# FastAPI - Rappels sur les requêtes HTTP

30 minutes Beginner

DataScientest • com

# APIs avec FastAPI

## 2. Rappels sur les requêtes HTTP

### a. Éléments d'une requête HTTP

Le protocole HTTP est le protocole utilisé pour le Web. Créé en 1990, il permet de demander à un serveur de renvoyer certaines données ou de prendre certaines actions. Sa version sécurisée HTTPS n'est qu'une version chiffrée de HTTP.

Une requête HTTP est constituée de différents éléments.

En premier lieu, on retrouve l'URI (Universal Resource Identifier). Il s'agit de l'adresse à laquelle doit être effectuée la requête. Elle est généralement constituée du protocole utilisé, d'un nom de domaine ainsi que d'un point de terminaison (endpoint).

Par exemple, l'URI suivante `http://example.org/resource` peut se lire comme suit:

- `http://` est le protocole utilisé
- `example.org` est le nom de domaine du serveur, c'est-à-dire une simplification de l'adresse IP du serveur.
- `/resource` est le point de terminaison que l'on souhaite solliciter

On pourra noter qu'une URI peut aussi comporter des paramètres. Par exemple, l'URI `http://example.org/resource?key1=value1&key2=value2` permet de faire passer les clefs `key1` et `key2` qui ont respectivement les valeurs `value1` et `value2` au serveur. On parle alors de query string ou query parameters, traduit dans ce cours par chaîne de requête ou paramètres de requête.

Machine status

Ubuntu Server 18.04 LTS SSD Volume Type 64-bit x86

Online

34.245.135.23

Connect

Reset Stop

Une requête peut aussi contenir un corps. (En général les requêtes de type `GET` ne peuvent pas comporter de corps). Le corps d'une requête permet de faire passer des données à la requête.

Enfin une requête peut aussi comporter des en-têtes (headers). Ces en-têtes contiennent les métadonnées relatives à la requête: type des données dans le corps de la requête, cookies, token d'authentification, ...

## b. Réponses HTTP

Lorsqu'une requête HTTP est émise, le serveur renvoie une réponse au client. Cette réponse est elle aussi composée de différents éléments:

- des en-têtes avec les métadonnées de la réponse
- un corps avec le contenu de la réponse
- un code d'état

Le contenu d'une réponse HTTP est très souvent du HTML pour des sites internet mais on retrouvera plus facilement du `JSON` ou du `XML` pour les APIs.

Le code d'état permet de comprendre facilement si la requête a réussi. Par convention, les codes d'erreur doivent correspondrent aux états suivants:

- `100`: Information
- `200`: Signifie que la requête a conclu / Succès
- `300`: Redirection
- `40X`: Signifie une erreur côté client / Erreur client
- `50X`: signifie une erreur côté serveur / Erreur serveur

Ainsi, un code 404 est une erreur du client qui n'a pas rentré la bonne adresse pour accéder à la resource alors qu'une erreur 503 sera une erreur du serveur qui n'arrive pas à faire tourner le service demandé.

## c. Liens avec le web

Les sites internet fonctionnent sur ce même principe d'architecture serveur-client que l'on requête via HTTP en utilisant un navigateur Web. Ainsi, en cliquant sur ce lien, le navigateur envoie une requête de type `GET` auprès du serveur du site DataScientest. Si l'adresse est bonne, le serveur renvoie une réponse qui contient un fichier HTML qui sera interprété ensuite par le navigateur.

Lorsque l'on remplit un formulaire sur un site internet, il s'agit généralement d'une requête de type `POST`. La requête contient alors les données remplies dans le formulaire.

## d. Protocole HTTPS

Le protocole `HTTPS` une version plus sécurisée que le protocole HTTP. C'est en fait le protocole HTTP auquel est ajoutée une couche de chiffrement SSL (Secure Socket Layer). Il protège l'authentification d'un serveur, la confidentialité et l'intégrité des données échangées, et parfois l'authentification du client: une clef publique est donnée au client pour que les données renvoyées vers le serveur soient chiffrées; ces données sont alors décodées grâce à une clef privée disponible sur le

# e. Clients HTTP

Nous avons vu dans les exemples précédents, le navigateur est un client qui permet d'effectuer des requêtes HTTP vers des serveurs qui sont capables de renvoyer des données selon la requête. Toutefois, il existe d'autres outils plus simples à utiliser lorsque l'on souhaite interagir avec une API.

## CURL

Nous allons interroger une API depuis le terminal à l'aide de l'interface en ligne de commande `cURL` (client URL Request Library). Pour faire une requête HTTP depuis le terminal avec cURL, la syntaxe pour faire une requête est la suivante :

```
curl -X GET http://example.com
```

L'argument `-X` introduit la méthode, ici, `GET`. Ensuite, nous pouvons écrire l'URI. L'API que nous allons interroger est une API web factice pour développeurs. Vous pouvez consulter ici la documentation de cette API.

```
curl -X GET https://jsonplaceholder.typicc
```

Avec cette requête, nous recevons un objet JSON "stringifié" contenant des informations sur un message. Ici, nous avons demandé le post d'ID `1`. Nous pouvons obtenir tous les posts avec une requête GET au point de terminaison (endpoint) `/posts`. C'est un cas de figure que nous rencontrons couramment: le endpoint avec un ID renvoie une observation individuelle tandis que sans aucun ID, il renvoie toutes les observations.

