**CREATING A NEW PROJECT**

npm init vue@latest

Project: name => music

Add TypeScript => No

Add JSX Support => No

Add vue Router for Single Page Application development => Yes

Add Pinia for state management => Yes

Add Vitest for Unit Testing => Yes

Add Cypress for End-to-End testing => Yes

Add ESLint for code quality => Yes

Add Prettier for code formatting => Yes

cd music

npm install

**REVIEWING THE FILES**

* **.eslintrc.cjs**

Esse arquivo contém as definições de configurações para o ESLint.

Root => Especificará que o arquivo de configuração raiz está sendo definido. Importante saber que o ESLint sempre buscará aplicar as configuraç~eos dos diretórios internos ao invés do raiz.

Extended => São as regras do ESLint.

Overrides => Regras específicas para arquivos específicos.

* **cypress.config.js**

Arquivo com as configurações do Cypress.

Cypress é uma ferramenta para realizar testes em uma aplicação.

* **package.json**

Dentro deste arquivo existe uma extensa lista de dependências.

* **.vscode (pasta)**

Pode apagar ele.

* **public**

Pode manter ele.

**O QUE É TAILWIND?**

* Não vem com elementos de User Interface.
* Possui foco na utilidade, faz uso de classes.
* Fácil de memorizar/lembrar o que cada classe faz.
* Alta curva de aprendizado.

**INSTALLING TAILWIND**

**npm i -D tailwindcss postcss autoprefixer**

**npx tailwindcss init -p**

* **tailwind.config.js:**

**/\*\* @type {import('tailwindcss').Config} \*/**

**export default {**

**content: ['./index.html', './src/\*\*/\*.{vue,js,ts,jsx,tsx}'],**

**theme: {**

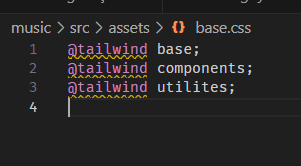
**extend: {}**

**},**

**plugins: []**

**}**

* **base.css (fica em src/assets/)**

****

**LOADING ASSETS**

Se tiver ja criado os estilos css e vai querer importar para o seu projeto, lembrar de colocar esse arquivo .css dentro do main.js. Outro detalhe importante é que o assets e as imagens precisam nesse caso ser colocadas dentro do public.

**ENTENDENDO ESTADOS**

Estados referem-se aos dados da sua aplicação. Tipicamente recuperado de um banco de dados ou API. Também são dados de aplicação: dropwdown, modal e current page.

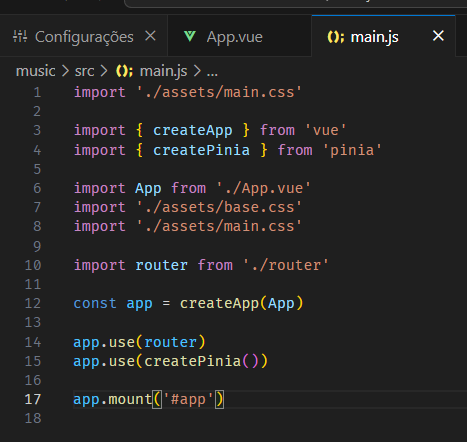
Um local onde todos os seus dados são armazenados e gerenciados. Pinia é a escolha mais popular para Vue. Esse estado é diretamente acessado pelos componentes.

**REVISANDO A CONFIGURAÇÃO DO PIN**

npm install pinia

Instalar plugins em projetos vue dependem de 2 passos: primeiro importar e depois registrar. Todo plugin precisa ser registrado antes de criar a instância Vue. Registrar um plugin é simples: basta passar como um objeto para a função use.

Este aqui é um detalhe muito importante sobre o Pinia: para que ele apareça no devtools, a instância dele tem que ser após o router e antes do mount:



* **STORES**

É uma abreviação para armazenamento.É uma localização única para armazenar dados. Podemos pensar nas lojas como armazéns. Uma armazém pode armazenar vários itens. E é comum que sejam subdivididos em seções específicas.

