**SESSÃO 2**

**Links:**

[**https://br.vuejs.org/v2/guide/**](https://br.vuejs.org/v2/guide/)

[**https://br.vuejs.org/v2/guide/syntax.html**](https://br.vuejs.org/v2/guide/syntax.html)

[**https://br.vuejs.org/v2/guide/events.html**](https://br.vuejs.org/v2/guide/events.html)

[**https://br.vuejs.org/v2/guide/computed.html**](https://br.vuejs.org/v2/guide/computed.html)

[**https://br.vuejs.org/v2/guide/class-and-style.html**](https://br.vuejs.org/v2/guide/class-and-style.html)

**AULA 18 – Binding de Atributos**

1. Para passarmos valores para as propriedades das tags html temos que utilizar diretivas, uma delas é a v-bind que irá resolver a variável link dento do href da tag <a>
2. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
3. <div *id*="app">
4. <p>{{titulo}}</p>
5. <p>{{saudacao()}}</p>
6. <a *v-bind:href*="link">Google</a>
7. </div>
8. <script>
9. **new** Vue({
10. el: '#app',
11. data: {
12. titulo: 'Usando VueJs 2!',
13. link: 'http://google.com.br'
14. },
15. methods: {
16. saudacao: function(){
17. return  *this*.titulo;
18. }
19. }
20. })
21. </script>
22. C

**Aula 20 – Evitando re-renderização**

1. Para evitar a re-redenrização de uma variável, temos que utilizar a diretiva v-once para que aquele valor somente seja carregado uma vez e não seja alterado.
2. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
3. <div *id*="app">
4. <p *v-once*>{{titulo}}</p>
5. <p>{{saudacao()}}</p>
6. <a *v-bind:href*="link">Google</a>
7. </div>
8. <script>
9. **new** Vue({
10. el: '#app',
11. data: {
12. titulo: 'Usando VueJs 2!',
13. link: 'http://google.com.br'
14. },
15. methods: {
16. saudacao: function(){
17. *this*.titulo = 'Bom dia!!!'
18. return  *this*.titulo;
19. }
20. }
21. })
22. </script>

**Aula 21 – Imprimir HTML puro**

1. Em algumas situações temos que imprimir html puro através do Javascript então temos que fazer uma interpolação. Para isso utilizamos a diretiva v-html=’variavel do data()’
2. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
3. <div *id*="app">
4. <p *v-once*>{{titulo}}</p>
5. <p>{{saudacao()}}</p>
6. <a *v-bind:href*="link">Google</a>
7. <hr>
8. <p *v-html*='linkHtml'></p>
9. </div>
10. <script>
11. **new** Vue({
12. el: '#app',
13. data: {
14. titulo: 'Usando VueJs 2!',
15. link: 'http://google.com.br',
16. linkHtml:'<a href="http://google.com.br">google</a>'
17. },
18. methods: {
19. saudacao: function(){
20. *this*.titulo = 'Bom dia!!!'
21. return  *this*.titulo;
22. }
23. }
24. })
25. </script>
26. DESAFIO

**Aula 24 – Eventos – diretiva v-on:click**

1. Podemos utilizar a diretiva v-on para chamar uma função, dentro do elemento html utilizamos v-on:’tipo de evento’ onde o tipo do evento seria por exemplo click.
2. No exemplo abaixo temos uma variável ‘contador’ que é mostrada dentro de um parágrafo. Temos também um botão que utiliza a diretiva v-on:click que irá executar o método somar.
3. Toda vez que clicarmos no botão o contador irá incrementar.
4. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5. <div *id*="app">
6. <p>{{contador}}</p>
7. <button *v-on:click*="somar">Somar 1</button>
8. </div>
9. <script>
10. **new** Vue({
11. el: '#app',
12. data: {
13. contador: 0
14. },
15. methods: {
16. somar() {
17. return *this*.contador++
18. }
19. }
20. })
21. </script>

