Exercice 2 - Fetch de données

Objectif

Charger des données depuis une API et gérer les états de chargement et d'erreur.

Concepts abordés

- useEffect pour les appels API
- Gestion des états : loading, error, data
- async/await dans useEffect
- Cleanup pour annuler les requêtes

Énoncé

Créer un composant `UsersList` qui :

1. Charge une liste d'utilisateurs depuis JSONPlaceholder
2. Affiche un loader pendant le chargement
3. Affiche un message d'erreur si la requête échoue
4. Affiche la liste des utilisateurs sinon
5. Permet de rafraîchir les données

API à utiliser

```
GET https://jsonplaceholder.typicode.com/users
```

Points d'attention

Async/await dans useEffect

```
// ERREUR : useEffect ne peut pas être async directement
useEffect(async () => {  // NON !
  const data = await fetch(...);
}, []);

// CORRECT : créer une fonction async à l'intérieur
useEffect(() => {
  async function fetchData() {
    const data = await fetch(...);
  }
  fetchData();
}, []);
```

Pattern complet avec gestion d'état

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

Correction

```
// src/exercices/module-2/ex02-fetch/UsersList.jsx

import { useState, useEffect } from 'react';
import './UsersList.css';

function UsersList() {
  // ═══════════════════════════════════════════════════════════
  // STATE
  // 3 états distincts pour gérer tous les cas
  // ═══════════════════════════════════════════════════════════
  const [users, setUsers] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  // ═══════════════════════════════════════════════════════════
  // FONCTION DE FETCH
  // Extraite pour pouvoir être réutilisée (rafraîchissement)
  // ═══════════════════════════════════════════════════════════
  async function fetchUsers() {
    setIsLoading(true);
    setError(null);

    try {
      const response = await fetch(
        'https://jsonplaceholder.typicode.com/users'
      );

      // Vérifier si la requête a réussi
      if (!response.ok) {
        throw new Error(`Erreur HTTP: ${response.status}`);
      }

      const data = await response.json();
      setUsers(data);
    } catch (err) {
      // err peut être une erreur réseau ou notre erreur custom
      setError(err.message || 'Une erreur est survenue');
    } finally {
      // finally s'exécute dans tous les cas (succès ou erreur)
      setIsLoading(false);
    }
```

```
  }

  // ═══════════════════════════════════════════════════════════════
  // EFFECT : Charger les données au montage
  // ═══════════════════════════════════════════════════════════════
  useEffect(() => {
    fetchUsers();
  }, []);  // [] = exécuté une seule fois au montage

  // ═══════════════════════════════════════════════════════════════
  // RENDU CONDITIONNEL
  // ═══════════════════════════════════════════════════════════════

  // État de chargement
  if (isLoading) {
    return (
      <div className="users-container">
        <div className="loading">
          <div className="spinner"></div>
          <p>Chargement des utilisateurs...</p>
        </div>
      </div>
    );
  }

  // État d'erreur
  if (error) {
    return (
      <div className="users-container">
        <div className="error">
          <p>Erreur : {error}</p>
          <button onClick={fetchUsers} className="retry-btn">
            Réessayer
          </button>
        </div>
      </div>
    );
  }

  // État normal : affichage des données
  return (
    <div className="users-container">
      <div className="users-header">
        <h2>Liste des utilisateurs ({users.length})</h2>
        <button onClick={fetchUsers} className="refresh-btn">
          Rafraîchir
        </button>
      </div>

      <ul className="users-list">
        {users.map(user => (
          <li key={user.id} className="user-item">
            <div className="user-info">
              <h3>{user.name}</h3>
```

```
              <p className="user-email">{user.email}</p>
              <p className="user-company">{user.company.name}</p>
            </div>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default UsersList;
```

Version avec AbortController (annulation)

