****

MEMO, USEMEMO, USECALLBACK, USECONTEXT

****

En este proyecto vamos a poner en practica los hooks de useContext, useMemo y useCallback.

Los Hooks useMemo y useCallback son muy parecidos. Los dos sirven para guardar informacion de la aplicación y cuando tenga que utilizar esa informacion llamo al hook en donde lo guardé. El useMemo sirve para guardar un componente y el useCallback para guardar una funcion. No hay que utilizarlos en exceso si no en algunos casos concretos.

El hook useContext nos crea un estado de forma global, para poderlo utilizar en cualquir lugar de la aplicación (En cualquier componente).

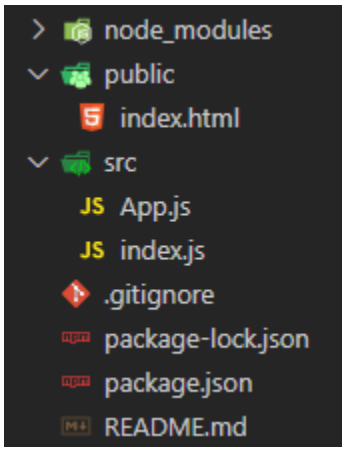
1. Creamos la aplicación

```
npx create-react-app nombreapp
```

2. Instalamos react-bootstrap

```
npm install --save react-bootstrap bootstrap
```

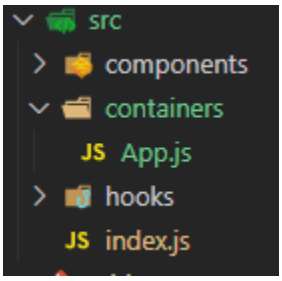
3. Borramos los archivos que no necesitamos y las conexiones de estos .



4. Abrimos la aplicación en el navegador

```
npm start
```

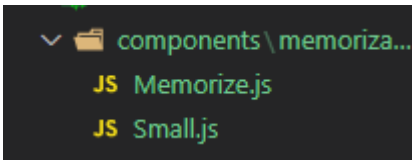
5. En src creamos la estructura de carpetas a utilizar . Creamos la carpeta de components, container y hooks. El archivo App.js lo movemos a la carpeta container y en el index actualizamos la importacion del componente



Archivo index.js:

```
import React from "react";
import ReactDOM from "react-dom";
import App from "./containers/App";

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById("root")
);
```

6. En la carpeta components creamos una carpeta llamada memorizacion. En esta carpeta vamos a crear los hooks que sirven para memorizar o guardar. Creamos dos archivos uno llamado Memorize.js y otro Small.js. A ambos archivos le damos la estructura con rafce



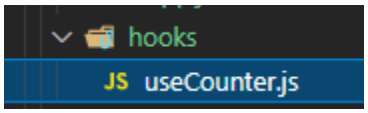
Archivo Memorize.js:

```
import React from 'react'

const Memorize = () => {
    return (
        <div>

        </div>
    )
}

export default Memorize
```

Archivo Small.js

```
import React from 'react'

const Small = () => {
    return (
        <div>

        </div>
    )
}

export default Small
```

7. Vamos a la carpeta hooks y creamos un archivo llamado useCounter.js. En este archivo vamos a crear funciones para sumar y restar. Creamos un estado inicial con el valor de 10 y creo las funciones anteriormente mencionadas. Le damos como estructura inicial rafce.El useCounter es un contador que nos permite sumar o restar.




```
import { useState } from "react";

const useCounter = (value = 10) => {
    const [counter, setCounter] = useState(value);

    const handleSumar = () => {
        setCounter(counter + 1);
    };
    const handleRestar = () => {
        setCounter(counter - 1);
    };
    const handleReset = () => {
        setCounter(0);
    };
    return { counter, handleSumar, handleReset, handleRestar };
};

export default useCounter;
```

