# Resumo

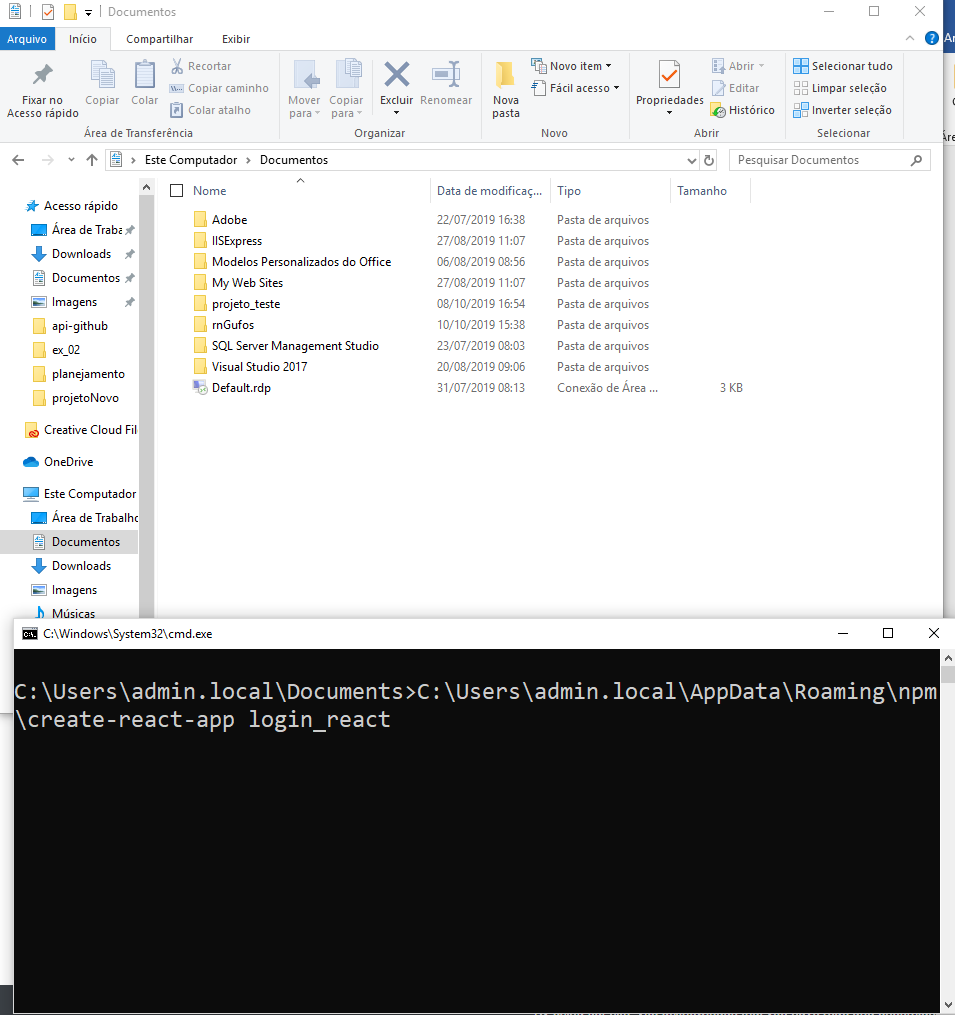
Dentro de uma aplicação existem diferentes usuários com diferentes funções. Algumas dessas funções não podem ser acessadas por qualquer pessoa, devido a isso são criados dentro do sistema, para os usuários, diferentes níveis de permissão baseado na função que cada um deles desempenha dentro do sistema.

Neste documento estaremos construindo uma aplicação React de duas páginas de Login e Categorias, onde o usuário irá realizar o Login com seu email e senha e ser direcionado a página de Categorias.

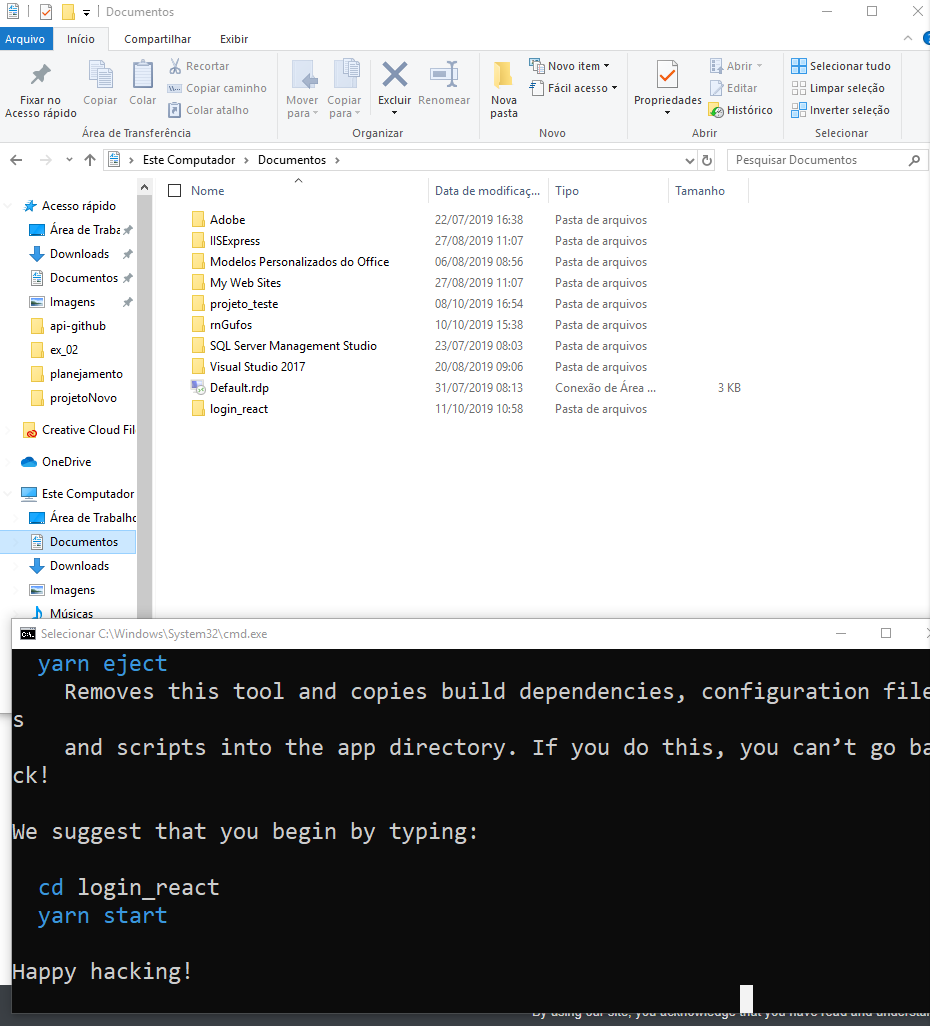
Em categorias, se o usuário tiver uma permissão de administrador ele poderá listar e cadastrar novas categorias; se o usuário tiver uma permissão diferente ele poderá apenas visualizar as categorias listadas.

# Iniciando o projeto

Começar abrindo o cmd na pasta em que vou criar o projeto, passando o caminho do meu create-react-app, referenciando o comando *create-react-app <nome\_projeto>*



Após a execução do comando o projeto irá baixar as dependências necessárias e no próprio cmd estarão sendo indicados os próximos comandos a serem seguidos para iniciar o projeto.

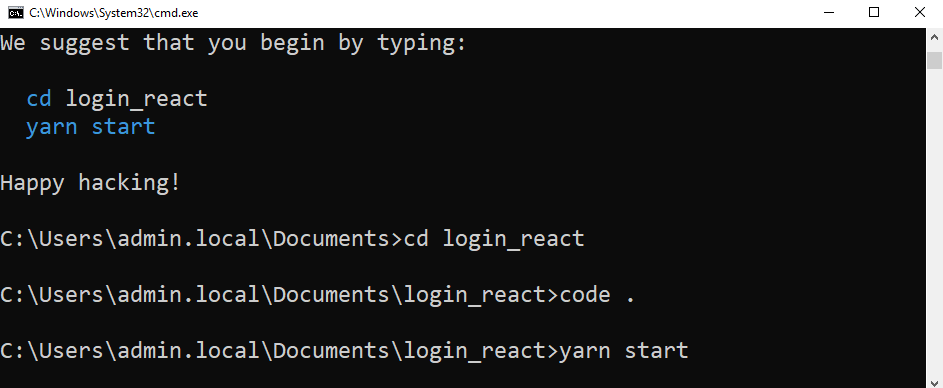


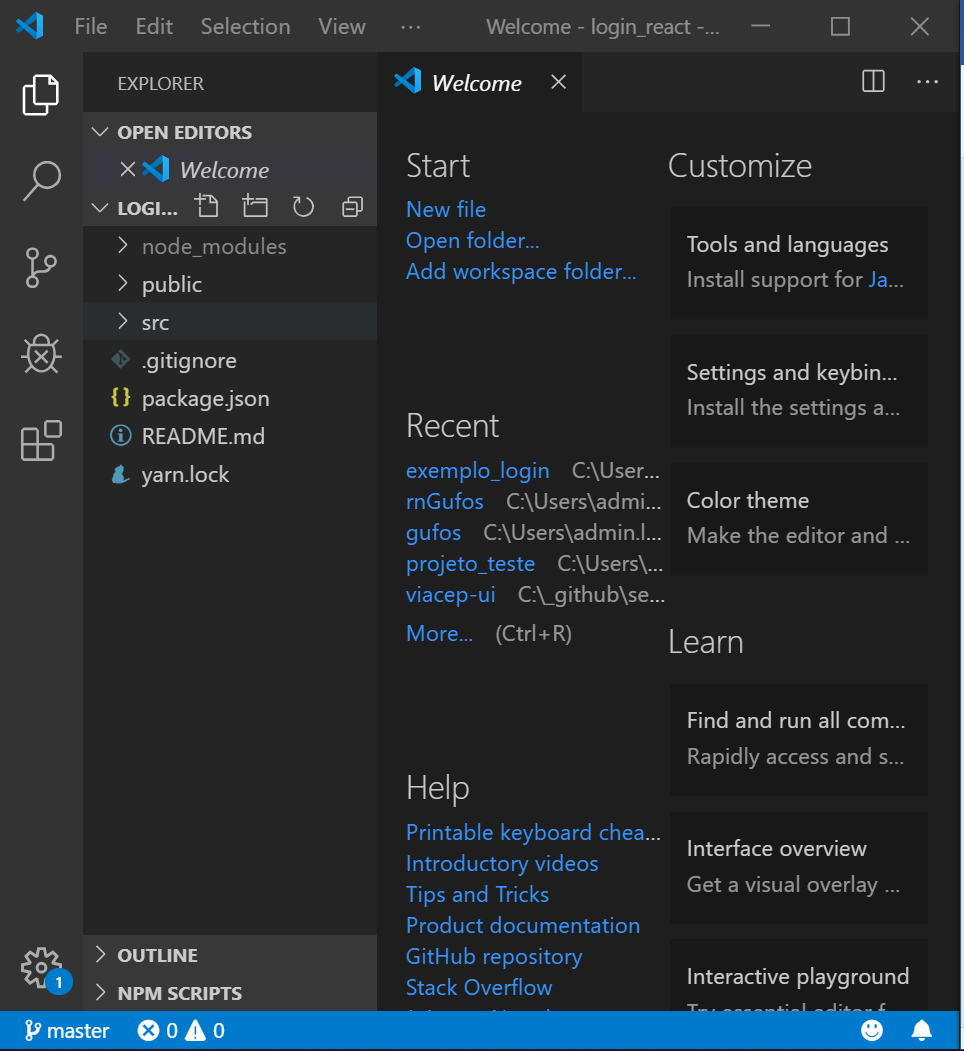
Executar dentro do cmd

*cd <nome-do-projeto>*

*code .*

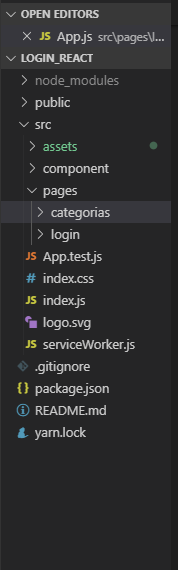
*npm start (yarn start caso você possua o yarn em sua máquina)*





