

IoT -- Cloud

Thème 5 -- API

Juliette NGUYEN, Corentin AUBRY,
Agathe DELAS, Garance CHAMALET





Plan du cours

1. Introduction

- Définition
- Les différents types d'API
- Historique

2. Présentation de REST

- REST vs SOAP
- Principe RESTful
- Versionning avec REST

3. Focus sur OpenAPI et de Swagger

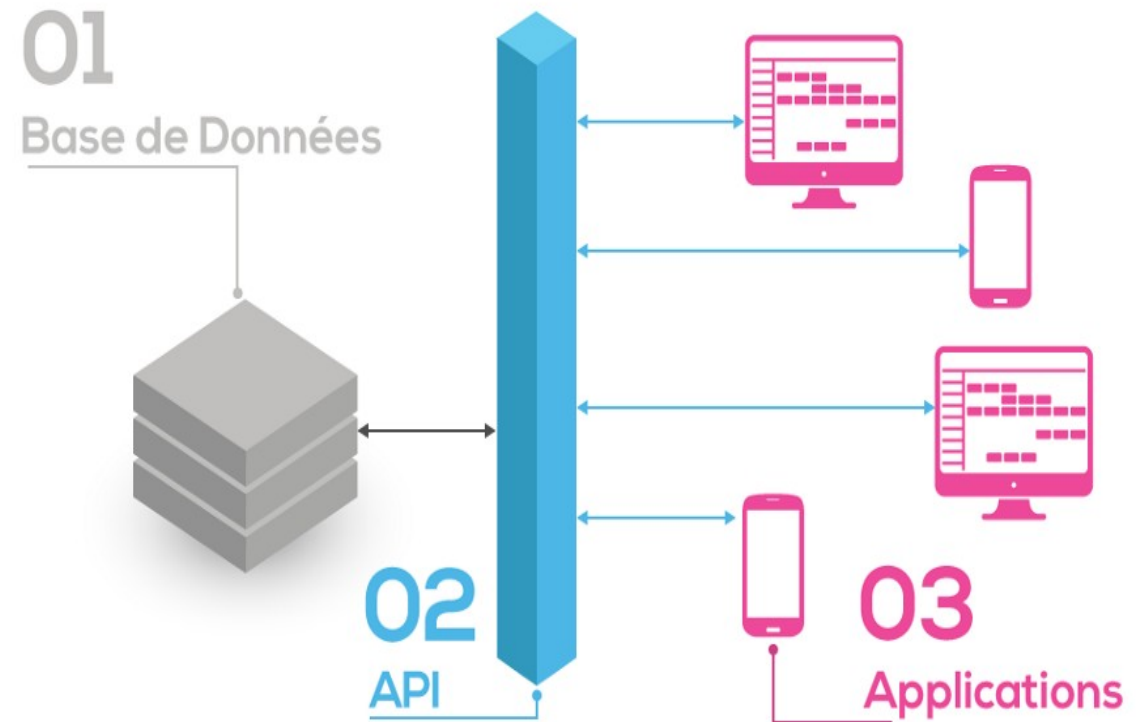
- Open API et Swagger: quelles différences ?
- Pourquoi utiliser Open API ?
- Pourquoi Swagger, c'est génial ?

4. Démonstration technique de OpenAPI

5. Rappel sur OpenAPI

Qu'est-ce qu'une API ?

Une API est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. API est un acronyme anglais qui signifie « Application Programming Interface », que l'on traduit par interface de programmation d'application



A

Application. Par application s'entend tout service avec lequel un développeur ou une autre application souhaite interagir. Cela peut être un service météorologique, une application de partage d'images, un portail Open Data.

P

Programme. Le programme est une fonction informatique à laquelle un développeur donne des instructions et qui va interagir avec l'application à notre place. Le programme peut par exemple récupérer des données à intervalles régulières ou soumettre une adresse postale (pour récupérer une coordonnée géographique)...

I

Interface. L'interface est la porte d'entrée par laquelle il sera possible d'interagir avec l'application.