8. Vamos al archivo Memorize.js y hay vamos a llamar a la funcion useCounter que creamos. Y desestructuramos solo las funciones que necesitamos. En este caso el counter y el handleSumar Vamos al return y lo pintamos.

```
import React from "react";
import useCounter from "../../hooks/useCounter";
import { Button } from "react-bootstrap";
```

```
const Memorize = () => {
    const { counter, handleSumar } = useCounter();
    return (
        <div>
            <h1>Contador: {counter}</h1>
            <Button onClick={handleSumar}>+1</Button>
        </div>
    );
};

export default Memorize;
```

9. Vamos al archivo App.js y pintamos el componente Memorize.js. No olvidar poner la linea de bootstrap css para que nos muestre los estilos de react-bootstrap

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

```
import "bootstrap/dist/css/bootstrap.min.css";
import Memorize from "../components/memorizacion/Memorize";

function App() {
  return (
    <div>
      <Memorize />
    </div>
  );
}

export default App;
```

10. Vamos al archivo Memorize.js y en vez de pasar el estado directamente se lo vamos a pasar como propiedad al component hijo que seria Small

```
import React from "react";
import useCounter from "../../hooks/useCounter";
```

```
import { Button } from "react-bootstrap";
import Small from "./Small";

const Memorize = () => {
    const { counter, handleSumar } = useCounter();
    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            <Button onClick={handleSumar}>+1</Button>
        </div>
    );
};

export default Memorize;
```

11. Vamos al contenerdor hijo que en este caso seria Small.js y desestructuramos la propiedad que le mandamos de Memorize.js. En este caso seria value. Pintamos value en el return

```
import React from "react";

const Small = ({ value }) => {
    return (
        <div>
            <small>{value}</small>
        </div>
    );
};

export default Small;
```

12. Creamos un estado para show y se lo asignamos a un boton.

```
import React, { useState } from "react";
import useCounter from "../../hooks/useCounter";
```

```
import { Button } from "react-bootstrap";
import Small from "./Small";

const Memorize = () => {
    const { counter, handleSumar } = useCounter();
    const [show, setShow] = useState(true);
    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            <Button onClick={handleSumar}>+1</Button>
            <Button onClick={() => setShow(!show)}>
                Show/Hide {JSON.stringify(show)}
            </Button>
        </div>
    );
};

export default Memorize;
```

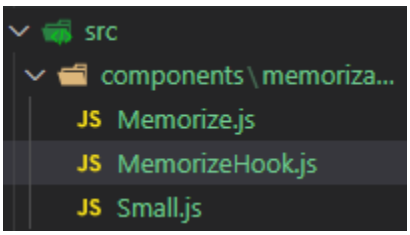
Lo que esta pasando en estos momentos es que cada vez que hay un cambio en el boton show/hidden se renderiza otra vez el componente hijo que en este caso es Small. Esto no es recomendable ya que si por ejemplo tenemos un cambio en un componente con una data de 1000 personas, cada vez que ocurra un cambio se va a renderizar el componente con toda la data.

Para solucionar ese problema de que se renderize los componentes hijos cada vez que hay un cambio en el componente padre, vamos a envolver todo el componente hijo en este caso Small.js en un memo. De esta forma estamos guardando en memo todo el componente Small.js

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

const Small = memo(({ value }) => {
```

```
    console.log("Me volvi a renderizar");
    return (
        <div>
            <small>{value}</small>
        </div>
    );
});

export default Small;
```

13. En la carpeta memorizacion creamos un archivo llamado MemorizeHook.js y esta va a ser una copia del Memorize.js

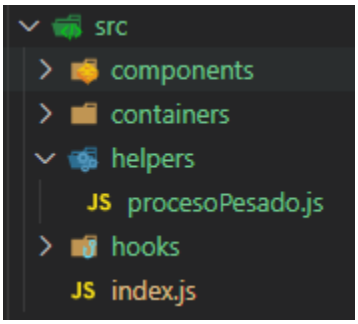



