

DÉFINITION DES API

QU'EST-CE QU'UNE API ?

Une **API** (Interface de programmation d'application) est un ensemble de **règles** et de **spécifications** utilisées pour faciliter la communication et l'intégration entre différents logiciels.

COMMUNICATION ENTRE LOGICIELS

Les **API** permettent d'échanger des données et d'exécuter des fonctions d'une application à une autre, pour automatiser les processus et faciliter l'utilisation des services.

POURQUOI UTILISER DES API ?

- **Faciliter l'intégration** entre différents logiciels
- **Réutiliser** du code et des fonctionnalités
- **Sécuriser** l'accès aux données et aux services

TYPES D'API

Il existe plusieurs types d'API, parmi lesquels on trouve :

- **API Rest**
- **API SOAP**
- **API GraphQL**

Chaque type d'API a ses propres **spécifications** et **avantages**, en fonction des besoins et des cas d'utilisation.

COMPRENDRE LES API REST

QU'EST-CE QU'UNE API REST ?

Une **API REST** (Representational State Transfer) est une architecture qui définit des **contraintes** pour la conception d'interfaces de programmation.

- Utilise les méthodes HTTP
- Manipule les données au format **JSON** ou **XML**
- Accepte les requêtes de clients et envoie des réponses
- Travaille avec des **ressources**, généralement représentées par des **URL**

ARCHITECTURE

API REST suit une architecture **client-serveur** basée sur les principes **REST**.

APPROCHE CENTRÉE AUTOUR DES RESSOURCES

Dans une API **REST**, les ressources sont au cœur de l'interaction entre le client et le serveur.

LES PRINCIPES D'UNE API REST

Les API REST sont basées sur les principes suivants:

- **Stateless**
- **Cacheable**
- **Client-server**
- **Layered system**
- **Uniform interface**

STATELESS

Chaque requête du client à l'API contient toutes les informations nécessaires pour que le serveur comprenne et traite la demande.

CACHEABLE

Les réponses d'une **API REST** peuvent être mises en **cache** pour améliorer les **performances**.

- Utilisez des en-têtes HTTP pour indiquer la durée de vie du cache.
- Les en-têtes communs incluent "Cache-Control", "ETag" et "Last-Modified".

CLIENT-SERVER

L'**architecture client-serveur** sépare les responsabilités des **applications clientes** et des **serveurs** qui gèrent les ressources.

Client	Serveur
Demande	Ressources
Interface	Stockage
Interaction	Traitement

LAYERED SYSTEM

Les **composants** d'une API REST sont organisés en **couches hiérarchiques**, facilitant la maintenance et l'évolution du système.

INTERFACE UNIFORME

Une API **REST** doit avoir une interface **uniforme**, simplifiant les interactions entre les différentes parties du système.

LES MÉTHODES HTTP

Les **API REST** utilisent des méthodes **HTTP** pour décrire les actions possibles sur les ressources :

- **GET** : Récupère des informations sur une ressource
- **POST** : Crée une nouvelle ressource
- **PUT** : Met à jour une ressource existante
- **DELETE** : Supprime une ressource
- **PATCH** : Modifie partiellement une ressource

GET

La méthode **GET** est utilisée pour récupérer des informations sur une **ressource**.

POST

La méthode **POST** est utilisée pour créer une **nouvelle ressource**.

PUT

La méthode **PUT** est utilisée pour **modifier** une ressource existante.

DELETE

La méthode **DELETE** est utilisée pour **supprimer** une ressource.

PATCH

La méthode **PATCH** est utilisée pour apporter des **modifications partielles** à une ressource.

LES FORMATS DE DONNÉES

JSON

Le **JSON** (JavaScript Object Notation) est un format de données **léger**, facile à lire et à écrire par l'homme, qui est couramment utilisé pour échanger des données entre une **API** et un **client**.

SYNTAXE

Les objets **JSON** sont délimités par des accolades ({}) et contiennent des paires clé-valeur séparées par des deux-points (:). Les tableaux JSON sont délimités par des crochets ([]) et contiennent des valeurs séparées par des virgules.

```
{  
    "key1": "value1",  
    "key2": ["value2", "value3"]  
}
```

EXEMPLES

```
{  
  "name": "John Doe",  
  "age": 28,  
  "address": {  
    "street": "123 Main St",  
    "city": "New York"  
  }  
}
```

XML

Le **XML** (eXtensible Markup Language) est un format de données plus ancien, qui utilise des **balises** pour décrire la structure et le contenu des données.

SYNTAXE

Les balises XML sont délimitées par des **crochets angulaires** (<>) et se ferment avec une **barre oblique** (</>).

```
<key1>value1</key1>
<key2>
  <item>value2</item>
  <item>value3</item>
</key2>
```

EXAMPLES

```
<person>
  <name>John Doe</name>
  <age>28</age>
  <address>
    <street>123 Main St</street>
    <city>New York</city>
  </address>
</person>
```

AVANTAGES ET INCONVÉNIENTS DE CHAQUE FORMAT

Format	Avantages	Inconvénients
JSON	Léger, facile à lire et à écrire Support natif dans la plupart des langages	Moins d'outils de validation
XML	Support étendu pour la validation Outils nombreux et matures	Plus lourd et moins lisible Syntaxe plus complexe

AUTHENTIFICATION ET AUTORISATION

MÉTHODES D'AUTHENTIFICATION

Différentes méthodes d'**authentification** sont utilisées pour sécuriser l'accès aux **API** :

