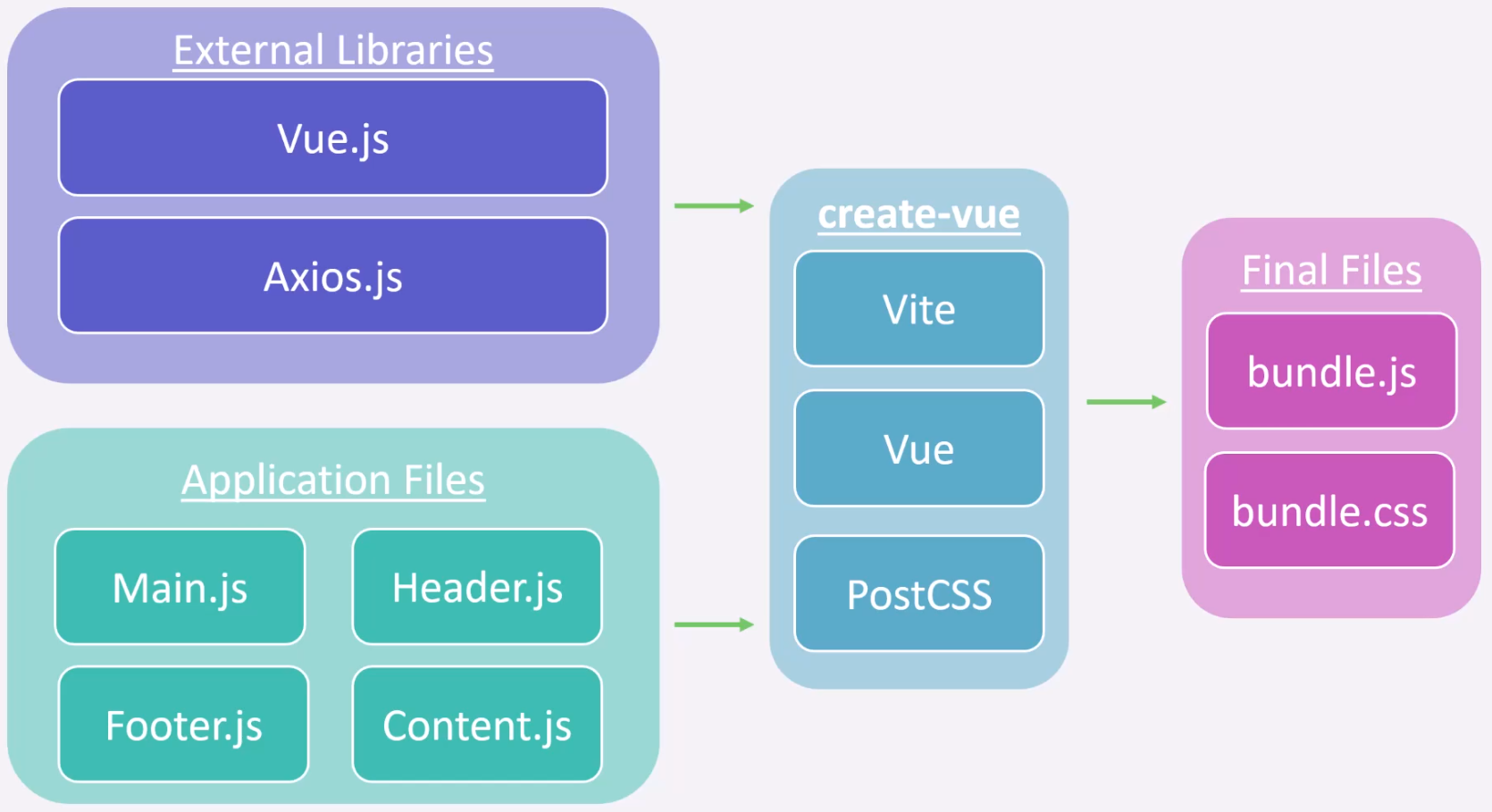
**SCAFFOLDING A VUE PROJECT**

****

**npm init vue@latest**

em seguida, responder as perguntas exibidas. Ir para a pasta raiz do projeto que foi criado.

**npm install**

**npm run dev**

**LINKS IMPORTANTES:**

[**https://vuejs.org/**](https://vuejs.org/)

**SIDEBAR: UNDERSTANDING SERVERS**

Uma das maiores diferenças entre um servidor e nossos computadores é a No GUI (No Graphic User Interface). Em geral nos servidores, nós não podemos clicar em ícones, botões ou interfaces gráficas. Um servidor pode ter softwares instalados e também lidar com requisições de rede. O trabalho do servidor é ouvir as solicitações de rede.

Quando digitamos um endereço na barra de endereços, o servidor é responsável por responder com algum tipo de dado, seja ele uma imagem HTML, CSS ou arquivo JavaScript. O grande benefício disso, é não precisarmos comprar um hardware e montarmos um servidor.

