



Vue.js

Prof. Rafael Menezes da Silva

Descrição

Abordagem da estrutura do framework Vue.js. Estudo de suas aplicações e principais características.

Propósito

O Vue.js é um framework JavaScript leve e de fácil aprendizagem, sendo adotado em larga escala no mercado mundial. Ele possui uma excelente documentação e permite um alto ganho de produtividade. O framework também é versátil, se tornando poderoso em aplicações de pequeno, médio e grande porte.

Preparação

Instale uma IDE (recomendamos o Visual Studio Code) ou um editor de texto, como, por exemplo, Sublime Text ou Atom. Softwares gratuitos, todos eles estão disponíveis para Windows, Linux e Mac OS.

Para realizar as atividades propostas nos vídeos, clique [aqui](#) para baixar os arquivos.

Objetivos

Módulo 1

Estrutura básica do Vue.js

Reconhecer a estrutura básica do Vue.js.

Módulo 2

Saindo do básico

Empregar uma lógica para interação com o usuário.

Módulo 3

Componentes

Aplicar a utilização de componentes para modularização do código.

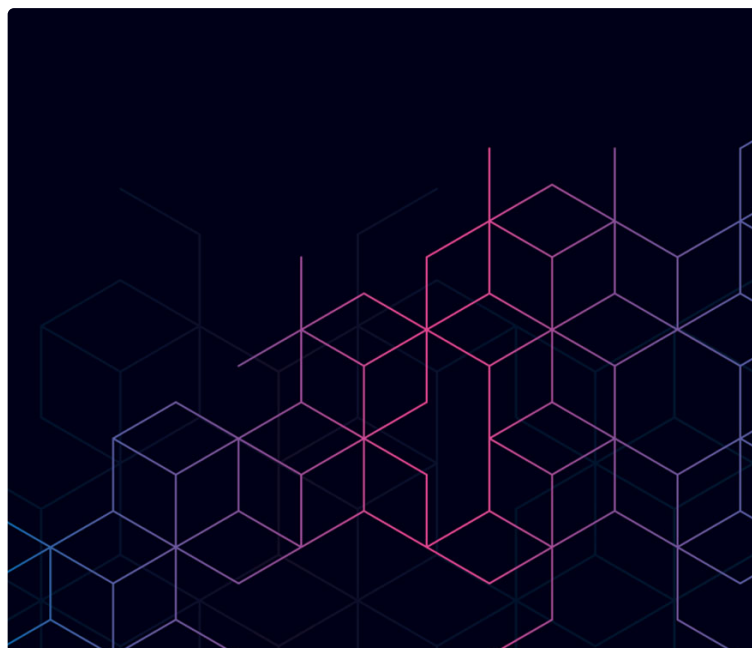
Introdução

A Internet atualmente move a maior parte dos serviços e da economia mundial. É impossível não perceber o crescimento de serviços públicos e privados realizados no modo on-line. O serviço de pagamentos digitais Pix é um exemplo marcante disso.

Ao passo que a procura pelo desenvolvimento desse tipo de aplicação cresce, o número de ferramentas também aumenta. Em um mercado tão sedento por profissionais de desenvolvimento de aplicações e com prazos cada vez mais curtos, o profissional capaz de atender aos requisitos de seus clientes em um tempo hábil é muito valioso.

O Vue.js se destaca como um framework da linguagem JavaScript (JS) bastante adotado no mercado mundial. Tendo uma boa documentação, ele permite um ganho de produtividade considerável se comparado à utilização de JS puro.

Framework leve e, ao mesmo tempo, escalável, o Vue.js se baseia em componentes cuja finalidade principal é criar interfaces de usuários ricas em comportamento com um código legível e de fácil manutenção, dando ao desenvolvedor que resolve adotá-lo o poder de criar muito em pouco tempo. Neste conteúdo, utilizaremos o Vue na versão 3 (definida por seus mantenedores como a versão principal em fevereiro de 2022).



1 - Estrutura básica do Vue.js

Ao final deste módulo, você será capaz de reconhecer a estrutura básica do Vue.js.

O que é Vue.js e por que usá-lo?

Antes de iniciarmos os estudos sobre o framework, vamos entender algumas características que fazem o Vue.js se destacar no mercado, sendo utilizado por aplicações por grandes empresas, start-ups e desenvolvedores autônomos no mundo todo.

Motivação

O Vue.js é uma ferramenta:

Open-source



Por ser mantido pela comunidade, o Vue.js não está associado a nenhuma Big Tech, o que confere mais segurança de que sua licença será permanente.

Reativa



Dado que a interface do usuário (HTML/CSS) reage às mudanças que acontecem “por baixo dos panos”.

Leve



Visto que ocupa menos de 100Kb na versão minificada para produção.

Flexível



Já que atende a projetos de pequenos a grandes, utilizando ECMAScript ou Typescript como um bloco único ou dividido em componentes.

Progressiva



Uma vez que começa pequeno e cresce (por meio do próprio código ou da inserção de plugins) conforme a sua necessidade.

Extensível



Pois o Vue conta com diversos mecanismos para estender suas funcionalidades ou facilitar o trabalho do desenvolvedor. Um exemplo é o plugin Vue Dev Tools (para Firefox e Google

Chrome), que permite um feedback em tempo real da aplicação, inclusive com os valores que cada componente possui no momento.

Exemplo prático

Se você ainda não ficou convencido(a) sobre a capacidade do Vue, considere o mesmo projeto de um contador simples implementado de duas maneiras diferentes:

Sem Vue

Contador é implementado com **JS puro**: <https://bit.ly/3upvLYZ>.

Com Vue

Contador é implementado com Vue.js: <https://bit.ly/35Sfbra>.

Não se preocupe em entender a sintaxe nesse primeiro momento.

Curiosidade

Perceba como se escreve menos com o Vue (produtividade) e como há uma melhoria considerável na legibilidade do código (melhor manutenção). Além disso, como o Vue não “suja” o HTML, o designer e o desenvolvedor podem trabalhar em conjunto sem se preocuparem em atrapalhar as tarefas um do outro.

O que veremos

Em primeiro lugar, entenderemos como o Vue resolve o problema de uma interface de usuário reativa, ou seja, uma parte da tela da aplicação Web monitorada pelo Vue, de modo que, quando os dados mudarem, a tela se atualizará automaticamente.

Analise a imagem a seguir:

Exemplo ilustrativo da janela de um navegador.

Sobre a arquitetura do Vue, enquanto o usuário vê a janela do navegador, “por baixo dos panos”, no “mundo Vue” existe uma variante chamada **nome** com o valor **João da Silv**. Enquanto o usuário digita, essa variável é atualizada “automaticamente”. Assim que ele digitar a última letra “a” do seu sobrenome, a variável receberá esse mesmo valor.

Incrível, não? Parece difícil implementar? Você será capaz de criar algo assim em questão de segundos! Como se sabe, quem entrega valor por

meio de aplicações Web em pouco tempo é um profissional de valor para o mercado.

Comentário

Em nossas atividades práticas, sempre trabalharemos com uma pequena história: nela, receberemos do designer um ou mais arquivos do template (você deverá fazer o download) e um conjunto de funcionalidades solicitado pelo cliente fictício.

Projeto – Carteirinha de Estudante

O que foi solicitado

A Universidade XPTO solicitou o desenvolvimento de uma página Web na qual os alunos possam se cadastrar para receber sua carteirinha de estudante. Ao digitar seus dados, o aluno poderá ver uma prévia da carteirinha (conforme imagem a seguir).

O designer da equipe já fez a parte dele. É nossa missão agora dar vida a esse template.

Resultado esperado pelo cliente (Universidade XPTO).

Conhecendo a estrutura do projeto

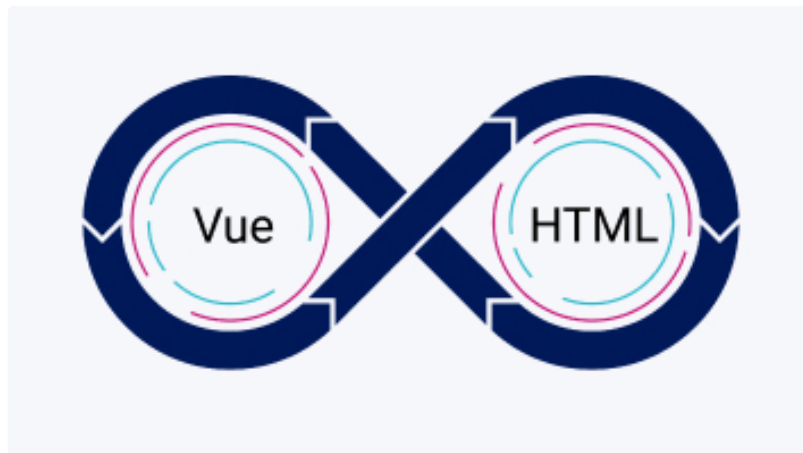
Abra o arquivo index.html referente ao módulo que estamos estudando.

Dica

Como IDE, sugere-se o uso do Visual Studio Code. Ele é gratuito e funciona no Linux, Windows e Mac OS. Caso decida utilizá-lo, instale também a extensão Vetur para obter uma melhor experiência no desenvolvimento. Se decidir usar outra IDE ou um simples editor de texto, tudo funcionará normalmente, mas você poderá perder produtividade. O Visual Studio com a extensão Vetur oferece facilidades que nos ajudam a digitar mais rápido, indentar o código corretamente e identificar erros com facilidade.

Verifique que o arquivo index.html não possui uma estrutura HTML muito grande e que, em seu cabeçalho, o documento HTML faz uma chamada externa para um arquivo de estilo (styles.css).

Ao percorrer as linhas, você verá uma div com id central. Nela, existem 2 div internas (formulário e prévia). Essas divs serão o nosso foco para a parte teórica e a prática deste módulo. O designer já indicou o local correto onde os dados devem aparecer, o que facilita muito o nosso trabalho.



Dados vão do Vue para o HTML e vice-versa.

Como falamos anteriormente, o Vue.js tem funcionalidades que permitem fazer a ligação de uma variável no mundo Vue com elementos da interface do usuário. Vamos nos valer dessa importante característica. No Vue, essa ligação é chamada de **binding** (ou, mais especificamente, **two-way data binding**, tendo em vista que ela ocorre nos dois sentidos).

Utilizando esse binding, faremos com que nossa interface reaja à inserção de caracteres nos campos de entrada (inputs) à esquerda, mostrando o que o usuário digita na prévia da carteirinha de estudante (quadrado avermelhado).

