REACT

Comunicación entre componentes

0.- Introducción

Este documento trata sobre la comunicación entre componentes.

Primeramente trataremos la comunicación de de Padre a Hijos a través de los “props”.

1.- Comunicación de padre a hijo - PROPS

Crearemos un componente padre y otro hijo.

Para pasar datos de uno a otro se hace a través de la llamada al componente:

```
<ComponenteHijo nombre:”David” apellido=”Betancor” />
```

Y el componente hijo lo recoge en los props:

```
const ComponenteHijo = (props) => {
  return (
      <h2>Mi nombre es {props.nombre}</h2>
      <h2>Mi apellido es {props.apellido}</h2>
  ); };
```

Ejemplo:

ComponentePadre.js

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

const ComponentePadre = () => {

  const apellido = "Betancor";

  const ficha_medica={
    altura:"1.80",
    grupo_sanguineo:"A+",
    salud:"Buena"
  }

  return (
    <div>
      <h2>Mando datos del ComponentePadre</h2>
      <h3>Envío: nombre, apellido y ficha médica</h3>
      <hr/>
      <ComponenteHijo
        nombre="David"
        apellido={apellido}
        ficha={ficha_medica}/>
    </div>
  );
};

export default ComponentePadre;
```

ComponenteHijo.js

```jsx
import React from "react";

const ComponenteHijo = (props) => {
  return (
    <div className="tercer-componente">
      <h2>Recivo datos del ComponentePadre</h2>
      <ul>
        <li>Nombre: {props.nombre}</li>
        <li>Apellido: {props.apellido}</li>
        <li>Altura: {props.ficha.altura}</li>
        <li>Grupo sanguíneo: {props.ficha.grupo_sanguineo}</li>
        <li>Salud: {props.ficha.salud}</li>
      </ul>
    </div>
  );
};

export default ComponenteHijo;
```

1.1.- Desestructuración de PROPS

En el componente ComponenteHijo.js yo puedo desestructurar los props y realizarlo como:

```
const ComponenteHijo = ({nombre, apellido, ficha}) => {
  return (
```

Y accede a ellos como:

```
<li>Nombre: {nombre}</li>
<li>Apellido: {apellido}</li>
<li>Altura: {ficha.altura}</li>
```

App.js

```
import './App.css';
import ComponentePadre from './components/ComponentePadre';

function App() {
  return (
    <div className="App">
      <ComponentePadre/>
    </div>
  );
}

export default App;
```

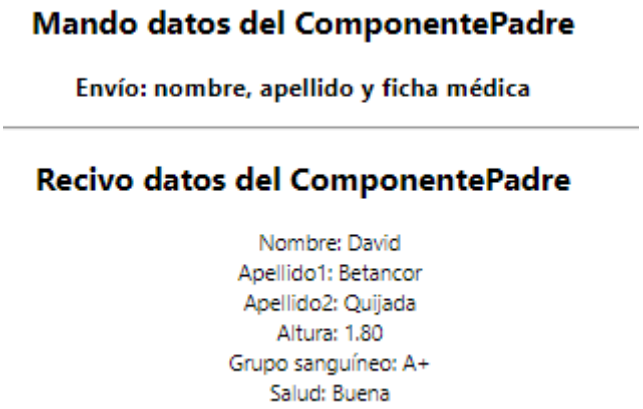
1.2.- Parámetros por defecto en las PROPS

En el componente hijo damos un valor al elemento que queremos por defecto.

En el ejemplo pasamos apellido2 por defecto “Quijada”.

```
const ComponenteHijo = ({nombre, apellido1, apellido2="Quijada",
ficha}) => {
  return (
    <div className="tercer-componente">
      <h2>Recivo datos del ComponentePadre</h2>
      <ul>
        <li>Nombre: {nombre}</li>
        <li>Apellido1: {apellido1}</li>
        <li>Apellido2: {apellido2}</li>
        <li>Altura: {ficha.altura}</li>
        <li>Grupo sanguíneo: {ficha.grupo_sanguineo}</li>
        <li>Salud: {ficha.salud}</li>
      </ul>
    </div>
  );
};
```

La salida es:



2.- Comunicación de hijo a padre - Props

La forma más sencilla es que el Padre posea un método que se le pase al Hijo mediante Props y este modifique. Veámoslo paso a paso:

En el ComponenPadre.js añado un método a los props que paso al ComponenteHijo.js, en el ejemplo el método es “handleChildClik”.

