React: how to handle the null value

In React, as in JavaScript and in other contexts in general, the null value can be problematic to manage.

If we are retrieving a single resource from a REST API, very often we receive a single JSON object as a response. This is a very frequent case of using the `null` value as the initial value in the state of a component.

In fact, `null` indicates that an object has no value. This happens if for example we have a parametric route and the REST API returns an HTTP 404 error. Obviously the response format depends on the design of the individual API, but it is not uncommon for this value to be returned. By properly handling HTTP status codes we can solve this problem by showing a view indicating that the resource was not found.

However, it should be remembered that there are remote APIs that do not strictly follow the REST paradigm whene it comes to HTTP status codes. In fact, some of them may always return a 200 status code and report the error only in the response format.

So let's imagine that we have a component structured like this:

```
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>
    );
}
```

The initial scenario of the example is the simplest. In case the API returns the HTTP status correctly, we can handle the fact that `resource` stays on the `null` value like so:

```
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>
        }
    );
}
```

We set a boolean flag to signal an error in the HTTP request via the `catch` block of the `axios` module. Using a ternary conditional construct in JSX, we display an error message if the flag is `true`.

A more concise solution is not to use the boolean flag but to handle the `null` value directly.

```
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` is a falsy value, so in this case the access to the variable `title` occurs if `resource` is not `null`.

A more complex scenario is when a resource contains objects with properties whose value can be `null`. In this case when trying to access these properties, for example in the case of nested objects, you get an error in the JSX expression.

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

In this case, if the `share` property of the `post` object is `null`, accessing the `views` property would return an error .

The solution is always to use conditional logic inside a JSX expression.

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

In general, these precautions are not always necessary when the reference REST APIs have been designed to avoid returning this type of value, but it may happen that you have to work with APIs that are not perfectly designed and therefore you must be prepared.