**INTRODUÇÃO À TESTES**

Os testes em uma aplicação são para checar se tudo está funcionando como o esperado.

**Tipos de Testes**

1. **End-to-End (E2E):** Tipo de teste usado para verificar se o comportamento da aplicação do início ao fim está correto.

Ex:

1. Abra o navegador
2. Carregue a url
3. Pressione 1 botão
4. Pressione o botão plus
5. Pressione 1 botão
6. Pressione botão de igual
7. Selecione o elemento que apresentou o resultado
8. Compare os valores dos 2
9. Fim

Problemas com o E2E:

* Ele é lento. Testes podem levar uma hora dependendo do tipo.
* Dificulta o debug porque você pode não saber exatamente o porquê o teste falhou.
* Alguns testes falham porque o serviço/API que você está usando pode estar inativa, e isso não seria um problema de fato na aplicação.

**2) Unit Testing:** Testes para pequenas seções de sua aplicação. Como por exemplo funções ou componentes.

Vantagens:

* Rápidos
* Ajuda desenvolvedores entenderem melhor o propósito de uma função

Desvantagens:

* Torna difícil refatorar o código
* Você pode testar somente seções individuais do seu código. Impossível testar se múltiplas seções podem trabalhar juntas.

**3) Snapshot Testing**

* Torna difícil refatorar o código
* Você pode testar somente seções individuais do seu código. Impossível testar se múltiplas seções podem trabalhar juntas.

Os testes mais recomendados são os units.

**INTRODUÇÃO AO VITEST**

Ferramenta oficial de testes para o vite. Boas alternativas são: Mocha, Jasmine e Jest.

**ADICIONANDO O VITEST UI**

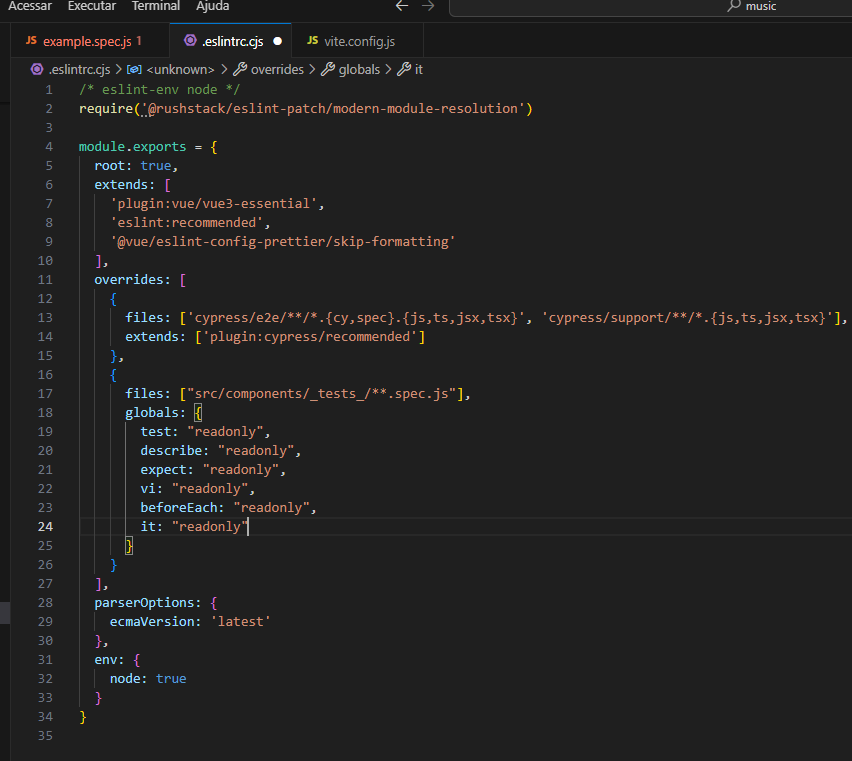
npm run test:unit

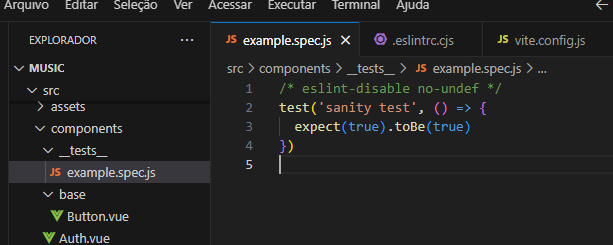
npm i -D @vitest/ui

npm run test:unit

**ESCREVENDO NOSSO PRIMEIRO TESTE**

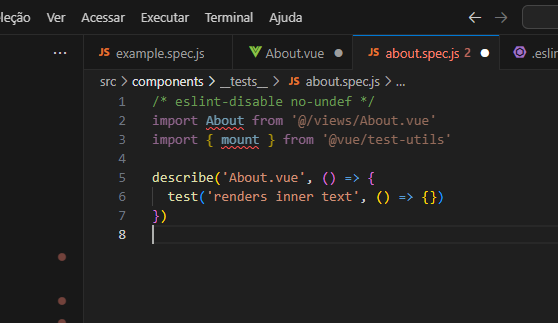
Inicialmente adicionei algumas propriedades no eslintrc.cjs para ignorar os arquivos .spec.js dentro da pasta de testes:





