Centro Acadêmico de Engenharia de Computação Alan Turing - CAECOM

I Semana Acadêmica de Engenharia de Computação - SECOM

Single Page Application: Um novo conceito de aplicações web

Instrutor: Otávio Augusto Paganotti Messias da Silva

Roteiro

- Introdução
- Criando um arquivo .html
- Entrar no site vuejs.org
    - Getting Started
    - CDN do Vue.js
- Anexar a CDN dentro do arquivo .html
- **Instanciando o Vue**
    - Iniciar o Bloco script instanciando o Vue

```
<div id="app">
  <h1>{message}</h1>
</div>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">script>
<script type="text/javascript">
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue'
  }
})
</script>
```

    - Breve explicação sobre como o Vue cria o DOM do navegador
        - O DOM (Document Object Model) é uma interface que representa como os documentos HTML e XML são lidos pelo seu browser. Após o browser ler seu documento HTML, ele cria um objeto que faz uma representação estruturada do seu documento e define meios de como essa estrutura pode ser acessada. Nós podemos acessar e manipular o DOM com JavaScript, é a forma mais fácil e usada.
    - Adicionar um método simples que retorna uma string (introdução aos métodos)

```
methods: {
  sayHello: function() {
    return 'Hello'
  }
}
```

    - Dentro do javascript do Vue, nós podemos chamar elementos (métodos, dados, etc.) por meio do “this”, mas temos que ficar atentos, pois o “this” só funciona dentro do bloco javascript, e não dentro do HTML, pois o Vue consegue identificar qual elemento estamos falando dentro do HTML, mas não consegue identificar dentro do HTML, pois pode ser que tenha uma variável local dentro do método, e isto pode gerar algum tipo de conflito.

```
digaOla: function() {
  return this.message
}
```

- **v-bind**
  - Foi-se criado uma variável dentro do vue com um link inserido.

```
data: {
  link: 'https://google.com'
}
```

  - A coisa mais lógica a se fazer, é utilizar a interpolação e adicionar o "link" dentro de uma tag <a> certo?

```
<h1>{{message}} - <a href="{{link}}">Google</a></h1>
```

  - Mas veremos que assim não funciona, dentro de "href" é esperado que seja inserido um link de verdade (um link estático), e não "{ link }".
  - Para adicionarmos um link dinâmico dentro de "href", precisamos de uma diretiva vue, chamada "v-bind" e esta diretiva diz ao Vue.js - "Ei, não utilize a atribuição normal aqui, não utilize uma atribuição html normal, ao invés disso, linke com esta variável".

```
<h1>{{message}} - <a v-bind:href="link">Google</a></h1>
```

  - E com isso, conseguimos 'linkar' a variável que queremos no atributo html que desejamos.
- **Entendendo o uso de diretivas**
  - Com o uso das diretivas, o vue nos permite fazer coisas assim, mas como isso funciona? O que é uma diretiva?
  - Uma diretiva é simplesmente uma instrução que você coloca no seu código, e o vue identifica esta instrução e adiciona ela nos atributos que serão dinamicamente montados na hora em que o DOM é montado.
  - Ou seja, você apenas diz ao vue, que irá linkar o valor do atributo, com um método "data" dentro da instância vue.
- **v-html (Saída bruta de HTML)**
  - Se quisermos colocar um bloco html inteiro dentro de uma variável vue, e queremos imprimir como um bloco HTML, o vue simplesmente iria imprimir todo o bloco html como se fosse um texto simples.

```
<h1><span v-html="finishedLink"></span></h1>
```

  - Utilizando a diretiva v-html, o vue entende que o bloco a ser montado no DOM, se trata de um bloco html a ser impresso na tela.
- **v-on Ouvindo Eventos**
  - Utilizamos a diretiva v-on para trazer o evento para dentro do vue, agindo de maneira oposta ao v-bind, que pega algo do vue e leva para dentro do html.
  - Temos um botão onde queremos contar quantas vezes ele foi clicado:

```
<button v-on:click="incremento">Clique aqui</button>
<p>{{counter}}</p>
```

```
counter: 0

incremento: function() {
  this.counter += 1
}
```

- Aqui tem outro exemplo de v-on utilizando as coordenadas do navegador, com o atributo "mousemove":

```
atualizaCoordenadas: function(event) {
  this.x = event.clientX
  this.y = event.clientY
}

x: 0,
y: 0

<p v-on:mousemove="atualizaCoordenadas">Coordanadas: {{x}} / {{y}}</p>
```

- Também podemos adicionar nossos próprios parâmetros dentro do v-on

```
<button v-on:click="incremento(2)">Clique aqui</button>

incremento: function(pulo) {
  this.counter += pulo
}
```