```
import React, { useState } from "react";
import { Button } from "react-bootstrap";
import useCounter from "../../hooks/useCounter";
import Small from "./Small";

const MemorizeHook = () => {
    const { counter, handleSumar } = useCounter();
    const [show, setShow] = useState(true);
    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            <Button onClick={handleSumar}>+1</Button>
            <Button onClick={() => setShow(!show)}>
                Show/Hide {JSON.stringify(show)}
            </Button>
        </div>
    );
};
```

```
export default MemorizeHook;
```

En este archivo vamos a utilizar el hooks de useMemo. Este se utiliza para procesos que sean muy pesados.

14. En src creamos una carpeta llamada helpers. Y es en esta carpeta donde vamos a guardar los procesos pesados. Dentro de este creamos un archivo llamado procesoPesado.js



Este archivo va a contener una funcion flecha la cual recibe como parametro un numero de veces que quiero que se imprima la data que hay en el for. Si escribimos 5 se va a imprimir 5 veces el ahí vamos.

```
export const procesoPesado = ( iteraciones ) => {
    for (let i = 0; i < iteraciones; i++) {
        console.log("ahi vamos...");
    }
    return `${iteraciones} iteraciones realizadas`;
};
```

15. Vamos al archivo MemorizeHook.js y llamamos la funcion proceso pesado y despues pintamos este componente en el container principal en App.js

Archivo MemorizeHook.js

```
import React, { useState } from "react";
import { Button } from "react-bootstrap";
import { procesoPesado } from "../../helpers/procesoPesado";
import useCounter from "../../hooks/useCounter";
import Small from "./Small";

const MemorizeHook = () => {
    const { counter, handleSumar } = useCounter();
    const [show, setShow] = useState(true);
    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            <p>{procesoPesado(10)}</p>
            <Button onClick={handleSumar}>+1</Button>
            <Button onClick={() => setShow(!show)}>
                Show/Hide {JSON.stringify(show)}
            </Button>
        </div>
    );
};

export default MemorizeHook;
```

Archivo App.js:

```
import "bootstrap/dist/css/bootstrap.min.css";
import MemorizeHook from "../components/memorizacion/MemorizeHook";
// import Memorize from "../components/memorizacion/Memorize";

function App() {
  return (
    <div>
      <MemorizeHook />
    </div>
  );
}

export default App;
```

16. En el archivo MemorizeHook vamos a utilizar el hook useMemo para que se ejecute esos procesos pesados cuando haya un cambio en el estado inicial

```
useMemo(() => procesoPesado(40), [counter]);
```

Cada vez que haya un cambio en counter, se va a ejecutar la funcion procesoPesado

```
import React, { useMemo, useState } from "react";
import { Button } from "react-bootstrap";
import { procesoPesado } from "../../helpers/procesoPesado";
import useCounter from "../../hooks/useCounter";
import Small from "./Small";

const MemorizeHook = () => {
    const { counter, handleSumar } = useCounter();
    const [show, setShow] = useState(true);

    useMemo(() => procesoPesado(40), [counter]);

    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            {/* <p>{procesoPesado(10)}</p> */}
            <Button onClick={handleSumar}>+1</Button>
            <Button onClick={() => setShow(!show)}>
                Show/Hide {JSON.stringify(show)}
            </Button>
        </div>
    );
};

export default MemorizeHook;
```

El useMemo lo guardo en una variable y la llamo en el return

```
import React, { useMemo, useState } from "react";
import { Button } from "react-bootstrap";
import { procesoPesado } from "../../helpers/procesoPesado";
import useCounter from "../../hooks/useCounter";
import Small from "./Small";

const MemorizeHook = () => {
    const { counter, handleSumar } = useCounter();
    const [show, setShow] = useState(true);

    const memorizacion = useMemo(() => procesoPesado(40), [counter]);

    return (
        <div>
            <h1>
                Contador:
                <Small value={counter} />
            </h1>
            <p>{memorizacion}</p>
            <Button onClick={handleSumar}>+1</Button>
            <Button onClick={() => setShow(!show)}>
                Show/Hide {JSON.stringify(show)}
            </Button>
        </div>
    );
};

export default MemorizeHook;
```

17. En la carpeta memorizacion creo dos archivos uno llamado CallBackHook.js y otro ShowIncrement.js. Le damos estructura a ambos con rafce. En el archivo CallBackHook.js vamos crear un estado inicial que inicia en 10 y creamos una funcion increment que recibe como parametro un num, y esa funcion se la pasamos a nuestro contenedor hijo como propiedad, que en este caso seria ShowIncrement el hijo.

CallBackHook.js:

```
import React, { useState } from "react";
import ShowIncrements from "./ShowIncrements";

const CallBackHook = () => {
    const [counter, setCounter] = useState(10);

    const increment = (num) => {
        setCounter(counter + num);
    };
    return (
        <div>
            <h1>UseCallBack: {counter}</h1>
            <ShowIncrements increment={increment} />
        </div>
    );
};

export default CallBackHook;
```

ShowIncrements.js:

```
import React from "react";
import { Button } from "react-bootstrap";

const ShowIncrements = ({ increment }) => {
    return (
        <div>
            <Button onClick={() => increment(5)}>Incrementar</Button>
        </div>
    );
};

export default ShowIncrements;
```

Vamos al archivo App.js y pintamos el CallBackHook.js

```
import "bootstrap/dist/css/bootstrap.min.css";
import CallBackHook from "../components/memorizacion/CallBackHook";
// import MemorizeHook from "../components/memorizacion/MemorizeHook";
```

```
// import Memorize from "../components/memorizacion/Memorize";

function App() {
  return (
    <div>
      <CallBackHook />
    </div>
  );
}

export default App;
```

18. Vamos al archivo useCallbackHooks.js y reemplazamos la funcion increment y utilizamos el hooks useCallBack que nos sirve para guardar funciones. La funcion se va a ejecutar cada vez que cambio el estado inicial . Para esto le pasamos la funcion que cambia el estado, y para que no haya dependencia de estado al setCounter le agregamos una variable, que en este caso es c

```
import React, { useCallback, useState } from "react";
import ShowIncrements from "./ShowIncrements";

const CallBackHook = () => {
    const [counter, setCounter] = useState(10);
    console.log("Me volvi a renderizar");

    const increment = useCallback(
        (num) => {
            setCounter((c) => c + num);
        },
        [setCounter]
    );

    // const increment = (num) => {
    //     setCounter(counter + num);
    // };
    return (
        <div>
            <h1>UseCallBack: {counter}</h1>
```

```
            <ShowIncrements increment={increment} />
        </div>
    );
};

export default CallBackHook;
```

19. El ShowIncrement.js lo guardamos en memo

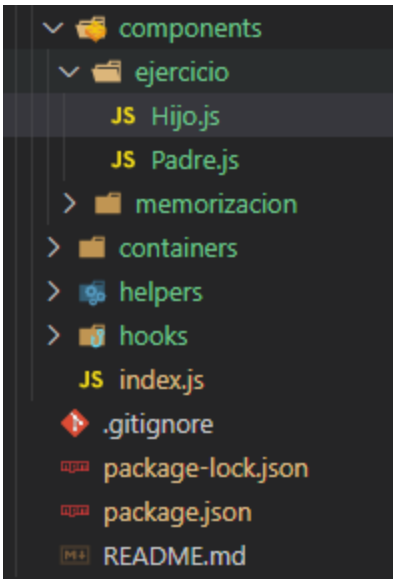
```
import React, { memo } from "react";
import { Button } from "react-bootstrap";

const ShowIncrements = memo(({ increment }) => {
    console.log("Me volvi a renderizar");
    return (
        <div>
            <Button onClick={() => increment(5)}>Incrementar</Button>
        </div>
    );
});

export default ShowIncrements;
```

****

EJERCICIO MEMORIZACION

****

1. En la carpeta de components creamos una carpeta llamada ejercicio. Y en esta creamos dos archivos, uno llamado Padre.js e Hijo.js. Les damos estructura con rafc



2. Vamos al archivo Padre.js, este va a contener una variable con un arreglo de numeros, despues creo el estado inicial que es 0, y creo una funcion para incrementar el estado. Voy al return y pinto la data de numeros, llamo al contenedor hijo y le paso la propiedad incrementar y numero.

