RESTful API in a nutshell

REST: **Re**presentational **S**tate **T**ransfer
(https://en.wikipedia.org/wiki/Representational_state_transfer)

Signification

Transfère via HTTP d'une **représentation** de l'**état** des ressources d'un système.

Par exemple, si une ressource du serveur est représentée par la classe `Alert`, la représentation transférée au client pourrait être en JSON et non pas l'instance de la classe en question. C'est donc en ce sens que le mot *représentation* est utilisé. De plus, cette représentation JSON ne contiendra que les attributs jugés utiles dans le contexte. Ces attributs et leurs valeurs associées représentent donc l'*état* de cette instance au moment de la requête. Bref, l'on **transfert** une **représentation** de l'**état** de la ressource en question.

Une **API REST** n'est pas dédiée au grand public. Ce n'est pas une jolie interface qui peut être accédée via un navigateur web. C'est au contraire une interface technique qui devrait être accédée par un logiciel client et qui permet au développeur de récupérer les ressources offertes par un serveur selon les besoins de son application.

Il est par la suite de la responsabilité du développeur (et de l'application client) d'afficher correctement les informations recueillit. Si le client est un navigateur, ça implique qu'il y a des scripts JavaScript qui se chargent de récupérer et d'afficher les données, l'utilisateur ne vois quant à lui que la page d'entrée du client HTML.

Bref, **REST API** implique **interface technique** — comme l'indique d'ailleurs l'utilisation de l'acronyme API (**A**pplication **P**rogramming **I**nterface).

Bon à savoir :

L'API REST utilise le protocole HTTP comme couche de transport. L'avantage est qu'aucun effort ne doit être fait pour le protocole d'échange d'information puisqu'il existe déjà une pléthore de librairie supportant HTTP. De surcroit, l'on hérite de la sécurité inhérente au protocole HTTPS et OAuth.

Le protocole HTTP permet à un client d'envoyer une requête à un serveur qui retourne une réponse.

La requête HTTP est constituée d'une entête (**header**) et d'un contenu (**body**).

L'entête contient (entre autre) :

- la méthode/verbe HTTP (telles que `GET, POST, PUT, PATCH, DELETE`.),
- une indication du type de contenu envoyé (par exemple, `Content-type=application/json`),
- une indication du type de contenu accepté (par exemple, `Accept=application/json`).

Le body contient les données à envoyer au serveur. Le body peut également être vide. Il est de pratique courante d'utiliser le format JSON dans les communications entre le serveur et le client, mais XML est également supporté.

La réponse HTTP est également constituée d'une entête (**header**) et d'un contenu (**body**).

L'entête contient (entre autre) le `status code` indiquant le succès ou l'échec de la requête (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

Le body peut être vide, ou contenir le résultat de la requête (en JSON généralement) ou encore une explication pour indiquer pourquoi la requête a échouée. Cette dernière explication, dédiée à l'utilisateur final, pourrait être affiché par l'application client.

Règles

Les règles pour construire une API REST sont simples :

