**PROJETO FINAL**

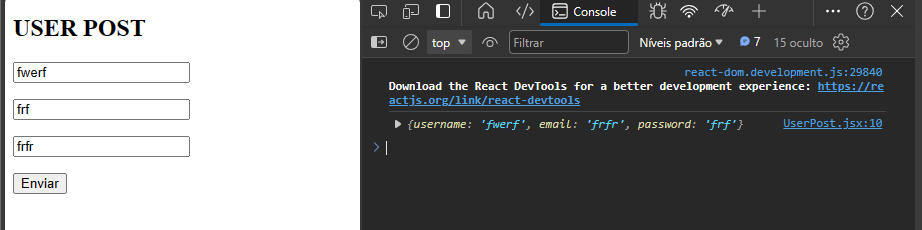
**1ª: Passo visualizar as páginas do site e sua distribuição**

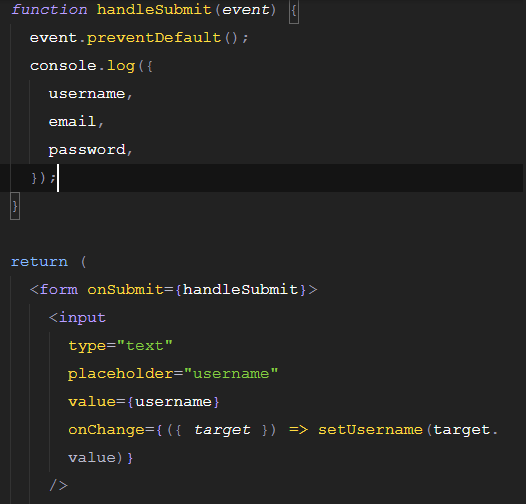
**2ª: Configuração**

**3ª: API, entender como ela funciona**

**4ª: Teste inicial da api para ve como ela funciona e seus retornos**

**4.1ª: USERPOST**

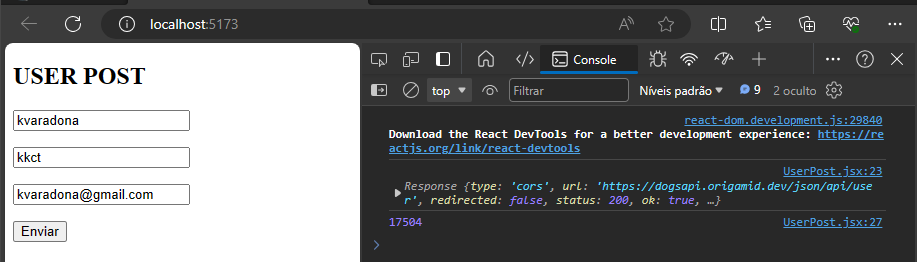
****

****

**Ao clicar no botão de enviar um objeto com os campos de username, email e password estão sendo “enviados”.**

****

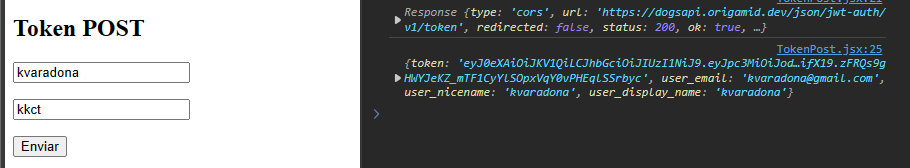
**Um post real com as saídas programadas para a API**

****

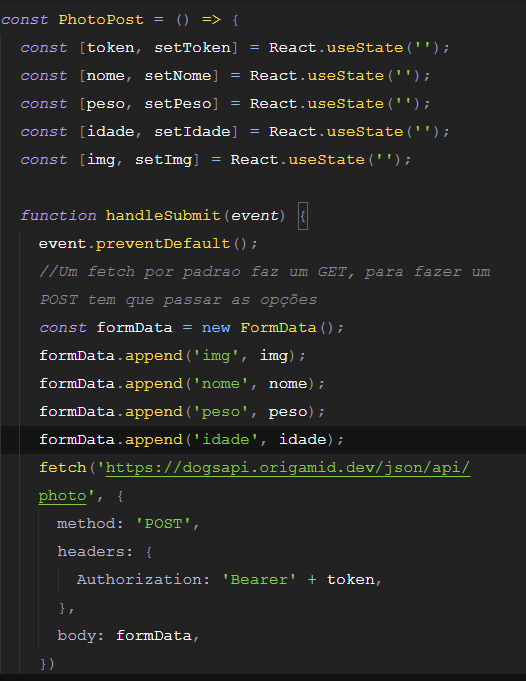
**4.2ª: TOKENPOST**

**Esse é tipo um método de login, no qual ele vai checar as credencias para ver se bate com a de algum usuário já cadastrado, basicamente, o mesmo método do user post só que sem o email pois não é necessário.**

