :

* Un format textuel pour représenter les documents hypertextes, l'HyperText Markup Language (HTML).
* Un protocole simple pour échanger ces documents, l'HyperText Transfer Protocol (HTTP).
* Un logiciel client pour exposer (et modifier) ces documents, le premier navigateur web nommé WorldWideWeb.
* Un serveur pour garantir l'accès au document, version initiale du httpd.

GET /monfichier.html

<HTML>

Une page HTML très simple

</HTML>

* **HTTP/1.0 – Mise en place de l'extensibilité**

1996

* La notion d'en-tête HTTP a été mise en place à la fois pour les requêtes et pour les réponses. Elle autorise la transmission de métadonnées, et rend le protocole très flexible et extensible.
* Avec ces nouveaux en-têtes HTTP, il est désormais possible de transmettre d'autres documents que des fichiers HTML bruts (grâce à l'en-tête Content-Type.
* **HTTP/1.1 – Le protocole standardisé**
* Mise en place de mécanismes de contrôle de caches additionnels.
* Grâce à l'en-tête Host, la capacité à héberger différents domaines sur la même adresse IP autorise désormais une colocation de serveurs.

HTTP/2 – Un protocole pour plus de performances

* C'est un protocole multiplexé. Plusieurs requêtes en parallèle peuvent être gérées au sein de la même connexion, supprimant ainsi la limitation séquentielle de HTTP/1.x.
* HTTP/2 compresse les en-têtes, étant donné que des en-têtes similaires sont échangés lors d'une suite de requêtes, on supprime ainsi la duplication et l'échange inutiles des données similaires.

HTTP/3 – QUIC

* QUIC est un protocole de la couche transport. Il améliore les performances des applications Web orientées connexion qui utilisent actuellement le protocole TCP, en établissant un certain nombre de connexions multiplexées au moyen du protocole UDP (User Datagram Protocol).

## Méthodes

HTTP définit un ensemble de méthodes de requête qui indiquent l'action que l'on souhaite réaliser sur la ressource indiquée.

* **GET** : La méthode GET demande une représentation de la ressource spécifiée. Les requêtes GET doivent uniquement être utilisées afin de récupérer des données.
* **POST** : La méthode POST est utilisée pour envoyer une entité vers la ressource indiquée. Cela entraîne généralement un changement d'état ou des effets de bord sur le serveur.
* **PUT** : La méthode PUT remplace toutes les représentations actuelles de la ressource visée par le contenu de la requête.
* **DELETE** : La méthode DELETE supprime la ressource indiquée.
* **OPTIONS** : La méthode OPTIONS est utilisée pour décrire les options de communications avec la ressource visée.
* **PATCH** : La méthode PATCH est utilisée pour appliquer des modifications partielles à une ressource.

## Codes réponses

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

* Informational responses (100–199)
* Successful responses (200–299)
* Redirection messages (300–399)
* Client error responses (400–499)
* Server error responses (500–599)

Les codes réponses les plus fréquements recontrés sont :

* 200 OK : The request succeeded.
* 201 Created : The request succeeded, and a new resource created as a result.
* 301 Moved Permanently : The URL of the requested resource has been changed permanently.
* 400 Bad Request : The server could not understand the request due to invalid syntax.
* 401 Unauthorized : Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.
* 403 Forbidden : The client does not have access rights to the content;
* 404 Not Found :The server can not find the requested resource.
* 405 Method Not Allowed : The request method is known by the server but is not supported by the target resource.
* 500 Internal Server Error : The server has encountered a situation it does not know how to handle.
* 504 Gateway Timeout : This error response is given when the server is acting as a gateway and cannot get a response in time.

# Présentation des standards REST

Representational State Transfer (REST) désigne un groupe de contraintes concernant l'architecture logicielle destiné à apporter aux systèmes efficacité, fiabilité et scalabilité. Un système est appelé "RESTful" lorsqu'il adhère à ces contraintes.

## Contraintes architecturales

Six contraintes architecturales définissent un système REST. Ces contraintes restreignent la façon dont le serveur peut traiter et répondre aux requêtes du client afin que, en agissant dans ces contraintes, le système gagne des propriétés non fonctionnelles désirables, telles que la performance, l'extensibilité, la simplicité, l'évolutivité, la visibilité, la portabilité et la fiabilité.