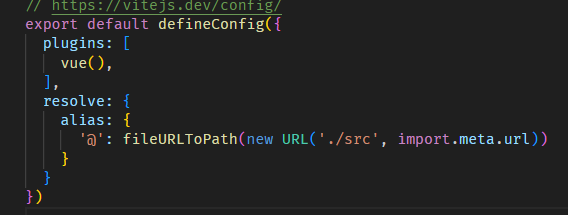
**REVIEWING THE FILES**

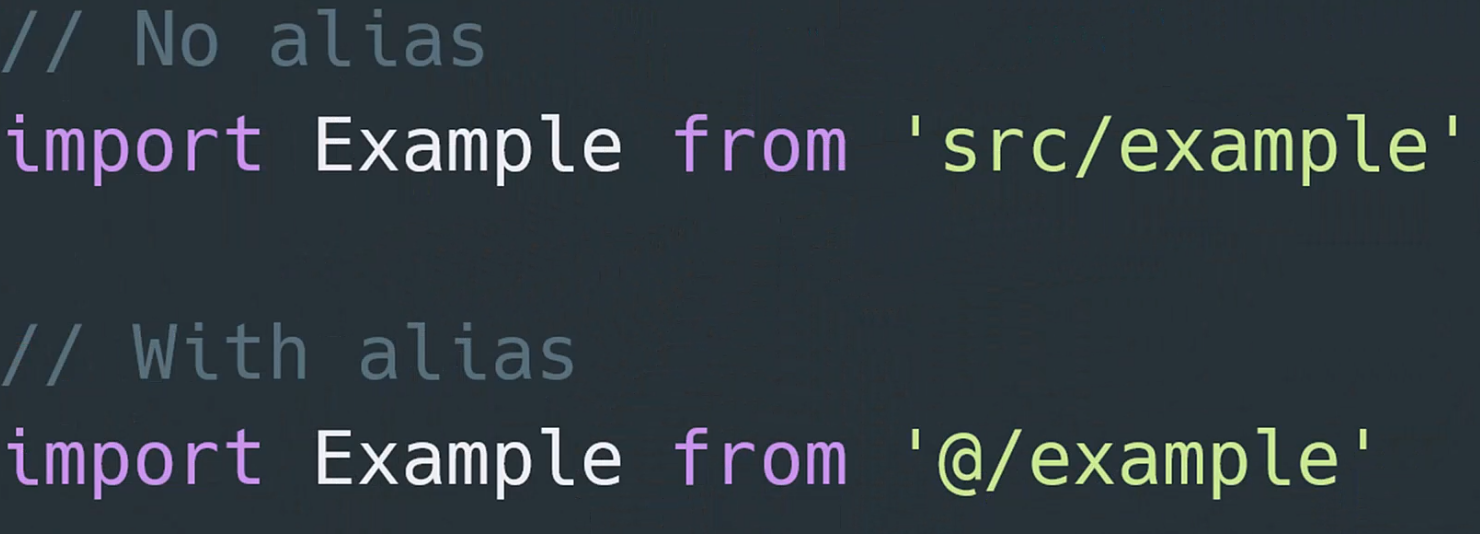
* **.gitignore**

Instrui o git sobre quais arquivos devem ser ignorados.

* **vite.config.js**

Está sendo exportado um objeto como parâmetro da função defineConfig, e nele dentro do resolve, existe um alias. Ali é possível definir os apelidos para as pastas do projeto para não precisar sempre escrever todo o diretório:





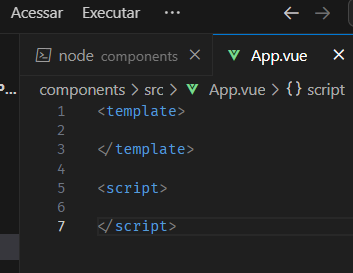
Para visualizar os arquivos com extensao vue, instalar a extensão volar no vscode.

**CREATING COMPONENTS**

Excluir as pastas assets e components dentro de src.

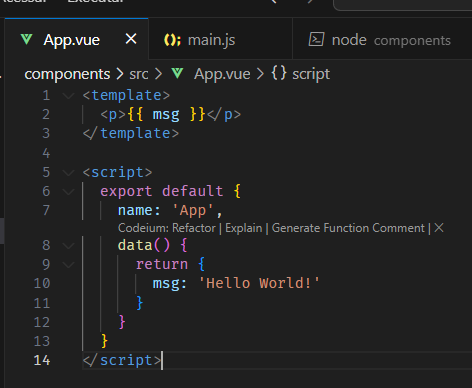
No arquivo main.js, remover a instrução de importação para o CSS também.

Deixar o app.vue somente com as tags de template e script:

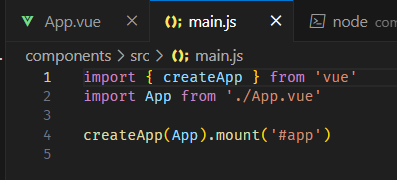


No export default, dentro das tags de script, é onde irão computed properties, data, watchers, etc. A propriedade name não é obrigatória, mas o Developer Tools usa ele para nomear o componente.