```
import React, { useState } from "react";
import { Hijo } from "./Hijo";

export const Padre = () => {
    const numeros = [2, 4, 6, 8, 10];
    const [valor, setValor] = useState(0);

    const incrementar = (num) => {
        setValor(valor + num);
    };
    return (
        <div>
            <h1>Padre</h1>
            <p>Total:{valor}</p>
            <hr />

            {numeros.map((n) => (
                <Hijo key={n} numeros={n} incrementar={incrementar} />
            ))}
        </div>
    );
```

```
};
```

3. En el archivo Hijo.js, desestructuro la info que recibo del padre y la pinto en return

```
import React from "react";

export const Hijo = ({ numeros, incrementar }) => {
    console.log("Me volvi a generar :(");
    return (
        <button
            className="btn btn-primary mr-3"
            onClick={() => incrementar(numeros)}
        >
            {numeros}
        </button>
    );
};
```

4. Voy al archivo App.js y pinto el contenedor Padre.js

```
import "bootstrap/dist/css/bootstrap.min.css";
import { Padre } from "../components/ejercicio/Padre";
// import CallBackHook from "../components/memorizacion/CallBackHook";
// import MemorizeHook from "../components/memorizacion/MemorizeHook";
// import Memorize from "../components/memorizacion/Memorize";

function App() {
  return (
    <div>
      <Padre />
    </div>
  );
}

export default App;
```

En estos momentos, cada vez que le doy en un boton de la aplicación, se esta renderizando los 5 numeros, para que esto no ocurra, primero guardamos el componente hijo con use memo. Despues creamos una constante y llamamos al useCallBack, en este guardamos la funcion que quiero que se ejecute cada vez que cambia el estado original. Los archivos Padre.js e Hijo.js nos quedan:

```
import React, { useCallback, useState } from "react";
import { Hijo } from "./Hijo";

export const Padre = () => {
    const numeros = [2, 4, 6, 8, 10];
    const [valor, setValor] = useState(0);

    const incrementar = useCallback(
        (num) => {
            setValor((valor) => valor + num);
        },
        [setValor]
    );

    // const incrementar = (num) => {
    //     setValor(valor + num);
    // };
    return (
        <div>
            <h1>Padre</h1>
            <p>Total:{valor}</p>
            <hr />

            {numeros.map((n) => (
                <Hijo key={n} numeros={n} incrementar={incrementar} />
            ))}
            {/*<Hijo />*/}
        </div>
    );
};
```

Hijo.js:

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

export const Hijo = memo(({ numeros, incrementar }) => {
    console.log("Me volvi a generar :(");
    return (
        <button
            className="btn btn-primary mr-3"
            onClick={() => incrementar(numeros)}
        >
            {numeros}
        </button>
    );
});
```

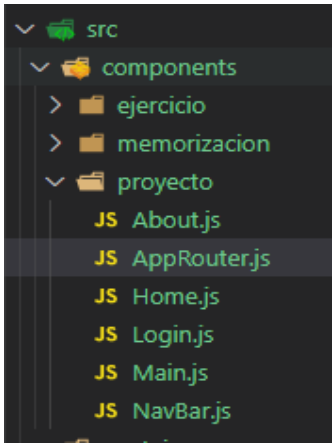
****

EJERCICIO USECONTEXT

****

1. Vamos a la carpeta de components y creamos una carpeta llamada proyecto.

   Dentro de esta creamos seis archivos llamados AppRouter.js, Login.js, Home.js, NavBar.js, Main.js, About.js