```
      <ComponenteHijo
        nombre="David"
        apellido1={apellido}
        ficha={ficha_medica}
        onChildClick={handleChildClick}/>
```

El método en el ComponentPadre se define:

```
  const handleChildClick = (newMessage) => {
    setMessage(newMessage);
  };
```

“setMessage” es la variable de un useState definido:

```
const [message, setMessage] = useState('');
```

Hasta aqui se intuye que cuando el Hijo llame a ese método, en realidad está llamando al método en el Padre que ejectará el cçodigo del método.

Ahora veamos al ComponenteHijo.js

El componente recibe:

```
  const ComponenteHijo = ({nombre, apellido1, apellido2="Quijada",
ficha, onChildClick}) => {
```

Nótese que se recibe el método onClidClick.

Ahora en el componente aAñado un botón que llamará a un método “handleClik:

```
<li><button onClick={handleClick}>¡Pulsame!</button></li>
```

Y el método se define como:

```
  const handleClick = () => {
    onChildClick('¡Hola desde el Hijo!');
  };
```

Que justamente está llamado al método que se le ha pasado por los props y le paso un mesaje que será recibido por el Padre

```
      <h2>Mensaje desde el Hijo: {message}</h2>
```

¡Y listo! Comunicación conseguida.

Dejo los códigos entero:

ComponentePadre.js

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

const ComponentePadre = () => {

  const [message, setMessage] = useState('');

  const apellido = "Betancor";

  const ficha_medica={
    altura:"1.80",
    grupo_sanguineo:"A+",
```

```
    salud:"Buena"
  }

  // Para que el Hijo se comunique con el Padre


  const handleChildClick = (newMessage) => {
    setMessage(newMessage);
  };

  return (
    <div>
      <h2>Mando datos del ComponentePadre</h2>
      <h3>Envío: nombre, apellido y ficha médica</h3>
      <hr/>
      <ComponenteHijo
        nombre="David"
        apellido1={apellido}
        ficha={ficha_medica}
        onChildClick={handleChildClick}/>
      <hr/>
      <h2>Mensaje desde el Hijo: {message}</h2>
    </div>
  );
};

export default ComponentePadre;
```

ComponenteHijo.js

```
import React from "react";

  const ComponenteHijo = ({nombre, apellido1, apellido2="Quijada",
ficha, onChildClick}) => {

  const handleClick = () => {
    onChildClick('¡Hola desde el Hijo!');
  };
```

```
  return (
    <div className="tercer-componente">
      <h2>Recivo datos del ComponentePadre</h2>
      <ul>
        <li>Nombre: {nombre}</li>
        <li>Apellido1: {apellido1}</li>
        <li>Apellido2: {apellido2}</li>
        <li>Altura: {ficha.altura}</li>
        <li>Grupo sanguíneo: {ficha.grupo_sanguineo}</li>
        <li>Salud: {ficha.salud}</li>
        <li><button onClick={handleClick}>¡Pulsame!</button></li>
      </ul>
    </div>
  );
};

export default ComponenteHijo;
```

App.js

```
import './App.css';
import ComponentePadre from './components/ComponentePadre';

function App() {
  return (
    <div className="App">
      <ComponentePadre/>
    </div>
  );
}

export default App;
```