Exemplo:



**O main está assim:**

****

**CHILD COMPONENTS**

**Instalar a extensão “Vue VSCode Snippets” sarah**

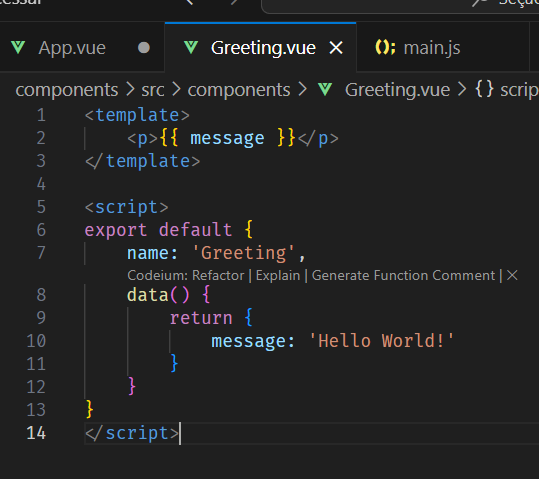
**Existem duas maneiras de carregar um componente:**

1. **Global**

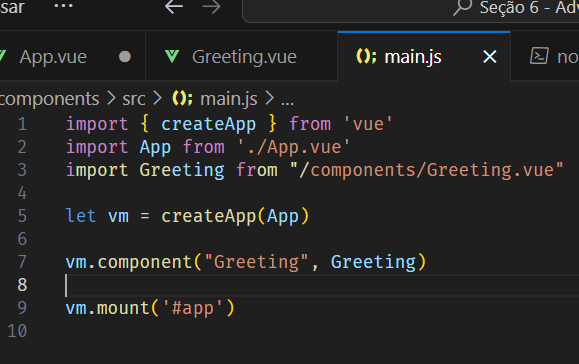
**Podem ser utilizados em qualquer lugar, incluindo componentes filhos. No entanto, seu uso não é muito recomendado.**

**Para utilizá-los, basta criar o componente em um arquivo separado, e ir no arquivo main.js, importar o componente lá e após instanciar o createApp, passar o componente como parâmetro da component, veja:**

**Componente criado:**

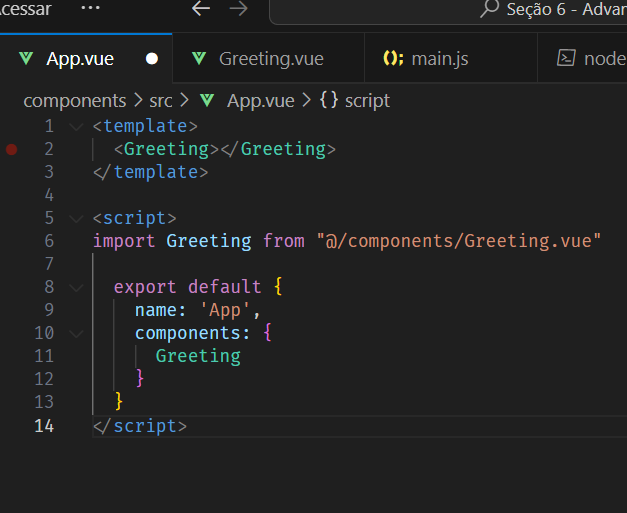
****

**Componente no main.js:**

****

**Detalhe importante: ele tem que estar antes do .mount!**

1. **Local**
   1. **Primeiro vamos no app.vue, e dentro da tag de script importamos o componente. No export default, digitamos à nível do name o components (seu conteúdo é json) e digitamos o nome do componente. Para podermos usá-lo, abrimos uma tag de abertura e fechamento com o nome do componente, dentro das tags de template.**

****

**COMPONENT STYLES**

**“In the future, browsers will support something called the Shadow Dom.” - 1:42**

O principal benefício do shadow dom é que ele vive no seu próprio mundinho. Quaisquer alterações feitas em determinada parte da página só serão refletidas naquela parte. Ele é uma forma de isolar os componentes do regular DOM, com seus próprios estilos, eventos e estrutura.

O virtual Dom é um objeto javascript que é uma cópia do Dom real.

Adicionar o atributo ‘scoped’ na tag de style do componente sinalizará que os estilos serão aplicados somente para componente em questão.