Ce que peut faire une api concrètement

Exposer des données

```
{
  - data: {
    - {
      acronym: null,
      - badges: [
        - {
          kind: "public-service"
        },
        - {
          kind: "certified"
        }
      ],
      created_at: "2014-04-17T18:21:57.325000",
      deleted: null,
      description: "
        Aux côtés de son cabinet, l'administration du Premier ministre comprend de nombreux services qui l'assistent et prennent part à l'élaboration de la politique du
        Gouvernement. Sont rattachés au Premier ministre les organismes chargés de missions de coordination interministérielle qui ne peuvent être attribuées à un seul
        ministère. ",
      id: "534fffa5a3a7292c64a7809e",
      last_modified: "2018-09-03T20:47:50.820000",
      logo: "https://static.data.gouv.fr/avatars/08/98902e29244685862bcd3198cef7b-original.png",
      logo_thumbnail: "https://static.data.gouv.fr/avatars/08/98902e29244685862bcd3198cef7b-100.png",
      - members: [
        - {
          role: "editor",
          - user: {
            avatar: "https://static.data.gouv.fr/avatars/aa/733490184c4c2684e793ea074d57d6-original.jpg",
            avatar_thumbnail: "https://static.data.gouv.fr/avatars/aa/733490184c4c2684e793ea074d57d6-100.jpg",
            class: "User",
            first_name: "Anne-Gaelle",
            id: "534fffa5a3a7292c64a773ab",
            last_name: "Javelle",
            page: "https://www.data.gouv.fr/fr/users/anne-gaelle-javelle/",
            slug: "anne-gaelle-javelle",
            uri: "https://www.data.gouv.fr/api/1/users/anne-gaelle-javelle/"
          }
        }
      ]
    }
  }
}
```

[View source](#)

Exposer des services

Votre entreprise

Nom commercial

Agence Debord

Siret

80133915100020

Adresse

Lim

📍 Limoges France

📍 Limeil-Brévannes France

📍 Limay France

📍 Limoux France

📍 Limonest France

powered by Google

Code postal

Logo

Pour un résultat optimal, utilisez une image carrée

📁 CHOISISSEZ UNE IMAGE

Quels types d'API existent ?

les APIs de système d'exploitation permettent aux logiciels d'interagir avec les périphériques (votre webcam), de reconnaître vos gestes (sur votre écran tactile)...

les APIs de langage de programmation permettent aux développeurs d'utiliser des fonctions prédéfinies.

les APIs d'infrastructure permettent la modification de ressources disponibles pour faire fonctionner une application via le cloud (machines virtuelles, serveurs, architecture réseau, etc). Exemple Amazon Web Services !

les APIs des services web : des fonds de carte Google Maps au Social Graph de Facebook en passant par le monitoring de tweets, c'est la catégorie d'APIs connaissant actuellement la plus forte croissance. Ainsi, le service Airbnb utilise l'API Google Maps pour transformer les adresses des biens à louer en points géolocalisés.