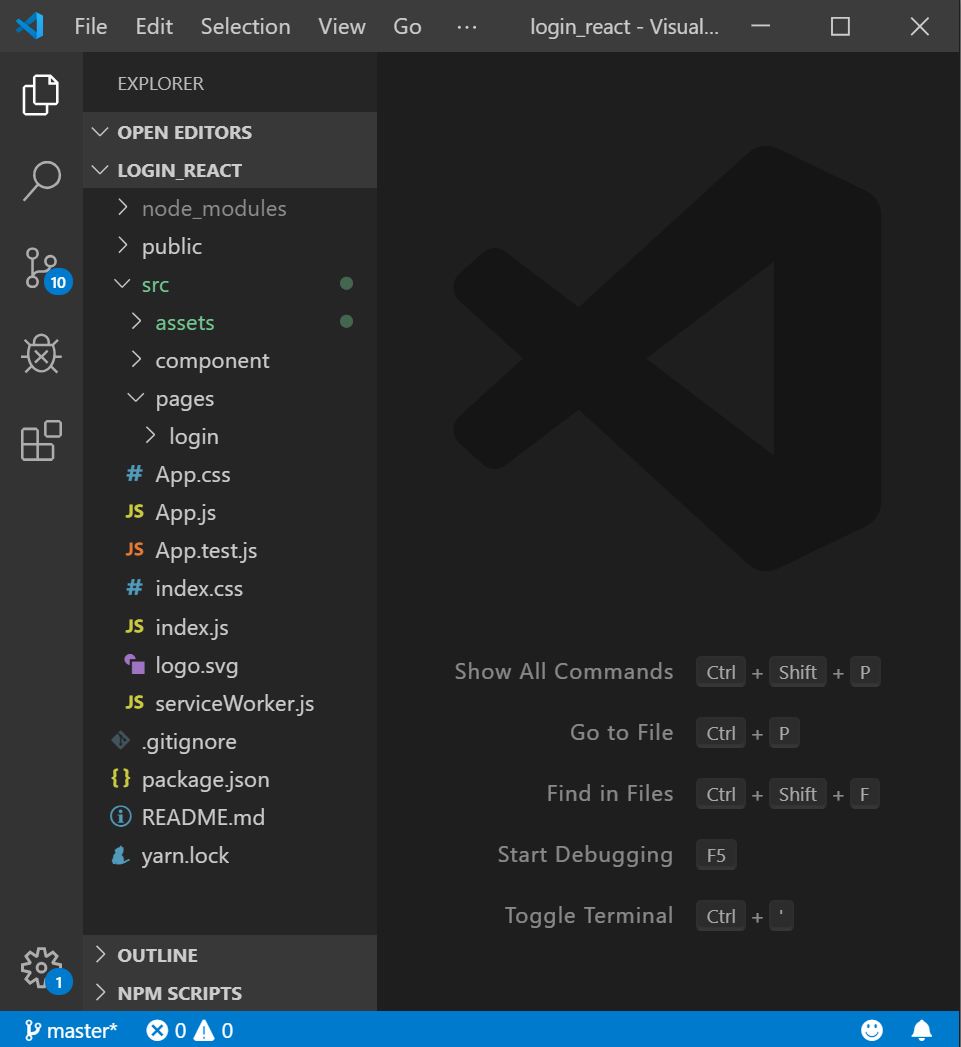
# Estrutura de pastas

Criar nossa estrutura de pastas básica, de pages e componente dentro do src. Dentro de pages, criar as patas categorias e login para seus respectivos componentes de categorias e login.



Adicionar a pasta assets localizada em Gufos.Base dentro do repositório da sprint de frontend

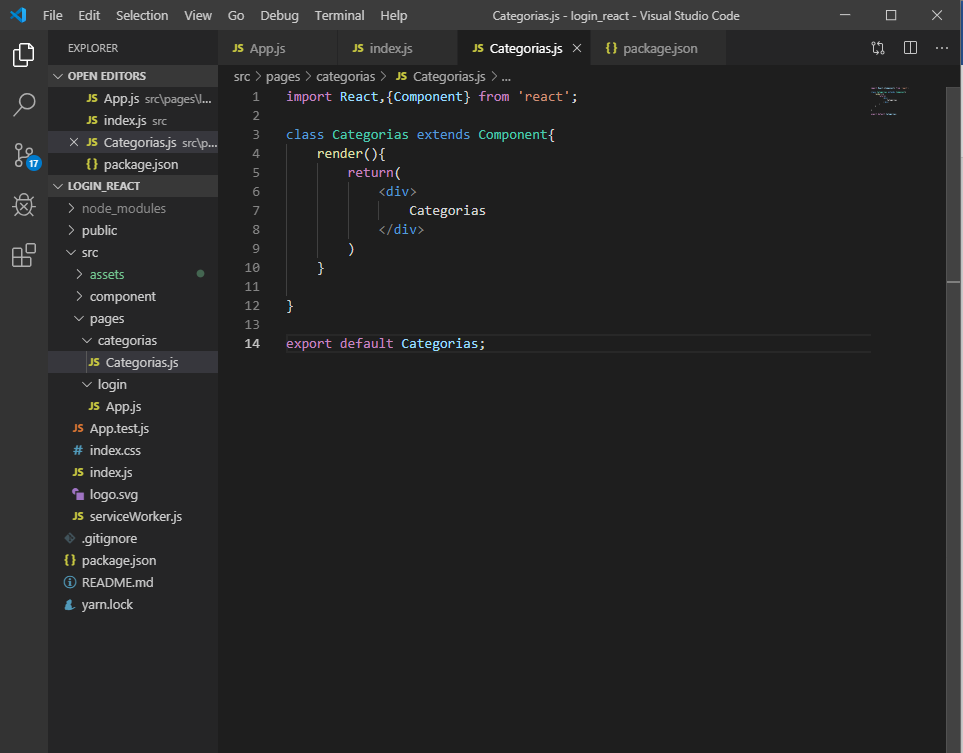
<https://github.com/senai-desenvolvimento/2s2019-t2-sprint-3-frontend/tree/master/gufos.base>



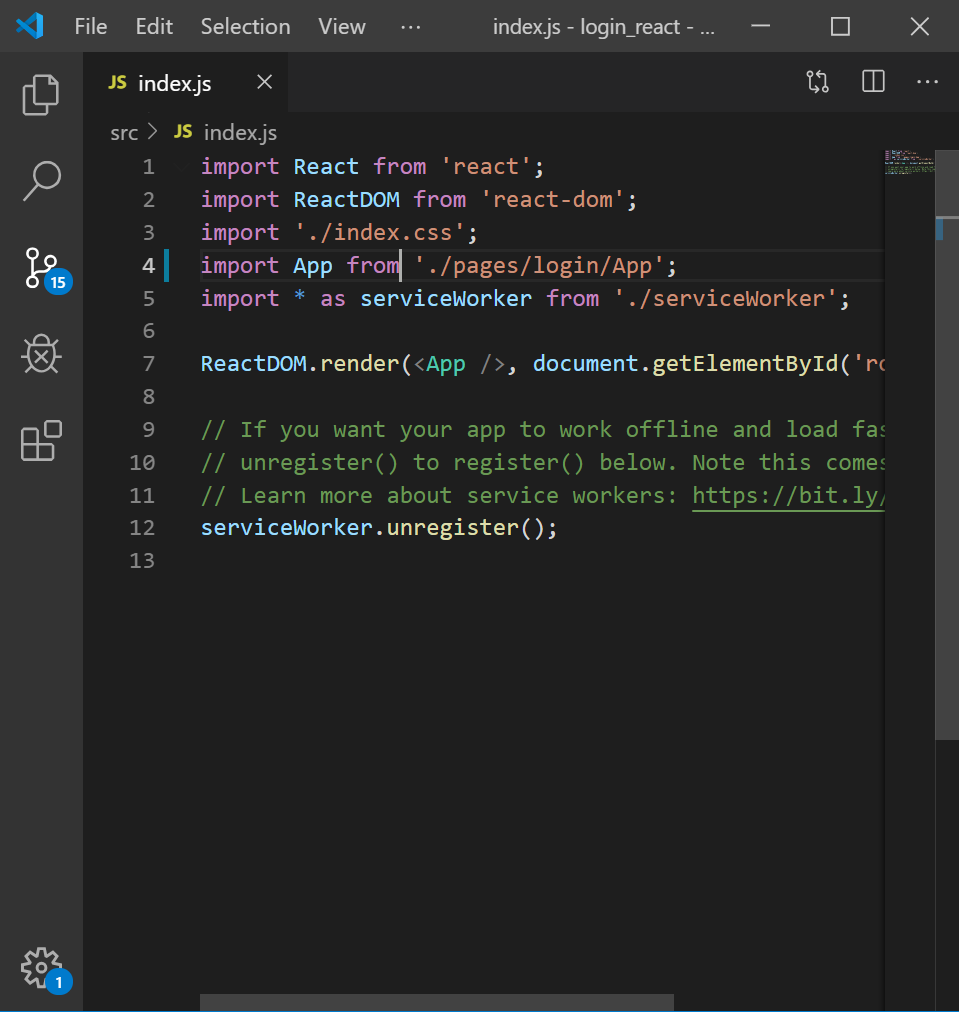
Mover o App.js para dentro de pages, dentro da pasta login, ele será agora nosso componente de login



Criar dentro de src/pages/categorias/ um componente Categorias.js com a estrutura básica de um componente statefull (criado por classe).



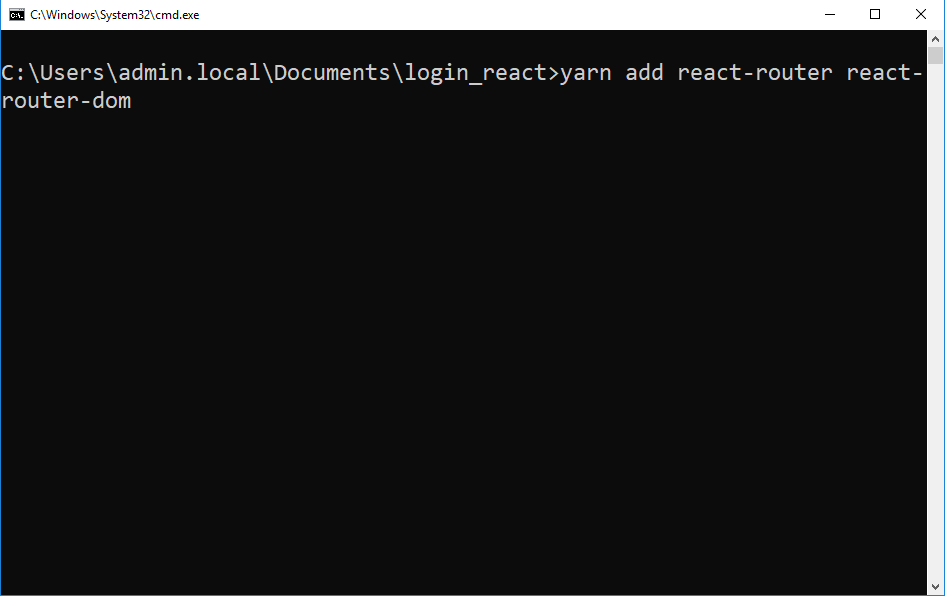
Ir ao index.js e alterar o caminho da referência, que antes era “./App”, para o novo caminho de App.js

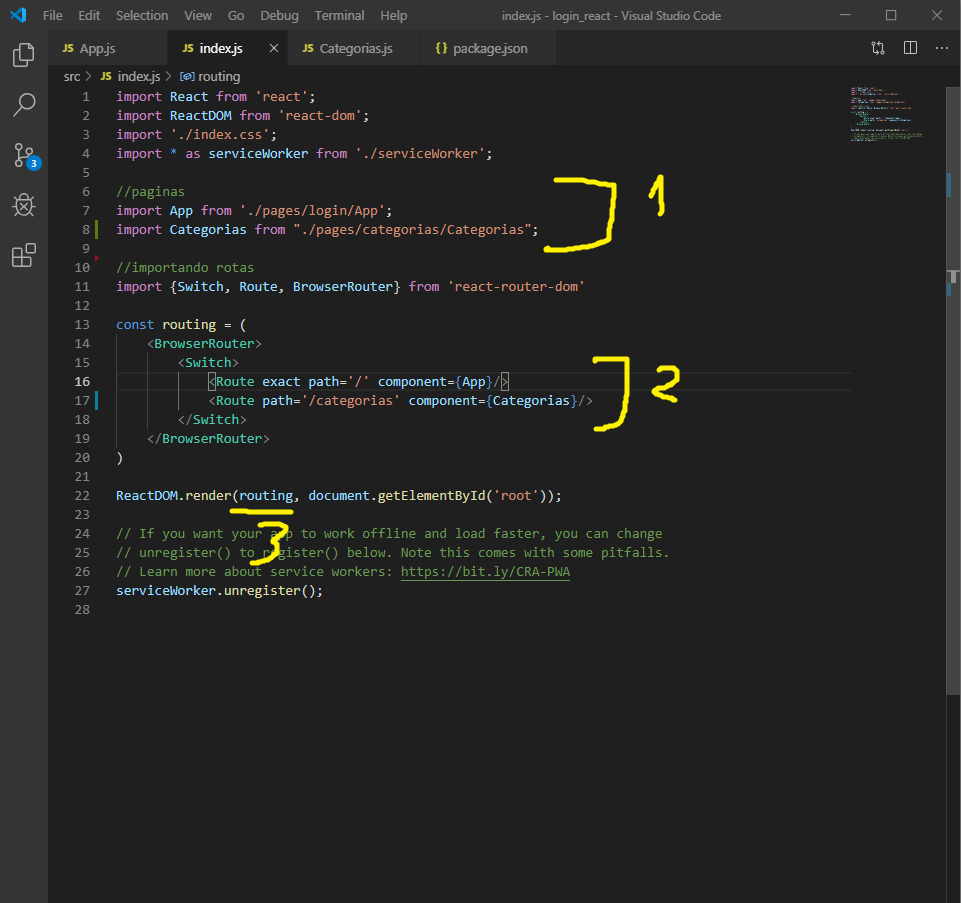


# Configurando rotas

Sabendo que nossa aplicação terá duas páginas precisaremos então configurar duas rotas: /login e /eventos.

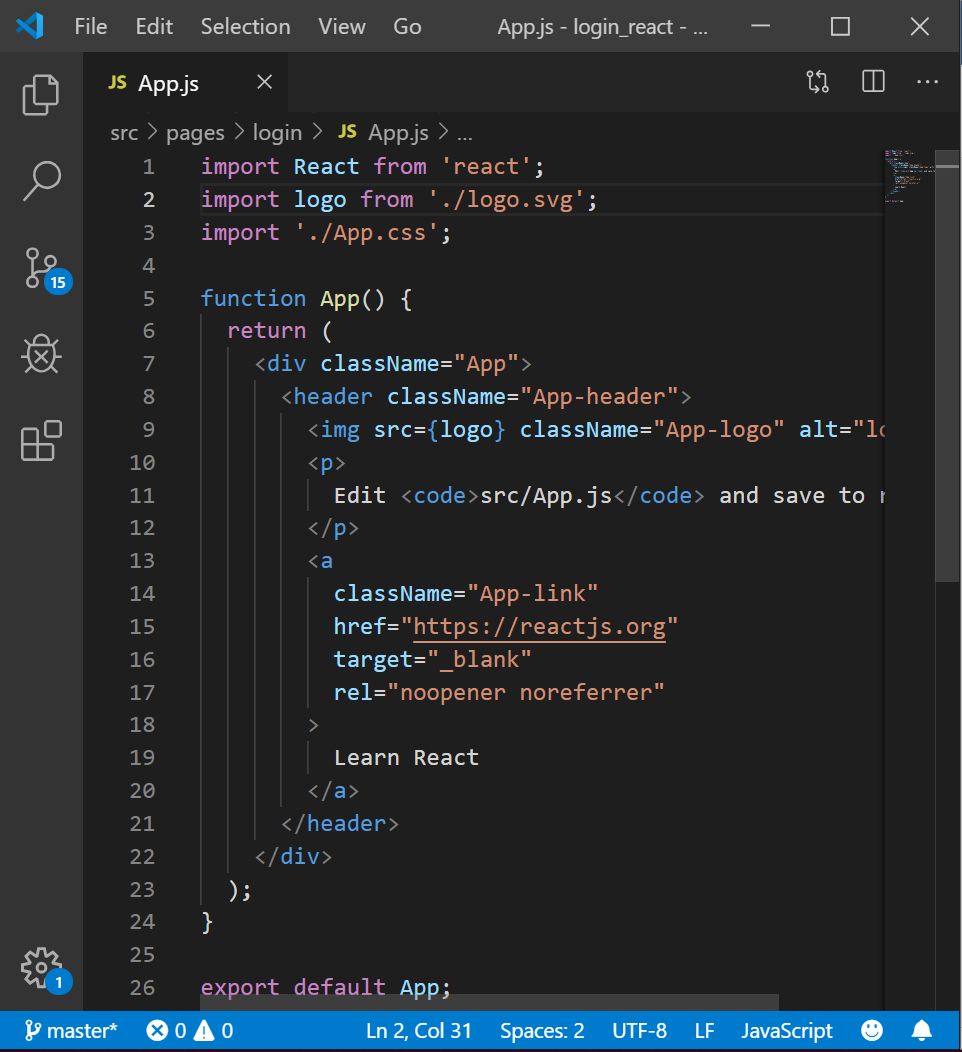
Iremos interromper nossa aplicação e instalar via npm o react-router-dom e o react-router.

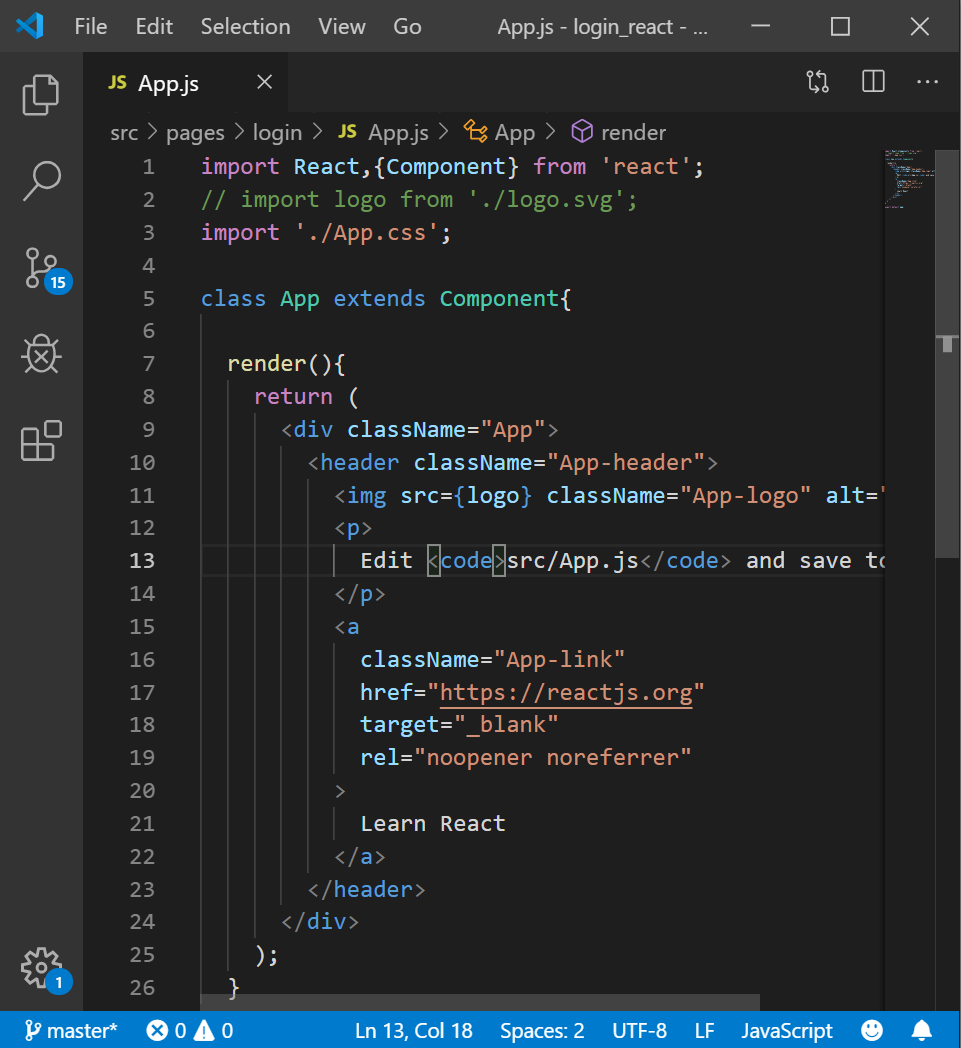




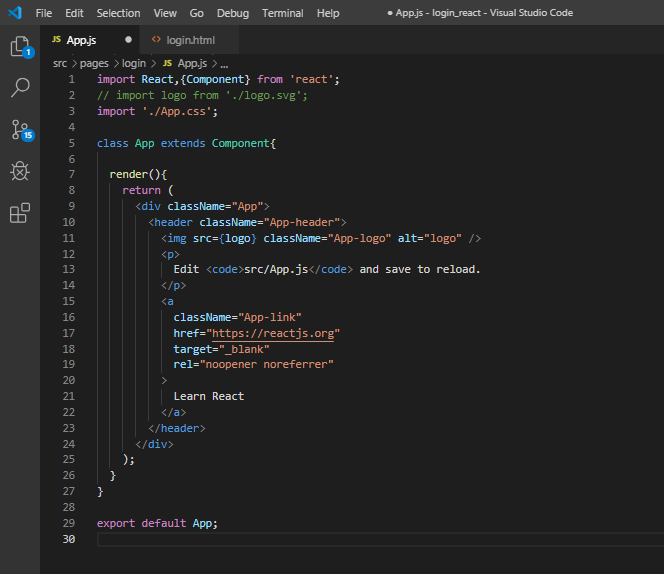
# Definindo componente: Login

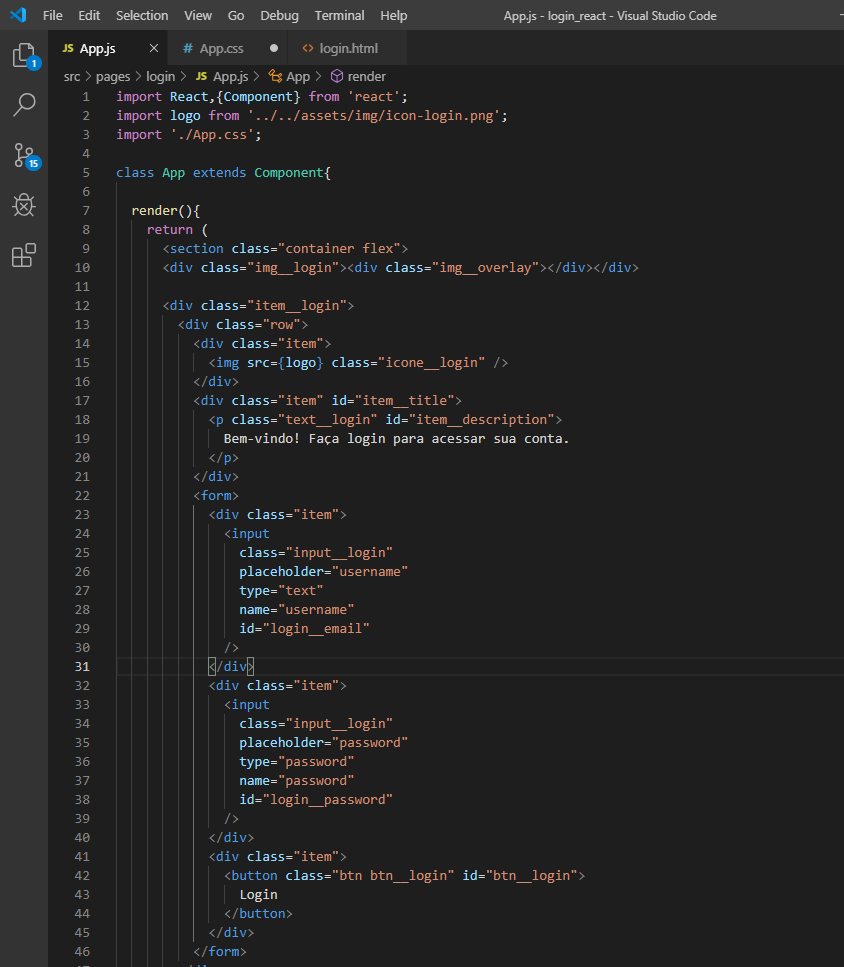
Dentro de App.js retirar a referência do logo no topo da página e trocar de uma função para uma classe.



