Concevoir une API RESTful

Support de formation pour la session “Concevoir une API RESTful” d’Octobre 2026.

Formateur : Dufrène Valérian (Webdevoo Formation)

"Déclaration d'activité enregistrée sous le numéro 32800232680 auprès du préfet de région HAUTS-DE-FRANCE"

Table des matières

L’histoire des APIs..... 4
Les origines : pourquoi les APIs ont-elles été créées ?..... 4
L'utilité initiale..... 4
De l'ordinateur local au Web mondial..... 5
La révolution REST (2000)..... 5
Le paysage en 2026..... 6
Fondations, architecture et modélisation REST..... 7
Comprendre le rôle d'une API et le standard REST..... 7
Définition et rôle d'une API distante..... 7
L'agnosticisme technologique..... 7
Les 6 principes fondamentaux d'une API RESTful..... 8
Ressources, verbes HTTP et démarrage du projet fil rouge..... 9
La structuration en ressources..... 9
Les verbes HTTP et l'idempotence..... 9
Les concepts clés : sécurité et idempotence..... 9
Détail des verbes..... 9
TP fil rouge - Étape 1 : modélisation du domaine "e-commerce"..... 10
Conception avancée, données et hypermédias..... 11
Manipulation des données et contrats de communication..... 11
Les conventions de nommage et de routage..... 11
Les Codes de Statut HTTP (Le langage du serveur)..... 12
2xx - Succès..... 12
4xx - Erreurs du Client..... 12
5xx - Erreurs du Serveur..... 12
Pagination, filtrage et tri des collections..... 13
Filtrage et tri (Query Parameters)..... 13
Pagination : la fin de l'offset..... 13
Standardisation des erreurs et HATEOAS..... 14
La gestion des erreurs : la RFC 7807 (Problem Details)..... 14
HATEOAS (Hypermedia as the Engine of Application State)..... 15
L'approche pragmatique de 2026..... 15
Exemple : Le cycle de vie d'une commande..... 15
TP fil rouge - Étape 2 : design JSON et machine à états..... 16
Sécurité, documentation et validation..... 17

Sécuriser et documenter (OpenAPI)..... 17
Principes de sécurité des APIs RESTful..... 17
Documenter avec OpenAPI (OAS)..... 17
Le Standard de l'industrie : OpenAPI v3.1 / v3.2..... 17
Les avantages de l'approche "Design-First" :..... 17
Maintenabilité, versioning et clôture..... 18
Stratégies de versioning..... 18
TP fil rouge - Étape 3 : spécification OpenAPI..... 19
Évaluation de connaissances - Fin de session..... 20

L’histoire des APIs

Bienvenue dans cette formation dédiée à l'architecture et la conception d'APIs RESTful. Avant de plonger dans les concepts techniques qui régissent notre monde interconnecté, il est essentiel de comprendre d'où nous venons.

Les origines : pourquoi les APIs ont-elles été créées ?

L'acronyme API (Application Programming Interface, ou Interface de Programmation d'Application) n'est pas né avec le Web.

L'idée fondatrice remonte aux années **1950**. Les informaticiens britanniques **Maurice Wilkes et David Wheeler**, travaillant sur le premier ordinateur à programme enregistré (EDSAC), ont créé ce qui est considéré comme la première documentation d'API.

À l'époque, les sous-routines de code étaient stockées sur des bandes perforées dans des tiroirs. L'API était littéralement un catalogue papier expliquant comment intégrer ces bandes pour utiliser une fonctionnalité sans avoir à réécrire le code machine.

L'utilité initiale

Le but premier était simple : **l'abstraction et la réutilisabilité**.

Permettre à des programmes de communiquer avec le matériel, ou avec d'autres programmes, sans avoir besoin d'en connaître la complexité interne.

L'API sert de "contrat" clair entre le fournisseur d'un service et son consommateur.

De l'ordinateur local au Web mondial

Au fil des décennies, le concept a évolué avec l'informatique réseau.

Dans les années 1990, des protocoles lourds et complexes comme **CORBA** ou **SOAP** dominaient les échanges entre serveurs.

Ils étaient puissants mais rigides et difficiles à maintenir.

La révolution REST (2000)

L'année 2000 marque un tournant historique pour deux raisons majeures :

1. **Février 2000 :** Lancement de la première API Web commerciale moderne par **Salesforce**, prouvant que les données d'entreprise pouvaient être échangées de manière fluide sur Internet.
2. **Fin 2000 : Roy Fielding**, l'un des principaux auteurs de la spécification HTTP, publie sa thèse de doctorat. Il y décrit un style architectural nommé **REST (Representational State Transfer)**.

La proposition de Fielding était révolutionnaire : pourquoi inventer des protocoles complexes par-dessus Internet, alors que le Web possède déjà tout ce dont nous avons besoin (HTTP, URIs, mécanismes de cache) ?

