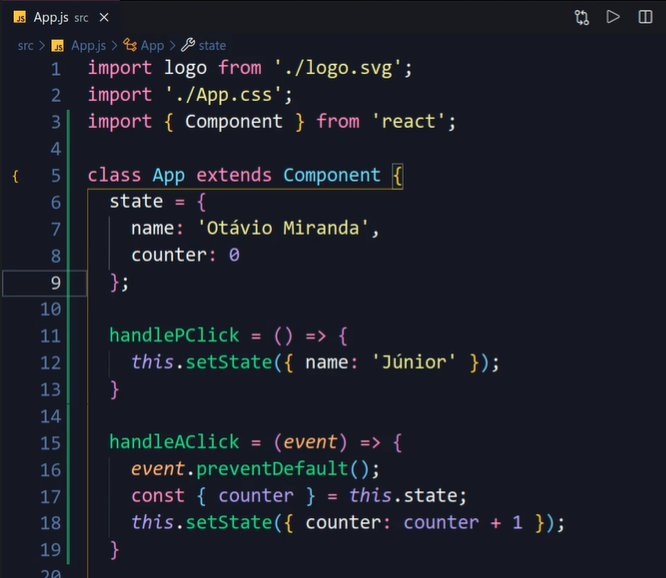
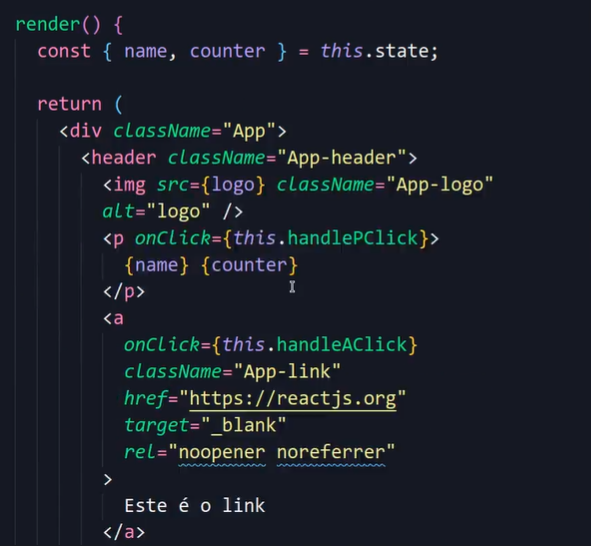
Sempre que o estado muda no react, a função render é chamada novamente.

Antesse usava uma *bind function* como abaixo para que o “this” possa ser usado em uma function:



Exemplo de classe com estado simples com arrow functions que não necessitam fazer o bind do “*this”:*  


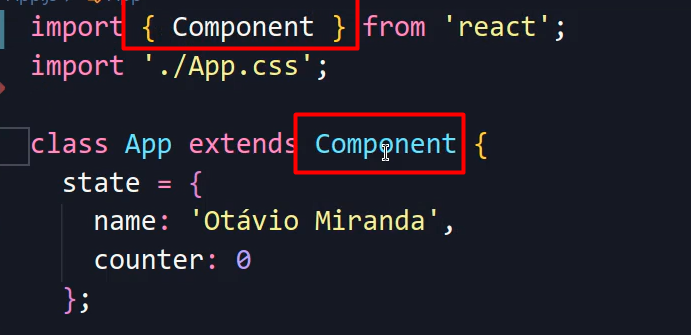




Podemos fazer os imports de duas maneiras. A primeira com a forma mais longa:



E a segunda mais usada:



Precisamos de uma chave única para cada elemento em que se usa map, exemplo:



No exemplo abaixo, incluímos um parenteses logo após da arrow function, indicando que vamos retornar algo de dentro da função mas este algo terá mais de uma linha:

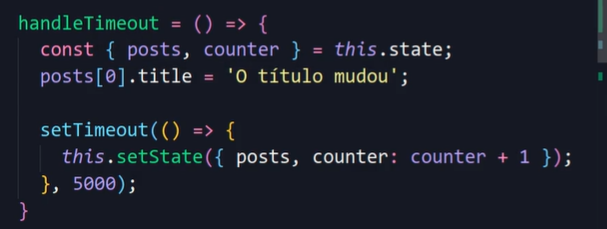


Se houverem mais elementos, a key fica no elemento mais externo (poderiam ser fragmentos):



* Sempre veremos os nomes de função começando com handle

Exemplo:

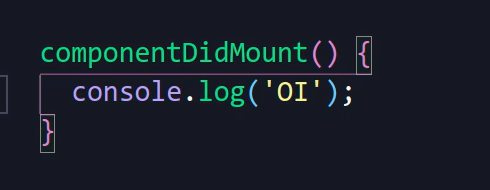


**Lifecycle Methods**

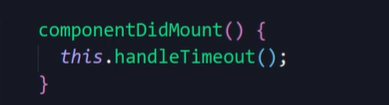
[*https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/*](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)

