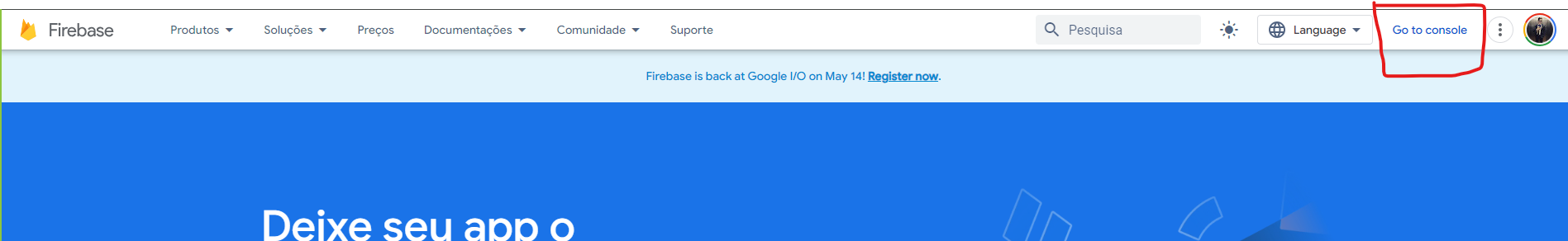
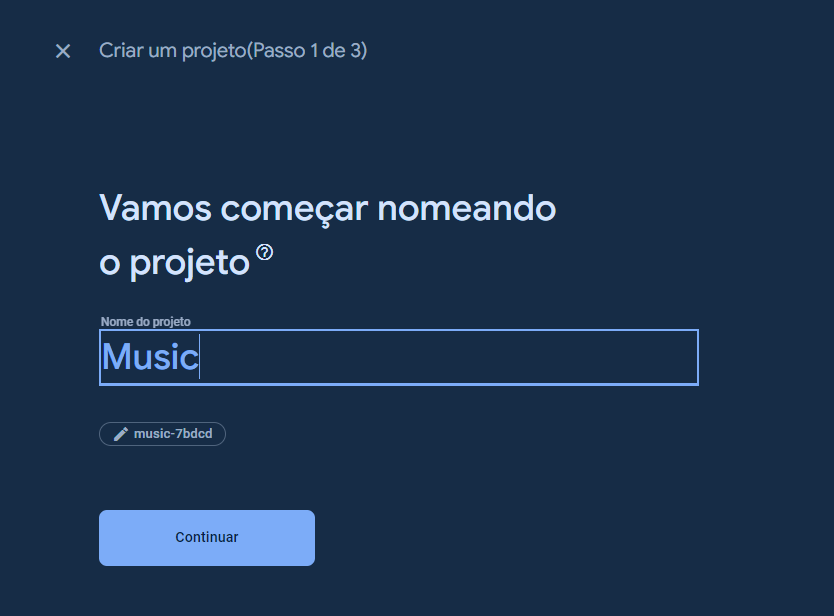
Primeira coisa importante a se destacar é que o Vue é completamente client side.

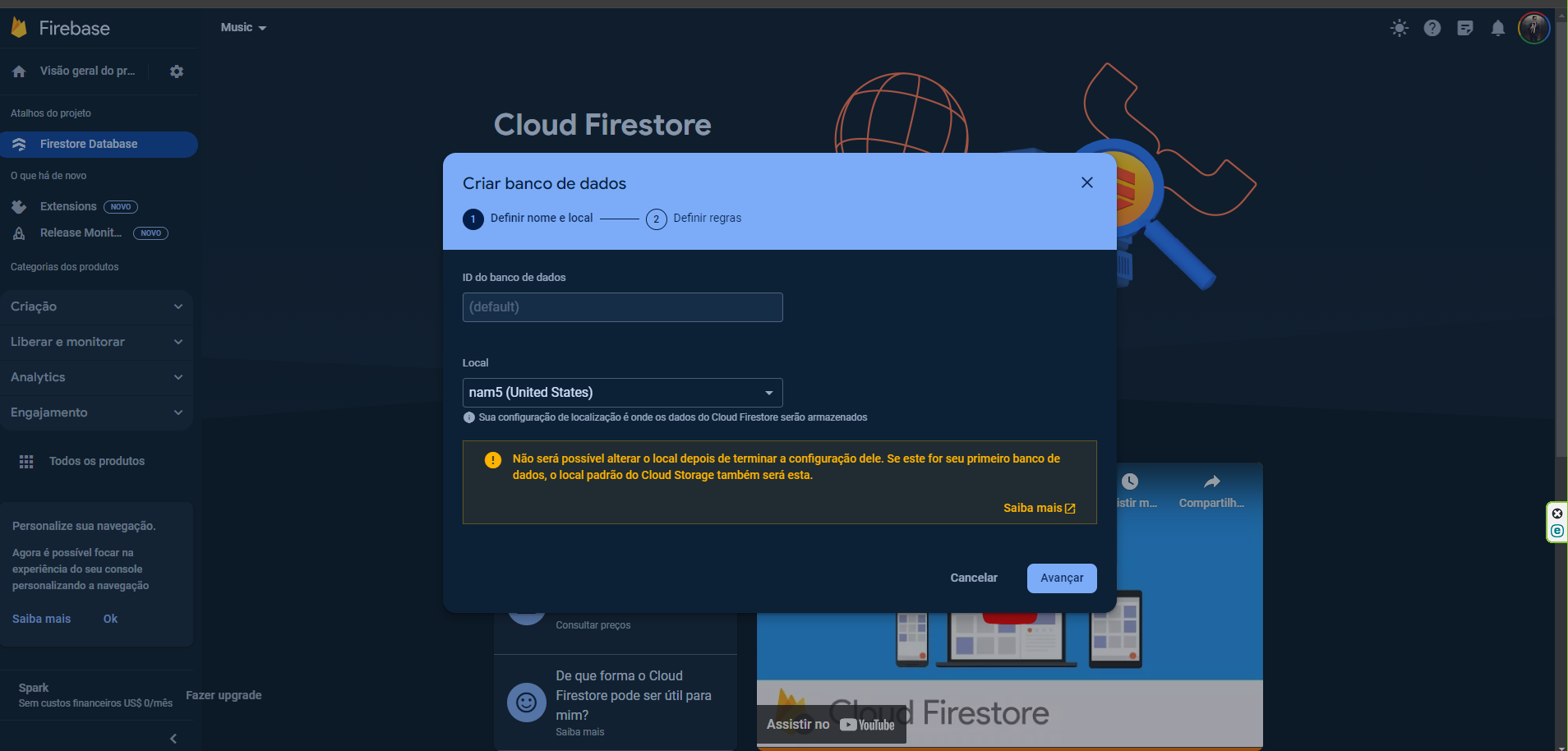
Firebase é uma solução gratuita para criar um servidor e realizar tarefas como por exemplo a autenticação.