* **counter.js**

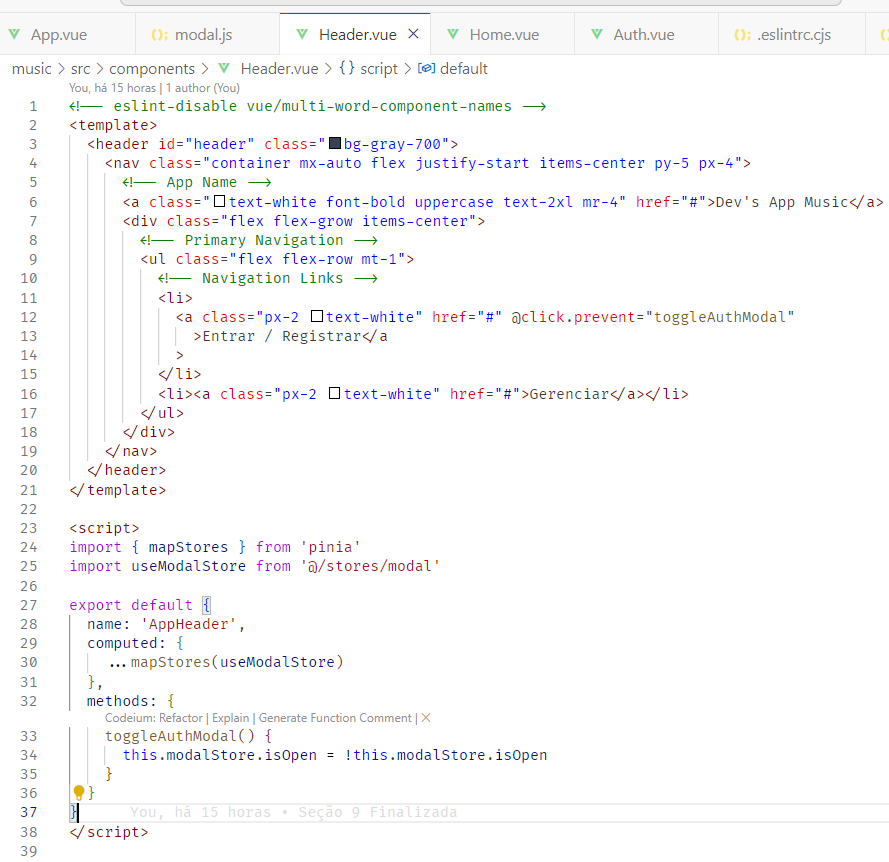
Neste arquivo é importado uma função do Pinia, chamada defineStore. Essa função irá criar um tipo especial de objeto, que terá os seus dados rastreados. No caso deste projeto, não o usaremos então pode deletá-lo.

**USING GETTERS**

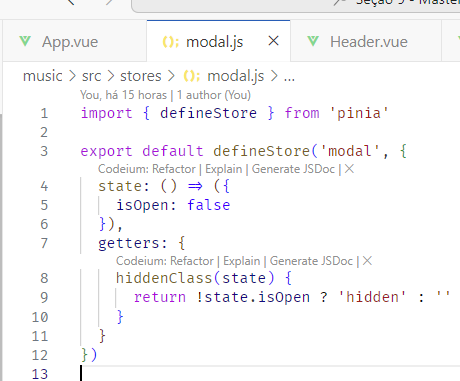
Pense neles como computed properties para sua store.

Acessível em todos os componentes. Eles somente irão atualizar se o estado mudar.

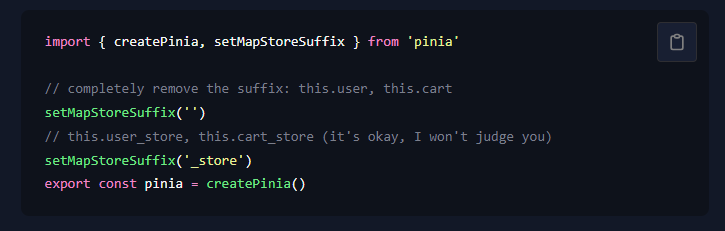
**DEFINIÇÃO DO CÓDIGO**

****

Vamos analisar o trecho de código acima: No export default, foi declarada a propriedade computed, onde foi passado …mapStores(useModalStore), basicamente isso faz o seguinte: pegará a store e acessará todo seu conteúdo. Se tivermos mais stores, basta colocar uma vírgula e digitar o nome das outras. Não é um array ali, são os nomes das stores. No caso de methods, temos uma função toggleAuthModal e o que acontece ali dentro não é uma surpresa, modalStore recebe sempre a negação dela mesma. Mas, de onde vem esse nome modalStore? Na importação chamamos de useModalStore, no mapStore também… Então, existe algo no Pinia que é automático: quando criamos essa store, declaramos ela dessa forma:

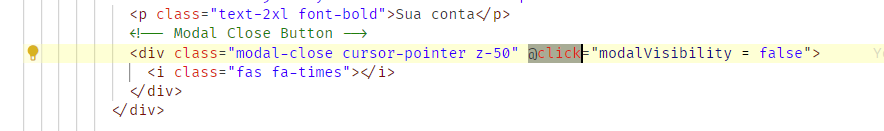


Observe que ao exportarmos modal.js, estamos declarando a defineStore. O primeiro parâmetro dela é o id, neste caso ‘modal’. Quando realizamos um mapStore em uma store, ela irá automaticamente montar id + store, e como a linguagem é camelCase, modalStore vem dessa montagem. Podemos alterar a nomenclatura se fizermos o seguinte:



O código acima fará com que uma modal seja aberta. Para fechá-la, vamos analisar o código abaixo:

O arquivo auth.vue possui o seguinte trecho:



Perceba que no @click passamos uma função, chamada modalVisibility e atribuímos seu valor como false.



A modalVisibility é o nome que foi escolhido nessa função para se referir ao invés da nomenclatura modalStore, e ‘isOpen’ é o seu valor. Ao clicar, o valor passado é false, isso faz com que não seja mais isOpen, e portanto, não tenha mais o valor de ‘escondido’:

