TP React 7
Appels API avec `fetch`

Objectifs pédagogiques

À la fin de ce TP, vous serez capable de :
— comprendre le principe d'une API
— effectuer un appel HTTP avec `fetch`
— stocker des données distantes dans le state
— afficher des données chargées dynamiquement

Introduction

Les applications web modernes communiquent très souvent avec des serveurs distants afin de récupérer ou d'envoyer des données.

React utilise l'API `fetch` du navigateur pour effectuer ces requêtes.

> Les appels API sont généralement effectués dans un `useEffect`.

1 Structure d'un appel `fetch`

Un appel `fetch` retourne une promesse.

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(response => response.json())
  .then(data => console.log(data));
```

2 Premier appel API dans React

Dans `src/App.jsx` :

```
import { useState, useEffect } from "react";

function App() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/posts")
      .then(response => response.json())
      .then(data => setPosts(data));
  }, []);

  return (
    <div>
      <h1>Liste des articles</h1>
    </div>
  );
}

export default App;
```

Les données sont chargées au démarrage et stockées dans le state.

3 Afficher les données reçues

Ajoutez l'affichage des articles :

```
<ul>
  {posts.map((post) => (
    <li key={post.id}>
      <strong>{post.title}</strong>
    </li>
  ))}
</ul>
```

4 Gestion du chargement

Ajoutons un indicateur de chargement.

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

useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/posts")
    .then(response => response.json())
    .then(data => {
      setPosts(data);
      setLoading(false);
    });
}, []);
```

Affichage conditionnel :

```
{loading ? <p>Chargement...</p> : <ul>...</ul>}
```

5 Gestion des erreurs

Il est important de gérer les erreurs réseau.

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

useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/posts")
    .then(response => {
      if (!response.ok) {
        throw new Error("Erreur réseau");
      }
      return response.json();
    })
    .then(data => setPosts(data))
    .catch(err => setError(err.message));
}, []);
```

Affichage de l'erreur :

```
{error && <p>Erreur : {error}</p>}
```

Conclusion

Dans ce TP, vous avez appris à :
— appeler une API avec `fetch`
— gérer le chargement et les erreurs
— afficher des données distantes

Les appels API sont indispensables dans toute application React moderne.