Découverte de l’API HAL

Dans ce premier TD, vous vous familiariserez avec l’API de recherche de HAL. Cette application permet d’effectuer une recherche dans la base de données de HAL à partir d’une requête HTTP de type `GET`. Au fil des étapes, vous serez amené·es à concevoir un script qui récupère une entrée utilisateur dans un formulaire HTML (un IdHAL) et qui le soumet à l’API HAL via l’interface *Fetch* pour finalement analyser la réponse au format JSON et en extraire le nombre de documents de l’auteur ou de l’autrice mentionné·e. Le tout, sans recharger la page. **URL de l’API de recherche de HAL :** https://api.archives-ouvertes.fr/search/ **Documentation :** https://api.archives-ouvertes.fr/docs/search/

HAL en bref

HAL est un acronyme qui signifie "Hyper Articles en Ligne". Il s'agit d'une archive ouverte maintenue par le CCSD (Centre pour la communication scientifique directe) où l'on peut déposer des documents scientifiques comme des références bibliographiques. Le CCSD distribue plusieurs APIs pour interagir avec ses données, que ce soit avec son outil de recherche ou avec ses référentiels (auteurs, structures, projets…).

Définir les objectifs

Identifier la donnée à récupérer

Rappelons l’objectif principal : recueillir le nombre de références bibliographiques sur l’API de recherche HAL pour un IdHAL donné. Selon la documentation officielle, une recherche manuelle sur un navigateur Web repose sur la structure suivante :

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

En réponse, l’API vous renvoie un document au format JSON sous la forme :

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

Remarquez tout de suite que la valeur à récupérer est assignée à la propriété `response.numFound`.

Le code de départ

Dans le dossier *debut*, vous trouverez une page nommée *index.html*. Il s’agit d’un formulaire très simple avec un champ de saisie pour l’IdHAL. Notez la ligne en dessous, censée accueillir le nombre de publications (références et documents confondus). Parmi les choses remarquables dans le code, notons :

- la balise `<script>` qui, contrairement à l’usage, figure dans l’en-tête de la page ;
- l’identifiant du champ de saisie (`IdHAL`) ;

- l’identifiant du champ `output` censé afficher la donnée (`nbPublis`).

Conception du script

Trois grandes étapes se dessinent pour votre script :

1. récupérer l’IdHAL fourni par l’utilisateur ;
2. interroger l’API de recherche HAL ;
    - analyser la réponse ;
    - isoler la donnée ;
3. afficher la donnée.

Ces étapes vous amèneront à écouter des événements, à manipuler le DOM et à mobiliser vos compétences avec l’API *Fetch*.

Étape 1 : récupérer la saisie de l’utilisateur

La méthode `document.getElementById()` vous permet de récupérer l’élément HTML `input` qui recueille la saisie de l’utilisateur :

```
const idHal = document.getElementById('IdHAL');
```

Maintenant que l’élément est disponible, vous pouvez placer un écouteur dessus. Reste à choisir l’événement approprié parmi les possibilités et à nommer la fonction *callback* à lancer lors de sa réalisation. L’événement `change` par exemple se déclenche lors d’un changement de valeur dans l’élément ciblé et, afin d’éviter le déclenchement à chaque nouveau caractère saisi, à la perte du focus.

```
idHal.addEventListener('change', getNbPublis);
```

Écrivez à présent la fonction `getNbPublis()` :

```
function getNbPublis(e) {
    const IdHAL = e.target.value;
}
```

Si vous exécutez votre script, un écueil se présente à vous. Une exception est levée : `Uncaught TypeError: idHal is null` .

Lancer le script au chargement de la page

La raison de la survenue de ce problème est simple : la balise `<script>` étant placée dans l’en-tête du document HTML, le code JS est interprété **avant** que les éléments du corps soient analysés par l’API DOM. La solution la plus simple reste de déplacer la balise `<script>` en bas du document HTML, juste avant la balise fermante `</body>` , comme le veut l’usage. Une solution plus conforme à la norme du W3C consiste à retarder l’exécution du code JS. Placez un écouteur sur la fenêtre du navigateur afin de déclencher une fonction `run()` lorsque le document HTML a fini d’être chargé :

```
window.addEventListener('load', run);
```