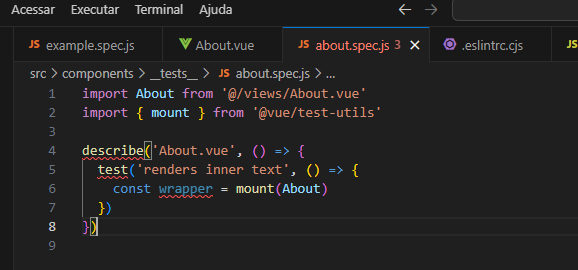
Considere o texto acima. A string é o nome do teste, e dentro da arrow function tem o esperado e o que é realmente.

**MONTANDO COM UTILITÁRIO VUE TESTE**

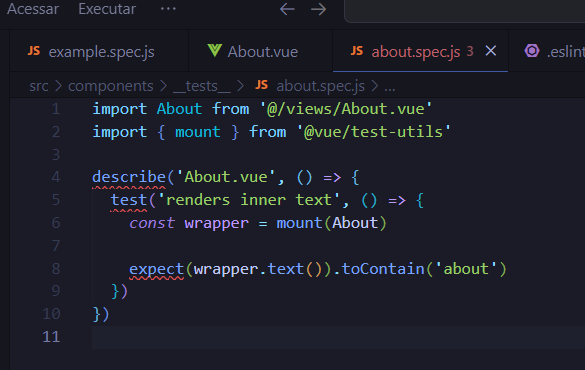


ATENÇÃO!

Componentes não são instâncias do Vue. Uma função construtora é o que cuida da criação de uma nova instância a partir de um objeto. Os componentes não são enviados com uma função construtora, por isso não poderemos montá-los. Essa questão pode ser resolvido usando a biblioteca de Utilitário de Testes do Vue.

****

**TESTANDO CONTEÚDO INNER**

****

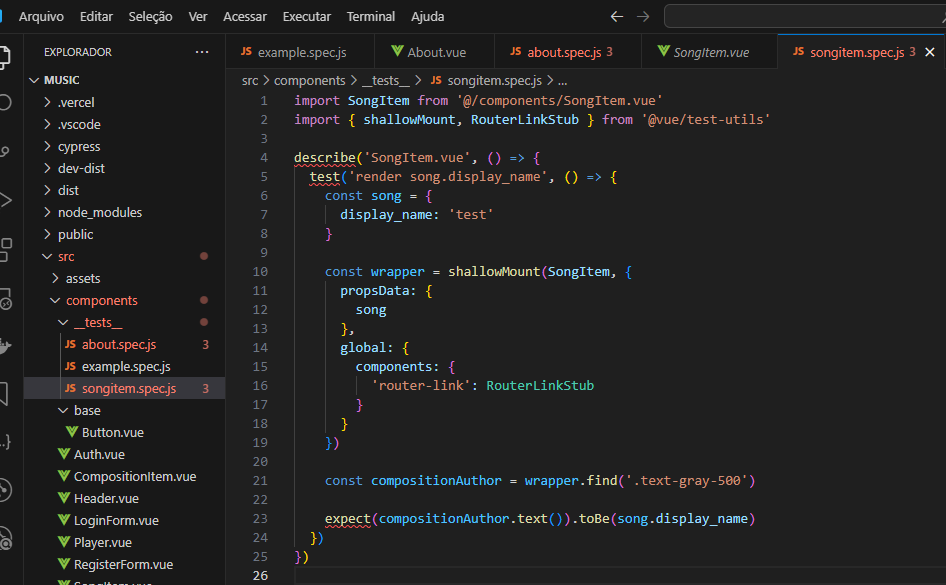
No lugar de mount, pode ser utilizado shallowMount. A diferença entre mount e shallowMount é que esse último só permite um nível de describe. O mount permite aninhações de testes e faz um carregamento de biblioteca mais complexo, por isso, sempre que possível, utilizar o shallow. Outro detalhe importante é que o mount e shallowMount aceitam um segundo argumento, além do componente, então no caso do mount, pode-se passar um segundo argumento em formato de objeto com a propriedade shallow true o que equivale com a shallowMount, veja:

const wrapper = mount(About, {

shallow: true

})