Entregando valor ao cliente

Seguiremos os seguintes passos para dar vida à aplicação com o Vue:



Passo 1

Importar o Vue

Consiste em trazer o arquivo-base do framework para nossa aplicação.



Passo 2

Criar uma instância do Vue

Resume-se a definir um objeto concreto, o qual utilizaremos para configurar nossa aplicação, lançando mão das funcionalidades do framework e guardando os dados gerenciados.



Passo 3

Montar a instância no HTML

Consiste em definir a parte do nosso HTML da qual o Vue terá controle.



Passo 4

Fazer as ligações necessárias

Baseia-se em apontar os dados pertencentes ao Vue e ligá-los em nosso HTML.

Importar o Vue

Primeiramente, façamos o básico! Vamos empoderar a nossa página, adicionando o poder do Vue. Para isso, precisamos importar o Vue.js.

A maneira mais simples de fazer isso é por meio de uma CDN (Content delivery network), que, em termos práticos, é um serviço Web que nos fornece arquivos com boa performance e confiabilidade. Logo antes do fechamento da tag body, adicione a seguinte linha: `<script src="https://unpkg.com/vue@3.0.0-rc.5/dist/vue.global.js" />`.

A imagem adiante mostra o posicionamento correto da tag script (entre o fechamento da última div e o do body):

```
JAVASCRIPT
</div>
</div>
<script src="https://unpkg.com/vue@3.0.0-rc.5/dist/vue.global.js"></script>
</body>
</html>
```

Posicionamento correto do código para importar o Vue.

Pronto. O Vue já está disponível, mas isso não significa que nossa interface seja reativa. Precisamos criar um objeto concreto do Vue, também chamado de instância (Vue Instance).

É nessa instância que definiremos em que parte do nosso HTML o Vue vai trabalhar e quais dados ele precisa monitorar, atualizar, gerenciar etc. Após isso, estaremos aptos a interagir com os dados e a apresentação deles no HTML, ou seja, na tela do usuário.

Criar uma instância do Vue

Para a criação de um Vue Instance, devemos criar mais um bloco de tag script abaixo do já existente (que importa o Vue da CDN), inserindo o bloco a seguir:

```
Javascript
Vue.createApp({
  // ...
})
```

O resultado será como o desta imagem:

```
JAVASCRIPT
</script>
Vue.createApp({
  // ...
})
</script>
```

Posicionamento correto do código citado para criar um Vue Instance.

Agora já temos nossas ferramentas à mão, podendo importar o Vue e declarar uma nova instância.

Montar a instância no HTML

O terceiro passo é bem simples. Devemos apontar um bloco no HTML ao qual o Vue terá acesso e controle. Em nosso caso, escolheremos a tag `#central`, pois ela engloba tanto o `#formulario` quanto a `#previa`, que são o foco da aplicação que queremos desenvolver.

Posicionamento correto do código para montar a instância no HTML.

Tendo controle sobre a div central como um todo, podemos monitorar o que é escrito no formulario e replicar as informações na previa, pois ambos são “filhos” da div com id central. Para fazermos o apontamento, diremos que a nosso Vue Instance deve ser montada na div com id central.

Basta, para isso, chamar o método `mount`, passando o seletor CSS de id (`#central`), conforme demonstra a imagem a seguir.

Posicionamento correto do código para fazer o apontamento.

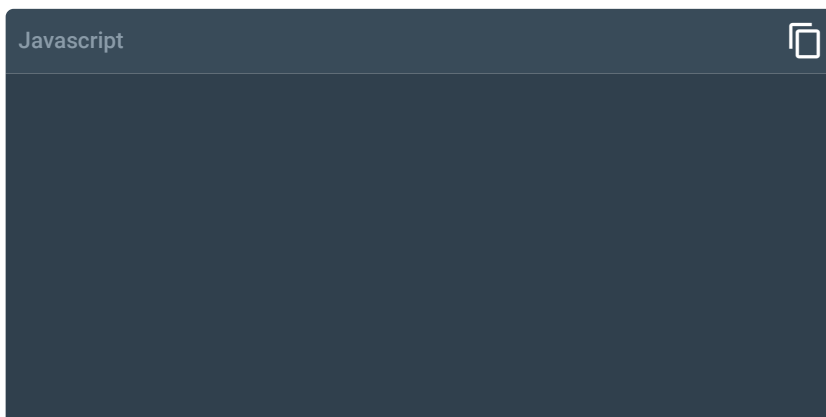
Uma leitura desse código seria: “crie um App (Vue Instance) e monte-o no elemento HTML, cujo id é central”.

Fazer as ligações necessárias

Expor os dados

Para o quarto passo, devemos primeiramente dizer ao Vue quais dados (data, em inglês) ele deve gerenciar e disponibilizar – no nosso caso, nome e matricula. No Vue, o atributo `data` precisa ser uma função que retorna um objeto, traduzindo em código.

Dentro das chaves (objeto JSON) do método `createApp`, devemos inserir o seguinte:



Como nome e matrícula constituem textos (string) e queremos mostrá-los inicialmente como um texto em branco, eles são inicializados como uma string vazia, abrindo e fechando aspas. Após essa inserção, o resultado será como o da imagem a seguir:

Posicionamento correto do código para expor os dados a gerenciar.

Esta imagem faz uma recapitulação dos passos na prática:

```
JAVASCRIPT

<!-- Importamos o Vue, da CDN unpkg.com -->
<script src="https://unpkg.com/vue@3.0.0-rc.5/dist/vue.global.js"></script>

<!-- Abrimos um bloco de script local para trabalharmos com o Vue.js -->
<script>
  //Criamos uma Vue Instance (um App)
  Vue.createApp({
    //Dizemos quais dados esse App irá gerenciar
    // data é uma função, () { .... }
    data() {
      // que retorna Objeto (JSON)
      return {
        //nome e matrícula inicialmente são Strings (textos) em branco ""
        nome: "",
        matrícula: "",
      },
    },
  }).mount("#central")
</script>
```

Recapitulação do posicionamento correto dos códigos.

Leia o código acima com os respectivos comentários. Isso vai ajudá-lo a entender o que fizemos até o momento. Essa base é muito importante para todo o seu aprendizado.

O Vue está pronto para enviar dados para o HTML. Porém, para finalizarmos as ligações, precisamos sair do mundo Vue e migrar para o HTML.

Elementos correspondentes ao nome e à matrícula estarão presentes em dois locais: no formulário e na prévia da carteirinha. No entanto, há uma diferença sutil.

Formulário

Os dados saem do elemento HTML para a nosso Vue Instance, ou seja, o que eu digito no meu input HTML vai para a variável nome constante nos dados do Vue.

Na prévia da carteirinha

Por outro lado, eles saem do Vue para o HTML, isto é, minha variável nome (dentro da função data) é representada no meu HTML.

Ligações de entrada de dados

O Vue utiliza um mecanismo chamado **diretivas** para criar uma ligação entre o HTML e o mundo Vue. Elas são aplicadas como atributos do elemento HTML.

Você já conhece os atributos mais comuns, como `id`, `class`, `placeholder`, `type` e outros. Eles são diferentes para cada tag HTML. Assim também são as diretivas. Veremos diversas delas ao longo deste conteúdo.

Para ligarmos nossos dados em um elemento HTML, usaremos a **diretiva** `v-model`. Essa diretiva cria uma ligação de duas vias (two-way data binding) para que, quando o valor do elemento muda, a variável a ele ligada também seja modificada.

Analise a aplicação dessa diretiva na imagem a seguir:

Posicionamento correto do código de utilização de diretiva (`v-model="nome"` e `v-model="matricula"`).

Dizemos ao Vue que nosso **primeiro** input representa o dado `nome`; o **segundo**, o dado `matricula`. Assim, ao atualizarmos o valor do input, isto é, ao digitarmos algum texto nele, a variável existente em nosso Vue Instance será atualizada.

Ligações de saída (impressão) dos dados

O Vue está expondo os dados que deseja por meio da função `data` (que retorna um objeto a ser exposto). Definimos de onde ele receberá as informações para popular esses dados (input de texto relacionado ao `nome` e à `matricula`), mas ainda não estamos vendo nenhuma saída (do mundo Vue para o HTML).

Precisamos dizer ao Vue onde ele tem de mostrar esses dados. Para isso vamos, usar bigodes. Aliás, bigodes duplos!

**Agora você deve estar perguntando:
bigodes? Ainda por cima duplos?**

Calma: é só uma maneira descontraída de dizer que o dado a ser impresso estará entre chaves – aliás, duplas de chaves. O termo correto é template **string**.

Exemplo

```
{{ dados-aqui }}
```

Repare que cada chave parece um bigode em pé. Desse modo, a comunidade de desenvolvimento (os *geeks*) frequentemente se refere a

esse tipo de representação como double mustache, ou seja, bigodes duplos.

Nosso amigo designer deixou uma pista para nós de onde deverão ser refletidas as variáveis **nome** e **matricula**; sendo assim, vamos usar os bigodes para mostrar essas variáveis em nosso HTML.

Nosso objetivo é substituir esse texto por uma indicação das variáveis do Vue mediante o uso das chaves (bigodes) duplas, criando, com isso, uma “janela” no nosso HTML onde tais informações serão mostradas. Observe as imagens a seguir que demonstram a substituição a ser realizada:

Posicionamento correto do código para criar a janela onde as informações serão mostradas.

Posicionamento correto do código para substituir as informações na janela.

Salve o arquivo index.html. Caso ele ainda não esteja aberto, abra-o no seu navegador favorito. Se já estiver aberto, atualize a página, digite algo no input de nome e matrícula e veja o resultado!

Nosso primeiro projeto está pronto. Chame todo mundo para vê-lo! Comemore as pequenas evoluções.

Refatorando: criando uma melhor experiência

Você reparou que o texto que nós escrevemos com as chaves duplas (bigodes) pisca na tela assim que atualizamos a página? Faça o teste. Atualize olhando para a prévia do cartão e observe.

Pode parecer algo trivial, porém, quando estivermos trabalhando com muitos dados, isso poderá ocasionar uma experiência ruim no nosso usuário final. Precisamos, portanto, refatorar nosso código.

A refatoração é um processo sistemático de melhoria em um código sem criar novas funcionalidades, somente melhorando o que já existe. Ela organiza o código ou reescreve as mesmas funcionalidades de uma maneira mais organizada e/ou corrigida.

Para resolvermos o problema dos elementos piscando, podemos seguir **duas** abordagens:

- Usando v-text em vez dos bigodes.
- Usando a diretiva v-cloak.