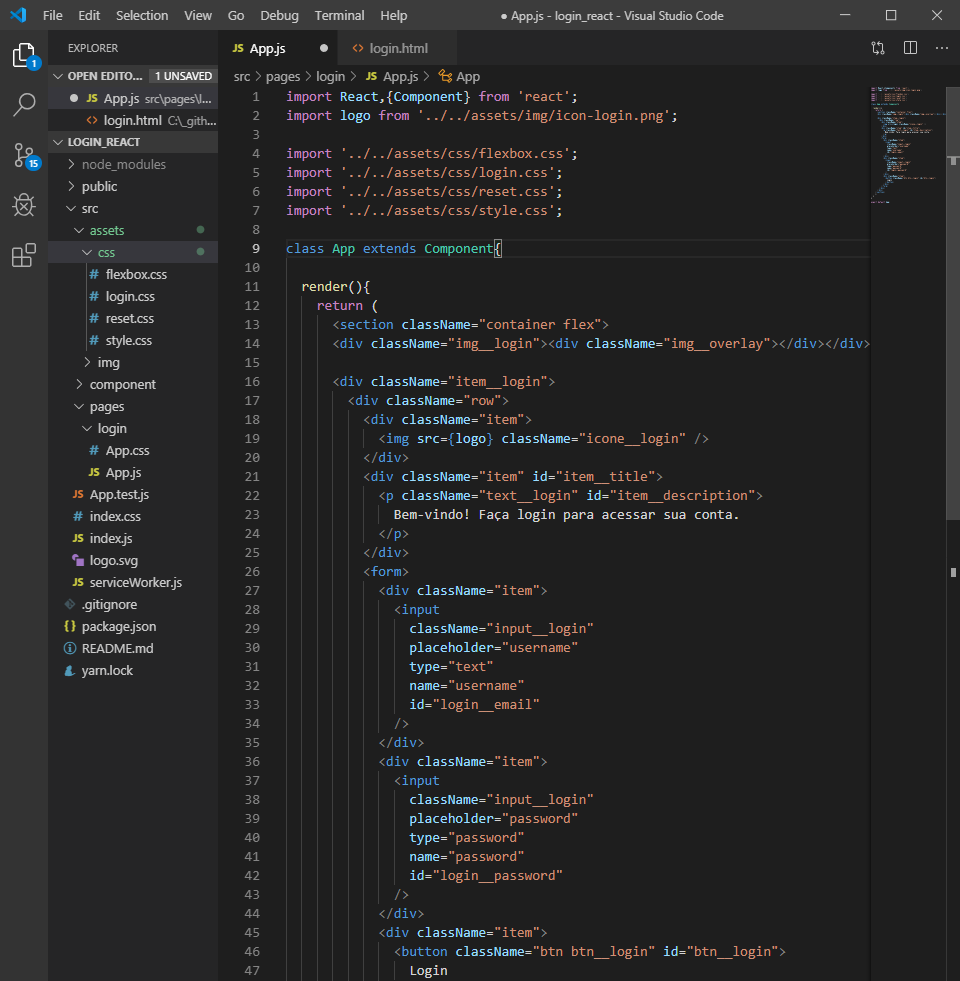


Substituir o componente JSX que está no retorno da class App para nossa estrutura do login

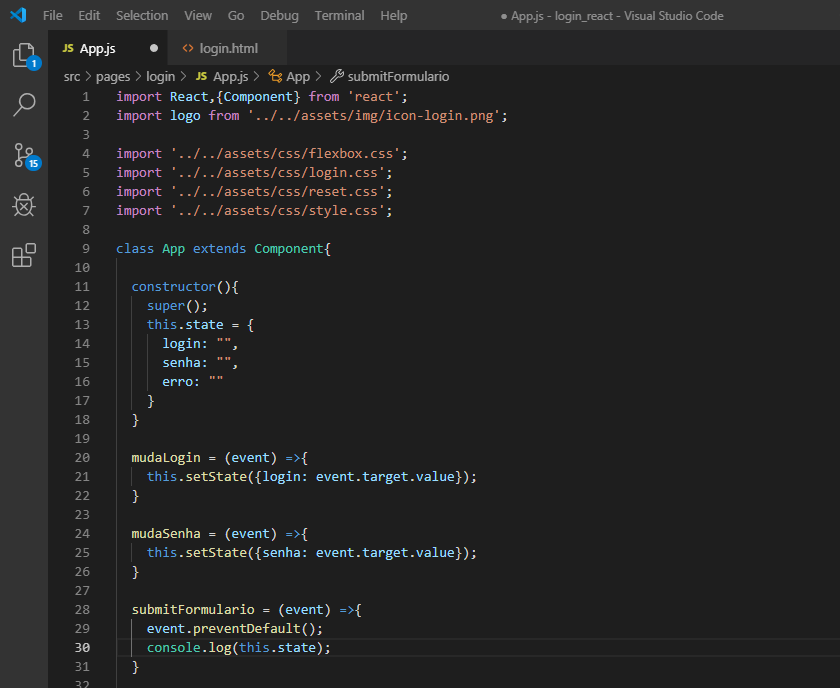




Importar módulos de CSS do assets trocando todos os atributos class por className



Construir nosso state da classe App, que armazenará o valor do nosso campo login, senha e a mensagem de erro.



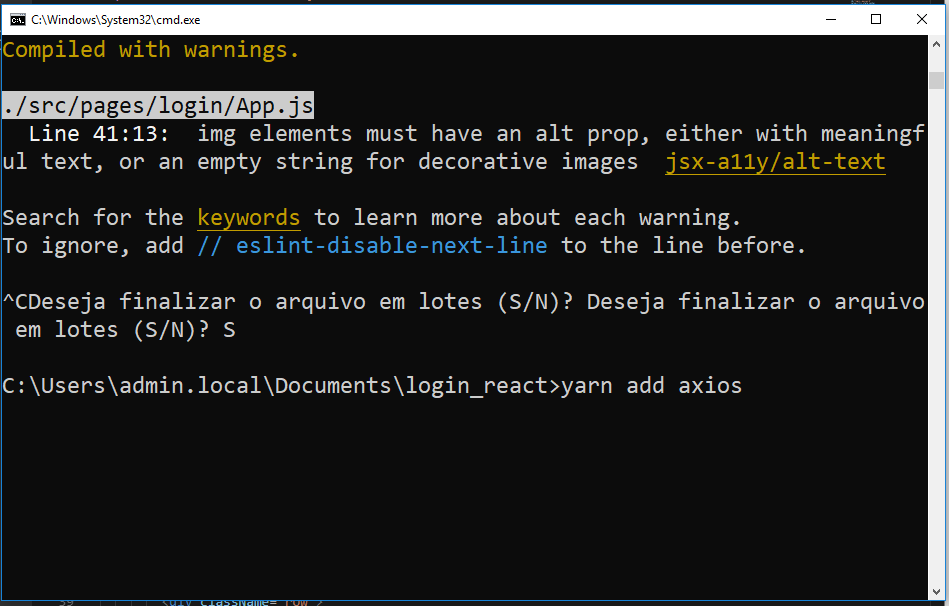
Adicionar eventos onSubmit no formulário referenciando a função submitFormulário e onChange aos dois inputs referenciando os eventos de mudar login e mudar senha.

Ligar também os valores dos dois inputs aos valores de login e senha do meu state

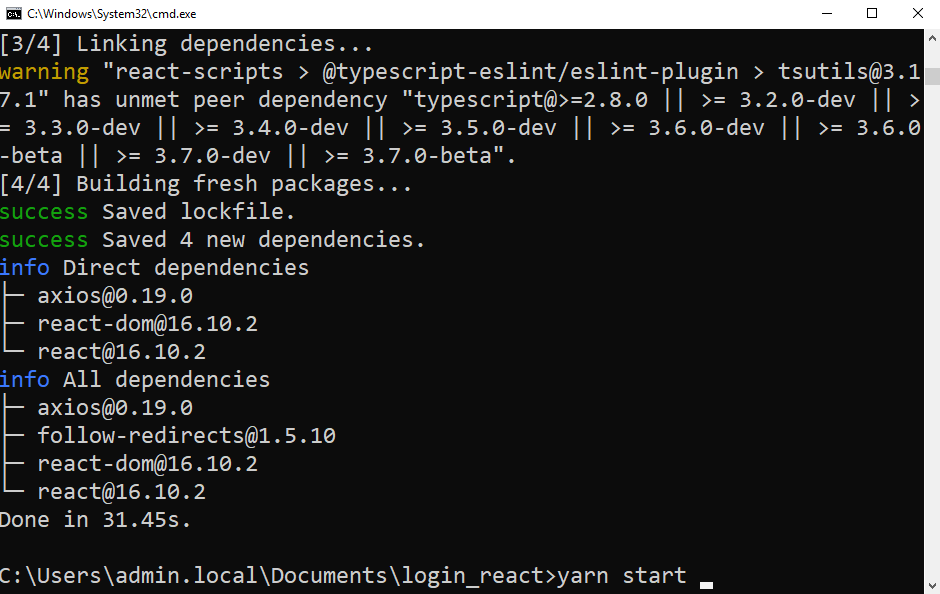


Conseguindo capturar os valores e armazena-los no state precisamos realizar a requisição.

Utilizaremos o axios para fazer essa requisição. Dessa forma deve-se interromper a aplicação (Ctrl+C) e instalar via npm install axios (caso você tenha o yarn instalado realize yarn add axios).