2. Instalamos react-router-dom

```
npm install react-router-dom
```

3. A los archivos creados les damos estructura con rafce, y el componente Main.js lo vamos a pintar en el archivo App.js

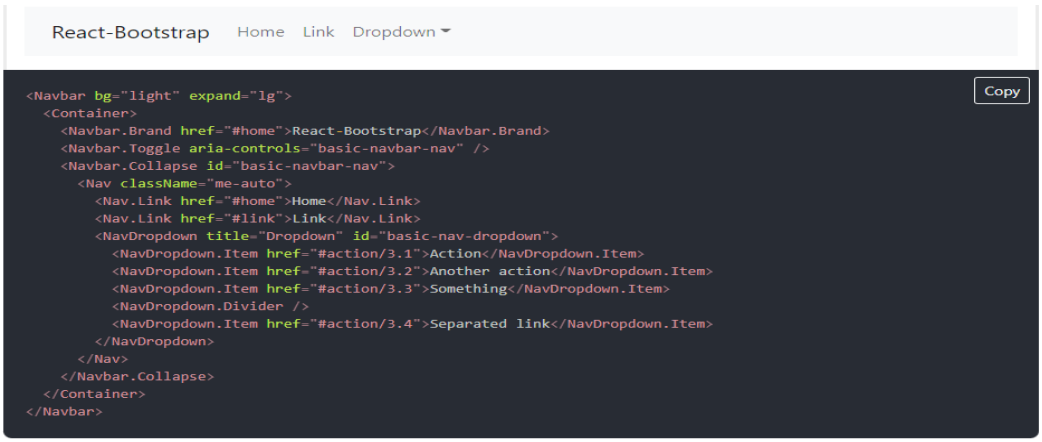
   App.js:

```
import "bootstrap/dist/css/bootstrap.min.css";
import Main from "../components/proyecto/Main";
// import { Padre } from "../components/ejercicio/Padre";
// import CallBackHook from "../components/memorizacion/CallBackHook";
// import MemorizeHook from "../components/memorizacion/MemorizeHook";
// import Memorize from "../components/memorizacion/Memorize";

function App() {
  return (
    <div>
      <Main />
    </div>
  );
}

export default App;
```

4. Vamos a la pagina de react-bootstrap y buscamos un nav a utilizar

   https://react-bootstrap.netlify.app/components/navbar/#navbars

   Para este caso, nos vamos a traer el que nos muestra de primero



Vamos al archivo NavBar.js y pegamos el nav que hemos seleccionado

```
import React from "react";
import { Container, Nav } from "react-bootstrap";

const NavBar = () => {
    return (
        <Navbar bg="light" expand="lg">
            <Container>
                <Navbar.Brand href="#home">Home</Navbar.Brand>
                <Navbar.Toggle aria-controls="basic-navbar-nav" />
                <Navbar.Collapse id="basic-navbar-nav">
                    <Nav className="me-auto">
                        <Nav.Link href="#home">Home</Nav.Link>
                        <Nav.Link href="#link">Link</Nav.Link>
                    </Nav>
                </Navbar.Collapse>
            </Container>
        </Navbar>
    );
};

export default NavBar;
```

5. Vamos al archivo AppRouter y especificamos las rutas que va a tener nuestra pagina. En la ultima ruta tenemos un * y el Navigate. Esto quiere decir que si el usuario escribe una ruta que no este especificada, me lleve a la ruta /.

```
import React from "react";
import { BrowserRouter, Navigate, Route, Routes } from "react-router-dom";
import About from "./About";
import Home from "./Home";
import Login from "./Login";
import NavBar from "./NavBar";

const AppRouter = () => {
    return (
        <BrowserRouter>
            <NavBar />
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/login" element={<Login />} />
                <Route path="/about" element={<About />} />
                <Route path="*" element={<Navigate to="/" />} />
            </Routes>
        </BrowserRouter>
    );
};

export default AppRouter;
```

6. Vamos al archivo NavBar.js y los Nav.Link que nos trae bootstrap le pasamos el parametro as para que se comporten como los Link de react router dom. Reemplazamos los href por los to del Link.

