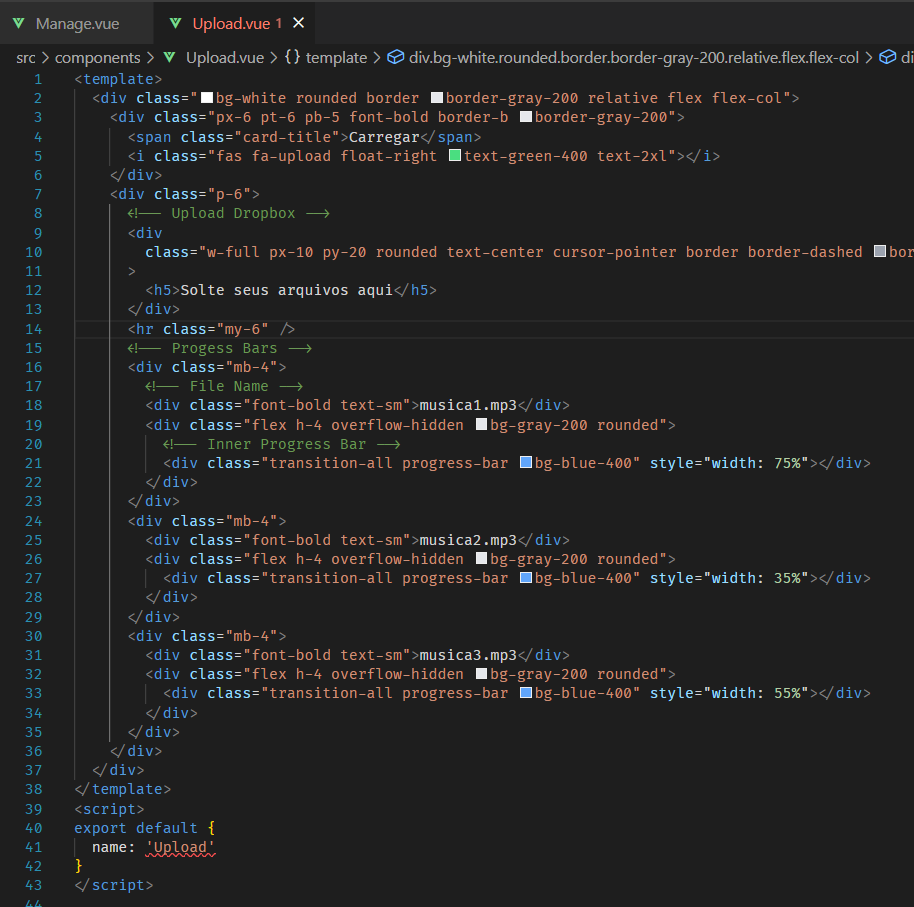
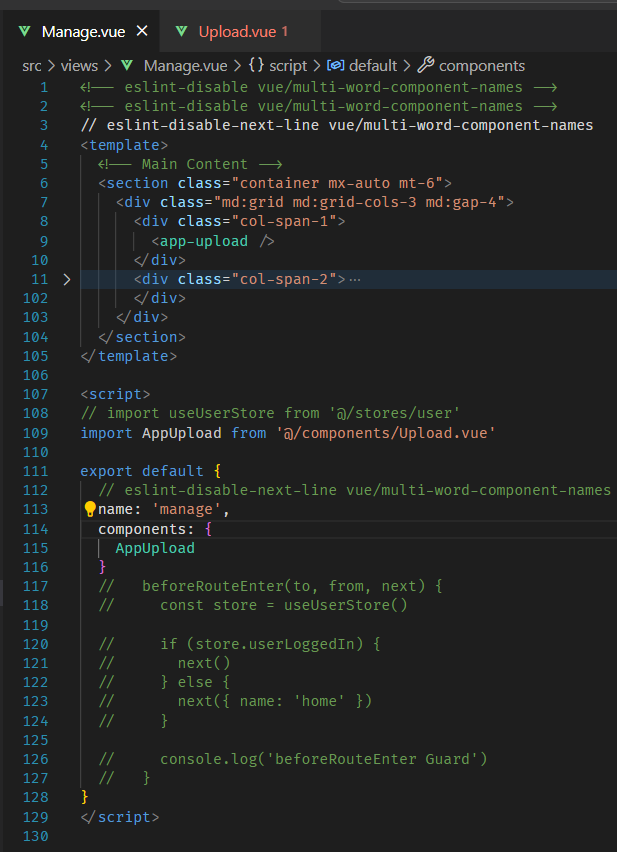
*Entendendo como fazer o upload de files com Vue.*

**PREPARANDO COMPONENTE PARA UPLOAD**

O upload de arquivos é uma responsabilidade entre cliente e servidor: o lado do cliente transfere para o servidor o arquivo e providencia uma forma para que o arquivo seja acessado. Por outro lado, o servidor valida o upload, armazena o arquivo, verifica as permissões do arquivo e uma cria uma API para o cliente para o envio do arquivo.

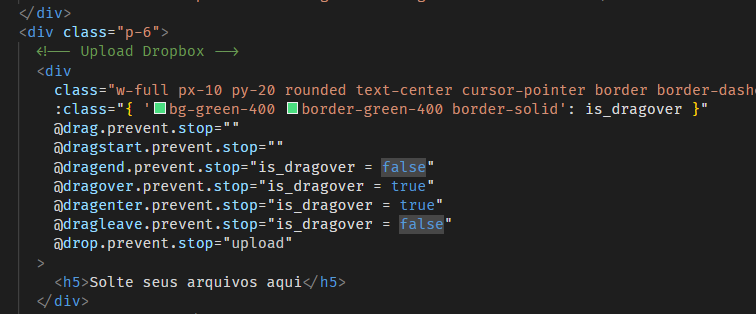
O primeiro passo é montar o componente e colocá-lo onde ele irá aparecer:



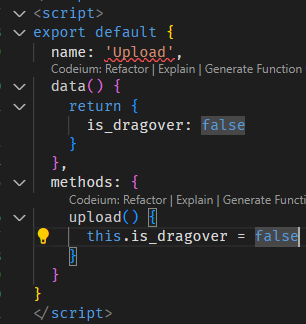


**LIDANDO COM EVENTOS DE ARRASTAR E SOLTAR**

Um efeito interessante é alterar o comportamento do local onde são soltos os anexos: quando o usuário passar o mouse ou arrastar algum item por cima do local, ele irá mudar de cor, e quando soltar deve voltar ao normal. Para isso:

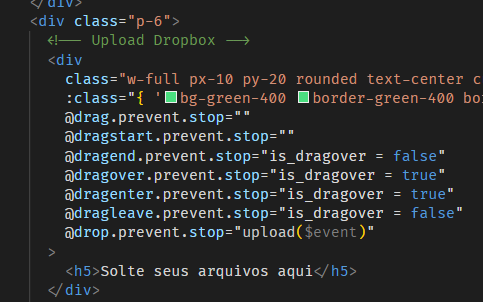


Os @ adicionados na div são para prevenir o comportamento padrão dos navegadores para esse tipo de ação, e o valor atribuído para is\_dragover afetará diretamente o comportamento da classe com bind. Como podemos ver na imagem, só irá exibir a classe bindada em dois momentos: quando passar o mouse com algum arquivo sendo arrastado ou quando soltar algum arquivo. Essa variável is\_dragover está declarada mais para baixo:

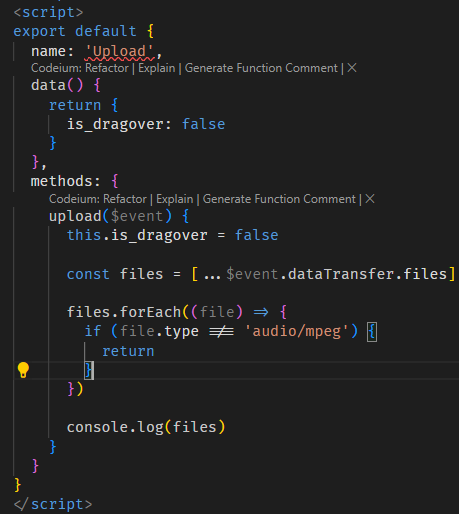


**LIDANDO COM O ARQUIVO**

Mime Types: Uma escrita para ajudar a identificar o tipo de dados em um arquivo. Quebrado em duas partes: o tipo e o subtipo separados por barra (/). Exemplos: audio/mp3, audio/ogg, application/msword, video/avi. Pode-se consultar os formatos de áudio suportados pelo html5.

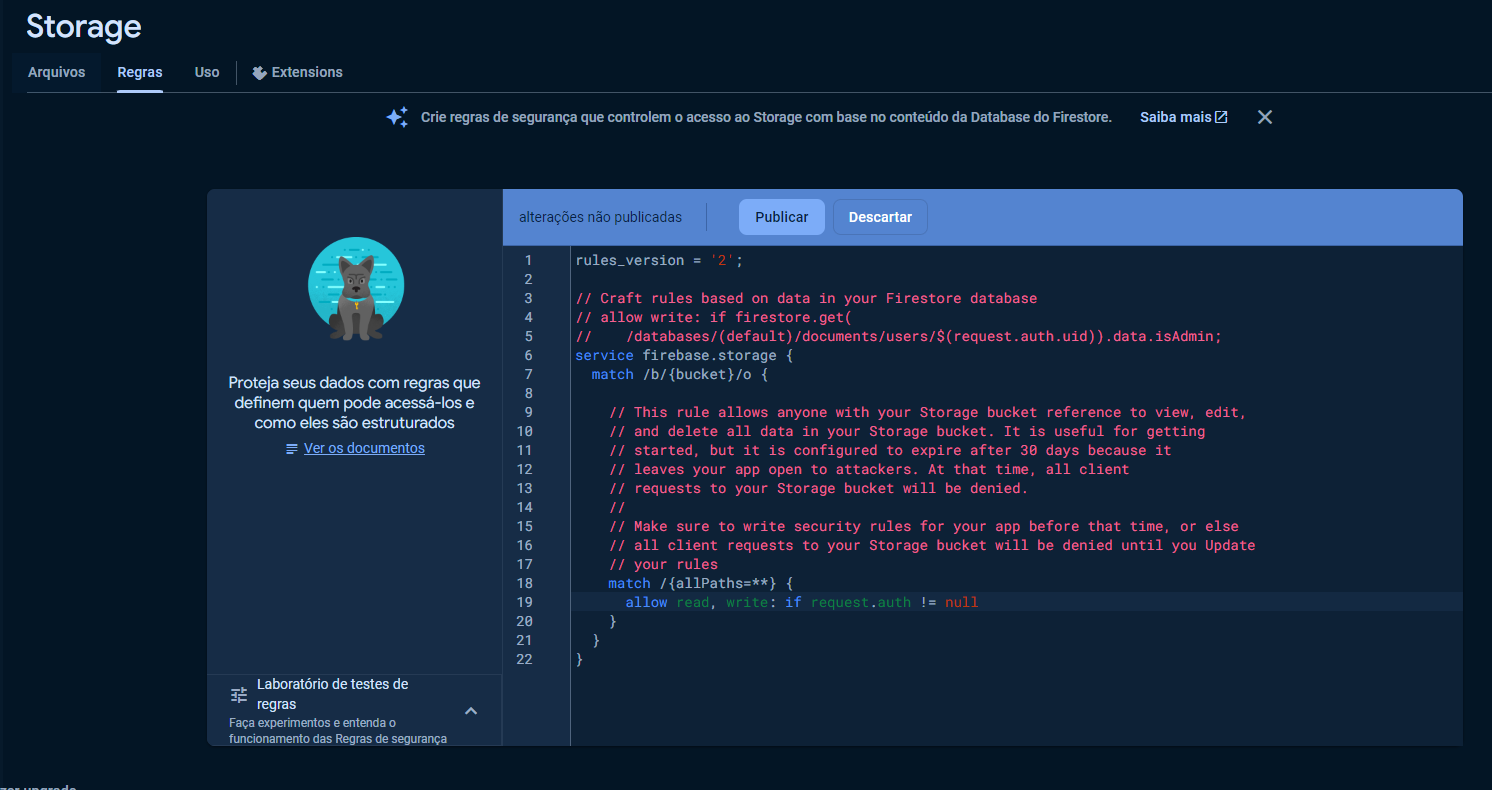


O evento de stop permite passar uma variável chamada $event.



Através do $event.dataTransfer.files podemos obter as informações do arquivo que foi solto no upload. Após isso, como é um app de música, vamos impedir que qualquer arquivo diferente de áudio/mpeg seja aceito e continue.

**HABILITANDO SERVIÇO DE ARMAZENAMENTO DO FIREBASE**

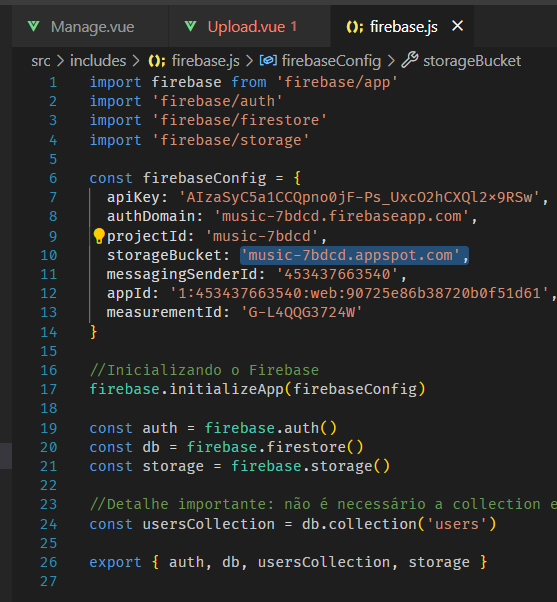


**CARREGANDO ARQUIVOS COM FIREBASE**

Para preparar os arquivos para o upload, precisamos fazer isso:

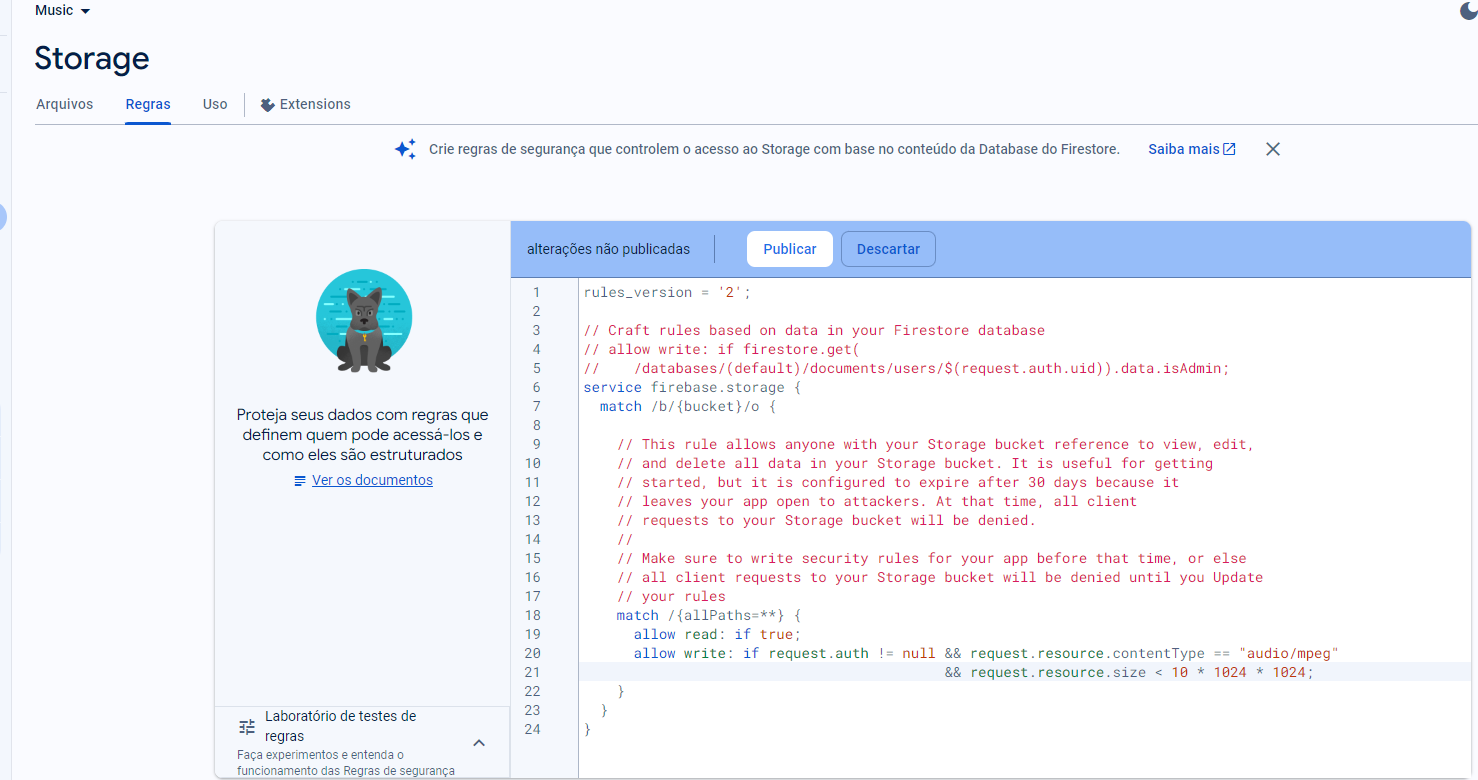


Após o if que irá retornar, sair da função se não for um arquivo mpeg soltado dentro do upload, criamos uma variável chamada storageRef que recebe storage.ref(). Esse storage vem do import que estamos realizando nesse mesmo script, no caso do storage.ref, o valor vem do storageBucket:



O Songsref está criando uma pasta com um arquivo (o do upload) dentro daquele bucket. O comando .put é o que de fato fará o upload!

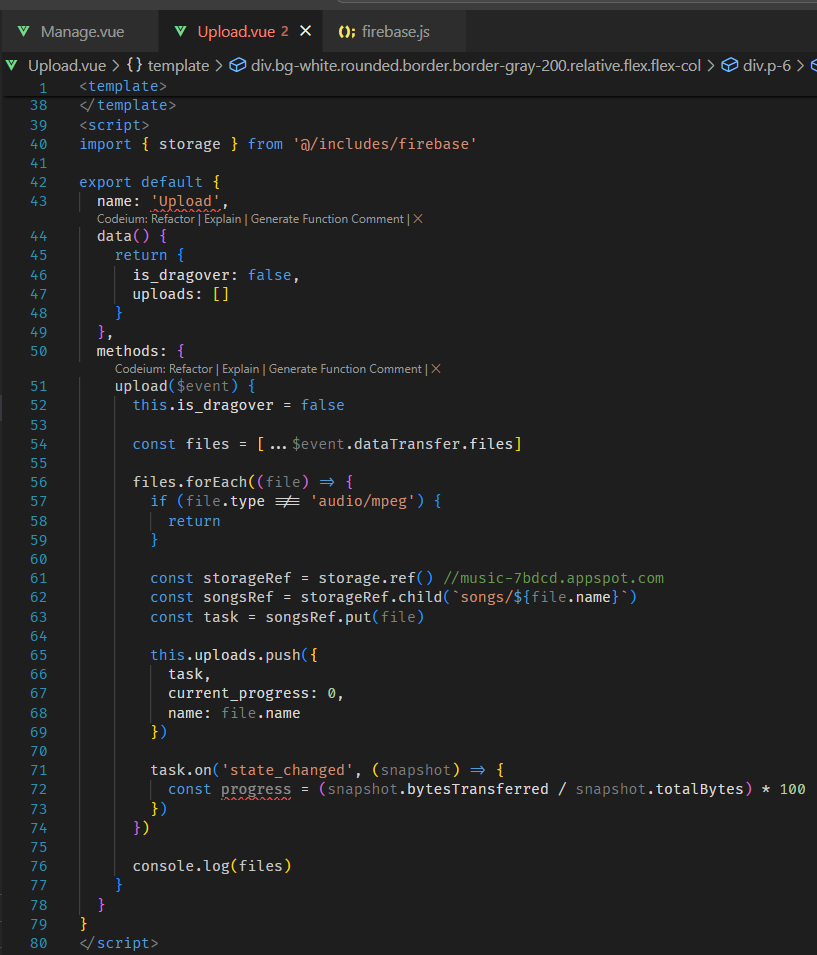
**FIREBASE - REGRAS E VALIDAÇÃO**



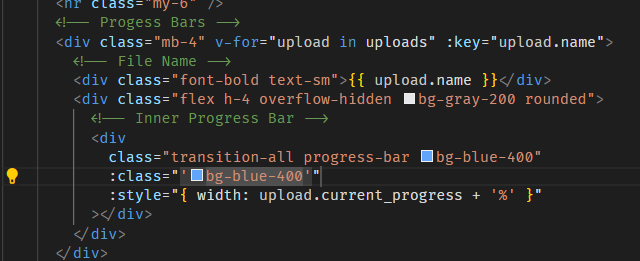
Adicionadas 3 regras no Firebase:

1. O usuário precisa estar autenticado
2. O conteúdo do upload precisa ser do tipo audio/mpeg
3. O tamanho tem que ser menor do que 10mb

**ADICIONANDO A BARRA DE PROGRESSO**

****

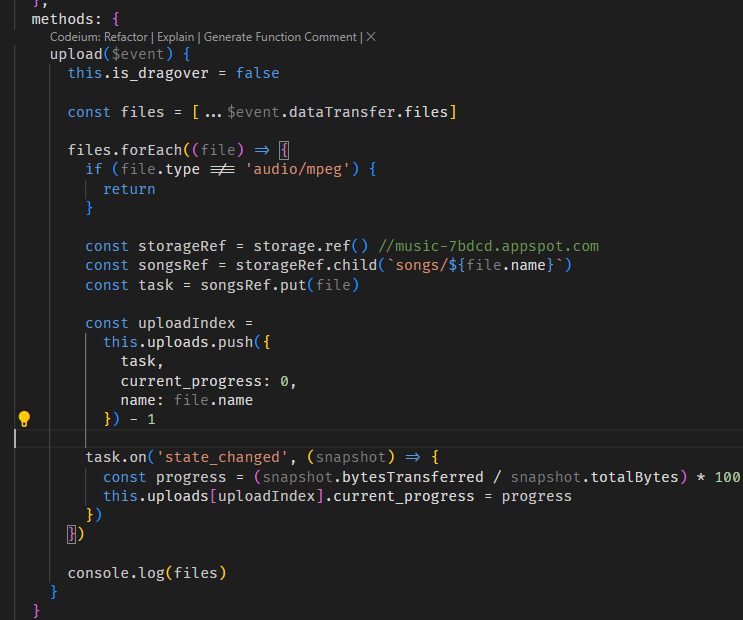
Para fazer a barra de progresso, é necessário pegar algumas informações. Primeiro, criamos um array chamado uploads no data. Esse array é populado após o arquivo ter sido enviado para o firebase. Colocamos para dentro do array a task, o current\_progress e o name. Em seguida, acionamos a task sempre que o estado mudar, pegando o parâmetro snapshot e fazendo um cálculo para a barra de progresso.



No template, tudo o que é necessário é fazer um for e distribuir os dados necessários.

**TORNANDO DINÂMICA A BARRA DE PROGRESSO**

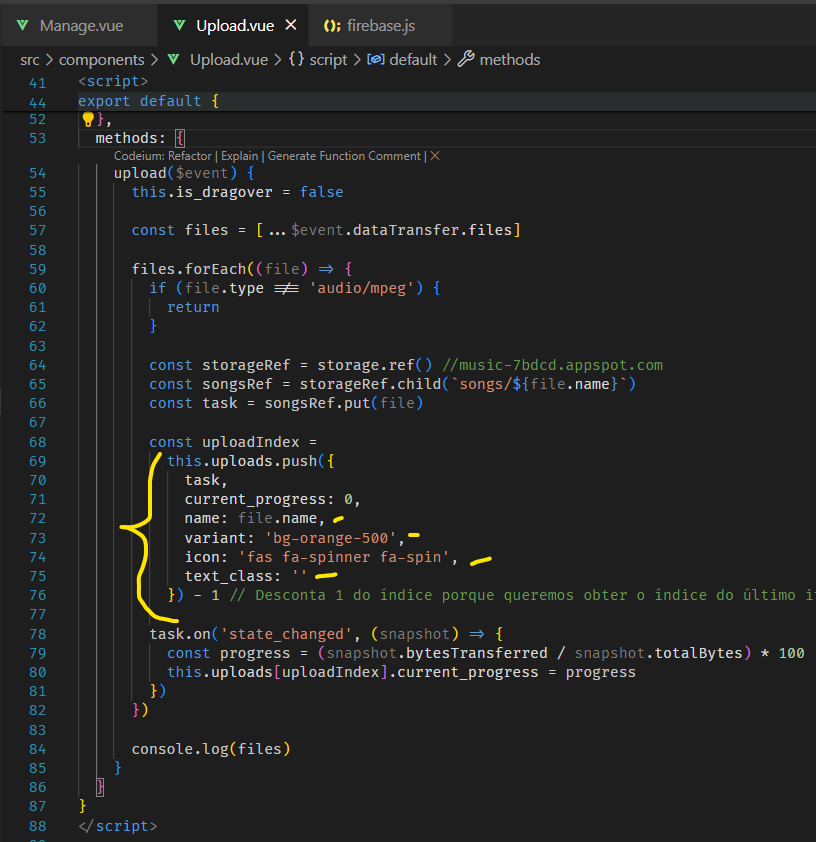
Para fazer funcionar a barra de progresso, configuramos dessa forma:

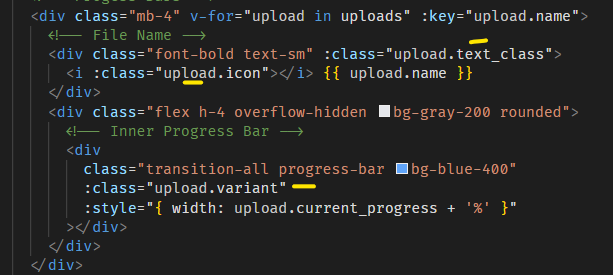


Atribuímos o resultado de progress para o current\_progress individual de cada um dos arquivos anexados, afinal perceba que tem um índice uploadIndex que é diferente para cada um dos anexos.