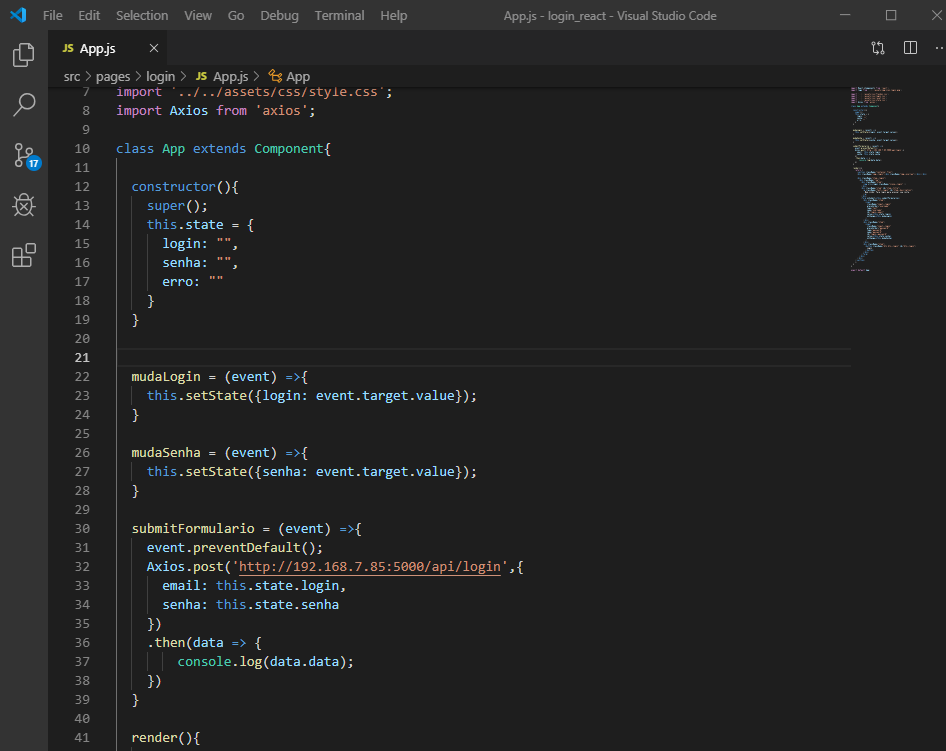


Ao finalizar a instalação iniciar de novo nossa aplicação

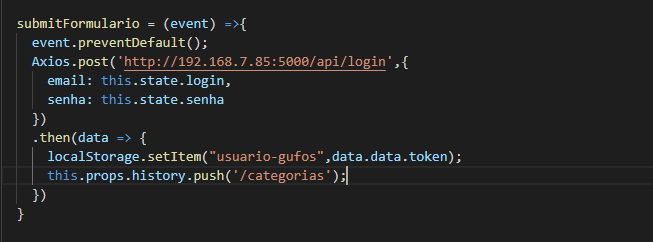


Importamos o Axios em nosso App.js e no submitFormulario configuramos a estrutura da requisição chamando o Axios e indicando o verbo POST. Como parâmetros passar o endpoint e o body da requisição.

Em seguida colocamos um *.then* para tratar a resposta da nossa requisição.

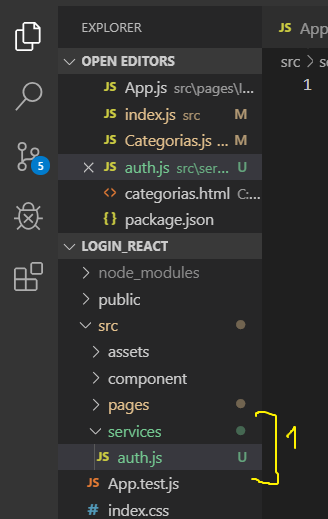


No .then da requisição especificar que no localStorage será criado um registro “usuário-gufos” armazenando o tolken que chega da nossa requisição.

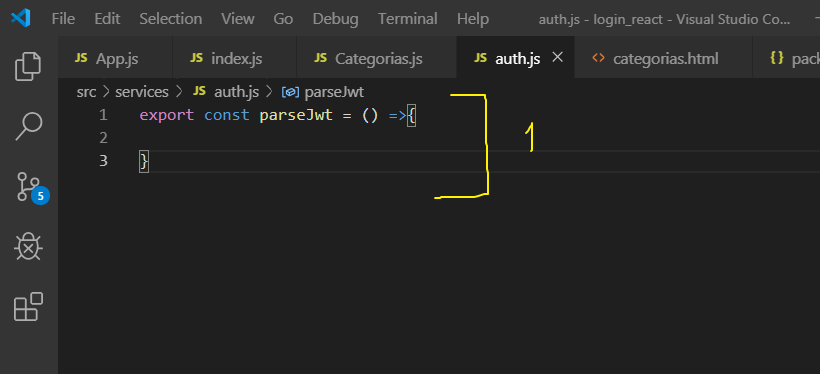


# Criar função que decifra o token

Criar uma pasta, dentro do nosso src chamada services, e dentro dessa pasta criar um arquivo auth.js.



Dentro do auth.js criar uma constante que armazena uma função exportando-a por padrão, e essa constante será responsável por decifrar nosso token.



Quando criamos nosso token no backend configuramos o Header, Payload e a Signature do nosso token. Cada um desses criptografado e ligados por um ponto ( . ) em uma única string.

Seguimos então dentro da nossa função capturando do nosso localStorage o token que lá armazenamos. Nesse token aplicaremos um .split(“.”) que irá retirar os pontos da nossa string e transformá-la num array.

Exemplo do nosso token antes do split(“.”):

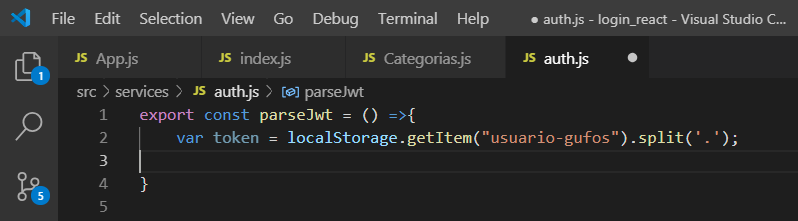
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImp0aSI6IjEiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBRE1JTklTVFJBRE9SIiwiUGVybWlzc2FvIjoiQURNSU5JU1RSQURPUiIsImV4cCI6MTU3MTIyOTkzNywiaXNzIjoiR3Vmb3MuV2ViQXBpIiwiYXVkIjoiR3Vmb3MuV2ViQXBpIn0.teU8ZoeK75z17OyYAOpJokvdNF2aaoZNzFhAOjXvHm0

Exemplo do nosso token depois do split(“.”):

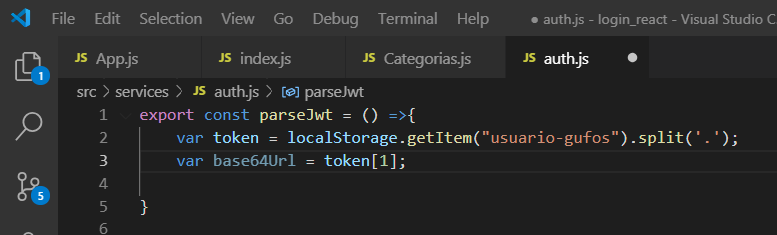
[

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9, eyJlbWFpbCI6ImFkbWluQGFkbWluLmNvbSIsImp0aSI6IjEiLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3JvbGUiOiJBRE1JTklTVFJBRE9SIiwiUGVybWlzc2FvIjoiQURNSU5JU1RSQURPUiIsImV4cCI6MTU3MTIyOTkzNywiaXNzIjoiR3Vmb3MuV2ViQXBpIiwiYXVkIjoiR3Vmb3MuV2ViQXBpIn0, teU8ZoeK75z17OyYAOpJokvdNF2aaoZNzFhAOjXvHm0

]



Sabendo que minha variável token retorna um array determino que o valor do índice 1 do meu array, que é onde se localiza meu payload, será armazenado em uma outra variável chamada base64Url.

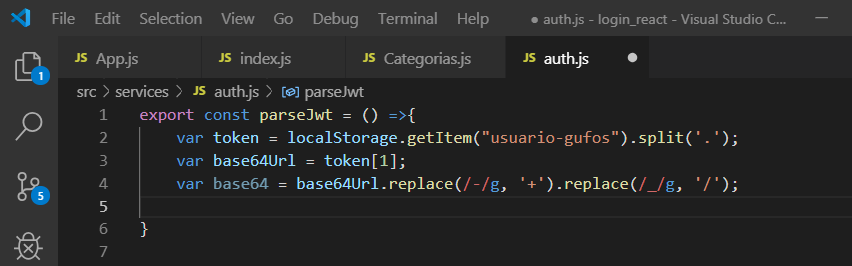


Tendo meu payload na variável base64Url aplicamos a função replace, que tem dois parâmetros, o primeiro é o valor que será substituído e o segundo é o valor que estrará no lugar.

Aplicar dois replace na base64Url:

- O primeiro replace irá substituir todos os hífens (“-”) da string base64Url pelo sinal de “+” e no final retorna a string resultante dessa ação.

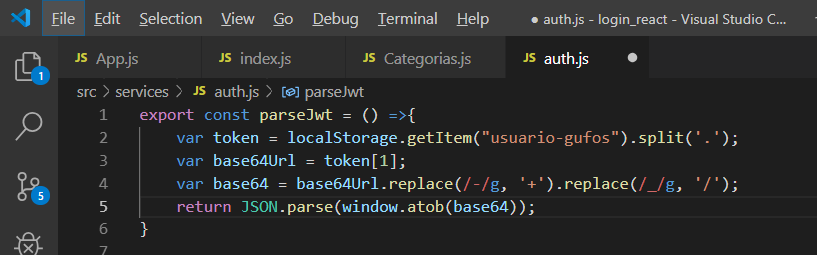
- O segundo trabalhará encima do valor retornado do primeiro replace, substituindo todos os “\_” por “/”.



O resultado dos dois replaces em sequência será armazenado na variável base64.

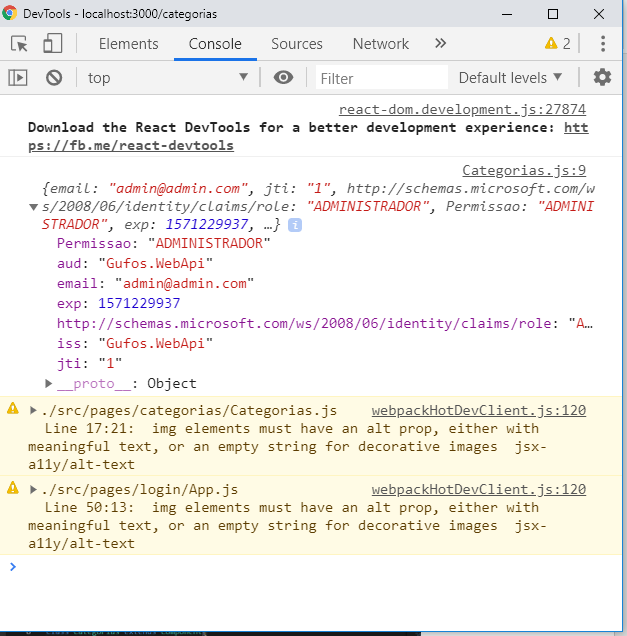
Por fim, usamos a função Window.atob() passando nossa base64 como parâmetro para decifrar nosso payload já capturado e tradado.

Esse window.atob() estará dentro de um JSON.parse, onde convertemos a resposta para o formato JSON.



