ThePower business school

🇪🇸 Proyecto Pokedex ReactJS

Introducción

La `Pokédex` es uno de los proyectos más conocidos para aprender React consumiento y manejando una API externa.

Lo primero que se nos viene a la cabeza al pensar en una `Pokédex` es un buscador donde encontrar al Pokémon que queramos y ver sus propiedades.

Para ello vamos a hacer uso de la `PokéAPI`, una API REST con todos los datos sobre Pokémon:

PokéAPI

An open RESTful API for Pokémon data

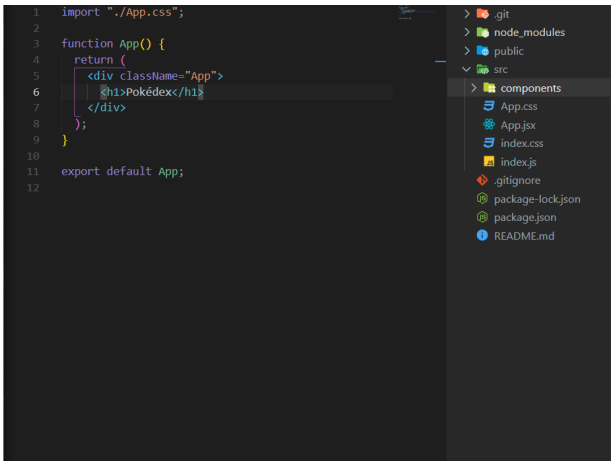
https://pokeapi.co/

Ahora mismo la API puede ser un poco confusa, así que de momento vamos a empezar creando nuestro proyecto de React:

```
npm create vite@latest
```

Una vez limpiemos nuestro proyecto y comprobado que arranca correctamente podemos empezar a trabajar en él.

Pokédex



Inicio

Al ser una aplicación bastante visual vamos a utilizar una paleta de colores basada en los videojuegos de Pokémon con tonos pastel. En cualquier editor de imágenes o a través de una extensión de Color Picker podremos obtener los valores de los mismos y aplicarlos a nuestra hoja de estilos.