REST propose d'utiliser les standards existants du Web de manière sémantique.

Le paysage en 2026

Aujourd'hui, en 2026, l'économie numérique repose sur l'API Economy.

Bien que d'autres technologies comme GraphQL ou gRPC soient apparues pour des cas d'usage spécifiques, REST demeure le standard absolu et universel pour exposer des services web, grâce à sa simplicité, sa scalabilité et son utilisation naturelle du protocole HTTP.

Fondations, architecture et modélisation REST

Comprendre le rôle d'une API et le standard REST

Définition et rôle d'une API distante

Une API distante est un ensemble de règles et de protocoles permettant à différentes applications logicielles de communiquer entre elles via un réseau.

Elle agit comme un intermédiaire, un "serveur de restaurant" qui prend la commande, la transmet en cuisine, et ramène le plat.

L'agnosticisme technologique

Le rôle majeur d'une API distante moderne est de découpler le client du serveur.

Le client (application mobile, site web) se moque totalement de savoir si le serveur est codé en Java, Python, ou Go. Ils communiquent grâce à un langage commun (souvent le format JSON) via le protocole HTTP.

Les 6 principes fondamentaux d'une API RESTful

Pour qu'une architecture soit véritablement considérée comme "RESTful", elle doit respecter strictement ces 6 contraintes :

1. **Client-Serveur :** Le client s'occupe de l'interface utilisateur, le serveur s'occupe du stockage et de la logique métier.
2. **Sans état (Stateless) :** Le serveur ne doit conserver aucun contexte client entre les requêtes. Chaque requête doit contenir toutes les informations nécessaires (y compris l'authentification).
3. **Mise en cache (Cacheable) :** Les réponses du serveur doivent explicitement se déclarer comme pouvant être mises en cache ou non.
4. **Système en couches (Layered System) :** Le client ne peut pas savoir s'il est connecté directement au serveur final ou à des intermédiaires (Load Balancer, Sécurité).
5. **Code à la demande (Optionnel) :** Le serveur peut envoyer du code exécutable au client (rare).
6. **Interface Uniforme :** C'est le cœur de REST. Elle s'appuyant sur l'identification des ressources.

Ressources, verbes HTTP et démarrage du projet fil rouge

La structuration en ressources

En REST, tout est ressource.

Une ressource est un objet ou une entité (un Utilisateur, un Produit).

Contrairement aux architectures basées sur des actions (**/creerUtilisateur**), REST utilise des noms, toujours au pluriel, pour identifier la ressource (ex: **/users**).

Les verbes HTTP et l'idempotence

L'action souhaitée sur la ressource est définie par le verbe HTTP.

Les concepts clés : sécurité et idempotence

- **Safe (Sûr) :** Une méthode est sûre si elle ne modifie pas l'état du serveur (lecture seule).
- **Idempotent :** Une méthode est idempotente si l'exécuter une fois a exactement le même effet que de l'exécuter 1000 fois de suite.

Détail des verbes

- **GET :** Récupérer une ressource. *(Sûr et idempotent)*.
- **POST :** Créer une nouvelle ressource. *(Ni sûr, ni idempotent)*.
- **PUT :** Remplacer **intégralement** une ressource existante. *(Non sûr, mais idempotent)*.
- **PATCH :** Modifier **partiellement** une ressource. *(Non sûr, non idempotent par défaut)*.
- **DELETE :** Supprimer une ressource. *(Non sûr, mais idempotent)*.

TP fil rouge - Étape 1 : modélisation du domaine "e-commerce"

Objectif : Poser les fondations de notre projet (API pour une plateforme e-commerce B2C).

Travail à réaliser en groupes :

1. Identifiez les entités principales du domaine (Ressources).
2. Définissez sur papier les URIs de base pour accéder à ces ressources et à leurs sous-ressources.
3. Listez les verbes HTTP autorisés pour chaque endpoint identifié.
   Livrable attendu : Un tableau croisant URIs et Verbes HTTP.

Conception avancée, données et hypermédias

Manipulation des données et contrats de communication

Les conventions de nommage et de routage

La conception de l'URL est la vitrine de votre API. En 2026, les standards sont stricts :

- **Kebab-case :** Utilisez des tirets pour séparer les mots (/product-categories).
- **Hiérarchie limitée :** Ne dépassez pas deux niveaux de profondeur.
    - ✅ **Bien :** GET /users/123/orders.
    - ❌ **Mauvais :** GET /users/123/orders/456/items (Préférer GET /orders/456/items).

Les Codes de Statut HTTP (Le langage du serveur)

Renvoyer **200 OK** pour tout (même des erreurs) est une antipattern.

Utilisez les statuts :

2xx - Succès

200 OK, **201 Created** (Ressource créée, souvent avec un POST), **204 No Content** (Succès, typique pour DELETE).

4xx - Erreurs du Client

400 Bad Request (Syntaxe invalide), **401 Unauthorized** (Non authentifié), **403 Forbidden** (Droits insuffisants), **404 Not Found**, **422 Unprocessable Entity** (Règle métier violée).