Voltar para o Categorias, importar nossa função parseJwt e executa-la no componentDidMount de nosso componente Categorias

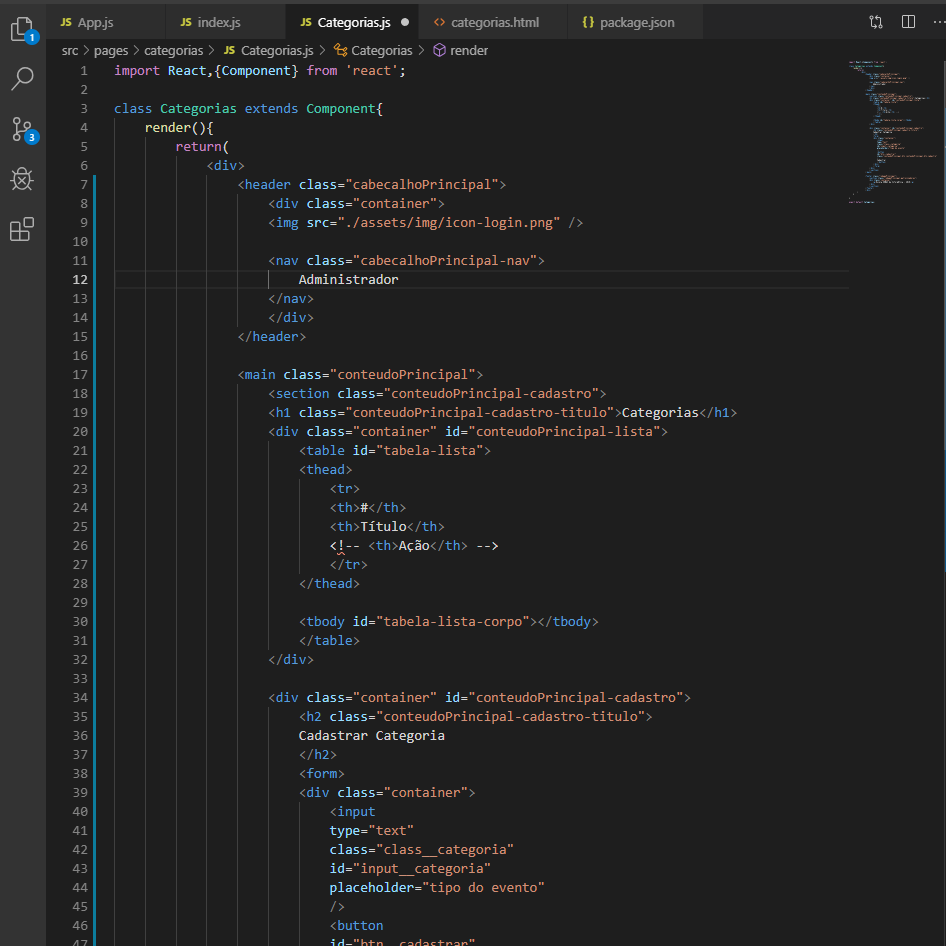


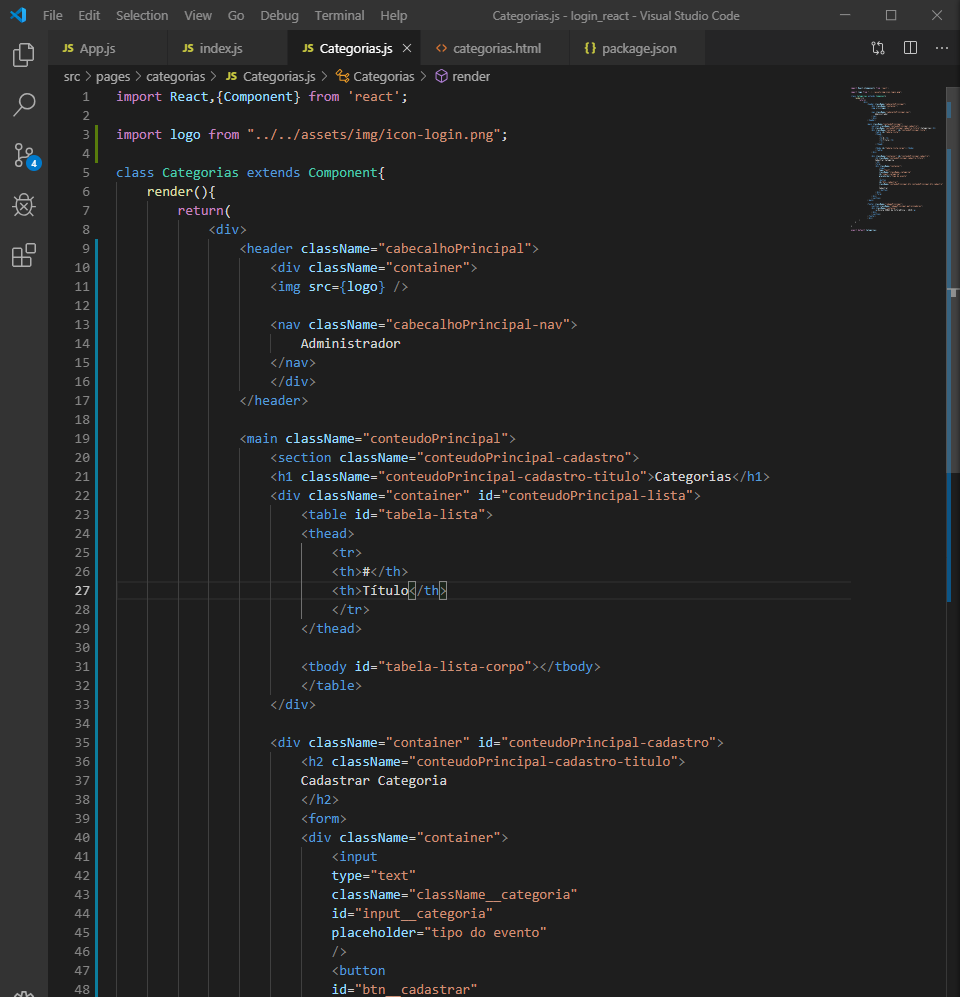


Em nosso console de desenvolvedor temos o token decifrado com as informações do login que realizamos.

# Definindo componente: Categorias

Colocar dentro do componente Categorias a estrutura do arquivo categorias.html localizado em Gufos.Base tirando comentários, os atributos class e importando a imagem.





Com nosso Categorias montado devemos aplicar nossa restrição de eventos, anteriormente citada. Conseguimos aplicar de duas maneiras: condicionando o retorno no render do nosso componente ou condicionando nossa rota, ambas as situações dependendo do token.

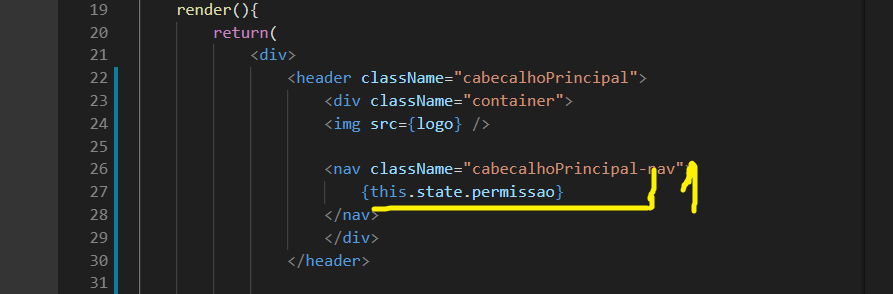
# Condicionando o retorno no render do nosso componente

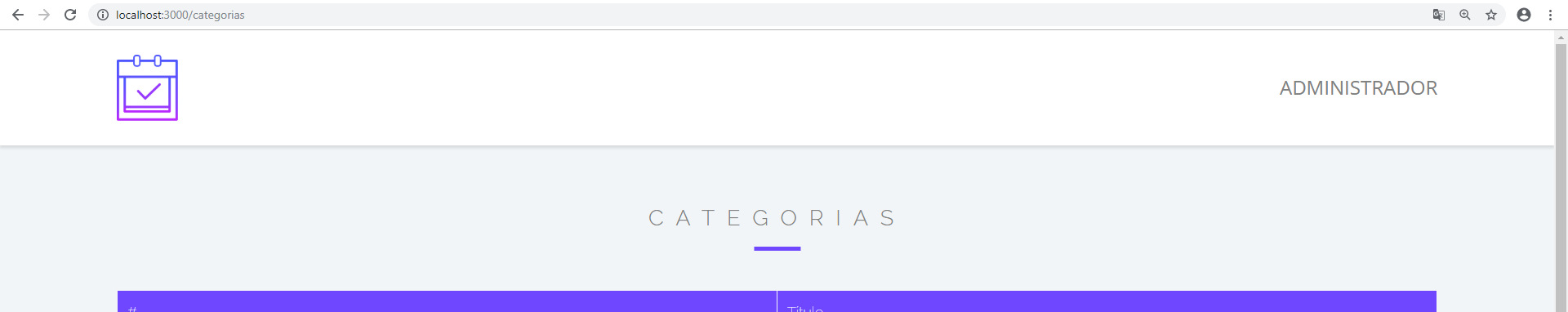
Construir estrutura de state com a chave permissão para armazenarmos o tipo de permissão.

Dentro do nosso componentDidMount definimos que nosso estado irá receber o valor da chave Permissao, do json do JWT que convertemos.



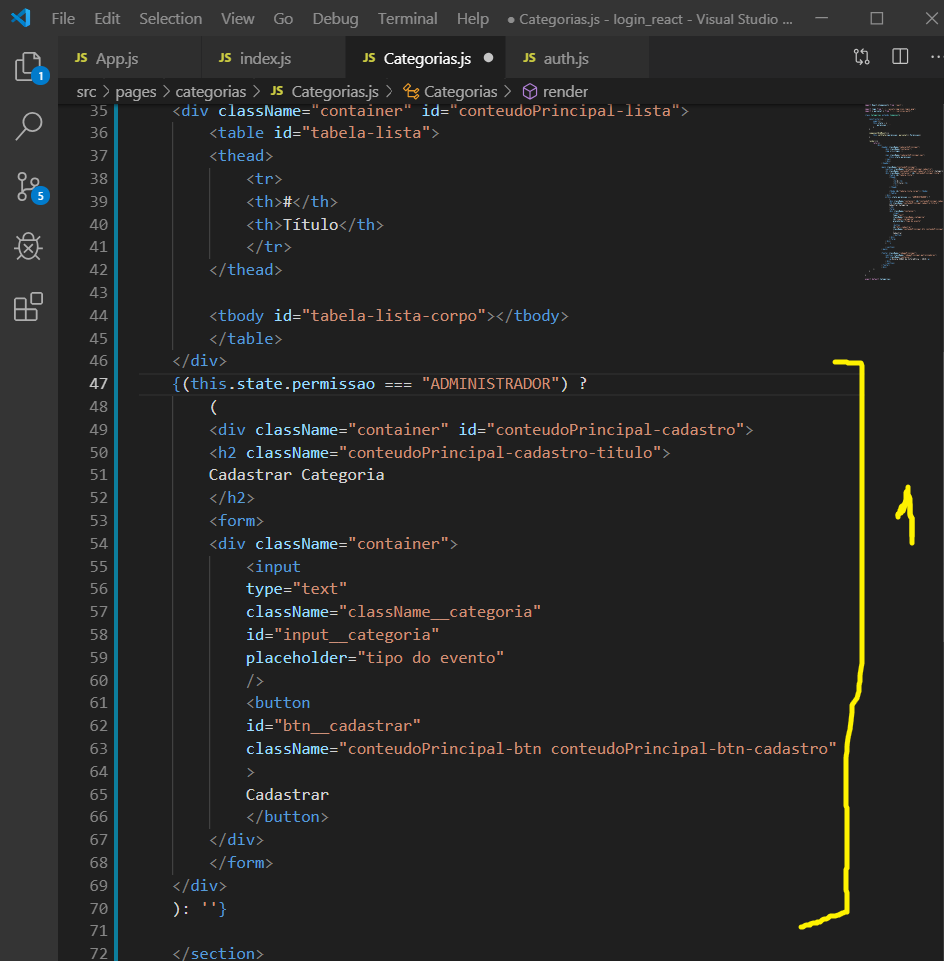
Substituir em nossa nav o texto pelo valor da nossa permissão em nosso state