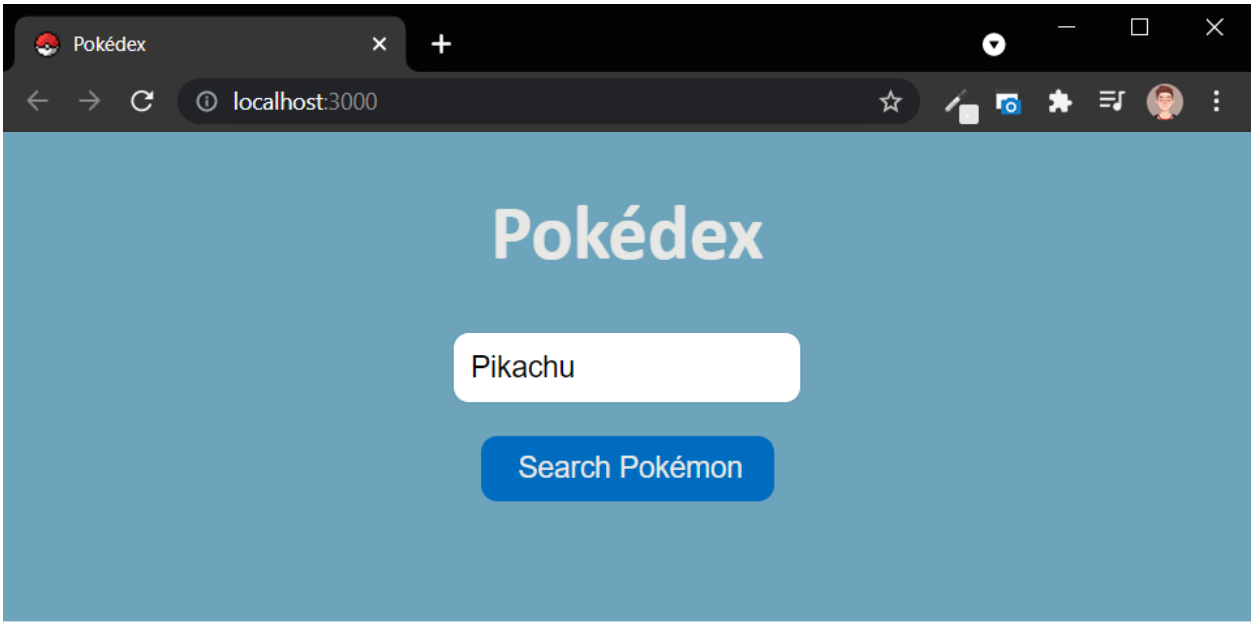
Vamos a crear un título, una barra de búsqueda y un botón que nos permita buscar a los Pokémon, ya le daremos funcionalidad más adelante:

```
import "./App.css";
import {useState} from "react";

const App = () => {

  return (
    <div className="App">
      <div className="TitleSection">
        <h1>Pokédex</h1>
```

```
        <input type="text" />
        <button>Search Pokémon</button>
      </div>
    </div>
  );
}

export default App;
```

```
body {
  margin: 0;
  padding: 0;
}

.App {
  display: flex;
  justify-content: center;
  font-family: Calibri;
  font-size: 24px;
}

.TitleSection {
  background-color: #6ea4bb;
  width: 100%;
  height: 300px;
  display: flex;
  flex-direction: column;
  align-items: center;
  color: #e7e7e6;
}

.TitleSection input {
  width: 200px;
  height: 40px;
  border-radius: 10px;
  border: 1px solid #5e98b1;
  font-size: 18px;
  padding-left: 10px;
}

.TitleSection button {
  width: 180px;
  height: 40px;
  margin: 20px;
  border-radius: 10px;
  font-size: 18px;
  padding-left: 10px;
  border: none;
  color: #e7e7e6;
```

```
  background-color: #006cbf
}

.TitleSection button:hover {
  cursor: pointer;
}
```



De esta manera le hemos dado un acabado general a nuestra página y podremos ir aplicando todas las funcionalidades. Recordamos que podéis darle el estilo que queráis a la página web, por lo que recomendamos que juguéis con la hoja de estilo para darle vuestro propio toque e incluso cambiar los elementos de lugar.

De momento solo tenemos la parte superior de nuestra aplicación, así que vamos a empezar a crear la lógica para pintar nuestros Pokémon.

Lógica de la Pokedex

Vamos a hacer que nuestro input recoja los nombres de los Pokémon que queremos buscar y nos lo setee en la constante `pokemonName` a través de un `evento` y un `useState`:

```
import "./App.css";
import { useState } from "react";

const App = () => {
  const [pokemonName, setPokemonName] = useState("");

  return (
    <div className="App">
      <div className="TitleSection">
        <h1>Pokédex</h1>
        <input
          type="text"
          onChange={(event) => {
            setPokemonName(event.target.value);
          }}
        />
        <button>Search Pokémon</button>
      </div>
    </div>
  );
};

export default App;
```

Cada vez que cambie el valor introducido en nuestro input nos lo seteará en `pokemonName`, pese a que de momento no estamos usando dicho valor. Vamos a solucionar esto dándole funcionalidad al botón `Search Pokémon` para hacerle una petición a la PokéAPI con el valor de `pokemonName`.

