UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Inicial
Front End Developer
React JS Eventos

Universidad Tecnológica Nacional - Facultad Regional Buenos Aires
Medrano 951 (C1179AAQ) C.A.B.A. Tel: (54 11) 4867-7511

Bloques temáticos:

- Eventos con react
- Cómo declarar los eventos
- Eventos disponibles en react
- Cómo recibir parámetros en los eventos
- Ejemplo aplicado a nuestro código
- Ejemplo aplicado a formularios
- React hook forms

Eventos con React

React tiene sus propios eventos, que cuentan con la misma interfaz de los eventos nativos del navegador, con la ventaja que los eventos de React tienen un comportamiento compatible con la mayoría de los navegadores.

Estos eventos reciben una función, o manejadores de eventos. Lo que hacen estas funciones es definir el comportamiento de la aplicación si se corre X o Y evento.

Algo muy común en React, es declarar los manejadores de eventos como funciones dentro de una clase. Cuando hacemos esto, debemos tener cuidado, ya que `this` no se conecta por defecto a la clase. Para resolver esto, tenemos tres opciones: hacer la conexión en el `render()`, en el `constructor()`, o usar `arrow functions`.

Cómo declarar los eventos

En **Javascript** la declaración de eventos la realizamos de la siguiente manera:

```
<div onclick="handleClick () ">click me</div>
function handleClick() {
    alert('clicked');
    return false:
}
```

Como vemos utilizamos el `onclick`, en el cual hacemos referencia a una función declarada.

En **React** lo haremos de la siguiente manera:

```
<div onClick={handleClick}>click me</div>
function handleClick(event) {
    alert('clicked');
    event.preventDefault();
    event.stopPropagation();
}
```

Como vemos también utilizamos el evento `onClick`, pero a diferencia de lo realizado en Javascript solo declaramos el nombre de la función.

La función declarada sólo recibe el argumento `evento`, sobre el cual podremos realizar la acción de `stop propagation` (detener propagación o comportamiento por defecto.

Consideraciones:

- El nombre del evento tiene que ser **camelCase** y no minúscula sostenida.
- Al evento se le pasa la función y no una cadena de texto.
- En react si quieres prevenir un comportamiento por defecto o la propagación de un evento debes hacerlo explícitamente llamando los métodos `preventDefault()` y `stopPropagation()` respectivamente.

Eventos sintéticos

En este caso `event` es un evento sintético de React, en React todos los manejadores de eventos son instancias de `SyntheticEvents`. Los eventos sintéticos son una envoltura de los eventos nativos del navegador, por lo que estos eventos cuentan con la misma interfaz de los eventos nativos, como por ejemplo `preventDefault()` y `stopPropagation()`, con la ventaja de que todos estos eventos funcionan idénticamente en la mayoría de los navegadores.

Eventos disponibles en react

React soporta los siguientes eventos:

- **Mouse**: onClick, onContextMenu, onDoubleClick, onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp,
- **Drag & Drop**: onDrag, onDragEnd, onDragEnter, onDragExit, onDragLeave, onDragOver, onDragStart, onDrop.
- **Focos y formularios**: onFocus, onBlur, onChange, onInput, onSubmit.
- **Touchs**: onTouchCancel, onTouchEnd, onTouchMove, onTouchStart.
- **Cortar y pegar**: onCopy, onCut, onPaste.
- **Scrolls**: onScroll, onWheel.

Cómo recibir parámetros en los eventos

Si queremos pasar un parámetro a la función llamada en la ejecución de un evento en React debemos hacerlo utilizando la función arrow de ES6. Por ejemplo:

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>
```

Aquí vemos que cuando se ejecuta el `onClick`, se llama a la función arrow que tiene como parámetro a **e.** Luego esta función llama en su ejecución a la función `deleteRow`, la cual debe estar declarada dentro del componente.

Ejemplo aplicado a nuestro código

Aplicaremos la utilización de eventos en nuestro componente **Login**, quedando de la siguiente manera:

```
class Login extends Component {
    ingresar(e) {
        alert("entre");
    }
    render() {
        return (
            <div>
                <form>
                    <div>
                        <label>Usuario</label>
                        <input type="text" placeholder="Introduzca su
usuario" />
                    </div>
                    <div>
                        <label>Contraseña</label>
                        <input type="text" placeholder="Introduzca su
contraseña" />
                    </div>
                    <button onclick={this.ingresar}>Ingresar</button>
                </form>
            </div>
        )
    }
}
```

Como vemos en el `onClick` debemos utilizar el `this` si el método lo tenemos declarado a nivel clase.

Ahora si declaramos la función ingresar dentro del render, lo hacemos sin el `this`**.** Quedando de la siguiente manera:

```
render() {
    ingresar(e){
        alert("entre");
    }
    return (
        <div>
            <form>
                <div>
                    <label>Usuario</label>
                    <input type="text" placeholder="Introduzca su
usuario" />
                </div>
                <div>
                    <label>Contraseña</label>
                    <input type="text" placeholder="Introduzca su
contraseña" />
                </div>
                <button onClick={ingresar}> Ingresar</button>
            </form>
        </div>
    )
}
```

En el caso de las funciones, el llamado se realiza sin `this` y la misma debe estar declara en el componente:

JSX

```
<button onClick={handleClick}>Registrarse</button>
```

Función declarada en el componente

```
const handleClick = (event) => {
    console.log("handleSubmit", form)
    event.preventDefault()
}
```

Ejemplo aplicado a formularios

En el siguiente ejemplo veremos cómo aplicar el uso de eventos a un formulario en un componente tipo función. Vamos a declarar un hook para almacenar los datos introducidos en el formulario:

```
const [form, setform] = useState({ nombre: '', apellido: '', email: '',
password: '' })
```

Por otro lado vamos a declarar una función que se ejecutará con el evento **submit** del formulario:

```
const handleSubmit = (event) => {
    console.log("handleSubmit", form)
    event.preventDefault()
}
```

El console.log nos retornará los datos que el usuario introdujo en el formulario.

Vemos el formulario (jsx) desarrollado:

```
return (
    <div> <form onSubmit={handleSubmit}>
        <div> <label>Nombre</label>
            <input type="text" name="nombre" value={form.nombre}
onChange={handleChange}></input>
        </div>
        <div> <label>Apellido</label>
            <input type="text" name="apellido" value={form.apellido}
onChange={handleChange}></input>
        </div>
        <div>
            <label>Email</label>
            <input type="email" name="email" value={form.email}
onChange={handleChange}></input>
        </div>
        <div> <label>Contraseña</label>
            <input type="password" name="password" value={form.password}
onChange={handleChange}></input>
        </div>
        <button type="submit"> Registrarse</button>
    </form>
```

```
    </div >
)
```

En React el flujo de datos es unidireccional, por lo cual debemos hacer una doble relación de los datos. En este caso asociamos el **value del input** a **la propiedad del hook form** correspondiente.

Ejemplo el **input nombre** queda asociado a `form.nombre`, esto implica que cuando se modifique dicha propiedad en el hook se modificara el **value en el input**.

Por otro lado capturamos el evento `onchange` y llamamos a la función `handleChange`, en la misma realizamos la modificación del hook form de la siguiente manera:

```
const handleChange = (event) => {
    const name = event.target.name
    const value = event.target.value
    console.log("hand leChange", name, value)
    setForm({ ... form, [name]: value })
}
```

Para que esto funcione el nombre de cada propiedad del hook form tiene que llamarse igual que el atributo name del elemento, mediante esta función identificamos el elemento que está siendo modificado y actualizamos el valor del hook.

El componente completo queda de la siguiente manera:

```
import React, { useState } from "react"
function RegistroPage() {
    const [form, setform] = useState({ nombre: '', apellido: '', email: '',
password: '' })
    const handleSubmit = (event) => {
        console.log("handleSubmit", form)
        event.preventDefault()
    }
    const handleChange = (event) => {
        const name = event.target.name
        const value = event.target.value
        console.log("handleChange", name, value)
        setForm({ ...form, [name]: value })
    }
    return (
```

```
        <div> <form onSubmit={handleSubmit}>
            <div> <label>Nombre</label>
                <input type="text" name="nombre" value={form.nombre}
onChange={handleChange}></input>
            </div>
            <div> <label>Apellido</label>
                <input type="text" name="apellido" value={form.apellido}
onChange={handleChange}></input>
            </div>
            <div>
                <label>Email</label>
                <input type="email" name="email" value={form.email}
onChange={handleChange}></input>
            </div>
            <div> <label>Contraseña</label>
                <input type="password" name="password" value={form.password}
onChange={handleChange}></input>
            </div>
            <button type="submit"> Registrarse</button>
        </form>
        </div >
    )
}
export default RegistroPage;
```

React hook forms

Es una biblioteca para React, en la cual nos permite realizar construcción y validaciones de formularios de manera sencilla. https://react-hook-form.com/

En el siguiente video se puede observar cómo aplicar la misma:

https://www.youtube.com/watch?v=xjkzYyoz5k8&list=PLhA16iIUESxnCoaLz74beZ0fUDm1gemYk&index=14&t=14s

Bibliografía utilizada y sugerida

Fedosejev, A. (2015). React.js Essentials (1 ed.). EEUU, Packt.

Amler, . (2016). ReactJS by Example (1 ed.). EEUU, Packt.

Stein, J. (2016). ReactJS Cookbook (1 ed.). EEUU, Packt.

https://medium.com/@shmesa23/eventos-en-react-b53179ec9683

https://www.mmfilesi.com/blog/react-4-eventos-y-estados/

https://reactjs.org/docs/handling-events.html

http://blog.codeando.org/articulos/manejadores-de-eventos-con-react.html

https://react-hook-form.com/ https://es.reactjs.org/docs/handling-events.html