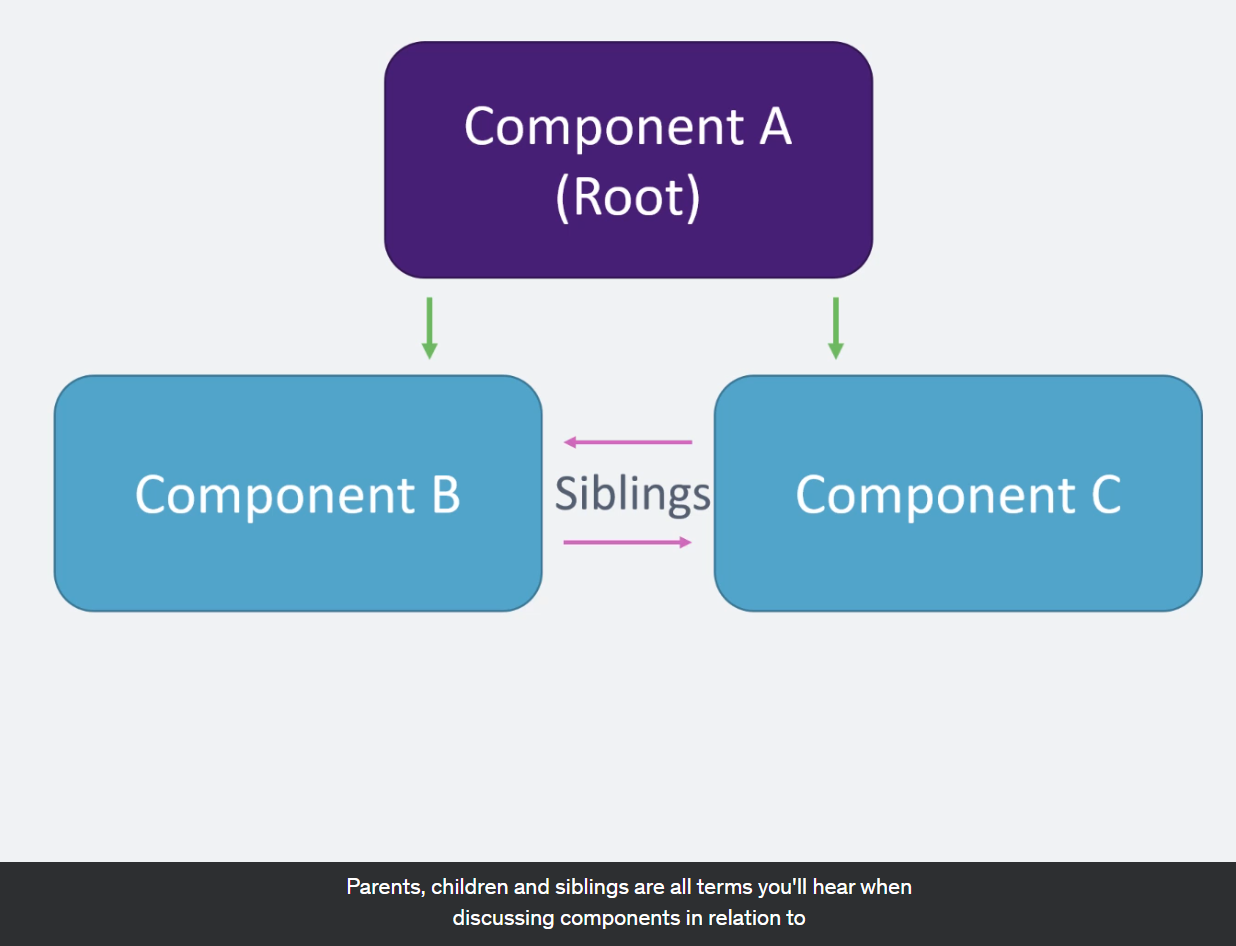


**USING SASS IN COMPONENTS**

**npm install sass –save-dev**

**Em seguida, ir até o componente, na tag style e após o scoped, adicionar lang=”scss”.**

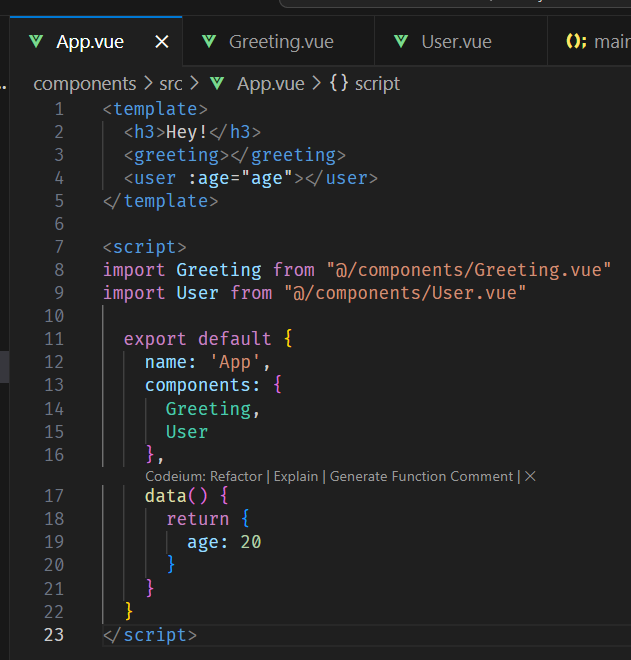
**COMMUNICATION BETWEEN COMPONENTS**

****

**PROPS**

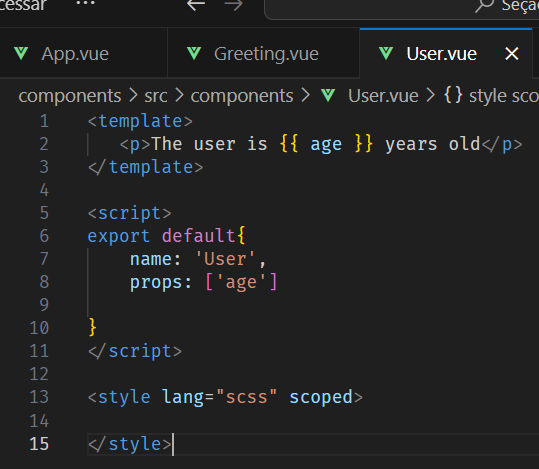
Props são como atributos customizados para os componentes.