**Aula 25 – Eventos – diretiva v-on:mousemove**

1. Neste exemplo iremos mostrar como capturar a posição do mouse, faremos isso utilizando a diretiva v-on:mousemove. Utilizaremos o exemplo anterior onde iremos acrecescentar um parágrafo ao template e nesse parágrafo iremos incluir x e y que serão as variáveis da posição do mouse.
2. Dentro do data() iremos incluir então as variáveis x e y com valor iniciamente de zero.
3. Iremos acrescentar outro método chamado atualizarXY que irá receber a posição do mouse através do evento clientX e clientY e irá passa-los para as variáveis x e y.
4. Para vermos os eventos que estão dentro de mousemove, podemos utilizar o console.log(event) para mostrar os eventos que estão dentro de mousemove.
5. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
6. <div *id*="app">
7. <p>{{contador}}</p>
8. <button *v-on:click*="somar">Somar 1</button>
9. <p *v-on:mousemove*="atualizarXY">Mouse: {{x}} e {{y}}</p>
10. </div>
11. <script>
12. **new** Vue({
13. el: '#app',
14. data: {
15. contador: 0,
16. x: 0,
17. y: 0
18. },
19. methods: {
20. somar() {
21. return *this*.contador++
22. },
23. atualizarXY(*event*){
24. //console.log(event);
25. *this*.x = *event*.clientX,
26. *this*.y = *event*.clientY
27. }
28. }
29. })
30. </script>

**Aula 26 – Eventos – Passando seus próprios argumentos e eventos**

1. Utilizando o exemplo anterior, iremos incluir no template na diretiva v-on:click=somar, um parâmetro que neste caso será o número 5. E iremos alterar o método soma passando para ele um parâmetro que iremos chamar de passo. Toda vez que clicarmos no botão o contador será acrescido do passo que é 5.
2. Podemos também passar eventos como parâmetros, mas para fazer isso temos que utilizar no template a expressão chave ‘$event’, essa expressão não pode ser diferente caso contrário ocorrerá um erro.
3. Dentro do método passamos então o outro parâmetro que neste caso será o evento que está sendo passado pelo template.
4. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5. <div *id*="app">
6. <p>{{contador}}</p>
7. <button *v-on:click*="somar(5, $event)">Somar 1</button>
8. <p *v-on:mousemove*="atualizarXY">Mouse: {{x}} e {{y}}</p>
9. </div>
10. <script>
11. **new** Vue({
12. el: '#app',
13. data: {
14. contador: 0,
15. x: 0,
16. y: 0
17. },
18. methods: {
19. somar(*passo*, *evento*) {
20. console.log(*this*.contador, *evento*);
21. return *this*.contador+=*passo*
22. },
23. atualizarXY(*event*){
24. //console.log(event);
25. *this*.x = *event*.clientX,
26. *this*.y = *event*.clientY
27. }
28. }
29. })
30. </script>

**Aula 27 – Modificadores de eventos**

1. Iremos utilizar o exemplo anterior onde iremos acrecentar um span ao final do parágrafo que possui a diretiva v-on com o evento mousemove recebendo o método atualizarXY. Ao movermos o mouse sobre o parágrafo, é passado a posição X e Y para as variáveis X e Y, então iremos limitar o posicionamento somente até o span.
2. Iremos utilizar então dentro do span a diretiva v-on novamente com o evento mousemove passando para ele o método parar.
3. Nesse método iremos receber o evento e iremos utilizar a função stopPropagation().
4. Ou podemos simplesmente utilizar o modificador de evento .stop no evento mousemove, ficando mousemove.stop e desta forma o evento do mouse será parado dentro do span.
5. Outro tipo de modificador de evento é o .stop.prevent que irá fazer com que o comportamento padrão não seja executado. Para exemplificar criamos uma âncora que terá a diretiva v-on:click e terá o link para o Google. Ao utilizarmos o .stop.prevent o link não irá funcionar.
6. Temos também o modificador de evento preventDefault() que será utilizado dentro de método chamado naoNavegar() esse método irá receber o evento do click porém será passado para esse evento a função .preventDefault() que irá interromper a execução padrão.
7. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
8. <div *id*="app">
9. <p>{{contador}}</p>
10. <button *v-on:click*="somar(5, $event)">Somar 1</button>
11. <p *v-on:mousemove*="atualizarXY">
12. Mouse: {{x}} e {{y}}
13. <span *v-on:mousemove*="parar">Parar Aqui!!!</span>
14. <span *v-on:mousemove.stop.prevent*>Parar Aqui!!!</span>
15. </p>
16. <a *v-on:click.prevent* *href*="http://google.com">Google</a>
17. <br>
18. <a *v-on:click*="naonavegar" *href*="http://google.com">Google</a>
19. </div>
20. <script>
21. **new** Vue({
22. el: '#app',
23. data: {
24. contador: 0,
25. x: 0,
26. y: 0
27. },
28. methods: {
29. somar(*passo*, *evento*) {
30. console.log(*this*.contador, *evento*);
31. return *this*.contador+=*passo*
32. },
33. atualizarXY(*event*){
34. //console.log(event);
35. *this*.x = *event*.clientX,
36. *this*.y = *event*.clientY
37. },
38. parar(*e*){
39. *e*.stopPropagation();
40. },
41. naonavegar(*e*){
42. *e*.preventDefault();
43. }
44. }
45. })
46. </script>

