

API REST / RESTFUL - LARAVEL

José Arthur OUEDRAOGO

Elisée Gontran KIEMDE

Ingénieur de conception en Systèmes d'Information

josearthur.oued@outlook.com

(+226) 61437073

Objectifs

À la fin du cours l'étudiant doit être apte à :

- ❑ Définir un API REST ;
- ❑ Contextualiser le fait que Laravel permettent d'implémenter des API;
- ❑ Faire des CRUD d'API;
- ❑ Créer une activité sur la plateforme (un chat et un forum).

CHAPITRE I : NOTIONS ET APPROCHES D'API

Contenu du chapitre I


- ❑ Introduction à la notion d'Api REST et RESTful ;
- ❑ Qu'est-ce qu'une API ? ;
- ❑ Pourquoi REST ?
- ❑ Quelques exemples d'API REST ;
- ❑ Format des données d'échange (XML, Json) ;
- ❑ Présentation du format Json.

Introduction à la notion d'Api REST / RESTful

Pour les applications web, une API (Application Programming Interface) ou « Interface de Programmation d'Applications » en français, est un ensemble de définitions et de protocoles qui permettent à des applications de communiquer et de s'échanger mutuellement des données ou des services.

Introduction à la notion d'Api REST / RESTful

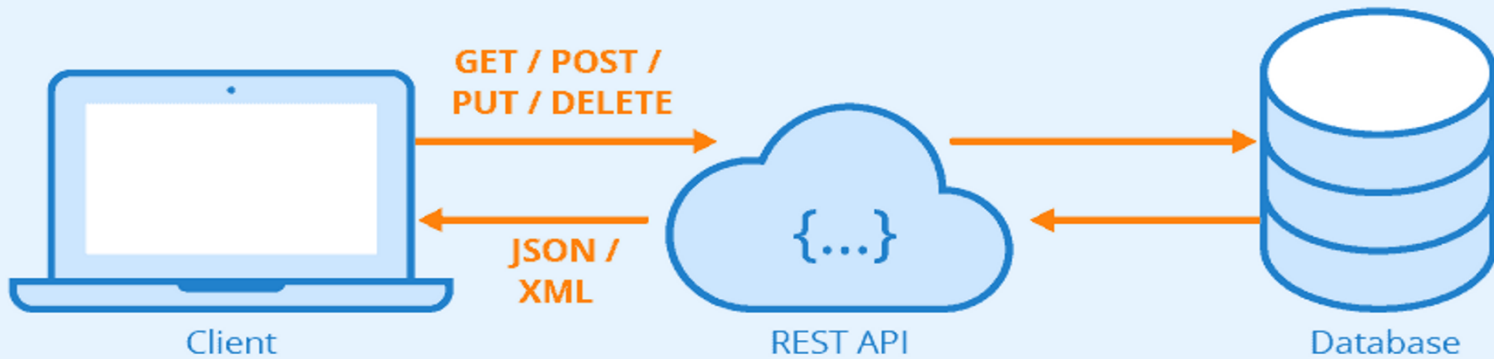
REST, c'est quoi

- ❖ Né autour des années 2000
- ❖ **RE**presentational **State Transfer**
- ❖ Repose sur le protocole HTTP
- ❖ Accès à des ressources (URI)
- ❖ Utilisation de verbes HTTP
- ❖ REST  RESTful

Qu'est-ce qu'une API REST ?

- ❖ **API** est un acronyme pour “Application Programming Interface” ou Interface de programmation d'application en français. Il s'agit d'une interface permettant l'interaction entre différentes applications. Elle définit quels appels ou requêtes peuvent être réalisés et comment les réaliser : le format des données à utiliser, la structure de la réponse, les conventions à respecter etc.
- ❖ Les **APIs** sont à la base de toutes les interactions entre différentes applications. De très nombreuses entreprises et organisations proposent aujourd'hui des APIs pour interagir avec leurs applications. Cela permet ainsi aux développeurs d'applications tierces de réaliser des opérations comme transmettre ou accéder à des données d'une application à une autre via cette API.

Qu'est-ce qu'une API REST ?



Pourquoi des APIs REST ?

❖ Définition de l'architecture REST

REST utilise les spécifications originelles du protocole **HTTP**, C'est un ensemble de conventions et de bonnes pratiques à respecter et non d'une nouvelle technologie à part entière.

