Front End III

React Toastify

Para ayudarnos con esta funcionalidad, vamos a utilizar la librería react-toastify, la cual nos permite crear este tipo de notificaciones de manera sencilla. En primer lugar, para instalarla, debemos emplear el siguiente comando:

```
npm install --save react-toastify
```

o

```
yarn add react-toastify
```

Una vez instalada, necesitamos agregarla en nuestra aplicación para luego poder emplearla en cualquier momento:

```
// Importamos las dependencias necesarias
import { ToastContainer, toast } from "react-toastify";
import "react-toastify/dist/ReactToastify.css";

// Recibimos el nombre del Pokémon como "prop"
export default function App() {
 // Una vez que tenemos la información, la renderizamos en la pantalla
 return (
   <div className="App">
     {/* Agregamos el contenedor para poder emplear luego
     las notificaciones
      */}
     <ToastContainer />
   </div>
```

```
  );
}
```

Es importante tener en cuenta que solo debemos llamar una vez al componente **<ToastContainer/>** en toda la aplicación. En caso de que tengamos dudas de donde debemos ubicarlo, podemos hacerlo en el componente de más alto nivel dentro de la misma (generalmente App.js).

Una vez que completamos esta configuración inicial, ya estamos en condiciones de utilizar las notificaciones toast. Para ello, construyamos rápidamente nuestro formulario de registro:

```
export default function App() {
  // Creamos estados para manejar los campos
  // de nuestro formulario
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

  // Creamos un manejador del evento submit
  // para cuando se envíe el formulario
  const onSubmit = async (e) => {
    // Evitamos que se recargue la página
    e.preventDefault();

    // Simulamos un request a la API retornando
    // una promesa que se resuelve luego de 2 segundos
    const request = () =>
      new Promise((resolve, reject) => {
```

```
      setTimeout(() => {
        // Retornamos el nombre de usuario
        resolve(username);
      }, 2000);
    });

  // Utilizamos un bloque try...catch para llamar
  // a nuestra falsa API.
  try {
    const response = await request();
    // Si la respuesta es exitosa, mostramos un toast
    // de bienvenida
    toast(`Registro Exitoso. Bienvenido ${response}`);
  } catch (error) {
    // En caso de error, mostramos un toast de error
    toast("Ha ocurrido un error. Intente nuevamente");
  }
};

return (
  <div className="App">
    <ToastContainer />
    {/*
       Creamos nuestro formulario
     */}
    <form onSubmit={onSubmit}>
      <input
```

```
        type="text"
        placeholder="Ingresa tu nombre de usuario"
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="password"
        placeholder="Ingresa tu password"
        onChange={(e) => setPassword(e.target.value)}
      />
      <button type="submit">Registrarme</button>
    </form>
  </div>
 );
}
```

Si guardamos los cambios e interactuamos con el formulario, veremos que al cabo de 2 segundos, luego de presionar el botón, aparecerá el mensaje de éxito dentro de un toast en el margen superior derecho de la pantalla.

Ahora bien, si quisiéramos probar que efectivamente funcione el mensaje de error, podríamos modificar el retorno de nuestra promesa para que en vez de ser resuelta sea rechazada:

```
const request = () =>
    new Promise((resolve, reject) => {
      setTimeout(() => {
        // Rechazamos la promesa
        reject("Error");
      }, 2000);
    });
```

De esta manera, al cabo de dos segundos de presionar el botón, deberíamos ver el toast en el mismo lugar, solo que esta vez mostrando el mensaje de error.

De esta manera, hemos preparado nuestra aplicación para brindar notificaciones al usuario indicando el resultado del proceso, ya sea en caso de éxito o error.