**Aula 28 – Modificadores de evento no teclado**

1. Vamos copiar o exemplo anterior e iremos criar um input do tipo texto, nesse input iremos utilizar a diretiva v-on:keyup e passaremos para ele um método chamado exibiralerta.
2. Nesse método iremos simplesmente exibir um alerta com a mensagem exibir alerta. Fazendo desta forma toda vez que digitarmos uma letra dentro do input e soltarmos a tecla será exibido um popup com um alerta e a mensagem exibir alerta.
3. Podemos modificar esse evento por exemplo acrescentando o .enter na frente de keyup e faremos com que o método exibirAlerta seja disparado somente quando apertarmos a tecla enter.
4. Podemos também fazer um encadeamento acrescentando em frente ao enter por exemplo .alt, desta forma o método somente será chamado quando for apertado as teclas alt+enter. Isso facilita a execução de métodos que precisam serer executados somente quando uma tecla ou várias teclas forem pressionadas.
5. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
6. <div *id*="app">
7. <p>{{contador}}</p>
8. <button *v-on:click*="somar(5, $event)">Somar 1</button>
9. <p *v-on:mousemove*="atualizarXY">
10. Mouse: {{x}} e {{y}}
11. <span *v-on:mousemove*="parar">Parar Aqui!!!</span>
12. <span *v-on:mousemove.stop.prevent*>Parar Aqui!!!</span>
13. </p>
14. <a *v-on:click.prevent* *href*="http://google.com">Google</a>
15. <br>
16. <a *v-on:click*="naonavegar" *href*="http://google.com">Google</a>
17. <hr>
18. <input *type*="text" *v-on:keyup.enter.alt*="exibirAlerta">
19. </div>
20. <script>
21. **new** Vue({
22. el: '#app',
23. data: {
24. contador: 0,
25. x: 0,
26. y: 0
27. },
28. methods: {
29. somar(*passo*, *evento*) {
30. console.log(*this*.contador, *evento*);
31. return *this*.contador+=*passo*
32. },
33. atualizarXY(*event*){
34. //console.log(event);
35. *this*.x = *event*.clientX,
36. *this*.y = *event*.clientY
37. },
38. parar(*e*){
39. *e*.stopPropagation();
40. },
41. naonavegar(*e*){
42. *e*.preventDefault();
43. },
44. exibirAlerta(*e*){
45. alert('Exibir alerta');
47. }
48. }
49. })
50. </script>
51. **Desafio 02**

**Aula 32 – Código javaScript no template**

1. Vamos ver mais alguns detalhes sobre a interpolação, para isso iremos copiar o exemplo 09 e renomear para 10\_interpolação.
2. No parágrafo iremos multiplicar a variável contador por 2 e veremos esse resultado ser aplicado na tela.
3. Vamos copiar o parágrafo anterior e vamos incluir dentro das chaves duplas uma expressão ternária para que seja exibido uma mensagem levando em conta o valor do contador.
4. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5. <div *id*="app">
6. <p>{{contador \* 2}}</p>
7. <p>{{contador > 10 ? 'Maior que 10' : 'Menor que 10'}}</p>
8. <button *v-on:click*="somar">Somar 1</button>
9. </div>
10. <script>
11. **new** Vue({
12. el: '#app',
13. data: {
14. contador: 0
15. },
16. methods: {
17. somar() {
18. *this*.contador++;
19. }
20. }
21. })
22. </script>

