Une liste de publications sur HAL

L'objectif de ce second TD sera pour vous de récupérer une liste de publications à la saisie de tout IdHAL dans le formulaire HTML. Vous mettrez également en place un système de pagination simplifié afin de n'afficher qu'un certain nombre de références à la fois. La mise en place de la chaîne de traitement vous amènera à manipuler l'API *Fetch* et la mécanique des *Promises* dans une logique de code asynchrone.
URL de l'API de recherche de HAL : https://api.archives-ouvertes.fr/search/
Documentation : https://api.archives-ouvertes.fr/docs/search/

Définir les objectifs

Identifier la donnée à récupérer

La requête traitée dans le premier TD permettait non seulement de récupérer le nombre de documents mais également la liste des trente premiers. La documentation officielle de l'API de recherche de HAL explique comment augmenter ce nombre ou, mieux, comment débuter la liste à un autre endroit qu'à l'indice 0. En suivant les recommandations, vous pouvez redéfinir le format de la requête de cette façon :

```
https://api.archives-ouvertes.fr/search/?q=authIdHal_s:<IdHAL>&start=<Number>
```

La réponse est toujours au format JSON :

```
{
    "response": {
        "numFound": Number(),
        "start": 0,
        "docs": […]
    }
}
```

Le code de départ

Le code de départ n'est plus tout à fait le même depuis le dernier TD. Vous le trouverez comme précédemment dans le dossier *debut*. La page *index.html* es agrémentée de quelques blocs pour accueillir la liste des documents ainsi que la pagination.

Dans le même dossier, un script à compléter *api-hal.js* reprend les apports du premier TD avec toutefois quelques aménagements pour vous permettre de ne vous concentrer que sur la mise en place de la chaîne de traitement.

Remarquez en tête l'objet `config`, composé de propriétés initialisées avec des valeurs par défaut (la plupart vides). Attendez-vous à interagir fréquemment avec cet objet dans toutes les fonctions et procédures que vous écrirez.

Conception du script

En se basant sur les fonctions disponibles dans le code, qu'elles soient entièrement ou partiellement rédigées, vous dégagerez sans doute une chaîne de traitement similaire à celle-ci :

1. lancer la requête ;
2. définir le nombre de documents ;

3. établir la pagination ;
4. afficher les documents ;
5. gérer les erreurs éventuelles.

Relevez dès maintenant les éléments d'interaction avec l'utilisateur :

- Comment régler le passage d'une page à une autre ?
- Que se passe-t-il lorsqu'une nouvelle recherche est soumise ?
- Et si la recherche n'aboutit pas ?

Étape 1 : lancer la requête

L'extrait de code suivant lance une fonction `getDocs()`, à savoir la chaîne de traitement, lorsque le champ de saisie de l'identifiant HAL perd le focus :

```
const inputIdHal = document.getElementById('inputIdHal');
inputIdHal.addEventListener('change', getDocs);
```

La première étape consiste donc à lancer la requête depuis cette fonction :

```
function getDocs(e) {
    request();
}
```

Et maintenant vous pouvez construire la fonction `request()`, chargée de traiter la requête `GET` et d'en extraire une réponse au format JSON. Attention, cette fonction, pour utiliser les facilités inhérentes aux *Promises*, devra être définie comme asynchrone :

```
async function request() {
    config.query = `?q=authIdHal_s:${config.idHal}&start=${config.start}`;
    const stream = await fetch(config.baseUrl + config.query);
    if (stream.ok) {
        return stream.json();
    }
}
```

Étape 2 : définir le nombre de documents

La fonction `setNbDocs()`, déjà écrite, modifie la propriété `nbDocs` de l'objet de configuration et lance une exception au cas où le nombre renvoyé par l'API HAL est égal à 0. Il ne vous reste plus qu'à la lancer dans la chaîne de traitement :

```
function getDocs(e) {
  request()
  .then( setNbDocs );
```

Étape 3 : établir la pagination

Même principe que plus haut pour la pagination :

```
function getDocs(e) {
  request()
```

```
  .then( setNbDocs )
  .then( setPagination );
```

Étape 4 : afficher les documents

La fonction `printDocs()` se charge de construire une liste à puces HTML et de placer à l'intérieur de chaque item la référence d'un document. Il est nécessaire à cette étape de savoir comment récupérer ces informations depuis le document JSON fourni par l'API HAL. La documentation officielle ou un `console.log()` vous apprennent que la liste des documents est disponible sous `response.docs` et que, pour chacun, la référence est enregistrée avec la propriété `label_s`. En résumé, pour atteindre la première référence, vous devriez interroger : `response.docs[0].label_s`.

Commencez par récupérer la liste des documents :

```
const printDocs = (data) => {
    const docs = data.response.docs;
}
```

Ensuite, construisez l'élément HTML `ul` qui servira de support aux items de la liste :

```
const printDocs = (data) => {
    const docs = data.response.docs;
    const ul = document.createElement('ul');
}
```

À présent, pour chaque document :

- construisez un élément HTML `li` ;
- ajoutez la référence au contenu HTML ;
- insérez finalement l'item à la liste.

```
const printDocs = (data) => {
    const docs = data.response.docs;
    const ul = document.createElement('ul');
    for (const doc of docs) {
        const li = document.createElement('li');
        li.innerHTML = doc.label_s;
        ul.appendChild(li);
    }
}
```

Enfin, rajoutez la liste complète au bloc `documents`, sans oublier de l'avoir préalablement vidée de tout contenu :

```
const printDocs = (data) => {
    const docs = data.response.docs;
    const ul = document.createElement('ul');
    for (const doc of docs) {
        const li = document.createElement('li');
        li.innerHTML = doc.label_s;
        ul.appendChild(li);
```

```
    }
    documents.innerHTML = '';
    documents.appendChild(ul);
}
```

Il ne vous reste plus qu'à rajouter la fonction `printDocs()` à la chaîne de traitement :

```
function getDocs(e) {
    request()
    .then( setNbDocs )
    .then( setPagination )
    .then( printDocs );
```

Étape 5 : gérer les erreurs

La méthode `catch()` de la chaîne de traitement intègre une fioriture, la remise à zéro de l'application avant l'apparition du message :

```
function getDocs(e) {
    request()
    .then( setNbDocs )
    .then( setPagination )
    .then( printDocs );
    .catch(error => {
        resetSearch(e);
        console.error(error.message);
    });
```

Un point final ?

Votre application n'est pas tout à fait terminée. Si la pagination s'affiche bien et que le mécanisme de clic est actif sur chaque numéro de page, la liste des publications ne change pas. Rien de plus normal : aucune ligne de code ne modifie jusqu'à présent la propriété `config.start`.

Demandez à la fonction `toPage()` de le faire pour vous :

```
config.start = (e.target.innerText - 1) * 30;
```

L'application, loin d'être parfaite, est tout de même bien aboutie. Parmi les améliorations envisageables, vous pourriez par exemple griser la page consultée, élaborer un mécanisme de "page précédente", "page suivante", permettre à l'utilisateur de modifier le nombre de résultats visibles…

Vous trouverez le code final des documents HTML et JavaScript dans le dossier *fin* de ce second TD.