Para utilizar a primeira opção, basta remover a indicação entre chaves e usar a variável que se quer imprimir dentro de um atributo v-text do elemento HTML. O resultado pode ser visto a seguir:

Posicionamento correto do código citado para usar v-text em vez dos bigodes (antes).

```
JAVASCRIPT

<div id="dados-cartao">

  <span class="label-cartao">Nome:</span>
  <span class="dado-cartao" v-text="nome"></span>

  <span class="label-cartao">Matrícula:</span>
  <span class="dado-cartao" v-text="matricula"></span>
</div>
```

Posicionamento correto do código citado para usar v-text em vez dos bigodes (depois).

Antes de iniciarmos a explanação da segunda opção, gostaríamos de abordar uma convenção muito comum para os desenvolvedores que trabalham com o Vue.js: a de utilizar uma div com id app para ser o local de montagem da instância Vue.

Repare que utilizamos a div com id central. Não há problema nessa abordagem, mas é sempre indicado trabalhar com o que a comunidade costuma utilizar a fim de facilitar o trabalho de outros desenvolvedores e seguir o padrão da comunidade, uma vez que frequentemente precisamos pedir ajuda em fóruns e grupos.

O ideal é que a div app envolva o máximo de HTML sem comprometer o estilo da página. No nosso caso, a colocaremos logo abaixo da tag body, envolvendo a div com id wrapper.

```
JAVASCRIPT

<body>
  <div id="app">
    <div id="wrapper">
      <h1>XPTO - Carteira do Estudante</h1>
    </div>
  </div>
</body>
```

Posicionamento correto do código citado para utilizar uma div com id app.

Agora estamos prontos para usar a diretiva v-cloak, de acordo com o que indicamos como segunda opção, para evitar que os elementos pisquem na tela. Para isso, devemos colocar a diretiva v-cloak na div#app e, em nosso arquivo de estilo, indicar display:none para os elementos que possuam essa diretiva.

Posicionamento correto do código citado para usar a diretiva v-cloak (no arquivo index.html).

JAVASCRIPT

```
[v-cloak] {display: none}
```

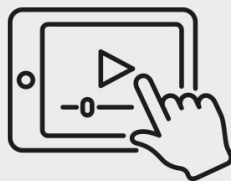
Posicionamento correto do código citado para usar a diretiva v-cloak (no arquivo styles.css).



Estrutura básica do Vue (implementando a carteira do estudante)

O vídeo a seguir apresenta a maneira correta de implementação do projeto e de navegação entre os arquivos, bem como cita boas práticas de desenvolvimento. Assista:

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Como vimos, o caminho mínimo para se trabalhar com o Vue é constituído de quatro passos sequenciais:

1. Importar o Vue;
2. Criar uma instância do Vue;
3. Montar a instância no HTML;
4. Fazer as ligações necessárias;

Qual seria a definição mais correta para a instância do Vue?

A

O arquivo-base do framework importado em nossa aplicação.

B

Um objeto que utilizamos para configurar nossa aplicação e gerenciar os dados.

C

A parte do nosso HTML da qual o Vue terá controle.

D

Somente os dados pertencentes ao Vue que estão ligados em nosso HTML.

E

Um objeto JSON retornado pela função `data()`.

Parabéns! A alternativa B está correta.

No Vue Instance, configuramos a nossa aplicação, além de definirmos os dados (expostos ou não) e os comportamentos da aplicação.

Questão 2

Qual das alternativas abaixo indica uma forma correta para se mostrar, dentro de uma tag `span` de nosso documento HTML, a variável `nome` constante na instância do Vue (mais precisamente, no objeto retornado pela função `data`)?

A

``

B

``

C

``

D

``

E

{{nome}}

Parabéns! A alternativa E está correta.

A alternativa E, que representa a variável nome entre chaves duplas, usa a técnica chamada de template string. Também conhecida como *double mustaches* (bigodes duplos), ela é utilizada para mostrar dados do Vue no documento HTML. Outra resposta válida – e que não consta nas alternativas da questão – seria esta: ``. O símbolo @ da alternativa A é utilizado para definir respostas a eventos. Por exemplo, @click define um comportamento do elemento HTML ao ser clicado. Já a diretiva v-cloak é utilizada para suprimir o fato de que alguns elementos piscam enquanto o Vue não é totalmente carregado. A diretiva v-model é empregada em elementos de entrada de dados (inputs). Por fim, a diretiva v-nome não existe.



2 - Saindo do básico

Ao final deste módulo, você será capaz de empregar uma lógica para interação com o usuário.

Projeto Todo.list

Requisitos do cliente

Apesar de termos percebido, desde o início deste estudo, o poder do Vue, ainda não exploramos todas as suas funcionalidades.

Por isso, neste momento, trabalharemos com listas dinâmicas e daremos um pouco mais de complexidade aos nossos projetos. Veremos ainda como trabalhar com condicionais, mostrando ou escondendo elementos na tela de acordo com alguma condição específica, por exemplo. Também exploraremos as classes CSS, além de eventos de mouse e teclado.

O pontapé inicial é a apresentação do projeto. Nosso aprendizado será construído na implementação das funcionalidades apresentadas pelo cliente.

Dessa vez, a empresa responsável por um aplicativo móvel de lista de afazeres denominado Todo.list nos solicitou a versão web do app. Sendo assim, o designer nos passou o template a seguir:



Resultado esperado pelo cliente Todo.list.

Além do template, ele nos passou os seguintes requisitos:

- Itens marcados como importante em vermelho (classe CSS: importante).
- Ao clicar no item, ele deve ser marcado como feito e ficar em branco e esmaecido (classe CSS: feito) independentemente de ser importante.

- O usuário digita o texto e clica em salvar ou aperta a tecla Enter, e o item vai para a lista. O botão salvar só deverá aparecer se houver um texto digitado. Após a inserção, os campos deverão ser limpos.
- O texto deve contar com até 20 caracteres, enquanto a contagem precisa ser mostrada a seguir do input de texto.

Setup inicial da aplicação

Primeiramente, é preciso importar o Vue por meio do link, criar uma nova instância do Vue e montá-la em nossa div de id app. Vamos fazer isso conforme aprendemos anteriormente, colocando os scripts entre o fechamento da última div e o da tag body.

Aproveitaremos para criar a função `data()` retornando um objeto, o qual vamos utilizar para dar vida ao nosso projeto. Caso tenha dúvidas sobre esse processo, analise o [código comentado](#).

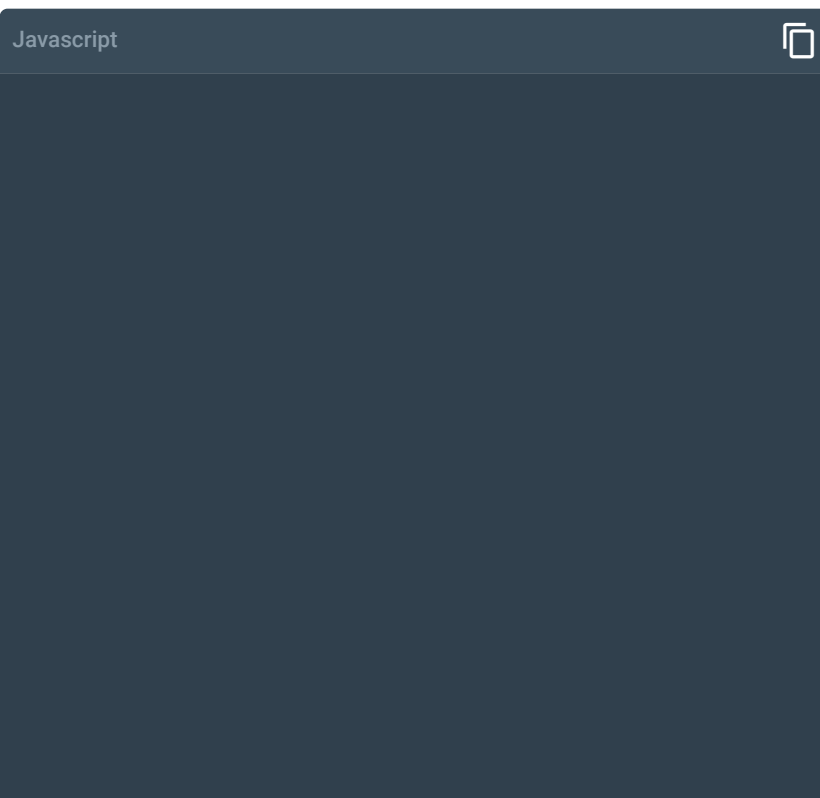
```
JAVASCRIPT

<!-- Importamos o Vue, da CDN unpkg.com -->
<script src="https://unpkg.com/vue@3.0.0-rc.5/dist/vue.global.js"></script>

<!-- Abrimos um bloco de script local para trabalharmos com o Vue.js -->
<script>
  //Criamos uma Vue Instance (um App)
  Vue.createApp({
    //Dizemos quais dados esse App irá gerenciar
    // data é uma função, () { .... }
    data() {
      // que retorna Objeto (JSON)
      return {
        //nome e matrícula inicialmente são Strings (textos) em branco ""
        nome: "",
        matrícula: "",
      }
    },
  })
  //Dizemos onde em nosso HTML o Vue irá viver, neste caso no bloco HTML com id central
  }).mount("#central")
</script>
```

Código comentado

Posicionamento correto do código para expor os dados a gerenciar.



Dentro do objeto retornado em data, vamos criar nossa lista de itens. Por enquanto, inicializaremos uma lista fictícia.

A lista será um **array** com três itens:

- Um item a fazer.
- Um item marcado como importante, mas não feito.
- Um item marcado como feito.

Array

Um array é um conjunto de dados definido por meio de colchetes.

Dentro dessa lista, cada item será um objeto do JavaScript com suas informações.

Renderizando a lista

Na imagem a seguir, vemos o escopo da lista de itens com os colchetes de abertura e fechamento em destaque. Essa lista é um elemento do objeto retornado pela função data, ou seja, é exposta e gerenciada pelo Vue.

```
JAVASCRIPT
return {
  lista: [
    {
      texto: "Item importante",
      importante: true,
      feito: false,
    },
    {
      texto: "Item importante",
      importante: true,
      feito: false,
    },
    {
      texto: "Item importante",
      importante: true,
      feito: false,
    }
  ]
}
```

Estrutura da lista de itens que será renderizada na tela.

A seguir, o que está em destaque é um dos objetos. Repare que todos têm a mesma estrutura. Da mesma forma, os itens possuem os mesmos atributos. Isso é recomendado para se saber o que esperar de cada objeto, mas não é algo obrigatório.

```
JAVASCRIPT
    },
    {
      texto: "Item importante",
      importante: true,
      feito: false,
    },
  ],
}
```

Destaque de um item a ser renderizado

Repare também nas chaves de abertura e fechamento de cada objeto: elas determinam o escopo de cada item. Os atributos de cada um deles são independentes.