**Aula 32 – Two-Way-Binding**

1. Iremos utilizar o exemplo anterior porém iremos formata-lo para que ele fique somente com um parágrafo dentro do template e neste parágrafo terá um doble mustache com a variável titulo.
2. Teremos dentro da propriedade data do Vue, a variável titulo como o texto ‘Usando o Vue Js’.
3. Iremos criar um input do tipo texto que terá a diretiva v-bind:value onde passaremos para ela a variável ‘titulo’ dessa forma o valor da variável ‘titulo’ será populada dentro do input quando a tela for renderizada. Isso é um bind de uma única via ou seja o dado somente vai para o template.
4. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5. <div *id*="app">
6. <p>{{titulo}}</p>
7. <input *type*="text" *v-bind:value*="titulo">
8. </div>
9. <script>
10. **new** Vue({
11. el: '#app',
12. data: {
13. titulo: 'Usando o Vue JS!'
14. }
15. })
16. </script>
17. Se quisermos alterar a variável input no backend temos que utilizar a diretiva v-model passando para ela a variável ‘titulo’, dessa forma ao alterarmos o valor no input a alteração será realizada também no backend, ou seja será realizada a alteração nas duas direções.
18. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
19. <div *id*="app">
20. <p>{{titulo}}</p>
21. <input *type*="text" *v-model*="titulo">
22. </div>
23. <script>
24. **new** Vue({
25. el: '#app',
26. data: {
27. titulo: 'Usando o Vue JS!'
28. }
29. })
30. </script>

