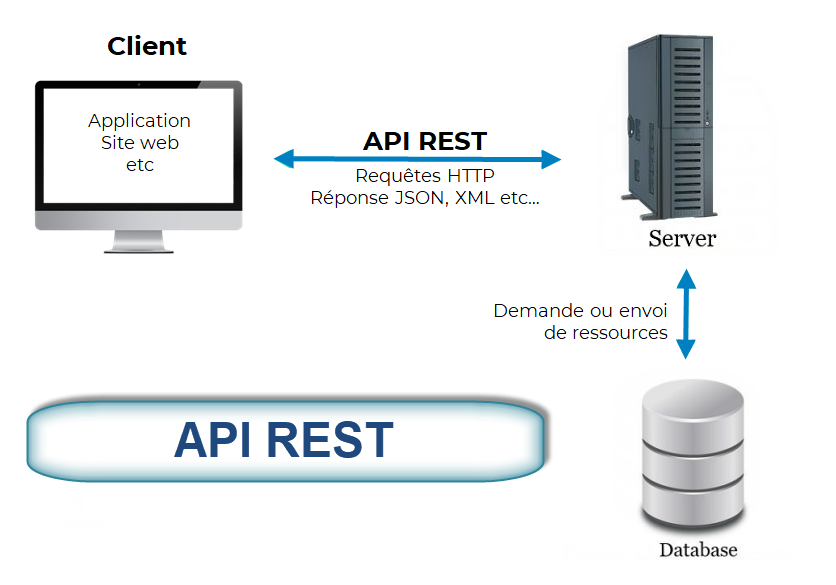
**L'architecture API REST**

25 JUILLET 2023[PYTHONTECH](https://www.tresfacile.net/author/admin/)[2-GLOSSAIRE INFORMATIQUE](https://www.tresfacile.net/category/2-glossaire-informatique/)[LEAVE A COMMENT](https://www.tresfacile.net/larchitecture-api-rest/#respond)



Contenu du cours

1. [Description d'une API RESTful](https://www.tresfacile.net/larchitecture-api-rest/#1)
2. [Les caractéristiques clés d'une API REST](https://www.tresfacile.net/larchitecture-api-rest/#2)
3. [Différence entre API REST et API RESTfull](https://www.tresfacile.net/larchitecture-api-rest/#3)
4. [Les avantages d'une application API REST](https://www.tresfacile.net/larchitecture-api-rest/#4)
5. [Liste des framework adoptant l' API REST](https://www.tresfacile.net/larchitecture-api-rest/#5)
6. [Exemple d'une API REST réalisée avec FastAPI en Python](https://www.tresfacile.net/larchitecture-api-rest/#6)

**1. Description d'une API RESTful**

Une **API REST (Representational State Transfer)** est une interface de programmation d'application qui suit les principes de l'**architecture REST**. Elle permet à différentes applications de communiquer entre elles via des **requêtes HTTP**standard, facilitant ainsi l'**échange de données** et la **gestion des opérations**. En adoptant ces principes, une **API REST** permet aux développeurs de **créer des applications web** et **mobiles flexibles**, interopérables et évolutives. Elle favorise également la réutilisation des fonctionnalités, car différentes applications peuvent communiquer efficacement en utilisant des standards ouverts et bien connus.

**2. Les caractéristiques clés d'une API REST**

1. **Ressources :** Une API REST expose des ressources, qui sont des entités identifiables, telles que des objets, des données ou des services. Chaque ressource est identifiée par une URI (Uniform Resource Identifier), ce qui permet de la localiser facilement.
2. **Verbes HTTP :** Les opérations sur les ressources sont effectuées en utilisant les verbes HTTP standard tels que GET (récupérer), POST (créer), PUT (mettre à jour) et DELETE (supprimer). Chaque verbe a une signification spécifique dans l'interaction avec les ressources.
3. **Représentations :** Les ressources peuvent être représentées dans différents formats, tels que JSON (JavaScript Object Notation), XML (eXtensible Markup Language), HTML, etc. La représentation de la ressource dépend souvent de l'en-tête "Accept" envoyé dans la requête HTTP par le client.
4. **État de l'application :** L'API REST est "stateless", ce qui signifie que chaque requête doit contenir toutes les informations nécessaires pour comprendre et traiter cette demande spécifique. Le serveur ne stocke pas l'état du client entre les requêtes, améliorant ainsi la fiabilité et la tolérance aux pannes.
5. **HATEOAS (Hypertext As The Engine Of Application State) :** C'est un principe clé de REST qui suggère que les réponses de l'API doivent inclure des liens hypertextes pour naviguer vers d'autres ressources associées. Cela permet au client de découvrir dynamiquement les actions possibles et les relations entre les ressources.
6. **Couche transport :** Une API REST utilise généralement HTTP comme protocole de communication, ce qui en fait une interface largement utilisée, car HTTP est déjà répandu sur Internet.
7. **Sécurité :** L'API REST peut être sécurisée en utilisant des mécanismes d'authentification et d'autorisation, tels que les jetons d'accès (tokens) ou les clés API, pour protéger les ressources et les données sensibles.
8. **Documentation :** Une bonne API REST est généralement bien documentée pour permettre aux développeurs d'apprendre comment l'utiliser correctement. Des descriptions claires des endpoints, des verbes HTTP, des formats de données, etc., aident à faciliter l'intégration avec l'API.

