React - Fetching Data

React - Fetching Data

"React's true power lies in its simplicity and the way it encourages you to think about your UI as a function of your application's state."

What it means to fetch data in React?

- Understanding how to fetch data into React applications is mandatory for every React developer who aims to build modern, real-world web applications.

- In this guide, we will cover the modern React data-fetching methods and learn how to handle our application's state while fetching data. Furthermore, we will cover how to handle the application's state when something goes wrong with the data.

Before we fetch data

- When we request data, we must prepare a state to store the data upon return. We can store it in a state management tool like Redux or store it in a context object. But, to keep things simple, we will store the returned data in the React local state.

- Next, if the data doesn't load, we must provide a state to manage the loading stage to improve the user experience and another state to manage the error should anything go wrong. This gives us three state variables like so:

Before we fetch data

```
const [data, setData] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
```

How itworks?

How it works?

- When we request to fetch data from the backend, we perform a side effect, which is an operation that can generate different outputs for the same data fetching. For instance, the same request returns a success or error.
- In React, we should avoid performing side effects directly within the component body to avoid inconsistencies. Instead, we can isolate them from the rendering logic using the useEffect Hook.

How it works?

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

How it works?

- The implementation above will run and fetch data on a component mount, that is, on the first render. This is sufficient for most of our use cases.

- In other scenarios, however, when we need to refetch data after the first render, we can add dependencies in the array literal to trigger a rerun of useEffect.

How it works?

- The Fetch API through the fetch() method allows us to make an HTTP request to the backend. With this method, we can perform different types of operations using HTTP methods like the GET method to request data from an endpoint, POST to send data to an endpoint, and more.
- Since we are fetching data, our focus is the GET method.

How it works?

`fetch()` requires the URL of the resource we want to fetch and an optional parameter:

```
fetch(url, options)
```

We can also specify the HTTP method in the optional parameter. For the `GET` method, we have the following:

```
fetch(url, {
  method: "GET" // default, so we can ignore
})
```

How it works?

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

export default function App() {
 const [data, setData] = useState(null);
 const [loading, setLoading] = useState(true);
 const [error, setError] = useState(null);

 useEffect(() => {
  fetch(`https://jsonplaceholder.typicode.com/posts`)
   .then((response) => console.log(response));
 }, []);

 return <div className="App">App</div>;
}
```

How it works?

In the code, we are using the fetch() method to request post data from the resource endpoint as seen in the useEffect Hook. This operation returns a promise that could either resolve or reject.

If it resolves, we handle the response using .then(). But at this stage, the returned data is a Response object, which is not the actual format that we need, although it is useful to check for the HTTP status and to handle errors.

How it works?

```
useEffect(() => {
  fetch(`https://jsonplaceholder.typicode.com/posts`)
    .then((response) => response.json())
    .then((actualData) => console.log(actualData));
}, []);
```

How it works?

```
useEffect(() => {
 fetch(`https://jsonplaceholder.typicode.com/posts`)
  .then((response) => response.json())
  .then((actualData) => console.log(actualData))
  .catch((err) => {
   console.log(err.message);
  });
}, []);
```

How it works?

```
useEffect(() => {
  fetch(`https://jsonplaceholder.typicode.com/posts`)
    .then((response) => {
      if (!response.ok) {
        throw new Error(
          `This is an HTTP error: The status is ${resp
        );
      }
      return response.json();
    })
    .then((actualData) => console.log(actualData))
    .catch((err) => {
      console.log(err.message);
    });
}, []);
```

Rendering the posts in the frontend

- Presently, we have the posts in the console. Instead, we want to render them in our app. To do that, we'll first limit the total post number to 8 instead of the 100 posts returned for brevity.

- fetch(`https://jsonplaceholder.typicode.com/posts?_limit=8`)

Further reading

Further reading

- https://use-http.com/#/
- https://www.freecodecamp.org/news/fetch-data-react/
- https://reactjs.org/docs/faq-ajax.html
- https://www.codingdeft.com/posts/react-fetch-data-api/

Assignment

Assignment

- - continuand ideea de la curs la
- 1. partea de News trebuie

- // documentatia NEWS API https://newsapi.org/docs/endpoints/everything
- implementat un input.
- Acest input ia valori de la tastatura
- la click pe buton se face query cu valoarea inputului.

- 2. partea de POD (NASA API)
- // documentatia NASA (Open APIs) https://api.nasa.gov/
- de implementat un previous and next button (care sa caute POD din ziua de dinainte / de ziua de dupa)
- + calendar picker din care se extrage date-ul, dupa care se face request cu Data respectiva.