**Aula 33 – Propriedades Computadas**

1. Neste exemplo iremos criar um template com um parágrafo onde termos uma interpolação com a variável ‘titulo’, para que toda vez que a variável for alterada essa alteração seja refletida na view.
2. Teremos um botão com a diretiva v-on:click que receberá o método aumentar esse método irá incrementar a variável e irá atualizar a variável resultado com a lógica de exibir a mensagem ‘Maior ou igual a 5’ ou ‘Menor que 5’.
3. Teremos também um parágrafo com a interpolação da variável ‘contador’ para exibir o valor dessa variável na view.
4. Teremos outro parágrafo com a interpolação da variável ‘resultado’ .
5. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
6. <div *id*="app">
7. <button *v-on:click*="aumentar">Aumentar</button>
8. <p>Contador: {{contador}}</p>
9. <p>Resultado: {{resultado}}</p>
10. </div>
11. <script>
12. **new** Vue({
13. el: '#app',
14. data: {
15. contador: 0,
16. resultado: ''
17. },
18. methods:{
19. aumentar(){
20. *this*.contador++
21. *this*.resultado = *this*.contador >= 5 ?
22. 'Maior ou igual a 5' : 'Menor que 5'
23. },
25. }
26. })
27. </script>
28. Vamos melhorar o exemplo anterior incluindo mais um botão com a diretiva v-on:click chamando o método ‘diminuir’.
29. Temos então que criar o método ‘diminuir’ esse método irá decrementar a variável contador além de atualizar a variável resultado.
30. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
31. <div *id*="app">
32. <button *v-on:click*="aumentar">Aumentar</button>
33. <button *v-on:click*="diminuir">Diminuir</button>
34. <p>Contador: {{contador}}</p>
35. <p>Resultado: {{resultado}}</p>
36. </div>
37. <script>
38. **new** Vue({
39. el: '#app',
40. data: {
41. contador: 0,
42. resultado: ''
43. },
44. methods:{
45. aumentar(){
46. *this*.contador++
47. *this*.resultado = *this*.contador >= 5 ?
48. 'Maior ou igual a 5' : 'Menor que 5'
49. },
50. diminuir(){
51. if(*this*.contador > 0){
52. *this*.contador--
53. *this*.resultado = *this*.contador >= 5 ?
54. 'Maior ou igual a 5' : 'Menor que 5'
55. }
56. }
57. }
58. })
59. </script>
60. Para melhorar mais o código retirando as repetições que estão ocorrendo dentro dos métodos. Iremos criar um função que irá retornar o resultado e portanto não iremos precisar da variável resultado.
61. Dentro do parágrafo onde está o resultado iremos alterar a interpolação passando o novo método resultado() dessa forma iremos mostrar o resultado.
62. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
63. <div *id*="app">
64. <button *v-on:click*="aumentar">Aumentar</button>
65. <button *v-on:click*="diminuir">Diminuir</button>
66. <p>Contador: {{contador}}</p>
67. <p>Resultado: {{resultado()}}</p>
68. </div>
69. <script>
70. **new** Vue({
71. el: '#app',
72. data: {
73. contador: 0
75. },
76. methods:{
77. aumentar(){
78. *this*.contador++
79. },
80. diminuir(){
81. if(*this*.contador > 0){
82. *this*.contador--
83. }
84. },
85. resultado(){
86. return *this*.contador >= 5 ?
87. 'Maior ou igual a 5' : 'Menor que 5'
88. }
89. }
90. })
91. </script>
92. C
93. Vamos agora utilizar a propriedade computada ‘computed’ no vue.
94. Para exemplificar a utilização da propriedade computada, vamos alterar o template incluindo um botão com a diretiva v-on:click passando para ela uma variável contador2 que será incrementada dentro da própria diretiva.
95. Iremos criar também dentro da propriedade data() a variável contador2.
96. Dentro do parágrafo iremos criar uma outra interpolação com a variável contador2.
97. Executando o arquivo veremos que ao clicarmos sobre o botão aumentar2 teremos o contador2 incrementado sendo mostrado na view.
98. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
99. <div *id*="app">
100. <button *v-on:click*="aumentar">Aumentar</button>
101. <button *v-on:click*="contador2++">Aumentar2</button>
102. <button *v-on:click*="diminuir">Diminuir</button>
103. <p>Contador: {{contador}} | {{contador2}}</p>
104. <p>Resultado: {{resultado()}}</p>
105. </div>
106. <script>
107. **new** Vue({
108. el: '#app',
109. data: {
110. contador: 0,
111. contador2: 0
112. },
113. methods:{
114. aumentar(){
115. *this*.contador++
116. },
117. diminuir(){
118. if(*this*.contador > 0){
119. *this*.contador--
120. }
121. },
122. resultado(){
123. return *this*.contador >= 5 ?
124. 'Maior ou igual a 5' : 'Menor que 5'
125. }
126. }
127. })
128. </script>
129. Porém se incluirmos dentro do método resultado um console.log com uma mensagem qualquer, veremos que mesmo clicando no botão que não chama o método resultado ele é executado assim mesmo.
130. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
131. <div *id*="app">
132. <button *v-on:click*="aumentar">Aumentar</button>
133. <button *v-on:click*="contador2++">Aumentar2</button>
134. <button *v-on:click*="diminuir">Diminuir</button>
135. <p>Contador: {{contador}} | {{contador2}}</p>
136. <p>Resultado: {{resultado()}}</p>
137. </div>
138. <script>
139. **new** Vue({
140. el: '#app',
141. data: {
142. contador: 0,
143. contador2: 0
144. },
145. methods:{
146. aumentar(){
147. *this*.contador++
148. },
149. diminuir(){
150. if(*this*.contador > 0){
151. *this*.contador--
152. }
153. },
154. resultado(){
155. console.log('metodo resultado chamado')
156. return *this*.contador >= 5 ?
157. 'Maior ou igual a 5' : 'Menor que 5'
158. }
159. }
160. })
161. </script>
162. Para resolver esse problema temos que utilizar a propriedade computed() e dentro dela colocamos o método resultado(). Porém temos alterar a interpolação do parágrafo alterando de resultado() para ‘resultado’, pois dentro das propriedades computadas os métodos são chamados como variáveis.
163. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
164. <div *id*="app">
165. <button *v-on:click*="aumentar">Aumentar</button>
166. <button *v-on:click*="contador2++">Aumentar2</button>
167. <button *v-on:click*="diminuir">Diminuir</button>
168. <p>Contador: {{contador}} | {{contador2}}</p>
169. <p>Resultado: {{resultado}}</p>
170. </div>
171. <script>
172. **new** Vue({
173. el: '#app',
174. data: {
175. contador: 0,
176. contador2: 0
177. },
178. computed: {
179. resultado(){
180. console.log('metodo resultado chamado')
181. return *this*.contador >= 5 ?
182. 'Maior ou igual a 5' : 'Menor que 5'
183. }
184. },
185. methods: {
186. aumentar(){
187. *this*.contador++
188. },
189. diminuir(){
190. if(*this*.contador > 0){
191. *this*.contador--
192. }
193. }
195. }
196. })
197. </script>
198. Fazendo isso conseguimos resolver o problema da execução do método resultado.

