

Les fonctions asynchrones et les requêtes HTTP en JavaScript

L'interface *fetch*

- La fonction **fetch** permet d'effectuer des requêtes HTTP de manière conviviale
- Anciennement, il fallait utiliser l'interface **XMLHttpRequest**, plus complexe
- L'interface **fetch** est supportée par tous les navigateurs récents

Chrome	Edge [*]	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS [*]	Samsung Internet	Opera Mini [*]	Opera Mobile [*]	UC Browser for Android	Android Browser [*]	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-39			2-33	10-26												
² 40			^{1 4} 34-38	² 27												
^{2 3} 41	12-13	3.1-10	⁴ 39	^{2 3} 28			3.2-10.2									
42-135	14-135	10.1-18.4	40-137	29-116	6-10		10.3-18.4	4-26		12-12.1		2.1-4.4.4				2.5
136	136	18.5	138	117	11	136	18.5	27	all	80	15.5	136	137	14.9	13.52	3.1
137-139		TP	139-141													

Source: <https://caniuse.com/fetch>

Les fonctions asynchrones en JavaScript

- ***fetch* est une fonction asynchrone**

- C'est à dire qu'elle ne retourne pas un résultat immédiatement après son exécution (celui-ci viendra plus tard!)
- L'exemple ci-dessous ne fonctionne pas:

```
const student = fetch('https://example.org/api/students/1842421');  
console.log(student.firstName, student.lastName);
```

- Après l'exécution de la première ligne de code, le programme passe immédiatement à la ligne suivante, sans attendre le résultat de la requête
- La variable « student » ne contient donc rien!

Les promesses

- La fonction *fetch* retourne une **promesse**
- Les promesses permettent de gérer les fonctions asynchrones de manière élégante
- Il existe deux façons de gérer les fonctions qui retournent des promesses:
 - La méthode « **then** »
 - Les mots clés **async/await**

La méthode « .then »

```
fetch('https://example.org/api/students/1842421')
```

```
.then(response => {
```

```
    return response.json(); // Retourne aussi une  
    promesse
```

```
});
```

```
.then(student => {
```

```
    console.log(student.firstName, student.lastName);
```

```
});
```

Les mots-clés « async/await »

```
async function fetchStudent(id) => {  
    const response = await fetch(`https://example.org/api/students/${id}`);  
    const student = await response.json();  
    console.log(student.firstName, student.lastName);  
}  
fetchStudent(1842421);
```

Personnaliser la requête

- Par défaut, *fetch* utilise la méthode GET et n'inclut pas d'en-têtes et de corps dans la requête
- On peut changer ce comportement en passant un objet comme deuxième paramètre:

```
const requestBody = {...};  
const response = await fetch('URL', {  
  method: 'post'  
  body: JSON.stringify(requestBody),  
  headers: new Headers({ 'content-type': 'application/json' }),  
});
```

Les callbacks

- **Avant l'apparition des `promesses` et des mots-clés `async/await`, les fonctions asynchrones utilisaient des `callbacks`**
 - Un callback est une fonction qu'on passe en paramètre à une autre fonction
 - Dans le cas d'une fonction `asynchrone`, celle-ci appelle le callback une fois qu'elle a un résultat à lui transmettre

Les callbacks – Exemple

```
maFonctionAsynchrone((resultat) => {  
    console.log('Résultat: ', resultat);  
});
```

Callback Hell

- **Le problème avec les callbacks, c'est que si on veut exécuter plusieurs fonctions asynchrones une après l'autre, on se ramasse avec un « escalier » de callbacks imbriqués**
- **C'est ce qu'on appelle le *callback hell* (l'enfer des callbacks)**

Callback Hell - Exemple

```
maFonctionAsynchrone((resultat) => {  
  maDeuxiemeFonctionAsynchrone(resultat, (resultat2) => {  
    maTroisiemeFonctionAsynchrone(resultat2, (resultat3) => {  
      maQuatriemeFonctionAsynchrone(resultat3, (resultat4) => {  
        console.log('Résultat final:', resultat4);  
      }  
    }  
  }  
});
```

Fin de la présentation

Des questions?



Photo par Matt Walsh sur Unsplash