Data fetching using hooks

You learned more about fetching data using hooks and that fetching data from a third-party API is considered a side-effect that requires the use of the **`useEffect`** hook to deal with the Fetch API calls in React.

You also explored how the response from fetching third-party data might fail, or be delayed, and that it can be useful to provide different renders, based on whether or not the data has been received.

In this reading, you will explore the different approaches to setting up the **`useEffect`** hook when fetching JSON data from the web. You will also learn why it can be useful to provide different renders, based on whether or not the data has been received.

You have previously learned about using the Fetch API to get some JSON data from a third-party website in plain JavaScript.

You'll be glad to learn that data fetching is not that different in React.

There is only one more ingredient that you need to keep in mind when working with React, namely, that fetching data from a third-party API is considered a side-effect.

Being a side-effect, you need to use the useEffect hook to deal with using the Fetch API calls in React.

To understand what that entails, let me illustrate it with a code example where a component is fetching some data from an external API to display information about a cryptocurrency.

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

export default function App() {
  const [btcData, setBtcData] = useState({});
  useEffect(() => {
    fetch(`https://api.coindesk.com/v1/bpi/currentprice.json`)
      .then((response) => response.json())
      .then((jsonData) => setBtcData(jsonData.bpi.USD))
      .catch((error) => console.log(error));
  }, []);

  return (
    <>
      <h1>Current BTC/USD data</h1>
      <p>Code: {btcData.code}</p>
      <p>Symbol: {btcData.symbol}</p>
      <p>Rate: {btcData.rate}</p>
      <p>Description: {btcData.description}</p>
      <p>Rate Float: {btcData.rate_float}</p>
    </>
  );
}
```

This example shows that in order to fetch data from a third party API, you need to pass an anonymous function as a call to the **`useEffect`** hook.

```
useEffect(
    () => {
        // ... data fetching code goes here
    },
    []
);
```

The code above emphasizes the fact that the `useEffect` hook takes two arguments, and that the first argument holds the anonymous function, which, inside its body, holds the data fetching code.

Alternatively, you might extract this anonymous function into a separate function expression or function declaration, and then just reference it.

Using the above example, that code could be presented as follows:

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

export default function App() {
  const [btcData, setBtcData] = useState({});

  const fetchData = () => {
    fetch(`https://api.coindesk.com/v1/bpi/currentprice.json`)
      .then((response) => response.json())
      .then((jsonData) => setBtcData(jsonData.bpi.USD))
      .catch((error) => console.log(error));
  };

  useEffect(() => {
    fetchData();
  }, []);

  return (
    <>
      <h1>Current BTC/USD data</h1>
      <p>Code: {btcData.code}</p>
      <p>Symbol: {btcData.symbol}</p>
      <p>Rate: {btcData.rate}</p>
      <p>Description: {btcData.description}</p>
      <p>Rate Float: {btcData.rate_float}</p>
    </>
  );
}
```

The code essentially does the same thing, but this second example is cleaner and better organized.

One additional thing that can be discussed here is the `return` statement of the above example.

Very often, the response from fetching third-party data might fail, or be delayed. That's why it can be useful to provide different renders, based on whether or not the data has been received.

The simplest conditional rendering might involve setting up two renders, based on whether or not the data has been successfully fetched.

For example:

```
 return someStateVariable.length > 0 ? (
    <div>
      <h1>Data returned:</h1>
      <h2>{someStateVariable.results[0].price}</h2>
    </div>
  ) : (
    <h1>Data pending...</h1>
  );
```

In this example, I'm conditionally returning an `h1` and `h2`, **if** the length of the `someStateVariable` binding's length is longer than 0.

This approach would work if the `someStateVariable` holds an array.

If the `someStateVariable` is initialized as an empty array, passed to the call to the `useState` hook, then it would be possible to update this state variable with an array item that might get returned from a `fetch()` call to a third-party JSON data provider.

If this works out as described above, the length of the `someStateVariable` would increase from the starting length of zero - because an empty array's length is zero.

Let's inspect the conditional `return` again:

```
return someStateVariable.length > 0 ? (
   <div>
     <h1>Data returned:</h1>
     <h2>{someStateVariable.results[0].price}</h2>
   </div>
 ) : (
   <h1>Data pending...</h1>
 );
```

If the data fetching fails, the text of "Data pending..." will render on the screen, since the length of the `someStateVariable` will remain being zero.

Conclusion

You learned more about fetching data using hooks and that fetching data from a third-party API is considered a side-effect that requires the use of the `useEffect` hook to deal with the Fetch API calls in React.

You also explored how the response from fetching third-party data might fail, or be delayed, and that it can be useful to provide different renders, based on whether or not the data has been received.