****

****

**4.3ª: PHOTOPOST**

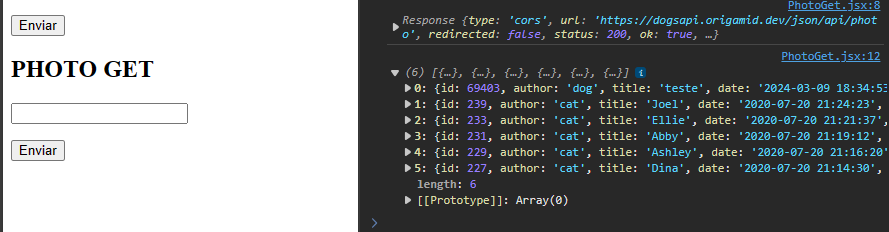
****

****

**Ta pegando o token para fazer a validação pois a foto so pode ser postada se a pessoa já tiver conta**

**4.3ª: PHOTOGET**

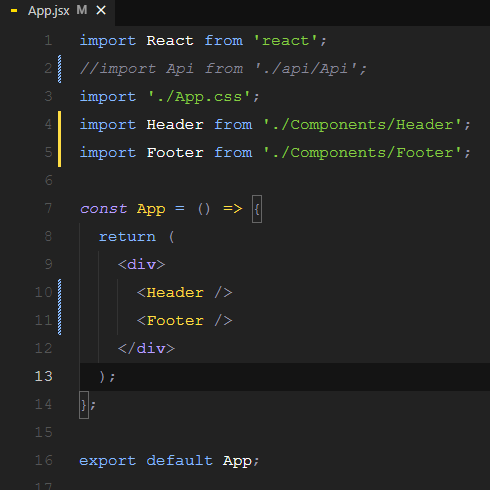
****

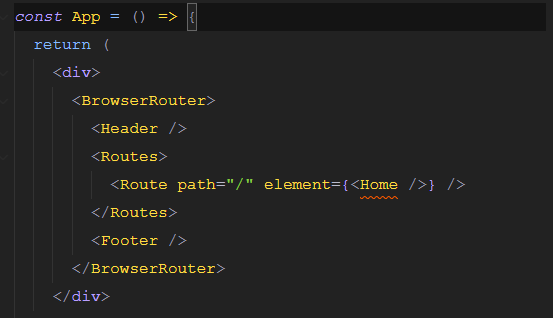
****

**5ª: Estrutura Inicial do projeto**

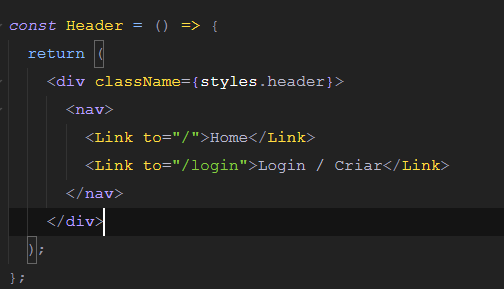
**5.1ª: Header e Footer**

**Primeiro cria-se a pasta de Components e dentro dele criamos os componentes de Header e Footer e adicionamos ao App.jsx, vale lembrar que eles vão ficar por fora do Routes pois vão estar presentes em todas as páginas.**

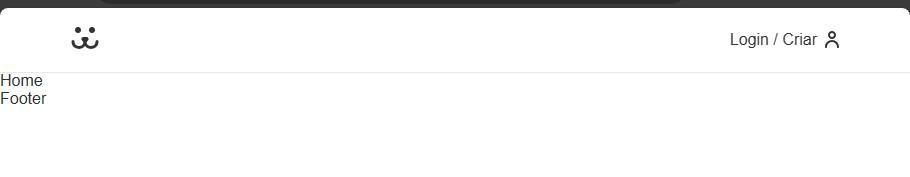
****

****

**No Header fazemos então a adição de Links que vão nos levar através de navegações rápidas sem renderizações para outras páginas como no exemplo da foto a seguir:**