❖ Concept de l'architecture REST (1/3)

Client-server : cette contrainte maintient le couplage faible du client et du serveur.

- Le client n'a pas besoin de connaître les détails de la mise en œuvre sur le serveur et le serveur ne s'inquiète pas de la façon dont les données sont utilisées par le client.
- Cependant, une interface commune est maintenue entre le client et le serveur pour faciliter la communication.

Pourquoi des APIs REST ?

❖ Concept de l'architecture REST (2/3)

Stateless (Sans état) : le service ne devrait pas avoir besoin de conserver les sessions des utilisateurs. En d'autres termes, chaque demande doit être indépendante des autres.

Cacheable : Cette contrainte doit prendre en charge un système de cache. L'infrastructure réseau doit prendre en charge un cache à différents niveaux. La mise en cache peut éviter des allers et retours répétés entre le client et le serveur pour récupérer la même ressource.

Pourquoi des APIs REST ?

❖ Concept de l'architecture REST (3/3)

Layered system(Système en couches) : le serveur peut avoir plusieurs couches pour la mise en œuvre. Cette architecture en couches contribue à améliorer l'évolutivité en permettant l'équilibrage de la charge. Il améliore également les performances en fournissant des caches partagés à différents niveaux. La couche supérieure, en tant que porte d'accès au système, peut également appliquer des stratégies de sécurité.

Code on demand (code à la demande) : Le code à la demande signifie que le serveur peut étendre sa fonctionnalité en envoyant le code au client pour téléchargement. C'est facultatif, car tous les clients ne seront pas capables de télécharger et d'exécuter le même code – donc ce n'est pas utilisé habituellement, mais au moins, vous savez que ca existe !

Pourquoi des APIs REST ?

Les avantages clés des API REST sont les suivants :

- La séparation du client et du serveur, qui aide à **scaler** plus facilement les applications ;
- Le fait d'être **stateless**, ce qui rend les requêtes API très spécifiques et orientées vers le détail ;
- La possibilité de mise en cache, qui permet aux clients de sauvegarder les données, et donc de ne pas devoir constamment faire des requêtes aux serveurs.

Quelques exemples APIs REST ?

Nous disposons de nombreuses APIs dont on peut citer entre autre:

- PayPal API pour intégrer le mode de paiement PayPal dans nos applications;
- Google Maps API permettant d'utiliser le Maps de Google dans nos applications;
- Twitter API pour avoir accès aux API de Twitter.
- Etc.

Format de données d'échanges (XML, JSON)

❖ XML (Extensible Markup Language)

XML a été conçu pour transporter des données, pas pour afficher des données. C'est une recommandation du W3C. Le langage XML est un langage de balisage qui définit un ensemble de règles de codage des documents dans un format lisible. Les objectifs de conception de XML se concentrent sur la simplicité, la généralité et la convivialité.

Exemple XML

```
1.  <employees>
2.    <employee>
3.      <name>Alex</name>
4.      <age>23</age>
5.    </employee>
6.    <employee>
7.      <name>Bob</name>
8.      <age>34</age>
9.    </employee>
10. </employees>
```

Format de données d'échanges (XML, JSON)

❖ JSON (JavaScript Object Notation)

JSON est un format léger d'échange de données indépendant du langage. Il est basé sur JavaScript et est facile à comprendre et à générer.

Exemple JSON

```
1.  {"employees":[  
2.      {"name":"Alex", "age":23},  
3.      {"name":"Bob", "age":34},  
4.  ]}
```

Format de données d'échanges (XML, JSON)

Table de comparaison

JSON	XML
JSON signifie JavaScript Object Notation.	XML signifie eXtensible Markup Language.
JSON est simple à lire et à écrire.	XML est moins simple que JSON.
JSON est facile à apprendre.	XML est moins facile que JSON.
JSON est orienté données.	XML est orienté document.
JSON ne fournit pas la possibilité d'afficher les données.	XML permet d'afficher les données car il s'agit d'un langage de balisage.
JSON prend en charge les tableaux.	XML ne supporte pas les tableaux.
JSON est moins sécurisé que XML.	XML est plus sécurisé.
Les fichiers JSON sont plus lisibles que XML.	Les fichiers XML sont moins lisibles par l'homme.
JSON prend en charge uniquement les types de données text et number.	XML prend en charge de nombreux types de données tels que texte, nombre, images, etc. De plus, XML offre des options pour transférer le format ou la structure des données avec les données réelles.