O atributo texto é uma string. Já os atributos “importante” e “feito” são do tipo booleano, ou seja, eles aceitam true ou false (verdadeiro ou falso, respectivamente).

O próximo passo é remover as divs de itens fixos no HTML que o designer enviou de amostra. Utilizaremos o Vue para varrer nossa lista de objetos e mostrá-la item a item.

```
</div>
<div id="list-wrapper">
  <div class="item">
    <p class="texto-item">Item a Fazer</p>
  </div>
  <div class="item">
    <p class="texto-item">Item a Fazer</p>
  </div>
  <div class="item feito">
    <p class="texto-item">Item Feito</p>
  </div>
  <div class="item importante">
    <p class="texto-item">Item Importante</p>
  </div>
  <div class="item feito">
    <p class="texto-item">Item Feito</p>
  </div>
  <div class="item importante">
    <p class="texto-item">Item Importante</p>
  </div>
</div>
```

Destaque do local no documento HTML a abrigar a lista.

No código anterior, note a utilização das cores: amarelo para o escopo da lista e rosa para o escopo do item.

Os mundos Vue e HTML estão intimamente ligados. Esse é o objetivo.

Em nosso HTML, removeremos quase todos os itens. Vamos deixar uma div (um item) de exemplo para que possamos ter a estrutura de HTML desejada pelo designer, porém preenchendo-a com os dados do mundo Vue.

Local da lista no HTML já tendo removido os dados de amostra inseridos pela equipe de design.

A recomendação é que você sempre escreva, como foi feito nessa imagem, o comportamento que espera utilizando comentários. Isso guiará seus passos com o Vue.

Vamos numerar os comentários para facilitar o entendimento:

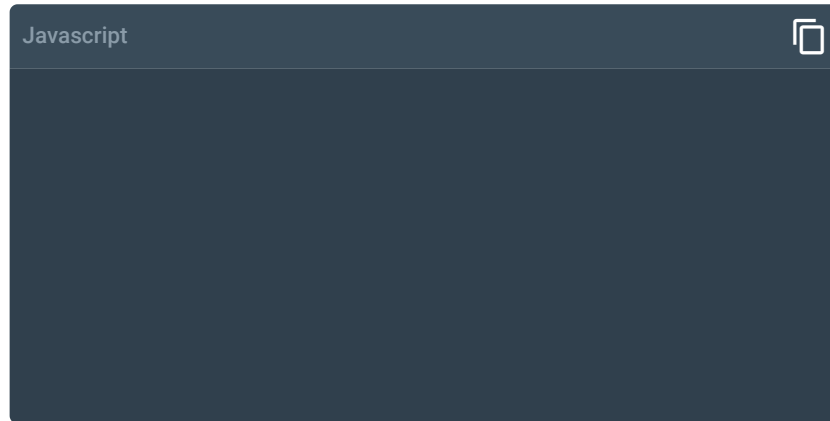
1. Para cada item da lista, é preciso ter uma div (como a div apresentada na imagem a seguir).
2. Essa div terá obrigatoriamente a classe item.

3. Dentro da div com classe item, haverá um parágrafo (tag p).

4. Dentro desse parágrafo, o texto do item será "impresso".

Agora vamos traduzir os comentários em código. O Vue possui uma diretiva chamada v-for que nos permite percorrer um array e, para cada item encontrado, repetir aquele bloco HTML.

A sintaxe do v-for para o nosso caso é a seguinte (código):



Veja como é possível ler a linha (frase):

"Insira uma **div** com a **classe** item **para** cada **item** na **lista** "

Dica

Releia o código e a frase algumas vezes até perceber a semelhança. Esse é o poder do Vue em ação! Com isso, já cumprimos os passos 1 e 2 dos nossos comentários.

Nossa próxima tarefa consiste em inserir uma tag de parágrafo (p) e imprimir o texto do item nela. Vamos utilizar a diretiva v-text em vez dos bigodes.

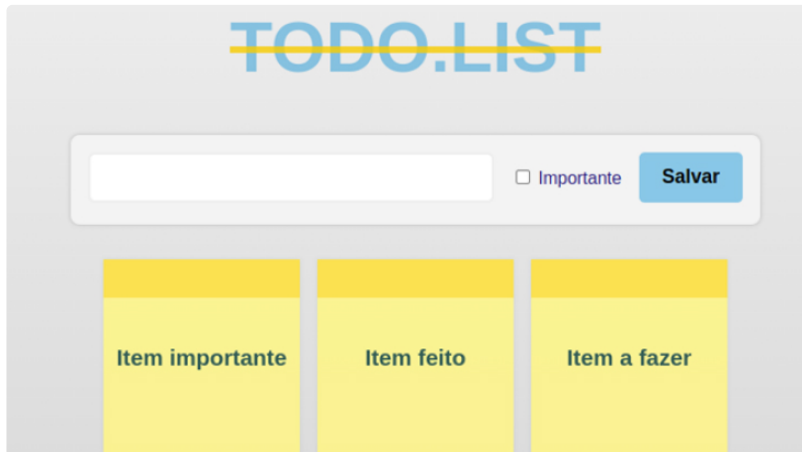
Revisando os objetos do array de itens na f9, veremos que cada um deles possui um atributo texto. Para acessar o texto de cada item, devemos, dentro do bloco definido pelo **v-for**, utilizar **item.texto**. O resultado pode ser visto na imagem a seguir:

Código para renderizar a lista e os respectivos itens.

O item em questão está dentro do bloco definido pela diretiva v-for. Sendo assim, ele não é um item fixo. O bloco determinado por essa diretiva será repetido uma vez para cada item da lista. A cada repetição (iteração), a variável item assume o valor de um dos objetos até chegar ao final da lista.

O código entre a segunda e a terceira div será repetido pelo Vue para cada item em nossa lista.

Com isso, teremos o seguinte resultado:



Resultado do código até o momento.

Perceba que já possuímos algo concreto: a lista está sendo percorrida, enquanto cada item é mostrado como um lembrete em amarelo. Estamos tendo progresso!

Classes e eventos

Classes CSS dinâmicas

Nossos itens estão aparecendo no HTML, mas todos eles estão iguais. De acordo com o que foi solicitado pelo cliente, eles têm cores diferentes. As cores são definidas por classes CSS.

Itens importantes (atributo importante marcado como true) recebem a classe importante. Aqueles já realizados (atributo feito marcado como true) recebem a classe feito.

Mãos à obra!

Para aplicar uma classe CSS quando uma condição for verdadeira, o Vue possuirá a diretiva v-bind. Observe como aplicá-la:

Aplicação da diretiva v-bind para a classe importante.

Em português, a palavra bind significa ligar, conectar. Por isso, a leitura da parte em destaque do código mostrado na imagem deve ser: “Ligue a classe importante caso o item seja importante”.

Falta agora a classe feito. Para isso, no mesmo bloco, vamos inserir mais uma informação. Observe o resultado:

```
JAVASCRIPT
<div id="list-wrapper">
  <div class="item" v-bind:class="{importante : item.importante, feito: item.feito}" v-for="item
  in lista">
    <p class="texto-item" v-text="item.texto"></p>
  </div>
</div>
```

Aplicação da diretiva v-bind para a classe feito.

A leitura do bloco definido pela diretiva v-bind deve ser a seguinte:

Atribua a classe importante caso o atributo importante do item atual seja true (verdadeiro). Atribua a classe feito caso o atributo feito seja true (verdadeiro).

Após termos feito isso, o nosso resultado será o seguinte:

Resultado após a aplicação das classes dinâmicas.

A diretiva v-bind tem uma forma reduzida definida simplesmente pelo sinal de dois pontos. Confira um exemplo:

Forma longa

```

```

Forma reduzida

```

```

O uso da diretiva v-bind para atribuir o endereço de imagens é muito comum. Diversos exemplos na Internet usam esse cenário para explicar a diretiva v-bind.

Apesar disso, ela pode ser utilizada para a atribuição de qualquer atributo HTML dinamicamente, recebendo dados do mundo Vue. Utilizamos o atributo de classe, pois ele se enquadra melhor em nosso exemplo.

Respondendo ao clique

Próximo requisito: quando o item for clicado, ele deverá ser marcado ou desmarcado como feito. Comumente, chamamos de toggle o ato de alternar o valor (estado) de um atributo entre dois estados predefinidos (ativo/inativo, ligado/desligado, true/false etc.). Seguiremos essa nomenclatura para o trabalho com o atributo (com a variável) feito existente em cada um dos itens da lista.

A ideia é que, ao receber um clique, o item seja marcado como feito caso não esteja. Caso esteja, ele será desmarcado. Esse comportamento é definido com os valores true ou false na variável feito daquele item.

Curiosidade

O Vue possui a diretiva v-on para lidar com eventos. Essa diretiva aceita como valor diversos tipos de eventos. Para o nosso caso, queremos que alguma ação aconteça quando o elemento for clicado. Sendo assim, trabalharemos com o evento click.

Observe como se atribui um evento de clique de mouse em nossa div representando um item:

```
JAVASCRIPT
<div id="list-wrapper">
  <div class="item"
    v-bind:class="{importante : item.importante, feito: item.feito}"
    v-for="item in lista"
    v-on:click="item.feito = !item.feito"
  >
    <p class="texto-item" v-text="item.texto"></p>
  </div>
</div>
```

Aplicação da diretiva v-on.

Pode parecer complexo, mas o que a linha `v-on:click="item.feito = !item.feito"` nos diz é: atribua à **variável feito** no item em questão (`item.feito` =) o **valor contrário** ao que ela já possui (`!item.feito`).

O sinal de exclamação inverte o valor de uma variável booleana (true vira false, false vira true). Em resumo, se `item.feito` for true, o contrário dele (`!item.feito`) será false. Com isso, ocorre o efeito toggle sobre o qual falamos anteriormente.

Salve o arquivo e veja como já temos o resultado desejado, alternando o estado dos nossos itens como feito ou não feito. Perceba ainda que você pode pular uma linha dentro do HTML para facilitar a legibilidade do código sem perder a funcionalidade.

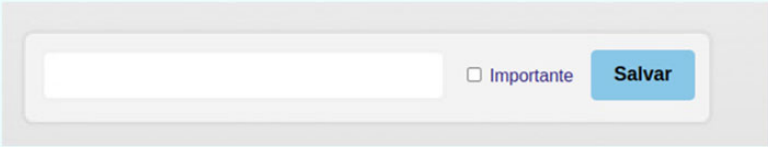
Adicionando elementos

Mais um evento de clique

Nossa lista já mostra os eventos, cada um com texto, cores e situações (importante e feito) sendo mostrados. No entanto, ela é estática, ou seja, é definida pelo(a) programador(a), isto é, por você, antes de a aplicação ser aberta – e isso não muda.

