Universidad Tecnológica Nacional

Facultad Regional Rosario

Tecnicatura Universitaria en Programación

Programación III

Unidad 2.2

Listas dinámicas y renderizado condicional

Introducción

Los últimos dos conceptos esenciales básicos que debemos aprender son: el renderizado condicional de elementos, es decir, que los componentes se muestran en pantalla si se cumple o no cierta condición, y el renderizado de listas de elementos, para poder renderizar de manera dinámica dichas listas.

Renderizando listas con datos

Como tal vez se imaginen, el listado que poseemos actualmente de lecturas del usuario no es muy pragmático ¿Por qué es eso? Debido a que tenemos de manera *hardcodeada* los componentes "BookItem", por lo que si el arreglo cambia y en vez de ser 4 lecturas son 5, 6, 10, 1024, nuestra aplicación nunca refleja ese cambio.

Además, cuando el usuario agrega una lectura realizada a través de nuestro formulario, esa lectura no se agrega nunca al arreglo. ¿Cómo podemos hacer para simplificar entonces nuestro código?

Utilizando la función *map()*

A través de "App", ya estamos pasando todas las lecturas al componente "Books". Lo que debemos hacer ahora es, dentro del jsx que devolvemos, realizar la siguiente declaración:

```jsx
<div className="d-flex justify-content-center flex-wrap my-5">
    {books.map(book => (
        <BookItem
            key={book.id}
            title={book.title}
            author={book.author}
            rating={book.rating}
            pageCount={book.pageCount}
            imageUrl={book.imageUrl}
            available={book.available}
            onBookSelected={handleBookSelected}
        />
    ))}
</div>
```

Podemos poner las llaves dentro del jsx ya que ese código se va a resolver a una expresión. Por cada elemento declarado en el arreglo *books*, nos devolverá un componente "BookItem", conformando efectivamente el arreglo.

Luego podremos borrar todos los "BookItem" que habíamos declarado abajo, y comprobar que funcione la aplicación correctamente. Veremos en consola que aparece una *warning* pero eso lo corregiremos a continuación.

Conceptos claves en listas: la *prop key*

¿Por qué tenemos una advertencia en consola que nos pide que cada elemento posea una *prop* llamada *key*? React, para el renderizado de listas de forma dinámica y la correcta actualización de las mismas (es decir, para no encontrarse con *bugs* al momento de actualizarlas) utiliza la *prop key* que permite **identificar unívocamente a cada elemento de la lista** de manera que sea más sencillo llevar cuenta de que elementos han sido creados, modificados o eliminados en la lista.

No es recomendable usar el *index* del arreglo ya que eso puede llevar a errores al momento de actualizar las listas.

Nota: recordar agregar id's al arreglo inicial de libros en caso de que no lo tengan.

Agregando una nueva lectura

Para poder agregar una nueva lectura, vamos a importar la función *useState* de React en “App” y luego vamos a realizar el seteo inicial del estado con el valor de la constante *books*.

```
const [bookList,setBookList] = useState(books);
```

Para poder agregar una lectura nueva correctamente, primero deberemos guardar en una variable un arreglo con las lecturas anterior distribuido junto a la nueva lectura (que ya la obtenemos del componente hijo);

```
57    const handleBookAdded = (enteredBook) => {
58      const bookData = {
59        ...enteredBook,
60        id: Math.random()
61      }
62
63      setBookList(prevBookList => [bookData, ...prevBookList])
64    }
65
66    return (
67      <div className="d-flex flex-column align-items-center">
68        <h2>Book champions app</h2>
69        <p>¡Quiero leer libros!</p>
70        <NewBook onBookAdded={handleBookAdded} />
71        <Books books={bookList} />
72      </div>
73    )
74  }
```

Comprobamos en nuestra aplicación que todo funcione correctamente. La distribución la realizamos al revés (ponemos primero el libro nuevo y luego los antiguos) ya que así queremos mostrárselo al usuario, pero generalmente se realiza de manera inversa.

Además, notemos que como estamos actualizando el estado en base a un estado anterior, lo hacemos de manera completa, enviando una *callback* que recibe el valor anterior como parámetro.

Desafío

Ejercicio de clase

Vamos a agregar ahora un buscador de libros. El objetivo es agregar un FormControl de tipo texto que en cada ingreso de una letra, vaya filtrando los libros que recibe el componente Books.

Pasos:

1. Crear una carpeta llamada "bookSearch" en *components* y luego crear el componente BooksSearch con el siguiente código:

```jsx
import { Form } from "react-bootstrap"

const BookSearch = () => {
    return (
        <Form.Group className="mb-3" controlId="searchBook">
            <Form.Control
                type="text"
                placeholder="Buscar libro..."
            />
        </Form.Group>

    )
}

export default BookSearch;
```

2. Crear una función que escuche el cambio del *input.*
3. En el componente "Books", agregar en la parte superior, el componente BooksSearch.
4. Debemos lograr que el valor de la búsqueda suba hasta “Books”, y que setee un estado allí, es decir, cada vez que cambia el valor del filtro seleccionado, el estado guardado en "Books" debe cambiar.

Renderizado condicional

Ahora que tenemos nuestro buscador de lecturas, vemos que hay algunos casos en que no muestran ningún valor. Es una buena práctica informarle al usuario de alguna manera que en ese año **no se encontraron lecturas**.A su vez, si no ha seleccionado ninguna lectura no tiene mucho sentido mostrar el mensaje “Ha seleccionado el libro:” y dejarlo vacío.

Para ello, nos aprovecharemos del concepto de renderizado condicional.

El renderizado condicional es una forma de renderizar diferentes elementos en pantalla, según una o más condiciones establecidas.

Agregaremos otra expresión entre llaves dentro de nuestro jsx, arriba del filtrado de lecturas. Como bien sabemos, expresiones largas como *if*, *for*, y *while* no están permitidas dentro del jsx, pero podremos utilizar una expresión ternaria para saltearnos esta dificultad.

Entonces, deseamos mostrar en el caso que la longitud de lecturas filtradas sea 0, un elemento *p* que nos indique que no hay lecturas que coincidan con esa búsqueda. En el caso de que sí, utilizamos la lógica que veníamos aplicando. Eso se puede escribir de la siguiente manera:

```
{bookSelected && <p>El libro seleccionado es: <span className="fw-bold">{bookSelected}</span></p>}
```

Allí, para no mostrar el mensaje de libro seleccionado utilizamos el operador **&&**, que indica que si y sólo si la expresión del lado izquierdo **tiene valor o es verdadera**, se ejecutará (mostrará) lo de lado derecho.

Desafío

Vamos a seguir practicando listas y renderizado condicional. Vamos a convertir los arreglos de asteriscos en estrellas de puntuación para las lecturas, para ello:

1. Instalaremos react-bootstrap-icons.
2. Luego, utilizaremos los iconos Star (para las estrellas vacías) y StarFilled (para las estrellas completas).
3. El objetivo es:
    a. Mostrar primero las estrellas llenas (esto refiere a la longitud del arreglo *rating*)
    b. A continuación, mostrar las estrellas vacías (estas resultan de 5 menos la longitud del arreglo *rating*)
4. Utilizar si es posible la función *Array.from* y operadores ternarios.

Fecha	Versionado actual	Autor	Observaciones
24/04/2022	1.0.0	Gabriel Golzman	Primera versión
30/11/2023	2.0.0	Gabriel Golzman	Segunda versión
11/12/2024	3.0.0	Gabriel Golzman	Tercera Versión