Vamos utilizar como exemplo o componente App:



Perceba que o componente user possui um atributo do tipo bind. O nome escolhido como chave pode ser qualquer um. O valor que ele recebe, obrigatoriamente tem que ser igual ao nome do data, eles se conectam.

Além disso, para poder usar esse atributo que está sendo passado na tag user como bind, tem que fazer a seguinte configuração dentro do componente:



A primeira coisa a se fazer é declarar uma prop contendo um array, e o valor colocado ali obrigatoriamente tem que ter o mesmo nome que a chave do bind do exemplo acima. O prop de forma geral diz quais tipos de binds o componente pode receber. Para usar o valor da age, basta colocar entre as chaves duplas o nome como ele está dentro do array. Se o valor da idade for atualizado lá no componente App.vue, ele será automaticamente atualizado também dentro do componente de usuário.

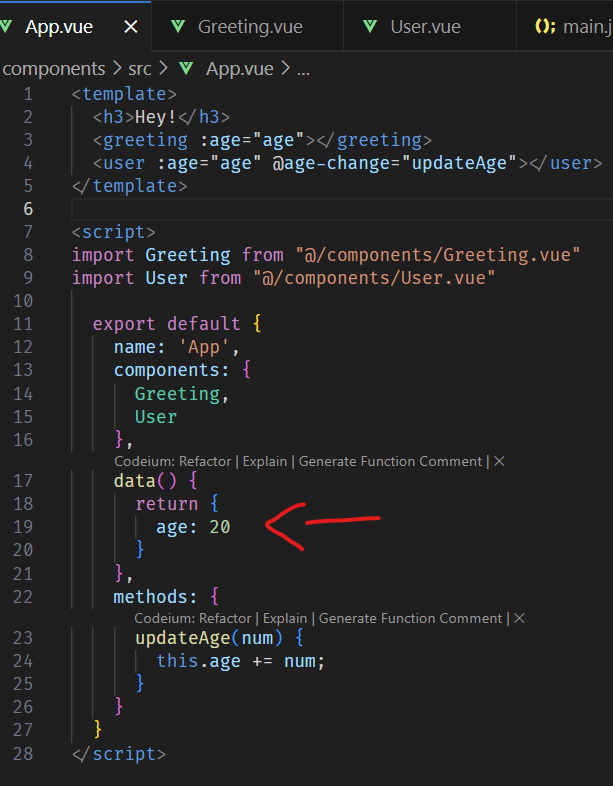
**LIMITAÇÕES DAS PROPS**

Como vimos, é bem tranquilo enviar informações por props dos componentes parentes para seus childrens. No entanto, o contrário não ocorre.

**EMITTING EVENTS**

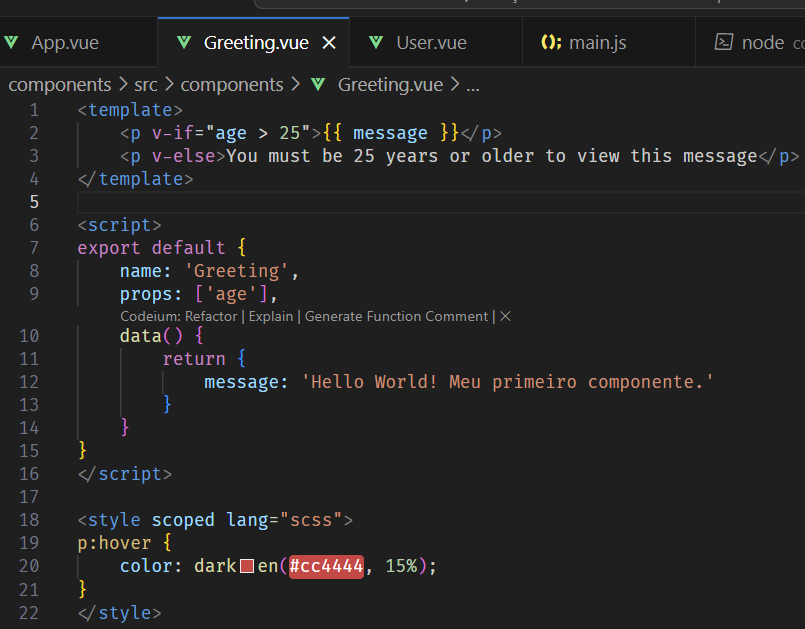
Emitir eventos é uma possibilidade para poder enviar dados de componentes childrens para simblings e parentes.

Vamos utilizar como exemplo o age que temos lá no app.vue:



Para enviá-lo para os componentes de greeting e user, estamos criando um bind :age e atribuindo como valor o nome exatamente igual como está no data. Como já vimos, isso permite manipular esse valor dentro dos outros componentes.