**Aula 34 – Monitorando as mudanças**

1. Agora iremos ver sobre uma propriedade da view instance chamada watch que nos possibilita chamar métodos assíncronos ou seja métodos que não possuem uma resposta pronta no exato momento em que é chamado.
2. Para exemplificar iremos utilizar o exemplo anterior.
3. Iremos cria a propriedade watch e dentro dela vamos passar a variável que será monitorada, neste caso contador, sendo que contador terá dois estados um antes e outro depois da alteração.
4. Se executarmos o código, veremos que no console será passado o valor anterior e o valor novo da variável contador.
5. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
6. <div *id*="app">
7. <button *v-on:click*="aumentar">Aumentar</button>
8. <button *v-on:click*="contador2++">Aumentar2</button>
9. <button *v-on:click*="diminuir">Diminuir</button>
10. <p>Contador: {{contador}} | {{contador2}}</p>
11. <p>Resultado: {{resultado}}</p>
12. </div>
13. <script>
14. **new** Vue({
15. el: '#app',
16. data: {
17. contador: 0,
18. contador2: 0
19. },
20. computed: {
21. resultado(){
22. console.log('metodo resultado chamado')
23. return *this*.contador >= 5 ?
24. 'Maior ou igual a 5' : 'Menor que 5'
25. }
26. },
27. watch:{
28. contador(*novo*, *antigo*){
29. console.log(*novo*, *antigo*)
30. }
31. },
32. methods: {
33. aumentar(){
34. *this*.contador++
35. },
36. diminuir(){
37. if(*this*.contador > 0){
38. *this*.contador--
39. }
40. }
42. }
43. })
44. </script>
45. c
46. Agora iremos incluir dentro da propriedade watch um setTimeout que irá zerar o contador após 2 segundos.
47. watch:{
48. contador(*novo*, *antigo*){
49. // console.log(novo, antigo)
50. setTimeout( ()=>{
51. *this*.contador=0
52. }, 2000)
53. }
54. },

**Aula 35 – Sintaxe Reduzida**

1. Nessa aula iremos ver uma outra forma de utilizar as diretivas no Vue.
2. Vamos criar outro arquivo e copiar o exemplo evento 1.
3. Iremos acrescentar um input tipo texto que terá a diretiva v-bind:value que receberá a variável contador, desta forma o valor da variável será mostrado no input.
4. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
5. <div *id*="app">
6. <p>{{contador}}</p>
7. <button *v-on:click*="somar">Somar 1</button>
8. <hr>
9. <input *type*="text" *v-bind:value*="contador">
10. </div>
11. <script>
12. **new** Vue({
13. el: '#app',
14. data: {
15. contador: 0
16. },
17. methods: {
18. somar() {
19. return *this*.contador++
20. }
21. }
22. })
23. </script>
24. Iremos reduzir o texto substituindo a diretiva v-on por @ e o v-bind por ‘:’ isso funcionará normalmente.
25. script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
26. <div *id*="app">
27. <p>{{contador}}</p>
28. <button *@click*="somar">Somar 1</button>
29. <hr>
30. <input *type*="text" *:value*="contador">
31. </div>
32. <script>
33. **new** Vue({
34. el: '#app',
35. data: {
36. contador: 0
37. },
38. methods: {
39. somar() {
40. return *this*.contador++
41. }
42. }
43. })
44. </script>

**Aula 36 - Desafio 3**

**Aula 38 – Estilos dinâmicos e classes css #01**

1. Vamos iniciar o trabalho com estilos dinâmicos no Vue JS.
2. Criar um novo arquivo que terá um template com uma div com um ID app e essa div terá mais três div’s e cada uma terá a mesma classe chamada demo.
3. Na tag style vamos estilizar a div com o ID app, com um display flex e um alinhamento horizontal de justify-content: space-between.
4. Vamos estilizar também a classe demo com uma largura e altura de 100px e um background-color grey.
5. Vamos criar três classes c1, c2 e c3 passando para elas três cores respectivamente red, green e blue.
6. Na propriedade data do vue vamos criar variável booleana chamada aplicarC1 com o valor de false.
7. Agora vamos utilizar a diretiva v-bind na primeira div do template e iremos passar para o v-bind uma class que será a c1 que receberá a variável aplicarC1. Ou seja nessa div teremos a possibilidade de utilizar o estilo de demo ou de c1 quando c1 for igual a true, para isso vamos utilizar dentro desta div a diretiva v-on:click que irá receber a variável aplicarC1 como true ou clicando novamente será passado para false.
8. <script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
9. <style>
10. *#app*{
11. display: flex;
12. justify-content: space-between;
13. }
14. *.demo*{
15. width: 100px;
16. height: 100px;
17. background-color: gray;
18. }
19. *.c1*{
20. background-color: red ;
21. }
22. *.c2*{
23. background-color: green ;
24. }
25. *.c3*{
26. background-color: blue ;
27. }
28. </style>
29. <div *id*="app">
30. <div *class*="demo" *:class*="{c1:aplicarC1}"
31. *@click*="aplicarC1 = !aplicarC1"></div>
32. <div *class*="demo"></div>
33. <div *class*="demo"></div>
34. </div>
35. <script>
36. **new** Vue({
37. el: '#app',
38. data: {
39. aplicarC1: false
40. },
41. methods: {
42. }
43. })
44. </script>

