ThePower business school

🇪🇸 Fetching data ReactJS

Introducción

Supongamos un listado de búsqueda que recibimos del server y cada vez que introducimos un cambio en el input queremos que se vayan filtrando los resultados enviando una petición al server y pintando la nueva lista.

Code

Para ello vamos a crear un componente llamado `CodeFetchingInput.jsx` . Este componente tendrá dos states, uno para guardar el filtro actual y otro para recoger la colección de pokemon.

```
const [filter, setFilter] = useState("ditto");

const [pokemonCollection, setPokemonCollection] = useState([]);
```

Ahora continuamos definiendo un Input que haga de filtro y pintando el Pokemon cuando coincida con lo escrito en su interior.

```
return (
   <div>
     <input value={filter} onChange={(e) => setFilter(e.target.value)} />

     <ul>
       {pokemonCollection.map((pokemon) => (
         <li key={pokemon.name}>
           <h1>{pokemon.name}</h1>
           <img src={pokemon.image} alt={pokemon.name} />
         </li>
       ))}
     </ul>
   </div>
 );
```

¡Pero para pintar al pokemon necesitamos traerlo pokemon de una API! ⚠️

```
useEffect(() => {
  const getPokemonFiltered = async () => {
    const pokemonList = await fetch(
      `https://pokeapi.co/api/v2/pokemon/${filter}`
    );

    const pokemonListToJson = await pokemonList.json();

    return {
      ...pokemonListToJson,
      name: pokemonListToJson.name,
      image: pokemonListToJson.sprites.front_shiny,
    };
  };

  getPokemonFiltered().then((pokemon) => setPokemonCollection([pokemon]));
}, [filter]);
```

¡Listo! Ya podemos probar nuestro componente y ver el uso que podemos hacer de este, como tenemos a Ditto por defecto nos lo pintará hasta que exista otro Pokemon válido, lanzaremos tantas peticiones como veces cambie el state de nuestro filter.

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

export const CodeFetchingInput = () => {
  const [filter, setFilter] = useState("ditto");
  const [pokemonCollection, setPokemonCollection] = useState([]);

  useEffect(() => {
    const getPokemonFiltered = async () => {
      const pokemonList = await fetch(
        `https://pokeapi.co/api/v2/pokemon/${filter}`
      );

      const pokemonListToJson = await pokemonList.json();

      return {
        ...pokemonListToJson,
        name: pokemonListToJson.name,
        image: pokemonListToJson.sprites.front_shiny,
      };
```

```
    };

    getPokemonFiltered().then((pokemon) => setPokemonCollection([pokemon]));
  }, [filter]);

  return (
    <>
      <input value={filter} onChange={(e) => setFilter(e.target.value)} />

     <ul>
       {pokemonCollection.map((pokemon) => (
         <li key={pokemon.name}>
           <h1>{pokemon.name}</h1>
           <img src={pokemon.image} alt={pokemon.name} />
         </li>
       ))}
     </ul>
    </>
  );
};
```

Y quedaría algo como esto.

useDebounce

¿Te has fijado en que hacemos muchísimas peticiones? Vamos a añadir una pequeña mejora para esto, aunque podemos añadir un tiempo de espera usando una libería

para manejar los tiempos de invocación en la petición con `use-debounce`, para ello instalamos en nuestro proyecto este hook personalizado.

```
npm i use-debounce
```

Y os dejamos el código por aquí.

```
import { useState, useEffect } from "react";
import { useDebounce } from "use-debounce";

export const MiniCodeFetchingDebounce = () => {
  const [filter, setFilter] = useState("ditto");
  // Esto hace que la función espere 500ms antes de ser invocada
  const [debounceFilter] = useDebounce(filter, 500);
  const [pokemonCollection, setPokemonCollection] = useState([]);

  useEffect(() => {
    const getPokemonFiltered = async () => {
      const pokemonList = await fetch(
        `https://pokeapi.co/api/v2/pokemon/${filter}`
      );

      const pokemonListToJson = await pokemonList.json();

      return {
        ...pokemonListToJson,
        name: pokemonListToJson.name,
        image: pokemonListToJson.sprites.front_shiny,
      };
    };

    getPokemonFiltered().then((pokemon) => setPokemonCollection([pokemon]));
  }, [debounceFilter]);

  return (
    <div>
      <input value={filter} onChange={(e) => setFilter(e.target.value)} />

     <ul>
       {pokemonCollection.map((pokemon) => (
         <li key={pokemon.name}>
           <h1>{pokemon.name}</h1>
           <img src={pokemon.image} alt={pokemon.name} />
         </li>
       ))}
     </ul>
    </div>
```

```
  );
};
```

Mini-ejercicio

Componetiza cada uno de los elementos aislando en cada componente la lógica asociada.