Formularios en React

Si de interacción hablamos, quizás el caso más emblemático (y con el que posiblemente nos vayamos a encontrar la mayoría de las veces en nuestro camino profesional) es el formulario.

Dentro de un formulario, podemos encontrar una variedad de eventos que se disparan a medida que la persona interactúa con el mismo. En React, los elementos de formularios en HTML funcionan un poco diferente a otros elementos del DOM, debido a que los elementos de formularios conservan naturalmente algún estado interno.

Pero antes de meternos de lleno a analizar el funcionamiento de un formulario en React, es necesario introducir un concepto que nos será de gran ayuda al momento de realizar dicha tarea: la diferencia entre componentes controlados y no controlados.  
  
Los componentes controlados son controlados por React.   
Los componentes no controlados son controlados por el DOM.  
  
React necesita controlar los inputs para un manejo eficiente de los cambios.   
  
 **onChange y onSubmit en acción**

Vamos a seguir ese paso a paso para construir el siguiente ejemplo de formulario de login.  
  
  
Para este formulario vamos a emplear dos inputs controlados, empleando useState para almacenar y actualizar el estado de cada uno de ellos a medida que la persona interactúa con dichos inputs.  
Partiendo de un proyecto base de React, dentro del componente <App> vamos a declarar los dos useState que utilizaremos más adelante:  
  
export default function App() {

*// Creamos un estado para cada input del formulario*

const [userName, setUserName] = useState("");

const [password, setPassword] = useState("");  
Por su parte, vamos a necesitar un manejador de evento para cada uno de los inputs, mediante el cual vamos a actualizar el estado a medida que la persona vaya ingresando los datos correspondientes:

*// Creamos un manejador de eventos para cada uno de los inputs*

const onChangeUserName = (e) => setUserName(e.target.value);

const onChangePassword = (e) => setPassword(e.target.value);

Con estos manejadores, podremos emplear el evento onChange que tiene la etiqueta <input>. Ahora bien, también necesitamos crear un manejador para el evento onSubmit, el cual se disparará cuando la persona envíe el formulario:  
*// Creamos el manjeador para el evento onSubmit*

const onSubmitForm = (e) => {

*// Evitamos que se recargue la pagina*

*// previniendo el comportamiento por*

*// defecto*

e.preventDefault();

*// Por ahora solo mostramos el nombre del usuario*

alert(`Bienvenido: ${userName}`);  
};

Ahora que contamos con el estado y los manejadores de eventos, podemos crear nuestro formulario utilizando JSX:  
  
export default function App() {

*// … Aquí va el estado y los manejadores de eventos* return (

<div className="App">

<h3>Iniciar Sesión</h3>

{*/\* Pasamos nuestro manejador al evento onSubmit \*/*}

<form onSubmit={onSubmitForm}>

{*/\*Creamos dos inputs controlados pasando el estado como value y el manejador al evento onChange \*/*}

<input

type="text"

placeholder="Nombre de usuario"

value={userName}

onChange={onChangeUserName}

/>

<input

type="password"

placeholder="Password"

value={password}

onChange={onChangePassword}

/>

{*/\*Mediante el type nos aseguramos que se dispare el evento onSubmit al hacer click en el botón \*/*}

<button type="submit">Enviar</button>

</form>

</div>

);

}

**Recordamos validaciones**

Ya logramos construir un formulario controlado utilizando React, agregando los eventos necesarios para poder almacenar los datos ingresados y manejar el envío de dicha información a través del evento onSubmit. Para que nuestro aprendizaje sea completo, debemos retomar un tema que hemos trabajado en este track de front end: validaciones.

En la vida real, cada vez que una persona envía un formulario se produce un intercambio de información entre el front end y el back end de la aplicación: el front end recupera la información del formulario y la envía al back end por medio de un request, con la intención de almacenar dicha información, o modificar alguna información previamente existente.

Dicha situación nos plantea el desafío de asegurarnos que la información que vamos a enviar cumpla con una serie de requisitos mínimos que dependen de cada caso concreto. Al aplicar validaciones en un formulario nos aseguramos que los datos que vamos a enviar pueden ser recibidos por el back end sin ningún inconveniente.

Además, al realizar la validación en forma previa al envío de la información, evitamos realizar peticiones innecesarias cuando ya sabemos de antemano que los datos enviados no serán válidos. Entonces, ya sea por una cuestión de optimización de recursos o integridad de los datos, debemos siempre asegurarnos de realizar las validaciones correspondientes antes de enviar cualquier tipo de información al servidor.

En la práctica, existen distintos momentos en los cuales pueden llevarse a cabo las validaciones en los inputs de un formulario:

* A medida que la persona va ingresando los datos.
* Cuando abandona un campo específico (blur).
* Al momento en que se envía el formulario.

Cada opción tiene sus ventajas y desventajas. La elección dependerá de cada caso concreto y la estrategia que se decida emplear.

A los fines de esta clase, vamos a retomar el ejemplo del formulario que vimos anteriormente y agregar las validaciones correspondientes al momento en que se dispara el evento onSubmit.  
  
**Sumamos validaciones**

Vamos a retomar el trabajo en nuestro formulario de login. En este caso, vamos a realizar

dos tipos de validaciones:

1. Valoraremos que el campo nombre de usuario tenga al menos 3 caracteres y que no contenga espacios en blanco al comienzo.

2. La contraseña deberá tener al menos 6 caracteres, uno de los cuales deberá ser un número.

En primer lugar, creamos dos funciones que nos ayudarán a realizar las validaciones correspondientes:  
  
*// Creamos una funcion para validar el nombre de usuario*

const validateUserName = (userName) => {

*// Eliminamos los espacios en blanco*

const withoutSpaces = userName.trim();

*// Validamos la extension*

if (wihtoutSpaces.length > 2) {

return true;

} else {

return false;

}

};

*// Creamos una funcion para validar el password*

const validatePassword = (password) => {

*// Eliminamos los espacios en blanco*

const withoutSpaces = password.trim();

*// Separamos el string en un array para luego*

*// recorrelo y validar si existe al menos un numero*

const passwordAsArray = withoutSpaces.split("");

*// Some nos retorna true si al menos una de las*

*// iteraciones es verdadera*

const hasNumber = passwordAsArray.some((character) => {

*// Si el valor es NaN, no es un numero*

if (isNaN(character)) {

return false;

} else {

return true;

}

});

*// Validamos la extension y que haya al menos un numero*

if (withoutSpaces.length > 5 && hasNumber) {

return true;

} else {

return false;

}

};

Luego, utilizamos estas funciones para realizar las validaciones dentro del manejador de

eventos que se ejecuta al enviarse el formulario:  
  
const onSubmitForm = (e) => {

e.preventDefault();

*// Realizamos las validaciones con los valores almacenados*

*// en el estado*

const isUsernameValid = validateUserName(userName);

const isPasswordValid = validatePassword(password);

*// Si al menos una de las validaciones es falsa*

*// mostramos un mensaje de error*

if (!isPasswordValid || !isUsernameValid) {

alert("Alguno de los datos ingresados no son correctos");

} else {

*// Por ahora solo mostramos el nombre del usuario*

alert(`Bienvenido: ${userName}`);

}

};  
  
De esta manera, si alguno de los valores ingresados no cumple con los criterios de

aceptación establecidos, mostraremos un mensaje de error y no enviaremos los datos al servidor.