Roteamento é o que em geral os servidores fazem. Uma máquina faz uma requisição para uma Home Page, e o servidor responde com o conteúdo da Home Page.

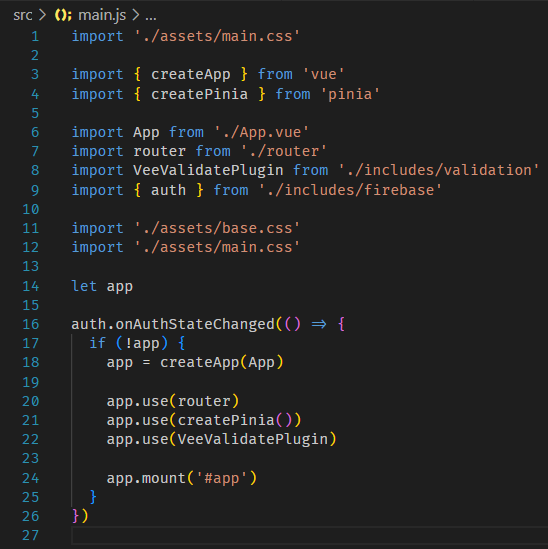
**ENTENDENDO O ROTEAMENTO**

O usuário faz uma requisição para um componente, este faz uma requisição ao server se for uma requisição envolvendo dados, o servidor responde com os dados, o componente renderiza e mostra para o usuário. Isso faz com que se economize largura de banda, e também alivie as coisas para o lado do servidor. Isso representa uma grande economia de recursos se for comparar com uma requisição direto para o servidor em que uma página toda precise ser renderizada por causa de uma simples modificação. Isso torna as páginas mais rápidas e também oferece uma melhor experiência para o usuário final.

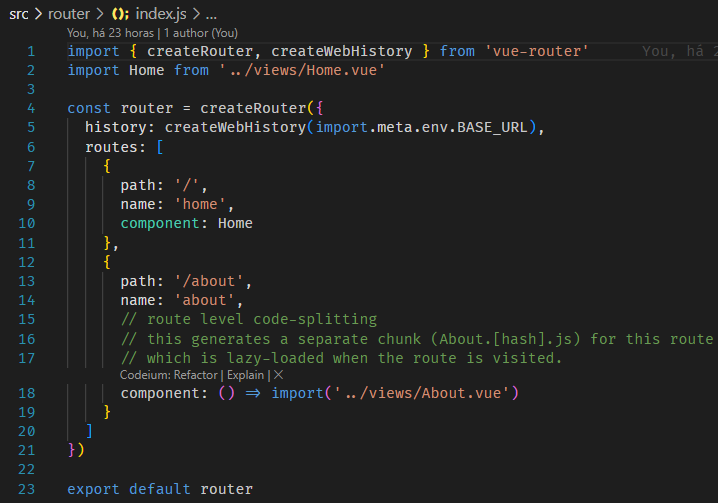
Esse é o pacote oficial do vue: <https://router.vuejs.org/guide/>

**REVISANDO A CONFIGURAÇÃO DO ROUTER, PACOTE OFICIAL DO VUE**

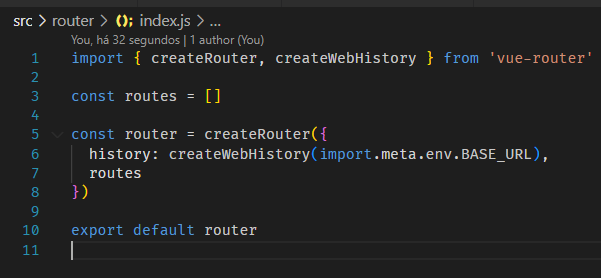
Vamos lá, primeiro o router precisa ser importado no main.js:



Num segundo momento, vamos abrir o index.js, dentro da pasta router:

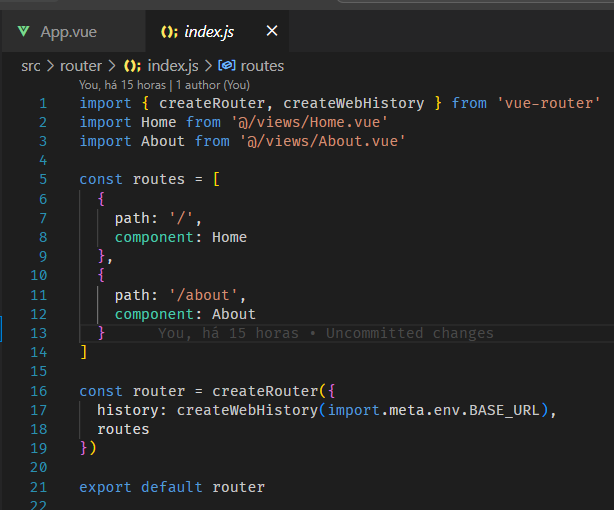


Nesse arquivo, tem 3 coisas bem importantes. As duas primeiras são as funções criadas, o createRouter e o createWebHistory. A terceira coisa é o componente Home. Apaguei algumas linhas e deixei assim:



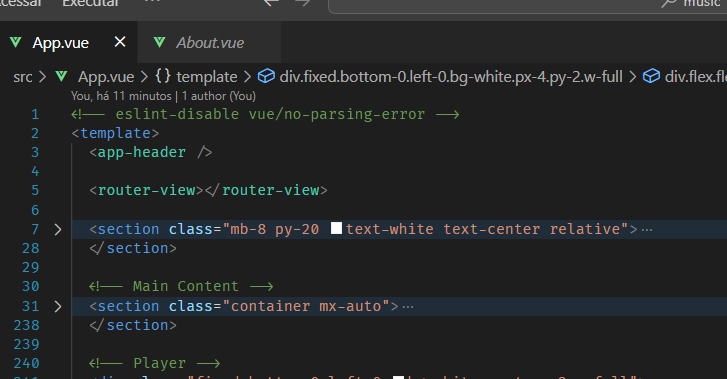
**CRIANDO ROTAS**

O roteamento permite dizer ao vue qual conteúdo queremos exibir em determinado local, de maneira dinâmica. O primeiro passo é definir as rotas no arquivo index.js em router:



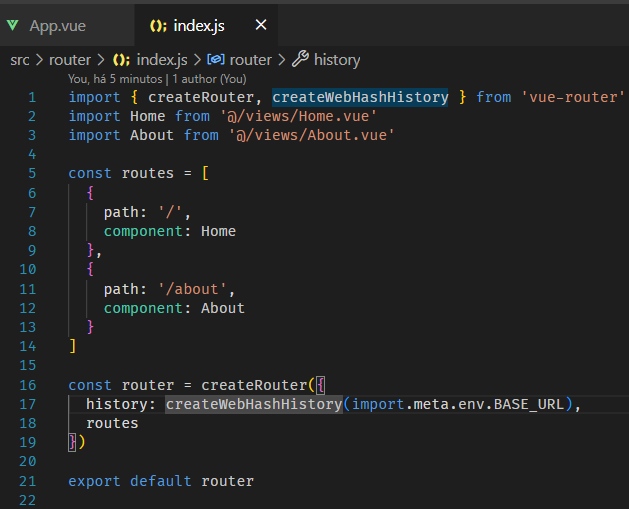
Neste caso, estamos fazendo o roteamento da Home e da About. O próximo passo é organizar esses componentes. Então na pasta views, eu crio a home e about e jogo o html dentro dos templates.

Por fim, eu chamo no app.vue o router-view, que de acordo com as rotas cadastradas no index.js, ele saberá o que deve exibir nessas tags, de acordo com a rota:



**MODO HISTÓRIA**

* createWebHashHistory



Esse modo adiciona um hash (#) na url. A página não dá refresh, mesmo assim exibe/esconde os conteúdos como se tivesse recarregado para outra página. Caso você já tenha usado o confluence, esse comportamento é similar à âncora. Pode ser útil em navegadores antigos ou em situações em que um você precisa mostrar determinado componente, mas não pode recarregar a página.

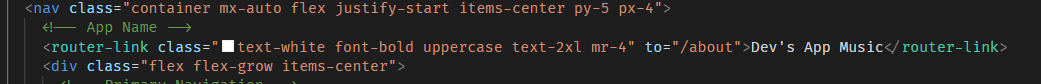
* **Memory**

Quando você quer manter as alterações da rota rastreadas internamente sem modificar a barra URL.