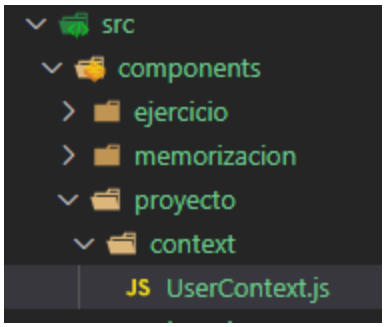
```
import React from "react";
import { Container, Nav, Navbar } from "react-bootstrap";
import { Link } from "react-router-dom";

const NavBar = () => {
    return (
        <Navbar bg="light" expand="lg">
            <Container>
                <Navbar.Brand as={Link} to="/home">
                    Home
                </Navbar.Brand>
                <Navbar.Toggle aria-controls="basic-navbar-nav" />
                <Navbar.Collapse id="basic-navbar-nav">
                    <Nav className="me-auto">
```

```
                        <Nav.Link as={Link} to="/home">
                            Home
                        </Nav.Link>
                        <Nav.Link as={Link} to="/login">
                            Login
                        </Nav.Link>
                        <Nav.Link as={Link} to="/about">
                            About
                        </Nav.Link>
                    </Nav>
                </Navbar.Collapse>
            </Container>
        </Navbar>
    );
};

export default NavBar;
```

7. Vamos al Main.js y pintamos el AppRouter.js

```
import React from "react";
import AppRouter from "./AppRouter";

const Main = () => {
    return (
        <div>
            <AppRouter />
        </div>
    );
};

export default Main;
```

8. En la carpeta proyecto creamos otra carpeta llamada contex y en esta creamos un archivo llamado UserContext.js



En el archivo UserContext.js creamos una variable con el hook createContext. Esta nos va a crear un estado inicial que se pueda pasar a cualquier componente de la aplicación sin necesidad de utilizar props.

```
import { createContext } from "react";

export const UserContext = createContext(null);
```

9. Vamos al archivo Main.js que es el contenedor padre y hay vamos a crear un estado inicial con un objeto con unos parametros. Ese estado inicial se lo queremos pasar a todo nuestro componente AppRouter.js. Para esto reemplazamos la etiqueta div, por el UserContext.js que creamos y le pasamos .Provider y le pasamos ese estado inicial que va a tener todos los contenedores de AppRouter a traves del value. Ya esos estados estan en todos los componentes de nuestra aplicación.

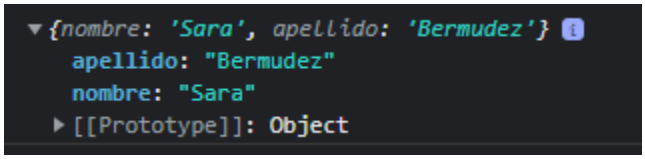
```
import React, { useState } from "react";
import AppRouter from "./AppRouter";
import { UserContext } from "./context/UserContext";

const Main = () => {
    const [user, setUser] = useState({
        nombre: "Sara",
        apellido: "Bermudez",
    });
```

```
    return (
        <UserContext.Provider value={{ user, setUser }}>
            <AppRouter />
        </UserContext.Provider>
    );
};

export default Main;
```

10. Para obtener esos estados iniciales en los componentes hijos lo hacemos de la siguiente manera:

Vamos al componente hijo donde queremos utilizar esos estados, es nuestro caso queremos utilizarlos en About.js. vamos al archivo y con el hook useContext los obtenemos. Desestructuramos los estados iniciales que queremos utilizar y dentro del parentesis del useContext ponemos en que contexto se encuentran esas propiedades, en nuestro caso se encuentran en la funcion userContext que creamos. Imprimos que lo que nos trae, como podemos ver en un json con el estado inicial.

```
import React, { useContext } from "react";
import { UserContext } from "./context/UserContext";

const About = () => {
    const { user, setUser } = useContext(UserContext);
    console.log(user);
    return <div></div>;
};

export default About;
```



