educación IT
CODE YOUR FUTURE

ReactJS Developer

Módulo 6

educación IT
CODE YOUR FUTURE

Consumo de *APIs* 2

API Fetch

La *API Fetch* proporciona **una interfaz para recuperar recursos** (incluso a través de la red). Resultará familiar a cualquiera que haya usado `XMLHttpRequest`.

El método **`fetch()` toma un argumento obligatorio, la ruta de acceso al recurso que desea recuperar.** Devuelve una **`Promise`** (en-US) que retorna en **`Response`** a esa petición, sea o no correcta. También puede pasar opcionalmente un objeto de opciones **`init`** como segundo argumento.

Una vez que **`Response`** es recuperada, hay varios métodos disponibles para definir cuál es el contenido del cuerpo y cómo se debe manejar. Este método devuelve una promesa, que puede ser gestionada con `async` y `await`.

Nota sobre promesas en *useEffect*

Si queremos usar **`async/await`** en `useEffect`, podríamos pensar en hacerlo como en la imagen a la derecha.

Sin embargo, esto no es recomendable ¿Por qué?

Recordemos que la función retornada por el *callback* que pasamos a **`useEffect` será llamada directamente al desmontar el componente.** Si el *callback* es un **`async function`, retorna una promesa en vez de una función, por lo que React no podrá *procesarla*.**

```
useEffect(async () => {

    console.log("Código");

    return () => {
        console.log("Limpieza");
    }
});
```

Nota sobre promesas en useEffect

Si queremos usar **`async/await`** en `useEffect`, podríamos pensar en hacerlo como en la imagen a la derecha.

Sin embargo, esto no es recomendable ¿Por qué?

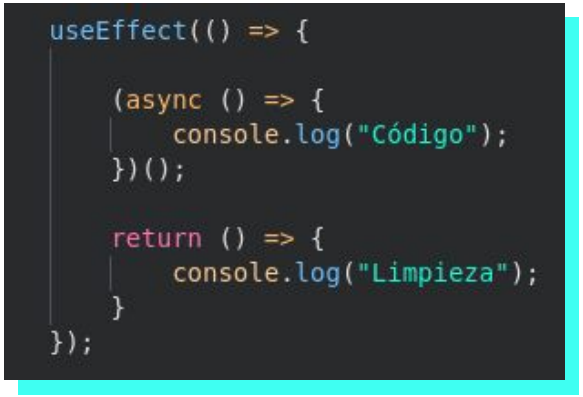
Recordemos que la función retornada por el *callback* que pasamos a **`useEffect` será llamada directamente al desmontar el componente.** Si el *callback* es un **`async function`, retorna una promesa en vez de una función, por lo que React no podrá *procesarla*.**

```
useEffect(async () => {

    console.log("Código");

    return () => {
        console.log("Limpieza");
    }
});
```

En su lugar, es recomendable usar `async function` como una expresión de función **autoinvocada.**

Al declarar y llamar una función asincrónica, se pone en marcha su proceso y puede hacer los cambios de estado correspondientes.

De esta forma **evitamos que el *callback* pasado a `useEffect` sea asincrónico,** por lo que React vuelve a tener acceso a la función de limpieza retornada.



Solicitud *GET* simple

- Usando **`async/await`** vemos que la sintaxis **es mucho más clara.**
- La función **`fetch()` retorna un objeto `Response`.**
- **`Response`** tiene métodos para **convertir el cuerpo de la respuesta.**

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

function Posts(props) {
    // Petición GET Fetch

    const [posts, setPosts] = useState([]);

    useEffect(() => {

        (async function() {
            const response = await fetch('/posts');
            const data = await response.json();
            setPosts(data);
        })();

    });

    return (
        <ul>
            {posts.map((p, i) => (
                <li key={i}>{p.titulo}</li>
            ))}
        </ul>
    );
}

export default Posts;
```

Envío de formulario

- Para enviar un formulario, necesitamos **hacer una petición `POST` enviando los datos como cuerpo de la petición.**
- Para enviar los datos, podemos usar o bien **`FormData`** o **convertirlos a JSON y enviarlos en el cuerpo de esa petición.**

```
function LoginForm(props) {
    const [email, setEmail] = useState("");
    const [response, setResponse] = useState("");

    const handleSubscribe = () => {
        const formData = new FormData();
        formData.append("email", email);

        (async function () {
            const response = await fetch("/subscribe", {
                method: "POST",
                body: formData,
            });
            const data = await response.json();
            setResponse(data);
        })();
    };

    return (
        <div>
            <h2>Subscribite</h2>
            <input
                onInput={(e) => setEmail(e.target.value)}
                value={email}
                type="email"
                placeholder="Tu email aqui..."
            />
            <button onClick={handleSubscribe}>Subscribite</button>
            {response}
        </div>
    );
}
```

Preloader

- En el caso de **`fetch`,** una función que no ofrece conocimiento sobre los estados intermedios, la única estrategia viable para implementar el *Preloader* **es la variable bandera.**
- Creamos una variable que tiene **un valor inicial y ese valor cambia cuando ya hay una respuesta.**

```
1  import { useEffect, useState } from "react";
2
3  function Posts(props) {
4      const [loaded, setLoaded] = useState(false);
5      const [posts, setPosts] = useState([]);
6
7      useEffect(() => {
8          (async function () {
9              const response = await fetch("/posts");
10             const data = await response.json();
11             setLoaded(true);
12             setPosts(data);
13         })();
14     });
15
16     return loaded ? <PostList data={posts} /> : <Preloader />;
17 }
18
19 export default Posts;
```

Librería *Axios*

`Axios` es un cliente HTTP basado en promesas para Node.js y el desarrollo del lado del cliente. Es una librería **isomórfica,** es decir, que **usa el mismo código para hacer peticiones del lado del cliente como del servidor.** Una de las características centrales es **la posibilidad de tener una instancia base común.** Esto permite **reducir la redundancia de código.**

`Axios` usa:

- Del lado del **servidor, el módulo HTTP nativo de Node.**
- Del lado del **cliente, `XMLHttpRequest`.**

Para usarlo necesitamos instalarlo.

Creando la instancia base común

- `Axios` permite crear **una instancia base común.** Esta instancia permite **aislar todos los parámetros que deban repetirse de petición en petición,** como la URL base o autorización.
- Este archivo se llama '***api.js***' y **exporta dicha instancia base.**
- Esta instancia **será utilizada luego por los componentes.**

```
1  import axios from "axios";
2
3  const base = axios.create({
4      baseURL: 'http://api.mistio.com/api/v2',
5      headers: {
6          'X-Access-Token': 'Tu token aquí'
7      }
8  });
9
10 export default base;
```

Petición *GET* simple

- La instancia base **expone métodos con los verbos HTTP más usados** (por ejemplo: **`instancia.get()`** hace una petición **`GET`** e **`instancia.post()`** hace una petición **`POST`**).
- El primer parámetro **es la URL a la cual se hace la petición** y **el segundo parámetro,** opcional, **son los datos extra.**
- Esta función **retorna una promesa.**
- Si se resuelve esta promesa, **contamos con los datos formateados como JSON directamente.**

```jsx
import { useEffect, useState } from "react";
import apiClient from './api';

function Posts(props) {
    const [posts, setPosts] = useState([]);

    useEffect(() => {

        (async function() {
            const axiosRes = await apiClient.get('/posts');
            setPosts(axiosRes.data);
        })();

    });

    return (
        <ul>
            {posts.map((p, i) => (
                <li key={i}>{p.titulo}</li>
            ))}
        </ul>
    );
}

export default Posts;
```

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!