Histoire de l'API

1985 : Microsoft et la fédération d'une communauté technique autour des APIs

2000 : Salesforce et les premières APIs web

2000 : eBay et la pénétration des marchés via les APIs

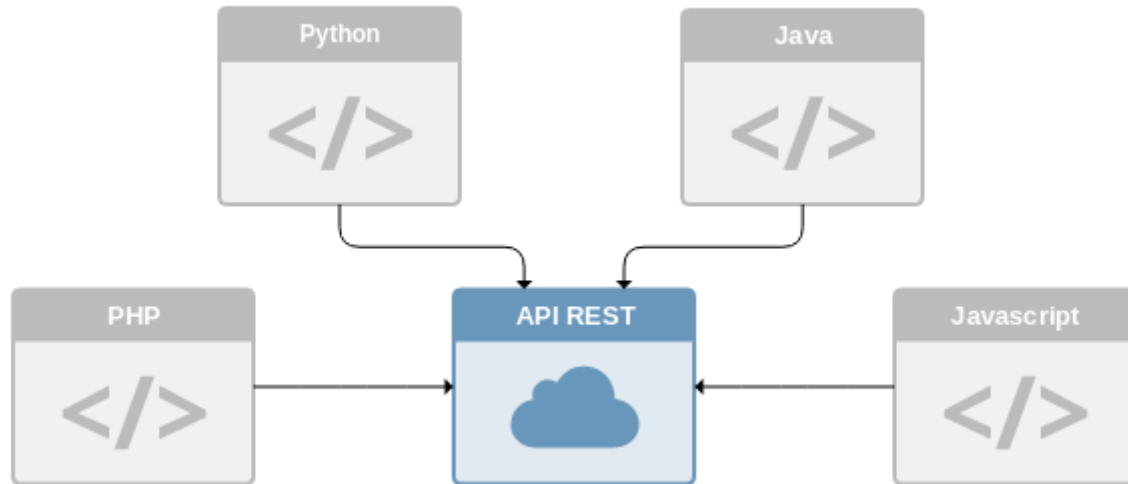
2004 : Flickr, APIs et BizDev 2.0

2006 : Amazon, naissance de l'Infrastructure-As-A-Service

2006 : Facebook et l'exposition de la base utilisateur

https://medium.com/@mercier_remi/02-il-%C3%A9tait-une-fois-les-apis-a8a723b2b96b

Plan du cours



1. Introduction
 - Définition
 - Les différents types d'API
 - Historique
2. **Présentation de REST**
 - REST vs SOAP
 - Principe RESTful
 - Versionning avec REST
3. Focus sur OpenAPI et de Swagger
 - Open API et Swagger: quelles différences ?
 - Pourquoi utiliser Open API ?
 - Pourquoi Swagger, c'est génial ?
4. Démonstration technique de OpenAPI
5. Rappel sur OpenAPI

SOAP

Utilisé à partir de la fin des années 90

Basé sur XML est un fichier de description

Avantages

- Très normé
- Validation automatique via les validateur XML
- Traitement native des erreurs

Inconvénient

- XML égal très verbeux et très volumineux
- XML non adapté pour certains langages en particulier JS
- peu performant et très lourd

REST

Créé en 2000

Signifie « Representational State Transfer »

Une API REST est entièrement basée sur le protocole http

Le standard REST définit plusieurs règles à respecter

Une API qui respectent l'ensemble de ces règles peut être considéré comme **RESTFUL**

Règle n°1 : Stateless

Règle n°2 : Cacheable

Règle n°3 : Client-Server Oriented

Règle n°4 : Uniform

Stateless vs Stateful

Stateless signifie protocole sans état :

- Une API REST doit être sans état
- La requête doit contenir l'ensemble des données nécessaires à son traitement
- Pour un serveur chaque requête est une entité totalement distincte d'une autre
- **La notion de session n'existe pas**

Le respect de cette règle permet de concevoir une API **scalable.**

Il y a néanmoins certaines exceptions à ce principe (Limitées et exceptionnelles**) :**

- *L'utilisation de compteurs pour une restriction d'utilisation*
- *Certains types d'authentification qui peut nécessiter une session temporaire*

L'API doit dire au client si l'information envoyée **peut être mis en cache** (combien de temps et dans quel circonstance)

On doit utiliser les implémentations standards de cache HTTP.

- *Cache-Control*
 - *Public/private/no-cache*
 - *Max-age/s-maxage*
- *Expires*
- *E-Tag*
- *Last-Modified*

Cela permet moins de requête inutile et augmente ainsi la capacité de l'Api (proxy-cache)

Le client et le serveur ne sont liés que par un contrat d'interface.

Une API ne doit pas être conçue pour une interface où une application particulière mais elle doit être conçue pour toutes les interfaces possibles. Elle doit donc être la plus générique possible dans les informations présentées, cela permet la pérennité et l'évolutivité de l'API.