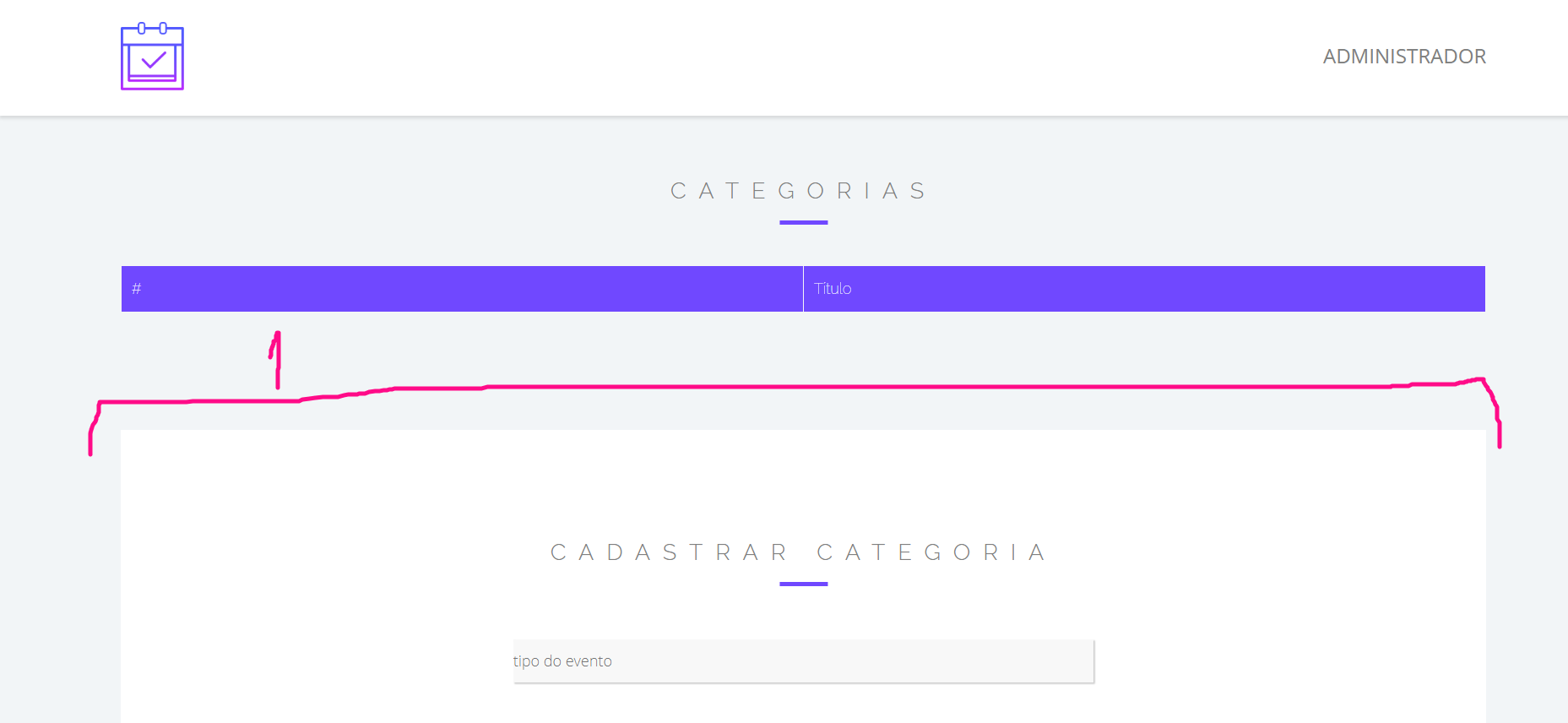




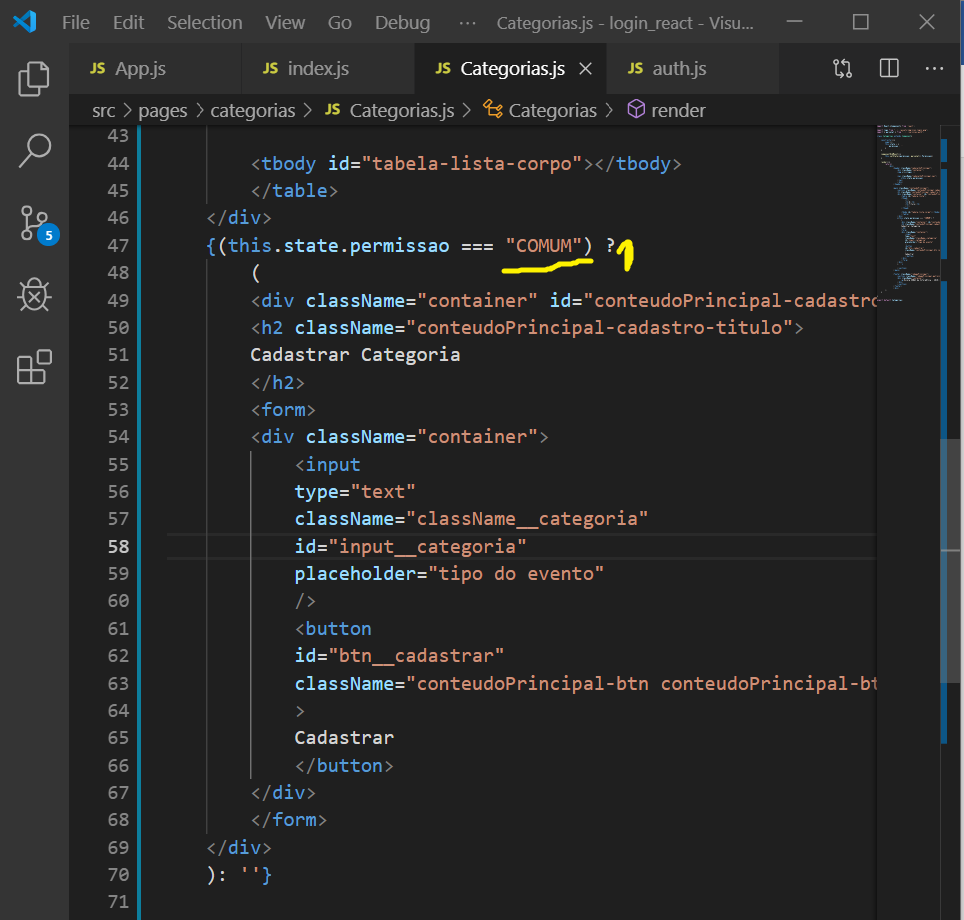
Condicionar nosso formulário para cadastro de categoria visível apenas para Administradores

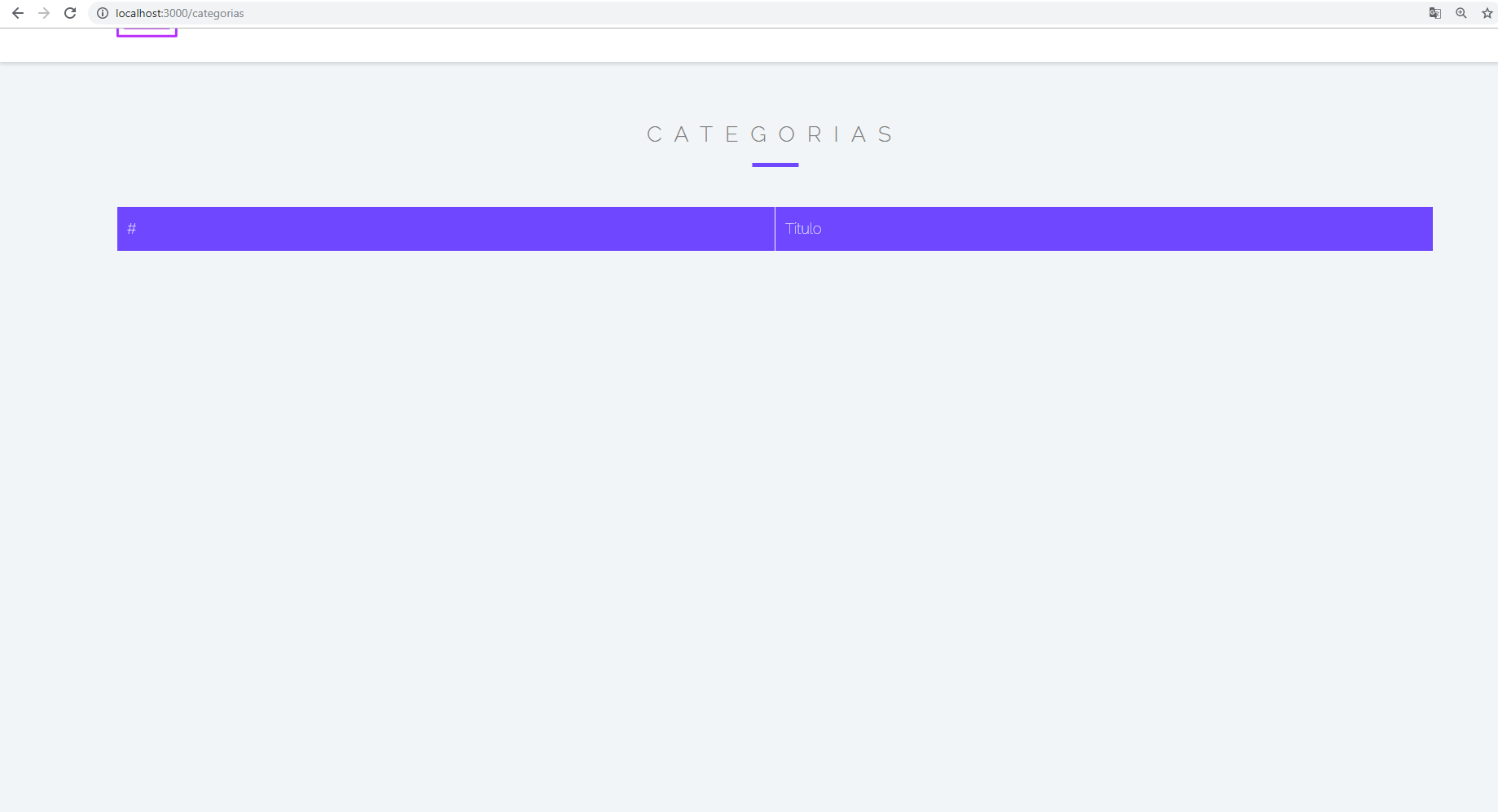
Usar operador ternário validando se o state.permissao é igual ADMINISTRADOR, caso seja a estrutura do cadastro de categoria é retornada, do contrário nada é retornado.





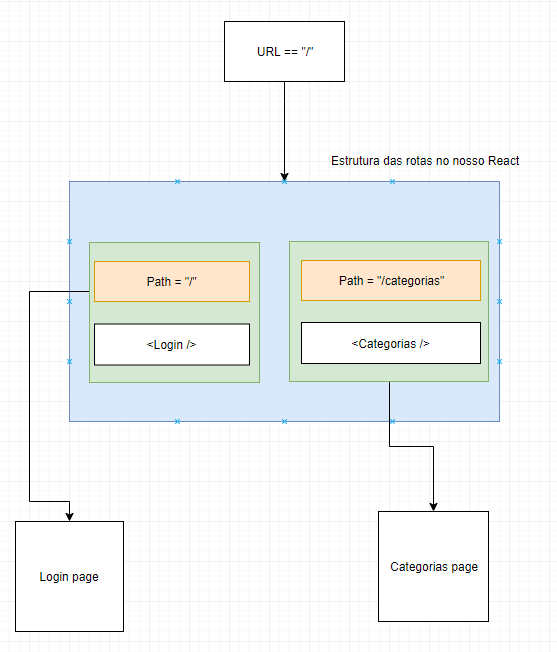
Alterando a condição para COMUM vemos que se recebermos uma condição diferente desse, nosso formulário de cadastro de categoria não é retornado.



