1. `useEffect` en React :

`useEffect` est un hook dans React qui permet d'exécuter du code **après le rendu** du composant. Il est souvent utilisé pour effectuer des actions comme :

- Récupérer des données via une API,
- Ajouter des écouteurs d'événements,
- Synchroniser des données avec une source externe.

Exemple simple :

```
import React, { useEffect, useState } from 'react';

const App = () => {
  const [message, setMessage] = useState('Chargement...');

  useEffect(() => {
    setTimeout(() => {
      setMessage('Bienvenue sur mon site !');
    }, 2000); // Change le message après 2 secondes
  }, []); // Le tableau vide signifie que l'effet s'exécute une seule fois

  return <h1>{message}</h1>;
};

export default App;
```

Explication :

- `useEffect(() => {...}, []);` :
  - La fonction dans `useEffect` s'exécute **après** le premier rendu du composant.
  - Le tableau `[]` signifie que cet effet ne s'exécutera qu'une seule fois (comme `componentDidMount` en classe).
  - Vous pouvez ajouter des dépendances dans le tableau (par ex. `[variable]`) pour relancer l'effet si cette variable change.

---

2. `fetch` :

`fetch` est une méthode **native** de JavaScript pour effectuer des requêtes HTTP (GET, POST, etc.). Elle retourne une **promesse** (Promise) qui permet de gérer les données récupérées.

Exemple avec `fetch` :

```
import React, { useEffect, useState } from 'react';

const App = () => {
  const [data, setData] = useState([]);
```

```
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts') // Appel API
      .then(response => {
        if (!response.ok) {
          throw new Error('Erreur HTTP'); // Gestion manuelle des erreurs
        }
        return response.json(); // Convertit la réponse en JSON
      })
      .then(data => {
        setData(data); // Stocke les données
        setLoading(false); // Indique que le chargement est terminé
      })
      .catch(error => {
        setError(error.message); // Capture les erreurs
        setLoading(false);
      });
  }, []); // Appel unique

  if (loading) return <p>Chargement...</p>;
  if (error) return <p>Erreur : {error}</p>;

  return (
    <ul>
      {data.map(post => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
};

export default App;
```

---

3. `axios` :

`axios` est une **bibliothèque externe** pour effectuer des requêtes HTTP. Elle offre des fonctionnalités avancées par rapport à `fetch`, notamment :

- Une gestion automatique des erreurs,
- Une meilleure gestion des en-têtes et des paramètres,
- La possibilité d'annuler des requêtes.

Exemple avec `axios` :

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';

const App = () => {
```

```
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts') // Appel API avec
axios
      .then(response => {
        setData(response.data); // Récupère directement les données
        setLoading(false);
      })
      .catch(error => {
        setError(error.message); // Capture les erreurs automatiquement
        setLoading(false);
      });
  }, []); // Appel unique

  if (loading) return <p>Chargement...</p>;
  if (error) return <p>Erreur : {error}</p>;

  return (
    <ul>
      {data.map(post => (
        <li key={post.id}>{post.title}</li>
      ))}
    </ul>
  );
};

export default App;
```

---

4. Différences entre `fetch` et `axios` :

Critères	`fetch`	`axios`
Disponibilité	Natif dans les navigateurs modernes (pas besoin d'installation).	Nécessite une installation avec `npm install axios`.
Gestion des erreurs	Les erreurs HTTP (comme 404) doivent être vérifiées manuellement (`if (!response.ok)`).	Les erreurs HTTP sont automatiquement rejetées comme des exceptions.
Syntaxe	Peut être verbeuse, surtout pour des cas complexes.	Plus concise et lisible.
Données reçues	Nécessite un appel explicite pour convertir la réponse (`response.json()`).	Les données sont directement accessibles via `response.data`.
Fonctionnalités avancées	Nécessite du code supplémentaire pour gérer les en-têtes ou annuler des requêtes.	Intègre facilement des options avancées (en-têtes, timeout, annulation, etc.).

Critères	`fetch`	`axios`
Support des navigateurs	Fonctionne uniquement dans les navigateurs modernes (polyfill nécessaire pour anciens navigateurs).	Compatible avec tous les navigateurs grâce à XMLHttpRequest.

5. Résumé simple :

- `useEffect` : Sert à exécuter des actions après le rendu (souvent pour récupérer des données d'API).
- `fetch` : Solution native, idéale pour des besoins simples.
- `axios` : Solution plus puissante et conviviale, adaptée aux projets complexes.

Choix entre les deux :

- Si vous voulez une solution native et légère → `fetch`.
- Si vous avez besoin de fonctionnalités avancées et d'une syntaxe propre → `axios`.