Le client n'est pas concerné par:

- La Manière ou la forme dans laquelle sont stockées les données par le serveur
- Les traitements effectués et la logique qui s'applique sur ces données pour traiter sa demande

Le serveur n'est pas concerné par :

- L'interface utilisateur qui va servir à consulter éditer traiter les données servi
- L'état de l'utilisateur

Le client d'une API fournit l'ensemble des données nécessaires au traitement de celles-ci dans la requête http envoyée :

- L'action attendue
- La ressource sur laquelle cette action doit être effectuée
- L'ensemble des données nécessaires à ce traitement
- Le format de la réponse attendue

L'API doit retourner dans la réponse l'ensemble des éléments nécessaires pour que le client puisse traiter celle-ci :

- Le résultat du traitement de sa requête (code d'erreur/réussite http)
- Le format dans lequel la réponse est retournée (http : header content type)
- Potentiellement la durée de validité de cette réponse (Cacheable)
- Le chemin des ressources pouvant être utilisé à la poursuite de ces traitements

Chaque ressource exposée par l'API est identifiée par une url

Les urls sont conçue sur la forme suivante :

- /[collection]
- /[collection]/[identifiant dans la collection]
- /[collection] /[identifiant dans la collection]/[sous-collection]

• Exemple

URL	Descriptif
/books	Action sur l'ensemble des livres
/books/978-8-1234-5680-3	Action sur le livre possédant l'ISBN indiqué
/books/978-8-1234-5680-3/pages	Action sur les pages du livre possédant l'ISBN indiqué
/books/978-8-1234-5680-3/pages/8	Action sur la page 8 du livre possédant l'ISBN indiqué

Une collection est un nom au pluriel car il représente un ensemble d'objets.

Garder une uniformité sur le nommage:

- Toujours en minuscules
- Mots composés séparés par des « - »

Généralement pour une ressource, il y a 4 opérations possibles :

Créer (create) => POST
Afficher (read) => GET
Mettre à jour (update) => PUT
Supprimer (delete) => DELETE

L'état du traitement de la requête est renvoyé via le code HTTP de la réponse

Code HTTP	Usage
200	Le code le plus utilisé, la requête a été traitée avec succès
201	Utilisé à la suite d'une création pour indiquer que la ressource est créée
204	La requête a été traitée avec succès et la réponse n'a pas de corps Utilisé principalement suite à un DELETE ou un PUT

La réponse du serveur est habituellement au minimum du format JSON

POST /products

- 201 Created
- Location: /products/1337

GET /products/1337

- 200 OK

PUT /products/1337

- 200 OK

DELETE /products/1337

- 204 No content

GET /products/1337

- 404 Not Found

Il est nécessaire de maintenir différentes versions de l'API au moins un certain temps pour maintenir le service.

« Le contrat d'interface vous engage vis-à-vis de vos clients. Si vous modifier le format d'échange vous risquez de casser les clients qui utilisent votre API. »

Le versionning permet de faire évoluer une API tout en gardant une rétro compatibilité.

La version souhaitée peut-être transmise par le client via:

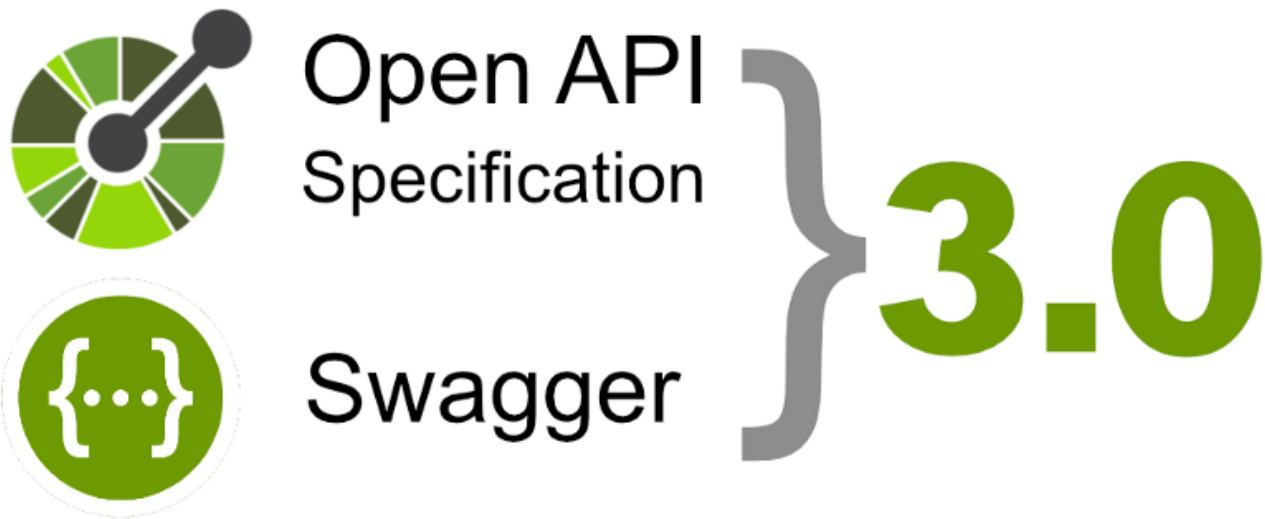
- L'url : <https://api.example.com/1.0/books>
- L'entête Accept :
« Accept:application/json;version=1.0 »
- Un entête spécifique : « X-API-Version:1.0 »

