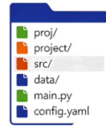
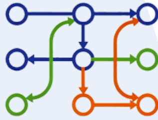
Data Storage

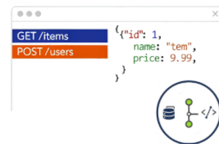
Naming Conventions



GIT Workflow



FastAPI Development



API : principes et mise en oeuvre

Fabien Baillon

02 mars 2026

Attribution - Partage dans les Mêmes Conditions : http://creativecommons.org/licenses/by-sa/4.0/fr/

Table des matières

- Objectifs
- Introduction
- API : généralités
- API RESTful : principes généraux
- Interroger une API avec Curl
    - méthode `GET`
    - méthode `POST`
    - méthode `PUT`
    - méthode `PATCH`
    - méthode `DELETE`
- Glossaire

1. Objectifs

- Connaitre les principes des **API RESTful** ;
- Savoir interroger une **API** avec **Curl**.

2. Introduction

Une ***API*** p.17 est une interface de programmation d'application, c'est-à-dire un ensemble de fonctions exécutant des actions spécifiques, et facilitant la conception et l'intégration d'applications.

La création et l'utilisation d'**API** favorisent grandement la maintenabilité des applications, notamment parce que cette approche permet une forte modularité, en définissant des actions unitaires, et permet une séparation aisée entre les traitements (*Backend* p.17) et les interfaces utilisateurs (*Frontend* p.17).

Les services publiques encouragent la création d'**API**, afin de favoriser l'interopérabilité entre les services, ainsi que l'accès aux données ouvertes.

3. API : généralités

Une *API* p.17 défini un ensemble de fonctions unitaires.

Il existe des **API** dans divers domaines d'application. Ces **API** définissent des fonctions permettant l'échange de données entre applications, ou à destination d'un utilisateur.

Une **API** permet de rendre accessible des contenus ou des données issues d'une application, d'une base de données ou d'un système de fichiers. En respectant des protocoles de communication, cette **API** favorise alors l'interopérabilité entre diverses applications, via un réseau local, ou via internet.

Complément

Le site https://api.gouv.fr/ répertorie l'ensemble des **API** en lien avec les administrations.

On peut notamment accéder ici aux **API** fournissant des données dans le domaine de l'énergie : https://api.gouv.fr/guides/api-energie

API Carto, module Codes Postaux

Exemple

L'**API** Carto module « codes postaux » permet de connaitre les communes rattachées à un code postal. Sa documentation est consultable ici :

https://api.gouv.fr/documentation/api_carto_codes_postaux

Si l'on souhaite interroge l'**API** avec le code postal 81000, il suffit de consulter la page internet :

https://apicarto.ign.fr/api/codes-postaux/communes/81000

L'**API** renvoie alors en réponse le **JSON** :

```
1 [
2     {
3         "codePostal": "81000",
4         "codeCommune": "81004",
5         "nomCommune": "Albi",
6         "libelleAcheminement": "ALBI"
7     },
8     {
9         "codePostal": "81000",
10        "codeCommune": "81144",
11        "nomCommune": "Lescure-d'Albigeois",
12        "libelleAcheminement": "ALBI"
13    },
14    {
15        "codePostal": "81000",
16        "codeCommune": "81285",
17        "nomCommune": "Sérénac",
18        "libelleAcheminement": "ALBI"
19    }
20 ]
```

4. API RESTful : principes généraux

Une **API RESTful** est une **API** qui suit les principes de l'architecture ***REST*** p.17 (*Representational State Transfer*).

Ces principes considèrent trois concepts clés :

- les **Ressources** : ce sont les éléments essentiels d'une application, c'est-à-dire les données, les objets, les services, etc ;
- les **Collections** : ce sont des groupes ou des listes de ressources ;
- les ***URI*** p.17 : des adresses uniques permettant d'accéder aux ressources et aux collections.

Exemple

Par exemple, pour une application web, on pourrait avoir :

- une donnée nommée `user`, correspondant à un utilisateur, comme ressource ;
- un ensemble d'utilisateurs, comme collection ;
- les URI `/user` et `/users` permettant de localiser cette ressource et cette collection.

Les opérations standards d'une **API REST** sont parfois regroupées dans l'acronyme **CRUD**, qui fait référence aux verbes *Create* (Créer), *Read* (Lire), *Update* (Mettre à jour) et *Delete* (Supprimer). Les opérations **CRUD** sont directement associées aux méthodes **HTTP**.

Fondamental

Le principe fondamental de **REST** est d'utiliser les méthodes standardisées du protocole **HTTP** pour manipuler des ressources :

- **`GET`** : pour récupérer une ressource ;
- **`POST`** : pour créer une nouvelle ressource ;
- **`PUT`** : pour mettre à jour une ressource existante ;
- **`DELETE`** : pour supprimer une ressource.

Complément

D'autres méthodes du protocole **HTTP** peuvent être employées :

- **`PATCH`** : pour appliquer une mise à jour partielle d'une ressource existante ;
- **`HEAD`** : pour récupérer des informations sur une ressource, sans en récupérer les données (contrairement au **`GET`**) ;
- **`OPTIONS`** : pour récupérer la liste des méthodes **HTTP** que supporte une ressource.

Dans une **API**, on définit des points de terminaison (*endpoint* p.17), pour interagir avec les ressources disponibles sur le serveur. Chaque *endpoint* correspond à une méthode standard **HTTP** pour spécifier l'action disponible pour la ressource liée à cet *endpoint*.