[*https://pt-br.legacy.reactjs.org/docs/react-component.html*](https://pt-br.legacy.reactjs.org/docs/react-component.html)

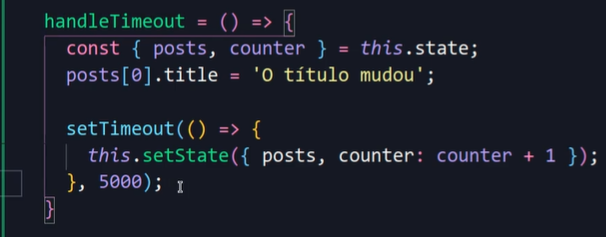
Exemplo de componente que será executado assim que o componente for montado na tela:



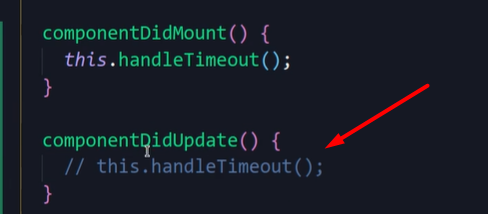
“Quando o componente for montado por inteiro, uma função será acionada”:



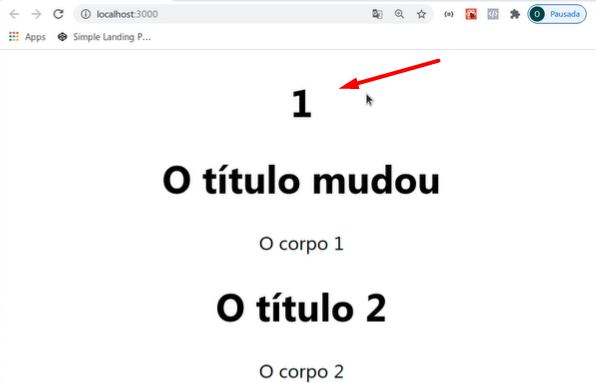
Neste exemplo, a função chamada irá atualizar o estado depois de 5 segundos:



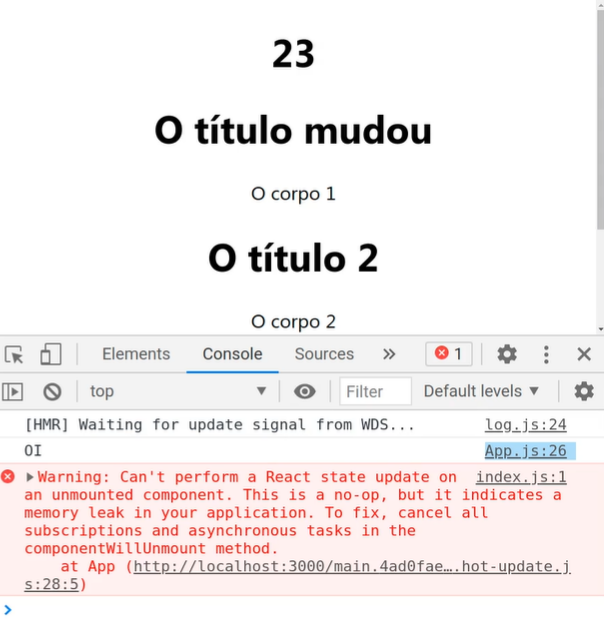
Se utilizarmos a função de ciclo de vida componentDidUpdate chamando novamente a handleTimeout, teremos um looping infinito



Neste caso queremos isto para que o contador fique atualizando o valor:



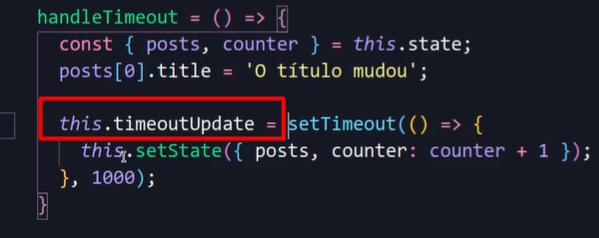
Temos que cuidar para não ficar “lixo” ao sairmos do componente com o caso acima, se for dado um reload na tela, vai acusar um erro e o estado pode não ser atualizado:



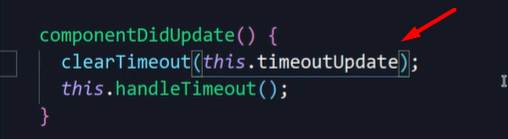
Para resolver, no caso do timeout, podemos declarar uma variável recebendo null