- Se eu quiser passar meu próprio evento, e o evento padrão do Vue, eu devo adicionar

```
<button v-on:click="incremento(2, $event)">Clique aqui</button>
```

- **v-on:keyup Ouvindo Eventos do teclado**
  - Podemos capturar eventos, não apenas com clique do mouse, ou com os movimentos do mesmo. Podemos utilizar todas as teclas do teclado disponíveis para serem criados eventos.

```
<input type="text" v-on:keyup.enter="meAlerte">

meAlerte: function() {
  alert('Alerta!')
}
```

  - v-on:keyp.enter.exact faz com que esta única tecla combinada execute a tal função
- **Escrevendo código JS dentro do Template**
  - O vue suporta trechos de código javascript dentro de suas interpolações HTML, e também dentro de suas diretivas.

```
<button v-on:click="counter++">Clique aqui</button>
{{counter * 2 > 10 ? 'Maior que 10' : 'Menor que 10'}}
```

- **Two-way-binding v-model**
  - Se quisermos alterar em tempo real o valor de uma variável, utilizando um misto entre v-on e v-bind (Ou seja, trazer o dado de dentro da instância vue, e depois enviá-lo para a instância vue), podemos fazer o uso da diretiva v-model.

```
<input type=”text” v-model=”name”>
name: ‘Otávio’
```

- **Estilos dinâmicos com Classes CSS**

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <style>
      .demo {
        width: 100px;
        height: 100px;
        background-color: #ccc;
        margin: 10px;
        display: inline-block;
      }
      .red {
        background-color: red;
      }
      .green {
        background-color: green;
      }
      .blue {
        background-color: blue;
      }
    </style>
  </head>
  <body>
    <div id="app">
      <div
        class="demo"
        @click="isRed = !isRed"
        :class="{red: isRed}"
      ></div>
      <div
        class="demo"
        @click="isBlue = !isBlue"
        :class="{blue: isBlue}"
      ></div>
      <div
        class="demo"
        @click="isGreen = !isGreen"
        :class="{green: isGreen}"
      ></div>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js">
    </script>
```

```
    <script type="text/javascript">
      new Vue({
        el: '#app',
        data: {
          isRed: false,
          isBlue: false,
          isGreen: false
        }
      })
    </script>
  </body>
</html>
```

```
<div
  class="demo"
  @click="isGreen = !isGreen"
  :style="{‘background-color’: color}"
></div>
<input type=”text” v-model=”color”>

color: ‘gray’
```

- **Listas e Condicionais**
  - v-if
    - v-if adiciona ou remove totalmente o bloco html de dentro do DOM do navegador. Isso significa que se o v-if retornar falso, o elemento HTML não será renderizado de forma nenhuma dentro do navegador.

```
<div id="app">
  <p v-if="show">Agora você me vê</p>
  <p v-if="!show">Agora não vê mais</p>
  <button @click="show = !show">Trocar</button>
</div>
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script type="text/javascript">
  new Vue({
    el: '#app',
    data: {
      show: true
    }
  })
</script>
```

  - v-else

```
<p v-else>Agora não vê mais</p>
```

  - v-show
    - O v-show apenas esconde ou não o elemento que deseja. Ao contrário do v-if, que remove ou adiciona o elemento do DOM do navegador.

```
<p v-show="!show">Aqui temos um v-show</p>
```

- v-for
  - Se utiliza de arrays e objetos para imprimir dinamicamente os elementos de dentro do vue para o html.
  - Todos sabemos que quando nós formos construir uma aplicação real, os dados não serão estáticos dentro da página, e estarão sujeitos a modificações a todo tempo. Temos que trazer estes dados dinâmicos de alguma forma, sem saber se está vindo 1 dado, ou 1000.
  - Para isso utilizamos o v-for, que é um laço de repetição que o vue fornece para nós.

```
<ul v-for="ingrediente in ingredientes">
  <li>{{ingrediente}}</li>
</ul>
ingredientes: ['Carne', 'Frutas', 'Ovos']
```

  - Acessar o Index (key) do laço de repetição

```
<ul v-for="(ingrediente, index) in ingredientes">
  <li>{{ingrediente}} ({{index}})</li>
</ul>
```

  - v-for aninhados para imprimir um array de objetos:

```
<ul v-for="pessoa in pessoas">
  <li v-for="(valor, index) in pessoa">{{index}}: {{valor}}</li>
</ul>
pessoas: [
  {nome: 'Max', idade: 28, cor: 'red'},
  {nome: 'Anna', idade: 'unknown', color: 'blue'}
]
```