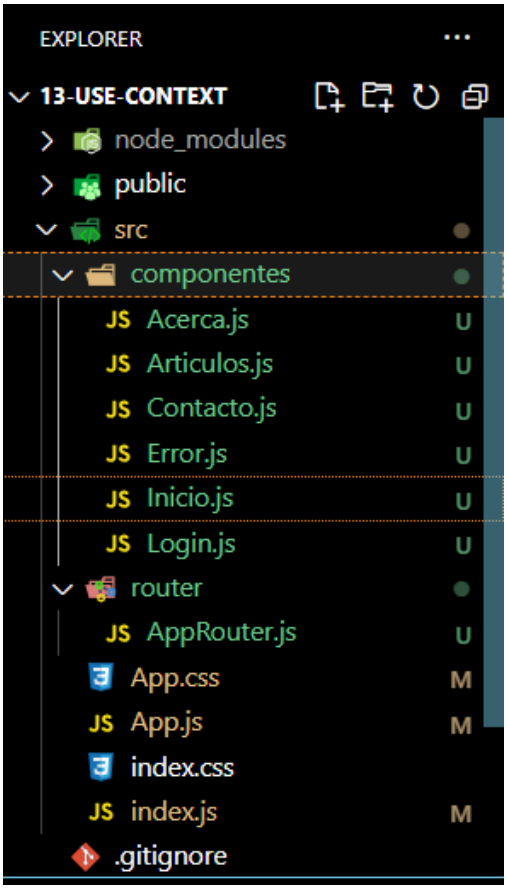
NOTA: pero esto no es muy efectivo si queremos pasar datos de un componente a varios y a distintos niveles del árbol de componentes. La solución es useContext. Lo vemos a continuación.

3.- Comunicación de hijo a padre - useContext

UseContext se apoya en el routing que hace React.

Creamos la estructura del proyecto primeramente:

3.1.- Estructura de carpetas y archivos del proyecto



3.2.- Código de ficheros principales

index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
```

```
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <App />
);
```

App.js

```
import './App.css';
import AppRouter from './router/AppRouter';

function App() {
  return (
    <div className="App">
      <AppRouter />
    </div>
  );
}

export default App;
```

AppRouter.js

```
import React from "react";
import { Routes, Route, NavLink, BrowserRouter } from
"react-router-dom";
import Inicio from "../componentes/Inicio";
import Login from "../componentes/Login";
import Articulos from "../componentes/Articulos";
import Acerca from "../componentes/Acerca";
import Contacto from "../componentes/Contacto";

const AppRouter = () => {
  return (
    <BrowserRouter>
      {/* MENU NAVEGACION */}
```

```
      <header className="header">
        <nav>
          <div className="logo">
            <h2>Aprendiendo useContext</h2>
          </div>
          <ul>
            <li>
              <NavLink to="/">Incio</NavLink>
            </li>
            <li>
              <NavLink to="/articulos">Articulos</NavLink>
            </li>
            <li>
              <NavLink to="/acerca">Acerca de</NavLink>
            </li>
            <li>
              <NavLink to="/contacto">Contacto</NavLink>
            </li>
            <li>
              <NavLink to="/login">Identifícate</NavLink>
            </li>
          </ul>
        </nav>
      </header>

      <section className="content">
        {/* CONFIGURAR RUTAS */}

        <Routes>
          <Route path="/" element={<Inicio />} />
          <Route path="/inicio" element={<Inicio />} />
          <Route path="/articulos" element={<Articulos />} />
          <Route path="/login" element={<Login />}></Route>
          <Route path="/contacto" element={<Contacto />} />
          <Route path="/acerca" element={<Acerca />} />
        </Routes>
      </section>
    </BrowserRouter>
  );
};
```

```
export default AppRouter;
```

Cada componente tiene una estructura como:

```
import React from 'react'

const Acerca = () => {
  return (
    <div>
      <h1>PAGINA DE ACERCA</h1>
      <p>Contenido de acerca de.</p>
    </div>
  )
}

export default Acerca
```

App.css

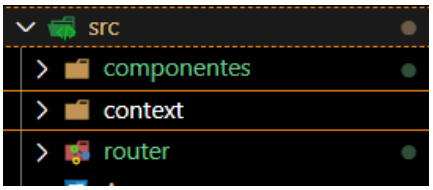
```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
```