**3. Différence entre API REST et API RESTfull**

La différence entre **"API REST"** et **"API RESTful"** est souvent une source de **confusion**, car les termes sont utilisés de manière interchangeable par de nombreuses personnes. Cependant, il existe une nuance subtile entre les deux termes :

**3.1 API REST (REST API)**

L'**API REST (ou REST API)** est une **API** qui suit les principes de l'**architecture REST (Representational State Transfer)**. Elle expose des ressources sous forme d'URL (Uniform Resource Locator) et utilise les verbes HTTP standard (GET, POST, PUT, DELETE, etc.) pour effectuer des opérations sur ces ressources. L'**API REST** utilise également les codes d'état**HTTP** pour indiquer le résultat d'une requête.

**Conclusion**

Une **API REST** est simplement une API qui suit les principes de l'**architecture REST**, mais le terme ne spécifie pas si l'API respecte complètement tous les principes de REST.

**3.2 API RESTful (RESTful API)**

L'**API RESTful (ou RESTful API)** est une **API** qui respecte complètement les principes de l'**architecture REST**. Cela signifie que l'API suit toutes les contraintes de REST, notamment :

1. L'utilisation d'un protocole client-serveur sans état.La manipulation des ressources par le biais de représentations, généralement au format JSON ou XML.L'utilisation des verbes HTTP (GET, POST, PUT, DELETE, etc.) de manière appropriée pour effectuer des opérations sur les ressources.La liaison de ressources connexes via des hyperliens (HATEOAS - Hypertext As The Engine Of Application State).
2. En résumé: une API RESTful est une API qui respecte strictement tous les principes de l'architecture REST.

**Pratiquement**

Les deux termes sont souvent utilisés de manière interchangeable et la différence peut ne pas être très importante dans la plupart des cas. Cependant, il est important de comprendre que **"API REST"** fait généralement référence à une **API** qui suit les principes de **REST**, tandis que **"API RESTful"** fait référence à une **API** qui respecte pleinement tous les principes de **REST**.

**4. Les avantages d'une application API REST**

Les applications **API RESTful** offrent de **nombreux avantages**, ce qui explique en partie leur popularité croissante dans le développement d'applications web et mobiles. Voici quelques-uns des principaux avantages d'une application **API REST** :

1. **Architecture simple et légère :** Les API RESTful sont basées sur des principes d'architecture simples et standardisés, ce qui les rend faciles à comprendre, à concevoir et à implémenter. Elles sont légères par rapport à d'autres solutions d'intégration complexes.
2. **Flexibilité :** Les API RESTful sont indépendantes de la plate-forme et du langage de programmation, ce qui signifie qu'elles peuvent être utilisées par différentes applications développées dans différents langages.
3. **Facilité d'intégration :** Les API RESTful permettent une intégration aisée entre différentes applications, services ou systèmes, ce qui favorise la réutilisation des fonctionnalités et la collaboration entre équipes de développement.
4. **Portabilité :** Étant basées sur les protocoles HTTP standard, les API RESTful peuvent être utilisées sur différentes plateformes (web, mobile, IoT, etc.), ce qui facilite la portabilité de l'application.
5. **Sécurité :** Les API RESTful peuvent être sécurisées en utilisant des mécanismes d'authentification standard tels que les jetons d'accès (tokens) ou les clés API, ce qui permet de protéger les ressources et les données sensibles.
6. **Performances :** Les API RESTful sont conçues pour être légères et efficaces, ce qui peut contribuer à des performances améliorées de l'application.
7. **Scalabilité :** Les API RESTful peuvent être conçues de manière à être facilement mises à l'échelle pour répondre à des demandes croissantes et à des volumes de trafic plus importants.
8. **Documentation automatique :** Grâce à l'utilisation de standards comme OpenAPI (anciennement Swagger), les API RESTful peuvent générer automatiquement leur documentation, ce qui facilite la compréhension et l'utilisation de l'API par les développeurs tiers.
9. **Évolutivité :** Les API RESTful sont conçues pour permettre une évolution sans rupture. Les nouvelles fonctionnalités peuvent être ajoutées sans affecter les clients existants, à condition de respecter la compatibilité des URI.
10. **Découplage client-serveur :** Les API RESTful encouragent un fort découplage entre le client et le serveur, ce qui signifie que les modifications du serveur n'affectent pas nécessairement le client et vice versa.
11. **Conclusion:** les applications API RESTful offrent une approche moderne et flexible pour le développement d'applications web et mobiles, permettant une intégration aisée, une évolutivité et une réutilisation des fonctionnalités. Ces avantages contribuent à la popularité et à l'adoption croissante des API RESTful dans le domaine du développement logiciel.