**MELHORANDO A BARRA DE PROGRESSO**

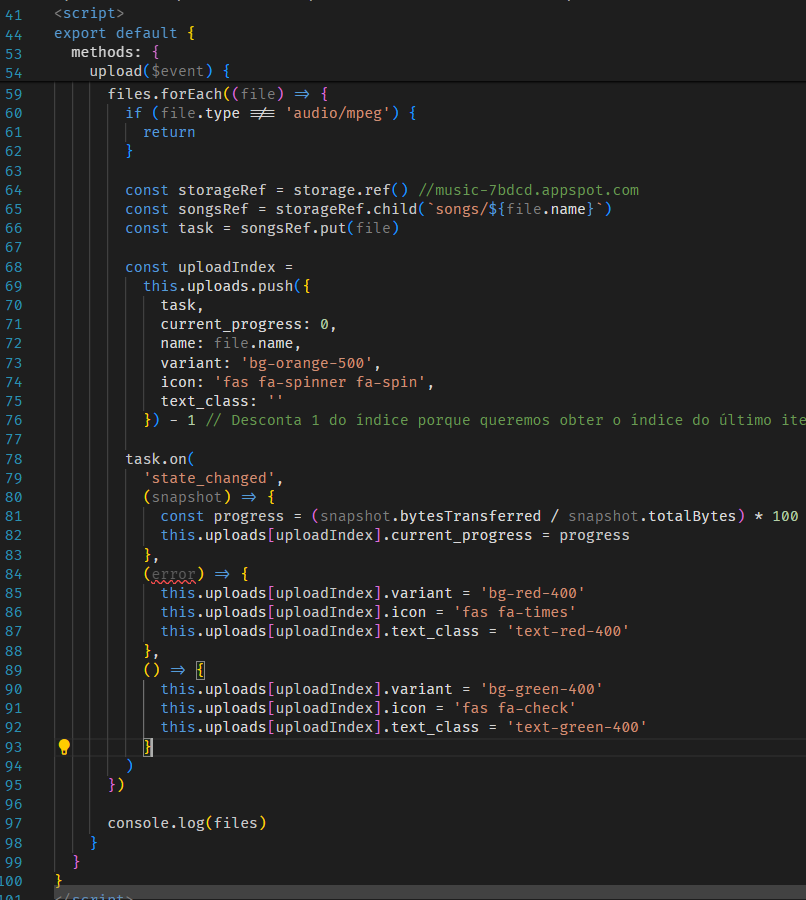
As alterações de cores, textos e ícones podem ser feitas no array uploads:





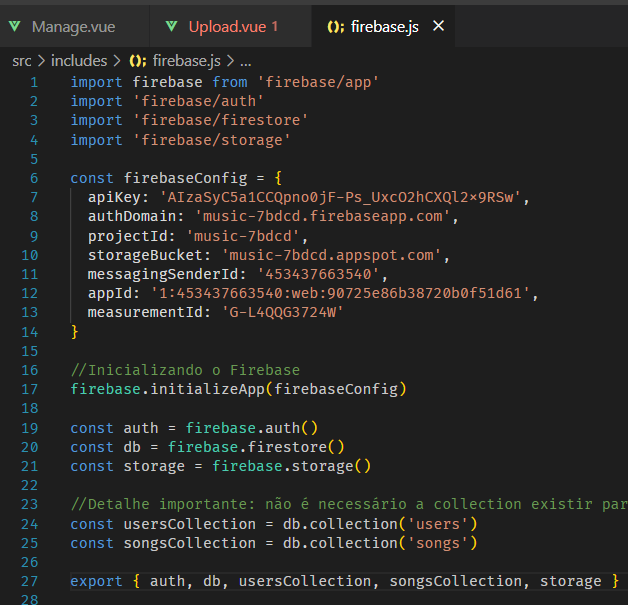
**LIDANDO COM UPLOADS DE ERRO E DE SUCESSO**

O tratamento deve ser feito no evento on da task:

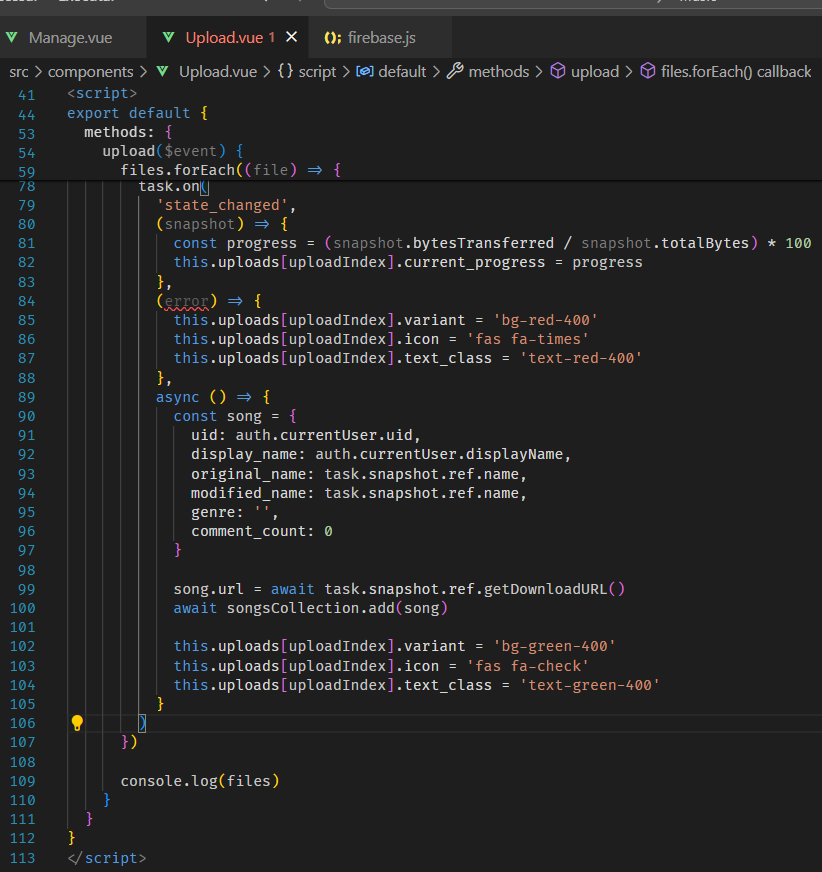


**ARMAZENANDO O FILE NO BANCO DE DADOS FIREBASE**

Para isso, é necessário criar uma nova collection vazia para os anexos. Em seguida, adicioná-la ao firebase.js:



Dentro da task, adicionamos uma função assíncrona para pegar os dados dos arquivos anexados. Trabalhamos com auth dentro dessa função porque para poder fazer o upload é necessário estar autenticado:



**DIFERENÇAS ENTRE REFERÊNCIAS E SNAPSHOTS**

* Referências
  + Objeto que aponta para um local de sua aplicação
  + Permite ler e escrever referências
  + Cria novas referências
* Snapshot
  + Objeto que é uma cópia de um local em sua aplicação
  + Somente leitura
  + Imutável

Referências e snapshots podem ler os dados na aplicação. Snapshots são somente leitura, mas são eficientes do ponto de vista de memória. Eles tornam sua aplicação mais leve e rápida.

**FALLBACK UPLOAD**

**CANCELANDO UPLOADS**

Importante: quando navega para outra página, o componente da página anterior é desmontado. Esse comportamento é atribuído aos filhos dos componentes também.

