REST

date	révision
Septembre 2019	Création ()
09/10/2019	Ajout codes d'erreurs. Ajout frontend et backend.
31/05/2022	Ajout des statuts HTTP (annexe), ajout du verbe PATCH et de quelques outils
11/05/2023	Ajout des statuts HTTP complet en annexe (remplacement)

SOMMAIRE

REST

L'activité proposée ici aborde l'utilisation des architectures RESTful.

À l'issue de cette activité, l'étudiant devra :

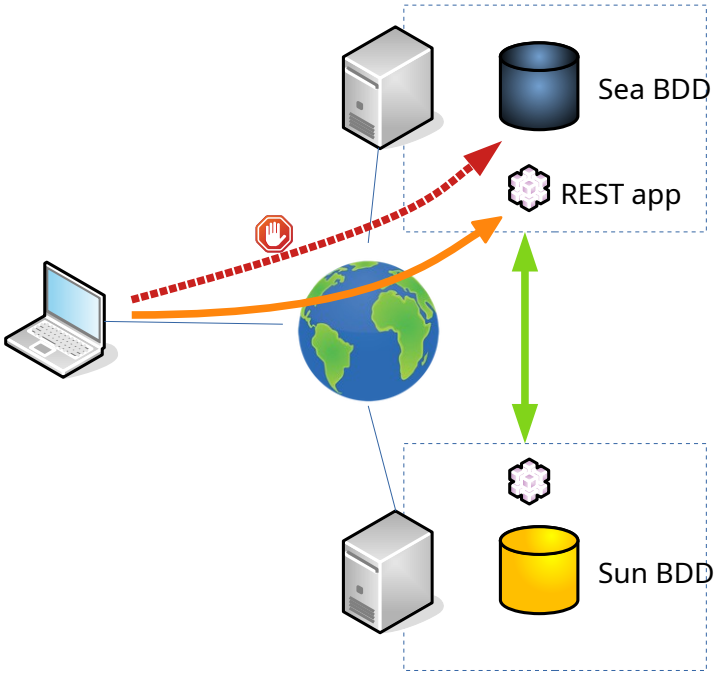
- Comprendre les avantages des applications REST
- Connaître la syntaxe REST
- Pouvoir utiliser une application REST

 **Durée**	 **Difficulté**	 **Taxonomie Bloom**
1 séance	❶ ❷ ❸ ④	Compréhension – Application

1 API RESTFUL

1.1 INTRODUCTION À L'ARCHITECTURE RESTFUL

Cette architecture[1] est née d'un besoin de simplifier les accès aux données des différents services, tout en assurant une parfaite sécurité.



Sur l'exemple ci-contre, il serait dangereux de laisser les utilisateurs Internet accéder directement à la base de données du site :

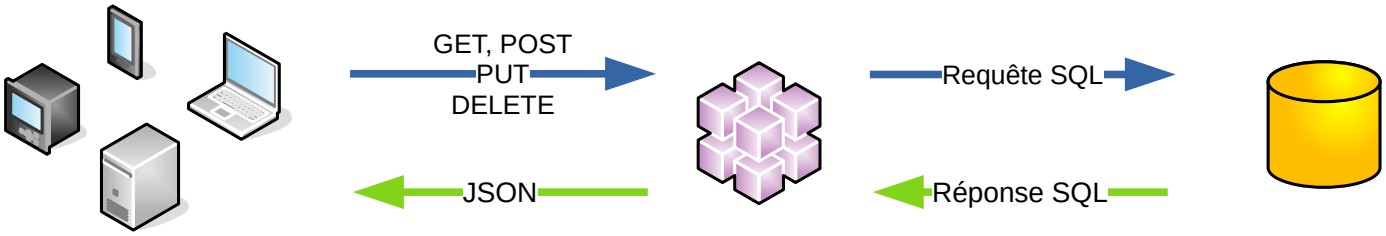
- risque de piratage
- risque de modifications
- obligation de créer des comptes génériques pour gérer les droits
- ...