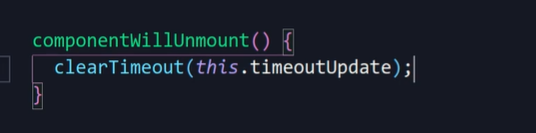
setTimeout sera atribuído a this.timeoutUpdate:



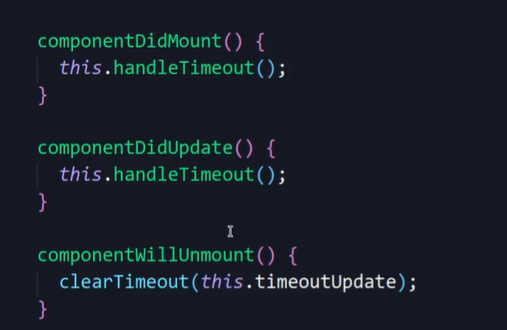
Para que tenhamos acesso a este timeout a qualquer momento, então podemos limpar o timeout:



Mesmo limpando o timeout podemos ter algum erro, para isso temos como limpar o lixo ao desmontar o componente com componerWillUnmount() e limparmos o timeout dentro desta etapa no ciclo de vida:

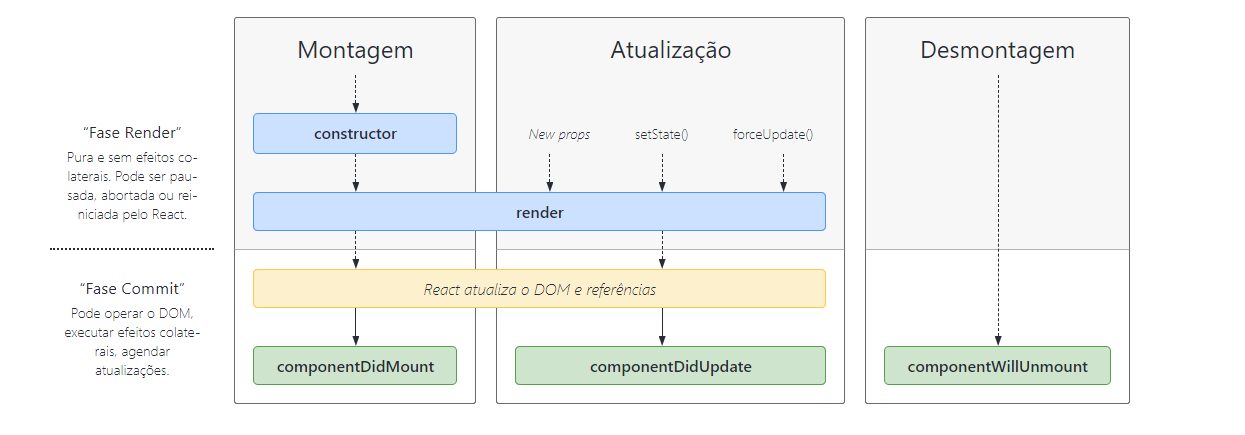


Neste caso de exemplo, usamos 3 etapas no ciclo de vida de um componente:



Ciclo de vida mais comuns utilizados:

[*https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/*](https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/)



Buscando dados de uma API

Vamos utilizar dados fake do seguinte site:

[*https://jsonplaceholder.typicode.com/*](https://jsonplaceholder.typicode.com/)

[*https://jsonplaceholder.typicode.com/posts*](https://jsonplaceholder.typicode.com/posts)

[*https://jsonplaceholder.typicode.com/photos*](https://jsonplaceholder.typicode.com/photos)

Vamos usar o fetch api para recuperar dados na internet que é Javascript puro (poderíamos usar o axios ou outro pra fetching de dados):

[*https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch\_API*](https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API)

PAREI NA AULA 19

\*Em um curso no youtube, mostraram como fazer o básico com Next com o que ele chamou de pre-rendering utilizando a getStaticProps():

Função utilizada na própria pagina que os dados serão exibidos, retorna um objeto com os dados, Next vai renderizar isso antes do acesso do usuário (pre-rendering) dando a impressão de um carregamento instantâneo pois já puxa os dados anteriormente ao usuário clicar.