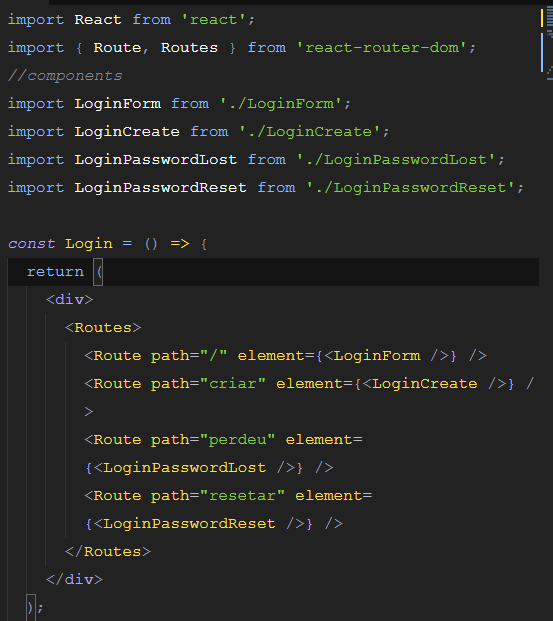
****

**Então estilizamos o Header.**

****

**6ª: Login**

**Na página de Login vai ser necessário, o nested route. Pois vai ter a página de login a de perdeu a senha e a de cadastro.**

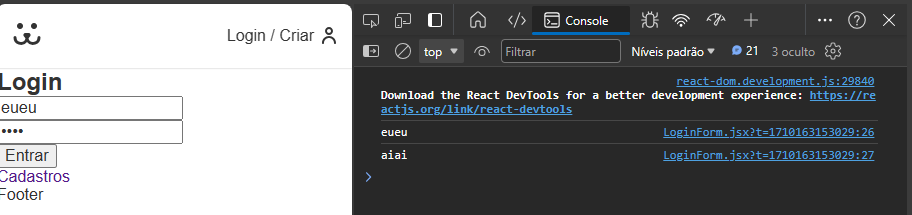
****

****

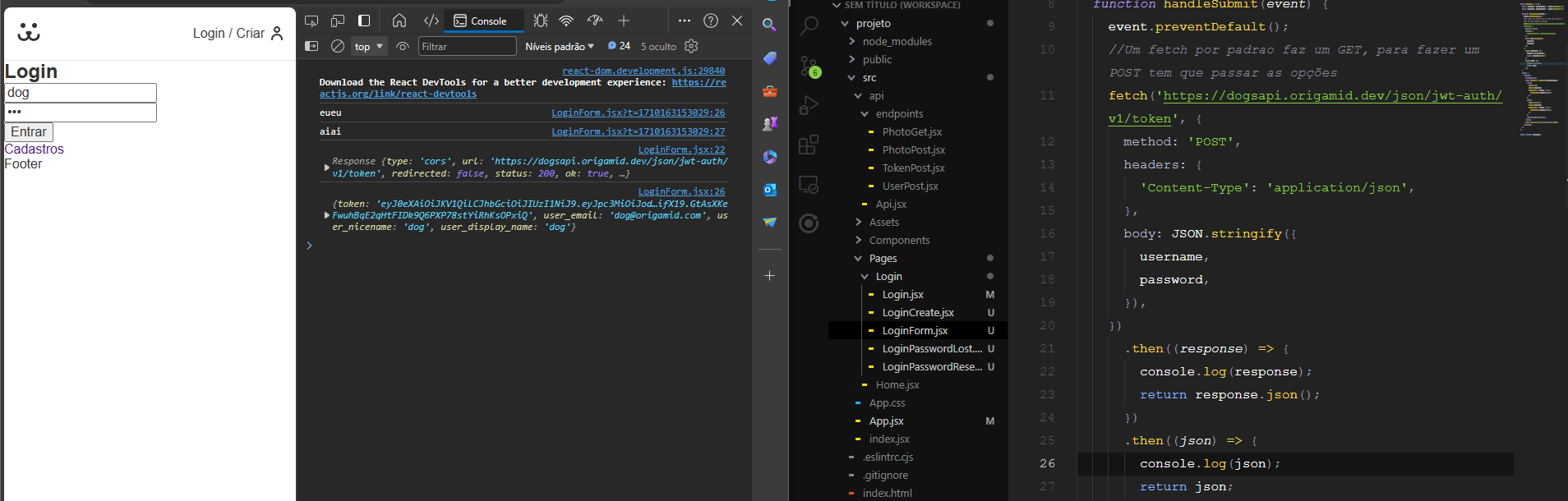
**Importante ressaltar que para o nested router funcionar, tem que colocar o \* na rota principal lá em App.jsx para dizer que tem mais rotas dentro daquela rota.**