O solicitado pelo nosso cliente é que o usuário possa criar sua lista de acordo com suas necessidades. Para isso, devemos receber do usuário as informações do novo item a ser adicionado.

Repare, na imagem a seguir, que o designer já deixou pronto um formulário para receber os dados. Há nele um input para texto, um checkbox para definir se o item é importante e um botão de salvar.

A imagem mostra um formulário de adição de novo item, contendo um campo de texto, um checkbox rotulado 'Importante' e um botão 'Salvar'.

Estrutura do formulário com input para texto, checkbox para marcar se o item é importante e botão salvar.

Para que o Vue monitore esses itens, devemos trazê-los para o mundo Vue. No caso dos inputs (text e checkbox), precisamos ligá-los a variáveis no objeto retornado pela função `data()`. Isso é feito por meio da diretiva `v-model` (assunto abordado no primeiro módulo).

No caso do botão salvar, temos de atribuir um comportamento ao evento de clique utilizando `v-on:click` ou sua forma simplificada: `@click`. Os dois passos citados são mostrados nas imagens a seguir:

Observe que, na primeira imagem, a diretiva `v-model` (destacada em rosa) é utilizada tanto no campo do tipo texto quanto no checkbox, porém existe uma diferença. Verifique-a a seguir:

```
<div id="o-form">
  <input type="text" id="text-novo-item"
  v-model="textoNovoItem">
  <div>
    <input type="checkbox" id="is.importante"
    v-model="novoItemImportante">
    <label for="is.importante" >Importante</label><br>
    <div>
      <button @click="salvarNovoItem">Salvar</button>
    </div>
  </div>
```

Aplicação da diretiva v-on.

Aplicação da diretiva v-on.

```
JAVASCRIPT
data() {
  return {
    textoNovoItem: "",
    novoItemImportante: false,
    lista: [
      {
```

Diretiva v-model nos campos texto e checkbox.

No campo do tipo texto, tal diretiva assume o valor digitado no campo como string. Por isso, ela está ligada à variável `textoNovoItem` na imagem anterior inicializada como uma string em branco.

Já no campo checkbox, só existem dois estados possíveis: **marcado** e **desmarcado**. Quando marcado, o valor associado a ele é `true`. Quando desmarcado, o valor é `false`.



Marcado

Quando marcado, o valor associado a ele é `true`.



Desmarcado

No entanto, quando desmarcado, o valor associado é `false`.

Esse apontamento garante o que observamos desde o projeto da carteira do estudante: uma ligação que funciona nos dois sentidos. Quando a variável é alterada, o valor do campo também muda. Quando o campo é modificado, a variável, no mundo Vue, reflete essa alteração.

Agora devemos, ao clicar no botão de salvar, capturar essas duas informações e inseri-las, como um novo item, em nossa lista. Observe as imagens sobre a aplicação da diretiva v-on. Essa foi a maneira que utilizamos para mudar o estado de um item.

Utilizamos a diretiva v-on:click e passamos para ela o comportamento esperado: `item.feito = !item.feito`. Do mesmo modo, utilizamos a forma simplificada de v-on, que é um `@`. As duas formas são equivalentes para o Vue: fique à vontade para escolher a que achar melhor.

Percebeu que a maneira de definir o comportamento após o clique mudou? Em vez de passarmos o comportamento esperado em forma de código JS a ser executado, passamos um nome (`salvarNovoItem`). Esse nome representa uma função que queremos chamar após o clique.

Veja a seguir onde e como se dá a criação do método:

```
JAVASCRIPT
Vue.createApp({
  methods: {
    salvarNovoItem() {
      let novoItem = {
        id: new Date().getTime(),
        texto: this.textoNovoItem,
        importante: this.novoItemImportante,
        feito: false,
      };
      this.lista.push(novoItem);
      this.textoNovoItem = "";
      this.novoItemImportante = false;
    },
  },
  data() {
    return {
```

Criação do método.

Repare que:

- **data** (no canto inferior) é uma função que retorna um atributo.
- **methods** (na parte superior) é um atributo que retorna uma ou mais funções.
- **salvarNovoItem** (abaixo de methods) é uma função.

Dica

Ora a chamamos de função; ora, de método. Saiba que isso é intencional! As duas formas estão corretas.

Métodos são funções atribuídas a um objeto – nesse caso, atribuídas a nosso Vue Instance. Essa maneira de atribuir métodos a eventos facilita a legibilidade do nosso HTML.

Observe como ficou enxuto e fácil de entender que, ao se clicar no botão, o método `salvarNovoItem` será chamado:

```
JAVASCRIPT
</div>
<button @click="salvarNovoItem">Salvar</button>
```

Atribuição do evento ao botão.

Além disso, a organização do nosso código é melhorada, uma vez que eu passo somente um nome na diretiva `v-on` (ou `@`) e escrevo o método no local mais indicado.

Ao isolarmos um comportamento esperado em um método, podemos invocá-lo em outros locais do nosso código (veremos um exemplo disso adiante). Além disso, reutilizar um código é uma boa prática, pois, caso precisemos mudar um comportamento, teremos de alterar somente em um local.

Você percebeu o `this` antes dos nomes atributos?

Utilizamos essa palavra-chave para dizermos ao JavaScript que o atributo referenciado (`textoNovoItem`, `novosItemsImportante`, `lista` etc.) pertence ao objeto dentro do qual estamos trabalhando – nesse caso, à instância do `Vue`. Conforme imagem:

```
JAVASCRIPT
salvarNovoItem() {
```

Uso da palavra-chave `this`.

Sempre que formos pegar ou mudar uma variável (atributo) retornada por `data()` ou chamar, dentro do código, algum método do `Vue Instance`, teremos de obrigatoriamente usar o `this`.

Você percebeu que criamos mais um atributo para nosso objeto?

O atributo `id` recebe o valor retornado por `new Date().getTime()`. Essa fração de código retorna a quantidade de milissegundos desde 1º de janeiro de 1970, sendo utilizada para sempre gerar um número diferente, ou seja, um número único para cada item. Observe sua utilização:

JAVASCRIPT

```
let novoItem = {  
  id: new Date().getTime(),
```

Utilização do método que retorna os milissegundos.

Sua finalidade é evitar conflitos em itens que possuam os mesmos texto e estado (importante e feito). Para evitarmos esse conflito, devemos usar tal id para identificar o item ao varrermos a lista. Ele será utilizado como a chave (key) de cada item.

Você percebeu que limpamos os campos?

Um dos requisitos passados pelo cliente foi o seguinte: “O usuário digita o texto e clica em salvar ou aperta a tecla Enter e o item vai para a lista. O botão salvar só deverá aparecer se houver texto digitado. Após a inserção, os campos deverão ser limpos.”

Por isso, após adicionarmos o novo item à lista, voltaremos o valor do atributo textoNovoItem para uma string em branco e o valor de novoItemImportante para false. Isso vai gerar uma experiência melhor ao usuário, pois ele poderá, imediatamente após salvar, inserir outro item sem a necessidade de apagar o texto do anterior. Observe como:

Reset das variáveis.

Perceba, contudo, que ainda não cumprimos todo o previsto. Faltava salvar quando o usuário apertar a tecla Enter e esconder o botão caso o texto esteja em branco. Vamos adiante.

Respondendo também ao teclado

Uma das vantagens de isolar o comportamento de salvar em um método, conforme frisamos anteriormente, é poder reutilizá-lo. Na prática, não há diferença entre salvar clicando no botão de salvar ou em Enter.

Dessa forma, diremos ao Vue que também vamos querer chamar o método salvarNovoItem quando o usuário apertar a tecla Enter em nosso input texto.

Comentário

Creio que já esteja imaginando que existe um evento do Vue para isso. E você está correto(a)! O evento em questão é o keyup. Ele é ativado após o usuário pressionar e soltar o teclado. No entanto, se não definirmos a tecla (key), o Vue vai executar o comportamento em qualquer uma que seja digitada.

Precisamos, portanto, informar o desejo de executar o método somente após a tecla Enter. Para isso, colocamos um ponto após o evento e definimos a tecla.

O resultado pode ser visto a seguir:

Atribuição de evento no input de texto.

Elementos dinâmicos

Em alguns casos, não queremos que determinado elemento HTML apareça na tela até que algum estado ou condição esteja presente.

O Vue.js nos confere justamente esse nível de controle. Vejamos como isso funciona:

Escondendo o botão

Concentrando-se em atender plenamente aos requisitos do nosso cliente, teremos de esconder o botão de salvar quando o texto estiver em branco.

Para controlar a presença de elementos na tela, existem duas diretivas básicas:

- `v-if`
- `v-show`

A diferença entre ambas é que, com a diretiva `v-if`, o elemento é removido ou inserido no HTML conforme alguma condição (valor ou expressão que corresponda a `true` ou `false`). Se for `true`, o elemento será renderizado na tela; se ele for `false`, todo o código correspondente será removido do HTML.

Já a diretiva `v-show` simplesmente coloca a propriedade CSS `display` do elemento em questão como `none`. Assim, caso a condição retorne `false`, o elemento existirá no HTML, mas não será mostrado.

Em nosso exemplo, utilizaremos a diretiva `v-if`, tendo em vista que o bloco a ser inserido é pequeno e não exige muito processamento do browser para renderizá-lo novamente quando necessário.

Recomendação

Nos elementos em uma lista grande ou que sejam complexos, sugerimos a utilização do v-show. Experimente as duas formas para verificar o comportamento do Vue e do HTML (utilize as ferramentas do navegador para isso).

Para verificar se há texto digitado, testamos o tamanho da variável textoNovoItem, já que ela está ligada (two-way data binding) a nosso input de texto. Caso o tamanho (length) da variável seja zero, significa que o input está vazio, retornando false. Caso ele tenha ao menos uma letra digitada, é maior que zero.

```
JAVASCRIPT
<button
  @click="salvarNovoItem"
  v-if="textoNovoItem.length > 0"
>
  Salvar
</button>
```

Atribuição da diretiva v-if para esconder o botão.

Observe que a condição (textoNovoItem.length > 0) retornará true se houver algo digitado, mostrando o botão. Ponto para nós!