[*https://github.com/matheusbattisti/curso\_nextjs\_yt/blob/main/10\_fetch\_data/pages/todos.js*](https://github.com/matheusbattisti/curso_nextjs_yt/blob/main/10_fetch_data/pages/todos.js)

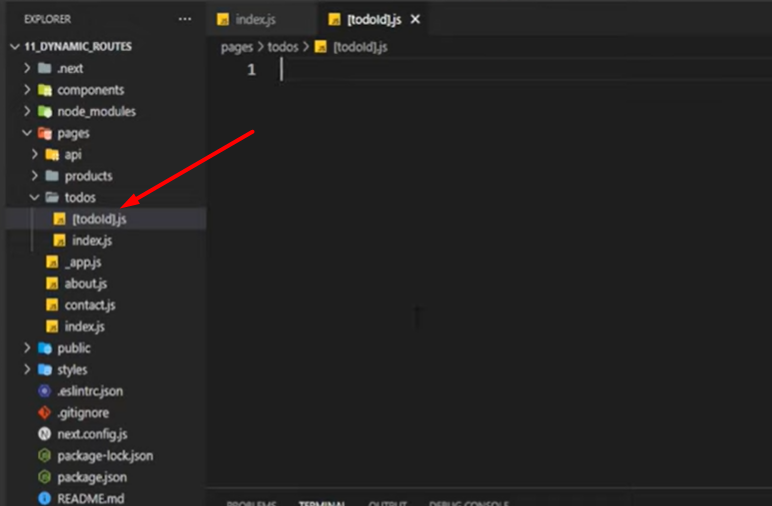
[*https://www.youtube.com/watch?v=v8vJawWOYkM&ab\_channel=MatheusBattisti-HoradeCodar*](https://www.youtube.com/watch?v=v8vJawWOYkM&ab_channel=MatheusBattisti-HoradeCodar)



Rotas dinâmicas geradas pelo Next

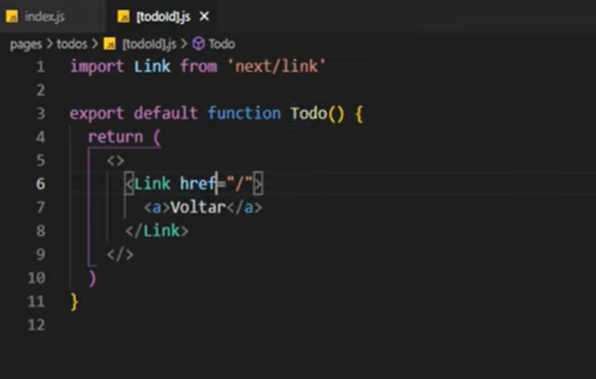
[*https://www.youtube.com/watch?v=iWpLD2wTsZ4&ab\_channel=MatheusBattisti-HoradeCodar*](https://www.youtube.com/watch?v=iWpLD2wTsZ4&ab_channel=MatheusBattisti-HoradeCodar)

Criar arquivo no padrão [id].js (id pode ser outro nome)

Na pratica:  


Assim como está abaixo não vai dar erro, vão abrir paginas em branco no caso de usar a url tipo