**6.1ª: LoginForm**

****

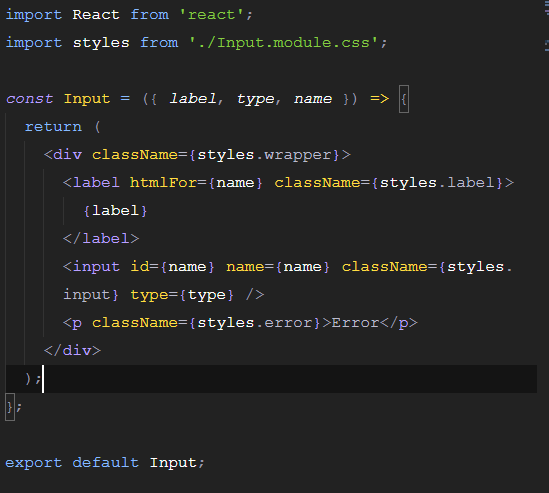
****

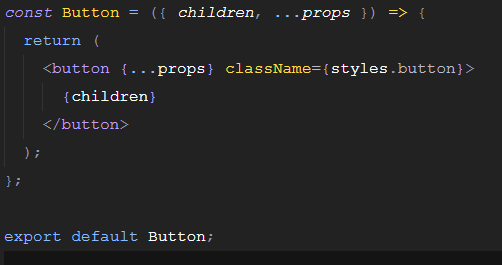
**Teste, faremos agora utilizando o fetch para a api, para fazer login tem que puxar o token do usuário correspondente**

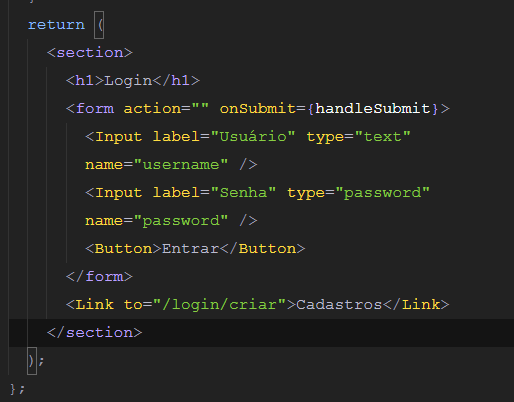
****

**7ª: Componentes de Formulário**

**Na verdade, esse passo pode ser realizado antes do passo anterior, e é apenas uma forma de deixar mais performático e mais padronizado com a sua cara os formulários do site.**

****

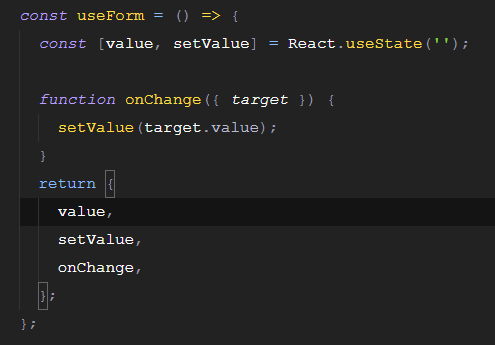
****

****

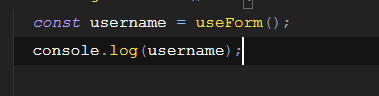
**8ª: useForm**

**Formato básico(teste):**

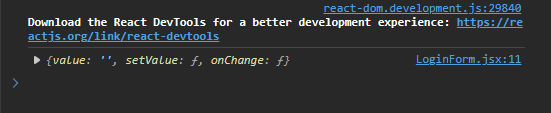
**Define-se os estados de value e setValue, junto com o método de onChange.**

****

**Então utilizamos, usename e password como tipos de useForm**

****

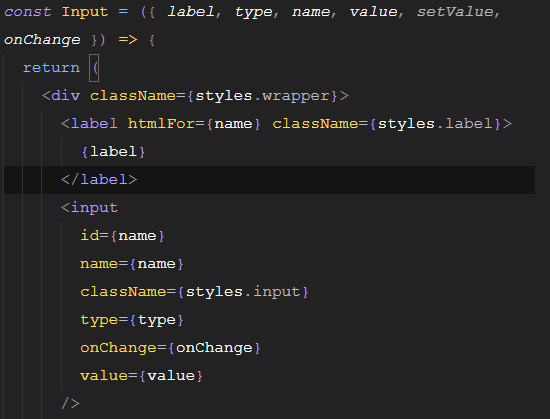
**Definidos isso, username e password passam a ter value, setValue e o método de onChange**