Tal y como vimos en el temario de Fetch & Axios vamos a instalar `axios` para hacer las peticiones de una manera más funcional y sencilla en nuestro proyecto:

```
npm install axios
```

Lo primero que tenemos que hacer es importar Axios en nuestro proyecto.

```
import Axios from axios;
```

Vamos a crear una función para buscar a los Pokémons e integrarlo en nuestro botón:

```
import "./App.css";
import { useState } from "react";
import Axios from "axios";
const App = () => {
  const [pokemonName, setPokemonName] = useState("");

  const searchPokemon = () => {
    Axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`).then(
      (res) => {
        console.log(res.data);
      }
    );
  };

  return (
    <div className="App">
      <div className="TitleSection">
        <h1>Pokédex</h1>
        <input
          type="text"
          onChange={(event) => {
            setPokemonName(event.target.value);
          }}
        />
        <button onClick={searchPokemon}>Search Pokémon</button>
      </div>
    </div>
  );
};
export default App;
```

Si le echamos un vistazo a la PokéAPI podemos ver que la manera de encontrar Pokémon por su nombre sería tal que así:

```
https://pokeapi.co/api/v2/pokemon/ditto
```

Esta ruta nos llevará a los datos del Pokémon que solicitemos devoliendonos un JSON con toda la información, por lo que el fetch lo vamos a realizar de la siguiente manera:

```
const searchPokemon = () => {
    Axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`).then(
      (res) => {
        console.log(res.data);
      }
    );
  };
```

De esta manera a la hora de realizar la petición vamos a usar `Template Strings (``)` para concatenar la ruta `/pokemon/` con nuestro valor almacenado en `pokemonName`. De esta forma, y al activar el botón `Search Pokémon`, nos hará fetch de la ruta por defecto de pokemon en la PokéAPI + el Pokémon que queramos buscar.

Al final del fetch hemos realizado una comprobación mediante `console.log` que nos pinte el resultado por consola, y podremos ver que al buscar cualquier Pokémon nos devolverá su JSON con la información:

Como podemos ver el JSON nos devuelve todas las propiedades de nuestro Pokémon, tales como la imagen, los stats, el tipo, los videojuegos donde aparece, etc...

Ahora necesitamos definir que información necesitamos del JSON y setterala en una variable que nos permita almacenarla para pintarla posteriormente:

```
const App = () => {
  const [pokemonName, setPokemonName] = useState("");
  const [pokemon, setPokemon] = useState([]);

  const searchPokemon = () => {
    Axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`).then(
        (res) => {
        setPokemon({
          name: pokemonName,
          number: res.data.id,
          species: res.data.species.name,
          image: res.data.sprites.front_default,
          hp: res.data.stats[0].base_stat,
          attack: res.data.stats[1].base_stat,
          defense: res.data.stats[2].base_stat,
          speed: res.data.stats[5].base_stat,
          type: res.data.types[0].type.name,
        });
      }
    );
  };
```

Hemos definido una variable llamada `pokemon` en la cual se almacenará la información en un array vacío como estado inicial de su `useState`.

Vamos a sobrescribir el console log que recogía nuestra información y vamos a hacerlo a través de setPokemon pasándole un objeto con la información desglosada.

Por ejemplo, hemos definido que `hp` sea `res.data.stats[0].base_stat` porque dentro del JSON hay un apartado llamado `stats` y el de `hp` es el primero, por lo tanto es la posición `[0]`. Y además queremos que se imprima la cifra del `hp`, que está definida en `base_stat`.

De esta manera hemos definido las posiciones del JSON de las cuales vamos a recuperar la información y desglosándolas en `name`, `number`, `species`, `image`, `hp`, `attack`, `defense`, `speed` y `type`.

Si hay cualquier información adicional que queráis recoger del JSON solamente tenéis que definir un nombre y pasarle la posición, por ejemplo, los videojuegos en los que aparecen, la altura o el peso del Pokémon.

Recomendamos que instaléis una extensión en el navegador como `JSON Viewer` para verlos como en la captura y poder acceder a todas sus propiedades de manera legible, ya que por defecto es una cadena de texto muy grande y bastante complicada de leer.

Vamos a recoger esta información y pintarla en nuestra aplicación para que nos muestre los Pokémon, pero antes de ello vamos a evitar ciertos errores a la hora de

llamarla creando un estado que nos indique si se ha seleccionado o no un Pokémon.