localhost:3000/todos/4



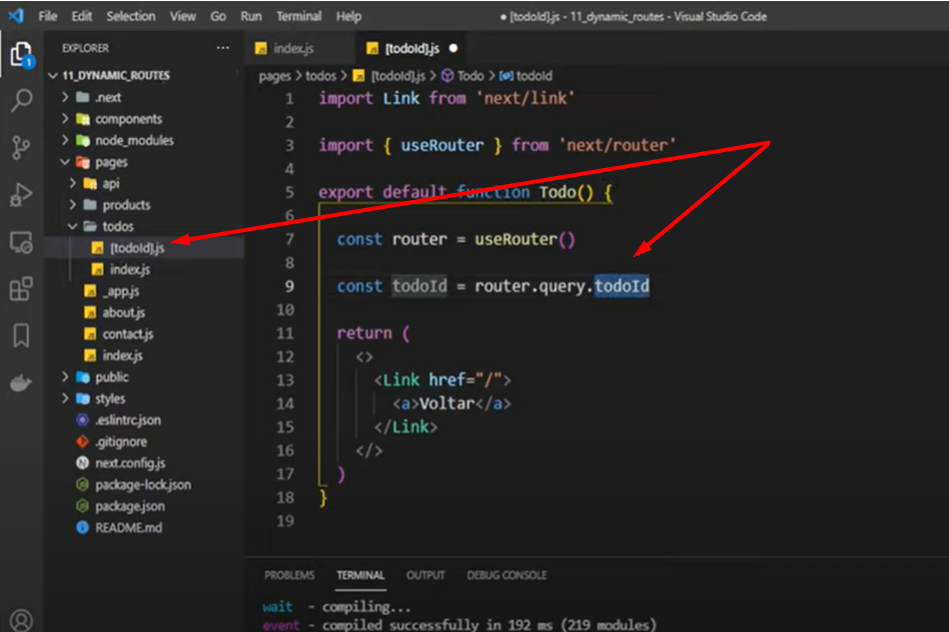


Para seguir, temos que importar um next/router e instanciar para pegar via url

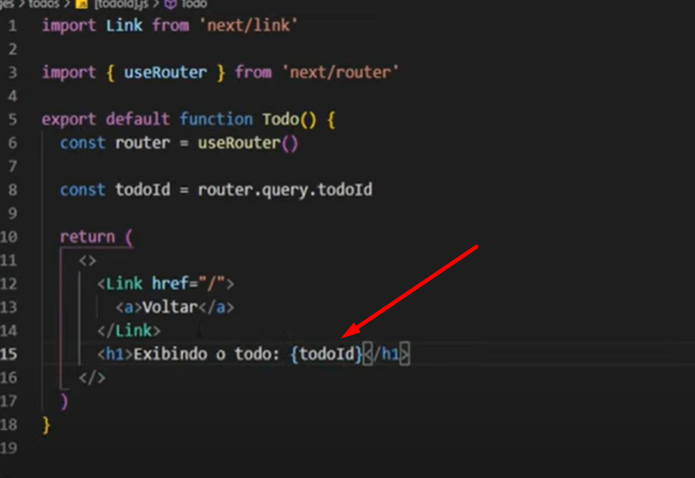
Bom exemplo de router:

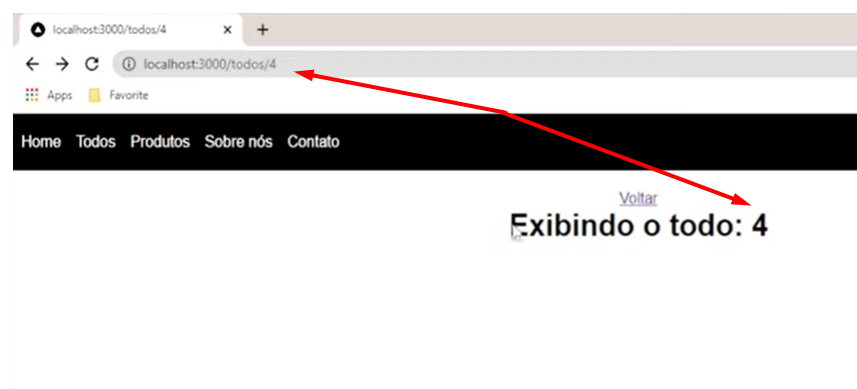
<https://nextjs.org/docs/pages/api-reference/functions/use-router>

Além disto, o Next associa automaticamente o nome do arquivo com o nome do parâmetro utilizado na rota:

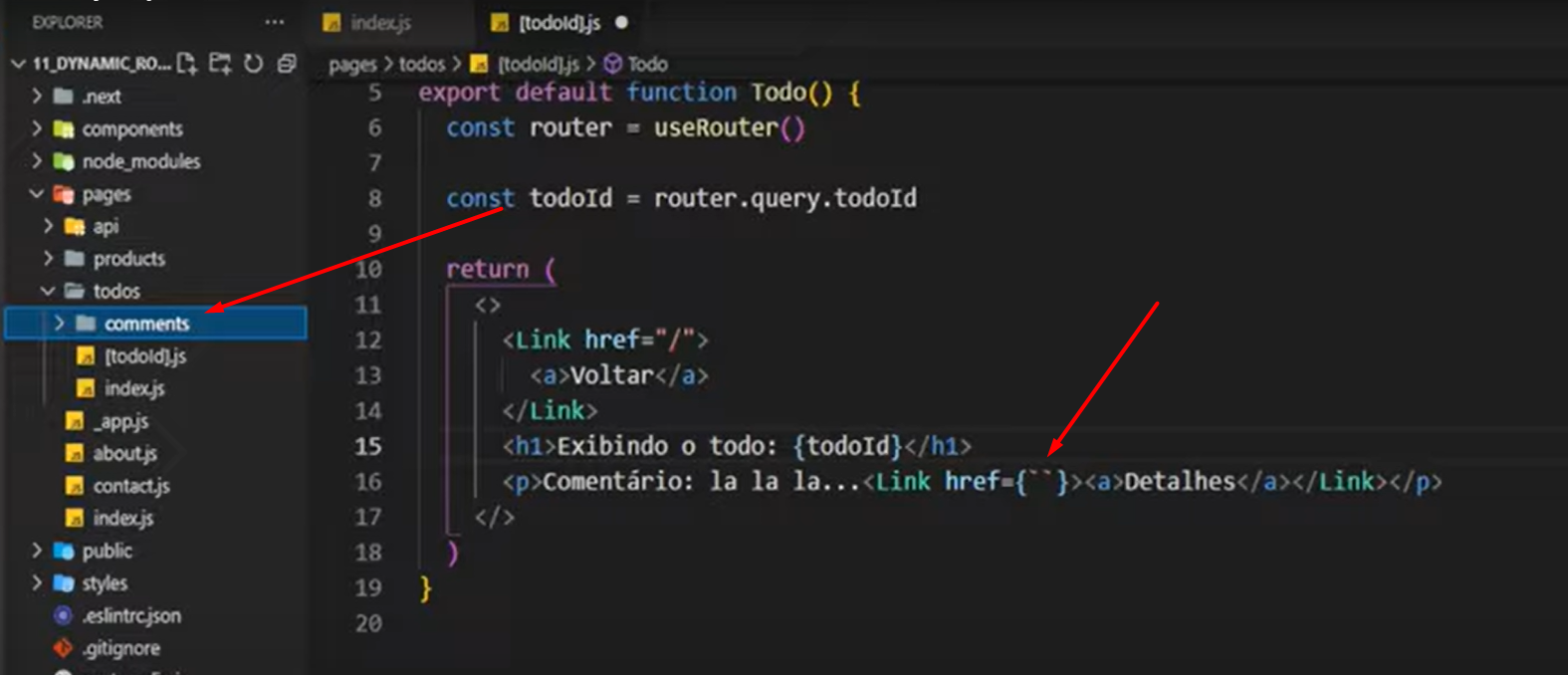


Podemos exibir o ID para ter uma ideia melhor de como funciona:

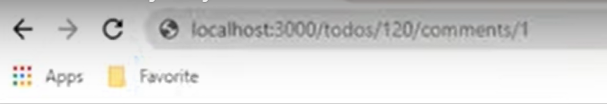




Se, por exemplo tivéssemos comentários dentro dos todos, poderíamos fazer um link para paginas dos de cada comentário criando um arquivo “comments” para a rota



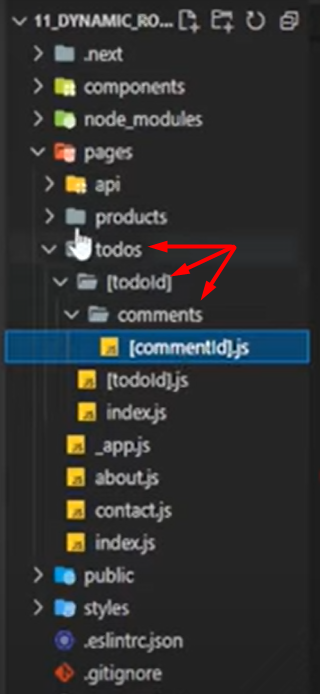
Na url vai ter algo como isto:



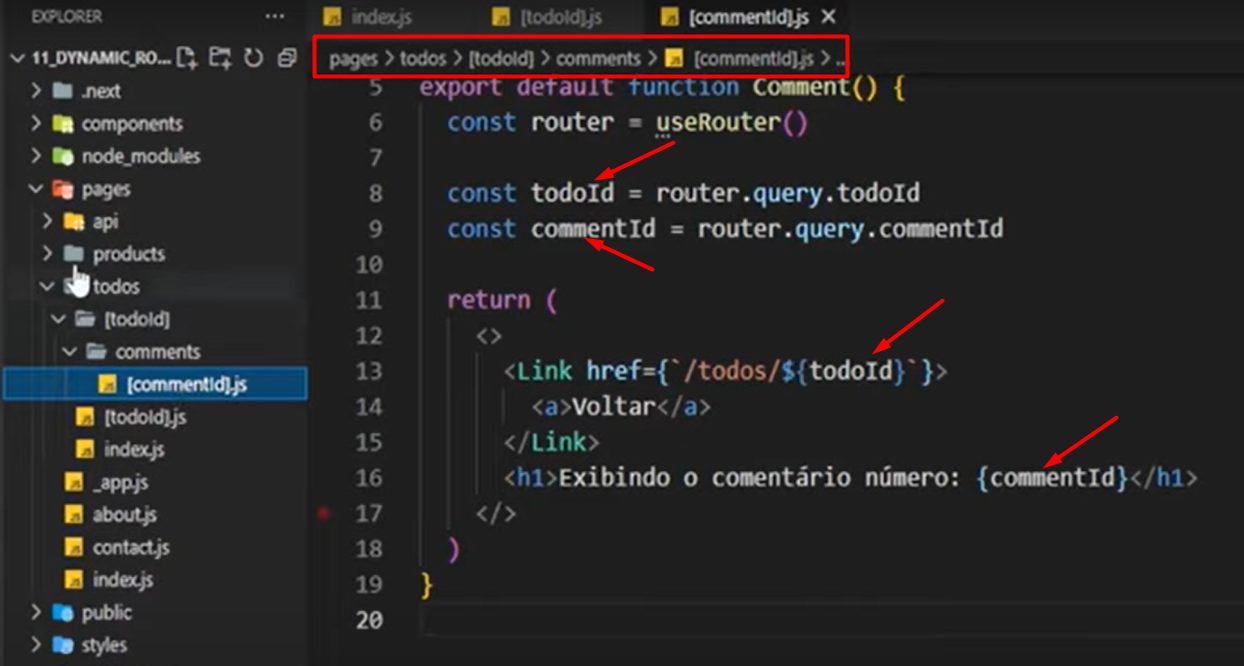
No exemplo abaixo os ids estão fixos para testes:



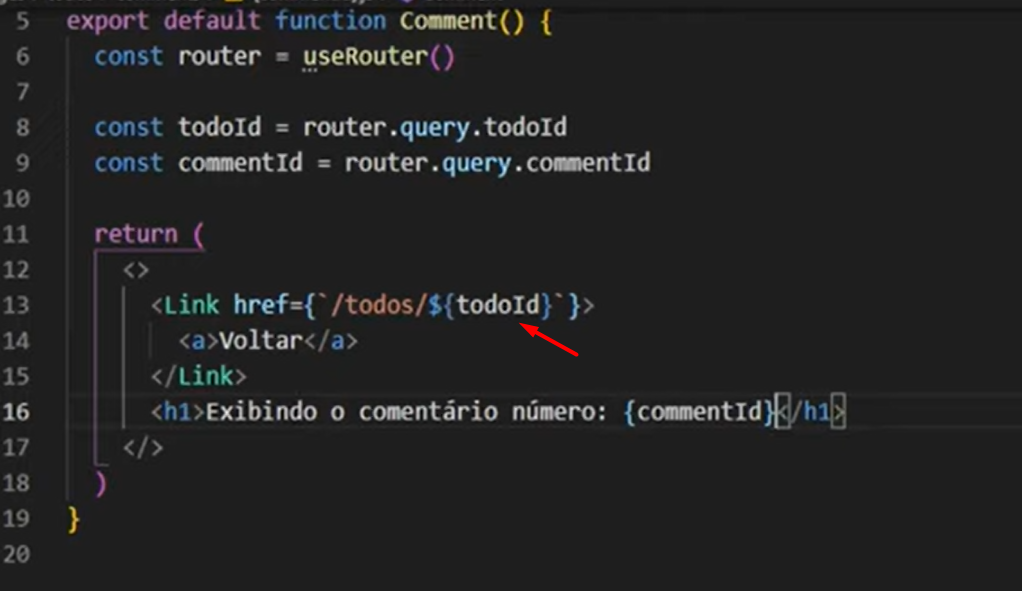
Observar que esta pasta está dentro da outra, no caso temos que criar mais uma pasta [todoid] e colocar a comments com a [commentid] dentro:



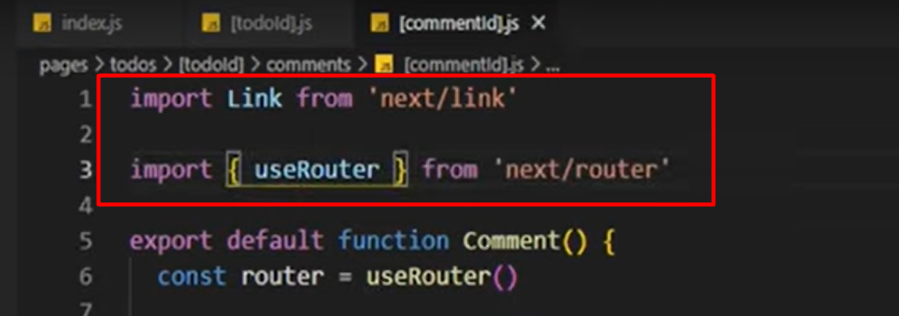
Desta maneira conseguimos pegar inclusive o Id do “todo”:



Assim podemos voltar a rota anterior:



Lembrando de importar sempre o useRouter:



Rotas dinâmicas com dados reais

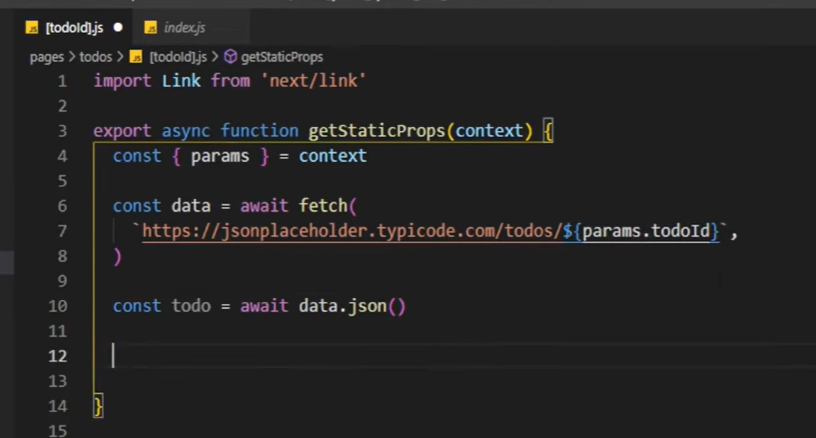
vamos utilizar o método getStaticProps que mapeia e busca os dados

e getStaticPaths para geração dinâmica das paginas HTML. (*Se exportarmos uma função chamada getStaticProps (geração de site estático) de uma página, o Next.js pré-renderizará essa página no momento da compilação usando as props retornadas por getStaticProps*.)