Mas ainda há um detalhe: o item também é adicionado à lista pelo teclado independentemente da existência do botão. Por isso, devemos editar o método **salvarNovoItem** a fim de que lá também seja testado o tamanho do texto, evitando que itens em branco sejam adicionados utilizando o evento do teclado.

```
JAVASCRIPT
salvarNovoItem(){
  if(this.textoNovoItem.length > 0){
    let novoItem = {
      texto: this.textoNovoItem,
      importante: this.novoItemImportante,
      feito: false,
    };
    this.lista.push(novoItem);
    this.textoNovoItem = "";
    this.importante = false;
  }
},
```

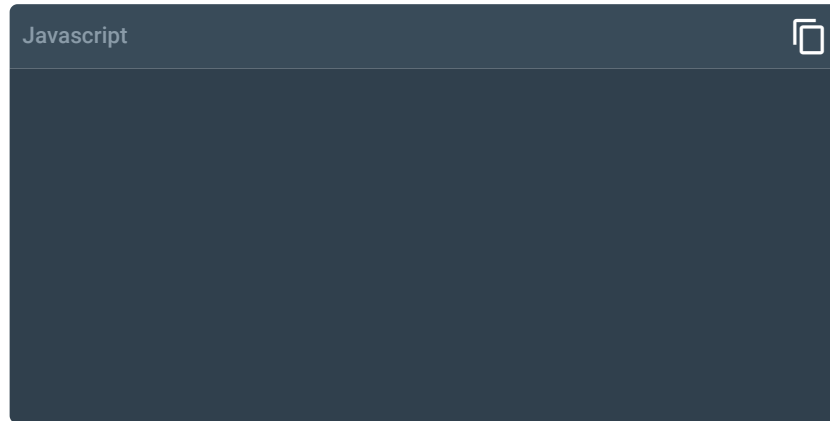
Aplicação de condição para salvar o item.

O **if** utilizado lembra muito aquele empregado na diretiva v-if do botão, mas devemos nos lembrar de utilizar o **this**, pois estamos referenciando um atributo do Vue Instance dentro de um código presente nela mesma.

Contando caracteres digitados

Agora é hora de atender ao último requisito do cliente: a contagem de caracteres. Como vimos, a quantidade daqueles digitados no input pode ser verificada por meio da propriedade length (tamanho) da variável textoNovoItem. Também sabemos que, para mostrar um valor de uma variável no nosso HTML, utilizamos a template string, colocando a variável ou expressão entre chaves.

Assim, para imprimir no HTML o tamanho do texto digitado, devemos utilizar o seguinte:



Existe uma maneira mais elegante, ou seja, com melhor legibilidade, para mostrar o mesmo resultado. Ela se dá utilizando as [computed properties](#).

Essas propriedades especiais são recomendadas para casos em que:

- Haja um processamento menos trivial com muito código ou um que exija um esforço para ser entendido.
- Exista a necessidade de reatividade (a propriedade muda e o HTML reage), caso ao qual os métodos (mesmo suportando lógicas nada triviais) não atendem: eles precisam ser chamados para que atualizem o HTML.

Computed properties

Em uma tradução literal, seriam as propriedades computadas (calculadas).

Veremos que uma computed property define um bloco de código como um método e a capacidade de ser reativa como uma variável. Trata-se, sem dúvida, de uma ferramenta útil e elegante!

Tenha sempre em mente que seu código será lido por terceiros (um amigo de trabalho, um entrevistador etc.). Assim, quanto mais clara a finalidade daquele bloco de código, melhor. As computed properties ajudam muito nesse sentido.

Para a criação de uma propriedade computada, deve-se primeiramente criar um objeto, chamado `computed`, dentro da nossa Vue Instance, que vai abrigar todas as computed properties. Veja o resultado a seguir:

Estrutura com a inclusão das propriedades computadas.

Perceba que o conteúdo de data e methods foi minimizado (destaque em vermelho). Pertencente ao VS Code, tal funcionalidade foi utilizada aqui somente para facilitar seu entendimento.

O objeto computed (destacado em amarelo) fica no mesmo nível (hierarquicamente falando) da função data() e do objeto methods. Não há uma ordem para esses elementos. Você precisa escolher qual deles vem primeiro e em segundo lugar – e assim por diante.

Comentário

Geralmente, data() é definida primeiramente, pois ela tem os principais dados do Vue Instance, mas isso é apenas uma convenção.

Veremos agora como é definida a propriedade computada referente ao tamanho do texto digitado:

```
JAVASCRIPT
    computed: {
      qtdeDigitada(){
        return this.textoNovoItem.length
      }
    }
  }
```

Propriedade para mostrar de maneira reativa a quantidade de letras digitadas.

Como já apontamos, uma propriedade computada tem a estrutura de um método e precisa ter um retorno. Além disso, ela será reativa, ou seja, quando o tamanho da variável textoNovoItem mudar, nossa propriedade computada notificará nosso HTML para mudar. Isso é fantástico!

Observemos a utilização dela em nosso HTML:

```
JAVASCRIPT
    </div>
    <p id="contagem">{{ qtdeDigitada }}</p>
  </div>
```

Imprimindo a propriedade computada no HTML.

O designer deixou um parágrafo, logo após o formulário, para que pudéssemos imprimir a quantidade digitada. É ali que chamaremos nossa computed property. Perceba que ela tem “cara” de variável. Isso nos dá uma boa legibilidade, e nosso código fica limpo.

Falta um último detalhe. Nosso campo de texto continua aceitando mais de 20 caracteres. Precisamos, portanto, adicionar a propriedade maxlength="20" em nosso input de texto. Com isso, o usuário não consegue digitar mais de 20 caracteres.

```
JAVASCRIPT
<input
  type="text"
  id="text-novo-item"
  v-model="textNovoItem"
  @keyup="this.$refs.textNovoItem.value.length < 100 ? '' : ''"/>
```

Propriedade que limita a quantidade de caracteres no input.

Pronto: atendemos às necessidades do nosso cliente! Você acha que esse projeto pode melhorar? Faça testes, melhore e mude. Desafie a si mesmo a criar funcionalidades novas. Isso com certeza vai melhorar seu desempenho.

Recomendação

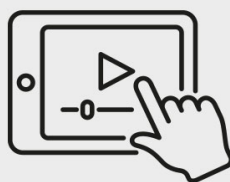
Por falar em desafio, volte e revise o projeto do contador. É provável que você consiga entender todo o código.



Implementando o sistema Todo.list

A seguir, o vídeo mostra a maneira correta de implementação do projeto Todo.list.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Este código para o método `salvarNovoItem` não se comporta como o esperado:

Qual das alternativas a seguir apresenta o problema no código?

A

Falta o uso do ponto e vírgula ao final de cada uma das linhas.

B

`novotem` deveria ser um array em vez de um objeto.

C Falta a declaração de parâmetros na função.

D O teste executado dentro do if deveria ser `textoNovoItem.length >= 0`.

E Falta o uso da palavra-chave `this`.

Parabéns! A alternativa E está correta.

Toda vez que utilizamos variáveis, métodos e computed properties, entre outros, dentro do escopo do Vue Instance, precisamos usar a palavra-chave `this`. Isso evita confusões com variáveis externas à instância. Exemplo: se houver outra variável `lista` fora do mundo Vue, só que no mundo JavaScript, poderá haver conflitos. O sinal de ponto e vírgula é opcional na linguagem Javascript. Tampouco existe a obrigatoriedade de `novotem` ser array. Nesse caso, como ele tem propriedades próprias, o mais indicado é ser um objeto. A função não necessita de parâmetros: ela busca dados do Vue Instance com o `this`. O teste não pode ser `>= 0`, pois queremos que tenha ao menos uma letra digitada. Quando o tamanho é igual a zero, significa que o campo está em branco.

Questão 2

Qual das opções a seguir cria uma div para cada elemento de uma lista chamada `list`, imprimindo o nome do usuário (variável `name`)?

A `<div for="user on list"> {{name}} </div>`

B `<div for="user in list"> {{name}} </div>`

C `<div for="list as user"> {{user.name}} </div>`

D `<div for="user in list"> {{user.name}} </div>`

E

```
<div for="user on list"> {{user.username}} </div>
```

Parabéns! A alternativa D está correta.

A alternativa D contém a sintaxe correta, criando uma variável temporária `user` para cada elemento em `list`. Além disso, imprime o atributo `name` dessa variável com a marcação ponto (`user.name`) dentro de uma template string (bigodes).



3 - Componentes

Ao final deste módulo, você será capaz de aplicar a utilização de componentes para modularização do código.

O que é um componente?

Componentes são blocos de código reutilizáveis usados para criar uma base de código modular (replicável) e de fácil manutenção (ponto único de mudanças). É possível imaginá-los como brinquedos de blocos de montar: juntando vários deles, obtêm-se construções incríveis.

Imagine agora uma rede social. Ela pode ter um componente para a foto de perfil; outro, para o feed de notícias; e um terceiro, para o serviço de mensagens. Juntos, todos formam uma página complexa. No entanto, eles são criados um a um.

Representação ilustrativa de componentes de uma rede social.

Sendo assim, podemos, em um único arquivo, representar uma parte do nosso sistema, englobando:



HTML

Sua estrutura.



JS

Seu comportamento.



CSS

Seu estilo.

Só que o poder dos componentes vai além da facilidade de estruturar nosso HTML.

Exemplo

Suponha que utilizemos um componente de botão de curtir e precisemos mudar a estrutura, a cor e/ou o comportamento desse botão. Utilizando componentes, não teremos de procurar pelo CSS específico dele em arquivos de CSS com centenas de linhas nem partir em uma caçada por todos os arquivos JS nos quais utilizamos suas funções e alterá-las, por exemplo. Temos um arquivo para aquele componente e podemos agrupar os códigos (HTML, CSS e JS) relativos ao botão nesse arquivo. Apesar dessa possibilidade, isso não significa que, ao reutilizar um componente, todas as suas instâncias devam ser exatamente iguais.

Observe a seguir uma demonstração de um componente:



Representação ilustrativa de um componente de aplicação de clima.

Ele representa o card de uma cidade pertencente a uma aplicação de clima. Nesse componente, é possível ver alguns elementos: o nome da cidade, uma descrição do clima atual, a temperatura atual e um ícone correspondente ao clima. Poderíamos, se desejássemos, reutilizar a mesma estrutura para outras cidades.

Nesse caso, teríamos de trocar as informações. Vejamos como o Vue nos ajuda nessa tarefa.

Para esse projeto, utilizaremos o Vite, uma ferramenta de construção de aplicações que ajuda bastante no desenvolvimento e que fornece muitos benefícios na hora de passar essa aplicação para produção. Ele foi criado por Evan You, que é o criador do próprio Vue. Apesar disso, ela pode ser usada com outros frameworks, como React.

