HATEOAS

Hypermedia As The Engine of Application State

HATEOAS c’est quoi ?

- Hypermedia As The Engine of Application State
  - Hypermédia en tant que moteur de l'état d'application
- C’est un contrainte de l’architecture d’application REST
- Le client interagit avec le serveur via des hypermédia
- Le client n’à pas besoin de connaître le fonctionnement du serveur, uniquement comment interagir avec l'hypermédia

un hypermédia c’est quoi ?

C’est une extension du terme Hypertexte pour inclure d’autres types de média (image, audio, vidéo, text, hyperlien)

HATEOAS et REST

Le modèle de maturité de Richardson permet de juger du niveau de conformité avec les contraintes de l'architecture REST.

https://nexworld.fr/hateoas-nouvelle-approche-des-apis-2/

Modèle de maturité de Richardson

0 - RPC

Le minimum pour faire du HTTP

Une seul URL

Uniquement le verbe POST

Code retour 200

Très proche du protocole SOAP

Modèle de maturité de Richardson

1 - Ressource

Une URI par ressource

Uniquement le verbe POST

Code retour 200

Très proche du protocole SOAP

Modèle de maturité de Richardson

2 - Verbe et code HTTP

Une URI par ressource

Utilisation des verbes HTTP
(GET, POST, PUT, DELETE et PATCH)

Utilisation des codes retours HTTP.

- 1XX : information
- 2XX : succès
- 3XX : redirection
- 4XX : erreur côté client
- 5XX : erreur côté serveur

Modèle de maturité de Richardson

3 - HATEOAS

Une URI par ressource

Utilisation des verbes HTTP (GET, POST, PUT, DELETE et PATCH)

Utilisation des codes retours HTTP.

Réponses enrichies par des liens hypermédias pour décrire les transitions entre les différents état de la ressource

Exemple

GET/owners/1

KEY	VALUE
Content-Type	application/hal+json


```
{
    "id": 1,
    "firstName": "Théo",
    "lastName": "HENRY",
    "_links": {
        "self": {
            "href": "http://localhost:8080/owners/1"
        },
        "allPets": {
            "href": "http://localhost:8080/owners/1/pets"
        }
    }
}
```

Avantages

- Découplage client-serveur
  - Le serveur peut évoluer sans impacter le client (modification des URI, de la logique métier, …)
  - Le client ne sait pas comment les URL sont constitué
- Réduire les erreurs côté client
  - uri mal formaté, écrite en dure, …
- La complexité métier est porté par le serveur
  - Réduit les calculs effectué par le client
  - On expose le comportement
- Auto-documentation par exploration des hypermédia
  - Ne remplace pas une documentation claire et exhaustive de l'API

Inconvénients

- La communication entre client et serveur est plus complexe
  - Le serveur doit fournir toutes les transitions possibles
  - Le client doit pouvoir interpréter les hypermédias
- Inadapté pour certaines méthodes
  - Il est compliqué de fournir des informations sur le body attendu
  - Mais fonctionne bien avec les requêtes sans body comme le GET

Dans quel cadre utiliser HATEOAS

- Est-ce qu’il répond à un besoin métier ?
  - HATEOAS n’est pas adapté à toutes les situations

- Est-ce que le rapport coût - avantage est positif

Spring HATEOAS

- Supporte le format HAL
  - Hypertext Application Language

- Fournit un DTO pour représenter la ressource et ses liens
  - RepresentationModel

- Des méthodes pour créer des liens vers les controller Spring MVC

Représenter une ressource HAL

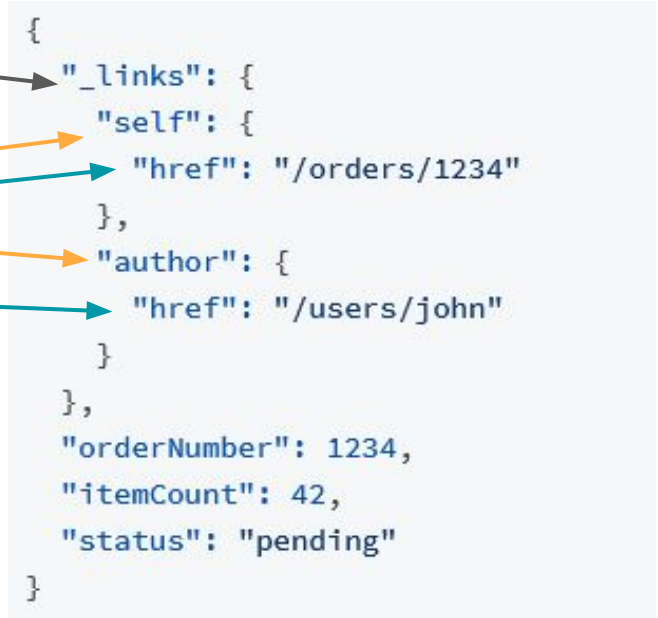
HAL permet de représenter les liens.