* Client–serveur : Les responsabilités sont séparées entre le client et le serveur.
* Sans état : La communication client–serveur s'effectue sans conservation de l'état de la session de communication sur le serveur entre deux requêtes successives.
* Avec mise en cache : Les clients et les serveurs intermédiaires peuvent mettre en cache les réponses.
* En couches : Un client ne peut habituellement pas dire s'il est connecté directement au serveur final ou à un serveur intermédiaire.
* Avec code à la demande (facultative) : Les serveurs peuvent temporairement étendre ou modifier les fonctionnalités d'un client en lui transférant du code exécutable. Par exemple par des applets Java ou des scripts JavaScript.
* Interface uniforme : La contrainte d'interface uniforme est fondamentale dans la conception de n'importe quel système REST. Elle simplifie et découple l'architecture, ce qui permet à chaque composant d'évoluer indépendamment. Les quatre contraintes de l'interface uniforme sont les suivantes.
* **Identification des ressources dans les requêtes** : Chaque ressource est identifiée dans les requêtes, par exemple par un URI
* **Manipulation des ressources par des représentations** : Chaque représentation d'une ressource fournit suffisamment d'informations au client pour modifier ou supprimer la ressource.
* **Messages auto-descriptifs** : Chaque message contient assez d'information pour savoir comment l'interpréter. Par exemple, l'interpréteur à invoquer peut être décrit par un type de médias.
* **Hypermédia comme moteur d'état de l'application (HATEOAS)** : Après avoir accédé à un URI initial de l'application (de manière analogue aux humains accédant à la page d'accueil d'un site web), le client doit être en mesure d'utiliser dynamiquement les hyperliens fournis par le serveur pour découvrir toutes les autres actions possibles et les ressources dont il a besoin pour poursuivre la navigation.

## Appliqué aux services web

Les API REST basées sur HTTP sont définies par :

* Un **URI** de base, comme http://api.example.com/collection/ ;
* Des **méthodes HTTP** standards (par ex. : GET, POST, PUT, PATCH et DELETE) ;
* un type de médias pour les données permettant une transition d'état. La nature hypermédia de l'application permet d'accéder aux états suivants de l'application par inspection de la représentation courante.

# Pratique : Consommation d’une api rest

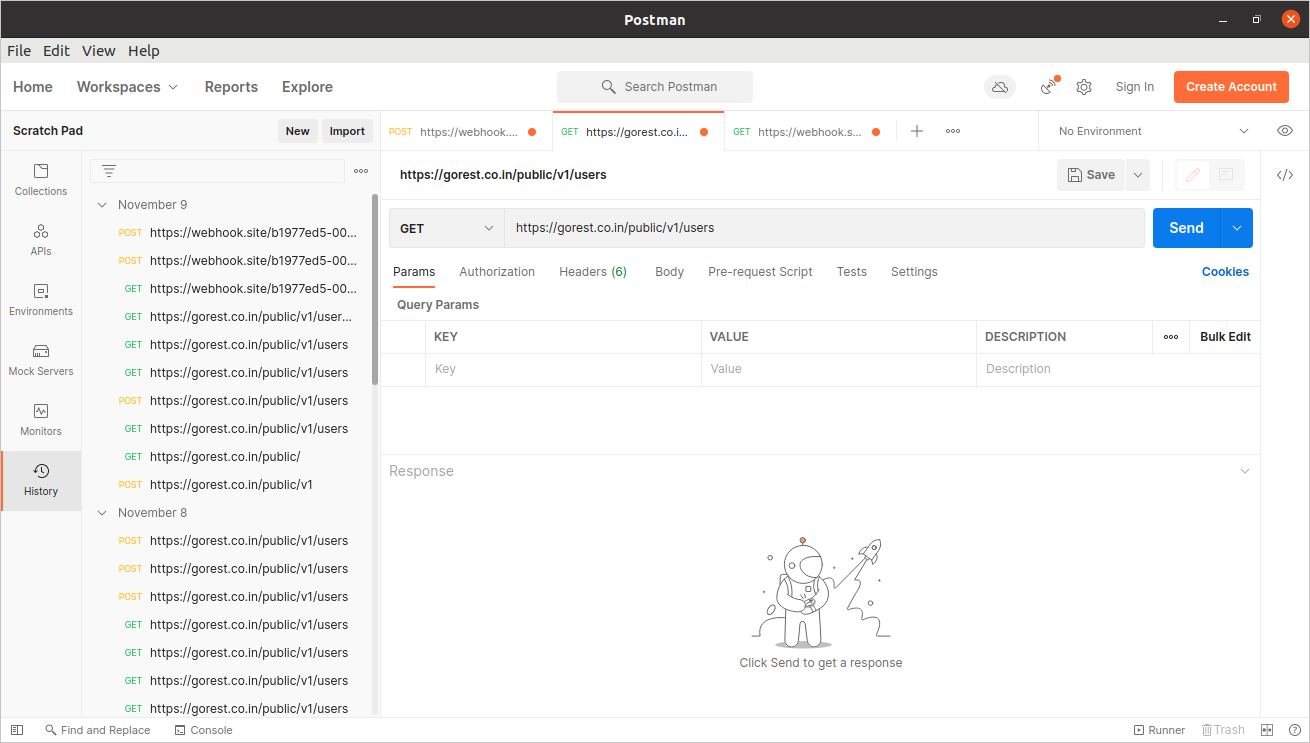
Dans cette section pratique nous allons présenter et utiliser 2 outils :

