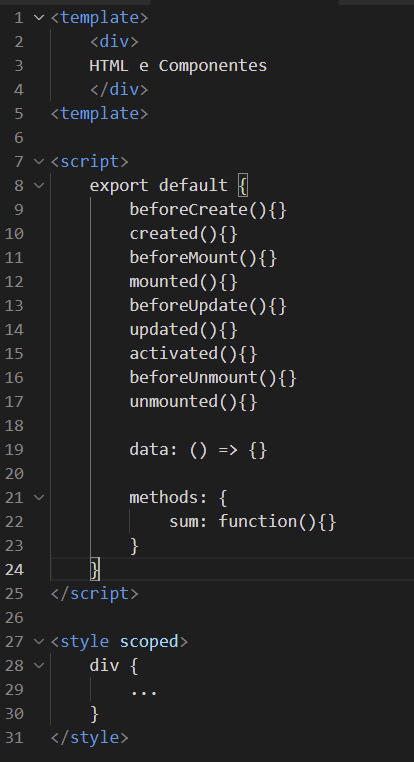
**´VUE.UE, VUEX e NUXT**

Vue.js é um framewrok que "atua” em cima de javascript e/ou typescript, facilitando o gerenciamento de elementos e estados.

A grande vatangem ao usar, é o Virtual DOM, que a ferramenta implementa. Se o código for feito corretamente utilizando os templates e componentes, a performance tende a ser superior ao uso de JS puro. Ao alterar o valor de uma váriavel, por exemplo, que tenha um bind com algum componente ou elemento, o próprio Vue sabe exatamente quais elementos devem ser alterados e utilizando o Virtual DOM, altera somente o que for realmente necessário, evitando de onerar processamento.

Um arquivo .vue, possui tudo que é preciso para ser independente. Possui a parte HTML, que é o seu template, possui o script, que é o JS ou TS que possui todo código para operar a página ou o componente, e possui o style, que é o CSS, que pode ser escopado somente para a página/componente em questão.

Normalmente teremos um “esqueleto” de arquivo .vue, dessa forma:



Os primeiros métodos da parte do script, são os métodos de lifeCycle, ou seja, são chamados pelo Vue como callback de cada etapa da renderização da página/componente.

Vue.js utiliza a ideia de criar componentes para serem reutilizados, como um input por exemplo. Criamos ele com a estrutura conforme o exemplo acima, e exportamos o input com algum nome que definirmos. Em outras páginas, podemos importar (import MeuInput from ...) e usar como tag no HTML <MeuInput ...></MeuInput>. Dessa forma ganhamos escalabilidade e temos uma forma de bind também.

**DIRETIVAS**

Alguns exemplos de diretivas em Vue, são:

**V-ON**  
Usamos o “v-on” como um vinculador entre elementos e seus listeners. Alguns “binds” usados em elementos:  
**v-on:click=”...”** -> faz vínculo entre o elemento clicado e a função que deve chamar quando click ocorrer.  
Outros que costumam ser usados: **v-on:keyup** | **v-on:mouseenter** | **v-on:focus** | **v-on:change**

Essas diretivas podem ser usadas com o atalho ‘@’, por exemplo: **@click=”...”** e também podem ser específicas, como o keyup ser acionado somente com Enter: **@keyup.enter="...”**

**V-BIND**

Usamos para fazer o vínculo entre uma propriedade de um elemento/componente com alguma variável. Por exemplo:

**<img v-bind:src=”imageSource”/>**

Acima, setamos o src de uma img com base no que estiver na váriavel/atributo imageSource.  
Exemplo em código:  


Se em algum momento a variável *imageSource* mudar seu valor, o elemento img terá seu src alterado também.