**CRIANDO SEU PROJETO FIREBASE**

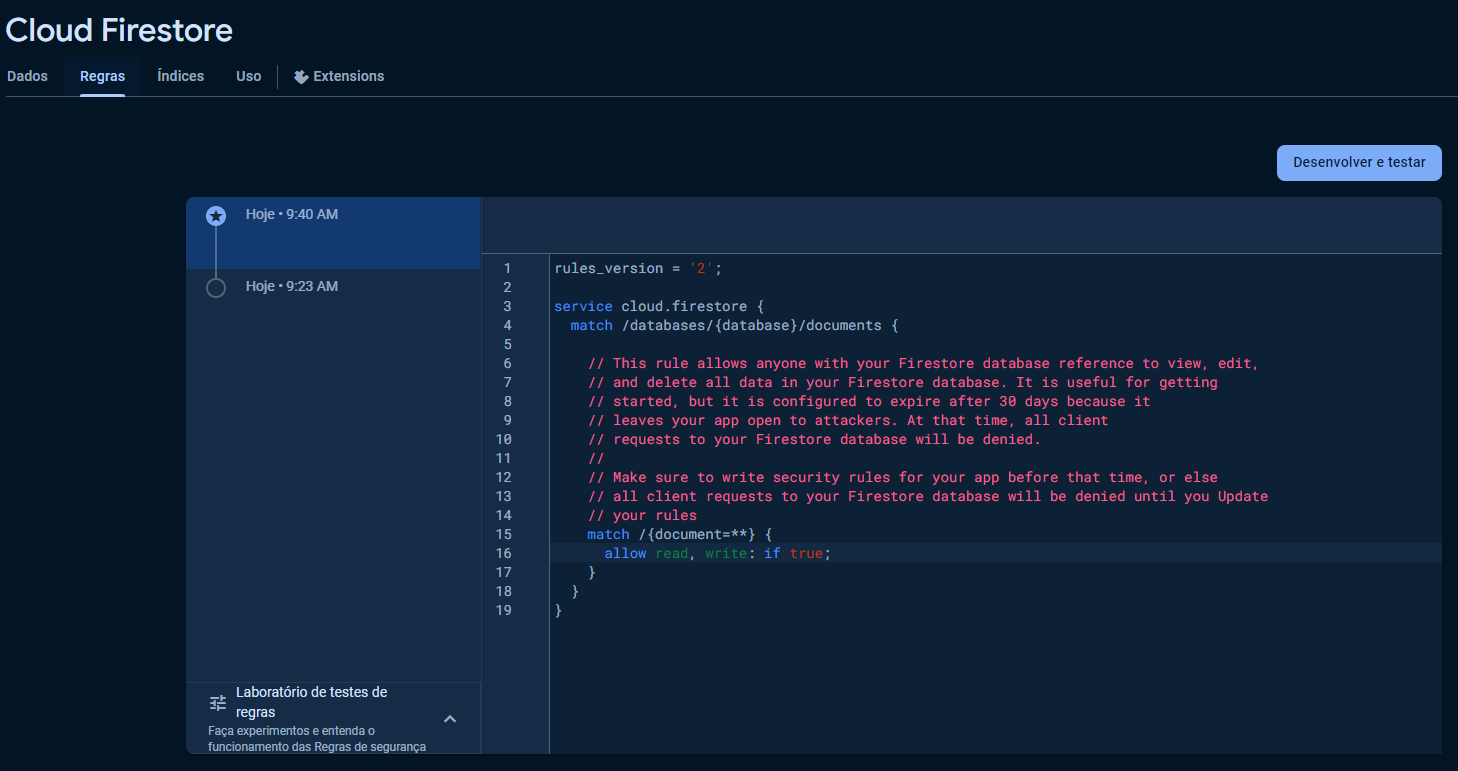




O próximo passo é criar um banco de dados firestore.







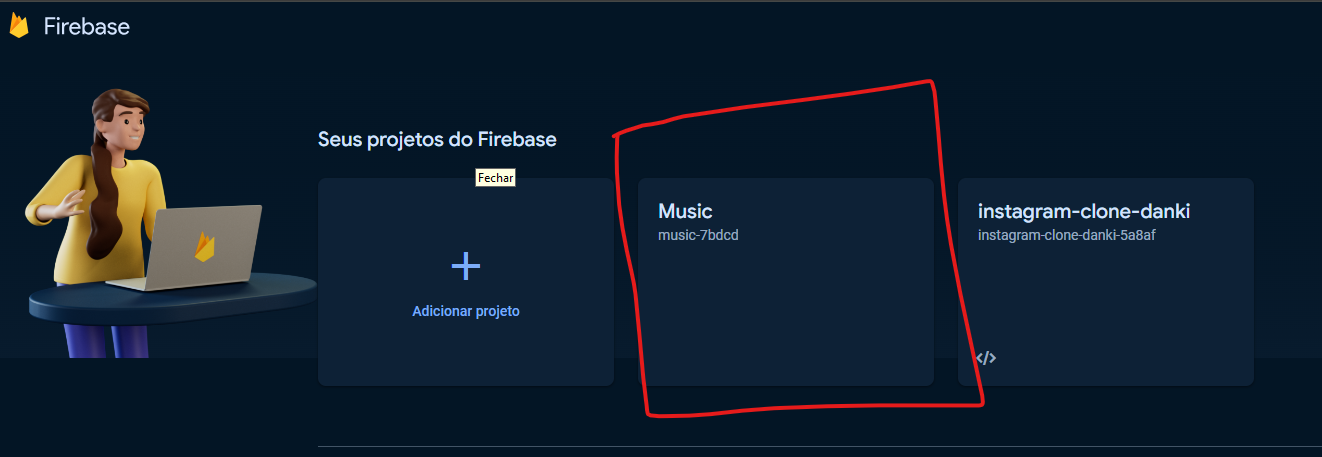
Alterar para allow read, write: if true;

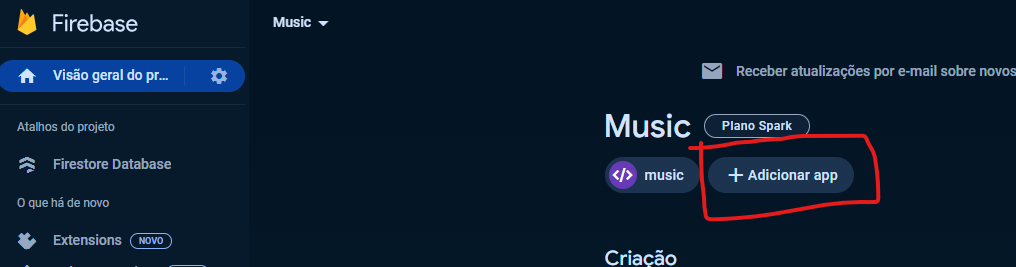
**CONFIGURANDO O FIREBASE SDK NO PROJETO VUE**

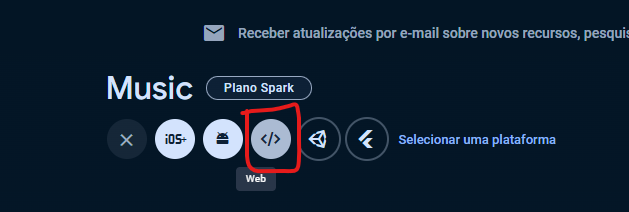
npm i firebase@8

Na pasta includes do projeto, junto com o validation.js, criar um arquivo chamado firebase.js.

O próximo passo:



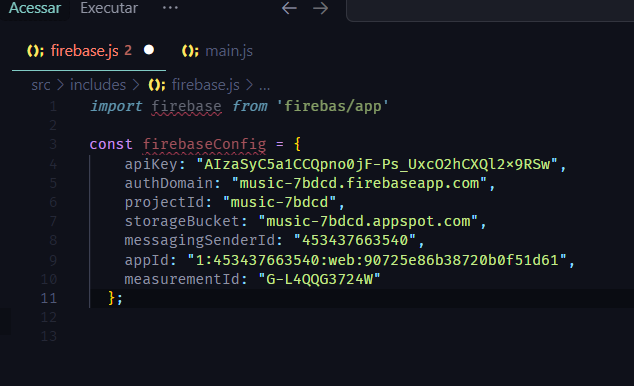




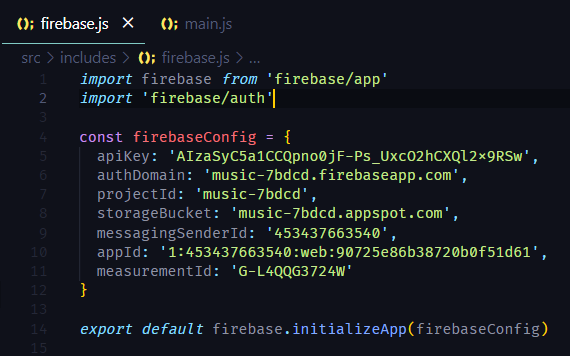
****

Copiar o trecho de código abaixo e ir para o editor de código:

****

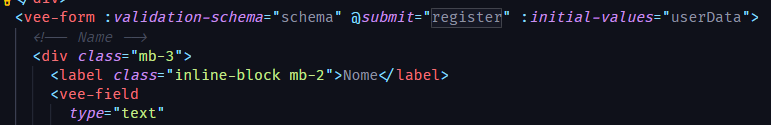


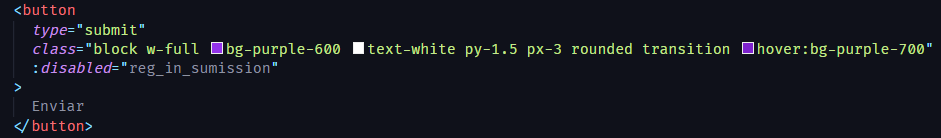
Como a intenção é utilizá-lo para autenticação, vamos o importar o auth do firebase:



