React: gestire il valore null

In React, come in JavaScript e in altri contesti in generale, il valore null può risultare problematico da gestire.

Se stiamo reperendo una singola risorsa da un API REST, molto spesso riceviamo come risposta un singolo oggetto JSON. Questo è un caso molto frequente di uso del valore `null` come valore iniziale nello state di un componente.

Infatti, `null` indica che un oggetto non ha un valore. Questo accade se ad esempio abbiamo una route parametrica e l'API REST restituisce un errore HTTP 404. Ovviamente il formato di risposta dipende dal design della singola API, ma non è raro che venga restituito questo valore. Gestendo correttamente i codici di stato HTTP possiamo risolvere questo problema mostrando una view che indica che la risorsa non è stata trovata.

Tuttavia, è bene ricordare che esistono API remote che non seguono alla lettera il paradigma REST per quanto riguarda i codici di stato HTTP. Alcune di loro, infatti, possono restituire sempre un codice di stato 200 e segnalare l'errore solo nel formato della risposta.

Immaginiamo quindi di avere un componente così strutturato:

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

export default function Test() {
    const [resource, setResource] = useState(null);

    useEffect(() => {
         axios.get('/api/resource').then(resp => {
             setResource(resp.data);
         });
    }, []);

    return (
        <h1>{resource.title}</h1>
    );
}
```

Lo scenario iniziale dell'esempio è il più semplice. Nel caso in cui l'API restituisca correttamente lo stato HTTP, possiamo gestire il fatto che `resource` resti sul valore `null` in questo modo:

```
export default function Test() {
    const [resource, setResource] = useState(null);
    const [isError, setIsError] = useState(false);

    useEffect(() => {
         axios.get('/api/resource').then(resp => {
             setResource(resp.data);
         }).catch(err => {
             setIsError(true);
         });
```

```
    }, []);

    return (
        {!isError ?
            <h1>{resource.title}</h1>
        : <p>Oops!</p>
        }
    );
}
```

Abbiamo impostato un flag booleano per segnalare la presenza di un errore nella richiesta HTTP tramite il blocco `catch` del modulo `axios`. Usando un costrutto condizionale ternario in JSX, mostriamo un messaggio di errore nel caso in cui il flag abbia valore `true`.

Una soluzione più concisa consiste nel non usare il flag booleano ma nel gestire il valore `null` direttamente.

```
export default function Test() {
    const [resource, setResource] = useState(null);

    useEffect(() => {
         axios.get('/api/resource').then(resp => {
             setResource(resp.data);
         }).catch(err => {
             console.log(err);
         });
    }, []);

    return (
        {resource && <h1>{resource.title}</h1>}
    );
}
```

`null` è un valore di tipo falsy, quindi in questo caso l'accesso alla variabile `title` avverrà solo se `resource` non è `null`.

Uno scenario più complesso è quello delle risorse che contengono oggetti con proprietà il cui valore può essere `null`. In questo caso quando si cerca di accedere a queste proprietà, ad esempio nel caso di oggetti annidati, si ottiene un errore nell'espressione JSX.