Vamos analisar o caso do greeting:



Perceba que a primeira coisa a se fazer para poder usar o dado é dizer ao vue que o componente do Greeting aceita receber uma propriedade chamada age. Por esse motivo, dentro do export default, abaixo do name, colocamos as props e dentro do array um valor de age. Esse valor age lembrando que tem que ser com o mesmo nome do bind que está lá no app vue.

O template do Greeting possui uma validação: se essa idade que foi recebida for menor que 25 é exibida a mensagem do data, caso contrário, é exibida a mensagem falando que para visualizar a mensagem é preciso ter mais de 25 anos. Uma das características do vue é ser reativo, então ao alterar o valor da age lá no app.vue, irá alterar também no Greeting e no User.

Vamos analisar agora o User:



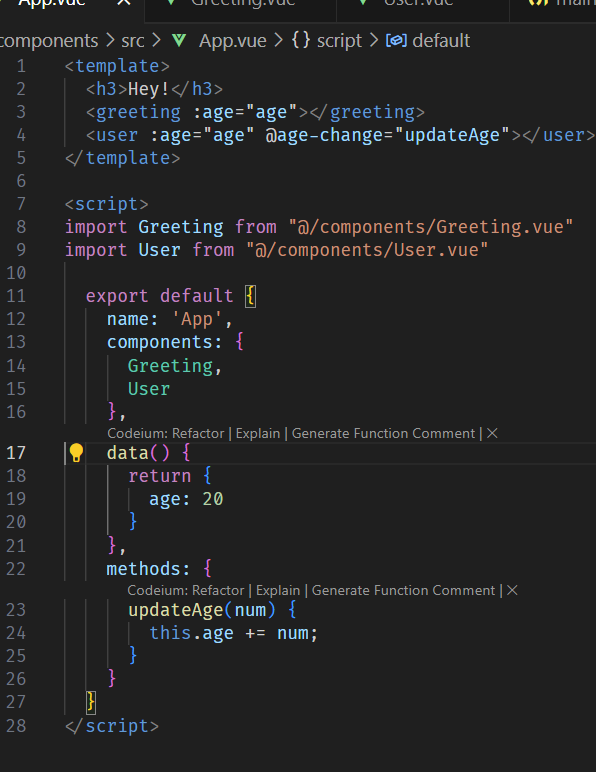
De maneira idêntica ao Greeting, para poder usar o valor passado por parâmetro na tag de user, precisamos dizer ao Vue que o user aceita reeber um parâmetro com o nome daquele bind, por isso colocamos ele no props. O valor de age é utilizado diretamente no template, no primeiro parágrafo: The user is {{ age }} years old.

Esse componente de usuário, além de exibir a idade do jeito que ela vem, também exibimos o valor multiplicado por 2 no parágrafo debaixo daquele The user is… . Para fazer isso, é simples: basta criar nos methods uma função que retorne a idade \* 2, e é exatamente o que fizemos: criamos nos methods uma função chamada ageDoubled() e retornamos aquilo que precisamos. Para exibir esse valor, basta chamar entre chaves duplas lá no template.

Agora, tem um detalhe: isso aqui funciona super bem quando o botão está no componente pai, que neste caso é o app.vue. Como o componente está em um children, precisamos avisar o componente pai que ele mudou, para que o componente pai avise o greeting que a idade mudou e o quanto ela mudou. É bem simples, veja:

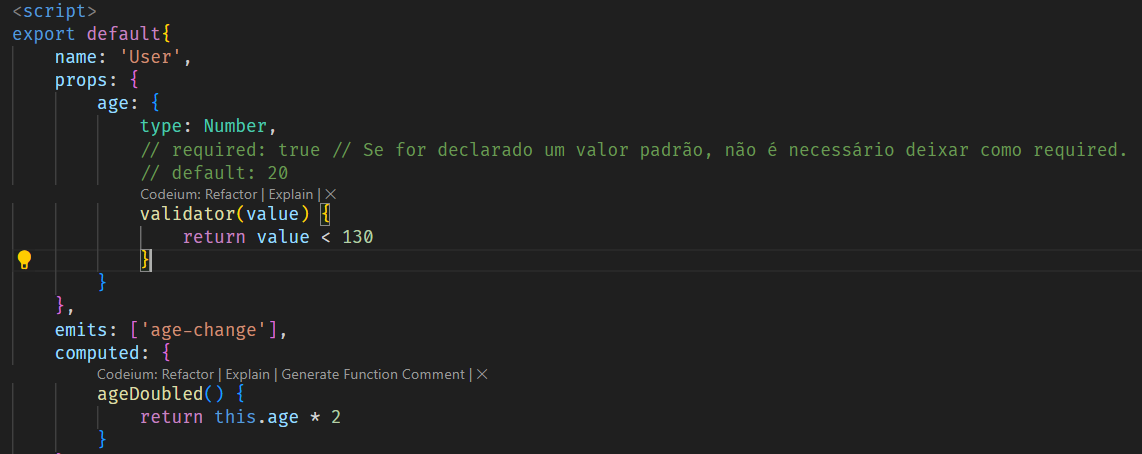
Vamos criar uma função chamada onClickAge lá no @click do botão. Essa função fica nos methods do User, e ela faz SOMENTE uma coisa: ela emite um evento que pode ser escutado. Veja que na função só tem a seguinte linha:

this.$emit(‘age-change’, 3). Pois bem, o primeiro parâmetro é o nome do evento, e o segundo é o que a gente quer enviar para quem estiver escutando. Nesse caso, estamos dizendo que o nome do evento é ‘age-change’ e que estamos enviando o número 3. Lembra que para atualizar a idade no greeting, precisamos que o app.vue saiba que o número mudou? Veja novamente o código do App.vue:



Nós nomeamos o evento do user.vue como ‘age-change’, para pegarmos esse evento e o seu valor, basta ir no componente que está emitindo o valor e colocarmos @nomedoevento e uma função nova. Nesse caso chamamos a função que vai tratar o que está sendo escutado de updateAge. Ela se encontra nos methods do app.vue, e o que ela faz é pegar o num (nome que escolhemos dar ao parametro 3 que foi passado na user), e somar na idade do data. E dessa forma, o ciclo de reatividade continua: clicamos no botão de aumentar a idade que está no user.vue, ela envia um evento passando o parâmetro 3 para o componente pai app.vue, ele por sua vez pega esse parâmetro e soma na idade que ele tem e devolve para o user e envia para o greeting também.

**VALIDATING PROPS**

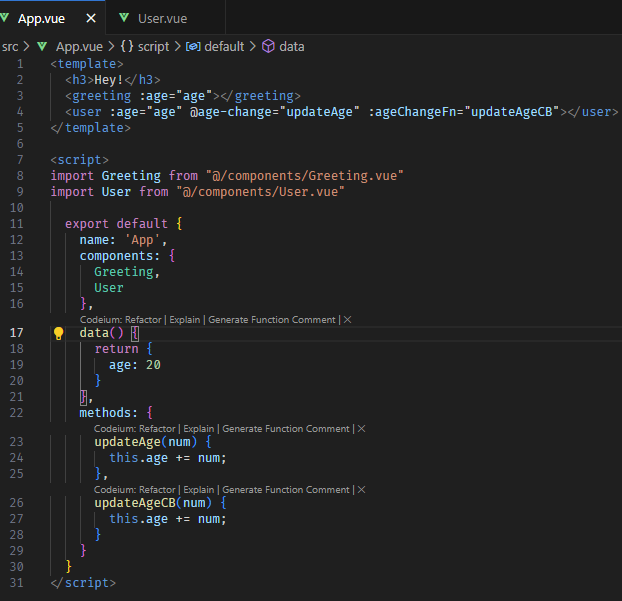


Utilizando como exemplo um recorte do user.vue, perceba que adicionamos algumas propriedades à prop de age. Configuramos o tipo de dado que será aceito para essa prop, e validamos se o valor dela é menor do que 130. Além disso, podemos escolher um valor padrão para o caso de essa prop não vir, e também podemos marcar essa prop como uma prop obrigatória. Importante destacar que se passarmos um valor como default, não é necessário e também nem faz sentido tornar o campo obrigatório, porque tendo um default, nunca será nulo.

**CALLBACK FUNCTIONS**

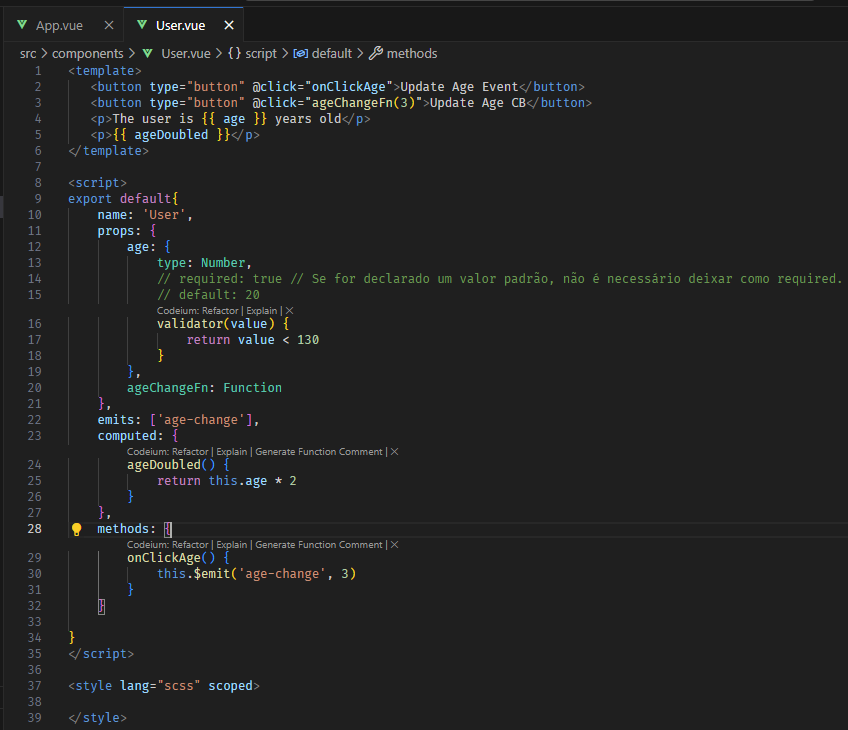
Como alternativa aos eventos, é possível enviar informações por meio de funções de callback:

Considere o app.vue:



Veja que passamos um bind chamado :ageChangeFn para dentro do componente de user, e atribuímos ao valor desse bind o retorno da função updateAgeCB. Essa função está nos methods e recebe um parametro ao qual chamamos de num e soma à idade.