```
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}

*{
  margin: 0px;
}

body{
  background-color: #ccc;
}

.header{
  margin: 0px;
}

.logo{
  color: white;
}

.header nav{
  background-color: black;
  height: 60px;
  display: flex;
```

```
  align-items: center;
  color: white;
  padding-right: 25px;
  padding-left: 25px;
}

.header ul{
  flex-basis: 80%;
  list-style: none;
  display: flex;
  justify-content: flex-end;
}

.header nav a{
  color: white;
  text-decoration: none;
  margin-right: 10px;
  margin-left: 10px;
}

.header nav a:hover{
  color: lightblue;
}

.content{
  width: 80%;
  min-height: 350px;
  background-color: white;
  border-radius: 15px;
  border: 1px solid #ccc;
  margin: 25px auto;
  padding: 25px;
}
```

3.3.- Aplicando useContext

A.- Creamos la carpeta contex:



B.- Creamos el fichero PruebaContext.js dentro de la carpeta context con el contenido:

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

// Este componente se comparte allá donde haga falta compartir datos
export const PruebaContext = createContext(null);
```

C.- Para que se aplique a toda la nevegación debe englobar a <AppRouter />.asi que hacemos:

```
function App() {
  return (
    <div className="App">
      <PruebaContext>
        <AppRouter />
      </PruebaContext>
    </div>
  );
}
```

Pero para que el elemento pueda compartir información he de añadir “.Provider” al componente PruebaContext.

```
function App() {
  return (
    <div className="App">
      <PruebaContext.Provider>
        <AppRouter />
      </PruebaContext.Provider>
    </div>
  );
```

```
}
```

3.3.1.- Pasando datos por el contexto

A.- En el PruebaProvider le paso un value:

```
function App() {
  return (
    <div className="App">
      <PruebaContext.Provider value="Datos desde el App,
PruebaContext">
        <AppRouter />
      </PruebaContext.Provider>
    </div>
  );
}
```

Que será accesible desde cualquier componente.

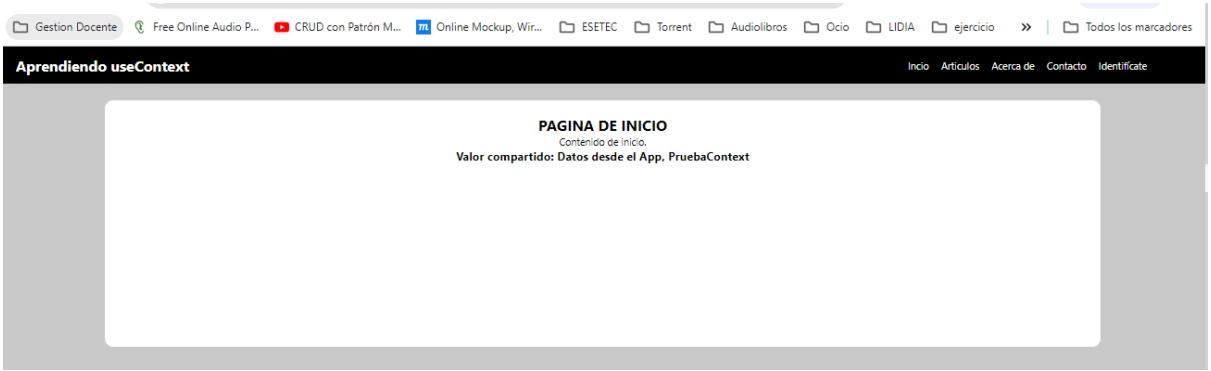
B.- En un componente, por ejemplo Inicio.js, creo una constante al que igualo a useContext. Y al useContext le paso el PruebaProvider:

```
import React, { useContext } from 'react'
import { PruebaContext } from '../context/PruebaContext';

const Inicio = () => {

  const contextoCompartido = useContext(PruebaContext);

  return (
    <div>
      <h1>PAGINA DE INICIO</h1>
      <p>Contenido de inicio.</p>
    </div>
  )
}

export default Inicio
```

De esta manera en la variable contextoCompartido ya tengo información:

Saquémoslo por pantalla:

```
import React, { useContext } from 'react'
import { PruebaContext } from '../context/PruebaContext';

const Inicio = () => {

  const contextoCompartido = useContext(PruebaContext);

  return (
    <div>
      <h1>PAGINA DE INICIO</h1>
      <p>Contenido de inicio.</p>
      <h3>Valor compartido: <strong>{contextoCompartido}</strong></h3>
    </div>
  )
}

export default Inicio
```



3.3.2.- Pasando variable useState por el contexto

Con ello conseguimos que desde un componente actualice el valor. Veamos un ejemplo:

Creamos la variable useState:

```
const [usuario, setUsuario] = useState({
  nombre: "David",
  web: "dbetqui@micorreo.es"
});
```

Y se la pasamos al Provider.

```
return (
  <div className="App">
    <PruebaContext.Provider value={{
      usuario, setUsuario
    }}>
      <AppRouter />
    </PruebaContext.Provider>
  </div>
);
```

App.js queda de la forma:

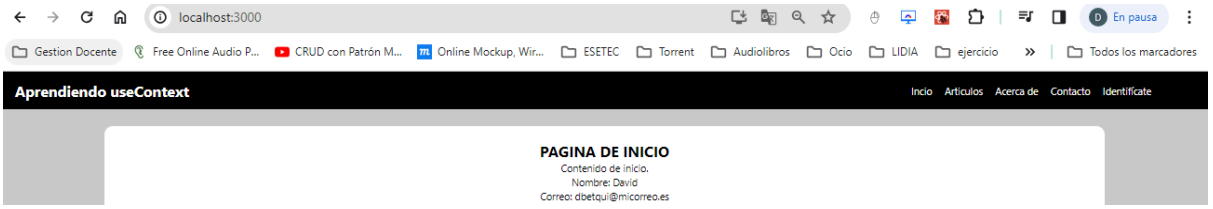
```
import { useState } from "react";
import "./App.css";
import { PruebaContext } from "./context/PruebaContext";
import AppRouter from "./router/AppRouter";

function App() {

  const [usuario, setUsuario] = useState({
    nombre: "David",
    correo: "dbetqui@micorreo.es"
  });
```

```
  return (
    <div className="App">
      <PruebaContext.Provider value={{
        usuario, setUsuario
      }}>
        <AppRouter />
      </PruebaContext.Provider>
    </div>
  );
}

export default App;
```

El Inicio.js lo modificamos para traer los datos.

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

const Inicio = () => {
  const contextoCompartido = useContext(PruebaContext);

  return (
    <div>
      <h1>PAGINA DE INICIO</h1>
      <p>Contenido de inicio.</p>
      <p>Nombre: {contextoCompartido.usuario.nombre}</p>
      <p>Correo: {contextoCompartido.usuario.correo}</p>
    </div>
  );
};

export default Inicio;
```

Esto es lo que observamos en pantalla:



3.3.3.- Modificando useState del contexto

Imaginemos que en Inicio.js creo un botón para modificar el useState que me llega desde el Provider.

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

const Inicio = () => {
  const {usuario, setUsuario} = useContext(PruebaContext);

  const changeName = () => {
    setUsuario({
      nombre: "Pepe",
      correo: "dbetqui@micorreo.es"
    });
}

  return (
    <div>
      <h1>PAGINA DE INICIO</h1>
      <p>Contenido de inicio.</p>
      <p>Nombre: {usuario.nombre}</p>
      <p>Correo: {usuario.correo}</p>
      <button onClick={changeName}>Cambiar nombre</button>
    </div>
  );
};

export default Inicio;
```

Si pulsamos veamos que ocurre en pantalla:

Ejercicio de entrenamiento no evaluable

1.- En el componente Login.js crear un formulario con los campos: nick, nombre y correo electrónico para que actualice los datos una vez autenticado. Modificar lo que haga falta.

2.- Modificar el menú para que cuando un usuario se identifique aparezca su nombre en la barra y el botón cerrar sesión.