De esta manera si no está recibiendo ninguna información no nos pintará nada, de otra forma nos saldrían espacios vacíos con información inexistente:

```
import "./App.css";
import { useState } from "react";
import Axios from "axios";
const App = () => {
  const [pokemonName, setPokemonName] = useState("");
  const [pokemonChosen, setPokemonChosen] = useState(false);
  const [pokemon, setPokemon] = useState({
    name: "",
    number: "",
    species: "",
    image: "",
    hp: "",
    attack: "",
    defense: "",
    speed: "",
    type: "",
  });

  const searchPokemon = () => {
    Axios.get(`https://pokeapi.co/api/v2/pokemon/${pokemonName}`).then(
      (res) => {
        setPokemon({
          name: pokemonName,
          number: res.data.id,
          species: res.data.species.name,
          image: res.data.sprites.front_default,
          hp: res.data.stats[0].base_stat,
          attack: res.data.stats[1].base_stat,
          defense: res.data.stats[2].base_stat,
          speed: res.data.stats[5].base_stat,
          type: res.data.types[0].type.name,
        });
        setPokemonChosen(true);
      }
    );
  };

  return (
    <div className="App">
      <div className="TitleSection">
        <h1>Pokédex</h1>
        <input
          type="text"
          onChange={(event) => {
```

```
            setPokemonName(event.target.value);
          }}
        />
        <button onClick={searchPokemon}>Search Pokémon</button>
      </div>
      <div className="DisplaySection"></div>
    </div>
  );
};
export default App;
```

Hemos sustituido el array vacío del estado de `pokemon` por un objeto con todas las propiedades inicializadas en `""`. De esta forma se rellenarán con los datos en los strings vacíos y si tuviera que imprimirlos no habría conflicto con el tipo de dato que devuelve cada propiedad, tenga o no tenga contenido.

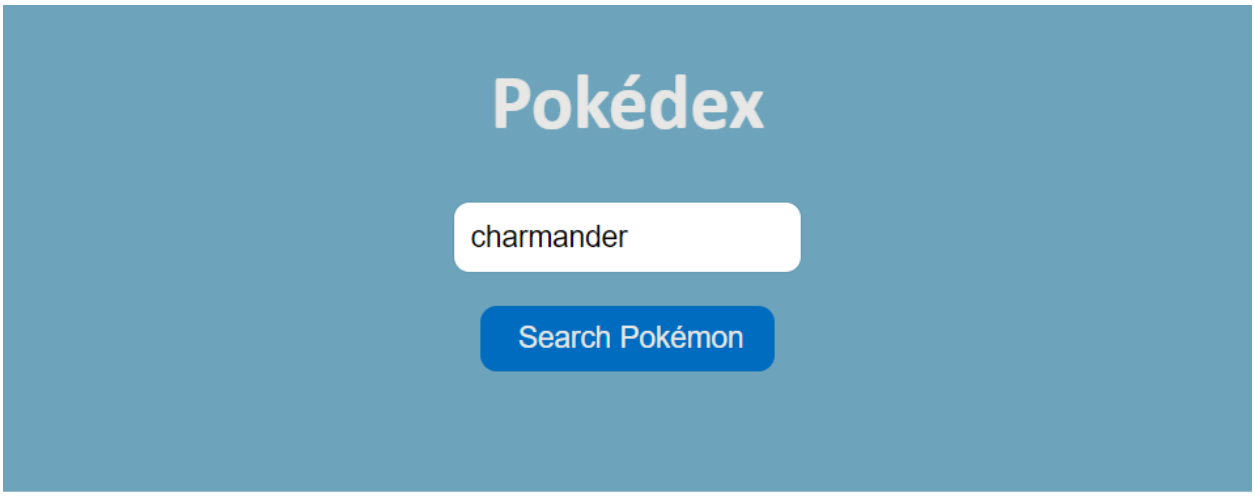
Además hemos creado un nuevo estado llamado `pokemonChosen` que nos indique si hay o no un Pokemon seleccionado seteado en su inicio como `false`. Hemos añadido a `setPokemon` la funcionalidad de setear `pokemonChosen` a `true` una vez ejecutada, asi nos indicará que efectivamente hemos seleccionado uno mediante el buscador.

Todo esto lo utilizaremos para indicar al usuario que elija un Pokémon si no lo ha hecho mediante un sencillo ternario en el return del componente `App`:

```
return (
    <div className="App">
      <div className="TitleSection">
        <h1>Pokédex</h1>
        <input
          type="text"
          onChange={(event) => {
            setPokemonName(event.target.value);
          }}
        />
        <button onClick={searchPokemon}>Search Pokémon</button>
      </div>
      <div className="DisplaySection">
        {!pokemonChosen ? (
          <h1> Please choose a Pokémon </h1>
        ) : (
          <h1>{pokemon.name}</h1>
```

```
        )}
      </div>
    </div>
```

Si no hay Pokémon seleccionado nos pedirá que lo hagamos, y si lo hay nos pintara en este caso el nombre del Pokémon para hacer la comprobación de que todo funciona.



charmander

Vamos a añadir el resto de propiedades que nos faltan:

```
<div className="DisplaySection">
        {!pokemonChosen ? (
          <h1> Please choose a Pokémon </h1>
        ) : (
          <>
            <h1>{pokemon.name}</h1>
            <img src={pokemon.image} alt={pokemon.name} />
            <h3>Number: #{pokemon.number}</h3>
            <h3>Species: {pokemon.species}</h3>
            <h3>Type: {pokemon.type}</h3>
            <h4>Hp: {pokemon.hp}</h4>
            <h4>Attack: {pokemon.attack}</h4>
            <h4>Defense: {pokemon.defense}</h4>
```

```
            <h4>Speed: {pokemon.speed}</h4>
          </>
        )}
      </div>
```

De esta manera y aplicándole algunos estilos de css, como por ejemplo el `text-transform: capitalize` a los valores que por defecto se nos pintan en minúscula tendremos nuestro buscador de Pokémon funcionando.

Con el case tenemos un problema desde el inicio, y que muchos os habréis dado cuenta al buscar un Pokémon, y es que si ponemos un nombre de Pokémon con la primera letra o todas en mayúscula no nos lo encuentra. Esto se debe a que el JSON solo recoge palabras en minúscula. Pero esto se puede solucionar de la siguiente manera:

```
<input
          type="text"
          onChange={(event) => {
            setPokemonName(event.target.value) ;
          }}
          value={pokemonName.toLowerCase()}
        />
```

Con esta función vamos a transformar el valor que insertemos en minúscula y da igual de la manera que lo escribamos que siempre nos va a leer el valor correcto.