Considere o user.vue:



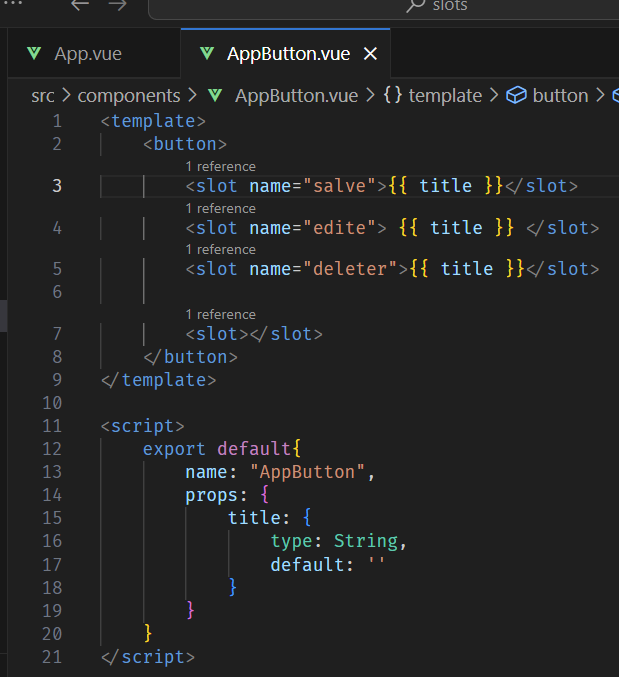
No user vue, a função que o app.vue enviou por bind é aceita na props e atribuído o valor Function a esta prop. Ao colocar essa prop em @click do botão de update age cb e passar um parâmetro, o que ocorre é que esse valor é enviado ao app.vue e lá a função que está nos methods irá tratar esse dado e retorná-lo. É uma outra forma de trocar informações entre os componentes.

**INSERTING CONTENTS WITH SLOTS**

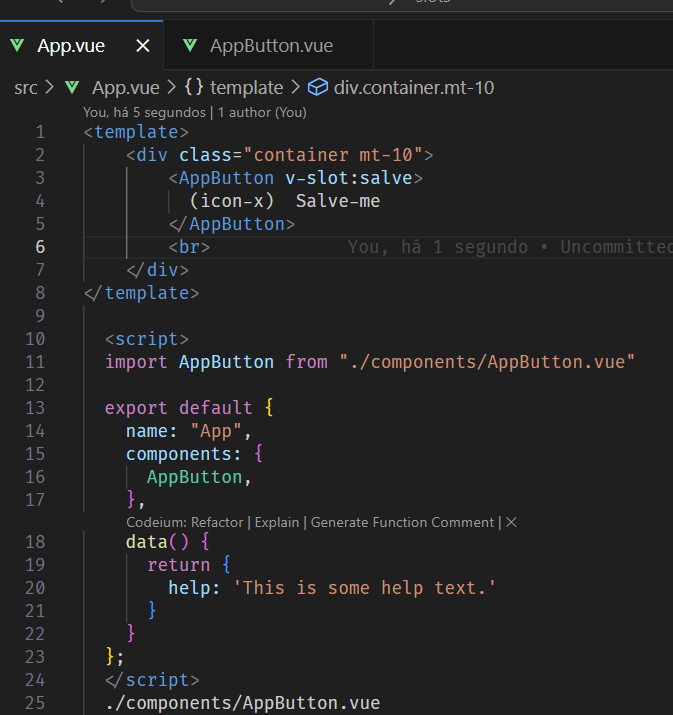
**npm init vue@latest**

**npm install**

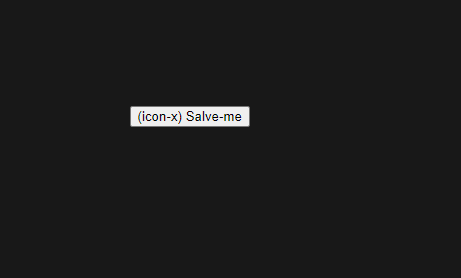
Slots são espaços que podem receber conteúdo. Vamos considerar um componente de button:



Perceba que esse botão possui 3 slots nomeados. Se lá no App.vue eu chamar apenas o slot que eu quero, só será exibido aquela parte que eu chamei:



O resultado na tela é esse:



Agora, se eu não escolher nenhum slot, virá toda a informação que está no componente de botão, e como na hora de chamar o botão no app.vue eu não passei nenhum slot, e o valor padrão no componente é nenhum slot, e existem 3 slots, o resultado será esse:

