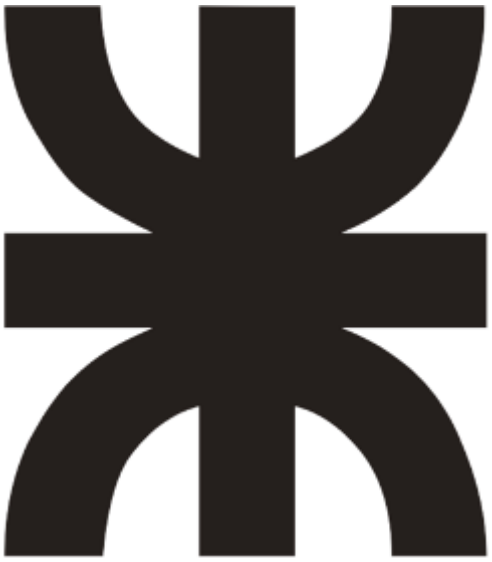
Universidad Tecnológica Nacional

Facultad Regional Rosario

Tecnicatura Universitaria en Programación



Programación III

Unidad 4.4

Optimizaciones

Introducción

En esta unidad nos concentraremos principalmente en todo lo relacionado a mejoras en performance del lado cliente (*frontend*) para así poder desarrollar una aplicación que no malgaste recursos de navegador, que recordemos es lo que a fin de cuentas aloja nuestra aplicación de React.

Estas optimizaciones se basan fuertemente en conceptos del ciclo de vida del componente (desde que este se evalúa, monta y renderiza en el navegador hasta que este se desmonta, pasando por sus sucesivas re evaluaciones), de manera que si el alumno no se siente lo suficiente capacitado en estos temas, puede releer los apuntes U2.1 y U4.1, referidos al manejo del estado y de los efectos secundarios.

Optimización mediante la memorización

La **memorización** (*memoization*) es una técnica que logra que las aplicaciones se vuelvan más eficientes, y por ende más rápidas (con mejor performance).Esto lo realiza guardando los resultados del cómputo en el caché y buscando esa información cuando realmente la necesita.

memo

const MemoizedComponent = memo(SomeComponent, arePropsEqual?)

Encerrar un componente en *memo* para obtener la versión memoizada de ese componente. La versión memoizada del componente no se va a re-evaluar cuando su componente padre sea re-evaluado siempre y **cuando sus *props* no cambien**.

Es importante mencionar que la misma documentación oficial de React no nos garantiza que no se vaya a realizar esta re-evaluación,ya que la memoización es una optimización de performance, no una garantía.

Parámetros:

- **Componente**: aquel componente que queremos memoizar.
- (*opcional*) *arePropsEqual?*: una función que acepta dos argumentos: las *props* previas del componente y las nuevas. Debería devolver verdadero si las *props* elegidas son iguales.y falso si no lo son, lo que va a especificar al componente si es necesaria su re evaluación.

Retorna:
El componente memoizado.

Podemos ver su aplicación en un componente sencillo de formulario:

```jsx
const App = () => {
  const [name, setName] = useState("");
  const [book, setBook] = useState("");

  const handleNameChange = (e) => {
    setName(e.target.value)
  }

  const handleBookChange = (e) => {
    setBook(e.target.value)
  }


  return (
    <>
      <label>
        Nombre{": "}
        <input value={name} onChange={handleNameChange} />
      </label>
      <label>
        Libro deseado{": "}
        <input value={book} onChange={handleBookChange} />
      </label>
      <Greeting name={name} />
```

```jsx
const Greeting = ({ name }) => {
  console.log("In Greeting");
  return <h2>Hola {name}, disfruta tu libro</h2>
};

export default Greeting;
```

Observamos que la consola registra cada vez que el componente se re-evalua, y este es re evaluado incluso si modificamos el input de *book* (el cuál Greeting no recibe por props y no le incumbe). Si lo modificaremos de esta forma:

```jsx
import { memo } from "react";

const Greeting = ({ name }) => {
    console.log("In Greeting");
    return <h2>Hola {name}, disfruta tu libro</h2>;
}

export default memo(Greeting)
```

El componente solo se re evalua en el caso que el valor de *name* sea modificado. Puede que necesitemos refrescar el proyecto para poder verlo correctamente.

useCallback

Supongamos que poseemos un componente Button de este estilo:

```
5   import { memo } from "react";
6
7   const Button = ({ onClick }) => {
8     console.log("In Button!");
9     return <button onClick={onClick}>¡Reservar!</button>;
10  };
11
12  export default memo(Button);
```

Y que posee un handler en App:

```
20    const handleClick = () => {
21      alert(`Libro ${book} reservado!`);
22    };
```

Resulta que , a pesar de agregarle memo, el mensaje en consola nos lo sigue mostrando. ¿Por qué sucede esto? Sucede que cuando React recibe por *props* una función (a modo de *callback*), en cada re-evaluación **recrea una nueva instancia de esa función**, diferente a la anterior.

Podemos evitar esto agregando un *useCallback*, un hook de React que le indica que no debe re crear esa función en cada re evaluación.

Toma dos parámetros (similar a *useEffect*), el primero es la función a memorizar y el segundo es un arreglo de dependencias que , al cambiar los valores guardados allí, la función se volverá a instanciar.

Vamos a aplicarlo a la función handleClick, qué es pasada a Button:

```
20    const handleClick = useCallback(() => {
21      alert(`Libro ${book} reservado!`);
22    }, [book]);
```

useMemo

useMemo es un hook de React que nos permite .memoizar **el resultado de una función o cálculo complejo**, de manera que evitemos Similar a *useCallback*, toma dos argumentos, la función a guardar y las dependencias que al cambiar, se ejecuta la anterior función.

```
const App = () => {
  const [count, setCount] = useState(0);
  const [todos, setTodos] = useState([]);
  const calculation = useMemo(() => expensiveCalculation(count), [count]);

  const increment = () => {
    setCount((c) => c + 1);
  };
  const addTodo = () => {
    setTodos((t) => [...t, "New Todo"]);
  };

  return (
    <div>
      <div>
        <h2>My Todos</h2>
        {todos.map((todo, index) => {
          return <p key={index}>{todo}</p>;
        })}
        <button onClick={addTodo}>Add Todo</button>
      </div>
      <hr />
      <div>
        Count: {count}
        <button onClick={increment}>+</button>
        <h2>Expensive Calculation</h2>
        {calculation}
      </div>
    </div>
  );
};


const expensiveCalculation = (num) => {
  console.log("Calculating...");
  for (let i = 0; i < 1000000000; i++) {
    num += 1;
  }
  return num;
};
```

En el ejemplo anterior, el cálculo complejo solo se realiza cuando el valor de *count* se actualiza, no cuando por ejemplo cambian los valores de arreglo de todos.

Fecha	Versionado actual	Autor	Observaciones
13/06/2023	1.0.0	Gabriel Golzman	Primera versión
29/01/2024	2.0.0	Gabriel Golzman	Segunda versión
12/05/2025	3.0.0	Gabriel Golzman	Tercera versión