```
return (
    {posts.map((post) => (
        <article key={post.id}>
            <h2>{post.title}</h2>
            <div>{post.share.views}</div>
        </article>
    ))}
);
```

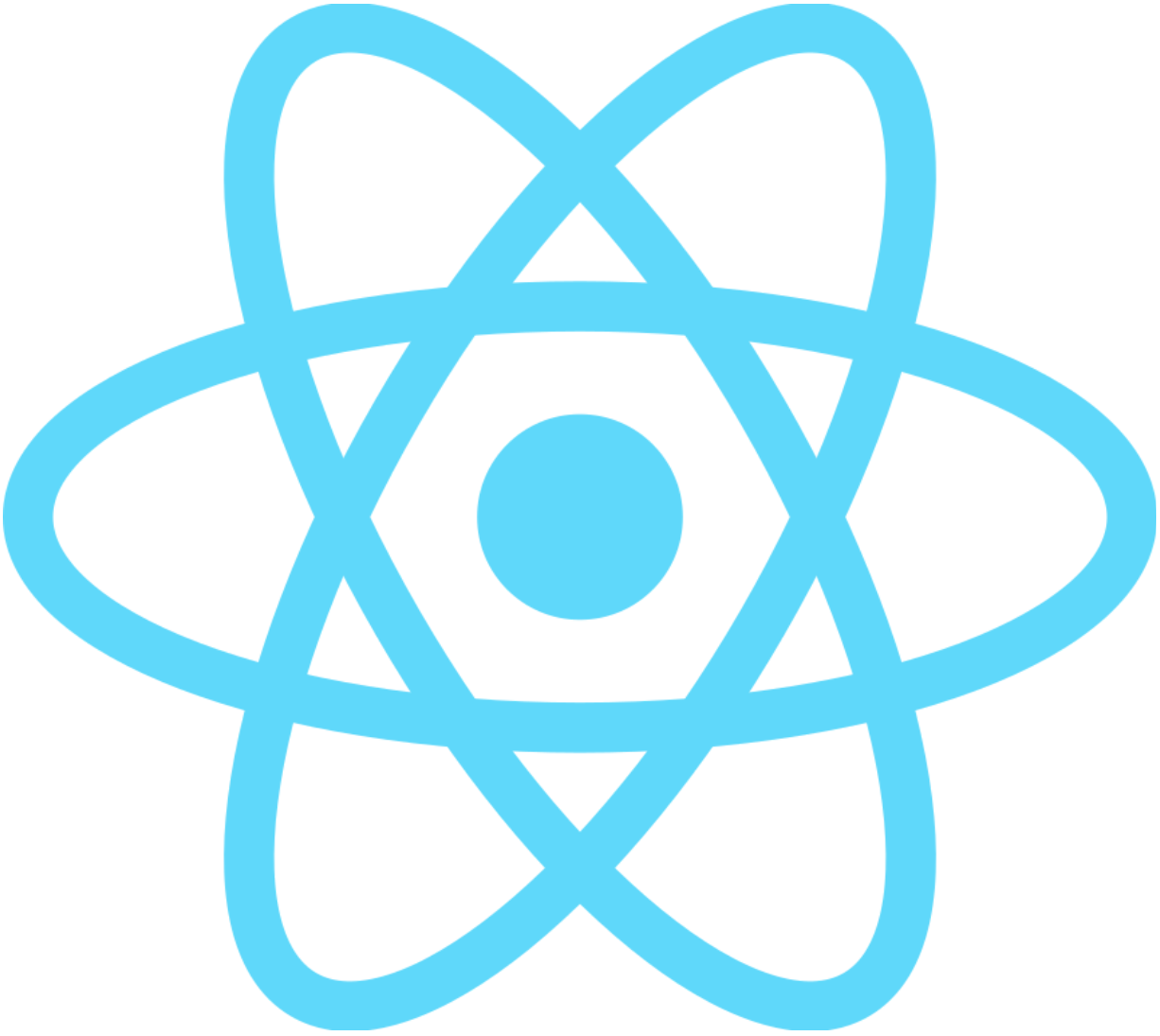
In questo caso, se la proprietà `share` dell'oggetto `post` è `null`, accedere alla proprietà `views` restituirebbe un errore.

La soluzione è sempre quella di usare la logica condizionale all'interno di un'espressione JSX.

```
return (
    {posts.map((post) => (
        <article key={post.id}>
            <h2>{post.title}</h2>
            <div>{post.share && post.share.views}</div>
        </article>
    ))}
);
```

In generale, questi accorgimenti non sono sempre necessari quando le API REST di riferimento sono state progettate per evitare di restituire questo tipo di valore, ma può accadere di dover operare con API non perfettamente concepite e quindi occorre essere preparati.

Applicazioni Correlate



- ### Book Store

  Un'applicazione in React con Node.js e Fastify come backend.
  DockerDocker ComposeNode.jsJavaScriptFastifyReact