[*https://nextjs.org/docs/pages/building-your-application/data-fetching/get-static-props*](https://nextjs.org/docs/pages/building-your-application/data-fetching/get-static-props)

-> Vamos mexer basicamente no arquivo [todoid].js

Vamos iniciar removendo o router que não será mais necessário pois traremos os ids direto no fetch (exemplo abaixo já na função *getStaticProps*):



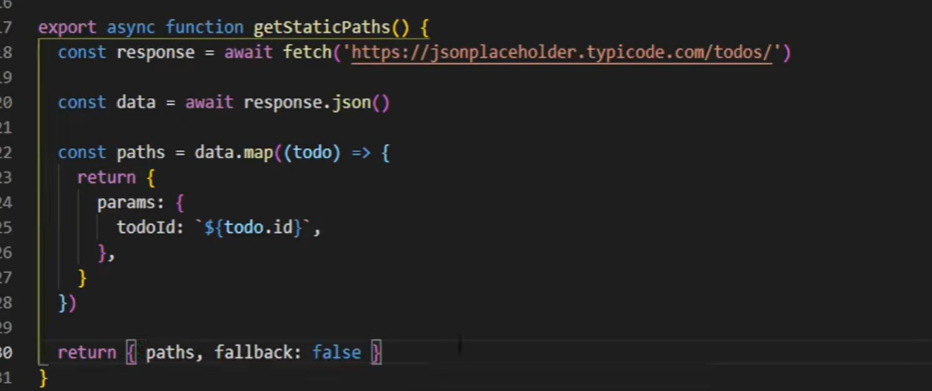
Temos que retornar um objeto com o props passando o todo para estar disponível mais abaixo:



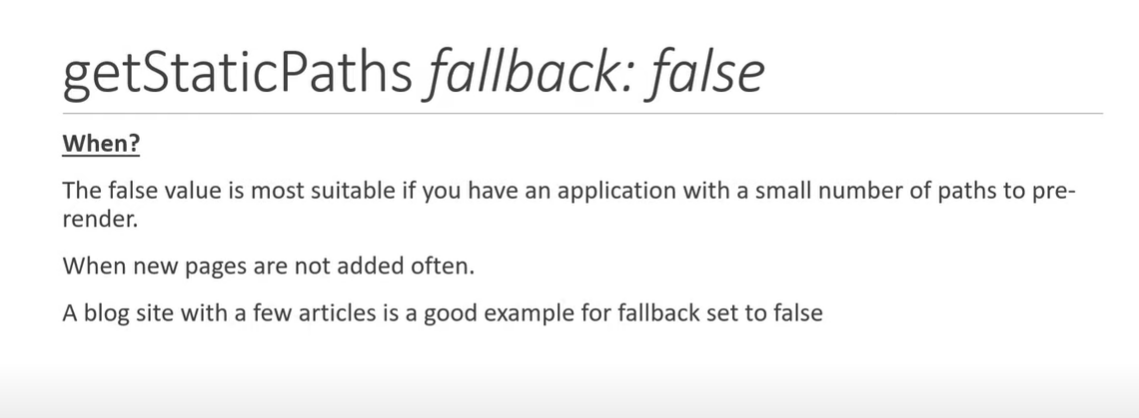
Para trabalhar com essa parte dinamicamente temos que usar o getStaticPaths fazendo o mapeamento de tudo que está disponível em uma determinada rota de uma api:



Então vamos retornar todos os Ids como paths por meio desta função.   
Normalmente se usa uma função de array map acessando os todos individualmente e vamos relacionar com o nome do arquivo [todoid] dentro do objeto params



retornando os paths e o fallback como false (com o false renderiza tudo pelo backend e para aparecer algum novo produto, o projeto terá q ser rebuildado para o produto aparecer e se usarmos o tru, a renderização de coisas novas serão carregadas na hora q acessa a pagina).



Links:  
curso de react basic – <https://www.youtube.com/watch?v=9P8mASSREYM&list=PLC3y8-rFHvwgC9mj0qv972IO5DmD-H0ZH>

Video mostrando o ecosystem (apps para ambiente) pra trabalhar em react em 2030:

[*https://www.youtube.com/watch?v=6j9tnGMbm2c*](https://www.youtube.com/watch?v=6j9tnGMbm2c)