Traitez la réponse du serveur

5–6 minutes

Le navigateur envoie une requête à l'API HTTP, mais nous n'avons pas encore appris à récupérer les données de la réponse. Pour cela, arrêtons-nous un instant sur la **programmation asynchrone** pour mieux appréhender la logique de cette opération.

Appréhendez la notion de programmation asynchrone

Lorsque le navigateur exécute la fonction fetch, il réalise une **opération asynchrone**. Cela veut dire que nous obtiendrons le résultat de cette opération plus tard. Par conséquent, le navigateur ne s'arrête pas de fonctionner le temps d'obtenir la réponse. Il garde en mémoire le fait qu'il attend la réponse à la requête tout en continuant à :

- exécuter le JavaScript ;
- traiter les autres événements de la page ;
- mettre à jour l'affichage ;
- et ainsi de suite…

Dans l'extrait de code suivant, l'instruction console.log est exécutée immédiatement après l'appel à la fonction fetch. Nous n'avons pas encore obtenu la réponse à notre requête, mais le navigateur continue l'exécution du JavaScript :

```
fetch("http://monsite.fr/ma-ressource");

console.log("Le script continue sans attendre la réponse");
```

Le langage JavaScript propose un mot clé pour résoudre ce problème : **await**. Il fera en sorte d'attendre que l'opération asynchrone se termine pour reprendre l'exécution du code.

Le mot clé await se place à gauche de l'expression asynchrone :

```
await fetch("http://monsite.fr/ma-ressource");

console.log("Le script continuera après avoir reçu la réponse");
```

Ainsi, l'instruction console.log ne sera exécutée qu'après avoir reçu la réponse de l'API.

Lorsque **await** est utilisé dans une fonction, il est doublé du mot clé **async** devant la définition de la fonction. C'est la raison pour laquelle nous avions le mot clé **async** devant notre fonction anonyme au chapitre précédent.

.then : l'autre syntaxe asynchrone

Jusqu'à présent, et tout au long de ce cours, nous avons utilisé la syntaxe await. Mais elle n'est apparue qu'en 2017, et s'appuie en réalité sur les promesses (Promise). Pour en apprendre plus sur les promesses, consultez ce guide sur Mozilla Developer Network.

Par exemple, pour les plus curieux, l'extrait de code précédent s'écrirait de la façon suivante si on utilisait les Promises :

```
fetch("http://monsite.fr/ma-ressource").then(function () {
      console.log("Le script continuera après avoir reçu la réponse");
});
```

Traitez la réponse de votre requête

Stockez la réponse du serveur avant de la traiter

Pour traiter la réponse du serveur, commençons par stocker le résultat de la fonction dans une constante. Modifiez la ligne dans l'event listener du clic sur le bouton "Afficher les avis" :

```
const id = event.target.dataset.id;

const reponse = await fetch("http://localhost:8081
/pieces/" + id + "/avis");
```

Maintenant que nous pouvons attendre le retour de la réponse, il nous faut traiter cette réponse pour en extraire les informations demandées. Dans l'extrait de code ci-dessus, nous récupérons les avis pour une pièce automobile, au format JSON.

Pour y parvenir, nous rajoutons un appel à la fonction JSON sur l'objet reponse. Il faut également utiliser le mot clé await, car cette opération est aussi asynchrone :

```
const id = event.target.dataset.id;

const reponse = await fetch("http://localhost:8081
/pieces/" + id + "/avis");

const avis = await reponse.json();
```

La constante avis contient désormais une liste d'objets de tous les avis pour une pièce en particulier. Il ne nous reste plus qu'à générer des éléments grâce aux fonctions createElement et appendChild :

```
const pieceElement = event.target.parentElement;

const avisElement = document.createElement("p");

for (let i = 0; i < avis.length; i++) {

   avisElement.innerHTML += `${avis[i].utilisateur}:
${avis[i].commentaire} <br>`;

}
```

La fiche produit Ampoule LED avec les avis clients affichés en bas

Et voilà : les avis des clients s'affichent en bas du produit ! 🤩

Récapitulons en vidéo

Vous pouvez revoir les différentes étapes de cette démonstration dans la vidéo ci-dessous :

À vous de jouer !

Depuis la branche P3C2-Exercice, dans le fichier pieces.js, remplacez la ligne qui récupère les pièces depuis le fichier JSON, de manière à les récupérer depuis l'API à l'adresse http://localhost:8081/pieces.

Corrigé

Vous pouvez vérifier votre travail en consultant la branche P3C2-Solution et en regardant la vidéo ci-dessous :

En résumé

- Pour traiter les réponses de vos requêtes :
- Utilisez la syntaxe async/await pour mettre en pause le programme en attendant la réponse.
- Désérialisez vos données pour décoder le contenu de la réponse.
- Avec cette réponse, vous pouvez mettre à jour la page web.

Maintenant que vous avez traité les réponses du serveur, rendez-vous dans le prochain chapitre pour sauvegarder les données grâce à une API HTTP !