# Types d'API

## Sommaire



## Pourquoi plusieurs types d’API ?

Toutes les API ne sont pas construites de la même façon. Leur **architecture dépend du besoin** : flexibilité, robustesse, compatibilité avec des systèmes anciens, performance, etc.

Trois grands types d’API sont couramment rencontrés :

# REST – REpresentational State Transfer

REST est aujourd'hui **le standard dominant pour les APIs web**. Proposé par Roy Fielding dans sa thèse de 2000, il repose sur **une architecture orientée ressources**.

## 🔧 Principes clés

- Chaque **ressource** est identifiée par une **URL** Exemple : `/users/42` fait référence à l'utilisateur n°42
- On utilise les **méthodes HTTP** standards :
  - `GET /users` → liste d'utilisateurs
  - `POST /users` → créer un utilisateur
  - `PUT /users/42` → remplacer un utilisateur
  - `PATCH /users/42` → mise à jour partielle
  - `DELETE /users/42` → suppression
- Les **données échangées** sont généralement en **JSON**
- L'échange est **stateless** (chaque requête est indépendante)

## ☑ Avantages

- Simple à comprendre et à implémenter
- Lisible et standardisé (très bien supporté)
- Compatible avec de nombreux outils (navigateurs, Postman, curl…)

# GraphQL – Une approche flexible (aperçu)

GraphQL est une alternative moderne à REST, développée par Facebook.

## 🔧 Fonctionnement

- Une **seule URL** (`/graphql`) pour toutes les requêtes
- Les clients peuvent **définir précisément les champs** qu'ils veulent
- Les **requêtes sont structurées** en JSON ou syntaxe proche

```
query {
  user(id: 42) {
    name
    email
    posts {
      title
    }
  }
}
```

## ☑ Avantages

- Évite le *overfetching* ou *underfetching* (on ne récupère que ce qu'on demande)
- Très pratique pour des interfaces frontend complexes (ex. apps mobiles)

## ❌ Inconvénients

- Courbe d'apprentissage plus raide
- Plus complexe à implémenter côté serveur
- Moins adapté pour des APIs simples

# SOAP – Protocole formel (aperçu)

SOAP (Simple Object Access Protocol) est une ancienne norme d'API basée sur **XML**.

## 🔧 Caractéristiques

- Nécessite un **format XML rigide** (schéma XSD, WSDL…)
- Basé sur **HTTP** ou **d'autres protocoles** (SMTP par exemple)
- Très structuré, avec des règles strictes d'échange

## ☑ Utilisé dans

- Les **systèmes bancaires**
- Les **applications d'entreprise anciennes** (ERP, etc.)

## ❌ Limitations

- Verbeux et complexe
- Moins flexible que REST ou GraphQL

# Pourquoi on choisit REST dans ce cours ?

- **Simple et accessible** à tous les niveaux
- Facile à **documenter automatiquement** (ex. avec FastAPI ou Swagger)
- Adapté à la **majorité des cas d'usage web et mobile**
- Parfait pour une **progression pédagogique claire** (URLs, méthodes HTTP, JSON…)

## 🔍 Exemple REST

> Requête HTTP pour récupérer un utilisateur :

```
GET /users/1 → Renvoie les informations de l'utilisateur avec l'ID 1
```

> Réponse JSON typique :

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```