Existem duas maneiras de parar uma requisição:

1. Lifecycle: elas são chamadas quando um componente é desmontado.
2. Referências:
   1. Permite você selecionar componentes
   2. Componentes são reutilizáveis, e isso torna difícil de selecionar elementos.
   3. Tem escopo definido para suas instâncias.

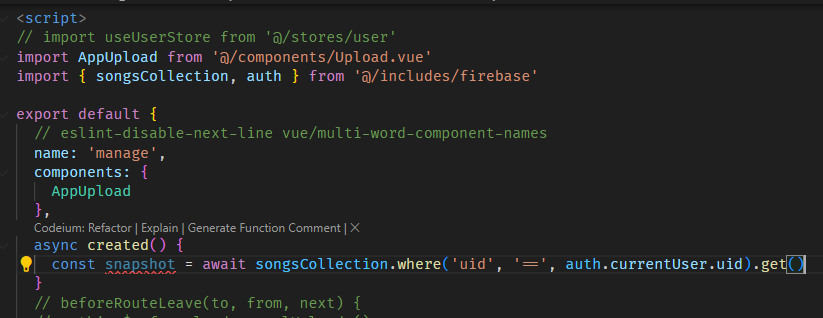
Nesse caso, lifecycle é a melhor opção porque além de ser menos código, não temos que chamar nós mesmos, o lifecycle fará isso por nós.

**MAIS UMA COISA SOBRE REFERÊNCIAS**

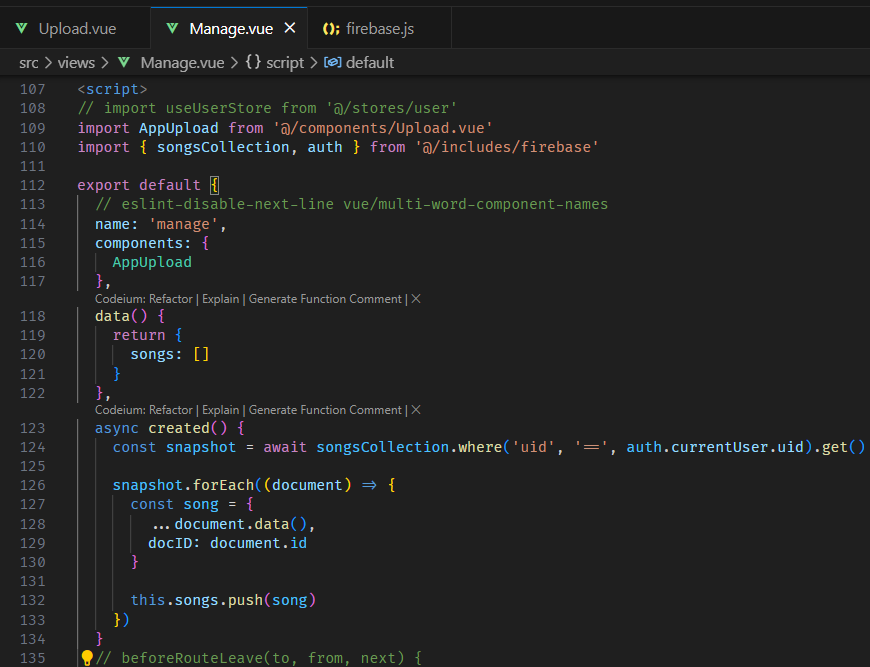
Com reatividade, o fluxo é o seguinte: dados são alterados -> instância do vue é atualizada -> vue atualiza o template -> template é renderizado para o DOM.

Por outro lado, as referências acessam diretamente o DOM sem atualizar a instância VUE.

**CONSULTANDO O BANCO DE DADOS**

****

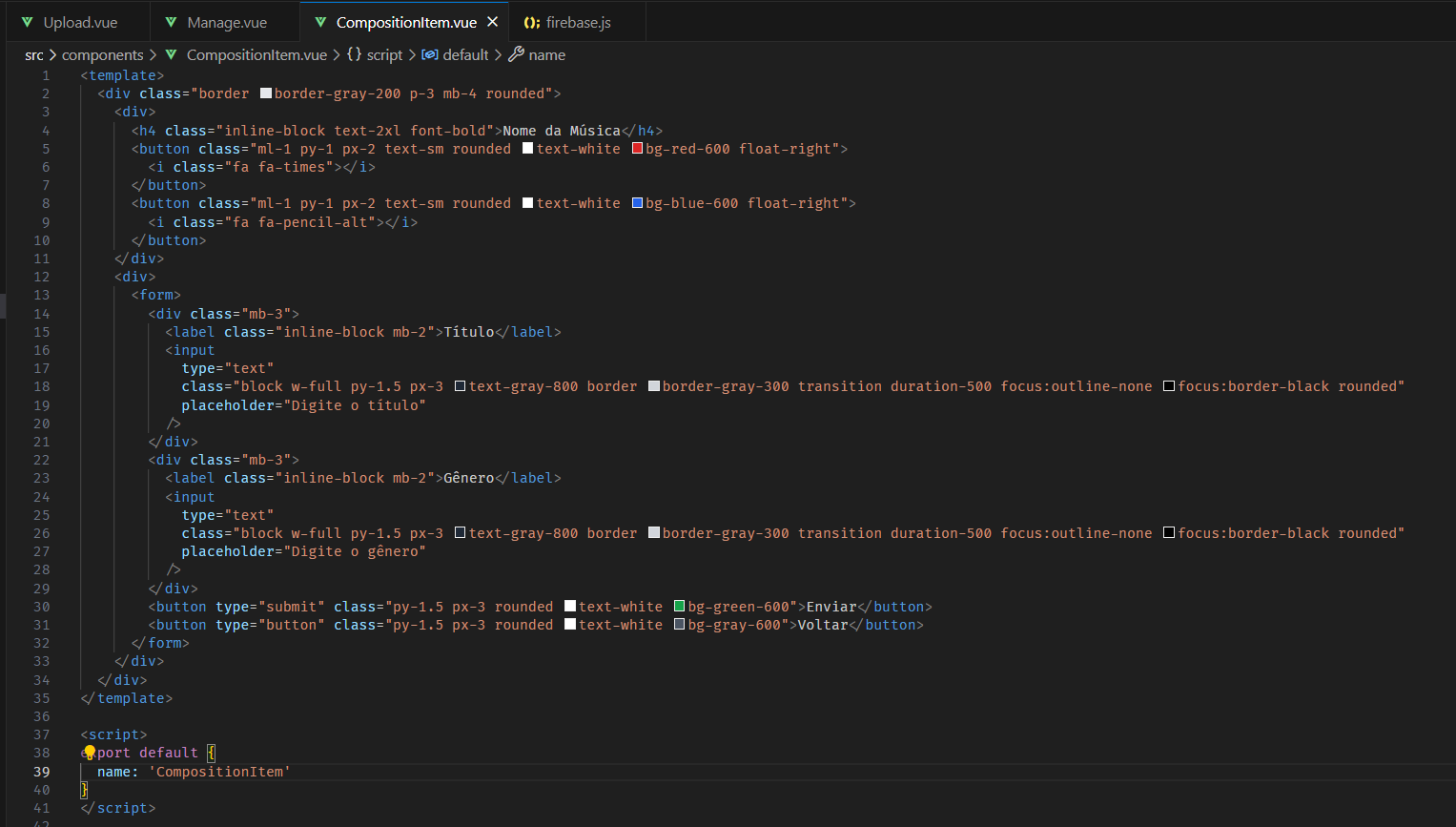
**ARMAZENANDO A LISTA DE MÚSICAS**

****

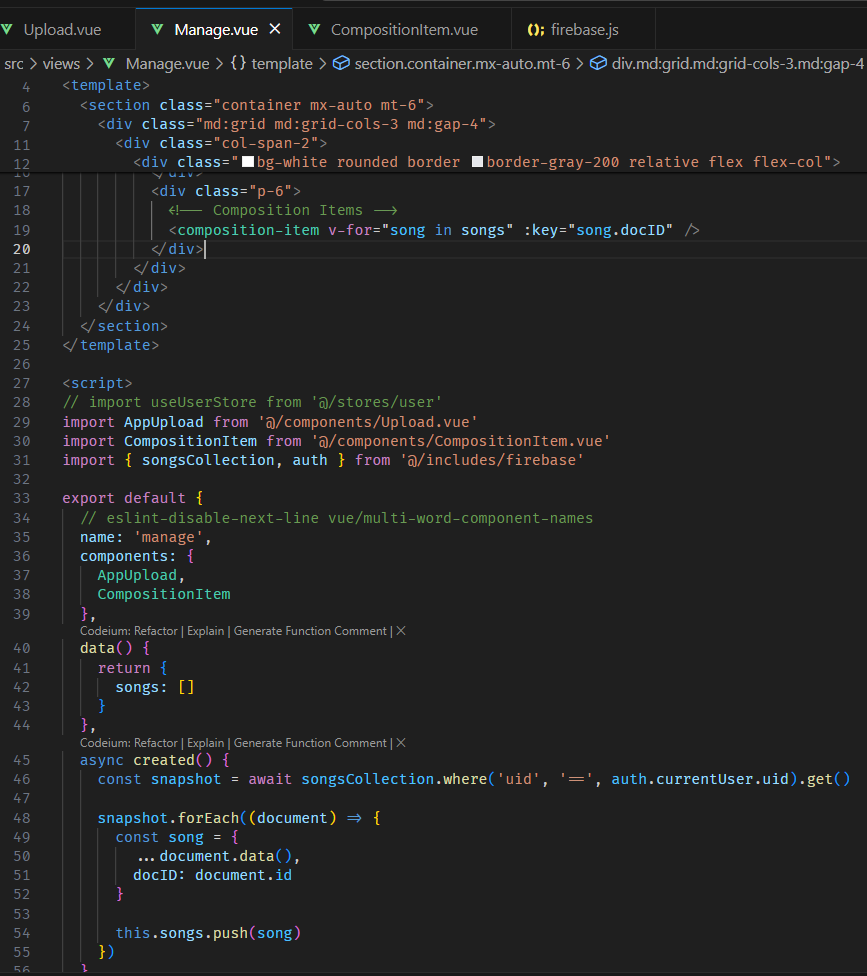
Após armazenar em snapshot os anexos do banco de dados, fazemos um foreach para popular o array songs, onde ficarão as músicas.

**EXIBINDO A LISTA DE MÚSICAS**

Montei um componente individual para cada música:



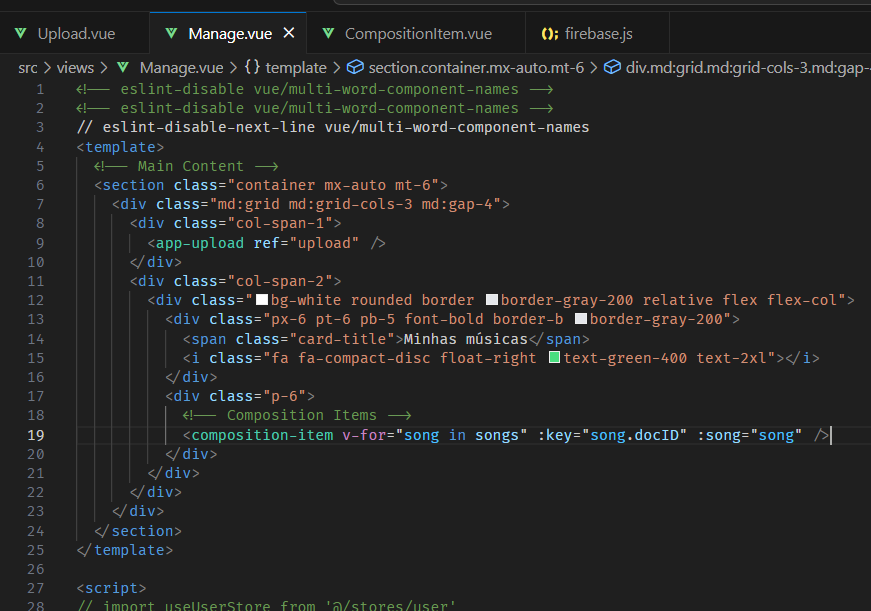
Em seguida, chamei esse componente na página manage, e fiz um foreach no componente, fazendo com que ele exibisse no número de vezes existentes dentro do array de song:



**VALIDAÇÃO DE PROP**

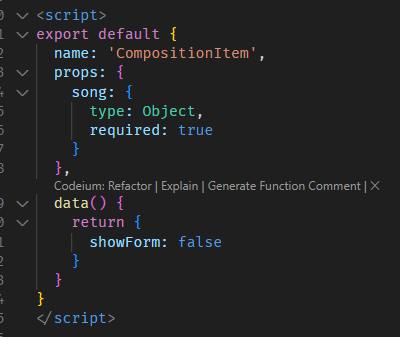
****

Definimos algumas propriedades deste componente. Isso significa, que ele precisa ter um bind com o nome dessa propriedade song quando colocarmos ele lá no manage:

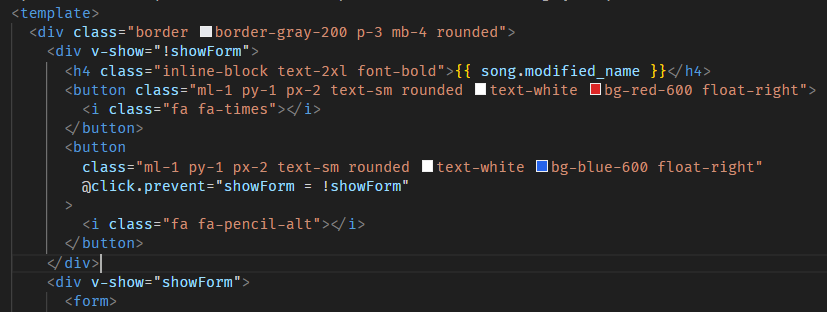


**ALTERNANDO A FORM**

Para exibir a form de edição quando clicar no lápis, criamos uma varíavel bool que irá ditar a visibilidade:



Em seguida, atribuímos a modificação deste bool para o clique no botão de lápis:

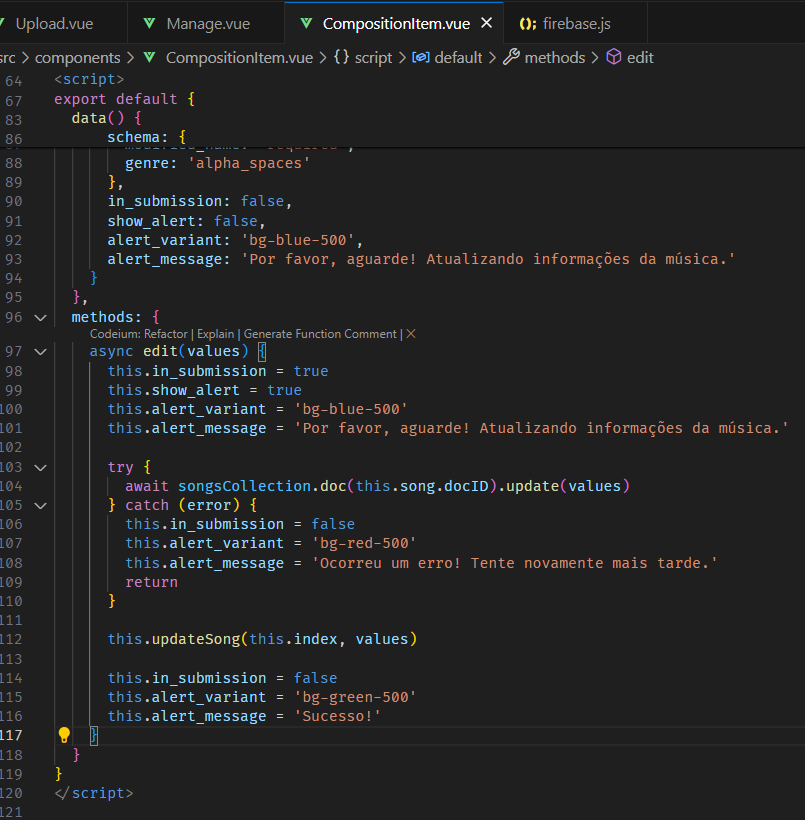


**VALIDANDO A MÚSICA NO FORMULÁRIO**

Quando trabalhamos com formulário, a primeira coisa a se fazer é prevenir que um usuário envie um formulários sem nada.

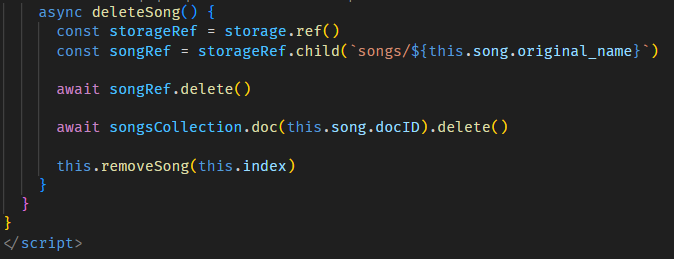
Primeiro passo então, substituímos as tags de form para vee-form, as tags dos campos de inputs para vee-field e adicionamos names a elas, e colocamos um evento de @submit no botão de enviar formulário. Além disso, criamos o :validation-schema com o schema das regras para cada input.

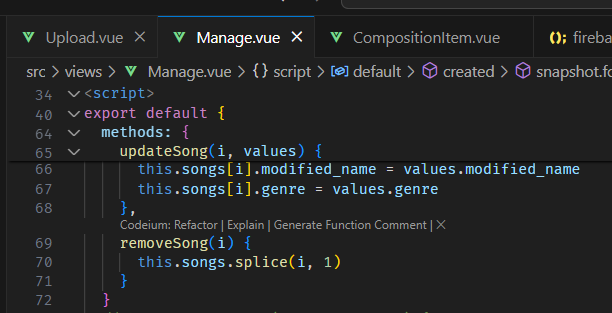
**EDITANDO A MÚSICA PELA FORM**

****

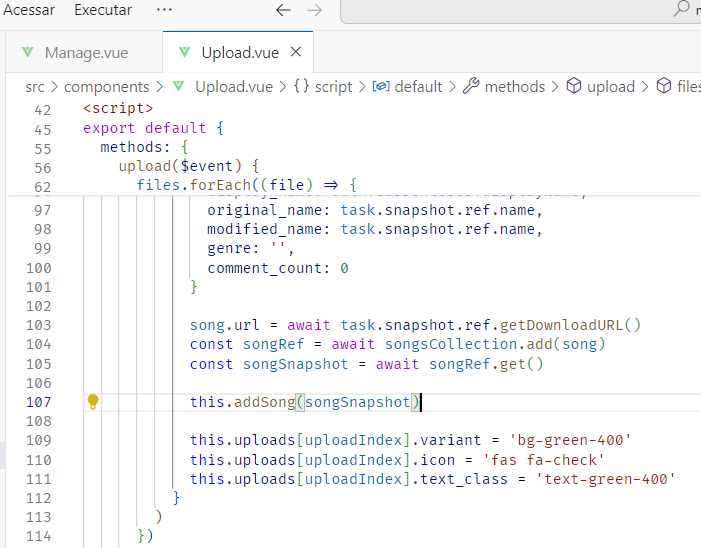
Montamos a função de edição no script e criamos dentro dela um bloco try/catch. Perceba que na edit estamos passando os values da form, e esses são os values que serão passados na parte try do try/catch. Essa função de edit fica no submit do vee-form.

**DELETANDO UMA MÚSICA DO BANCO DE DADOS FIREBASE**

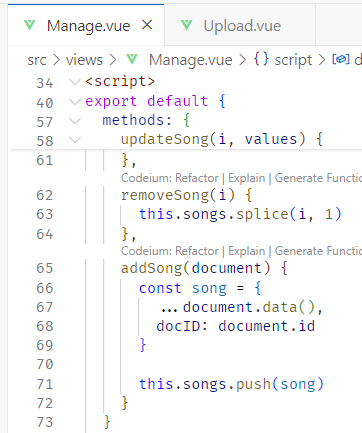
****

****

**ATUALIZANDO A LISTA DE MÚSICAS APÓS UM UPLOAD**



Os dados da música que foi adicionada estão sendo enviados para a função addSong do Manage:



Esta, por sua vez, está adicionando ao array de músicas que é listado.

**ROUTER LEAVE GUARDS**