**P4 React Github Users Project**

**Setup:**

npx create-react-app 🡪 Install all Dependencies

React Router Dom 🡪 Inside App.js: Path y Path Error. Switch para Evitar ver Error inside te Homepage (Switch Renders the first Child that matches).

<Router>

      <Switch>

        <Route path='/' exact>

          <Dashboard></Dashboard>

        </Route>

<Route path='\*'>

          <Error> </Error>

        </Route>

      </Switch>

</Router>

React Icons 🡪

Styled Components 🡪

const Error = () => {

    return (

    <Wrapper>

      <div>

        <h1>404</h1>

        <h3>The page cannot be found</h3>

        <Link to='/' className='btn'>

          Back to main page

        </Link>

      </div>

    </Wrapper>

  );

};

**Pages:**  En orden de creación

1. Components
2. Pages
3. Error Page / Login Page (before user auth)
4. Importamos Components a Dashboard (main)
5. **Context API** (context.js)

const GithubContext = React.createContext();

// Provider

const GithubProvider = ({ children }) => {

  return (

    <GithubContext.Provider value={'Hello'}>{children}</GithubContext.Provider>

  );

};

// GithubContext 🡪 Pasamos el Context dentro del provider y los children.

// IMPORTANTE 🡪 Value = {} 🡪 Pasaremos los STATE

export { GithubContext, GithubProvider };

// Luego vamos a Index.js and Import GithubProvider

// then 🡪 Wrap the APP between <GithubProvider> APP </GithubProvider>

* Comprobamos el Funcionamiento en otra page (No main, pero subpage de Main). 🡪Ej: Info.js

const UserInfo = () => {

  const data = React.useContext(GithubContext);

  return <h2>user info component : {data}</h2>;

};

* Ahora podremos poner toda la Logic en el Context!
* STATE 🡪 luego Hook it using useContext.

1. Mock Data (Static Data to not run out of requests on Github API). Vamos a Github API y pegamos nuestro User como MockUser.js
2. Set our STATE with the Mock Data (context.js)

🡪 Inside GithubProvider

 // Set the state and return a useState(Data, Function to setState)

  const [githubUser, setGithubUser] = useState(mockUser);

  const [repos, setRepos] = useState(mockRepos);

  const [followers, setFollowers] = useState(mockFollowers);

* Cambiamos en Return 🡪 VALUE ={githubUser, repos, followers}.
* En Info.js LOG(data) y vemos el Object de toda la Data.

1. Destructure la Data en Info.js y Creamos un Array(ítems) con 3 objects con dicha data.

Destructuramos dicho Items para obtener sus propiedades en una Arrow Function que nos devuelva un Article:

const Item = ({ icon, label, value, color }) => {

  return (

    <article className='item'>

      <span className={color}>{icon}</span>

      <div>

        <h3>{value}</h3>

        <p>{label}</p>

      </div>

    </article>

  );

};

Luego pasamos dicho Article en el Return de Info + el Spread operator de Item:

return <Item key={item.id} {...item}></Item>;

1. User Card 🡪 Usando la Data de GithubUser, destructuramos para armar una Tarjeta de Usuario

  const { githubUser } = React.useContext(GithubContext);

Donde SI existe, pasamos data como Location, Business, etc. (Ternary Operator: True ? Display : Null)

10) Followers 🡪 Creamos el Display de Followers. Destructurando la Data necesaria y haciendo un Display.

**\*\*SI NO HAY FOLLOWERS??\*\***

11) Fusion Charts 🡪 Usaremos para display los Charts, son Estilizables.

12) Calculate Most Used Language 🡪 Primero usaremos el Dummy Data de ChartData, para set la functionality y luego we will destructure the real Object to pass the Real Data.

13)

14)

15)