Et définissez maintenant la fonction `run()` en replaçant le code qui écoute le champ de saisie à l’intérieur :

```
function run() {
    const idHal = document.getElementById('IdHAL');
    idHal.addEventListener('change', getNbPublis);
}
```

Étape 2 : interroger l'API de recherche HAL

Grâce à l'IdHAL récupéré à l'étape précédente, il vous est possible de formater l'URL de la requête au sein de la fonction `getNbPublis()` :

```
const baseUrl = 'https://api.archives-ouvertes.fr/search/';
const query = `?q=authIdHal_s:${IdHAL}`;
```

Cette URL est ensuite à fournir à l'API *Fetch* pour en sortir une réponse au format JSON :

```
fetch(baseUrl + query).then(response => {
    if (response.ok) {
        return response.json();
    }
});
```

Un objet JSON renvoyé, il est désormais permis de chaîner une nouvelle méthode `then()` afin d'isoler la donnée convoitée (`response.numFound`) :

```
fetch(baseUrl + query).then(response => {
    if (response.ok) {
        return response.json();
    }
})
.then(data => {
    const numFound = data.response.numFound;
});
```

Étape 3 : afficher la donnée

Pour afficher la donnée, il suffit de :

1. récupérer l'élément HTML concerné ;
2. modifier son contenu.

Au sein de la fonction `run()`, écrivez le code suivant :

```
const nbPublis = document.getElementById('nbPublis');
```

Puis fournissez à `nbPublis` la valeur de `numFound` :

```
.then(data => {
    const numFound = data.response.numFound;
    nbPublis.innerHTML = numFound;
})
```

Quelques améliorations

Une aide contextuelle

Que se passe-t-il lorsque vous saisissez dans le formulaire un IdHAL qui n'existe pas ? Eh bien, le nombre de publications reste à zéro. Pourquoi ne pas indiquer à l'utilisateur qu'il a peut-être commis une erreur lors de sa saisie ? Ajoutez un bloc d'aide à la suite de la dernière colonne dans la page HTML et rendez-la invisible par défaut grâce à la classe utilitaire de *Bootstrap* `invisible` :

```
<div class="col-auto">
    <span id="nbPublisHelp" class="form-text invisible">
        Vérifiez que l'IdHAL soumis est bien orthographié.
    </span>
</div>
```

À présent, il vous reste à tester la valeur de la variable `numFound` et, si elle est égale à `0`, retirer la classe `invisble` au bloc d'aide :

```
.then(data => {
    // enregistrer la donnée
    const numFound = data.response.numFound;
    // insérer le nb de publis dans le champ ad hoc
    nbPublis.innerHTML = numFound;
    // si le résultat est égal à 0
    if (numFound == 0) {
        // le champ d'aide est rendu visible
        nbPublisHelp.classList.remove('invisible');
    }
});
```

En prime, vous pourriez en profiter pour lever une exception et la traiter avec la méthode `catch()` :

```
.then(data => {
    // enregistrer la donnée
    const numFound = data.response.numFound;
    // insérer le nb de publis dans le champ ad hoc
    nbPublis.innerHTML = numFound;
    // si le résultat est égal à 0
    if (numFound == 0) {
        // le champ d'aide est rendu visible
        nbPublisHelp.classList.remove('invisible');
        // on lance une exception : peut-être un problème de saisie ?
        throw new Error(`Aucun résultat trouvé pour l'IdHAL ${IdHAL}.`);
    }
})
.catch(error => {
    // afficher l'erreur dans la console
    console.error(error.message);
});
```

Un brin d’ergonomie

Cette amélioration engendre un autre problème : une fois le bloc d’aide apparu, il ne disparaît plus. Et si, lorsque l’utilisateur sur le champ de saisie, vous en profitiez pour le faire disparaître à nouveau ? Cela vous demande tout d’abord d’écrire une fonction :

```
function eraseField(e) {
    // disparition du bloc d'aide
    nbPublisHelp.classList.add('invisible');
    // suppression de la saisie utilisateur
    e.target.value = '';
}
```

Puis, ensuite, de la déclencher au clic sur le champ de saisie :

```
idHal.addEventListener('click', eraseField);
```

Remarquez au passage que le code prévoit la suppression, toujours au clic, de la saisie utilisateur.

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