Preparando o ambiente

Você acompanhou a instalação do Node e a utilização do npm para inicializar uma aplicação com o Vite. A partir desse passo, já estamos prontos para iniciar nossa jornada rumo aos componentes do Vue.js.

Observe adiante a divisão definida para o app. Cada bloco em destaque corresponde a um componente a ser criado:



Perceba que há três componentes na imagem:

- Na parte superior: o cabeçalho da aplicação.
- Na parte inferior: a lista das cidades.
- No centro: o card da cidade.

Não podemos esquecer que, em si, a página é um componente. Nesse caso, como o componente a representá-la agrupa todos os outros componentes, ele é chamado de componente raiz (root component).

Veremos outra forma de representar os componentes. Essa maneira foca a hierarquia entre eles:

Hierarquia dos componentes da aplicação de clima..

Tendo ciência de toda a estrutura, é hora de colocar a mão na massa. Vamos **componentizar** nossa aplicação.

Comentário

A palavra **componentizar** não existe no português, mas, no dia a dia de desenvolvedores, ela é comum; por isso, a trouxemos para o texto.

Começamos pelo componente mais simples, o cabeçalho, pois essa abordagem pretende justamente facilitar seu aprendizado.

Single file components

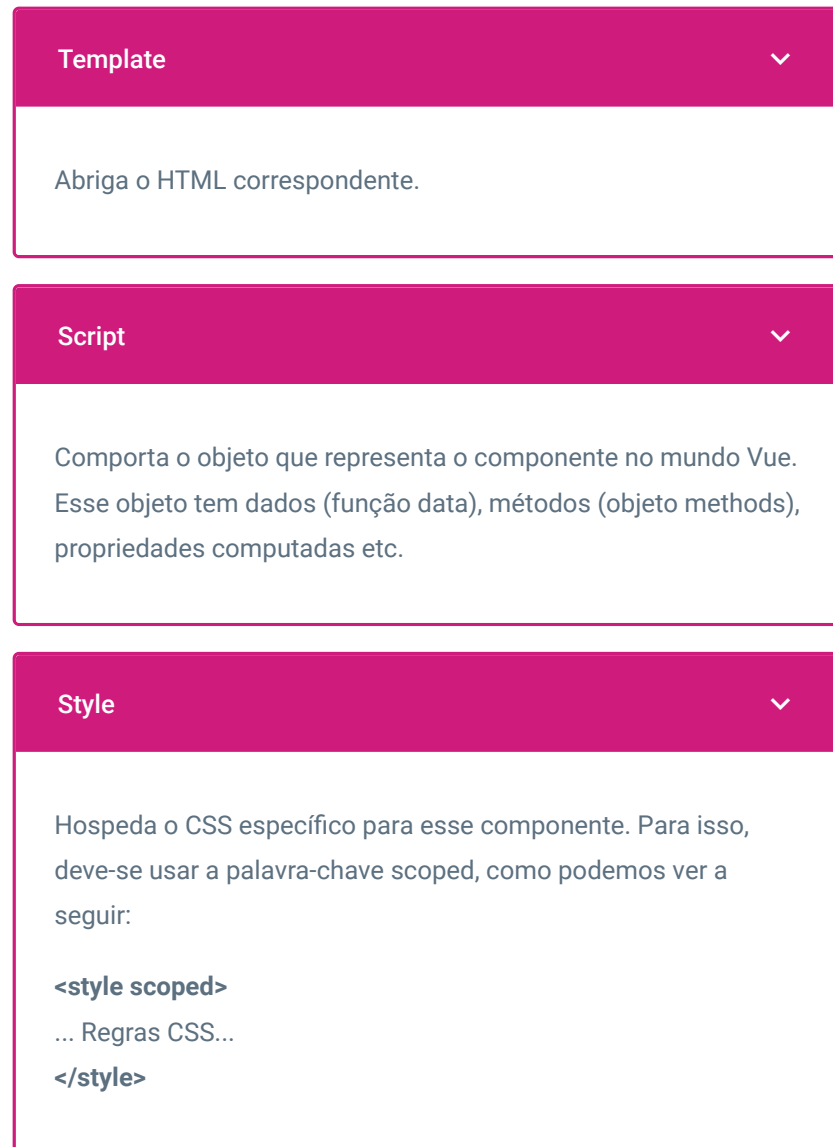
Uma das abordagens possíveis – e a mais recomendada – para a criação de componentes com o Vue é a utilização de um arquivo para cada componente. Esses componentes definidos em um arquivo único são chamados de single file components.

A extensão utilizada por esses arquivos é .vue. Sendo assim, antes de iniciar o desenvolvimento do componente, devemos criar o arquivo que o abrigará. Como destacamos, começaremos pelo cabeçalho.

Desse modo, vamos criar o arquivo Cabecalho.vue dentro da pasta src/components. Podemos excluir o componente HelloWorld.vue, criado por padrão pelo Vite, já que ele não será utilizado.

Estrutura básica do componente

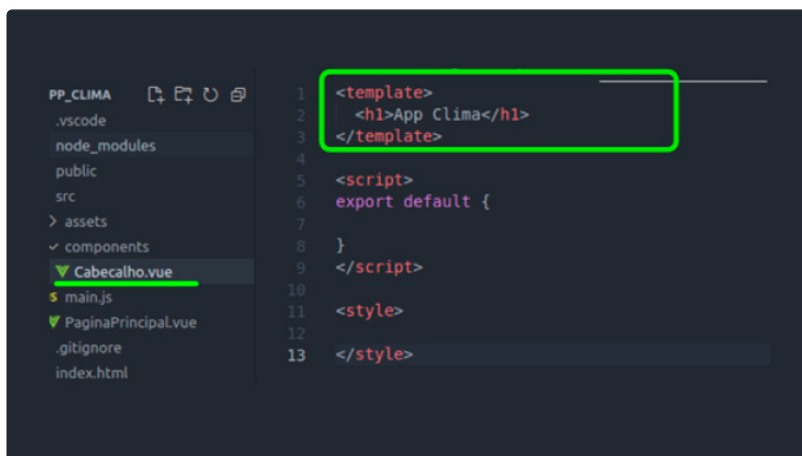
A estrutura básica (três blocos básicos) de um single file component é a seguinte:



Neste conteúdo, manteremos todo o estilo em um só arquivo CSS, ou seja, não utilizaremos a tag de estilo diretamente nos componentes filhos. O CSS será importado no componente raiz (PaginaPrincipal.vue), enquanto o estilo será aplicado a todos os outros.

Como o componente do cabeçalho não terá nenhum comportamento específico, a tag script desse arquivo ficará em branco. Para o componente do cabeçalho, é preciso somente trazer a estrutura HTML.

Tal decisão visa facilitar seu aprendizado, evitando muita complexidade nos primeiros exemplos. O foco é que, em um primeiro momento, você entenda a base de como criar e utilizar um componente. Observe como fica a estrutura do arquivo Cabecalho.vue:



Estrutura e posição do arquivo do componente de cabeçalho.

Do componente `PaginaPrincipal.vue`, trouxemos somente a tag `h1`, que representa nosso cabeçalho nesse primeiro exemplo. Agora já podemos utilizar nosso componente dentro daquele que representa nossa página principal.

Utilizando componentes

Para utilizar o componente recém-criado, é necessário seguir três passos:



Passo 1

Primeiramente, importar o arquivo correspondente no componente pai.



Passo 2

Em seguida, registrar o componente filho no pai.



Passo 3

Por fim, utilizar o componente como uma tag HTML.

Verifique na imagem adiante a implementação desses passos:



Utilização do componente cabeçalho dentro do componente da página principal.

Agora vamos **componentizar** a lista de cidades. Chamaremos esse componente de ListaCidades.vue.

A imagem a seguir mostra a estrutura desejável do componente após se criar o arquivo e trazer o HTML e o JavaScript correspondente:

Estrutura (HTML) e comportamento (JS) do componente lista de cidades.

O próximo passo é trazê-lo para o componente raiz (importar, registrar e utilizar):

Utilização do componente relativo à lista de cidade no componente raiz.

Perceba como a estrutura do nosso componente raiz (PaginaPrincipal.vue) ficou enxuta. Fica claro que a página principal tem um cabeçalho e uma lista de cidades.

A legibilidade aumenta, enquanto a manutenção fica mais fácil. Caso seja preciso mudar o cabeçalho da página, é preciso ir ao componente que o representa, assim como à lista de cidades.

Por fim, deve-se **componentizar** o componente do card que representa uma cidade:

Estrutura (HTML) do componente card de cidade.

Observe o template anterior. Ele contém a div que representa o card, mas como ele saberá que cidade mostrar?

Para fazer isso, ele precisa receber os dados da cidade a fim de que, só assim, eles consigam ocupar as posições corretas (nome, descrição e temperatura). Cada card deverá obter essas informações de seu componente pai, ou seja, da lista de cidades.

Passar informações de um componente para outro? Sim, isso mesmo: é possível!

Para isso, deve haver uma maneira de criar um canal de comunicação entre um componente pai e um componente filho. Lembre-se de que essa comunicação é unidirecional, ocorrendo somente do pai para o filho.

Passando informações para componentes filhos com props

A maneira que o Vue disponibiliza para passar informações de um componente pai para um componente filho é chamada de propriedades – mais comumente, de props. As props são atributos somente de leitura e podem ser de qualquer tipo aceito pelo JavaScript (string, inteiros, objetos, arrays etc.).

Em nosso caso, queremos passar o objeto representando uma cidade para dentro do card. Para isso, o card deverá estar pronto para receber esse objeto.

Como essa função não é responsabilidade do HTML, e sim do mundo Vue, indicamos no bloco script correspondente que o componente CardCidade.vue tem de estar pronto para receber do elemento pai (ListaCidades.vue) uma propriedade chamada cidade e que o elemento passado deve ser um objeto.

Confira, no final do código abaixo, em amarelo:

Criação de uma propriedade no componente card de cidade.

Perceba que removemos o v-for. Quem vai varrer o array de cidades será o componente pai (ListaCidades.vue), passando para o card somente as informações dele. O card fica isolado: recebendo apenas os dados, ele mostra as informações da cidade sob sua responsabilidade.

Isso atende ao princípio da responsabilidade única, que prevê que um trecho de código só deve ter um motivo para ser alterado. Nesse caso, somente se quiséssemos alterar a estrutura do card que representa a cidade ou o seu comportamento, voltaríamos a alterar o arquivo correspondente (MARTIN, 2009).