Links úteis: <https://router.vuejs.org/guide/essentials/history-mode.html> | <https://developer.mozilla.org/en-US/docs/Web/API/History_API>

**NAVEGANDO COM LINKS**

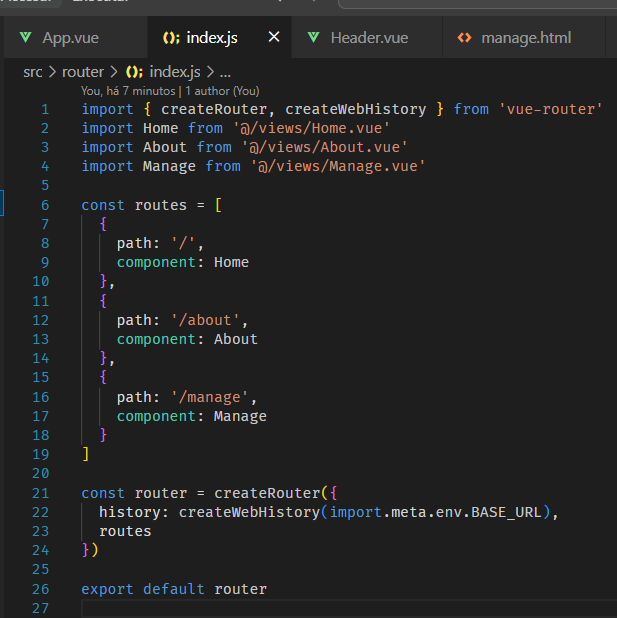
Utilizamos o componente router-link, que possui uma propriedade chamada ‘to’:



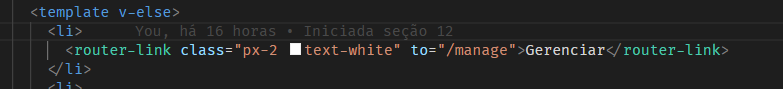
**CUSTOM LINKS**

Para criar links customizados, o passo a passo é o seguinte:

Dentro da pasta src, cria uma pasta se ainda não existir chamada views, e ali cada arquivo.vue é um componente de rota. Em seguida, no arquivo index.js, dentro do router, criar as rotas desejadas:

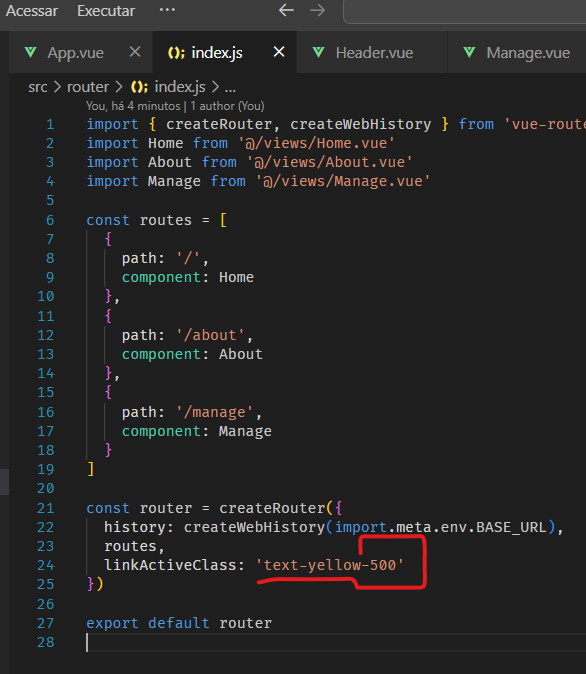


Perceba que as importações, além do createRouter e webHistory, são dos componentes das rotas. Para cada rota, tem que colocar o path e o component atrelado.

