

Universidad Tecnológica Nacional
Facultad Regional Rosario
Tecnicatura Universitaria en Programación



Programación III

Unidad 3.1

Formularios avanzados

Formularios avanzados	3
Introducción	3
Creando el Login	3
Ejercicio	4
Implementando useRef	4
Manipulando el DOM con las referencias	5
Ejercicio	8
Componentes controlados versus componentes no controlados	8
Desafío	9

Formularios avanzados

Introducción

Mediante la creación de una pantalla de *Login* para el usuario, mostraremos los usos del *hook useRef* y la implementación del enrutado en un sitio web, que nos permite movernos entre componentes que simulan ser diferentes páginas de HTML.

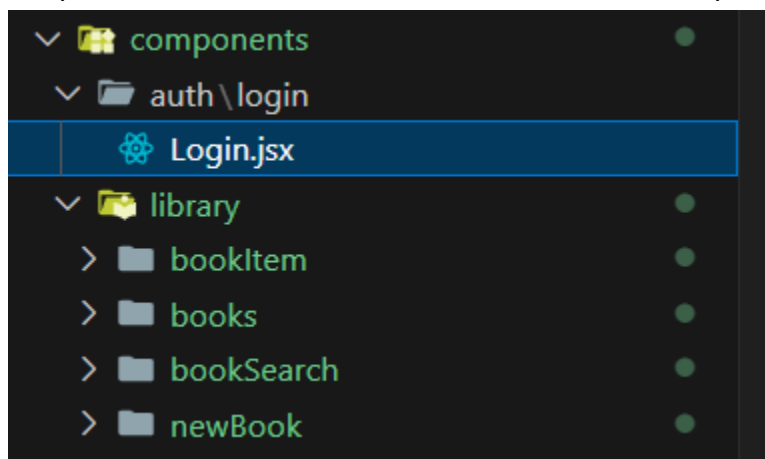
Creando el *Login*

Nuestro componente Login va a ser un sencillo componente de inicio de sesión, que poseerá dos *input* (uno para el email y otro para el password) y un botón que, al iniciar sesión correctamente, le muestra nuestra página al usuario.

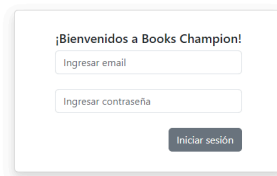
Es momento también de mejorar nuestra estructura de carpetas. Si bien React no posee una guía de estilos explícita, debemos ser prolijos a la hora de ordenar nuestras carpetas y archivos para mantener las buenas prácticas en nuestros proyectos.

Dentro de *components* entonces, dividiremos por áreas nuestra app, por ejemplo todos los componentes relacionados con *books* los podemos meter en una carpeta llamada *books*, o *booksManagement* o incluso *biz* (acrónimo usado comúnmente para lógica de negocio, viene de la palabra *bussiness*). Luego, tendremos otro tipo de áreas comunes en la app no relacionadas con el negocio, por ejemplo podríamos agregar una carpeta de *auth* (*authentication*), otra de UI (*user interface*) o también a veces llamada *shared*.

Crearemos entonces nuestro Login, dentro de la carpeta *auth* y pondremos todos los componentes relacionados a los libros dentro de la carpeta *library*:



[Código del componente Login](#)

A login form titled "¡Bienvenidos a Books Champion!". It contains two input fields: "Ingresar email" and "Ingresar contraseña". Below these fields is a button labeled "Iniciar sesión".

Para poder verlo debemos comentar todo el JSX que se encuentra dentro de App e importar el componente Login.

Ejercicio

Armar los manejadores de eventos y los estados necesarios para guardar en dos estados distintos los valores actualizados en tiempo real que ingresa el usuario de "email" y "password".

Luego, agregar un manejador de submit que cuando el usuario clickee "Iniciar Sesión", le muestre una [alerta de Navegador](#) que diga "El email ingresado es: (valor del email) y el password (valor del password)"

Implementando *useRef*

El concepto de referencias (o *refs* en React) nos permite acceder a los elementos nativos del DOM, de manera que tengamos dentro del código, un *snapshot* de lo que está sucediendo en el DOM.

¿Para qué se utiliza esto? Esto nos permite tanto obtener el valor actual del elemento (como por ejemplo en los formularios) como también aplicar eventos en el mismo (*focus*, *blur*).

Vamos a crear dos referencias a los *input* de email y password en nuestro Login. Los mismos se escriben de la siguiente manera:

```
const nameRef = useRef(null);
```

Recordemos importar *useRef* de react para poder utilizarlo. El parámetro que enviamos dentro del *useRef* (en este caso *null*) es el valor con el que va a comenzar esa referencia.

Luego, para finalizar agregamos las referencias a los elementos de JSX:

```
8   const emailRef = useRef(null);  
9   const passwordRef = useRef(null);
```

```
<Form onSubmit={handleSubmit}>  
  <FormGroup className="mb-4">  
    <Form.Control  
      type="email"  
      required  
      ref={emailRef}  
      placeholder="Ingresar email"  
      onChange={handleEmailChange}  
      value={email} />  
  </FormGroup>  
  <FormGroup className="mb-4">  
    <Form.Control  
      type="password"  
      required  
      ref={passwordRef}  
      placeholder="Ingresar contraseña"  
      onChange={handlePasswordChange}  
      value={password}  
    />  
  </FormGroup>
```

Manipulando el DOM con las referencias

Lo que buscamos es que si el usuario no ingresa email o password, se haga foco en el componente que ha dejado vacío (o en alguno de ellos si dejó vacío los dos) y además se ponga el borde rojo a manera de hacerle entender que hubo un error. Adicionalmente, comprobaremos que la contraseña tenga 7 o más caracteres.

