Front End III

Pokemones

Supongamos que tenemos un componente que efectúa un request a la API para obtener y mostrar la información de un determinado Pokémon. En tal caso, vamos a crear un Hook personalizado para manejar este tipo de request, utilizando como base lo que vimos en clases anteriores. Dicho Hook se vería como sigue:

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

// Creamos un objeto para almacenar
// los posibles estados
export const statuses = {
 SUCCESS: "SUCCESS",
 ERROR: "ERROR",
 LOADING: "LOADING"
};

export const useFetch = (url) => {
 // Almacenamos el estado del request y
 // la información proveniente de la API
 // en dos estados dentro del Hook personalizado
 // Inicialmente, nuestro estado será "Cargando"
 const [status, setStatus] = useState(statuses.LOADING);
 const [data, setData] = useState();

 // Creamos una función asincrónica para
```

```
// obtener la información de la API
const fetchData = useCallback(async () => {
  // Seteamos el estado "Cargando" nuevamente.
  // Esto es útil en caso de que cambie la URL
  // en algún momento y se vuelva a realizar el request
  setStatus(statuses.LOADING);

  try {
    const response = await fetch(url);

    // En caso de que la respuesta no sea
    // satisfactoria, arrojamos un error
    if (!response.ok) throw new Error("Request Error");

    const json = await response.json();

    // Si todo salió bien, guardamos la información
    // y actualizamos el estado del request
    setData(json);
    setStatus(statuses.SUCCESS);
  } catch (error) {
    // En caso de error, almacenamos dicho estado
    // para poder retornarlo y utilizarlo en el componente
    setStatus(statuses.ERROR);
  }
}, [url]);
```

```
// Disparamos la función cada vez que se actualice la URL
// La URL está como dependencia del useCallback, lo que
// va a actualizar dicha función (y por ende disparar el useEffect)
// en caso de que cambie la misma
useEffect(() => {
  fetchData();
}, [fetchData]);

// Retornamos los dos estados de nuestro Hook para poder
// utilizarlo en el componente
return { data, status };
};
```

Como puede verse, dentro del Hook personalizado estamos implementando un bloque **try…catch** para manejar los distintos estados de la petición en caso de éxito o error. Este paso es necesario, ya que nos permitirá utilizar dicho estado dentro del componente que llame al Hook y, con ello, poder mostrar un mensaje personalizado al usuario, dependiendo de qué valor tenga dicho estado en un momento dado. Veamos entonces cómo podríamos implementar este Hook dentro de un componente para poder obtener y mostrar la información de un Pokémon. Para esto, crearemos un componente “Pokemon” el cual se verá como sigue:

```
// Recibimos el nombre del Pokémon como "prop"
export default function Pokemon({ pokemon }) {
 // Utilizamos nuestro Hook personalizado, pasándole una URL
 // construida dinámicamente con el nombre del pokemon
 const { data, status } = useFetch(
   `https://pokeapi.co/api/v2/pokemon/${pokemon}`
 );
```

```
  // Dependiendo del estado de la petición, damos feedback
  // visual a la persona
  if (status === statuses.LOADING) return <p>Cargando...</p>;
  if (status === statuses.ERROR)
    return <p>Ha ocurrido un error al obtener el pokemon</p>;

  // Una vez que tenemos la información, la renderizamos en la pantalla
  return (
    <div className="pokemonCard">
      <h3>{data.name}</h3>
      <img src={data.sprites.front_default} alt={data.name} />
    </div>
  );
}
```

Para utilizar el componente, simplemente lo llamamos dentro de nuestra aplicación y le pasamos el nombre del Pokémon cuya información deseamos obtener:

```
<Pokemon pokemon="pikachu" />
```