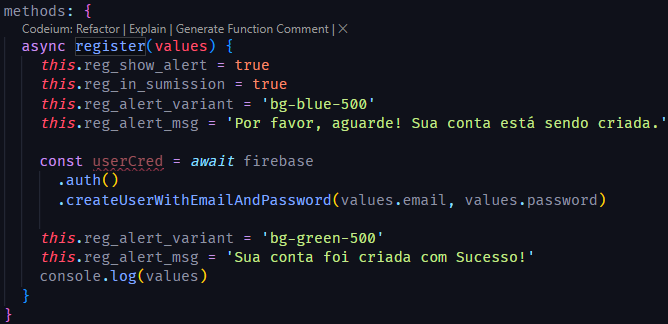
**REGISTRO DE USUÁRIO**

Na tag de abertura do formulário, tem um @submit passando uma função. Em outras palavras, quando o botão do tipo submit for clicado, o @submit é acionado e a função é chamada:





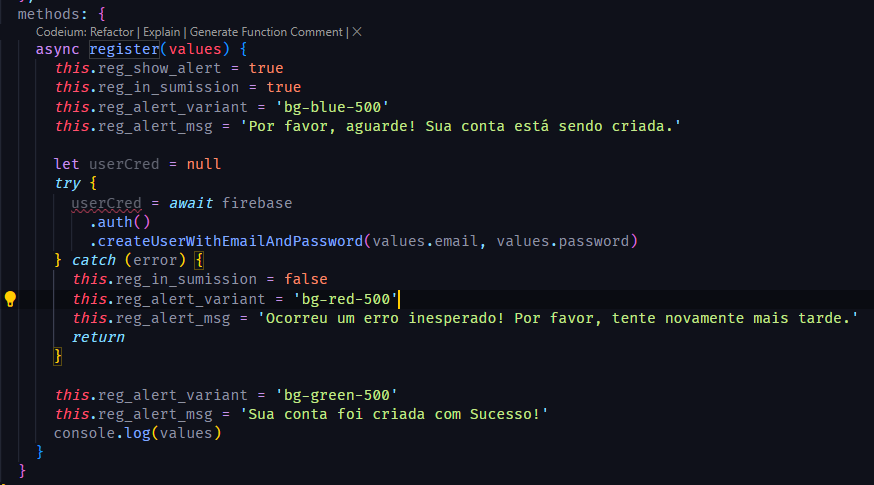
A função register chama a função e envia os dados do nome e senha para a criação do usuário no firebase:



As funções de .auth() e .createUserWithEmailAndPassword com seus parâmetros são do firebase.

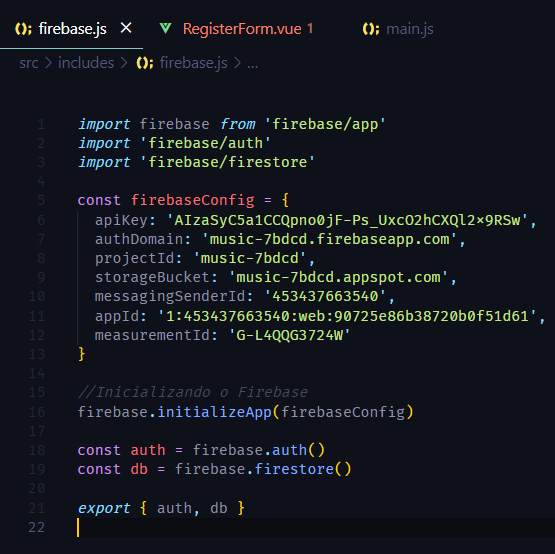
**LIDANDO COM A REQUISIÇÃO**

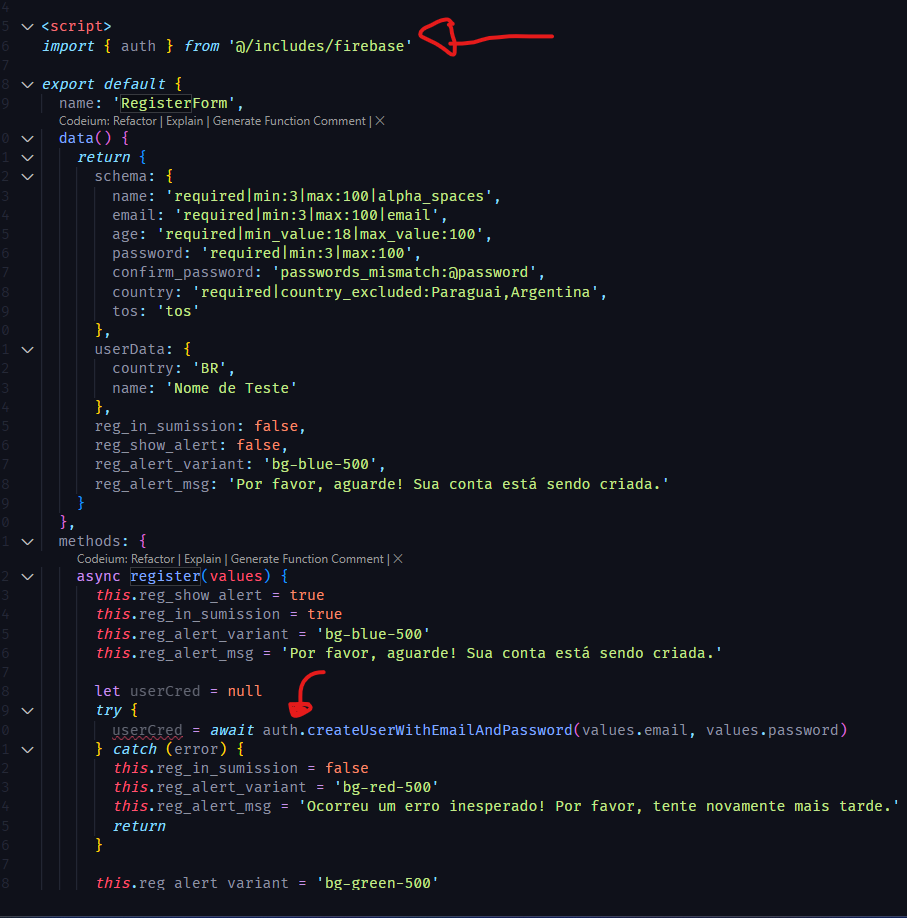
Para isso, colocamos a requisição dentro de um bloco try/catch:



**EXPORTANDO SERVIÇOS**

Foi necessário fazer umas alterações para tornar o código mais adaptável:

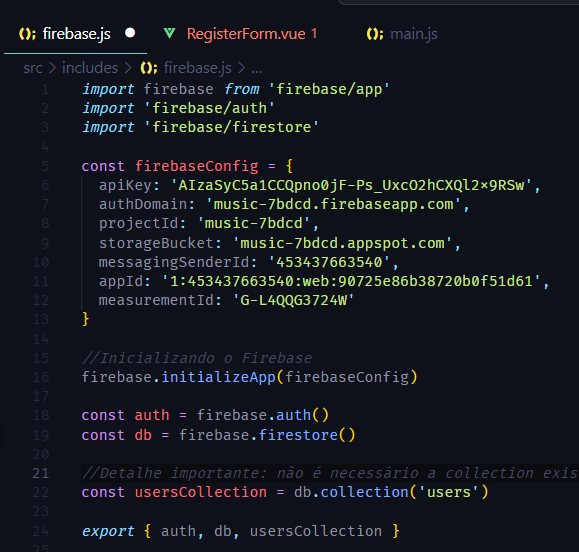




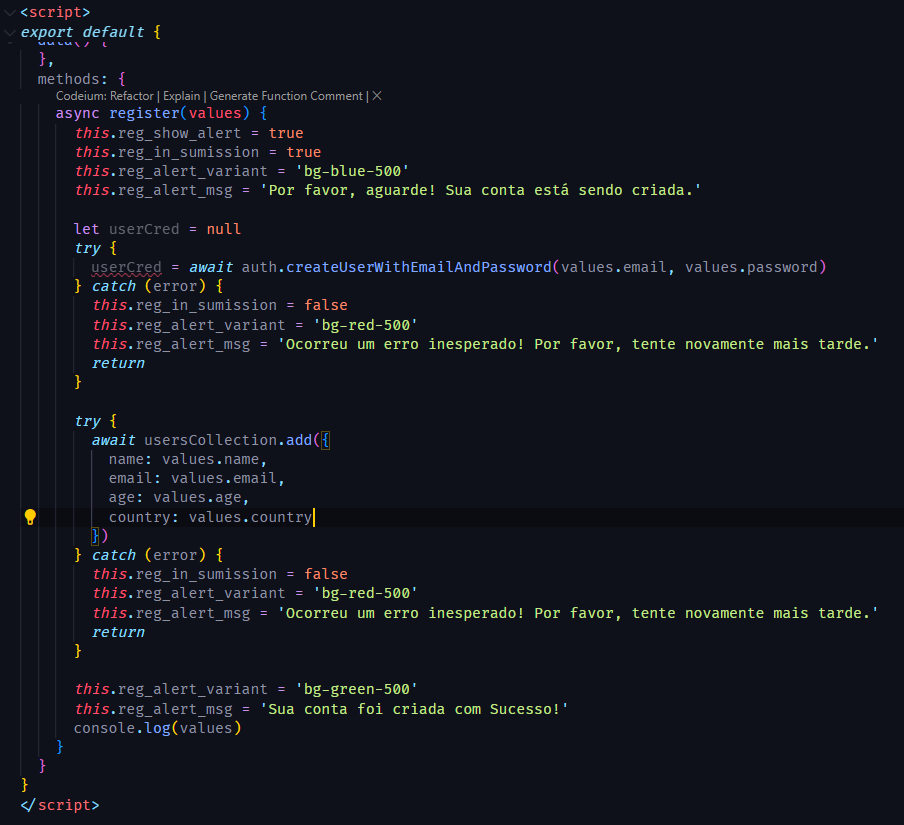
**ARMAZENANDO OS DADOS DO FORMULÁRIO DE USUÁRIO**

* **Definição de alguns termos importantes:**
  + **Buckets -** São locais físicos onde seus dados são armazenados. Você pode criar múltiplos baldes se tiver um plano premium. Pode ser necessário fazer isso se a intenção é manter alguns dados separados.
  + **Collections -** São os dados armazenados em seu banco de dados.
  + **Documents -** São os objetos no banco de dados. Essa é uma forma de categorizar os dados, por exemplo: ‘User Collection, Song Collection’.

Sabendo disso, vamos fazer o seguinte para poder armazenar os dados do formulário de usuário: primeiro adicionamos uma nova collection:



Após, o código é o seguinte para adicionar os dados na nova collection:



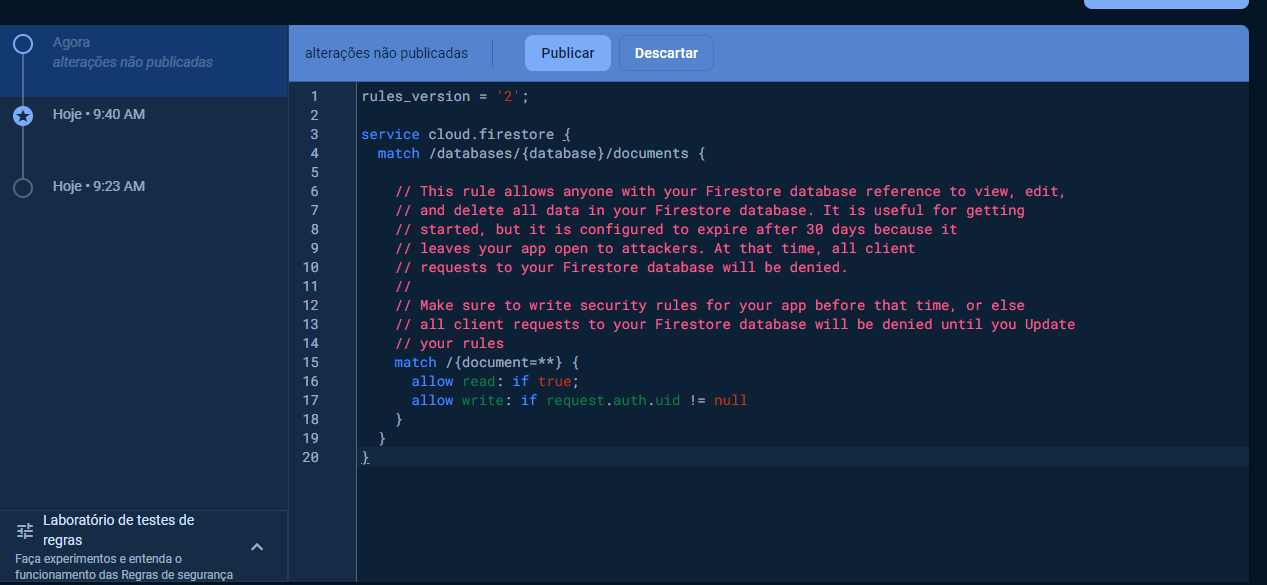
Perceba que adicionei um novo bloco try/catch com o add usersCollection, passando os campos que eu quero criar com seus respectivos valores.

**ENTENDENDO A AUTENTICAÇÃO (VUE e FIREBASE)**

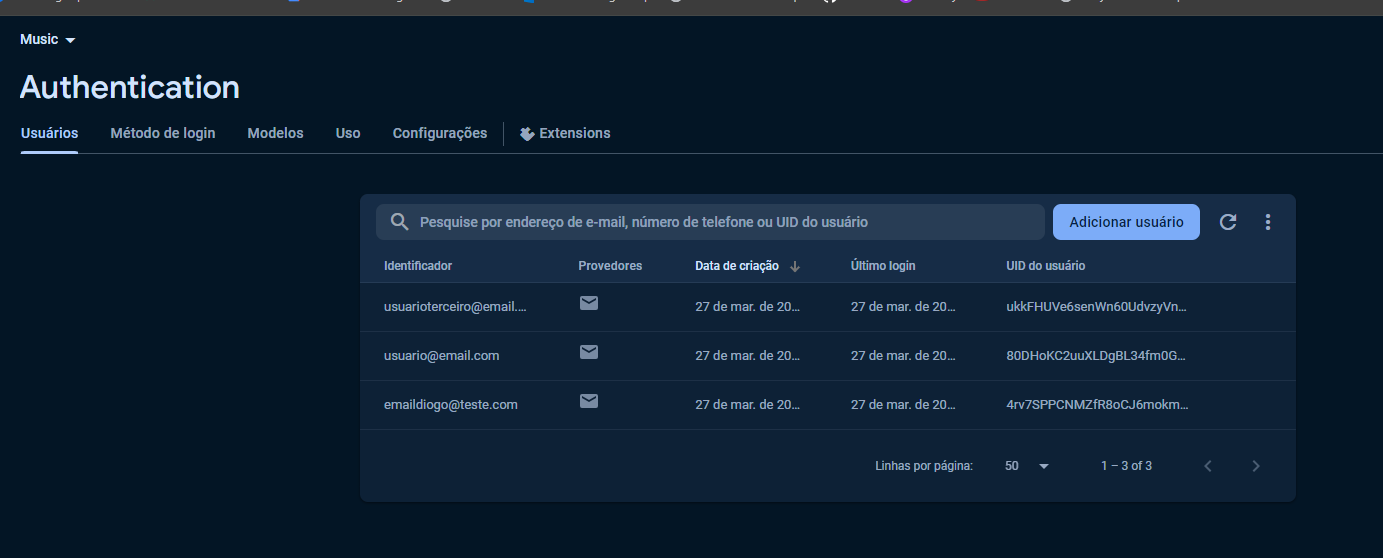
A aplicação em Vue irá enviar dados de login do usuário para o Firebase. Se a autenticação ocorrer com sucesso, o Firebase irá gerar um token e o enviará de volta como resposta, tipicamente no formato de uma string única. É essencial armazenarmos esse token.

Nesse caso, utilizaremos o local storage, que é uma web API que já vem com o navegador por padrão. Sempre que for necessário armazenar ou atualizar dados relacionados ao usuário, enviaremos esse token junto, pois ele será usado para verificar o usuário. O servidor não monitora ativamente se o usuário está logado, em vez disso, o token é usado para verificar o usuário. Essa autenticação é chamada de de Stateless.

Melhoramos as regras de autenticação no firebase:



Ajustamos as regras. O request.auth.uid pode ser encontrado em:



**ENTENDENDO ACTIONS**

Actions são funções para terceirizar a lógica de negócios.