* Postman : c’est un outil graphique dédié à l’exécution de requêtes http. Il permet de tester les APIs REST construite. <https://www.postman.com/>
* webhook.site : c’est un outil de capture des requêtes http. Il donne une url unique à laquelle, en envoyant une requête http ou un mail, on peut consulter le contenu exacte de la requête envoyée. <https://webhook.site/>

## Postman

Pour les utilisateurs de windows comme ceux de linux, postman et se télécharge facilement sur le site (https://www.postman.com/). Ensuite il suffit de suivre les instructions d’installation. Il exsite aussi une version web, utilisable directement dans le navigateur.

L’interface de l’application est assez riche et offre plusieurs fonctionnalités, les principales étant :



5

4

2

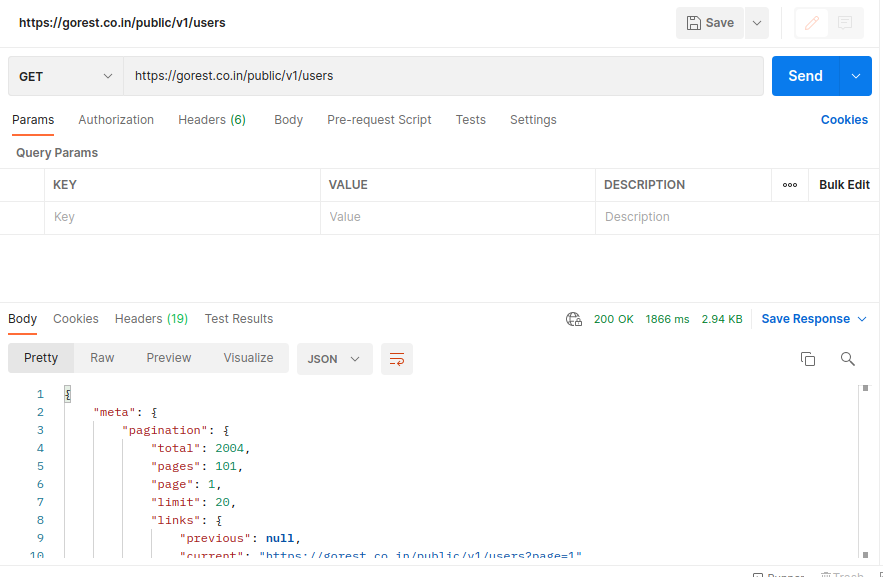
3

1