****

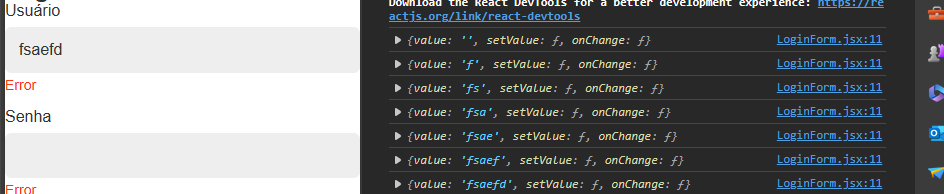
**Ao desestruturarmos no input, o componente de input passa a ter acesso a value, setValue e onChange**

****

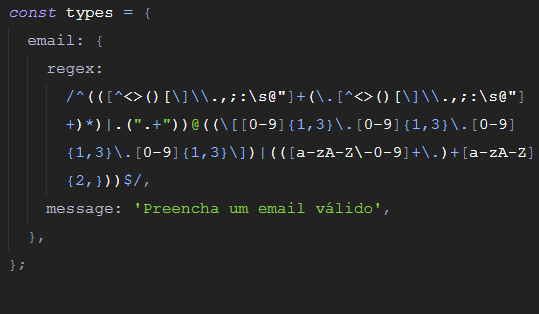
**Então definimos que o value do input vai ser igual ao value definido no componente que toma valor através do método de onChange, que seta o setValue(value).**

****

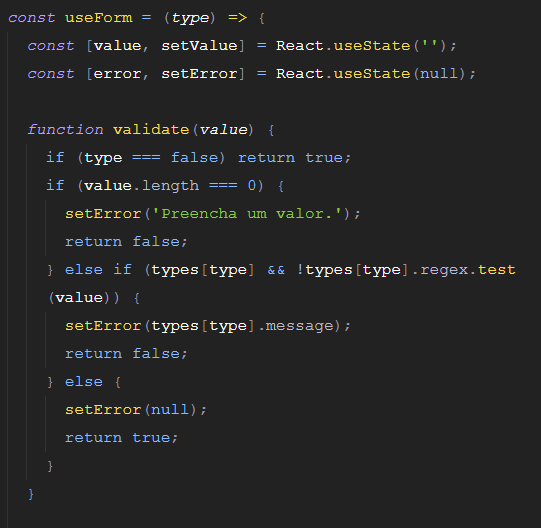
**Resultado:**

****

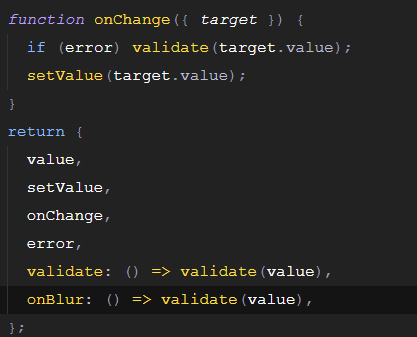
**AGORA VALENDO**

****

**Não vai ser necessário no nosso projeto, porém podemos definir um objeto de objetos com tipos de regex para serem validados**

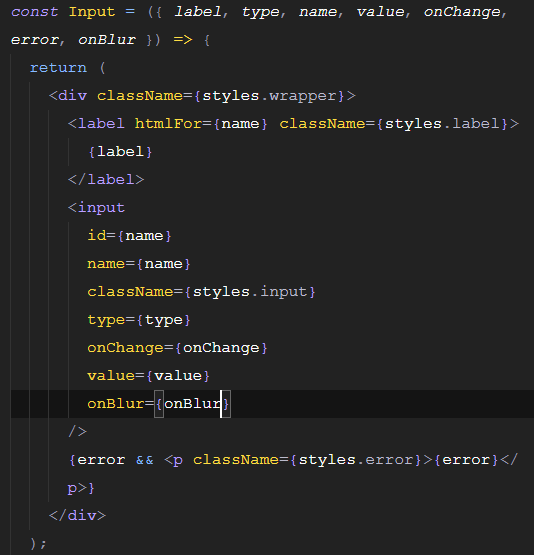
****

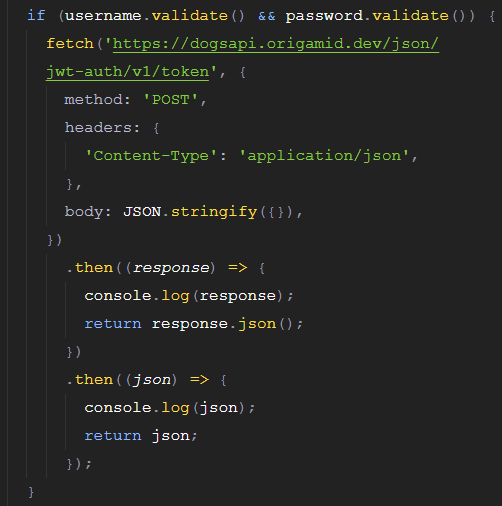
**Método de validate, acontece da seguinte forma: se o tipo definido no useform for false retorne true, caso não se o valor no campo for 0 retorne um erro para o usuário preencher valor, se tiver um tipo e o tipo não for atendido retorne um erro caso nada disso retorne nulo.**