São disponíveis para todos os componentes.

Podem ser assíncronas.

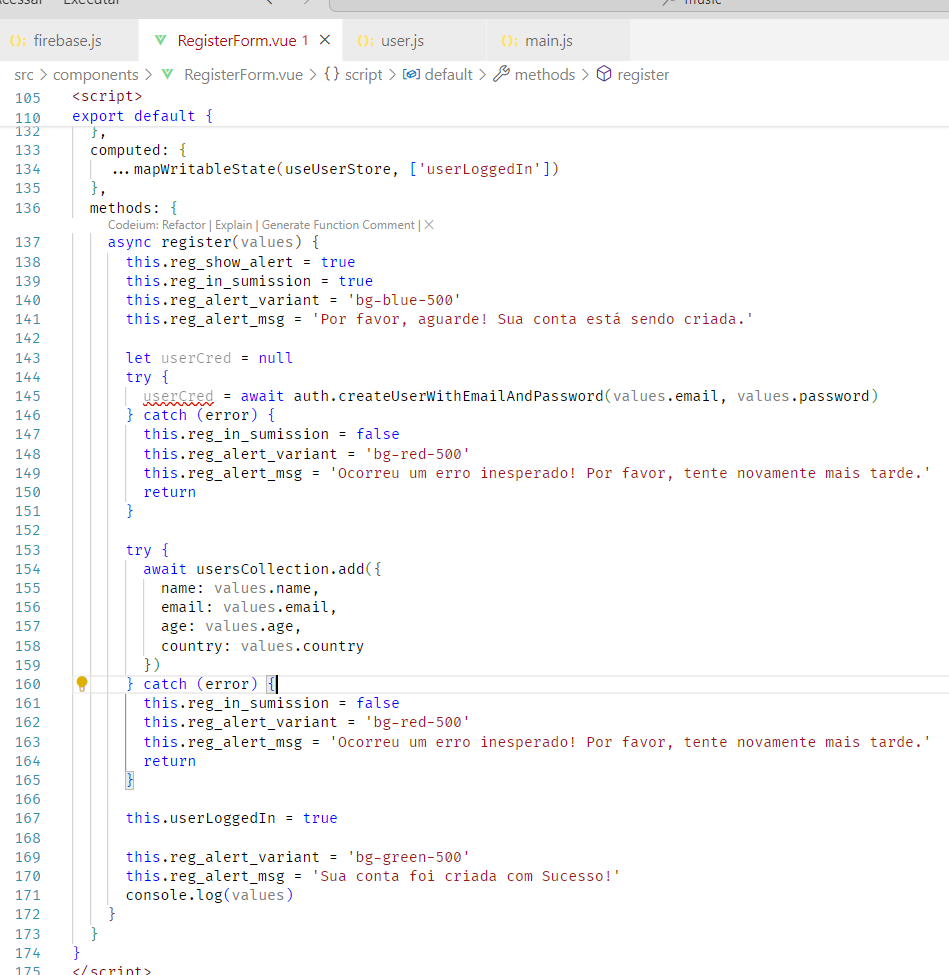
Pode cometer múltiplas mutações.

Actions vs Components

* Defina lógica em componentes se as alterações afetarem o componente ou os filhos.
* Define lógica em actions se as alterações afetarem a store.

**USING ACTIONS**

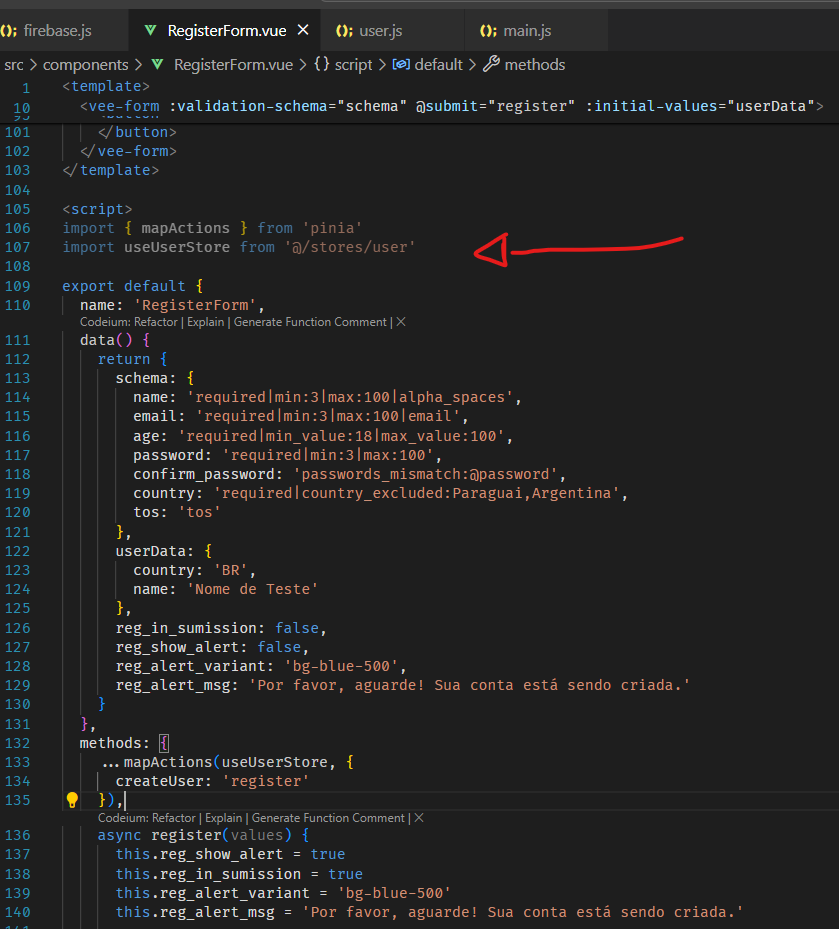
Analise o código abaixo, e veja que estamos fazendo duas requisições e duas atualizações de componentes:

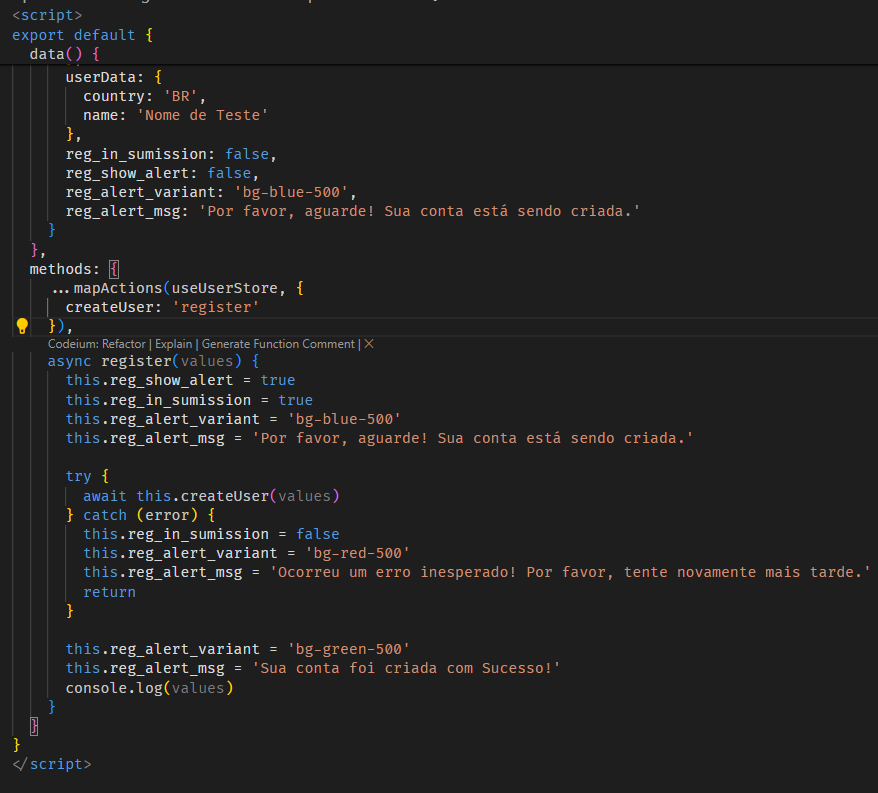


A ideia aqui é separar as atualizações de componentes das requisições, estas últimas se tornarão actions.