- **API Key**
- **OAuth2**
- **JSON Web Token (JWT)**

API KEY

- Clé **unique** fournie pour chaque utilisateur
- Incluse dans les **requêtes** (généralement dans l'en-tête Authorization)
- **Facile** à mettre en œuvre, mais **moins sécurisée** que d'autres méthodes

OAUTH2

- Protocole d'authentification **standardisé**
- Autorise des applications **tierces** à accéder en toute sécurité à des ressources
- Utilise des **jetons d'accès** à durée limitée plutôt que des identifiants d'utilisateur
- Plusieurs "**grant types**" pour différents cas d'utilisation

JWT (JSON WEB TOKEN)

- Jeton d'**authentification** compact et autoporteur
- Encapsule les informations de l'utilisateur et les **permissions**
- Signé numériquement pour garantir l'**intégrité** des données
- Peut être utilisé avec d'autres protocoles (par exemple, **OAuth2**)

RÔLES ET PERMISSIONS

Les **API** peuvent également gérer l'**autorisation** en fonction des rôles et des permissions :

- Utilisateur
- Administrateur
- Application (Client)

UTILISATEUR

- **Accès limité** aux ressources qui lui appartiennent
- Peut **interagir** avec les fonctionnalités de base de l'API

ADMINISTRATEUR

- **Accès** à toutes les **ressources** et **fonctionnalités** de l'API
- Peut **gérer** d'autres **utilisateurs** et leurs **droits**

APPLICATION (CLIENT)

- **Accès limité** aux fonctionnalités nécessaires pour accomplir sa tâche
- **Permissions** définies par les développeurs et les administrateurs

INTERAGIR AVEC DES API REST

UTILISATION D'UN CLIENT REST

Les **clients REST** facilitent l'interaction avec des **API REST** sans écrire de code.

POSTMAN

Postman est un outil populaire pour concevoir, développer et **tester des API**.

INSOMNIA

Insomnia est une alternative à **Postman** pour tester des **API REST**.

COMPRENDRE LES RÉPONSES D'API

Les réponses d'API contiennent des informations sur le résultat de la requête, y compris les **données** et les **codes de statut HTTP**.

Parties d'une réponse d'API	Description
Statut HTTP	Indique si la requête a réussi ou non
En-têtes	Contiennent des informations sur la réponse
Corps	Contient les données demandées ou un message d'erreur

CODES DE STATUT HTTP

Code Signification

200 OK

201 Created

400 Bad Request

401 Unauthorized

404 Not Found

500 Internal Server Error

CORPS DE RÉPONSE

Le corps de réponse contient des informations sous forme de **JSON** ou **XML**.

COMMENT LIRE LA DOCUMENTATION D'UNE API

La documentation d'une **API** explique comment interagir avec l'API.

- **Endpoints** : adresses URL pour accéder aux différentes ressources de l'API.
- **Méthodes HTTP** : actions possibles sur les ressources (GET, POST, PUT, DELETE).
- **Paramètres** : informations supplémentaires à envoyer avec la requête (query, header, body).
- **Format de réponse** : structure des données renvoyées par l'API (JSON, XML).
- **Codes d'erreur** : messages d'erreur lorsque des problèmes surviennent.

ENDPOINTS

Les **endpoints** représentent les URLs de l'API pour accéder aux **ressources**.

RESSOURCES

Les **ressources** sont des objets manipulés par l'**API** (ex : utilisateurs, produits).

MÉTHODES HTTP

Les **méthodes HTTP** décrivent les actions possibles sur les ressources :

- GET
- POST
- PUT
- DELETE
- PATCH

AUTHENTIFICATION

La **documentation** explique comment **s'authentifier** pour accéder aux ressources **protégées**.

PRATIQUES ET CONVENTIONS

SÉCURITÉ ET CONFIDENTIALITÉ

CHIFFREMENT

- **Protéger** les communications entre **clients** et **serveur**
- Utilisation de **HTTPS** (SSL/TLS)

AUTHENTIFICATION

- Confirmer l'**identité** des utilisateurs
- Mise en place de **méthodes d'authentification** (API Key, OAuth2, JWT)

Méthode	Utilisation
API Key	Authentification simple
OAuth2	Authentification avec autorisations
JWT (JSON Web Token)	Authentification avec stockage d'informations

RATE LIMITING

POURQUOI LIMITER LE TAUX D'APPEL D'UNE API ?

- **Prévenir les abus**
- **Garantir les performances** et la disponibilité
- **Gérer les ressources serveur**

IMPACT SUR L'UTILISATION

- **Planifier** les appels d'**API** en tenant compte des **limites**
- Gérer les **erreurs** liées au dépassement des limites

PAGINATION ET FILTRAGE

POURQUOI PAGINER LES RÉSULTATS

- Améliorer la **performance** des requêtes
- Rendre la **gestion des résultats** plus facile

UTILISER DES FILTRES POUR AFFINER LES REQUÊTES

- **Filtrer** les résultats selon des critères spécifiques
- **Réduire** le nombre de données à traiter

GESTION DES ERREURS

COMPRENDRE ET TRAITER LES ERREURS D'API

- **Codes de statut HTTP** (4xx et 5xx)
- Messages d'erreur **détaillés**

BONNES PRATIQUES DE GESTION DES ERREURS

- Gérer les erreurs côté **client** (affichage, réessayer)
- Prévoir les erreurs et mettre en place un **traitement approprié** pour chacune
- Utiliser des **outils de surveillance** et de **débogage** des erreurs d'API

