Présentation de l’API Fetch et de la méthode fetch()

L’API Fetch fournit une définition pour trois interfaces Request, Response et Headers et implémente également le mixin Body qu’on va pouvoir utiliser avec nos requêtes.

Les interfaces Request et Response représentent respectivement une requête et la réponse à une requête. L’interface Headers représente les en-têtes de requête et de réponse tandis que le mixin Body fournit un ensemble de méthodes nous permettant de gérer le corps de la requête et de la réponse.

L’API Fetch va également utiliser la méthode globale fetch() qui représente en quelques sortes le coeur de celle-ci. Cette méthode permet l’échange de données avec le serveur de manière asynchrone.

La méthode fetch() prend en unique argument obligatoire le chemin de la ressource qu’on souhaite récupérer. On va également pouvoir lui passer en argument facultatif un liste d’options sous forme d’objet littéral pour préciser la méthode d’envoi, les en-têtes, etc.

La méthode fetch() renvoie une promesse (un objet de type Promise) qui va se résoudre avec un objet Response. Notez que la promesse va être résolue dès que le serveur renvoie les en-têtes HTTP, c’est-à-dire avant même qu’on ait le corps de la réponse.

La promesse sera rompue si la requête HTTP n’a pas pu être effectuée. En revanche, l’envoi d’erreurs HTTP par le serveur comme un statut code 404 ou 500 vont être considérées comme normales et ne pas empêcher la promesse d’être tenue.

On va donc devoir vérifier le statut HTTP de la réponse. Pour cela, on va pouvoir utiliser les propriétés ok et status de l’objet Response renvoyé.

La propriété ok contient un booléen : true si le statut code HTTP de la réponse est compris entre 200 et 299, false sinon.

La propriété status va renvoyer le statut code HTTP de la réponse (la valeur numérique liée à ce statut comme 200, 301, 404 ou 500).

Pour récupérer le corps de la réponse, nous allons pouvoir utiliser les méthodes de l’interface Response en fonction du format qui nous intéresse :

* La méthode text() retourne la réponse sous forme de chaine de caractères ;
* La méthode json() retourne la réponse en tant qu’objet JSON ;
* La méthode formData() retourne la réponse en tant qu’objet FormData ;
* La méthode arrayBuffer() retourne la réponse en tant qu’objet ArrayBuffer ;
* La méthode blob() retourne la réponse en tant qu’objet Blob ;

fetch("https://www.une-url.com")

.then(response => response.json())

.then(response => alert(JSON.stringify(response)))

.catch(error => alert("Erreur : " + error));

Expliquons ce code ensemble. Tout d’abord, la méthode fetch() a besoin d’un argument obligatoire, qui correspond à l’URL des ressources à récupérer. On utilise ici une/url (remplacez bien évidemment par une vraie URL en pratique).

fetch() retourne ensuite une promesse contenant la réponse (si tout se passe bien). On ne peut pas exploiter la réponse renvoyée dans cette promesse en l’état : il faut indiquer le format de réponse souhaité. Ici, on choisit JSON avec response.json().

response.json() renvoie également une promesse contenant la réponse à votre demande en JSON. On utilise JSON.stringify() pour transformer notre objet JSON en une chaine JSON et on affiche cette chaine.

Finalement, on traite les erreurs avec le bloccatch et on affiche l’erreur rencontrée si on en rencontre effectivement une.

Passer des options à fetch()

Comme on l’a dit plus tôt, la méthode fetch() accepte un deuxième argument. Cet argument est un objet qui va nous permettre de définir les options de notre requête. On va pouvoir définir les options suivantes :

* method : méthode utilisée par la requête. Les valeurs possibles sont GET (défaut), POST, etc.) ;
* headers : les en-têtes qu’on souhaite ajouter à notre requête ;
* body : un corps qu’on souhaite ajouter à notre requête ;
* referrer : un référant. Les valeurs possibles sont "about:client" (valeur par défaut), "" pour une absence de référant, ou une URL ;
* referrerPolicy : spécifie la valeur de l’en-tête HTTP du référent. Les valeurs possibles sont no-referrer-when-downgrade (défaut), no-referrer, origin, origin-when-cross-origin et unsafe-url ;
* mode : spécifie le mode qu’on souhaite utiliser pour la requête. Les valeurs possibles sont cors (défaut), no-cors et same-origin ;
* credentials : les informations d’identification qu’on souhaite utiliser pour la demande. Les valeurs possibles sont same-origin (défaut), omit et include ;
* cache : le mode de cache qu’on souhaite utiliser pour la requête. Les valeurs possibles sont default (défaut), no-store, reload, no-cache, force-cache et only-if-cached ;
* redirect : le mode de redirection à utiliser. Valeurs possibles : follow (défaut), manual, error ;
* integrity : contient la valeur d’intégrité de la sous-ressource de la demande. Valeurs possibles : "" (défaut) ou un hash ;
* keepalive : permet à une requête de survivre à la page. Valeurs possibles : false (défaut) et true ;
* signal : une instance d‘un objet AbortSignal qui nous permet de communiquer avec une requête fetch() et de l’abandonner.

let promise = fetch(url, {

method: "GET", //ou POST, PUT, DELETE, etc.

headers: {

"Content-Type": "text/plain;charset=UTF-8" //pour un corps de type chaine

},

body: undefined, //ou string, FormData, Blob, BufferSource, ou URLSearchParams

referrer: "about:client", //ou "" (pas de réferanr) ou une url de l'origine

referrerPolicy: "no-referrer-when-downgrade", //ou no-referrer, origin, same-origin...

mode: "cors", //ou same-origin, no-cors

credentials: "same-origin", //ou omit, include

cache: "default", //ou no-store, reload, no-cache, force-cache, ou only-if-cached

redirect: "follow", //ou manual ou error

integrity: "", //ou un hash comme "sha256-abcdef1234567890"

keepalive: false, //ou true pour que la requête survive à la page

signal: undefined //ou AbortController pour annuler la requête

});

### Anatomie

Dans l'exemple précédente ☝️, nous avons:

* fetch(...) déclarant que fetch doit aller chercher le fichier à l'adresse fournie entre parenthèses et nous [retourner une promesse](https://smnarnold.com/cours/javascript/promesses).
* .then() ou en français "ensuite" attend que le serveur tienne sa promesse et nous réponde.
* data nom donné à l'objet retourné par fetch (ce nom est arbitraire).
* .json() indique que nous souhaitons appeler la méthode .json() de l'objet response, afin de convertir les données en JSON.
* .then() attend que la promesse de conversion des données soit respectée.
* actor correspond au nom donné au JSON reçu. (Encore une fois, ce nom est arbitraire).