A atualização ficou dessa forma:

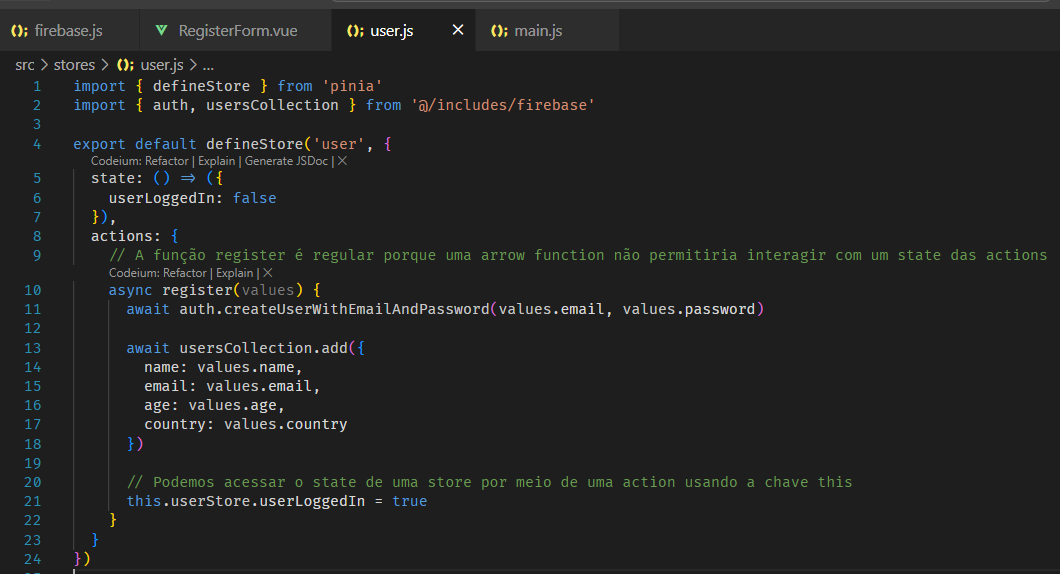
* A store continua sendo importada aqui no topo:





Está acontecendo o seguinte: ao clicar no no botão enviar do form que tem o @submit=’register’, está caindo ali no script nessa função async. Ela vai fazer um try/catch, onde no try ela chama o createUser passando os value do formulário. O createUser é o nome do parâmetro dado ao fazer um map na useUserStore, passando como valor o nome da função register. Isso quer dizer que atribuimos a chamada da função register que está no useUserStore para a chave createUser.

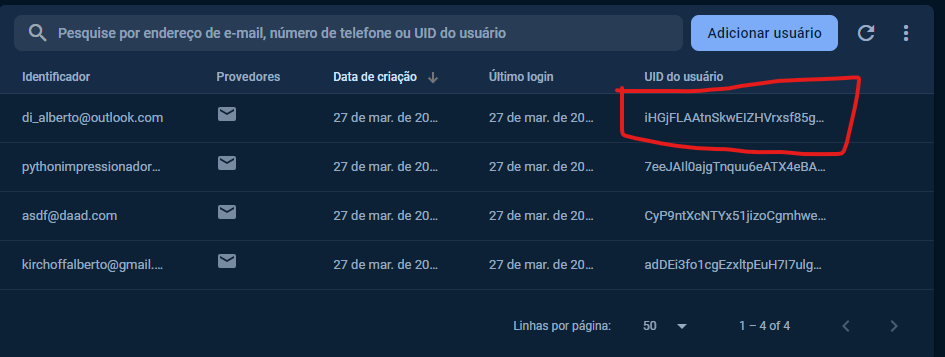
Ao ir lá na store, que nesse caso é no user.js, vamos ver a função register:

****

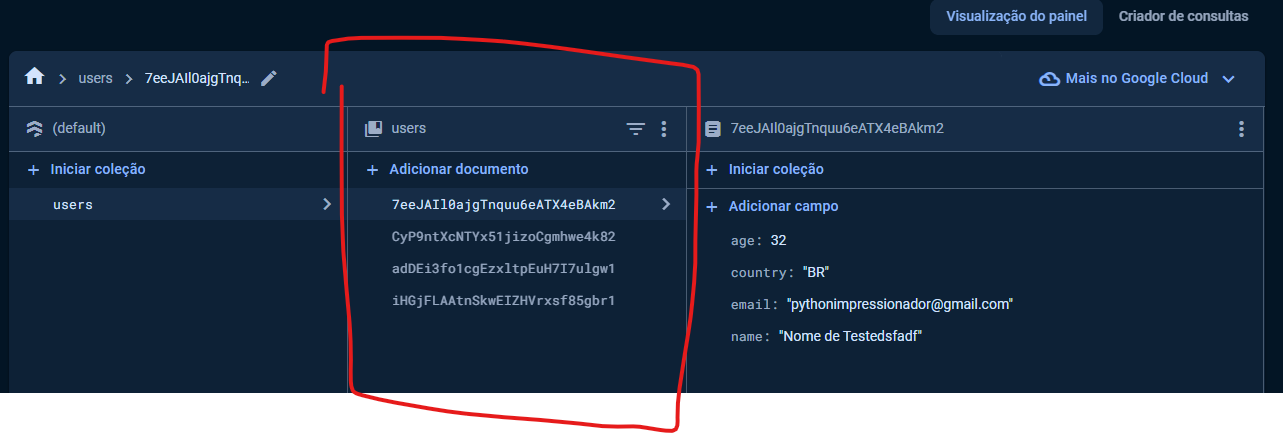
É nessa função que o usuário é criado no firebase e também a collection, além de definir como true o estado userLoggedIn.

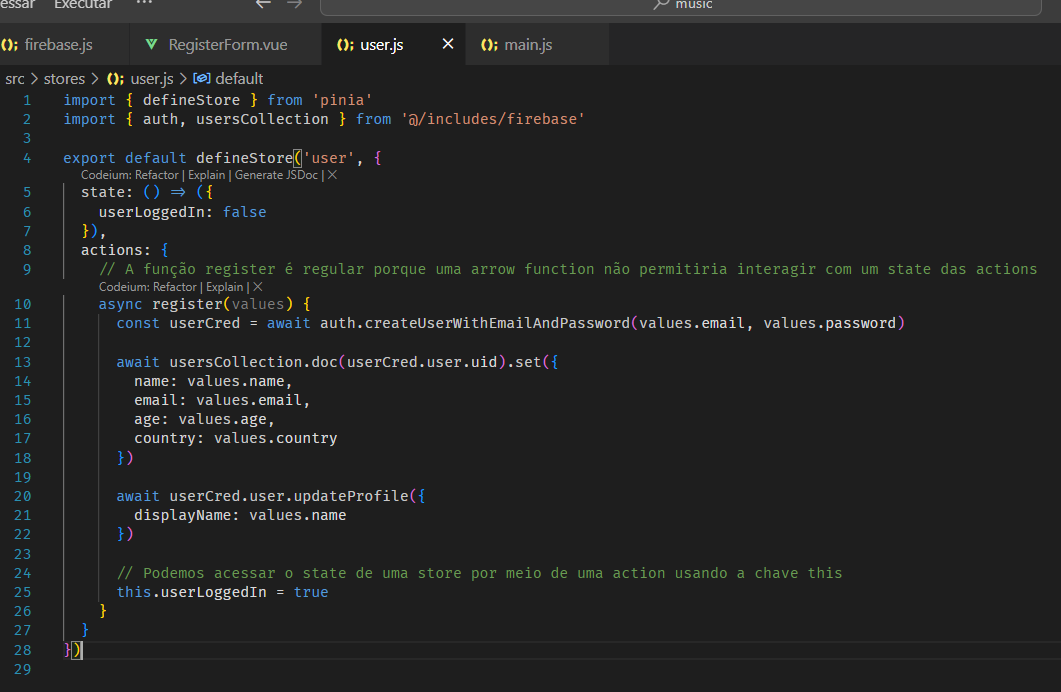
**CONECTANDO O USUÁRIO COM SEUS DADOS**

O User ID de cada usuário que está na authentication é único. Vamos querer armazená-lo no banco de dados para saber qual dado pertence a cada usuário:



Tenha em mente o seguinte: o print acima é da da authentication. No banco de dados, nós temos as collection que seriam como tabelas, onde cada registro é um document, que seria o registro, e os campos são os campos dele. Para fazer a conexão dos users da autenticação com seus respectivos no banco de dados, a gente precisa que o document tenha uma chave que nunca mude, por isso vamos usar o UID do usuário quando ocorrer o registro do document. Pensa assim, ao autenticar um novo usuário, já em seguida é criado um novo document com os campos registrados. Se o document tiver o mesmo UID do usuário, os dados estarão conectados!