RESUME DU CHAPITRE I

- ❖ **Microservices** est un modèle d'architecture orientée service dans lequel les applications sont construites sous la forme d'une collection de différentes unités de service indépendantes. Il s'agit d'une approche de génie logiciel axée sur la décomposition d'une application en modules à fonction unique dotés d'interfaces bien définies. Ces modules peuvent être déployés et exploités indépendamment par de petites équipes qui maîtrisent l'ensemble du cycle de vie du service.
- ❖ C'est dans cette optique que les APIs REST / RESTFul ont vu le jour.

CHAPITRE II : LES BASES DU JSON

JSON (Javascript Object Notation)

- JavaScript Object Notation ;
- Initialement créé pour la sérialisation et l'échange d'objets JavaScript ;
- Langage pour l'échange de données semi-structurées (et éventuellement structurées) ;
- Format texte indépendant du langage de programmation utilisé pour le manipuler.

Utilisation première : échange de données dans un environnement Web (par exemple applications Ajax)

Extension : sérialisation et stockage de données

Les Bases de JSON (1/4)

- Structure de base : paire clef-valeur (key-value)

```
"title": "The Social network"
```

- **Qu'est-ce qu'une valeur** ? On distingue les valeurs **atomiques** et les valeurs **complexes** (construites)
- **Valeurs atomiques** : chaînes de caractères (entourées par les classiques guillemets anglais (droits)), nombres (entiers, flottants) et valeurs booléennes (true ou false).

```
"year": 2010
```

```
"oscar": false
```

Les Bases de JSON (2/4)

- Valeurs complexes : les **objets**.
- Un objet est un ensemble de paires **clef-valeur**.
- Au sein d'un ensemble de paires, une clef apparaît au plus une fois (NB : les types de valeurs peuvent être distincts).

```
{"last_name": "Fincher", "first_name": "David"}
```

- Un objet peut être utilisé comme valeur (dite complexe) dans une paire clef-valeur.

```
"director": {  
  "last_name": "Fincher",  
  "first_name": "David",  
  "birth_date": 1962  
}
```

Les Bases de JSON (3/4)

- Valeurs complexes : les **tableaux**.
- Un tableau (array) est une liste de valeurs (dont le type n'est pas forcément le même).

```
"actors": ["Eisenberg", "Mara", "Garfield", "Timberlake"]
```

Imbrication sans limite : tableaux de tableaux, tableaux d'objets contenant eux-mêmes des tableaux, etc.

Les Bases de JSON (4/4)

- Un document est un objet. Il peut être défini par des objets et tableaux imbriqués autant de fois que nécessaire.

```
{
  "title": "The Social network",
  "summary": "On a fall night in 2003, Harvard undergrad and \n
             programming genius Mark Zuckerberg sits down at his \n
             computer and heatedly begins working on a new idea. (...)"
  "year": 2010,
  "director": {"last_name": "Fincher",
               "first_name": "David"},
  "actors": [
    {"first_name": "Jesse", "last_name": "Eisenberg"},
    {"first_name": "Rooney", "last_name": "Mara"}
  ]
}
```


JSON vs XML

- JSON plus léger et intuitif que XML,
- Facile à parser pour n'importe quel langage de programmation,
- JSON n'a pas (encore) de langage de spécification de schéma associé,
- JSON n'a pas (encore) de langage de requête associé.

Quelques liens utiles

- Tout sur JSON : <http://json.org/>
- Un validateur de documents JSON : <http://jsonlint.com/>
- Une proposition de schéma pour JSON : <http://json-schema.org/>
- Un langage de requêtes JSON : JAQL, <http://code.google.com/p/jaql/>

CHAPITRE III : MÉTHODES ET CODES HTTP

Contenu du chapitre III

- ❑ Notion de ressources, requêtes, méthodes et réponse ;
- ❑ Comprendre les verbes HTTP (Get, Post, Put, Patch, Delete) ;
- ❑ Les codes HTTP (1xx, 2xx, 3xx, 4xx, 5xx).