La réponse HTTP à cette requête contient seulement le corps. Pour voir les en-têtes, nous pouvons ajouter l'argument `-i` à la commande :

```
curl -X GET -i https://jsonplaceholder.typ
```

Dans l'en-tête, vous pouvez voir des informations sur le contenu de la réponse comme par exemple le code d'état qui vaut ici `200`.

Nous avons vu comment faire une requête avec la méthode `GET`. Maintenant, si vous voulez utiliser la méthode PUT pour ajouter des données sur l'API, vous allez probablement vouloir préciser un corps et des en-têtes. Pour cela, il faut précéder vos en-têtes de l'argument `-H` et le corps de l'argument `-d`.

```
curl -X PUT -i\
 -H "Content-Type: application/json"\
 -d '{"id": 1, "content": "hello world"}'\
 https://jsonplaceholder.typicode.com/post
```

la suite du cours.

## Postman

**Postman** est une plateforme collaborative pour le développement des API. Elle permet de construire et d'exécuter des requêtes HTTP, de les stocker dans un historique afin de pouvoir les rejouer, et de les organiser en collections.

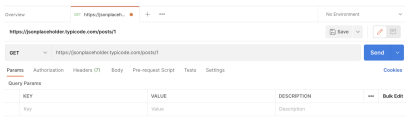
Entre autres, Postman permet :

- d'envoyer rapidement et facilement des requêtes REST, SOAP et GraphQL
- d'automatiser les tests manuels et de les intégrer dans une pipeline CI / CD pour s'assurer qu'aucune modification de code n'interrompra l'API en production.
- de communiquer le comportement attendu d'une API en simulant les points de terminaison et leurs réponses sans avoir à configurer un serveur backend.
- de générer et publier une belle documentation lisible par machine pour rendre l'API plus facile à utiliser.
- de rester à jour sur la santé de son API en vérifiant les performances et les temps de réponse à intervalles planifiés.
- de collaborer en temps réel avec le contrôle de version intégré.

Postman propose donc une interface très simple pour faire des requêtes HTTP complexes. C'est un outil très utile pour tester/explorer une API. De plus, les fonctionnalités de Postman permettent d'exporter facilement une requête effectuée avec Postman dans le langage de votre choix.

Vous pouvez télécharger Postman ici

Voici une capture d'écran correspondant à la requête d'un exemple précédent sur Postman:



## Librairies HTTP de Python

La librairie la plus simple pour effectuer des requêtes avec Python est sans doute la librairie Requests.

On pourra l'installer en utilisant le gestionnaire de package de Python

```
pip3 install requests
# or conda install requests

```

```
import requests

# creating a GET request
r = requests.get('https://jsonplacehold
```

```
response_header = r.headers
status_code = r.status_code
```

On peut bien sûr passer un corps, des en-têtes grâce à cette librairie.

```
r = requests.put(
    url='https://jsonplaceholder.typicoc
    data={"id": 1, "content": "hello wor
    headers={"Content-Type": "applicatic
    )

```

Une autre librairie populaire est la librairie Urllib installée nativement avec Python mais plus difficile à utiliser.

## e. La norme REST

Chaque API possède une architecture spécifique, ainsi que des règles à respecter qui détermine les formats de données et commandes acceptées pour communiquer avec. Afin de favoriser l'accessibilité et la standardisation des API pour les développeurs, il existe désormais des architectures d'API classiques qui sont très souvent utilisées.

Par exemple, **l'architecture REST** (Representational State Transfer) est une architecture qui est très souvent utilisée dans la création de services WEB. Elle permet aux applications de communiquer entre elles quel que soit le système d'exploitation via le protocole HTTP. Une **API REST** utilise les requêtes HTTP pour communiquer et doit respecter les principes suivants :

- **Architecture client-serveur** : le client doit faire des requêtes HTTP pour demander des ressources. Il y a indépendance entre le côté client et l'application serveur de manière à ce que les modifications apportées à un point de terminaison n'affectent pas les autres.
- Interface uniforme : architecture simplifiée et qui permet à chaque partie d'évoluer indépendamment. Pour parler d'interface uniforme, il faut respecter les 4 contraintes :
- **Identification des ressources dans les requêtes** : les ressources sont identifiées dans les requêtes et sont séparées des représentations retournées au client.
- **Manipulation des ressources par des représentations** : les clients reçoivent des fichiers qui représentent les ressources. Ces représentations doivent contenir suffisamment d'informations pour être modifiées ou supprimées.
- **Messages autodescriptifs** : tous les messages renvoyés au client contiennent assez d'informations pour décrire la manière dont celui-ci doit traiter les informations.
- **Hypermédia comme moteur du changement des états applicatifs (HATEOAS)** : après avoir accédé à une ressource, le client REST doit être en mesure de découvrir toutes les autres actions disponibles par des hyperliens.
- **Avec mise en cache** : le client doit pouvoir mettre en cache les données que l'API fournie en réponse (https://aws.amazon.com/fr/caching/)
- **Système en couches** : la communication peut s'effectuer à travers des serveurs intermédiaires (serveurs proxy ou dispositifs de répartition de charge).
- **Sans état** : aucune information n'est stockée entre deux requêtes et l'API traite ainsi chaque requête comme une première demande

suivants:

- `GET` pour récupérer des informations
- `POST` pour ajouter des nouvelles données au serveur
- `PUT` pour modifier des données déjà présentes sur le serveur
- `DELETE` pour supprimer des donnée

Cette norme n'est pas la seule et n'est pas obligatoire à mettre en oeuvre pour obtenir une API efficace mais c'est sans doute la plus connue.

Validated