Conférence ETNA, Antoine PACAUD

APIs REST : Bonnes pratiques & Sécurité

<https://blog.webnet.fr/apis-rest-bonnes-pratiques-securite/>

- > Notion de sécurité API key, OAuth1, OAuth2,....
- > Ensemble de bonnes pratiques



Plan du cours

1. Introduction
 - Définition
 - Les différents types d'API
 - Historique
2. Présentation de REST
 - REST vs SOAP
 - Principe RESTful
 - Versionning avec REST
3. **Focus sur OpenAPI et de Swagger**
 - Open API et Swagger: quelles différences ?
 - Pourquoi utiliser Open API ?
 - Pourquoi Swagger, c'est génial ?
4. Démonstration technique de OpenAPI
5. Rappel sur OpenAPI

Focus sur Open API & Swagger

Open API & Swagger: quelles différences ?



Open API Specification

#Specification

- Anciennement nommé « Swagger Specification »
- Donné à la fondation Linux en 2015
- Complètement open-source et gratuit
- Standard de description pour les API REST, permettant à la fois aux humains et aux machines de comprendre les capacités offertes par le service sans accéder à son code ou sa documentation



Swagg

#Outil

- Propriété de Smartbear Software depuis 2015
- Ensemble d'outils open-sources, gratuits et commerciaux
- Permet d'implémenter plus facilement et visuellement les API utilisant l'OAS

Pour une description plus complète,

Pourquoi utiliser la **spécification**

Open

- Pour sa facilité
- Pour sa lisibilité, aussi bien par un développeur que par un utilisateur du service
- Pour son format: YAML ou JSON
- Pour sa communauté et les outils associés ([liste non exhaustive ici](#))
- Et puis c'est gratuit, pardi !

```
openapi: 3.0.0
info:
  title: Gestionnaire de la bibliothèque
  description: API de la bibliothèque de Arkham City.
  version: 1.0.0
servers:
  - url: http://bibliotheque-arkham-city.com:8000
paths:
  /books/{author}:
    get:
      summary: Renvoie la liste des livres écrit par l'auteur passé en paramètre.
      parameters:
        - in: path
          name: author
          description: L'auteur des livres.
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Une JSON array des livres écrits par l'auteur.
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
      default:
        description: Unexpected error.
```

De la doc, vous en voulez ?

Focus sur Open API & Swagger

Pourquoi **Swagger** est si **génial** que ça ?

- Outil de génération de code automatique
- Outils supportant plein de frameworks
- Outil de génération de code automatique
- Outil de génération de documentation automatique
- On vous a parlé de la génération automatique de code ?