HTTP

HyperText Transfer Protocol

- A l'origine, protocole permettant de publier et de retrouver des pages
- Version actuelle: HTTP/1.1

Construit au-dessus de TCP, port 80 par défaut (ou 443 en SSL)

Protocole requête/réponse **sans état**

HTTP est stateless

- cookies (client-side)
- session (server side)

HTTP: format requête/réponse

Basé sur du texte

Requête

- Méthode + chemin + version HTTP
- Hôte
- Ligne vide
- Contenu optionnel

```
GET /index.html HTTP/1.1  
Host: www.example.com
```

HTTP: format requête/réponse

Basé sur du texte

Réponse

- Version HTTP + code d'état
- Entêtes
- Ligne vide
- Contenu optionnel

```
HTTP/1.1 200 OK
Date: Mon, 10 November 2008 08:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 02 Jan 2008 23:11:55 GMT
Accept-Ranges: bytes
Content-Length: 10
Connection: close
Content-Type: text/html; charset=UTF-8

<html><body>Hello world!</body></html>
```

HTTP: méthodes

Quelles sont les méthodes HTTP ?

- pour récupérer la représentation d'une ressource: **GET**
- pour créer une nouvelle ressource: **PUT** à un nouvel URI, **POST** à un URI existant
- pour modifier une ressource existante: **PUT** à un URI existant
- pour supprimer une ressource existante: **DELETE**
- pour obtenir les méta-données d'une ressource existante: **HEAD**
- pour connaître les verbes que la ressource comprend: **OPTIONS**

Verbes HTTP

GET

- Utiliser pour la récupération d'une ressource
 - Cachable (peut être mis en cache);
- Ne jamais utiliser pour modifier quoi que ce soit côté serveur

Verbes HTTP

POST

- Utiliser pour la création d'une nouvelle ressource
 - Peut être utilisé à la place d'un GET si la requête est trop grande (paramètres)
- Ne jamais utiliser pour récupérer une information
 - N'est jamais mis en cache

Verbes HTTP

PUT

- Utiliser pour la modification d'une ressource existante
 - Sensé être idempotent
 - Doit être préféré à PATCH même s'il est sensé être plus approprié dans les mises à jour partielles
- Ne jamais utiliser pour récupérer une information
 - N'est jamais mis en cache

Verbes HTTP

DELETE

- Utiliser pour supprimer une ressource existante
- Ne jamais utiliser pour autre chose qu'une suppression de ressource
 - N'est jamais mis en cache

Verbes HTTP

OPTIONS

- Doit au minimum retourner une réponse HTTP **200 OK**
- Présenter un header « Allow » listant les méthodes utilisables sur la ressource ciblée
- Exemple :

200 OK

Allow: HEAD, GET, POST, PUT, DELETE, OPTIONS

- Peut aussi retourner d'autres informations

Résumé des Verbes HTTP

Verbe (méthode)	Sémantique	Propriété
<i>GET</i>	recupère une représentation	idempotent
<i>PUT</i>	met à jour une représentation	idempotent
<i>DELETE</i>	supprime une ressource	idempotent
<i>POST</i>	crée une nouvelle ressource	-

Les codes HTTP

Quels sont les codes d'état de HTTP ?

- **1xx** : méta-données
- **2xx** : tout va bien
- **3xx** : redirection
- **4xx** : le client a fait quelque chose de pas correct
- **5xx** : le serveur a fait quelque chose de pas correct

Les codes HTTP

100 – Continue

200 – OK, **201** – Created, **202** – Accepted, **203** – Non Authoritative Information, **204** – No Content, **206** – Partial Content ;

300 – Multiple Choices, **301** – Moved Permanently, **303** – See Other, **304** – Modified, **307** – Temporary Redirect;

400 – Bad Request, **401** – Unauthorized, **403** – Forbidden, **404** – Not Found, **405** – Method Not Allowed, **406** – Not Acceptable, **409** – Conflict, **410** – Gone, **411** – Length Required, **412** – Precondition Failed, **413** – Request Entity Too Large, **414** – Request URI Too Long, **415** – Unsupportable Media Type;

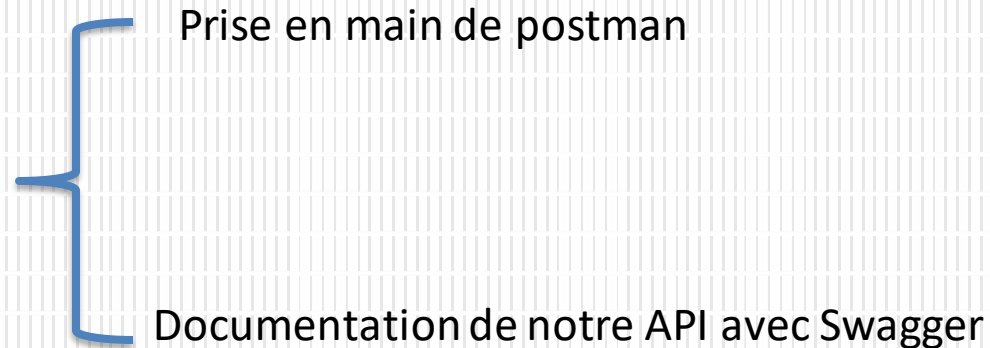
500 – Internal Server Error, **501** – Not Implemented, **502** – Bad Gateway, **503** – Service

