Architecture Logicielle

REST : mise en pratique

Comment traduire les contraintes de l'architecture REST en pratique ?

Rappels

- Séparation claire entre le client et le serveur ;Le serveur doit gérer et exposer les **ressources**, pas de connaissance du client
- Le serveur ne maintient pas d'état
- Interface uniforme
- *Cache*
- *Système en couches*

Ressources

- Les ressources correspondent aux entités en BD ou les objets en mémoire
- Représentées par des noms (pas de verbes)
- Accessibles via des url
- Échangées par des représentations (json,xml,. . .)
- Organiser les ressources dans un modèle **arborescent**
- On peut distinguer :
  - Collection de ressource
  - Instance

Interface uniforme

Format des URL : bonnes pratiques

- nom pluriel pour les collections
- id d'une instance pour accéder à un élément
- séparer les éléments par des /
- identifier les ressources de manière unique
- refléter la hiérarchie des ressources
- pas de / final

Exemples d'url bien formée

- `https://monsite.fr/blog/api/v4/articles`
- `https://monsite.fr/blog/api/v4/articles/12`
- `https://monsite.fr/blog/api/v4/articles/12/comments`
- `https://monsite.fr/blog/api/v4/articles/12/comments/3`

Si une ressource est liée à une autre ressource, l'url doit le refléter

- Tous les commentaires de l'article numéro 12 (GET) `https://monsite.fr/blog/api/v4/articles/12/comments`
- Le commentaire numéro 3 de l'article numéro 12 (GET) `https://monsite.fr/blog/api/v4/articles/12/comments/3`

Format des URL : bonnes pratiques

- Paramètres pour des filtres, des tri, ...

- `https://monsite.fr/blog/api/v4/articles?author=toto`
- `https://monsite.fr/blog/api/v4/articles/12/comments?sort=date`

Tous les articles

URL mal formées

- `https://monsite.fr/blog/api/v4/get_all_articles`

URL bien formées

- GET https ://monsite.fr/blog/api/v4/articles

Créer un article

- `https://monsite.fr/blog/api/v4/createArticle`

- POST `https://monsite.fr/blog/api/v4/articles`

Supprimer un article

- `https://monsite.fr/blog/api/v4/deleteArticle`

- DELETE `https://monsite.fr/blog/api/v4/articles`

Un article

- `https://monsite.fr/blog/api/v4/getArticle?id=12`

- `https://monsite.fr/blog/api/v4/articles/12`

HTTP

- GET
- POST
- PUT
- DELETE

CRUD

- CREATE
- READ
- UPDATE
- DELETE

Utiliser PUT pour mettre à jour une instance

Propriétés nécessaires

- GET : Sans effet de bord, Idempotent
- PUT : Idempotent
- DELETE : Idempotent

Idempotent : une opération est idempotente, si elle a le même effet, qu'on l'applique une fois, ou plusieurs fois

HTTP : réponse (succès)

- Renvoi d'un contenu `200 OK`
- Création d'une ressource : `201 Created` (+représentation)
- Modification d'une ressource
  - `200 OK` + représentation si la modification a eu lieu
  - 204 si la ressource n'existe pas (aucun retour)
- Suppression
  - `204 No Content` Si aucun retour
  - 200 + Si message retour

HTTP : réponse (erreur client)

- 400 : Requête mal formée
- 404 : Ressource non trouvée

Négociation de contenu

- Dans le header HTTTP ou en paramètre de l'url
- type de media MIME types

Serveur sans état

- La connexion entre le client et le serveur est sans état
- C'est le client qui doit maintenir cette état
- Le client transmet dans la requête le contexte

En tant que développeur, vous devez identifier

- les éléments contextuels
- les attributs de session
- les paramètres de la requête (ce qui ne change pas)

Cookies

- S'ils contiennent un identifiant de session, cela signifie que les données de la session sont conservées sur le serveur, donc ne respectent pas la connexion sans état
- ne pas utiliser en REST

Session et authentification

- Une session permet au serveur de maintenir une interaction avec le client et se rappeler d'un utilisateur
- données sur le serveur
- problème de ressources, pas RESTful

Comment faire pour transmettre cette gestion au client ?

Authentification

- Token pour la validation de l'identité de l'utilisateur (uniquement)
- Gestion authentification peut être confiée à un tiers
- Moins de consommation mémoire et facilité pour augmenter le nombre de serveur

Hypermedia As The Engine Of Application State

- Le serveur propose des interactions en fournissant des liens
- Le client choisit quels liens suivre
- Pas de gestion du flux applicatifs coté serveur
- Le client doit pouvoir comprendre le sens des liens (ressource obtenue, type de représentation, ...)

Client

- Il est possible de réaliser le client qui va utiliser ces ressources
- Client mobile, desktop, web, ...
- Les ressources peuvent également être utilisées par d'autres serveurs
- Possibilité de combiner plusieurs API