Consumare API esterne in React + SharePoint

1. Fare fetch in React con useEffect

Il modo più comune è usare gli hook useState e useEffect per fare richieste API e salvare la risposta nello stato del componente.

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

function MyComponent() {
  const [data, setData] = useState(null);

  useEffect(() => {
    const fetchData = async () => {
      try {
        const response = await fetch('https://api.esterno.com/endpoint');
        const json = await response.json();
        setData(json);
      } catch (error) {
        console.error('Errore nel fetch:', error);
      }
    };
    fetchData();
  }, []);

  return data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Caricamento...';
}
```

2. Gestione di loading ed errori

È utile mostrare messaggi di caricamento ed errori. Si può creare un hook personalizzato useFetch per centralizzare la logica.

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

const useFetch = (url, options = {}) => {
  const [data, setData] = useState(null);
  const [isLoading, setIsLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    setIsLoading(true);
    fetch(url, options)
      .then(res => res.json())
      .then(json => {
        setData(json);
        setIsLoading(false);
      })
      .catch(err => {
        setError(err);
        setIsLoading(false);
```

```
      });
  }, [url, options]);

  return { data, isLoading, error };
};
```

3. Problemi di CORS e soluzioni

- Attivare CORS sul server esterno (consigliato) - Usare un backend proxy che fa da ponte - Proxy di sviluppo (create-react-app) - Soluzioni temporanee: estensioni browser o CORS-anywhere - In SPFx: attenzione alle policy SharePoint, meglio usare un backend proxy

```
// Esempio di proxy in package.json (CRA)
{
  "proxy": "http://localhost:5000"
}

// Esempio di fetch con opzioni no-cors (limitato)
fetch('https://api.esterno.com/endpoint', { mode: 'no-cors' })
```

4. Cancellare richieste e ottimizzare performance

- Usare AbortController per annullare fetch interrotti - Evitare doppie chiamate in Strict Mode (avviene solo in dev) - Controllare l’array delle dipendenze di useEffect per non ripetere fetch inutilmente

```
useEffect(() => {
  const controller = new AbortController();
  const fetchData = async () => {
    try {
      const res = await fetch(url, { signal: controller.signal });
      const data = await res.json();
      setData(data);
    } catch (err) {
      if (err.name !== 'AbortError') {
        setError(err);
      }
    }
  };
  fetchData();
  return () => controller.abort();
}, [url]);
```

5. Conclusione

Passi consigliati: 1. Usare fetch/axios in useEffect 2. Gestire stato di loading ed errori 3. Risolvere problemi di CORS lato server o proxy 4. Pulire le fetch con AbortController 5. Ottimizzare le dipendenze in useEffect