Ceux-ci se trouve dans le champ “_links”

HAL présente les relations liées comme une paire :

- un identifiant de la relation
- une url

Ces liens se trouve sous le champ Il est habituel qu’une ressource est une relation liée à elle même (“self”)



Représenter une collection avec HAL

lien vers la collection elle même est toujours là

La collection est encapsulé dans le champ “_embedded”

On retrouve un “_links” pour chaque ressource de la collection

L’encapsulation peut n’être que partielle ou contenir elle même des une ressource “embedded”

Les relations

IANA (Internet Assigned Numbers Authority) à créer une liste des relations standards qu'il peut exister entre des ressources. Link Relations

Quelques exemples :

self	Conveys an identifier for the link's context.
author	Refers to the context's author.
first	An IRI that refers to the furthest preceding resource in a series of resources.
payment	Indicates a resource where payment is accepted.

DTO

Le DTO doit étendre **RepresentationModel**

```
public class Dto extends RepresentationModel<Dto>
```

Cette super classe permet d'ajouter des liens sur votre DTO en faisant un :

```
Dto dto = new Dto();
```

```
dto.add(Link.of("/api/dto"));
```

Une collection de DTO

Pour représenter une collection il faut utiliser la classe `CollectionModel<T>`

Exemple :

```
CollectionModel.of(orderCollection,

linkTo(methodOn(OrderContoller.class)
    .getAllOrders())
.withSelfRel());
```


```
{
  "_links": {
    "self": { "href": "/orders" }
  },
  "_embedded": {
    "order": [
      {
        "_links": {
          "self": { "href": "/orders/1" }
        },
        "orderNumber": "1",
        "status": "pending"
      },
      {
        "_links": {
          "self": { "href": "/orders/2" }
        },
        "orderNumber": "2",
        "status": "cancelled"
      }
    ]
  }
}
```

Jackson Json

Pour que les attribut soit sérialisés,

il faut soit :

- qu'il soit public
- fournir un getter
- utiliser **@JsonProperty**
  - avec ou sans modification du nom

```java
public class Owner extends RepresentationModel<Owner> {

    private final int id;
    @JsonProperty
    private final String firstName;
    @JsonProperty("family_name")
    private final String lastName;
    private final String adress;

    public int getId() {
        return id;
    }
```

```json
{
    "id": 1,
    "firstName": "Théo",
    "family_name": "HENRY"
}
```

Jackson Json

Pour que les attribut soit désérialisés,

il faut soit :

- qu'il soit public
- fournir un getter
- fournir un setter
- utiliser **@JsonProperty**
    - avec ou sans modification du nom

Pour qu'un attribut soit ignorer par Jackson → @JsonIgnore

Liens

Spring fournit un objet Link pour représenter les liens portés par le DTO

- HRef → hypertexte référence
- Rel → relation

On peut définir une href à la main avec :
```
Link.of("/dto/" + id).withSelfRel();
```

On peut récupérer l'url d'un controller REST et rajouter des information avec :
```
linkTo(WebController.class).slash(id).withSelfRel();
```

Ou on peut créer automatiquement des liens avec un méthode d'un controller Spring MVC
```
linkTo(methodOn(WebContoller.class).getDtoById(dto.getId())).withSelfRel();
```

Relations :

ici on définit la référence comme "SELF" avec le `.withSelfRel()`

mais on peut définir notre propre relation avec `.withRel("allDtos")`

Webographie

https://apisyouwonthate.com/blog/api-versioning-has-no-right-way

https://restfulapi.net/versioning/

https://datatracker.ietf.org/doc/html/rfc7231

https://www.youtube.com/watch?v=pIkA-aPtkNs

https://dzone.com/articles/versioning-rest-api-with-spring-boot-and-swagger

https://cloud.google.com/architecture/rate-limiting-strategies-techniques

https://en.wikipedia.org/wiki/Token_bucket

https://github.com/bucket4j/bucket4j/blob/master/README.md

https://systemsdesign.cloud/SystemDesign/RateLimiter

https://www.javadevjournal.com/spring/etags-for-rest-with-spring/

https://www.youtube.com/watch?v=pIkA-aPtkNs

https://nexworld.fr/hateoas-nouvelle-approche-des-apis-2/