1. **Historique** : cet section permet de consulter l’historique des requêtes émises. Chaque fois qu’une requête est envoyée, elle est conservée dans cette section (les headers et les données envoyées dans la requêtes).
2. **Onglet** : on peut avoir à préparer plusieurs requêtes pour tester une API, on peut créer une requête dans chaque onglet.
3. **Zone de requête** : Ici on choisit la méthode (GET, POST, etc.) et on saisit l’url de la cible.
4. **Paramètres de la requête (headers et contenus)** : On est capable ici d’ajouter des Headers (Authorisation, etc) mais aussi des données dans la section Body.
5. **Zone de réponse** : Ici l’outil affiche la réponse qu’il reçoit du serveur, avec notamment le code réponse.

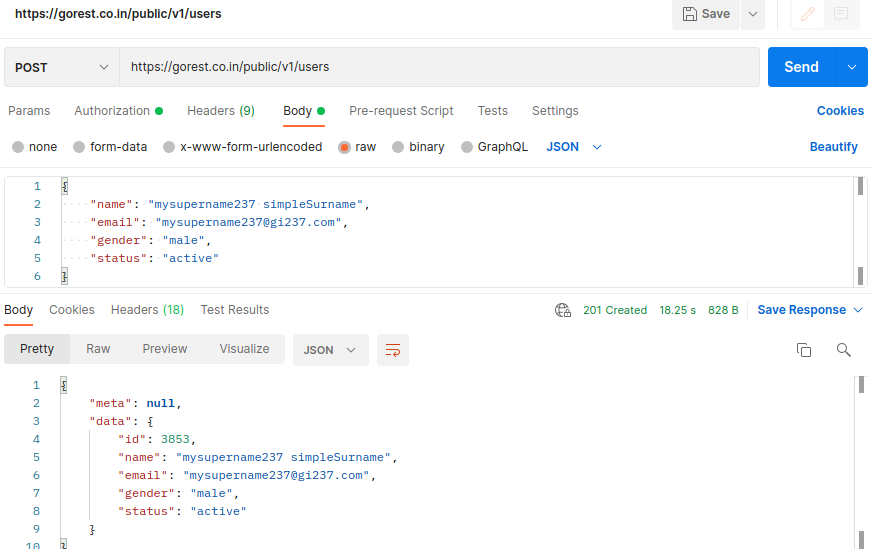
* **Exemple méthode GET**

Ici le test consiste à lire la liste des utilisateurs enregistrés.



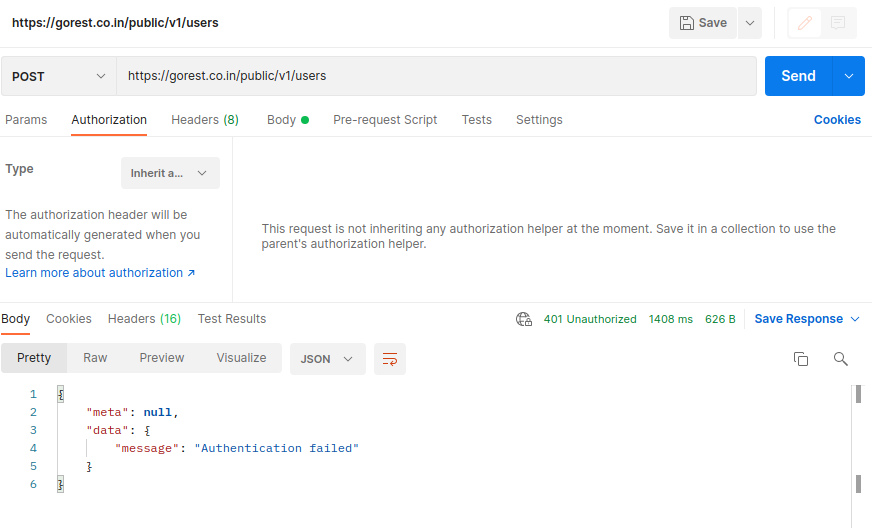
* **Exemple méthode POST**

Pour ce test nous soumettons des données de création d’un utilisateur dans le format JSON.