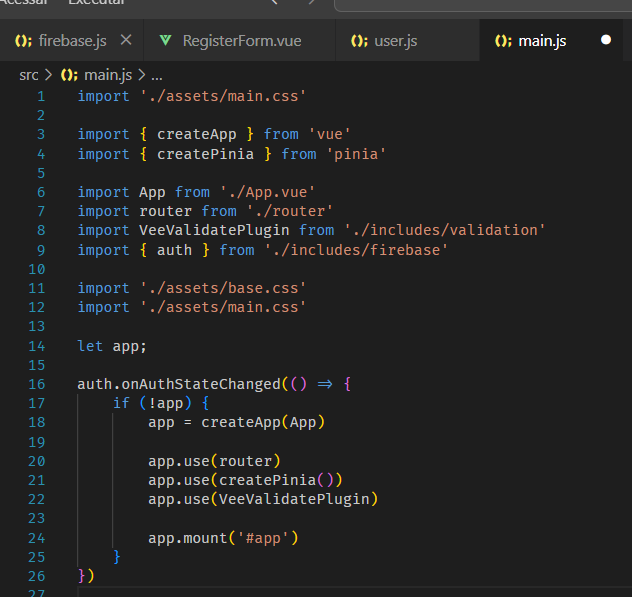




Assim, modificamos o add da usersCollection para um set, além de passar o uid do usuário como propriedade de doc. Quando usamos o add o banco de dados como default gera uma nova string como doc, mas se passarmos uma string e colocarmos o set, estamos obrigando ele a fazer o que queremos.

**INITIALIZING FIREBASE FIRST**

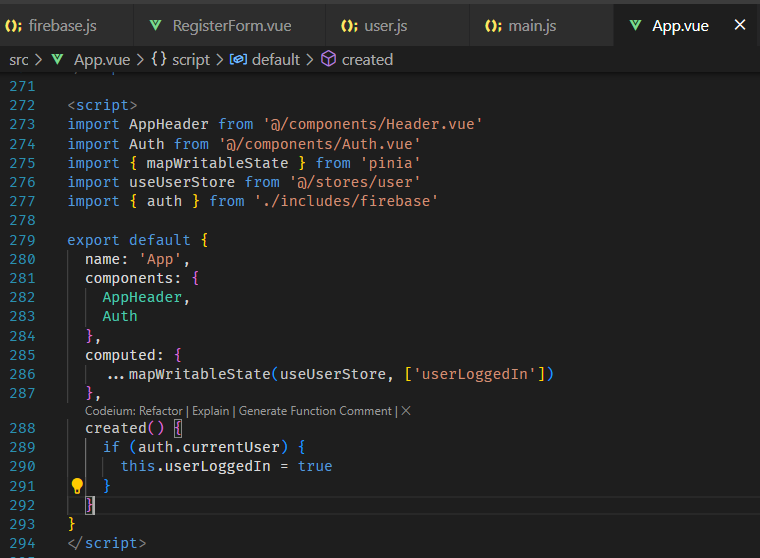
Inicializar o firebase antes que o componente principal seja montado permite que ao recarregar a página, por exemplo, o usuário logado não caia fora e seja necessário um novo login. Por isso, colocamos a montagem do app.vue dentro de uma função do firebase:



**PERSISTINDO A AUTENTICAÇÃO DE USUÁRIO**

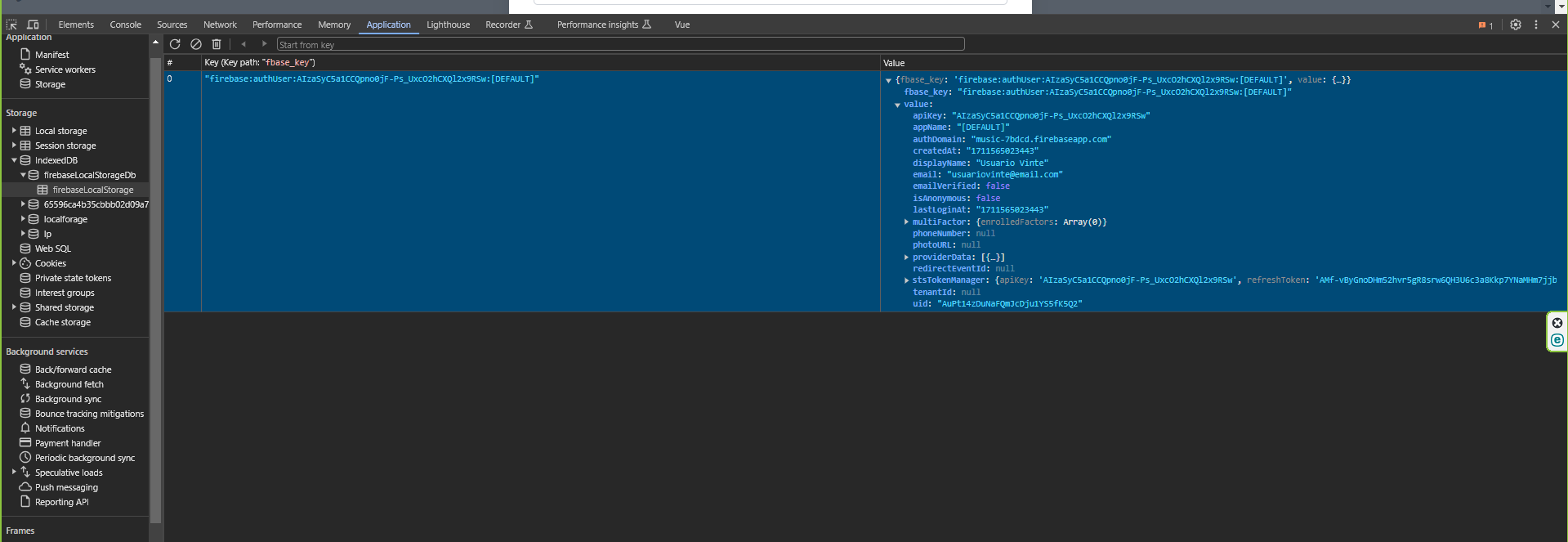
O app.vue é considerado o componente root, pois é o primeiro componente inicializado.

Para persistir a autenticação, fizemos basicamente o seguinte:



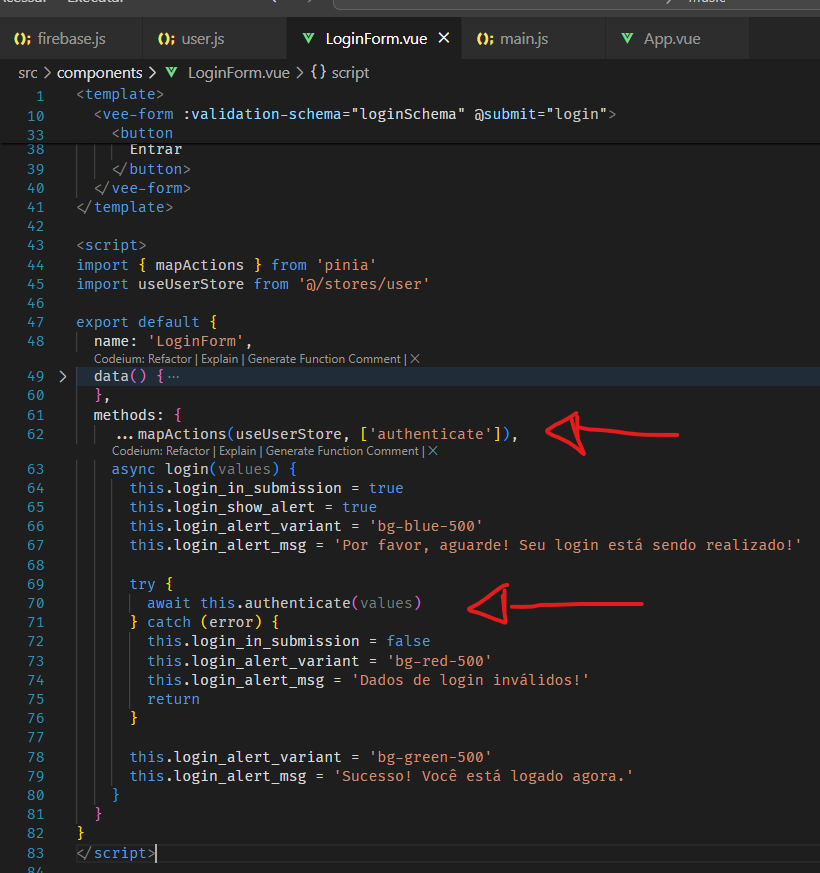
1. Importamos o mapWritableState, useUserStore e o auth.
2. Montamos uma computed na store useUserStore, escolhendo para alterar a propriedade userLoggedIn.
3. Feito isso, chamamos a função created() e adicionamos uma validação, se o auth do firebase tiver um usuário atual, o userLoggedIn do useUserStore passa a ser true.