Para poder acceder al valor que se encuentra en el input tenemos dos opciones:

- Tomarlo del valor que se encuentra en el estado.
- Tomarlo del valor que se encuentra en la referencia.

Según React, la opción más prolija es **la que usa el estado** (ver [componentes controlados versus componentes no controlados](#) más abajo). A modo de ejemplo, vamos aplicar uno por cada *input*.

```
23   const handleLogin = (event) => {  
24     event.preventDefault();  
25  
26     if (!emailRef.current.value.length) {  
27       alert("¡Email vacío!");  
28       return;  
29     }  
30  
31     else if (!password.length || password.length < 7) {  
32       alert("¡Password vacío!");  
33       return;  
34     }  
35  
36     alert(`El email ingresado es: ${email} y el password es ${password}`)  
37   }
```

!emailRef.current.value.length es lo mismo que preguntar *emailRef.current.value.length === 0*

Para acceder al valor de una referencia, debemos acceder al objeto *current* y luego a *value*.

Agregamos un estado que llevará el seguimiento de los errores en nuestro formulario:

```
7     const [errors, setErrors] = useState({  
8       email: false,  
9       password: false,  
10    });
```

```

23  const handleLogin = (event) => {
24      event.preventDefault();
25
26      if (!emailRef.current.value.length) {
27          setErrors({ ...errors, email: true });
28          alert("¡Email vacío!");
29          emailRef.current.focus();
30          return;
31      }
32
33      else if (!password.length || password.length < 7) {
34          setErrors({ ...errors, password: true });
35          alert("¡Password vacío!");
36          passwordRef.current.focus();
37          return;
38      }
39
40      setErrors({ email: false, password: false })
41      alert(`El email ingresado es: ${email} y el password es ${password}`)
42  }

```

```

54      className={errors.email && "border border-danger"}

```

Básicamente, cuando hay un error actualizamos el estado objeto *errors* y el estilizado está escuchando ese estado. Al momento de que el error se vuelva *true*, la clase *border* y *border-danger* de bootstrap se aplicará.

¡Bienvenidos a Books Champion!

Ahora, en los controladores de *onChange* debemos hacer que cuando el usuario escriba sobre esos *input*, se quiten los bordes rojos. Para ello:

```
15     const handleEmailChange = (event) => {
16         setEmail(event.target.value);
17         setErrors({ ...errors, email: false });
18     }
19
20     const handlePasswordChange = (event) => {
21         setPassword(event.target.value);
22         setErrors({ ...errors, password: false });
23     }
```

Ejercicio

Mediante uso de renderizado condicional, agregar un texto *p* abajo de los *input* que aparezca explicando al usuario que debe completar los campos para iniciar sesión.

Componentes controlados versus componentes no controlados

Debido a que el uso de *refs* en React nos permite cambiar el valor e interactuar con los elementos del formulario, ha nacido durante los últimos años dos terminologías distintas para sí el componente maneja los valores internos mediante React o mediante *refs*.

Si por ejemplo, los valores de título, autor, puntuación, cantidad de páginas, imagen y disponibilidad lo modificamos accediendo por *refs* (es decir, *nombreDeRef.current.value*), el componente sería **no controlado**, ya que las actualizaciones de esos valores no corre por React, sino por el DOM del navegador.

Si los valores de esos *inputs* los actualizamos por estado, el componente es **controlado**, ya que React *controla* qué está pasando internamente mediante el estado.

Hay que tener en cuenta que el uso de *refs* **no dispara una reevaluación del componente**. React en su documentación recomienda que, en la mayor parte de los casos, utilicemos componentes controlados, de manera de que el código sea más limpio y trazable.

Desafío

Debemos agregar un botón a cada uno de los *BookItem* que permita eliminar el libro de la lista al seleccionarlo:

1. Agregar el botón a cada uno de los libros.
2. Al seleccionar el botón, se debe abrir un [modal de bootstrap](#) dentro de la carpeta ui o la carpeta *shared*, que le pregunte al usuario si desea realmente eliminarlo, junto con un botón de cancelar y de “Sí, deseo eliminarlo”.
 - a. Al seleccionar cancelar, el modal se esconde y el libro no es eliminado.
 - b. Al seleccionar “Sí, deseo eliminarlo”, el libro se elimina de la lista.

Fecha	Versionado actual	Autor	Observaciones
12/05/2023	1.0.0	Gabriel Golzman	Primera versión
12/01/2024	2.0.0	Gabriel Golzman	Segunda versión
16/01/2024	2.1.0	Gabriel Golzman	Agregado desafío y enrutado desplazado a su propio apunte
14/12/2024	3.0.0	Gabriel Golzman	Tercera versión