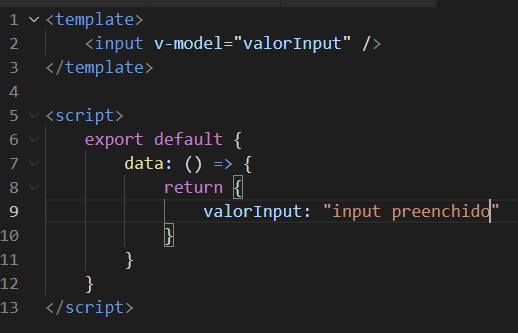
**V-IF, V-ELSE e V-ELSE-IF**

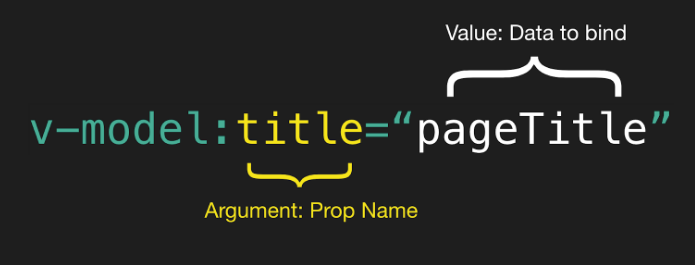
Usados para renderizar condicionalmente algum elemento ou componente.  


**V-SHOW**

Igual ao v-if, porém o elemente é renderizado, só tem sua propriedade ‘display’ setada para none em caso de não corresponder a true

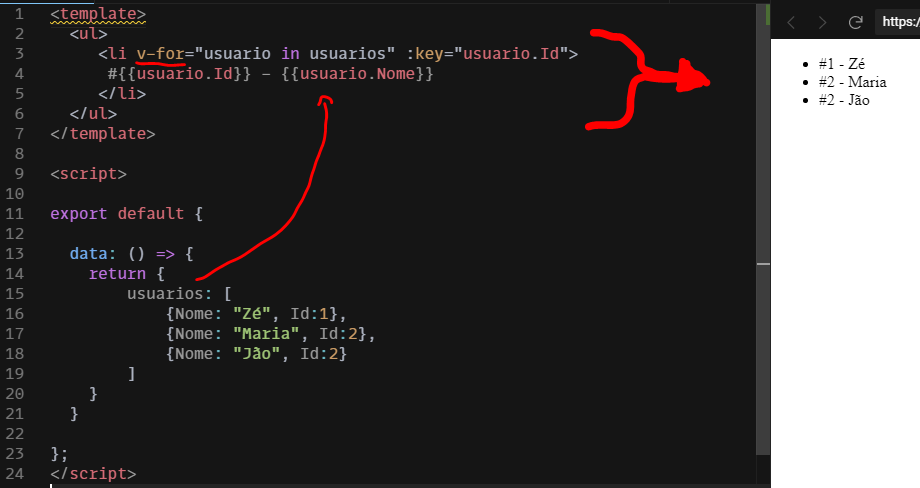
**V-MODEL**

Usado principalmente em inputs para vincular o value a uma variável. Até a versão 2.x é sempre vinculado a propriedade “value”, por exemplo:   
  
Acima, o bind é two-way, se usuario digitar no input, o valor da variável ”valorInput”, vai mudar, e se mexermos no código e mudar a mesma variável, o input terá seu value alterado.

Já na versão 3.x, podemos especificar qual propriedade será vinculada:  


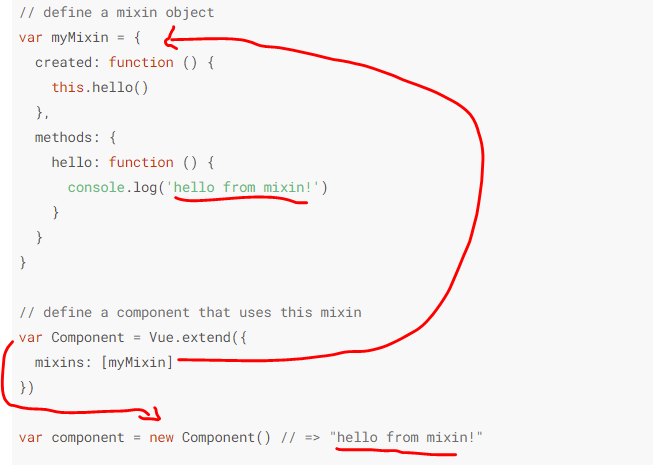
**V-FOR**

Muito útil para exibição de listas, pois irá criar o elemento/componente com os devidos atributos para cada um dos elementos do array que estiver sendo iterado, como no exmeplo:

Acima, temos 3 usuarios com nome e Id, com uso do v-for (onde setamos a chave única sendo a propriedade Id em :key=”...”) teremos 3 <li>, cada uma com os dados de cada um dos 3 usuários.

**MIXINS**

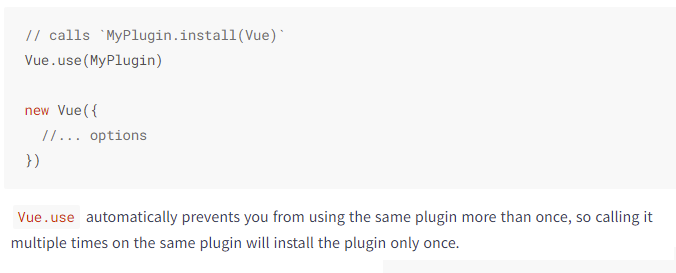
Mixins são uma forma de distribuir métodos reutilizáveis entre componentes Vue. Quando um componente usa um Mixin, ele “herda” (ou faz um mix) de todas propriedades/options desse mixin em suas próprias propriedades/options.



Um mixin pode ter qualquer propriedade de um componente Vue, seja hook de lifecycle, seja data, sejam métodos, quando outro componente consumir esse mixin, tudo isso será incorporado pelo componente.  
Se um componente que for consumir um MIxin, possuir uma propriedade igual ao mixin, mas com valor diferente, o valor que está no componente terá prioridade e não será sobreescrito.

**PLUGINS**

Normalmente são usados para adicionar funcionalidades a nível global na aplicação Vue. Plugins podem ser usados para adicionar métodos e propriedades a nível global; adicionar assets; adicionar libraries e/ou APIs.  
Para utilizar um plugin, é preciso chamar o use dele antes de instanciar a aplicação Vue.



**Diferença entre Mixins e Plugins:**

**Mixins** tendem a ser usados num nível de componente, onde um Mixin possui funções e propriedades que vão ser reutilizadas e incorporadas por Componentes, enquanto os **plugins** são mais globais, podendo ser uma lib de funções ou até mesmo uma combinação de mixins, acessíveis em todos os níveis da aplicação.  
Routes e State management são bons exemplos de utilização de plugin, pois tendem a ser globais e podem (ou não) afetar componentes

**DEPENDENCY INJECTION**

é quando um objeto ou função recebe outros objetos ou funções dos quais ele depende para funcionar corretamente.

Um exemplo básico: uma classe Carro e uma classe Motor. Para o Carro funcionar precisa de um Motor, para isso no construtor nós iríamos fazer algo como: *this.Motor = new Motor().* Com dependency Injection, poderíamos receber esse Motor como parâmetro, “injetando-o” no Carro, desta forma, até poderíamos ter mais de um tipo de motor, sendo algo abstrato.  
*public Carro(iMotor motor) { this.Motor = motor*}

Em Vue.js, o module export também é uma forma de injeção de dependência. Os plugins, citados acima, também são uma forma de injeção de dependência, mas num nível mais de framework ou aplicação.

**Slots**

Slots são espaços “deixados” para serem preenchidos pelo conteúdo de outro componente. Exemplo:

Se criarmos um componente chamado “MyButton”, como qualquer componente ele tem suas props e data. Mas aqui, o mais importante está na estrutura HTML do componente, que terá um <slot>  
Quando usarmos este componente, podemos colocar dentro dele um conteudo, que será então renderizado dentro deste slot, mais ou menos assim:

Estrutura HTML do componente MyButton:  
<button>   
 <slot></slot>  
</button>

Quando utilizarmos o MyButton, faremos assim:  
<MyButton>   
 “Clique aqui!”  
</MyButton>

O texto “Clique Aqui!”, será inserido na tag <slot>.

É possível também, nomear slots, para que se possa “endereçar” onde o conteúdo deve ir, no exemplo abaixo, atente ao slot com name “header”