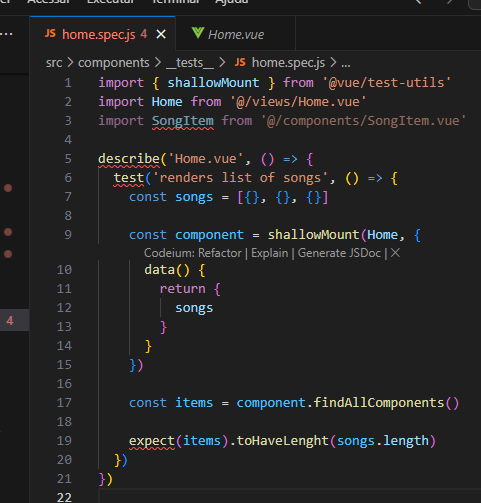
**PASSANDO DADOS PELO COMPONENTE**

****

**EVITE ASSERÇÕES BOOLEANAS**

A asserção booleana é quando se compara valores booleanos.

**TESTANDO COMPONENTES FILHOS**

****

**MOCKING METHODS**

Mocking é o processo de criar uma feature fake para poder fazer os testes.

**TESTANDO ATRIBUTOS**

**TESTANDO ACTIONS PINIA**

**MOCKING PROMISES**

**TESTANDO COMPONENTES ROUTER**

**TESTANDO SNAPSHOT**

É quando duas imagens de uma aplicação são comparadas. As imagens não são confiáveis porque diferentes navegadores e tamanhos podem causar um teste falho. O Vitest irá serializar o valor, isso significa basicamente que qualquer valor JavaScript pode ser convertido para uma string.

Regras gerais para testes snapshot:

* deve sempre passar várias vezes se não foi alterado o teste ou a aplicação
* altamente suscetíveis à falhas. Portanto, limite o seu teste snapshot para 1 à 3 por componente.

Caso tenha alterado algo no arquivo original do componente que está sendo testado, por exemplo, foi adicionado uma nova classe, basta apertar a tecla u para atualizar o snapshot.

**VISÃO GERAL SOBRE TESTES E2E**

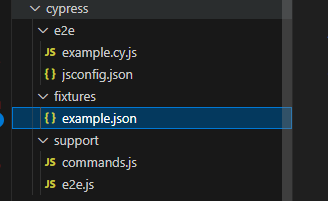
É uma forma de testar se uma aplicação está se comportando corretamente ao automatizar um navegador para imitar o comportamento de um usuário, criando um programa que pode clicar links, escrever em formulários, scrollar a tela, etc.

Problemas com E2E:

* Lento
* Não confiável
* Difícil para debugar

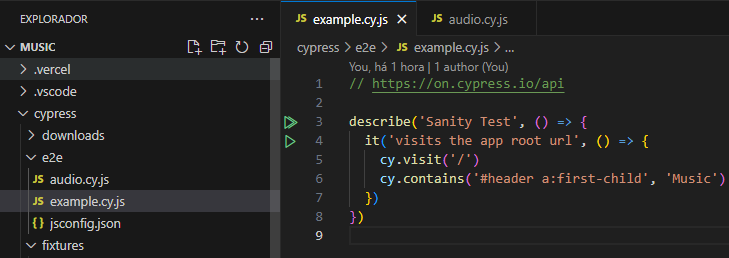
Vantagens:

* Extremamente útil para ver a aplicação totalmente funcional sem ter que testar tudo manualmente
* Permite ver se tudo está funcionando junto, ao contrário do unit e snapshot.

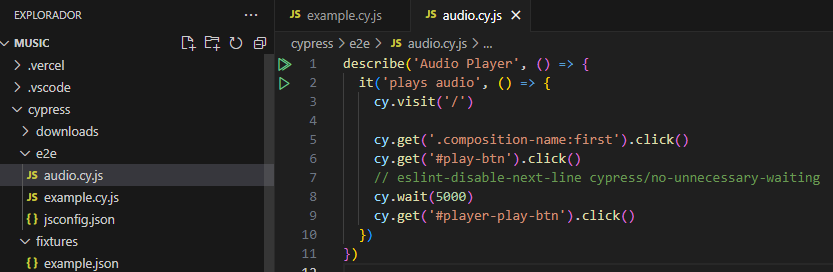
****

Na pasta e2e, ficam as configurações do cypress. Em fixtures, ficam dados fictícios, como por exemplo, dados de formulários para serem preenchidos, e em support ficam os comandos adicionais para serem rodados.

**ESCREVENDO TESTES COM E2E**

****

Esse teste com cypress verifica se na rota / no cabeçalho, tem uma tag a como primeiro-filho em que sua descrição é ‘Music’.



Esse outro teste vai interagir com elementos de play e pause na tela, aguardar alguns segundos.

Para executar os testes e poder interagir com o cypress no navegador, é necessário rodar o build, o servidor de testes e também escolher o tipo de teste:

npm run build

npm run preview

npm run test:e2e