1. **Premier niveau** : Ressources
   - ✓ **Toutes les URL** doivent conduire à des ressources. Par voie de conséquences les URL doivent être constituées de **noms**, et ne doivent pas contenir de verbes. Par exemple `api/alerts` est conforme, mais `api/getAlerts` ne l'est pas.
   - ✓ Par défaut, un lien donne accès à une collection (un ensemble) de donnée, on emplois donc le pluriel pour clairement l'indiquer. Par exemple `api/user/alerts`, indique que ce lien retournera l'ensemble des alertes de type `user`.
   - ✓ Pour avoir accès une ressource spécifique, l'on doit ajouter au « *path* » de l'URL l'identifiant unique de cette ressource. Par exemple `api/user/alerts/DNS-3452776` donnerait accès à l'alerte de type `user` ayant l'identifiant unique `DNS-3452776`.
2. **Second niveau** : Verbes HTTP
   - ✓ Utiliser les verbes/méthodes HTTP (`GET, POST, PUT, PATCH, DELETE`) dans le header de la requête pour indiquer l'action. Par exemple un `GET` sur l'URL `api/user/alerts` retournera l'ensemble des alertes de type `user`, tandis qu'un `POST` ajoutera une nouvelle alerte à la collection. À noté dans ce dernier cas, le body de la requête (et non pas les paramètres de l'URL) doivent contenir une représentation de l'alerte que l'on désire créer. (Voir tableau plus bas pour l'interprétation des verbes selon le contexte.)
   - ✓ La réponse HTTP retournée par le serveur doit contenir un `status code` cohérent avec la requête. Par exemple, un `GET` réussis devrait retourner un `status code 200 OK`, tandis qu'un `POST` réussi devrait retourner un `status code 201 Created`.
   - ✓ Si la requête échoue, le `status code` de la réponse doit être représentatif de la raison de l'échec (`4xx` pour `client error`, `5xx` pour `server error`). De plus, il est de bonne pratique que le body de la réponse contienne au moins un message explicatif pour être affiché par le client (et possiblement le stack que le client cacherait par défaut mais qui pourrait être visualiser pour débogage).
   - ✓ Les paramètres de l'URL sont typiquement utilisés pour filtrer la quantité d'informations retournées par les collections. Par exemple, `api/user/alerts?north=45.5&south=44.5&west=-75.5&east=-74.5` permettrait de ne retourner les alertes de type user contenues entre les latitudes et longitudes passées en paramètre. Un autre exemple typique, l'URL `api/books?start=75&size=32` pourrait permettre de paginer la liste de livre retourner en spécifiant l'identifiant de départ dans une liste chronologique et le nombre d'élément à retourner à partir du livre de départ.
3. **Troisième niveau** : Contrôle par Hypermédia (HATEOAS)
   - ✓ Dans ce contexte, hypermédia est une URL qui pointe vers une ressource.
   - ✓ La réponse à une requête (réussie ou non) doit contenir, en plus de son contenu normal, des liens permettant de naviguer. Par exemple, le lien identifier par `rel="Add User Alert"` indiquerait comment ajouter une nouvelle alerte.
   - ✓ HATEOAS (**H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate), prononcé hate us, défini un format JSON et des règles permettant de guider l'implémentation d'une telle interface.
   - ✓ Le but est de faire en sorte que l'application client n'ait qu'à connaître une seule URL et les mots clefs définissant l'interface (par exemple, `"Add User Alert"`, `"Update User Alert"`) pour pouvoir naviguer sur le serveur. Il n'a pas besoin de savoir construire l'URL pour ajouter une nouvelle alerte, car elle est fournie par l'interface à chaque appel. Ceci permet au serveur de modifier les URL sans impacter les clients.

Table 1 Verbes HTTP en RESTful API

Requête HTTP		Réponse	
Méthode	**body**	**Collection** `localhost:8080/api/user/alerts/`	**Élément** `localhost:8080/api/user/alerts/{id}`
GET	vide	Retourne une représentation de la collection.	Retourne une représentation de l'élément.
POST	représentation du nouvel élément	Ajoute un nouvel élément à la collection.	Considère l'élément comme une collection et ajoute un sous-élément constituant de ce dernier. (Par exemple, chaque alerte pourrait avoir une collection de photos.)
PUT	représentation du nouvel élément/collection	Remplace la collection par une nouvelle.	Remplace l'élément par un nouveau.
PATCH	représentation des attributs modifiés	Erreur	Mets à jours les attributs modifiés de l'élément.
DELETE	vide	Détruit la collection au complet	Détruit l'élément.

L'idéal RESTful API

Level 3	Hypermedia Controls (HATEOAS)
Level 2	HTTP Verbs
Level 1	Resources
Level 0	The swamp of POX (Plain Old XML)