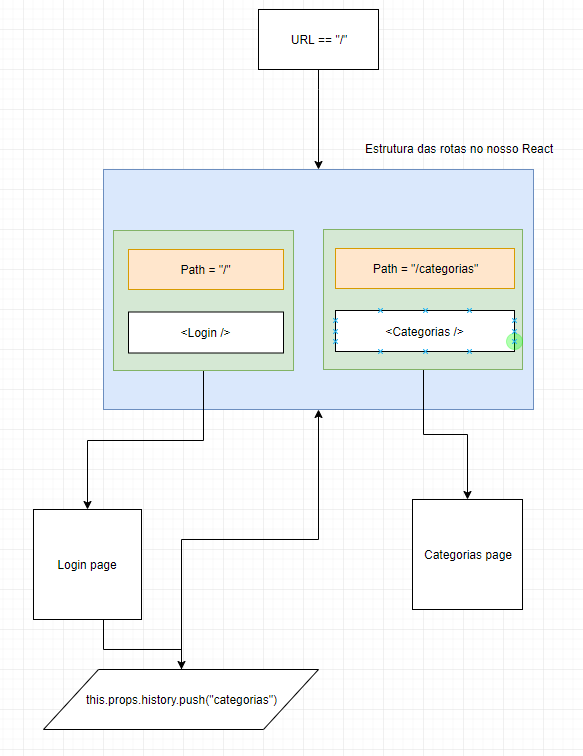


# Condicionando rotas

Quando trabalhamos com rotas temos a seguinte estrutura:

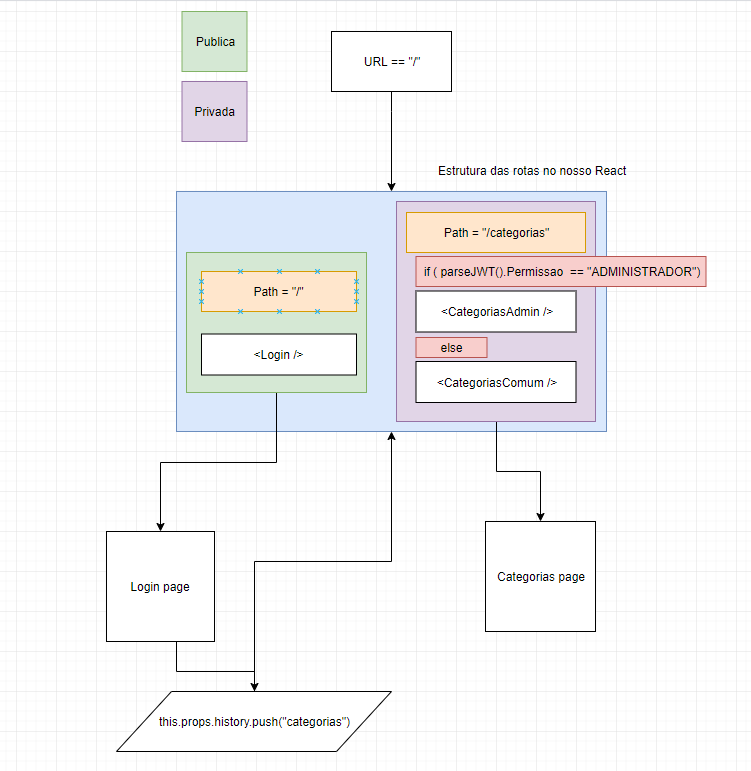


E quando realizamos um redirecionamento de um componente para o outro, do login para Categorias por exemplo, nossa estrutura de rotas checa o caminho que enviamos e nos redireciona



Dessa forma sempre que usamos um <Link /> do nosso react-router-dom, ou this.props.history.push(“/categorias”) essa solicitação passa pela nossa estrutura de rotas e é direcionado para seu respectivo componente.

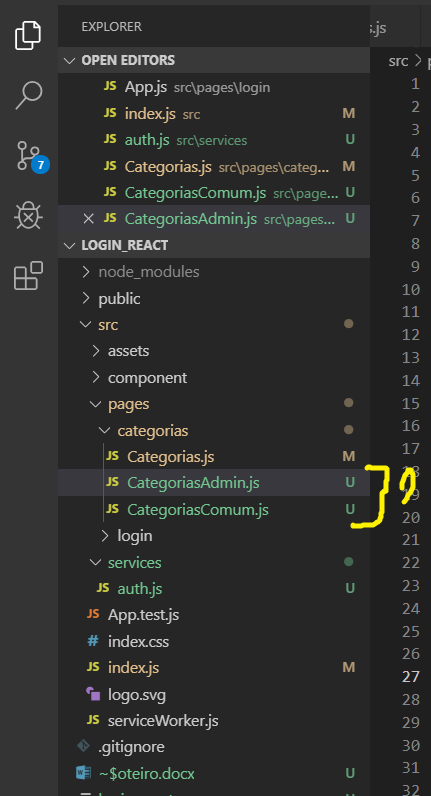
Em casos que temos uma rota privada temos o seguinte esquema:



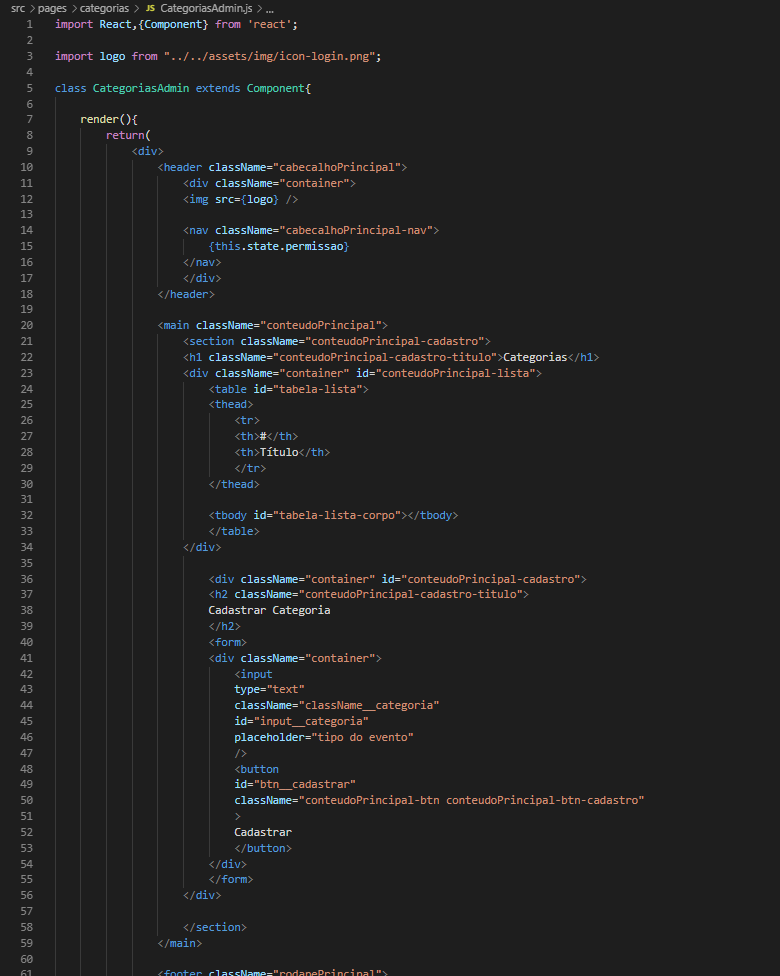
Onde o retorno do componente da rota desejada pelo usuario depende de uma outra condição, nesse caso do nosso token JWT convertido.

# Aplicando nas nossas rotas:

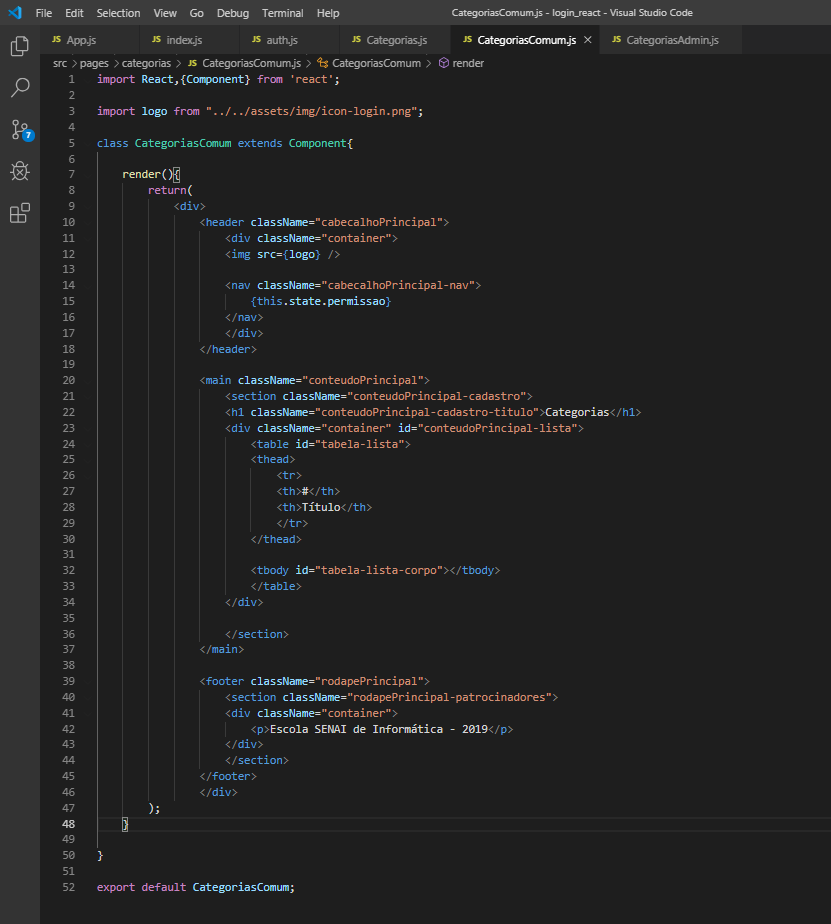
Copiar nosso arquivo categorias para um arquivo CategoriasAdmin e um CategoriasComum



Categorias Admin terá tanto a listagem das categoria quanto o formulário de cadastro delas



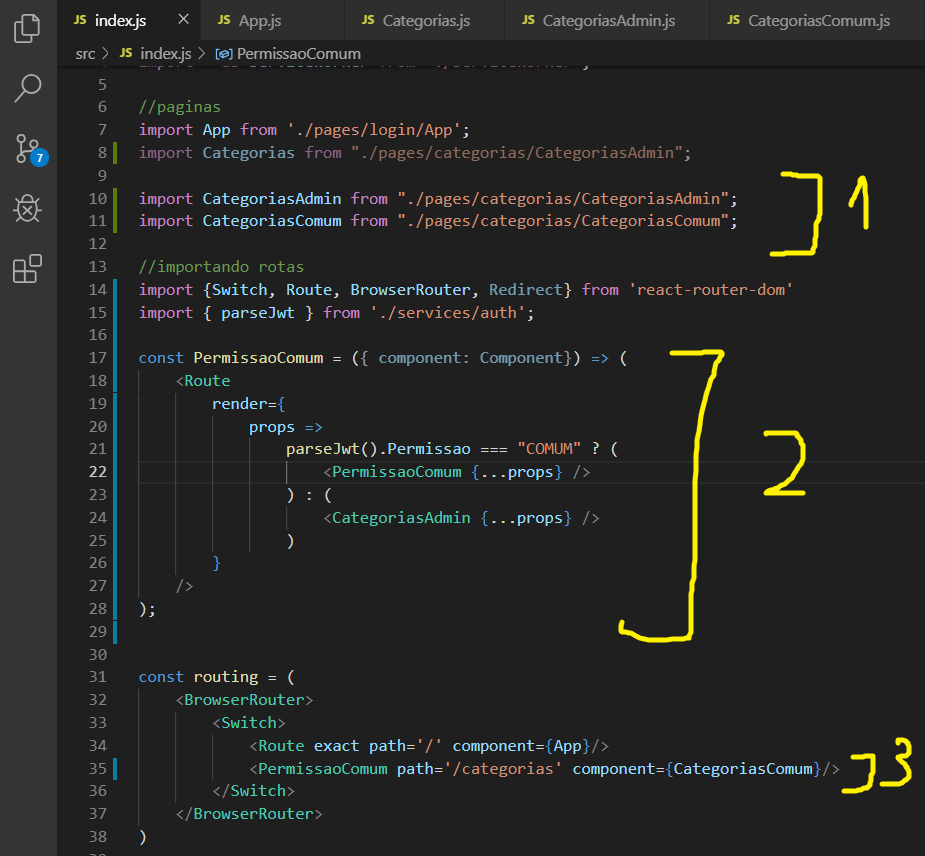
Nosso Categorias Comum não terá nosso formulário de cadastro



Vamos até nossas rotas para definir uma condicional a nossa rota /categorias

Importamos nossos dois componentes de categorias, para nosso usuário admin e usuário comum.

Criamos nossa rota privada passando no render uma condicional, onde dependendo do valor do Token na chave Permissao, que fizemos anteriormente, ele retorne um componente diferente.



Dessa forma condicionamos a renderização do componente dependendo do login do usuário.