Pintamos lo que nos trae user. Para eso con la propiedad JSON.stringify convertimos ese objeto que nos trae en un string.

```
import React, { useContext } from "react";
import { UserContext } from "./context/UserContext";

const About = () => {
    const { user, setUser } = useContext(UserContext);
    console.log(user);
    return <div>{JSON.stringify(user)}</div>;
};

export default About;
```

Desestructuramos la info para pintarla en return y creamos un boton con un evento onClick para que cada vez que se le de en el boton me cambie el estado inicial por el nuevo estado que cree

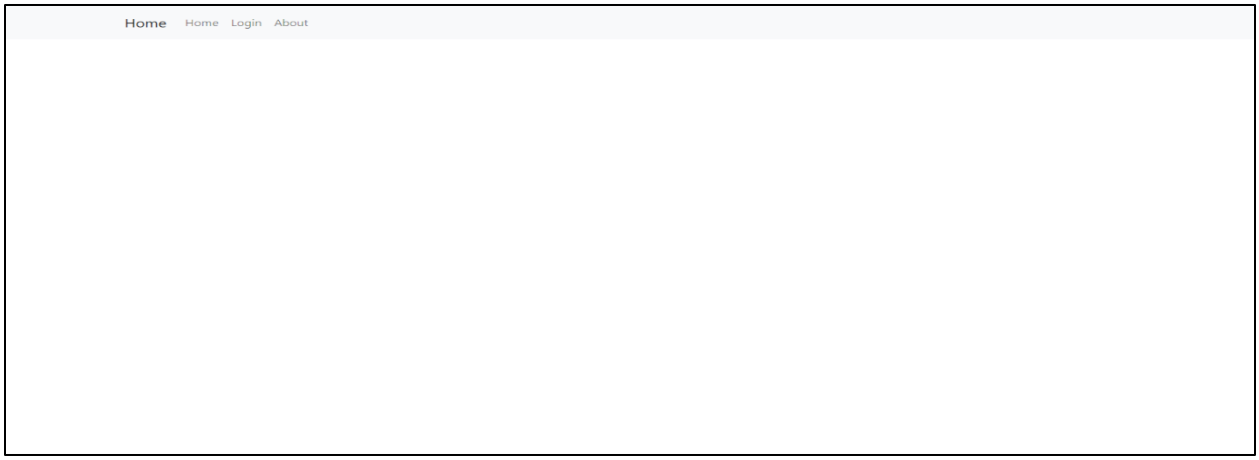
```
import React, { useContext } from "react";
import { Button } from "react-bootstrap";
import { UserContext } from "./context/UserContext";

const About = () => {
    const { user, setUser } = useContext(UserContext);
    console.log(user);
    return (
        <div>
            {JSON.stringify(user)}
            <h1>{user.nombre}</h1>
            <h1>{user.apellido}</h1>

            <Button
                onClick={() =>
                    setUser({
                        nombre: "Pepito",
                        apellido: "Perez",
                    })
                }
            >
```

```
                Cambiar estado
            </Button>
        </div>
    );
};

export default About;
```

11. Ahora vamos a simular un formulario de logeo. Para esto vamos al archivo Login.js y creamos nuestro estado inicial con las propiedades vacias. Llamamos al estado inicial que viene del useContext

```
import React, { useContext, useState } from "react";
import { UserContext } from "./context/UserContext";
import { Button } from "react-bootstrap";

const Login = () => {
    const [form, setForm] = useState({
        nombre: "Susana",
        apellido: "Torres",
    });

    const { user, setUser } = useContext(UserContext);
    return (
        <div>
            <h1>Login</h1>
            <Button onClick={() => setUser(form)}>Login</Button>
        </div>
    );
};

export default Login;
```

Link pagina: https://use-context.netlify.app/


Home
Home
Login
About


Home
Home
Login
About
Login
Login


Home
Home
Login
About
{"nombre":"Susana","apellido":"Torres"}
Susana
Torres
Cambiar estado