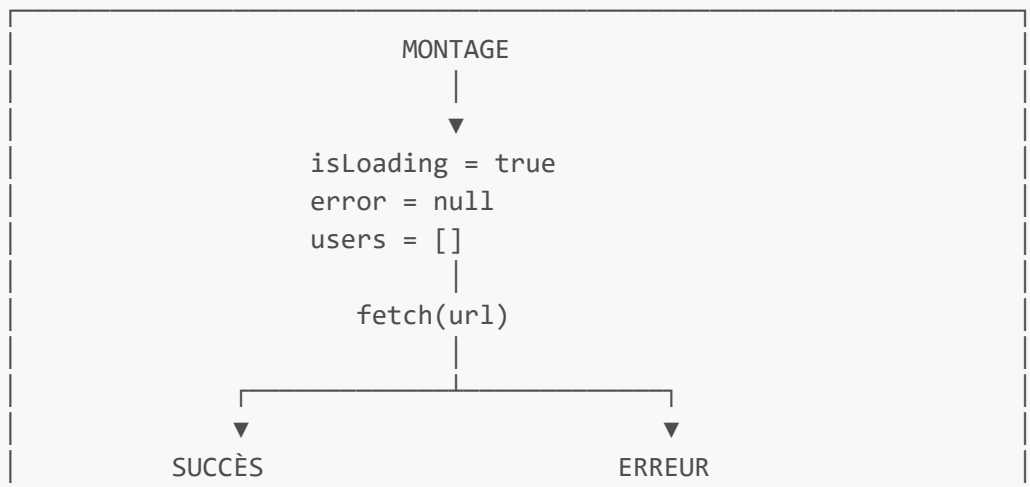
```
// Version avancée avec annulation des requêtes
useEffect(() => {
  // Créer un AbortController pour pouvoir annuler la requête
  const controller = new AbortController();

  async function fetchUsers() {
    setIsLoading(true);
    setError(null);

    try {
      const response = await fetch(
        'https://jsonplaceholder.typicode.com/users',
        { signal: controller.signal }  // Passer le signal
      );

      if (!response.ok) {
        throw new Error(`Erreur HTTP: ${response.status}`);
      }

      const data = await response.json();
      setUsers(data);
    } catch (err) {
      // Ne pas traiter l'erreur si c'est une annulation
      if (err.name === 'AbortError') {
        console.log('Requête annulée');
        return;
      }
      setError(err.message);
    } finally {
      setIsLoading(false);
    }
  }

  fetchUsers();

  // Cleanup : annuler la requête si le composant est démonté
  return () => {
```

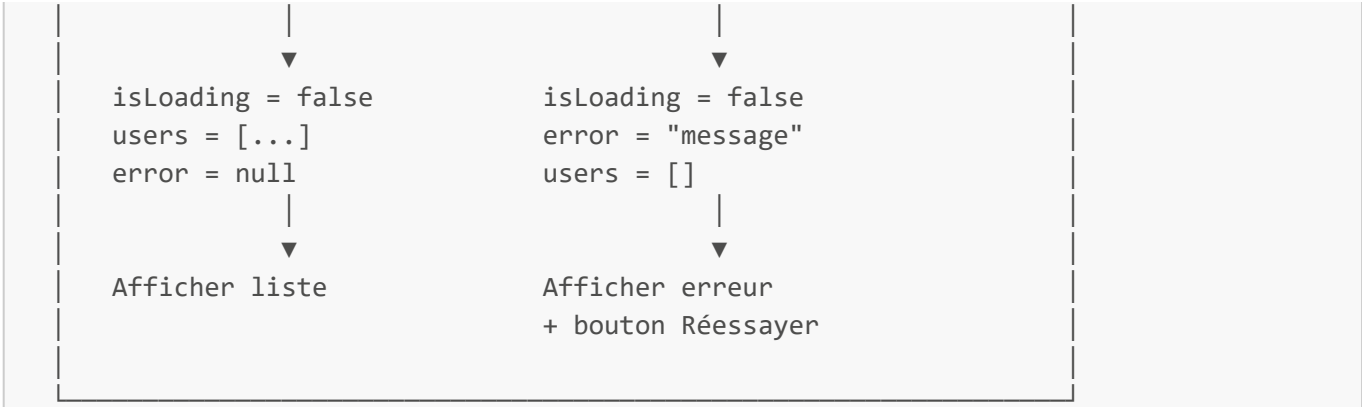
```
    controller.abort();
  };
}, []);
```

```css
/* src/exercices/module-2/ex02-fetch/UsersList.css */

.users-container {
  max-width: 600px;
  margin: 0 auto;
  padding: 1rem;
}

.loading, .error {
  text-align: center;
  padding: 3rem;
}

.spinner {
  width: 40px;
  height: 40px;
  border: 4px solid #e5e7eb;
  border-top-color: #2563eb;
  border-radius: 50%;
  margin: 0 auto 1rem;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  to { transform: rotate(360deg); }
}

.error {
  color: #dc2626;
}

.retry-btn, .refresh-btn {
  padding: 0.5rem 1rem;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.retry-btn {
  background: #dc2626;
  color: white;
}

.refresh-btn {
  background: #2563eb;
  color: white;
}
```

```css
.users-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 1rem;
}

.users-list {
  list-style: none;
  padding: 0;
}

.user-item {
  padding: 1rem;
  border: 1px solid #e5e7eb;
  border-radius: 8px;
  margin-bottom: 0.5rem;
}

.user-item h3 {
  margin: 0 0 0.25rem 0;
}

.user-email {
  color: #2563eb;
  margin: 0;
}

.user-company {
  color: #6b7280;
  font-size: 0.875rem;
  margin: 0;
}
```

Schéma mental : États du fetch





Pour aller plus loin

1. Ajouter une pagination
2. Ajouter un champ de recherche
3. Mettre en cache les résultats
4. Créer un hook personnalisé `useFetch`