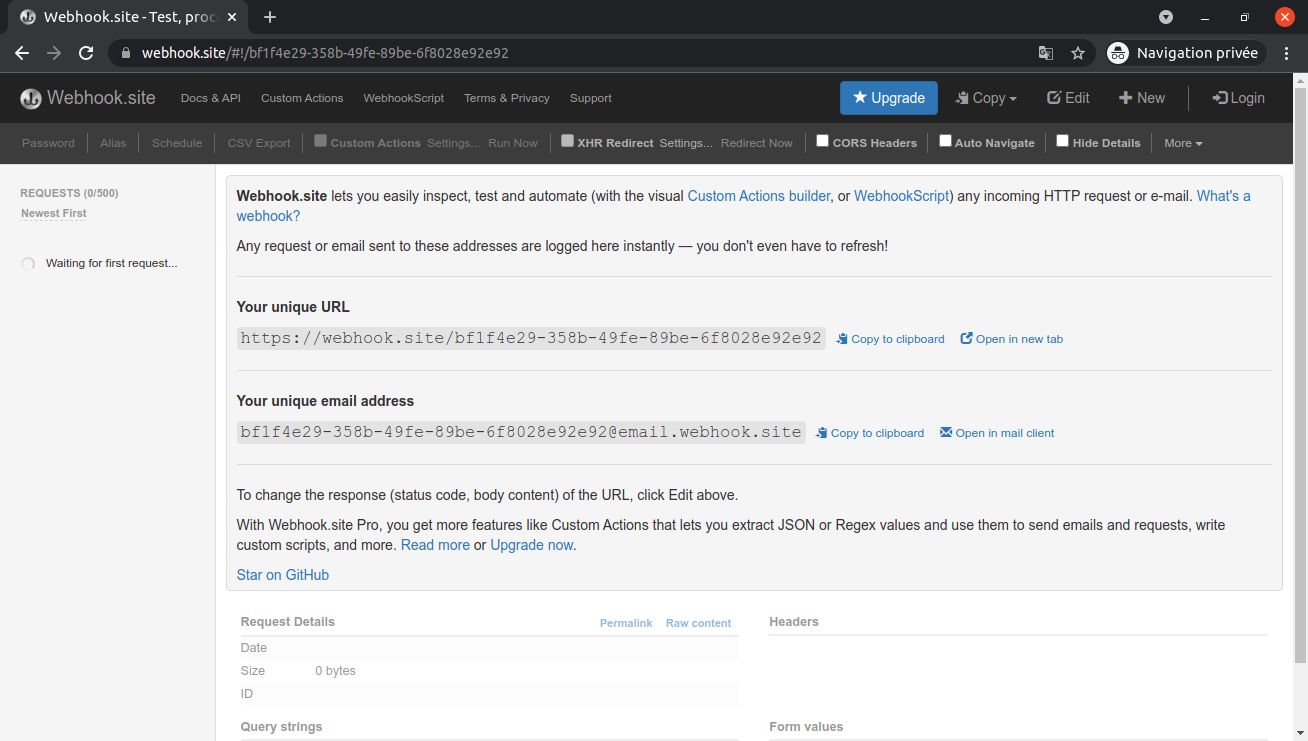
* **Tests des codes d’erreur**

Pour ce test d’erreur nous essayons d’enregistrer des données sans avoir un token d’accès à la plateforme. Dans ce cas nous recevons une erreur 401 d’accès non autorisé.