La solution est un ensemble de codes réagissant aux requêtes définies dans le protocole HTTP.

Décrite initialement dans la thèse de Roy Thomas FIELDING en 2000, architecture est basée sur un ensemble de contraintes nécessaires pour permettre la distribution de données dans les services web.

L'acronyme REST provient de "Representational state transfer", ce qui signifie transfert d'état représentationnel.

L'intérêt est de permettre à n'importe quel client, d'accéder aux données que l'on veut rendre accessible. La description de ces données utilise JSON, pour ne plus être dépendant de l'encodage de chaque base de données. Le client accède au "*frontend*" qui interroge le "*backend*".



1 *Ce n'est pas un protocole !!!*

1.2 RÈGLES ET SYNTAXE

Pour utiliser un service RESTful ou pour en créer un, il faut respecter un certain nombre de règles.

1.2.1 URI sert d'identifiant pour les ressources

Une URI sert de point d'entrée pour un type de ressource. Il est important de formater les URI de manière logique, en utilisant correctement la syntaxe des URL HTTP.

- Liste de livres : on utilisera le mot au pluriel ; http://localhost:3000/livres
- filtre sur la liste : on utilisera les paramètres GET et un query string ; http://localhost:3000/livres?filtre=roman
- affichage d'un seul livre : http://localhost:3000/livres/87
- affichage de tous les commentaires sur ce livre ; http://localhost:3000/livres/87/comments

Si l'application doit gérer des utilisateurs et des prêts, la logique sera identique : le point d'entrée du site détermine l'ensemble des fonctions choisies. On n'affiche pas forcément une page HTML, ce sont donc des services.

Note : on n'utilise pas de verbe mais plutôt les noms des objets à traiter.

1.2.2 Méthodes HTTP pour les opérations

Le protocole HTTP propose plusieurs verbes et méthodes, il suffit de les employer correctement :

- Créer : **POST**
- Afficher : **GET**
- Mettre à jour (totale sur l'élément) : **PUT**
- Mettre à jour (uniquement des partie de l'élément) : **PATCH**
- Supprimer : **DELETE**

Imaginons une application permettant d'agir sur la table "client" dans une base, on trouvera donc les liens suivants dans l'API :

• GET	/api/clients	afficher la liste des clients
• GET	/api/clients/14	afficher le client N°14
• PUT	/api/clients/14	mettre à jour client 14 (données dans le message)
• DELETE	/api/clients/14	effacer le client 14
• POST	/api/clients	ajouter un client à la liste (données dans message)

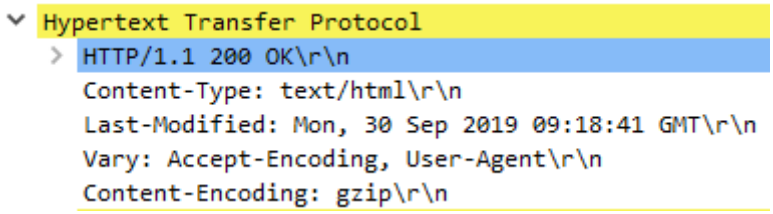
1.2.3 Réponses HTTP et représentation des ressources

L'entête des réponses HTTP doit préciser le format des données suivantes.de choisir un type largement reconnu, comme JSON.

Réponse HTML : text/html

Réponse XML : application/xml

Réponse JSON : text/json



1.3 Les codes de status

Dans le protocole HTTP, il est prévu un code de statut, inspiré des autres protocoles comme FTP, Telnet...

