


Qu'est-ce que `useEffect` ?

Imagine que tu construis un site web, et tu veux que quelque chose se passe automatiquement dès que la page se charge. Par exemple, tu veux afficher un message de bienvenue ou récupérer des informations sur Internet (comme les dernières nouvelles) et les afficher.

Dans React, on utilise `useEffect` pour faire ce genre de choses. C'est une manière de dire : "Quand ce composant s'affiche pour la première fois, je veux que telle chose se passe."

```
useEffect(() => {  
  fetch('https://fakestoreapi.com/products')  
    .then(response => response.json())  
    .then(data => setProducts(data))  
    .catch(error => console.error('Erreur de récupération des produits:', error));  
}, []);
```

- `useEffect` : Hook qui exécute un effet de bord après que le composant est monté. Ici, il est utilisé pour récupérer des produits depuis une API.
- `fetch('https://fakestoreapi.com/products')` : Appel à l'API `fakestoreapi.com` pour récupérer une liste de produits.
- `.then(response => response.json())` : Conversion de la réponse en format JSON.
- `.then(data => setProducts(data))` : Mise à jour de l'état `products` avec les données récupérées.
- `.catch(error => console.error('Erreur de récupération des produits:', error))` : Gestion des erreurs si la récupération échoue. 
- `[]` : Le tableau vide en deuxième argument signifie que cet effet s'exécute une seule fois, après le premier rendu du composant.

Exemple simple pour mieux comprendre

Imaginons que tu as un composant qui dit "Bienvenue" quand il s'affiche pour la première fois.

```
jsx Copier le code

import React, { useEffect } from 'react';

function Welcome() {
  useEffect(() => {
    // Ce code s'exécute une seule fois, quand le composant est affiché pour la première fois
    console.log('Bienvenue sur la page !');
  }, []); // Le tableau vide signifie que ça se passe une seule fois

  return <h1>Bonjour !</h1>;
}
```

Décomposition de ce qui se passe ici :

1. **Affichage initial** : Quand tu ouvres la page pour la première fois, React affiche "Bonjour !" à l'écran.
2. **Effet `useEffect`** :
 - Après que "Bonjour !" est affiché, React regarde si tu as demandé à faire quelque chose d'autre.
 - Il voit le `useEffect` avec un tableau vide `[]`, ce qui lui dit d'exécuter le code à l'intérieur (ici `console.log('Bienvenue sur la page !');`).
 - Ce code affiche "Bienvenue sur la page !" dans la console une seule fois.
3. **Pourquoi le tableau `[]` ?**
 - Si le tableau `[]` est vide, ça veut dire : "Fais ça **une seule fois** après le premier affichage."
 - Si on avait mis quelque chose dans ce tableau, React vérifierait à chaque rendu si quelque chose a changé, et il réexécuterait le code si c'est le cas.

- ``fetch`` : C'est une fonction native de JavaScript qui permet de faire des requêtes HTTP pour récupérer ou envoyer des données sur un serveur.
- ``https://fakestoreapi.com/products`` : C'est l'URL de l'API que vous interrogez. Cette URL renvoie des données au format JSON qui contiennent une liste de produits.

Résumé : Cette ligne envoie une requête HTTP GET à l'API pour récupérer la liste des produits.

Ligne 3 : ``.then(response => response.json())``

- ``.then`` : ``fetch`` retourne une *Promise*, qui est un objet représentant la réponse future de la requête. ``.then`` est une méthode qui permet de dire "quand la requête est réussie, fais ceci...".
- ``.response`` : C'est l'objet de réponse renvoyé par le serveur après que la requête ``fetch`` a été exécutée avec succès. Cet objet contient des informations sur la réponse, telles que le statut et le corps de la réponse.
- ``.response.json()`` : Cette méthode extrait le corps de la réponse au format JSON et le convertit en un objet JavaScript. Cette méthode retourne aussi une *Promise*, c'est pourquoi vous pouvez enchaîner un autre ``.then``.



enchaîner un autre ``.then``.


Résumé : Cette ligne traite la réponse du serveur pour la convertir en un format que vous pouvez utiliser en JavaScript (un objet ou un tableau).

Ligne 4 : ``.then(data => setProducts(data))``

- ``.data`` : C'est l'objet ou le tableau de données qui a été extrait du JSON dans l'étape précédente.
- ``.setProducts(data)`` : ``.setProducts`` est une fonction que vous avez obtenue en utilisant ``.useState`` dans le composant. Elle est utilisée pour mettre à jour l'état ``.products`` avec les données récupérées (``.data``).

Résumé : Cette ligne met à jour l'état ``.products`` avec les données obtenues de l'API.