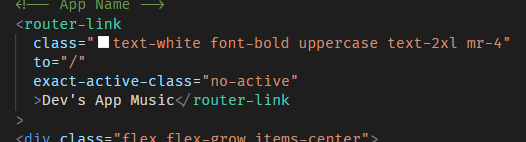


**TAILWIND STYLES FOR ACTIVES LINKS**

É um efeito interessante destacar no menu o link que está ativo. Para isso, existe uma propriedade que pode ser passada no index.js do router:



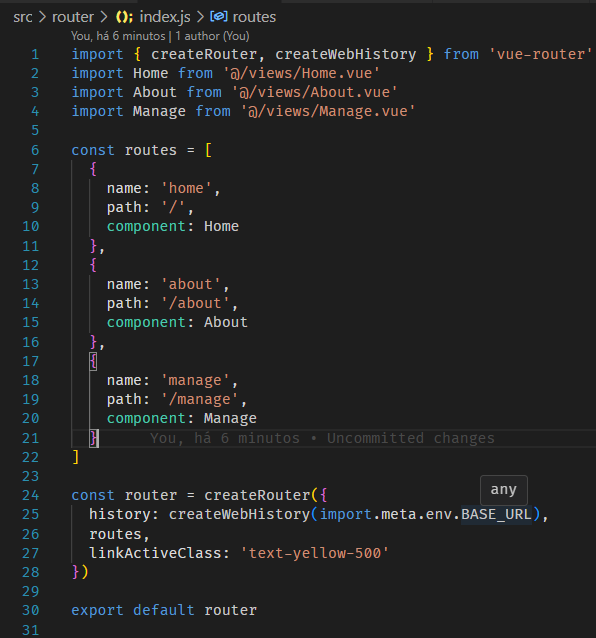
Se algum link do menu não queremos que seja afetado, podemos fazer o seguinte:

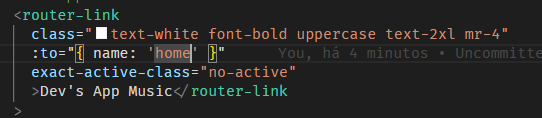


A propriedade exact-active-class.

**NOMEANDO ROTAS**

As rotas podem ser nomeadas, adicionando a propriedade name lá no index.js do router:



O próximo passo será atrelar esse nome para as opções desejadas no header, ou qualquer outro link:

Então, modificamos o to para ser uma propriedade bind, e dentro passamos o nome. Talvez nesse ponto você se questione, “mas dessa maneira, escrevo mais código do que antes”. Nesse caso, é um mal necessário. Imagine que você tenha 30 locais contendo link para a página principal, e por algum motivo você precise alterar a rota: você teria que ir nesses 30 locais e atualizar a rota. Se utilizar o nome, você pode fazer quantas alterações forem necessárias na rota, que não precisará alterar o to.

**SETTING UP CATCH-ALL AND REDIRECT ROUTES**

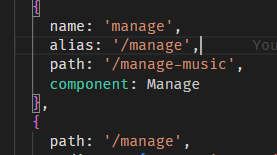
Pense no seguinte: você alterou uma rota de uma página do seu site, mas um usuário seu acessava digitando o caminho na barra de endereço, ou já estava salvo para ele o caminho antigo. Para não deixá-lo no limbo, podemos fazer duas coisas interessantes:



Adicionamos rotas de redirecionamento. No caso da /manage, o path alterou para /manage-music, então se o usuário digitar /manage, irá redirecionar para a rota com name ‘manage’, e no caso da última rota, se o usuário digitar qualquer rota que não esteja cadastrada, automaticamente irá redirecionar para a home.

**ROUTE ALIAS**

Adicionar a propriedade alias irá permitir que determinada rota seja acessada também por ele:

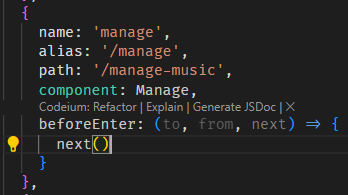


**GUARDING ROUTES**

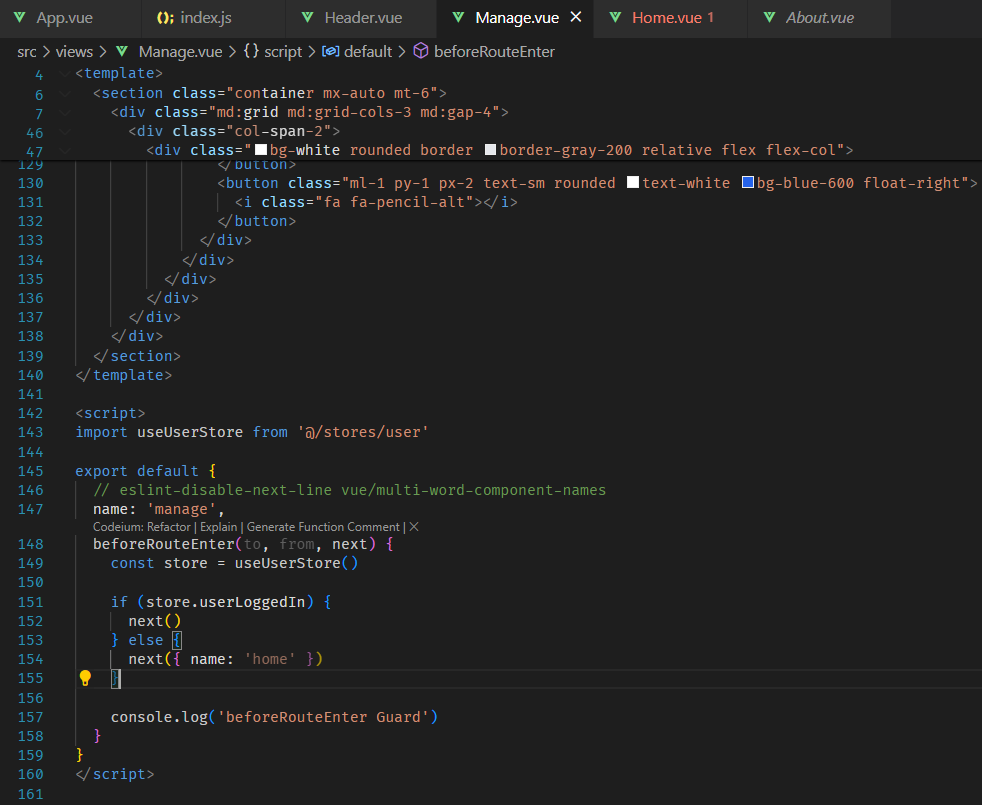
**ROUTE SPECIFIC GUARDS**

**GUARDING AUTHENTICATION ONLY ROUTES**

Utilizando a seguinte ideia: vamos dizer que ao deslogar, queremos que fique na página home, faríamos o seguinte:



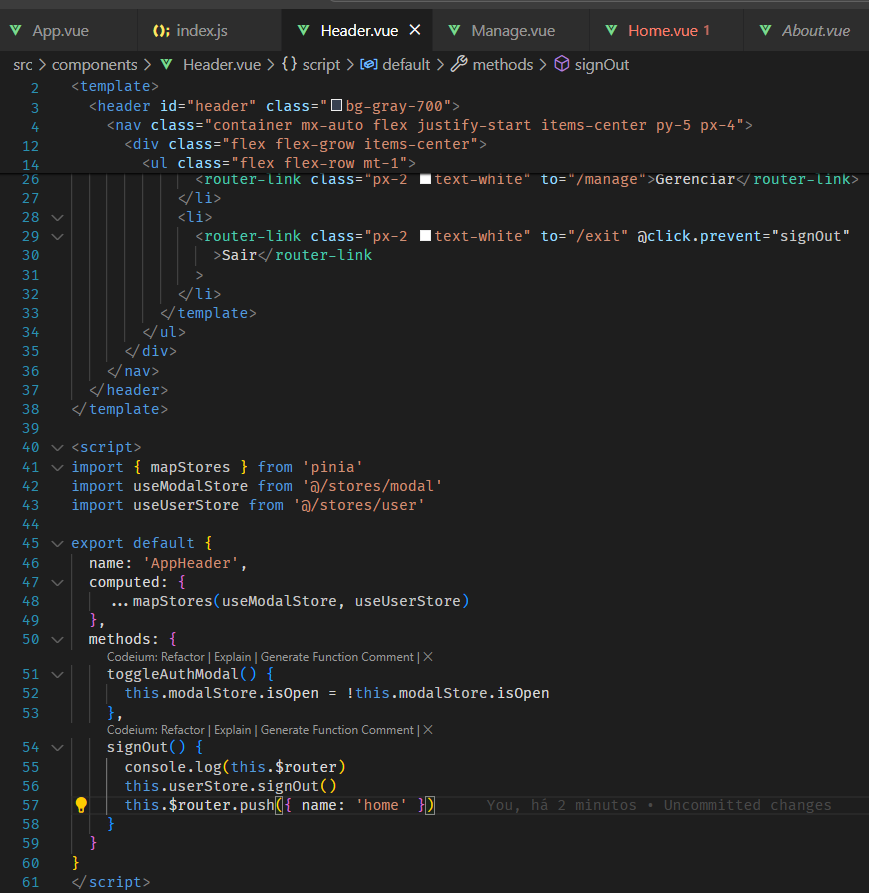
Adicionamos o beforeEnter, e lá no componente do manage, adicionamos essa mesma função definindo o que irá acontecer.