No navegador, podemos verificar se o firebase possui algum usuário atual, no seguinte local:



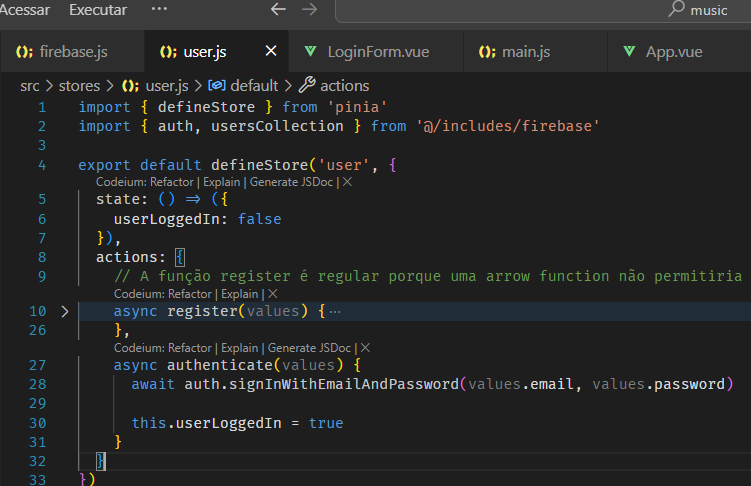
**CONFIGURANDO O LOGIN**

Acrescentei isso:



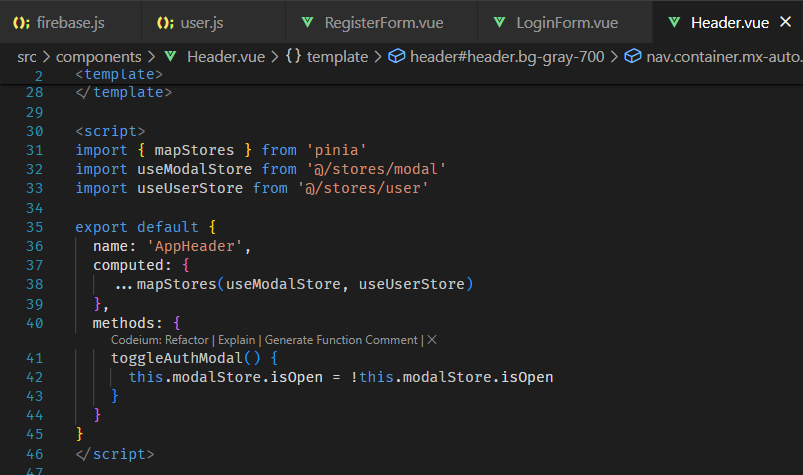
Fiz um mapActions na useUserStore, pegando a função authenticate. Em seguida, dentro da função de login, realizei um bloco try/catch passando para função authenticate os valores do formulário de login.

Tudo o que acontece em seguida é isso:

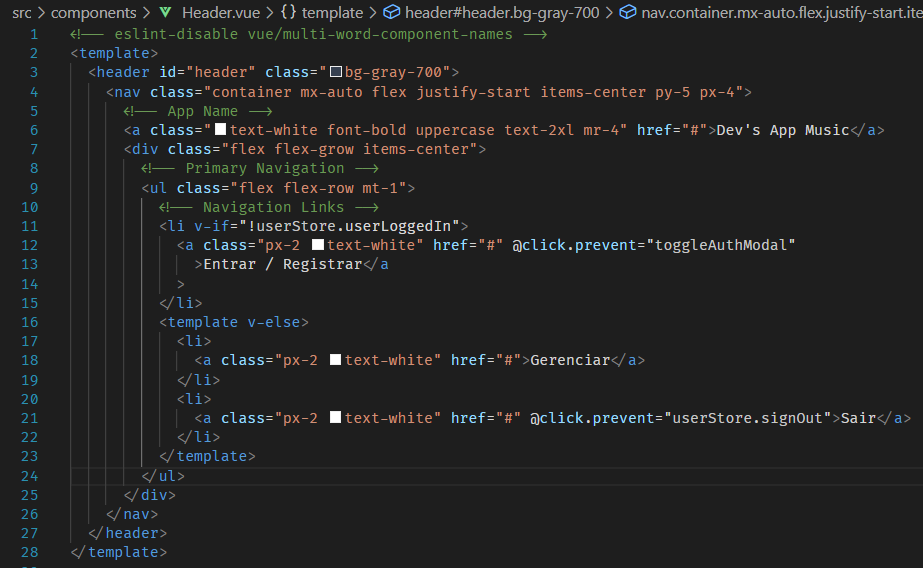


Se o retorno der true na validação do e-mail e da senha quer dizer que o usuário está autenticado, senão não.

**DESCONECTANDO**

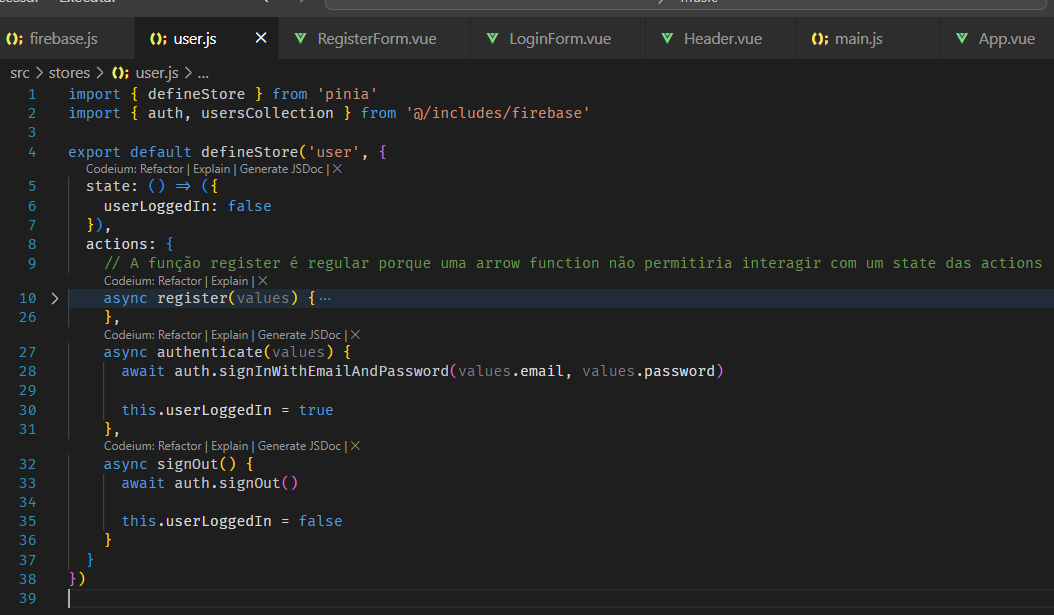
****

Importamos o useUserStore, local onde está a função de signOut, e acrescentamos essa store dentro da computed, onde tem um mapStores sendo feito já com a useModalStore.



Na li onde exibe o entrar/cadastrar, adicionei uma diretiva if para verificar se é true a variavel userLoggedIn, veja que estou acessando a store e a variável assim: userStore.userLoggedIn. Se for verdadeiro que a variável é falsa, vai exibir o entrar/registrar.

Na li de sair, tem um @click que também vai em direção à store userStore, acionando a função signOut:



Além de executar a função do firebase para a auth, também atribui como false o userLoggedIn.