****

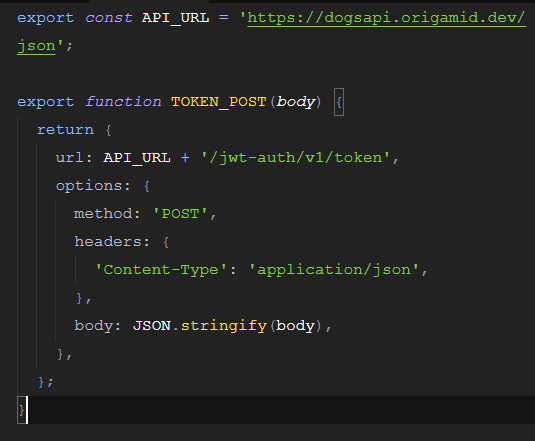
**A função de onChange foi mudada de forma que foi acrescentado um if(error) esse if é para caso o usuário não preencha o campo e saia clicando fora, quando ele voltar a preencher o erro suma.**

**Retorna o validate em forma de call-back pq ai ele já avalia o input quando for desestruturado la dentro, da mesma forma o onblur**

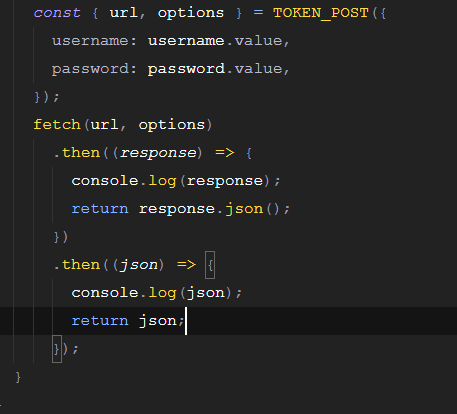
****

****

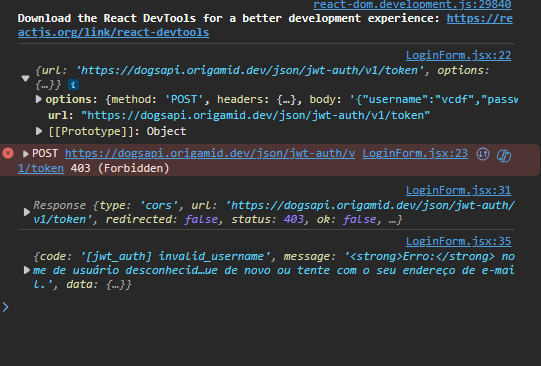
**9ª: API fetch**

****

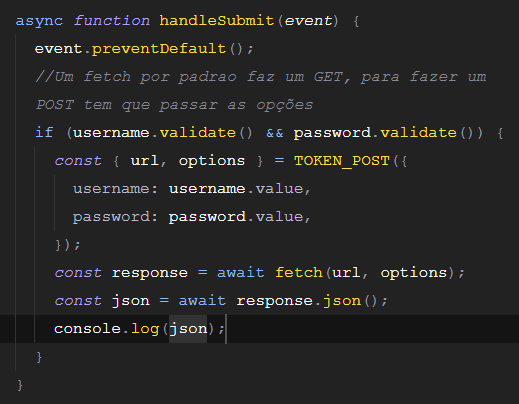
**Cria um arquivo de api.jsx onde todos os endpoints vao ser denifidos lá, o de token post retorna um objeto com url e options**

****

**Então em LoginForm desestruturamos um objeto do tipo token post e atribuímos username e password como os valores do seus values, e então fazemos o post com eles, resumindo bastante o método.**