Dart



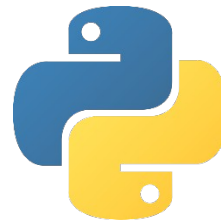
Go



ERLANG



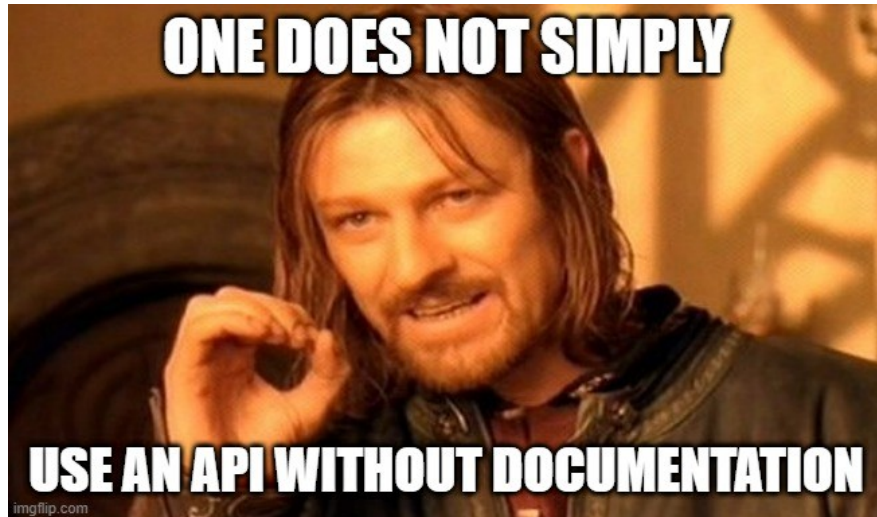
ANGULAR



Flask



La génération de code, c'est cool ! Mais la **documentation**, en a-t-on **vraiment besoin...**?



Vous aimez monter des meubles IKEA sans la notice, vous ? C'est bien ce que je pensais, oui...

- Un document contenant toutes les informations requises
- Plus besoin d'embêter Raoul le développeur toutes les 10 minutes pour comprendre comment ça fonctionne
- Plus besoin de passer par une formation avec Raoul pour les nouveaux utilisateurs
- Plus facile de dénicher les bugs
- Avec OpenAPI, on peut se concentrer sur le « Design First »



Plan du cours

1. Introduction
 - Définition
 - Les différents types d'API
 - Historique
2. Présentation de REST
 - REST vs SOAP
 - Principe RESTful
 - Versionning avec REST
3. Focus sur OpenAPI et de Swagger
 - Open API et Swagger: quelles différences ?
 - Pourquoi utiliser Open API ?
 - Pourquoi Swagger, c'est génial ?
4. **Démonstration technique de OpenAPI**
5. Rappel sur OpenAPI



Plan du cours

1. Introduction
 - Définition
 - Les différents types d'API
 - Historique
2. Présentation de REST
 - REST vs SOAP
 - Principe RESTful
 - Versionning avec REST
3. Focus sur OpenAPI et de Swagger
 - Open API et Swagger: quelles différences ?
 - Pourquoi utiliser Open API ?
 - Pourquoi Swagger, c'est génial ?
4. Démonstration technique de OpenAPI
5. **Rappel sur OpenAPI**

Les métadonnées

Utilisée pour: Décrire les métadonnées à propos de l'API.

Extrait de la documentation :

Field Name	Type	Description
title	string	REQUIRED. The title of the API.
description	string	A short description of the API. CommonMark syntax MAY be used for rich text representation.
termsOfService	string	A URL to the Terms of Service for the API. MUST be in the format of a URL.
contact	Contact Object	The contact information for the exposed API.
license	License Object	The license information for the exposed API.
version	string	REQUIRED. The version of the OpenAPI document (which is distinct from the OpenAPI Specification version or the API implementation version).

Exemple :

```
openapi: 3.0.0
info:
  title: Ma super API
  description: Voici ma super API !
  version: 0.0.1
```

Les serveurs

Utilisée pour: Indiquer les différents serveurs hébergeant l'API.

Extrait de la documentation :

Field Name	Type	Description
url	string	REQUIRED. A URL to the target host. This URL supports Server Variables and MAY be relative, to indicate that the host location is relative to the location where the OpenAPI document is being served. Variable substitutions will be made when a variable is named in <code>{brackets}</code> .
description	string	An optional string describing the host designated by the URL. <code>CommonMark syntax</code> MAY be used for rich text representation.
variables	Map[string, Server Variable Object]	A map between a variable name and its value. The value is used for substitution in the server's URL template.