HTTP : headers

Les headers HTTP à connaître

- **Authorization:** contient les certificats de sécurité (basic, digest, ...)
- **Content-Length:** longueur du contenu
- **Content-Type:** le format de représentation de la ressource
- **Etag/If-None-Match:** checksum, pour les GET conditionnels
- **If-Modified-Since/Last-Modified:** pour les GET conditionnels
- **Host:** domaine de l'URI
- **Location:** localisation d'une ressource créée/déplacée

TRAVAUX PRATIQUE



MARDI 12 Avril 2022

PRISE EN DE SWAGGER

Qu'est-ce que Swagger ?

- Par le passé, en raison des multiples technologies et langages de programmation existants, la description des **API** était très complexe. Une première grande étape dans le processus d'organisation des API a été la création du **paradigme de programmation REST**. Les sites Internet comme Google, Amazon et Twitter utilisent des API RESTful. Auparavant, les interfaces étaient décrites dans le Web Service Description Language **WSDL**. WSDL 2.0 permettait aussi de décrire des **API REST** d'un point de vue purement technique, une pratique cependant très peu commode pour les développeurs. Le Web Application Description Language (WADL) devait remédier à ce problème, mais n'a jamais été standardisé en raison de sa structure en XML.

Qu'est-ce que Swagger ?

- Ainsi, **Swagger** est vite devenu la technologie la plus appréciée pour la **documentation API**. Plus précisément, pour documenter les **API REST**, très souvent utilisées. Swagger a été développé par **Reverb**, mais il s'agit désormais d'une solution indépendante et **open source**, sous la gouvernance de la fondation Linux, et plus précisément de l'**OpenAPI Initiative**. Avec ce changement d'acteurs, Swagger a été rebaptisé « **spécification OpenAPI** », même s'il reste officieusement connu sous le nom plus accrocheur de « Swagger ».
- **Swagger** peut servir notamment dans des applications développées avec Spring Boot, Django tout comme dans **Laravel** et **Symfony**.

Qu'est-ce que Swagger ?

- L'élément central de Swagger est, en fonction de son champ d'application, un fichier au format **JSON** ou **YAML**. L'interface utilisateur, qui permet de représenter très facilement la documentation, fonctionne avec HTML et JavaScript. En principe, Swagger présente un format indépendant de tout langage informatique et lisible par une machine. À partir de l'interface utilisateur, on peut non seulement administrer la documentation, mais aussi s'en servir pour que Swagger effectue, par exemple, des **tests ad hoc**.
- Swagger présente un avantage évident avec son mode développement, assuré par une bibliothèque centrale, **Swagger Core**. L'interface utilisateur s'appelle **Swagger UI**, le générateur de code **Swagger Codegen**, et il existe aussi un **Swagger Editor**.
- Mais la grande force de Swagger, tout comme de nombreuses autres solutions open source, réside dans l'écosystème complet de **GitHub**. Là, les développeurs trouveront des générateurs de code pour presque tous les langages de programmation. Swagger documente chaque interface avec toutes les informations.

Dans quelle cas utilise t'on swagger ?

- Développer une API exige de disposer d'une documentation ordonnée et compréhensible. Elle seule permet aux développeurs de se servir d'une interface. C'est encore plus vrai pour les API publiques : sans documentation, elles sont inutilisables pour la communauté, ne sont pas complétées et ne rencontrent aucun succès.
- À l'heure actuelle, Swagger est la meilleure solution pour documenter une **API REST**, car le système est capable de représenter presque tous les services Web et informations ayant trait à l'interface. La documentation évolue en même temps que le système et enregistre automatiquement les modifications. Swagger se montre particulièrement efficace pour parvenir à ce résultat parce qu'il consigne la documentation d'une API REST directement dans son code.

Dans quelle cas utilise t'on swagger ?