Exemple

Par exemple, on pourrait avoir les *endpoints* suivants :

- `GET /users` : pour récupérer la liste de tous les utilisateurs ;
- `GET /users/{userId}` : pour récupérer les détails de l'utilisateur spécifié par son ID ;
- `POST /users` : pour créer un nouvel utilisateur ;
- `PUT /users/{userId}` : pour mettre à jour les détails de l'utilisateur spécifié par son ID ;
- `DELETE /users/{userId}` : pour supprimer l'utilisateur spécifié par son ID.

5. Interroger une API avec Curl

Les **API RESTful** sont basées sur les méthodes standards **HTTP**, on peut donc utiliser l'outil **Curl** pour interagir avec ce type d'**API**. Nous allons voir ici comment utiliser **Curl** pour créer les différentes méthodes de requête HTTP.

Syntaxe

La syntaxe générale d'une commande `Curl` comporte a minima l'appel de l'exécutable `curl` et l'URL du serveur ciblé, et peut contenir différentes options :

```
1 curl [options] url
```

Syntaxe

Les options principales sont :

- `-X [Méthode_HTTP]` pour préciser la méthode standard à utiliser ;
- `-H [En_tête_HTTP]` pour préciser au serveur comment interpréter les données qui lui sont envoyées ;
- `-d [Données]` pour préciser les données à envoyer au serveur.

5.1. méthode GET

C'est la méthode par défaut pour **Curl**, on n'est donc pas obligé de la préciser.

Syntaxe

```
1 curl -X GET url
```

Exemple

Le serveur https://jsonplaceholder.typicode.com/ propose une **API** de test qui comportent différents articles (*posts*). On peut récupérer l'ensemble des articles par la commande :

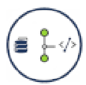
```
1 curl -X GET https://jsonplaceholder.typicode.com/posts/
```

Ou un article spécifique, le numéro 1 par exemple, par la commande :

```
1 curl -X GET https://jsonplaceholder.typicode.com/posts/1
```

Remarque

Dans les deux cas, le serveur renvoie en réponse des données `JSON`.



Exemple

On peut utiliser **Curl** pour interroger l'**API** Carto module « codes postaux » :

```
curl -X GET https://apicarto.ign.fr/api/codes-postaux/communes/81000 | jq .

```

Remarque

L'option `-o` permet de spécifier le nom d'un fichier dans lequel enregistrer le résultat de la requête **Curl**.

Exemple

```
curl -o 81000.json -X GET https://apicarto.ign.fr/api/codes-postaux/communes/81000
cat 81000.json | jq .
```

5.2. méthode POST

La méthode `POST` sert à envoyer des données, il faut donc préciser le type de données envoyées dans une option `-H` et les données elles-mêmes avec une option `-d`

Syntaxe

```
curl -X POST -H [En_tête_HTTP] -d [Données] url
```

Exemple

```
curl -X POST -H 'Content-Type: application/json' -d '{"title": "foo","body": "bar","userId": 123}' https://jsonplaceholder.typicode.com/posts
```

Si les données sont déjà dans un fichier `data.json`, on peut pointer vers ce fichier avec le symbole @, et la syntaxe devient :

```
curl -X POST -H 'Content-Type: application/json' -d @data.json https://jsonplaceholder.typicode.com/posts
```

Dans les deux cas, le serveur renvoie en réponse des données `JSON`.

5.3. méthode PUT

La méthode `PUT` sert à mettre à jour des données, il faut donc préciser le type de données envoyées dans une option `-H` et les données elles-mêmes avec une option `-d`.

Syntaxe

```
curl -X PUT -H [En_tête_HTTP] -d [Données] url
```

Exemple

Par exemple, pour mettre à jour l'article 3 :

```
1 curl -X PUT -H 'Content-Type: application/json' -d '{"title":
  "foo_updated","body": "bar_updated","userId": 123}'
  https://jsonplaceholder.typicode.com/posts/3
```

Si les données sont déjà dans un fichier `data.json`, on peut pointer vers ce fichier avec le symbole @, et la syntaxe devient :

```
1 curl -X PUT -H 'Content-Type: application/json' -d @data.json
  https://jsonplaceholder.typicode.com/posts/3
```

Dans les deux cas, le serveur renvoie en réponse des données `JSON`.

5.4. méthode `PATCH`

La méthode `PATCH` sert à mettre à jour des données partiellement, il faut donc préciser le type de données envoyées dans une option `-H` et les données elles-mêmes avec une option `-d`.

Syntaxe

```
1 curl -X PATCH -H [En_tête_HTTP] -d [Données] url
```

Les données à mettre à jour peuvent être dans un fichier, et la syntaxe sera alors :

```
1 curl -X PATCH -H [En_tête_HTTP] -d @[fichier_de_Données] url
```

Exemple

Par exemple, pour mettre à jour l'article 1, en spécifiant un nouveau champ `body` :

```
1 curl -X PATCH -H 'Content-Type: application/json' -d '{"body": "nouveau
  contenu"}' https://jsonplaceholder.typicode.com/posts/1
```

5.5. méthode `DELETE`

La méthode `DELETE` sert à supprimer des données :

Syntaxe

```
1 curl -X DELETE url
```

Exemple

Par exemple, si l'on souhaite supprimer l'article 3 :

```
1 curl -X DELETE https://jsonplaceholder.typicode.com/posts/3
```

Glossaire

API
Backend
endpoint
Frontend
REST
URI