
{ REST-API }

Sommaire

1/ Histoire de REST

Le terme REST a été introduit par **Roy Fielding**, l'un des créateurs du protocole HTTP, dans son doctorat. Thèse intitulée "Architectural Styles and the Design of Network-based Software Architectures" en 2000.

2/ Introduction aux API REST

Qu’est-ce que REST ?

REST (Representational State Transfer) est un style d’architecture qui définit comment les API doivent être conçues pour être simples et efficaces.

- Les API REST utilisent HTTP pour envoyer et recevoir des données entre un client et un serveur
- REST est basé sur des ressources accessibles via des URL (Uniform Resource Locator)

Pourquoi REST est intéressant ?

- **Simplicité** : Utilise des protocoles web standards
- **Interopérabilité** : Fonctionne avec différents langages et plateformes
- **Scalabilité** : Facile à adapter à un grand nombre d’utilisateurs

3/ Principes fondamentaux

Les 5 règles clés qui définissent REST:

1/ Architecture client-serveur

- Séparation entre le client (application utilisateur) et le serveur (qui gère les données).

2/ Stateless (Sans état)

- Chaque requête envoyée au serveur doit contenir toutes les informations nécessaires à son traitement.
- Le serveur ne garde pas d'historique des interactions du client

3/ Cache

- Les réponses peuvent être mises en cache pour améliorer les performances.

4/ Interface uniforme

- L’API doit respecter des conventions communes pour être compréhensible et facile à utiliser
- Utilisation des méthodes HTTP standard :
  - **GET** → Récupérer
  - **POST** → Ajouter
  - **PUT** → Mettre à jour
  - **DELETE** → Supprimer

5/ Système en couches

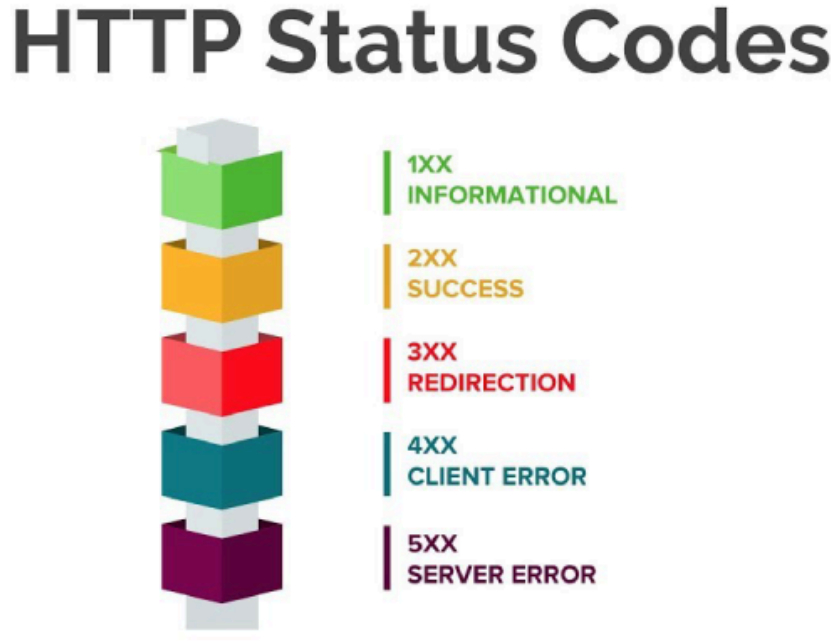
- Possibilité d’ajouter des proxys et des bases de données intermédiaires
- Cela améliore la sécurité et les performances

4/ Fonctionnement

Les verbes HTTP

Chaque requête HTTP contient :

- Une **méthode** (GET, POST, PUT, DELETE)
- Une **URL**
- Des **en-têtes** (Headers) contenant des informations comme l'authentification.
- Un **corps de requête** (Body) (pour POST et PUT).



5/ Sécurisation

Pourquoi sécuriser une API ?

Protection des données utilisateur et éviter les cyberattaques

Authentification et Autorisation

- API Keys → Un jeton unique pour chaque utilisateur
- OAuth 2.0 → Authentification sécurisée utilisée par Google, Facebook.
- JWT (JSON Web Token) → Jeton d'authentification envoyé dans chaque requête.

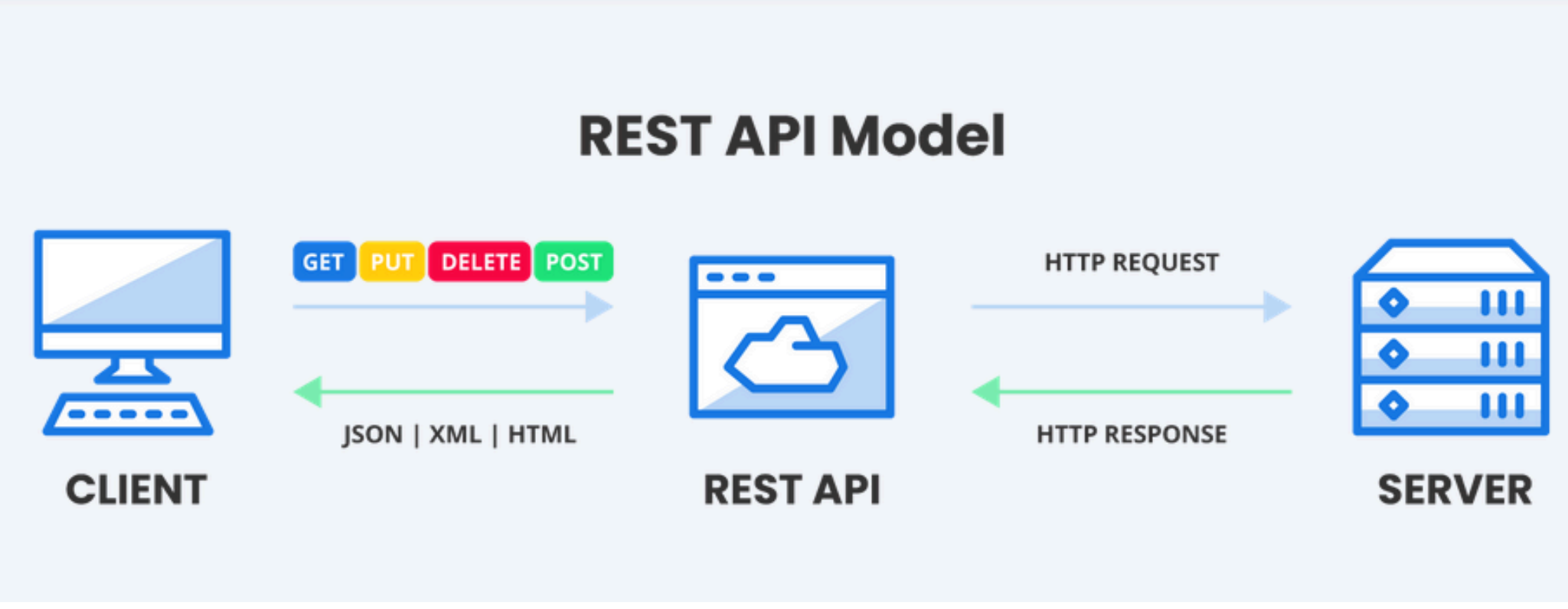
Sécurisation des échanges

- Toujours utiliser HTTPS (https:// au lieu de http://)

Protection contre les abus

- Rate Limiting → Limiter le nombre de requêtes par minute.

6/ Schéma


REST API Model
GET
PUT
DELETE
POST
JSON | XML | HTML
HTTP REQUEST
HTTP RESPONSE
CLIENT
REST API
SERVER