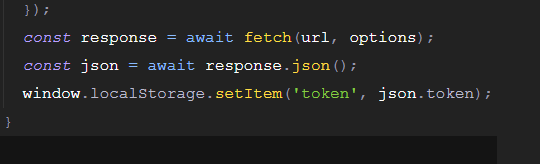
****

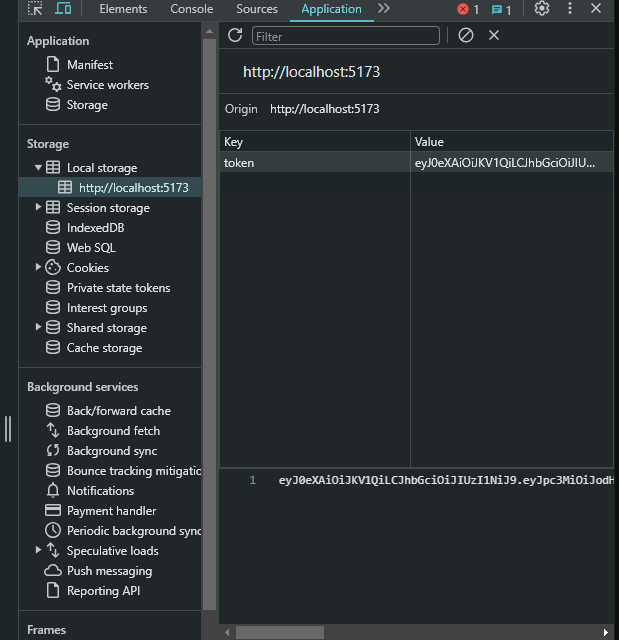
**Melhorando mais ainda...**

****

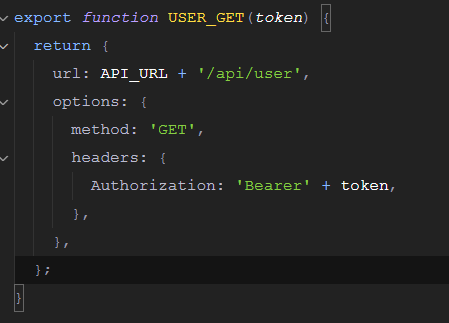
**Utilizando async await resumimos o nosso fetch a isso**