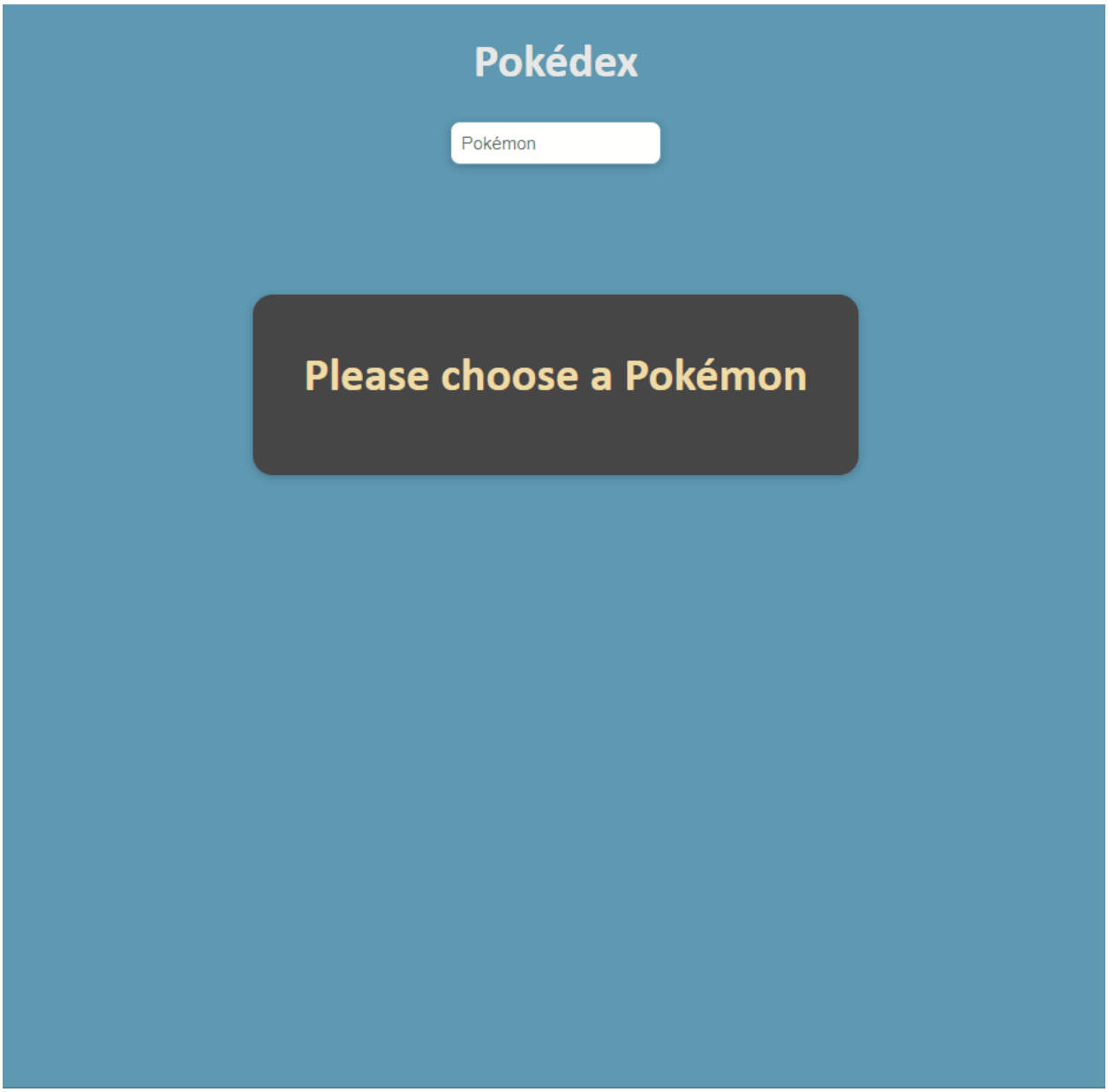
En este caso por ejemplo le he aplicado un `capitalize` también en css para que, aunque escriba en minúscula me salga debidamente escrito. Esto es completamente opcional y podéis jugar con las opciones de estilo hasta el infinito.

Hemos incluido como extra un ternario que nos oculte nuestro botón anidado en un div siempre y cuando no tenga valor la barra de búsqueda:

```
<div>
  {pokemonName && <button onClick={searchPokemon}>Search Pokémon</button>}
</div>
```

De esta manera haremos más dinámica la web viendo diferentes maneras de modificar el comportamiento de los elementos de una aplicación.

Solamente nos queda buscar a nuestro Pokémon favorito y tomar nota para saber cuál es el mejor de todos:



Ahora podéis probar mediante un fetch general listar todos los Pokémon, hacerlos clickables para pintar los detalles de cada uno o controlar todo tipo de errores y eventos de la aplicación.