**Aula 39 – Estilo dinâmico e classe css #02**

1. Nós conseguimos criar várias classes css para o template, mas faz mais sentido nós criarmos uma propriedade computada que incluir todas as classes no template.
2. Então vamos utilizar o exemplo anterior e vamos incluir no v-bind:class a classe c2 que irá receber o contrário de aplicarC1.
3. <div *class*="demo" *:class*="{c1:aplicarC1, c2: !aplicarC1}"
4. Mas se isso crescer muito o template ficará muito carregado, para resolver isso vamos criar a propriedade computed.
5. Dentro de computed iremos criar uma função estilo1() que irá retornar um objeto com as chaves que são os nomes das classes css.
6. computed:{
7. return {
8. c1: *this*.aplicarC1,
9. c2: !*this*.aplicarC1,
10. }
11. }
12. Após isso basta retirarmos as classes dentro da diretiva :class passando para ela somente a função criada em computed.
13. <div *class*="demo" *:class*="estilo1"

**Aula 40 – Estilo dinâmico e classe css #03 – Aplicando classes com**

1. Nesta aula veremos que temos a possibilidade de setar o nome da classe css dinamicamente utilizando uma propriedade. E iremos ver também o uso de array para aplicar varias classes ao mesmo tempo.
2. Vamos utilizar o exemplo anterior.
3. Dentro do template vamos criar uma div com uma classe chamada caixas e dentro dessa div iremos colocar as div’s demo.
4. <div *class*="caixas">
5. <div *class*="demo" *:class*="estilo1"
6. *@click*="aplicarC1 = !aplicarC1"></div>
7. <div *class*="demo"></div>
8. <div *class*="demo"></div>
9. </div>
10. Vamos apagar #app na tag style e escrever .caixas.
11. *.caixas*{
12. display: flex;
13. justify-content: space-between;
14. }
15. O resultado será o mesmo.
16. Vamos dentro da propriedade data() do vue vamos criar a variável classeCss passando para ele inicialmente a classe ‘c1’.
17. data: {
18. aplicarC1: false,
19. classeCss: 'c1'
20. },
21. Agora no último div da classe demo vamos utilizar a diretiva v-bind:class passando para ela a variável classeCss, desta forma toda vez que a variável classeCss for alterada essa alteração será refletida dentro de class.
22. <div *class*="demo" *:class*="classeCss"></div>
23. Para vermos isso funcionar vamos inserir um input fora da div caixa e esse input terá a diretiva v-model recebendo a variável classeCss, desta forma toda vez que alterarmos a classe dentro do input, essa alteração será refletida no template.
24. <hr>
25. <input *type*="text" *v-model*="classeCss">
26. Outra coisa que podemos fazer é utilizar a sintaxe de array para utilizar mais de uma classe no v-bind:class.
27. <div *class*="demo" *:class*="[classeCss]"></div>
28. Para testar a utilização de várias classes, vamos por exemplo criar uma classe para girar o retângulo dentro da tag style.
29. *.girar*{
30. transform: rotate(35deg);
31. }
32. Agora vamos inserir dentro do array no v-bind:class um objeto que terá uma chave que será a classe girar e ela irá receber um valor true ou false, porém, neste caso será a variável aplicarGiro da propriedade data(), sendo que aplicar giro recebe true ou false, desta forma a classe girar só será aplicada quando estiver com o valor true.
33. <div *class*="demo" *:class*="[classeCss, {girar: aplicarGirar} ]"></div>
34. data: {
35. aplicarC1: false,
36. classeCss: 'c1',
37. aplicarGirar: true
38. },
39. Para aplicar o giro ao quadrado da div demo vamos criar um método dentro de methods, chamado giraGira que irá passar para a variável aplicarGirar o seu valor negado, ou seja o contrário, desta forma quando o método for chamado a variável irá mudar o valor de true para false ou vice versa.
40. methods:{
41. giraGira(){
42. *this*.aplicarGirar = !*this*.aplicarGirar
43. }
44. }
45. Agora para podemos colocar para funcionar vamos inserir no input uma diretiva v-on:keyup que irá receber o método giraGira, desta forma toda vez que uma tecla for liberada o método será executado e o valor da variável aplicarGirar irá mudar.
46. <input *type*="text"  *v-on:keyup*="giraGira" *v-model*="classeCss">