Ce code doit être utilisé pour informer le client du résultat de sa requête :

Code de statut	Explications	niveau
1xx	Information. Utile pour indiquer qu'une requête a été reçue mais pas encore traitée.	
2xx	Succès. Traitement de la requête terminé avec succès	
3xx	Redirection. 301 et 302 indiquent une ressource déplacée temporairement ou définitivement. 304 indique que le cache n'a pas été modifié.	
4xx	Erreur du client. La requête demandée ne peut pas être traitée. Par exemple, le client demande une page inexistante, cela génère une erreur 404.	
5xx	Erreur du serveur. Le serveur ne peut pas traiter la demande. L'erreur 502 est souvent associée au message "Bad Gateway" et 500 à "Erreur Interne".	

Voir annexe pour les détails

1.3.1 Autres règles

Il est utile de s'intéresser aux autres règles, mais elles sont moins intéressantes dans ce cours.

https://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles/

1.4 Outils

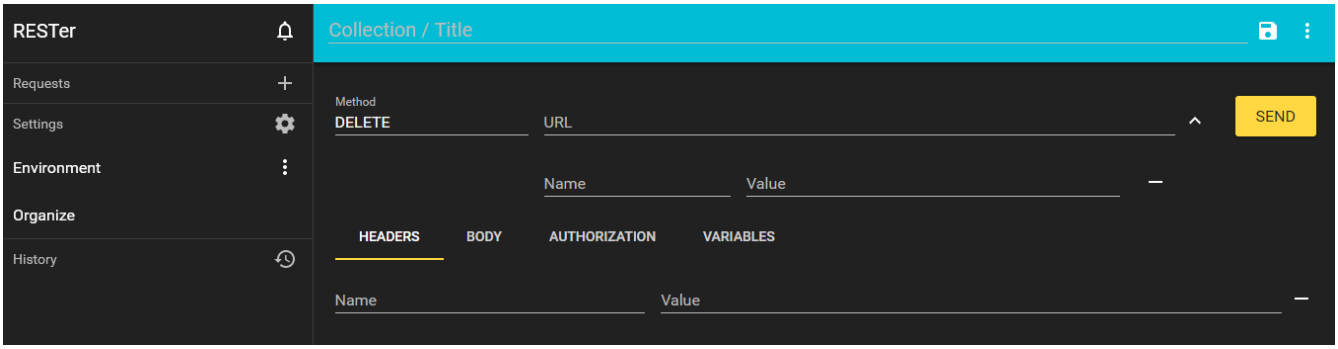
Pour simuler les requêtes GET, une simple consultation de l'URL suffit, les données sont renvoyées au format JSON (en général).

Ex : https://ipinfo.io/193.49.246.51/geo

Cependant, la création de requête POST nécessite un formulaire HTML et les requêtes PUT et DELETE ne peuvent pas être facilement simulées.

1.4.1 RESTER

RESTer est une extension disponible sous Chrome ou FireFox qui permet d'éditer des requêtes HTTP.



1.4.2 RESTman ou POSTman

Une autre extension connue est RESTman, cependant elle n'existe que sous Chrome.

1.5 Conclusion

Les API REST deviennent un standard de fait : c'est-à-dire que ce format et cette architecture est tellement pratique que de plus en plus de développeurs l'utilisent, jusqu'au moment où elle surclasse totalement les autres méthodes.

Elle remplace notamment l'architecture SOAP, basée sur XML, un peu trop verbeux.

2 ANNEXES

2.1 SOURCES

https://www.npmjs.com/package/node-rest-client

https://la-cascade.io/api-les-protocoles/

https://www.prevision-meteo.ch/services

https://blog.nicolashachet.com/niveaux/confirme/larchitecture-rest-expliquee-en-5-regles/

https://www.webnots.com/3xx-http-status-codes/

https://openclassrooms.com/fr/courses/4087036-construisez-une-api-rest-avec-symfony/4280556-une-architecture-pas-un-protocole

2.2 DÉTAILS STATUTS HTTP

Détails des statuts pour le protocole HTTP

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

304 fonctionne avec le système eTag et le caching
401 est une ressource pour laquelle il faut s'identifier (authentification)
403 est une ressource pour laquelle l'utilisateur est authentifié mais n'a pas les droits