Exemple :

```
servers:  
- url: https://api.example.com/v1  
  description: Prod  
- url: http://staging-api.example.com  
  description: Dev HTTP  
- url: https://staging-api.example.com  
  description: Dev HTTPS
```

Les tags

Utilisée pour: Regrouper sous une même section différentes requêtes.

Extrait de la documentation :

Field Name	Type	Description
name	string	REQUIRED. The name of the tag.
description	string	A short description for the tag. CommonMark syntax MAY be used for rich text representation.
externalDocs	External Documentation Object	Additional external documentation for this tag.

Exemple :

```
tags:
  - name: users
    description: Tout ce qui concerne les utilisateurs
  - name: books
    description: Tout ce qui concerne les livres
```

Déclaration

```
post:
  tags:
    - users
  summary: Crée un nouvel utilisateur
  responses:
    '200':
      description: OK
    '400':
      description: Nécessite de donner un nom pour créer un utilisateur
  default:
    description: Une erreur inconnue s'est produite.
```

Utilisation pour ajouter une requête au tag

Les chemins – bloc « paths »




Utilisée pour: Déclarer les différents chemins de l'API.

Extrait de la documentation :

Field Pattern	Type	Description
/ {path}	Path Item Object	<p>A relative path to an individual endpoint. The field name MUST begin with a forward slash (/). The path is appended (no relative URL resolution) to the expanded URL from the <code>Server Object</code>'s <code>url</code> field in order to construct the full URL.</p> <p>Path templating is allowed. When matching URLs, concrete (non-templated) paths would be matched before their templated counterparts. Templated paths with the same hierarchy but different templated names MUST NOT exist as they are identical. In case of ambiguous matching, it's up to the tooling to decide which one to use.</p>

Exemple :

```

paths:
  /users:
    get: 
    post: 
  /books:
    get: 
  
```

Les chemins – description de chaque requête

Utilisée pour: Décrire un chemin spécifique, avec son action (get, put...) avec toutes ses infos (paramètres, réponses, ...).

Extrait de la documentation :

Field Name	Type	Description
\$ref	string	Allows for an external definition of this path item. The referenced structure MUST be in the format of a Path Item Object . In case a Path Item Object field appears both in the defined object and the referenced object, the behavior is undefined.
summary	string	An optional, string summary, intended to apply to all operations in this path.
description	string	An optional, string description, intended to apply to all operations in this path. CommonMark syntax MAY be used for rich text representation.
get	Operation Object	A definition of a GET operation on this path.
put	Operation Object	A definition of a PUT operation on this path.
post	Operation Object	A definition of a POST operation on this path.
delete	Operation Object	A definition of a DELETE operation on this path.
options	Operation Object	A definition of a OPTIONS operation on this path.

NB: Extrait coupé.

Exemple :

```
/books:
  get:
    summary: Retourne la liste de la bibliothèque
    description: Retourne une magnifique collection provenant de notre bibliothèque
    tags:
      - books
    parameters:
      - name: language
        in: query
        description: Code langue dans laquelle le titre des livres doit être retourné
        schema:
          type: string
          example: 'FR'
    responses:
      '200':
        description: Un tableau au format JSON contenant la liste des livres
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
                example: La Passe-Miroir
```

Pour la description complète des spécifications, vous avez le droit

Pour aller plus loin

- https://medium.com/@mercier_remi/02-il-%C3%A9tait-une-fois-les-apis-a8a723b2b96b
- https://medium.com/@mercier_remi/c-est-quoi-une-api-f37ae350cb9
- <https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces?fbclid=IwAR3OYV9FaCV1kZOz8u1OgxMDBexWjK21ZJpmU1DCr0U3nsB5610mIUw7yyk>
- Documentation sur la spécification d'Open API
- Autre documentation avec des exemples sur la spécification d'Open API
- Page Web interactive pour vous aider à mieux comprendre les spécifications d'Open API
- Editeur en ligne de Swagger