Agora podemos, em nossa lista de cidades, utilizar o componente CardCidade.vue para mostrar todas as cidades da lista novamente:



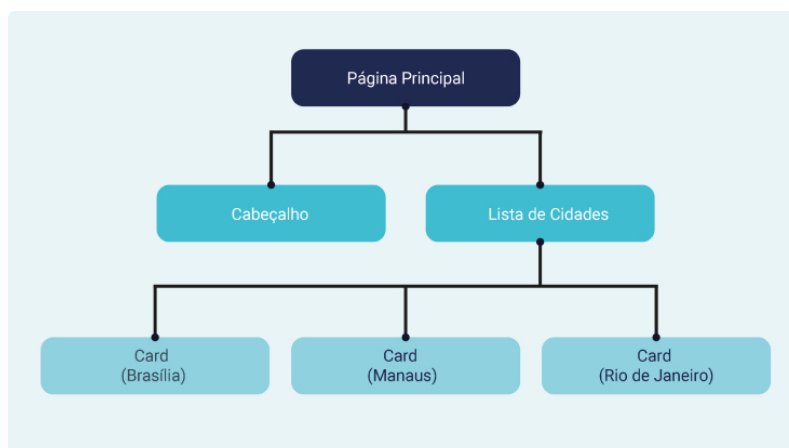
Utilização do componente card de cidade dentro do componente de lista de cidades.

A leitura do HTML deve ser a seguinte:

- Construa um card de cidade: **<CardCidade**.
- Para cada cidade no array cidades: **v-for="cidade in cidades**.
- Esse card deverá ser diferenciado (chave) dos outros pelo seu id: **:key="cidade.id"**.
- Passe para a propriedade cidade do card o objeto cidade atual: **:cidade="cidade"**.

Pronto. Nossa aplicação está toda dividida em componentes! Revise a imagem que mostra a [hierarquia entre componentes](#) e percorra a estrutura do código, começando do componente PaginaPrincipal.vue.

Veja como fica fácil entender a divisão e hierarquia dos blocos da aplicação.



Hierarquia dos componentes

Hierarquia dos componentes da aplicação de clima.

Passando bloco de HTML para componentes filhos com slots

Além de transferir valores, variáveis e objetos, entre outros itens, para nosso Vue Instance com as props, também podemos passar todo um bloco de HTML e até mesmo um componente com slots.

Uma analogia possível quando se trata de slots é pensar neles como uma “abertura”, uma “janela” ou um “recorte” que deixamos aberto em nosso componente, podendo o desenvolvedor colocar nesse espaço o que ele desejar.

Veja a ilustração a seguir:



Ilustração sobre slots em componentes do Vue.

Observe que deixamos um slot no componente pai e podemos colocar nele diversos elementos HTML ou até mesmo um componente Vue.

Vejamos agora um exemplo prático:

Criaremos uma “abertura” (slot) em nosso componente de cabeçalho (Cabeçalho.vue) para inserir nele um subtítulo. A imagem adiante mostra a criação do slot:

Criação de um slot não nomeado no componente de cabeçalho.

No componente que utiliza o cabeçalho (no caso, PaginaPrincipal.vue), podemos inserir um ou mais elementos HTML ou os componentes que desejamos. O bloco que inserirmos ocupará a posição marcada para o slot, ou seja, abaixo da tag h1.

Observe a maneira de se inserir o bloco entre a abertura e o fechamento da tag do nosso componente:

Utilização do slot não nomeado.

Vejamos agora o resultado:



Representação ilustrativa do resultado da utilização de slots.

Pode haver mais de um slot por componente, porém, se utilizarmos dois ou mais, teremos de nomear esses componentes para que o Vue saiba onde colocar cada bloco.

```
JAVASCRIPT

<template>

  <h1>App Clima</h1>

  <slot name="subtitulo"></slot>
  <slot name="extra"></slot>

</template>

<script>

export default {

}

</script>
```

Criação dos slots nomeados: Cabecalho.vue.

Utilização dos slots nomeados: PaginaPrincipal.vue.

Esses são alguns motivos da fama do Vue como ferramenta muito flexível e por qual motivo, como podemos ver na primeira página de sua documentação, ele é chamado de framework progressivo.

Passando informações para componentes-pai com emissão de eventos

Para fechar nosso conteúdo com chave de ouro, vamos aprender como se cria um canal de comunicação de um componente filho para os componentes pais. O Vue, afinal de contas, não fica limitado à passagem de informações para baixo, isto é, de pai para filho.

Existe também uma maneira de notificar o componente pai de que algo aconteceu no componente filho. A forma com que isso acontece é por

meio da emissão de eventos.

Vamos, portanto, criar uma maneira de a lista de cidades (componente ListaCidades.vue) saber qual das cidades foi selecionada (clificada). Para isso, o card (componente CardCidade.vue) que for clicado deverá avisar à lista (seu pai) que cidade ele abriga. Ao receber esse evento, a lista precisa alertar o nome da cidade clicada.

Primeiramente, informaremos ao componente card que ele emite determinado evento e que esse evento deve ser disparado ao receber um clique:

Definição de evento a ser emitido no componente filho após o clique.

Dessa maneira, o componente card já sabe que está emitindo a seu componente pai um evento chamado cidadeclificada e passando como parâmetro para esse evento a cidade que ele abriga. É preciso agora receber esse componente no pai.

Verifique a seguir como dizemos à lista de cidades que devemos esperar que seu filho (CardCidade) emita um evento chamado cidadeclificada e que, quando o filho emitir esse evento (v-on:cidadeclificada), ele deverá chamar o método mudarcidadeselecionada. Tal método, nesse caso, só alerta o atributo name da cidade, como pode ser visto nas últimas linhas da segunda ilustração. Perceba que primeira imagem apresenta a definição do comportamento ao receber um evento do componente filho com parâmetro e sem parâmetro:

JAVASCRIPT

```
<h2>Cidade Selecionada: {{ cidadeselecionada }}</h2>
<div class="lista">
  <CardCidade
    v-for="cidade in cidades"
    :key="cidade.id"
    :cidade="cidade"
    v-on:cidadeclificada="mudarcidadeselecionada"
  />
</div>
</template>
```

Definição do comportamento ao receber um evento do componente filho (method e listener), com parâmetro.

JAVASCRIPT

```
<script>
import CardCidade from "../CarCidade.vue";
export default {
  components: { CardCidade },
  methods: {
    mudarcidadeselecionada(cidade) {
      alert (cidade.name)
    }
  },
  data() {
```

Definição do comportamento ao receber um evento do componente filho (method e listener), sem parâmetro.

O resultado no HTML após o clique pode ser observado adiante:

Representação ilustrativa do resultado após o clique (emissão do evento).

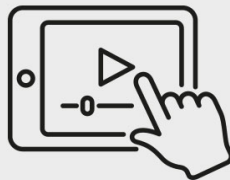
Conseguimos, enfim, uma conversa bidirecional entre o componente pai e o componente filho.



Iniciando um projeto com ferramentas de mercado

O vídeo a seguir apresenta uma demonstração da instalação de Node.js, npm e Vite, bem como a criação de um novo projeto com essas ferramentas.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Um desenvolvedor está trabalhando em um componente que mostra o resumo de um perfil de usuário. Nesse componente, são utilizados três outros: a foto do usuário, a bio dele (um texto resumido) e a lista de seguidores. Apesar de muito esforço, nada é mostrado na tela.

Observe a estrutura a seguir e responda à pergunta:

Do que o desenvolvedor se esqueceu?

- A** De criar o bloco de estilo.
- B** De criar métodos.
- C** Somente de importar os componentes filhos.
- D** De criar propriedades computadas.
- E** De importar e registrar os componentes filhos.

Parabéns! A alternativa E está correta.

O Vue precisa conhecer os componentes filhos a serem utilizados. Para isso, é preciso importar o arquivo correspondente, registrar o componente e, somente após isso, utilizá-lo no HTML.

Questão 2

Uma aplicação de rede social tem um componente de perfil de usuário (Perfil.vue) que mostra a foto, a bio e os seguidores de determinado usuário. Esse componente espera receber um objeto usuário com os dados necessários. De que maneira um componente pai pode passar informações para esse componente filho?

- A** Por meio de computed properties.
- B** Por meio de slots.
- C** Utilizando props.

D

Por meio de métodos.

E

Por meio do console.log.

Parabéns! A alternativa C está correta.

Ao utilizar props, é possível passar dados para os componentes filhos. Em nosso exemplo, um componente perfil precisa receber as informações para mostrá-las corretamente. Por isso, os dados do usuário são passados como propriedade do perfil.

Considerações finais

Este conteúdo tem como principal objetivo habilitá-lo(a) para a utilização do Vue.js na construção de interfaces de usuário (UI) reativas de pequeno porte. Por isso, vimos a utilização do Vue no controle de pequenas partes do HTML, bem como a divisão da aplicação em componentes, o que facilita a manutenção da aplicação.

Abordamos tais assuntos visando à maior quantidade de prática possível. Recomendamos que você procure criar aplicações próprias. Tente coisas simples, como um contador de cliques que muda a cor de um elemento se a contagem passar de 5, por exemplo. Após isso, avance para coisas mais complexas.

Pontuamos, por fim, que o Vue é um framework progressivo cheio de bibliotecas, embora não precisemos utilizar todo o seu poder de uma vez. Não pressione demais a si mesmo para aprender os mais complexos exemplos dessa ferramenta.

Em vez disso, crie coisas simples. Mas crie! Faça isso sem se preocupar de estar usando a biblioteca do Vue mais famosa do momento.

O podcast mostra as vantagens de utilizar bibliotecas que estendam as funcionalidades do Vue.js e facilitem o trabalho do desenvolvedor.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



Explore +

Interessado em saber mais detalhes sobre os assuntos abordados neste conteúdo? Consulte os tópicos a seguir no site MDN Web Docs:

Atributos HTML

MDN Web Docs. **Atributos – HTML: linguagem de marcação de hipertexto**. Consultado na Internet em: 24 jun. 2022.

Método getTime()

MDN Web Docs. **Date.prototype.getTime()** - JavaScript. Consultado na Internet em: 24 jun. 2022.

Seletores CSS

MDN Web Docs. **Seletores CSS – CSS**. Consultado na Internet em: 24 jun. 2022.

Trabalhando com JSON

MDN Web Docs. **Trabalhando com JSON – introdução a objetos em JavaScript**. Consultado na Internet em: 24 jun. 2022.

Referências

MARTIN, R. C. **Código limpo**: habilidades práticas do Agile Software. Rio de Janeiro: Alta Books, 2009.



Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.



Download material

O que você achou do conteúdo?



Relatar problema