**5. Liste des framework adoptant l' API REST**

De nombreux framework modernes adopte **l'API REST!** Nous allons citer ici une liste non exhaustive, mais elle comprend certains des **frameworks** les plus **populaires** dans différents langage de programmation. Chaque **framework** a **ses propres caractéristiques** et avantages, donc le choix dépendra des besoins spécifiques du projet et des préférences de l'équipe de développement. Voici une liste de certains des frameworks populaires qui adoptent l'**API REST** pour faciliter la création d'applications web et services :

1. **Django REST framework (Python) :** Une extension puissante pour Django qui facilite la création rapide et flexible d'API RESTful en utilisant Python.
2. **Flask-RESTful (Python) :** Une extension de Flask qui permet de créer des API RESTful de manière simple et élégante en utilisant Python.
3. **Express.js (JavaScript) :** Un framework web léger et flexible pour Node.js, souvent utilisé pour créer des API RESTful côté serveur en utilisant JavaScript.
4. **Spring Boot (Java) :** Un framework Java populaire qui facilite la création d'API RESTful sécurisées et évolutives avec Spring.
5. **ASP.NET Web API (C#) :** Un framework pour construire des API RESTful en utilisant C# sur la plateforme .NET.
6. **Ruby on Rails (Ruby) :** Un framework Ruby qui facilite le développement d'API RESTful en utilisant le langage Ruby.
7. **Laravel (PHP) :** Un framework PHP populaire qui offre un support complet pour la création d'API RESTful.
8. **Hapi.js (JavaScript) :** Un framework JavaScript pour Node.js, qui facilite la création d'API RESTful robustes et modulaires.
9. **Play Framework (Scala) :** Un framework web basé sur Scala et Java, qui offre une prise en charge native pour les API RESTful.
10. **Phoenix (Elixir) :** Un framework web basé sur Elixir, qui permet de créer des API RESTful performantes et extensibles.
11. **FastAPI (Python) :** Un framework Python moderne et rapide pour créer des API RESTful basées sur les types Python.
12. **Restify (JavaScript) :** Un framework JavaScript léger et rapide pour Node.js, conçu pour créer des API RESTful côté serveur.
13. **Jersey (Java) :** Un framework Java qui implémente la spécification JAX-RS pour faciliter la création d'API RESTful.
14. **Symfony (PHP) :** Un framework PHP complet qui offre un composant dédié pour créer des API RESTful.
15. **Gin (Go) :** Un framework Go léger et rapide pour créer des API RESTful performantes.

**6. Exemple d'une API REST réalisée avec FastAPI en Python**

Voici un exemple simple d'une **API REST** créée avec **FastAPI** en **Python** :

**Tout d'abord, assurez-vous d'avoir installé FastAPI en utilisant pip :**

Python

|  |  |
| --- | --- |
| 1 | pip install fastapi |

**Ensuite, nous allons créer un fichier Python pour notre API REST. Par exemple, nommez-le "main.py" :**

Python

|  |
| --- |
| from fastapi import FastAPI    # Créez une instance de FastAPI  app = FastAPI()    # Une liste de données simulées pour notre exemple  fake\_data = [      {"id": 1, "name": "Produit 1"},      {"id": 2, "name": "Produit 2"},  ]    # Définissez une route pour récupérer tous les produits  @app.get("/produits/")  def get\_all\_products():      return fake\_data    # Définissez une route pour récupérer un produit par son ID  @app.get("/produits/{product\_id}")  def get\_product\_by\_id(product\_id: int):      product = next((item for item in fake\_data if item["id"] == product\_id), None)      if product:          return product      else:          return {"message": "Produit non trouvé"}    # Définissez une route pour ajouter un nouveau produit  @app.post("/produits/")  def add\_product(name: str):      new\_product = {"id": len(fake\_data) + 1, "name": name}      fake\_data.append(new\_product)      return new\_product |

Dans cet exemple: nous avons créé une API avec trois endpoints :

1. GET /produits/ : Récupère tous les produits de la liste "fake\_data".
2. GET /produits/{product\_id} : Récupère un produit en fonction de son ID.
3. POST /produits/ : Ajoute un nouveau produit à la liste "fake\_data" en utilisant le nom fourni dans le corps de la requête.

#### Execution de l'application

Pour exécuter l'API, lancez le serveur FastAPI en exécutant le fichier "main.py" :

Python

|  |  |
| --- | --- |
| 1 | uvicorn main:app --reload |

Une fois que le serveur est en cours d'exécution, vous pouvez accéder à l'API en utilisant l'URL "http://localhost:8000/produits/" dans votre navigateur ou un outil tel que cURL ou Postman



#### Remarque

Cet exemple est très simple, mais il faut mettre à l'esprit que FastAPI offre de nombreuses fonctionnalités avancées pour la validation des données, la documentation automatique de l'API, la gestion des erreurs, et bien plus encore. Vous pouvez explorer la documentation officielle de FastAPI pour en savoir plus sur ses fonctionnalités avancées : <https://fastapi.tiangolo.com/>.