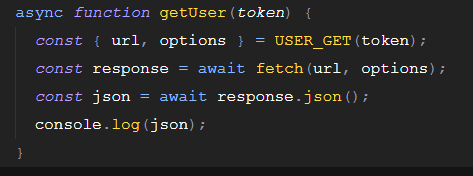
**9.1ª LOCAL STORAGE**

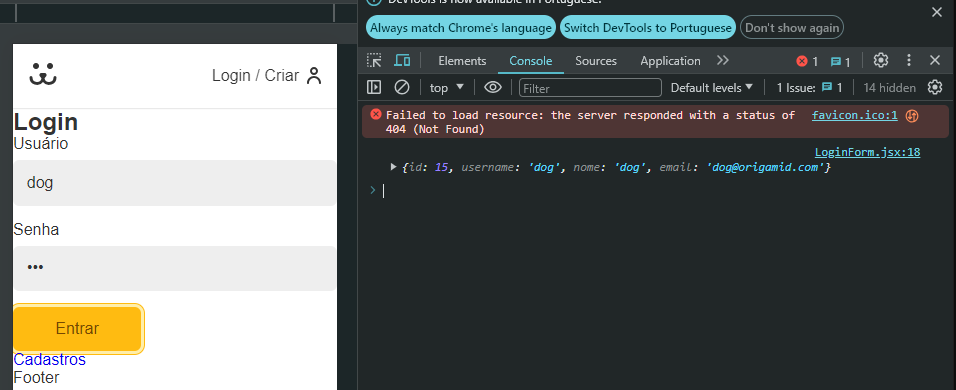
****

****

**9.2ª USER\_GET**

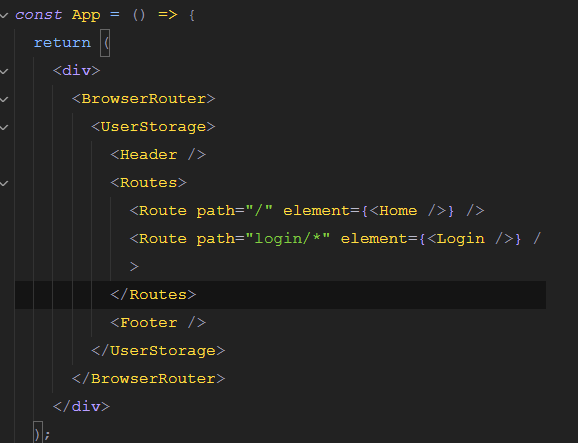
****

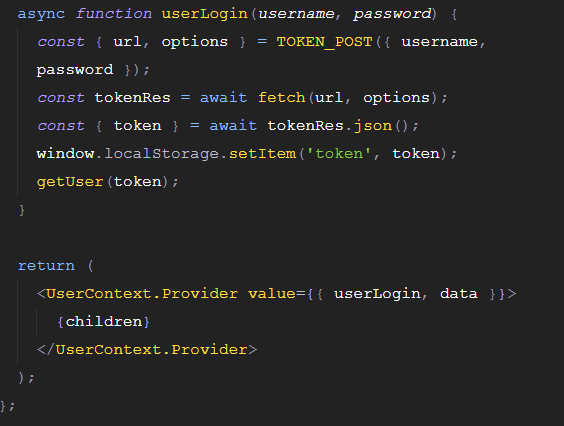
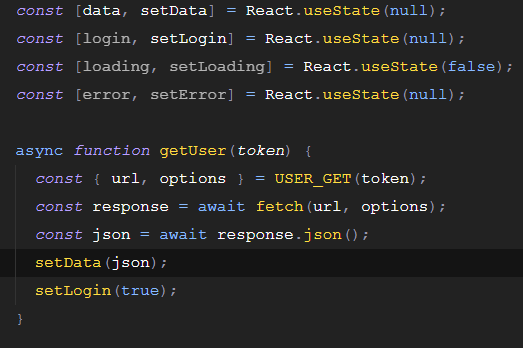
****

****

**10ª USER CONTEXT**

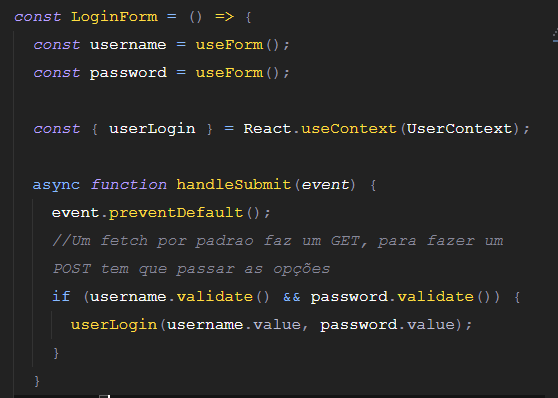
**UserStorage -> É o elemento que você tem que envolver todos aqueles elementos que terão acesso ao contexto.**

****

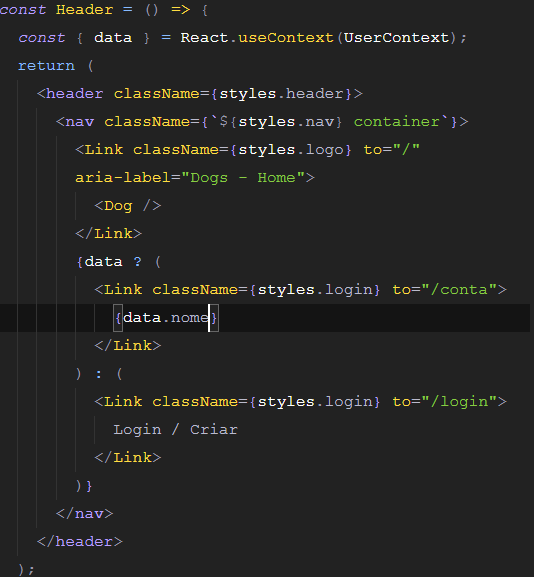
**Podemos atribuir ao UserContext a função de fazer login e tudo relacionada a ela como a de puxar token...então faremos algumas alterações. **

**No UserContext agora faremos a funções de login e de pegar usuário através do token, por isso para começar setamos data, login e loading e error. A função de Login, utiliza o endpoint de TOKEN\_POST passando um objeto como parâmetro que contem dois campos username e password esses campos serão destinados ao body. Esse é um método de POST que retorna um token, o qual será armazenado no localStorage e ao fazer o armazenamento já puxamos o usuário correspondente ativando a função de getUser.**

**O getUser por sua vez, recebe um token e retorna um objeto com username, id, email... o que seta atribuído ao objeto estático data através do setData.**

****

**Agora já em LoginForm ao invés de todo aquele código anterior apenas estanciamos um hook do tipo UserContext, de forma a já desestruturarmos e passamos então o userLogin com os valores de username e password recolhidos pelos campos de input do formulário.**

****

**Já no Header, para mostrarmos a pessoa logada, utilizamos um ternário mas antes importamos o hook de userContext já desestruturado pois precisamos apenas do data e fazemos a lógica do ternário.**