

Les 6 contraintes REST

REST (*REpresentational State Transfer*) n'est ni un protocole ni une norme : c'est un **style d'architecture**, décrit par Roy Fielding en 2000. Une API est dite **RESTful** quand elle respecte ses contraintes.

Pour les illustrer, on reprend l'analogie du restaurant, et l'API publique de GitHub.

REST est décrit par 6 contraintes : [article Wikipédia](#).

1 - Client-serveur

Le client (l'interface) et le serveur (les données, la logique) sont séparés et évoluent indépendamment. Tant que le contrat est respecté, on peut refaire tout le front sans toucher au back, ou l'inverse.

Au restaurant : vous ne passez pas en cuisine. Le chef peut changer de four sans que vous changiez votre façon de commander.

2 - Sans état (*stateless*)

Chaque requête contient **tout** ce qu'il faut pour être traitée. Le serveur ne garde aucun contexte de session entre deux requêtes.

```
curl -H "Authorization: Bearer mon-token" https://api.github.com/user
```

Au restaurant : le serveur a une mémoire de poisson rouge. Rappelez-lui votre numéro de table à chaque commande.

Avantage : n'importe quel serveur peut répondre à n'importe quelle requête, ce qui facilite la montée en charge.

3 - Cache

Les réponses indiquent si elles peuvent être mises en cache, et combien de temps. Le client (ou un proxy) peut alors réutiliser une réponse sans redemander.

```
curl -i https://api.github.com/users/thibaudfrichet
```

HTTP/2 200

cache-control: public, max-age=60, s-maxage=60

Ici, GitHub autorise à garder la réponse 60 secondes.

Au restaurant : la carte ne change pas toutes les 5 minutes. Pas besoin de la redemander à chaque fois.

4 - Interface uniforme

La contrainte centrale de REST. Toutes les ressources se manipulent de la même manière :

- chaque ressource est **identifiée** par une URI : `/users/thibaudfrichet`, `/repos/thibaudfrichet/homelab-agent` ;
- on la manipule à travers ses **représentations** (JSON, XML...) ;
- les messages sont **autodescriptifs** : méthode, `Content-Type`, code de statut ;
- les réponses contiennent des **liens** vers les actions possibles : c'est **HATEOAS**, vu au chapitre 9.

Conséquence : quand on connaît une API REST, on sait déjà à moitié utiliser toutes les autres.

5 - Système en couches

Le client ne sait pas (et n'a pas besoin de savoir) s'il parle directement au serveur ou à un intermédiaire : reverse proxy, load balancer, CDN, cache, passerelle d'API...

6 - Code à la demande (facultatif)

Le serveur peut envoyer du code exécutable au client, typiquement du JavaScript. C'est la seule contrainte facultative, et la moins utilisée dans les API.

La plupart des API qui se disent « REST » n'en respectent pas toutes les contraintes, en particulier HATEOAS. Fielding s'en plaint publiquement depuis 2008. Dans la pratique, « API REST » veut souvent dire « API HTTP + JSON avec des URL propres ».

Revision #5

Created 2 October 2026 20:00:41 by Thibaud FRICHET

Updated 11 October 2026 20:15:41 by Thibaud FRICHET