O que ocorre é que no componente Manage estamos importando a store de user para poder fazer uma validação dentro da beforeRouteEnter: se o usuário estiver logad, irá dar um next, ou seja, irá de fato entrar na rota /manage. Senão, irá chamar a rota com name ‘home’. Por isso, tentar forçar o acesso à página /manage, sem estar logado irá redirecionar automaticamente para a home.

**REDIRECTING AFTER LOGGING OUT**

Para fazer com que o usuário seja redirecionado após o logout, adicionamos essas configurações:



Vamos à explicação: o método signOut é chamado ao clicar em sair. Em seguida, esse método chama o signOut que está dentro do userStore, e em seguida dá um push no router para a rota com name ‘home’. Isso faz com que ao deslogar, o usuário seja direcionado para a home, independentemente de onde ele esteja.

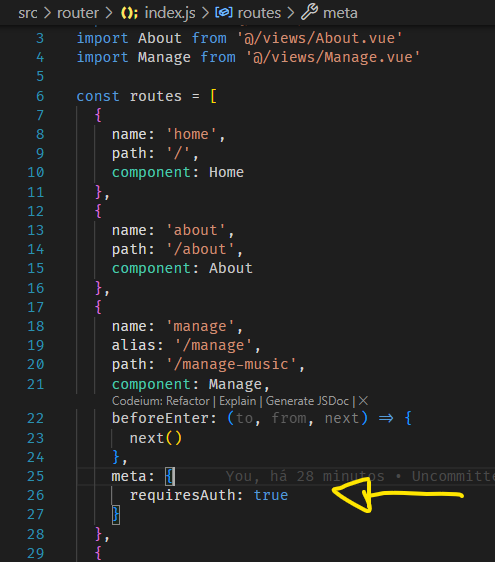
Além disso, é possível fazer outro comportamento: se você quiser que ao sair de uma página específica ele vá para um local específico, basta colocar uma validação nesse sentido:

if (this.$route.name === ‘manage’)

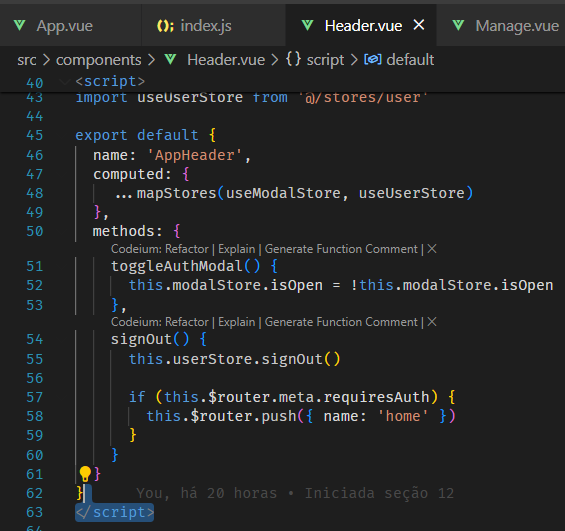
this.$router.push({name:’home’})

**ROUTE META FIELDS**

Se existirem muitos campos para validar, o método acima pode ser muito trabalhoso. Nesse caso, existe uma propriedade chamada meta que pode ser atribuída diretamente para as rotas, veja um exemplo:



Essa propriedade permite que seja feita a seguinte validação. Se a rota de onde foi saído, requer autenticação, vai direcionar para a home, senão não.



Para tornar mais dinâmico o processo, podemos fazer isso acima de maneira global:



Basicamente, o código acima está no index.js do router, e sempre antes de fazer um direcionamento, a função beforeEach vai verificar se a rota para a qual está indo requer autenticação, e se o usuário está logado ou não.