Ligne 5 : ``.catch(error => console.error('Erreur de récupération des produits:', error))``

- ``.catch`` : C'est une méthode qui permet  capturer et de traiter les erreurs qui pourraient survenir pendant l'exécution de la *Promise*. Par exemple, si la requête échoue, ``.catch`` sera

survenir pendant l'exécution de la *Promise*. Par exemple, si la requête échoue, `.catch` sera exécuté.

``error`` : Ceci est l'objet d'erreur qui est passé à la fonction si quelque chose ne va pas (par exemple, si l'API n'est pas disponible ou si la connexion échoue).

``console.error('Erreur de récupération des produits:', error)`` : Cette ligne affiche un message d'erreur dans la console du navigateur. Le message ``Erreur de récupération des produits:`` est un texte explicatif, suivi par l'objet ``error``, qui contient des informations détaillées sur l'erreur.

Résumé : Cette ligne gère les erreurs en les affichant dans la console pour vous aider à diagnostiquer ce qui a mal tourné.

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

- `.then`` : Une fois que les informations sont récupérées, on fait quelque chose avec ces informations.
- ``response.json()`` : On transforme ces informations en un format que l'ordinateur peut comprendre (un objet JavaScript).

4. `.then(data => setProducts(data))``

- ``data`` : Ce sont les informations sur les produits que vous avez récupérées.
- ``setProducts(data)`` : On place ces informations dans un endroit appelé ``products`` pour que votre application puisse les utiliser.


5. `.catch(error => console.error('Erreur de récupération des produits:', error))``

- `.catch`` : Si quelque chose tourne mal (par exemple, si le site n'est pas disponible), cette partie du code s'exécute.
- ``console.error(...)`` : Affiche un message d'erreur dans la console pour vous dire ce qui ne va pas.



Fonction pour ajouter un produit au panier

javascript

 Copier le code

```
const addToCart = (product) => {  
  setCart([...cart, product]);  
};
```

Les éléments clés :

1. ``const addToCart = (product) => { ... };``

- Qu'est-ce que c'est ?

C'est une fonction fléchée (arrow function) en JavaScript. Cette fonction prend un paramètre appelé ``product``. Ce paramètre représente l'article que tu veux ajouter au panier.

- Pourquoi utiliser une fonction ?

Dans une application React, tu peux définir des fonctions comme celle-ci pour faire des actions spécifiques. Ici, la fonction ``addToCart`` est utilisée pour ajouter un produit au panier.

2. ``setCart([...cart, product]);``

- Qu'est-ce que ``setCart`` ?

``setCart`` est une fonction qui est utilisée pour mettre à jour l'état du panier (``cart``). Dans React, on utilise souvent des hooks comme ``useState`` pour gérer l'état d'un composant

- Qu'est-ce que ``cart`` ?

``cart`` est une variable qui représente l'état actuel du panier. C'est un tableau qui contient tous les produits qui sont dans le panier.

- Que signifie ``[...cart, product]`` ?

- Le ``...`` est appelé l'**opérateur de décomposition** (spread operator). Il est utilisé ici pour copier tous les éléments actuels du panier (``cart``) dans un nouveau tableau.

- ``[...cart, product]`` crée un nouveau tableau qui contient tous les éléments déjà présents dans ``cart``, plus le nouveau produit (``product``) à la fin. C'est une façon de dire "ajoute ce produit à la liste des produits déjà dans le panier".

- Pourquoi faire cela ?


En React, il est important de ne pas modifier directement l'état. Au lieu de ça, on crée une nouvelle version de l'état avec les changements, et on utilise ``setCart`` pour mettre à jour l'état avec cette nouvelle version.

Exemple plus détaillé

Imaginons que tu as un panier qui contient déjà deux produits, et tu veux ajouter un troisième.

1. État actuel du panier (`cart`) :


javascript

 Copier le code

```
const cart = [  
  { id: 1, name: "Produit A" },  
  { id: 2, name: "Produit B" }  
];
```

2. Nouveau produit que tu veux ajouter :


javascript

 Copier le code

```
const product = { id: 3, name: "Produit C" };
```

2. Nouveau produit que tu veux ajouter :

javascript


 Copier le code

```
const product = { id: 3, name: "Produit C" };
```

3. Que fait la fonction `addToCart(product)` ? :

- `setCart([...cart, product]);` crée un nouveau tableau :

javascript

 Copier le code

```
[  
  { id: 1, name: "Produit A" },  
  { id: 2, name: "Produit B" },  
  { id: 3, name: "Produit C" } // Nouveau produit ajouté  
]
```