- Le point de départ de tout projet de développement d'API est soit le code de programmation (approche « **Code first** »), soit la description de l'interface (approche « **Design first** »). Dans le cas d'un projet en **Code first**, le point de départ convenu est le code. Swagger peut directement déduire la documentation à partir du code de programmation et produire des ressources indépendantes du langage de programmation utilisé, lisibles aussi bien par des machines que par des humains.
- À l'autre bout du paradigme, l'approche **Design first**. Comme nous l'avons déjà évoqué, bien souvent, le client et le serveur sont gérés par des développeurs différents. L'approche **Design first** préconise de commencer par rédiger la description. Ensuite, les codes sont simplement générés par Swagger. Pour ce faire, il existe des générateurs pour tous les langages de programmation courants. Même les modèles peuvent encore être adaptés ou complétés.

Dans quelle cas utilise t'on swagger ?

- Swagger propose ainsi presque automatiquement un **code API cohérent**. Si quelque chose ne correspond pas dans la programmation manuelle, on rencontre des erreurs de compilation, que l'utilisation d'un système d'intégration continue permet de visualiser automatiquement.

Swagger : Les avantages d'une programmation bien ordonnée

- Les avantages de Swagger sont si convaincants qu'on peut tout à fait qualifier Swagger d'**application standard exceptionnelle pour la description d'interfaces REST**. Comme beaucoup d'autres applications open source, Swagger bénéficie d'une large diffusion, accompagnée d'un support complet. Le comité de Swagger se compose de géants de la tech, comme Microsoft, IBM ou Google, ce qui assure un soutien pérenne à la spécification OpenAPI, même s'il existe des alternatives. Le Restful API Modelling Language (RAML) s'appuie par exemple aussi sur le YAML et peut même créer des définitions encore plus complexes que celles de Swagger. Le créateur de RAML (Mulesoft) a cependant lui-même intégré l'OpenAPI Initiative.

Swagger : Les avantages d'une programmation bien ordonnée

- L'intelligibilité et la lisibilité de la documentation de Swagger restent un petit inconvénient. La solution propose certes un format déjà plutôt bien structuré, mais il faut malgré tout un peu de temps pour se l'approprier. Pourtant, certains langages démontrent qu'il est possible de faire plus simple, comme API Blueprint et sa syntaxe en **Markdown**.

Documentation de notre API avec Swagger

The screenshot shows the Swagger UI for an API titled "Management of User and Product List Api". The interface includes a top navigation bar with the Swagger logo, a search bar containing the URL "http://127.0.0.1:8000/docs/api-docs.json", and an "Explore" button. Below the header, the API title is displayed with version "1.0" and "OAS3" tags. A subtitle "Demo of User and Product List Api" is also present. A "Filter by tag" input field is located below the subtitle. The main content is organized into two sections: "Products Management" and "Users Management", each with a list of API endpoints. Each endpoint entry shows the HTTP method (GET, POST, PUT, DELETE), the endpoint path, and a brief description. The endpoints are color-coded by method: GET (blue), POST (green), PUT (orange), and DELETE (red). Each entry has a dropdown arrow on the right side.

Swagger
Supported by SMARTBEAR

http://127.0.0.1:8000/docs/api-docs.json Explore

Management of User and Product List Api ^{1.0} OAS3

<http://127.0.0.1:8000/docs/api-docs.json>

Demo of User and Product List Api

Filter by tag

Products Management ^

- GET** `/api/products` Get List of all products
- POST** `/api/products` Create Product
- GET** `/api/products/{id}` Get Detail Product
- PUT** `/api/products/{id}` Update Product
- DELETE** `/api/products/{id}` Delete Product

Users Management ^

- GET** `/api/users` Get List of all users
- POST** `/api/users` Create User
- GET** `/api/users/{id}` Get Detail User
- PUT** `/api/users/{id}` Update User
- DELETE** `/api/users/{id}` Delete User

REFERENCES BIBLIOGRAPHIQUE

- [1] Hinault Romaric, "Comprendre la spécification OpenAPI (Swagger) et apprendre à utiliser Swagger Editor", Décembre 2017;
- [2] Lionel Médini, "Representational State Transfer (REST) et Web AAPs", Décembre 2020;
- [3] Work in progress sur la spécification : <https://tools.ietf.org/html/draft-kelly-json-hal-08>
- [4] Condensé des spécifications en cours de validation : http://stateless.co/hal_specification.html