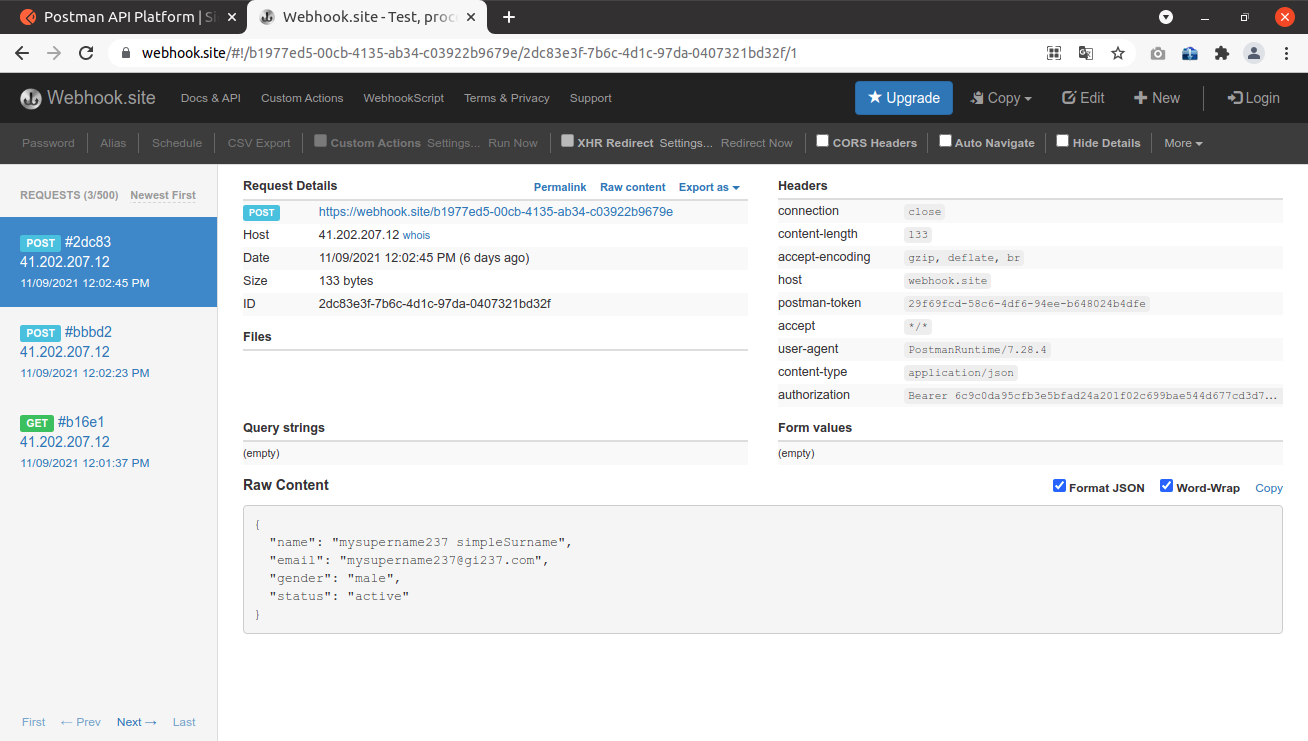
## L’outil webhook.site

Il s’utilise en ligne, à l’adresse <https://webhook.site/>. Cet outil est particulière utile lorsque qu’on n’est pas capable dire exactement ce qu’on effectue comme requête, notamment si les headers sont bien positionnés et les champs de formulaire correctement remplis et nommés, ou lorsqu’on a une erreur de code 400 relative au format des données.



Il se présente comme ci-dessus. L’information principale sur cette interface est l’url unique générée par la plateforme. En soumettant une requête à cette url, l’outil capture la réquête et la présente comme sur la figure suivante.

Ici nous pouvons voir qu’il s’agit d’une requête de type POST que nous avons effectué.



On peut voir les headers positionnés pour cette requête, notamment le header **Authorization**.

On peut voir que pour cette requête les données ont été envoyées dans le format JSON en RAW content et on peut voir le contenu exacte de ce qui a été envoyé.

# Références

« HTTP | MDN ». Consulté le 27 octobre 2021. <https://developer.mozilla.org/fr/docs/Web/HTTP>.

« Online REST API for Testing and Prototyping | GO REST ». Consulté le 8 novembre 2021. <https://gorest.co.in/>.

« Representational state transfer ». In *Wikipédia*, 29 septembre 2021. <https://fr.wikipedia.org/w/index.php?title=Representational_state_transfer&oldid=186719402>.

« Representational state transfer — Wikipédia ». Consulté le 2 novembre 2021. <https://fr.wikipedia.org/wiki/Representational_state_transfer>.

« Un Aperçu de HTTP - HTTP | MDN ». Consulté le 27 octobre 2021. <https://developer.mozilla.org/fr/docs/Web/HTTP/Overview>.