5xx - Erreurs du Serveur

500 Internal Server Error, **503 Service Unavailable**.

Pagination, filtrage et tri des collections

Filtrage et tri (Query Parameters)

Utilisez les paramètres d'URL : **GET /products?category=shoes&sort=-price,name**.

Pagination : la fin de l'offset

En 2026, la pagination par curseur (Cursor-based) est la norme pour les APIs professionnelles : **GET /products?cursor=eyJpZCI6MTIzfQ==&limit=50.**

Elle surpasse la pagination par pages (**?page=2**) en termes de performances sur les grandes bases de données.

Standardisation des erreurs et HATEOAS

La gestion des erreurs : la RFC 7807 (Problem Details)

Le standard de l'industrie est la spécification **"Problem Details for HTTP APIs"**.

L'API renvoie une structure JSON normalisée :

```
{
  "type": "https://api.monsite.com/errors/out-of-stock",
  "title": "Produit en rupture de stock",
  "status": 409,
  "detail": "L'article 'Chaussures X' n'a plus que 0 unité en stock.",
  "instance": "/orders/cart/123"
}
```

HATEOAS (Hypermedia as the Engine of Application State)

C'est l'aboutissement (Niveau 3 de Richardson) d'une API REST.

Le client navigue dans l'API grâce aux liens fournis par le serveur.

L'approche pragmatique de 2026

Aujourd'hui, l'industrie utilise HATEOAS principalement pour gérer les **Machines à États**.

Le serveur dicte les actions possibles selon le statut d'une ressource.

Exemple : Le cycle de vie d'une commande

```
{
  "id": "ord_890",
  "status": "pending_payment",
  "total": 150.00,
  "_links": {
    "self": { "href": "/orders/ord_890", "method": "GET" },
    "pay": { "href": "/orders/ord_890/pay", "method": "POST" },
    "cancel": { "href": "/orders/ord_890", "method": "DELETE" }
  }
}
```

*Si la commande passe au statut **shipped**, les liens **pay** et **cancel** disparaissent dynamiquement de la réponse.*

TP fil rouge - Étape 2 : design JSON et machine à états

Objectif : Designer le payload de nos ressources.

Exercice :

1. Rédigez la structure JSON attendue pour la création d'une commande (POST).
2. Rédigez la réponse d'erreur formatée en RFC 7807 pour une erreur métier.
3. Dessinez la machine à états d'une commande (Panier -> Validé -> Payé -> Expédié).
4. Rédigez les JSON des réponses HATEOAS pour deux étapes de ce cycle de vie.

Sécurité, documentation et validation

Sécuriser et documenter (OpenAPI)

Principes de sécurité des APIs RESTful

- **Chiffrement :** HTTPS (TLS) obligatoire.
- **Authentification Stateless (JWT) :** On utilise des Jetons Web JSON (**JWT**) transmis dans l'en-tête **Authorization: Bearer <token>**, car les sessions serveurs sont interdites en REST.
- **Autorisation :** OAuth2 / OIDC.
- **Rate Limiting :** Protéger l'API en limitant le nombre de requêtes par IP (**statut 429 Too Many Requests**).

Documenter avec OpenAPI (OAS)

Une API non documentée est inutilisable.

Le Standard de l'industrie : OpenAPI v3.1 / v3.2

Format déclaratif (JSON ou YAML) permettant de décrire formellement l'API.

Les avantages de l'approche "Design-First" :

1. **Portail Développeur :** Génération de documentation web interactive (Swagger UI).
2. **Génération de code :** Création automatique de SDKs clients.
3. **Tests automatisés :** Génération de collections de tests.

Maintenabilité, versioning et clôture

Stratégies de versioning

L'API va évoluer sans casser les clients existants.

- **Dans l'URI (ex: /api/v1/users) :** Le plus courant et facile à mettre en cache.
- **Par l'en-tête HTTP "Accept" :** (ex: Accept: application/vnd.ecommerce.v2+json). Plus RESTful mais complexe à tester.

TP fil rouge - Étape 3 : spécification OpenAPI

Objectif : Créer un véritable contrat formel.

Exercice :

1. Ouvrez **Scalar Sandbox (https://sandbox.scalar.com/)**.
2. En syntaxe **spécification OpenAPI 3.1 en YAML**, décrivez l'endpoint **GET /products** et **POST /orders**.
3. Intégrez le schéma de données d'une erreur **RFC 7807** (components/schemas).
4. Vérifiez le rendu visuel généré.

Évaluation de connaissances - Fin de session

Pour évaluer vos connaissances, à la fin de cette session, vous devrez répondre au formulaire d'évaluation présent à cette adresse : https://forms.gle/dnZj3dffBDYzynLD9.