**Melhorando o exemplo:**

1. Para tornar o exemplo mais interessante, vamos fazer com que o método giraGira que está dentro da diretiva v-on:keyup no input, capture o evento e passe ele como parâmetro para o método.
2. <input *type*="text"  *v-on:keyup*="giraGira($event)" *v-model*="classeCss">
3. No método iremos capturar o evento através da variável especial chamada event, iremos verificar através do event.key se a tecla precisonada é 1, 2 ou 3 e só iremos alterar a variável aplicarGirar neste caso.
4. giraGira(*event*){
5. if(*event*.key=='1' || *event*.key=='2' || *event*.key=='3'){
6. console.log(*event*)
7. *this*.aplicarGirar = !*this*.aplicarGirar
8. }
9. }

**Aula 41 – Estilo dinâmico sem classe css**

1. Alguma vezes iremos trabalhar aplicando estilos diretamente nas tags html, para isso vamos utilizar o exemplo anterior, fazendo algumas alterações.
2. O arquivo ficará assim:
3. script *src*="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
4. <style>
5. *.caixas*{
6. display: flex;
7. justify-content: space-between;
8. }
9. *.demo*{
10. width: 100px;
11. height: 100px;
12. background-color: gray;
13. }
14. *.c1*{
15. background-color: red ;
16. }
18. *.c2*{
19. background-color: green ;
20. }
21. *.c3*{
22. background-color: blue ;
23. }
24. *.girar*{
25. transform: rotate(45deg);
26. }
27. </style>
28. <div *id*="app">
29. <div *class*="caixas">
30. <div *class*="demo" ></div>
31. <div *class*="demo"></div>
32. <div *class*="demo"></div>
33. </div>
34. <hr>
35. <input *type*="text" >
36. </div>
37. <script>
38. **new** Vue({
39. el: '#app',
40. data: {
42. },
43. computed:{
45. },
46. methods:{
48. }
49. })
50. </script>
51. Vamos inicialmente acrescentar na primeira div demo um v-bind:style passando para ela uma variável cor dentro de background-color.
52. <div *class*="demo" *:style*="{'background-color': cor}" ></div>
53. Iremos inserir a variável cor dentro da propriedade data()
54. data: {
55. cor: ''
56. },
57. Essa cor será alterada na propriedade data() através do v:model dentro do input, pois iremos passar para ele a variável cor. Desta forma o que digitarmos no input será passado para a div.
58. Podemos digitar tanto em inglês como ‘blue’ ou em hexdecimal.
59. <input *type*="text" *v-model*="cor" >
60. Outra possibilidade ao invés de utilizarmos a propriedade css background-color com aspas é utilizarmos a mesma propriedade porém no formato camelcase ou seja backgroundColor.
61. Quando formos utilizar várias propriedades css, o melhor será utilizar uma propriedade computada e criarmos um objeto que terá as propriedades necessárias.\
62. Vamos criar a variável largura dentro de data() passando 100 para ele
63. data: {
64. cor: 'red',
65. largura: 100
66. },
67. Então vamos acrescentar na propriedade computada uma propriedade chamada meuEstilo e dentro de return vamos inserir as variáveis largura e a cor, sendo que temos que concatenar a largura com px.
68. computed:{
69. meuEstilo(){
70. return{
71. backgroundColor : *this*.cor,
72. width : *this*.largura + 'px'
73. }
74. }
75. },
76. Se quisermos alterar por exemplo a altura do elemento div demo, podemos alterar a propriedade computada meuEstilo dentro da diretiva :style para um array e acrescentar um objeto onde height será a chave e altura será o valor.
77. <div *class*="demo" *:style*="[meuEstilo, {height: altura}]"></div>
78. Temos que acrescentar altura dentro de data()
79. data: {
80. cor: 'red',
81. largura: 100,
82. altura: 20
83. },

**Aulas 42 e 43 – Desafio estilos**

1. C