TD6 : collection, filtrage et pagination

L'objectif du TD est de programmer des requêtes portant sur des collections et comportant du filtrage de valeurs et de la pagination.

1. collection

Programmer la requête qui permet d'obtenir la collection complète des commandes.
L'objet json retourné doit contenir des méta-données incluant notamment le nombre d'éléments dans la collection, et un tableau de commandes. Chaque commande est décrite par ses id, nom du client, date de création et date de livraison, montant et contient un lien vers la ressource commande correspondante.
Les commandes sont triées par date de livraison.
Le json aura donc la forme suivante :

```
{
        "type": "collection",
        "count": 1750,
        "orders": [ {
                "order": {
                        "id": "30d5909a-a273-4220-a19a-e2c907410ccf",
                        "client_name": "Marcel.Renard",
                        "order_date": "2020-12-17 15:09:48",
                        "delivery_date": "2020-12-17 18:29:01",
                        "status": 5 },
                "links": {
                        "self": {"href": "/orders/30d5909a-a273-4220-a19a-e2c907410ccf/" }
                        }
                }, {
                "order": {
                        "id": "6133d128-9781-4f18-92da-dc691b7fe2c4",
                        "client_name": "Gilles.Tessier",
                        "order_date": "2020-12-17 23:28:37",
                        "delivery_date": "2020-12-19 10:43:34",
                        "status": 5 },
                "links": {
                        "self": {"href": "/orders/6133d128-9781-4f18-92da-dc691b7fe2c4/" }
                        }
                }, { ... }
        ]
}
```

2. filtrage

On souhaite pouvoir implanter une fonctionnalité d'accès à l'historique des commandes d'un client. Pour cela, on met en place un mécanisme de filtrage des commandes. On met en place un paramètre optionnel dans l'uri collection sur les commandes, permettant de sélectionner le client choisi.

Ainsi, la requête :
`GET /orders` retourne toutes les commandes enregistrées et la requête :
`GET /orders?c=amelie.poulain%40montmartre.fr` retourne toutes les commandes dont la cliente est Amélie Poulain .
L'objet json retourné aura la même forme que dans le cas de l'uri non filtré.

3. tri
Mettre en place un mécanisme de tri de la réponse. On doit pouvoir trier selon la date (inverse) et selon le montant total décroissant :

```
GET /orders?c=amelie.poulain%40montmartre.fr&sort=date
GET /orders?c=amelie.poulain%40montmartre.fr&sort=amount
```

4. pagination
On souhaite mettre en place un mécanisme de pagination permettant de parcourir la collection lorsqu'il y a un grand nombre de données. Le mécanisme de pagination doit être utilisable en même temps que les filtres.

Le principe consiste à ajouter dans l'uri un numéro de page. Par défaut, on considère que les pages contiennent 10 éléments.
Ainsi, l'uri :

- `GET /orders` : par défaut, retourne la 1ère page, c'est-à-dire les 10 premiers éléments,
- `GET /orders?page=3` : retourne la 3ème page, c'est-à-dire, 10 éléments à partir du 21ème.

Les méta-données de la réponse indiquent :

- `count` : le nombre d'éléments total répondant à la requête (indépendamment de la pagination)
- `size` : le nombre d'éléments dans la page courante

5. pagination avancée

Compléter le mécanisme de pagination avec les fonctionnalités suivantes :

1. si le numéro de page demandé est < 0, retourne la 1ère page,
2. si le numéro de page demandé est supérieur au nombre de pages, retourne la dernière page.
3. des liens permettant de parcourir les pages sont ajoutés dans la réponse.

```
GET /orders?page=4
```

```
{
 "type": "collection",
 "count": 1502,
 "size": 15,
 "links": {
        "next": {
            "href": "/orders/?page=5"
        },
        "prev": {
            "href": "/orders/?page=3"
        },
        "last": {
            "href": "/orders/?page=94"
        },
        "first": {
            "href": "/orders/?page=1"
        }
    },


 "orders": [

     …
 ]
}
```