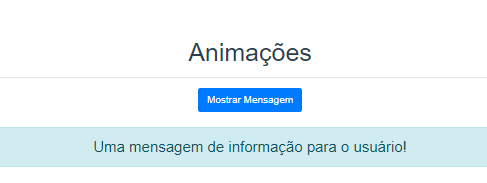
**MELHORANDO ESTÉTICA-TRANSIÇÕES-E-ANIMAÇÕES NO** **VUE.JS**

**Aula 210 – Preparando o código para as transições**

1. Vamos inserir algumas informações no template para podermos utilizar as transições.
2. Primeiramente incluía do campo data() as variáveis msg e exibir sendo que msg irá conter a mensagem que deverá ser mostrada para o usuário e exibir ser do tipo boolean que a principio será false.
3. No template vamos incluir um botão utilizando a estilização do bootstrap, o projeto foi configurado com o bootstrap, para isso bastou-se adiciona-lo ao projeto com npm i bootstrap-vue .
4. Vamos incluir também um b-alert.
5. Dentro do botão teremos uma chamada para o evento @click que ficará alterando o valor de exibir entre true e false toda vez que o botão for clicado.
6. Dentro de b-alert teremos um v-if que receberá a variável exibir, desta forma o alert somente será exibido quando a variável exibir for true.
7. <template>
8. <div *id*="app" *class*="container-fluid">
9. <h1>Animações</h1>
10. <hr>
11. <b-button *variant*="primary" *class*="mb-4"
12. @*click*="exibir = !exibir">Mostrar Mensagem</b-button>
13. <b-alert *variant*="info" *show* v-if="exibir">{{msg}}</b-alert>
14. </div>
15. </template>
16. <script>
17. export default {
18. data() {
19. return {
20. msg: 'Uma mensagem de informação para o usuário!',
21. exibir: false
22. }
23. },
24. }
25. </script>
26. <style>
27. *#app* {
28. font-family: 'Avenir', Helvetica, Arial, sans-serif;
29. *-webkit-font-smoothing*: antialiased;
30. *-moz-osx-font-smoothing*: grayscale;
31. text-align: center;
32. color: #2c3e50;
33. margin-top: 60px;
34. font-size: 1.5rem;
35. }
36. </style>
37. Para utilizarmos o efeito de transições temos que utilizar a tag <transition></transition> que executará a transição do elemento que está dentro da tag.
38. 

**Aula 211 – Configurando transição**

**Aula 212 – Definindo as classes CSS para transição**

1. Para utilizarmos as transições devemos definir as classes que serão utilizadas.
2. Na tag transition vamos incluir o name=’fade’
3. E na área de style iremos então definir as classes.
4. .fade-enter{} .fade-enter-active{}, .fade-enter-to{}, .fade-leave{}, .fade-leave-active{}, .fade-leave-to{}.
5. <style>
6. *#app* {
7. font-family: 'Avenir', Helvetica, Arial, sans-serif;
8. *-webkit-font-smoothing*: antialiased;
9. *-moz-osx-font-smoothing*: grayscale;
10. text-align: center;
11. color: #2c3e50;
12. margin-top: 60px;
13. font-size: 1.5rem;
14. }
15. *.fade-enter*{
16. }
17. *.fade-enter-active*{
18. }
19. *.fade-enter-to*{
20. }
21. *.fade-leave*{
22. }
23. *.fade-leave-enter*{
24. }
25. *.fade-leave-to*{
27. }
28. </style>

**Aula 213 – Criando transição Fade com propriedade CSS transition**

1. Vamos incluir na classe .fade-enter{} o atributo opacity com valor zero, ou seja no momento que o objeto for exibido ele estará invisível, na próxima classe .fade-enter-active{} utilizaremos agora o atributo transition passando para ele opacity 2s, ou seja o elemento irá demorar 2 segundos para sair da opacidade zero até um que é o máximo.
2. A classe .fade-enter-to{} por padrão o atributo opacity será 1, por isso podemos retirá-lo, mas caso a opacidade final não seja 1, devemos informar a opacidade de destino.
3. Temos então a transição de entrada, para fazermos a transição de saída basta configuramos as classes .fade-leave{} para 1 ou seja quando o elemento for ocultado ele começará com 1 e irá finalizar com 0 e o tempo de transição será 2s.
4. *.fade-enter*, *.fade-leave-to*{
5. opacity: 0;
6. }
7. *.fade-enter-active*, *.fade-leave-active*{
8. transition: opacity 2s;
9. }

**Aula 214 – Criando transição Slide com propriedade CSS.**

1. Vamos duplicar a tag transition anterior porém vamos nomeá-la com ‘slide’.
2. Teremos que definir as mesmas classes que foram definidas na transição anterior, ou seja, .slide-enter{} .slide-enter-active{} .slide-enter-to{} .slide-leave{} .slide-leave-active{} .slide-leave-to{}.
3. Porém vamos utilizar um recurso css chamado keyframes que são os atributos css que se quer aplicar em uma transformação.
4. Desta forma iremos agir nos eixos x, y e z.
5. Vamos então definir que o elemento irá sair de um ponto 0 e irá se mover 40px para cima.
6. Fazemos isso definindo da seguinte forma:
7. Entrada do slide -
8. @keyframes *slide-in*{
9. from { transform: translateY(40px);}
10. to {transform: translateY(0);}
11. }
12. Saída do slide –
13. @keyframes *slide-out*{
14. from { transform: translateY(0);}
15. to {transform: translateY(40px);}
16. }
17. Mas para a transição funcionar, agora devemos configurar as classes do slide.
18. Na classe .slide-enter-active{} nós iremos determinar uma animação do slide-in de 2s com o algoritmo ease, esse algoritmo é que dára a vida a animação e exite vários tipos.
19. Temos que configurar agora a animação de saída na classe .slice-leave-active{} que terá a animação slide-out com 2s e o mesmo algoritmo ease.
20. *.slide-enter-active*{
21. animation: slide-in 2s ease;
22. }
23. *.slide-leave-active*{
24. animation: slide-out 2s ease;
25. }

**Aula 215 – Misturando as propriedades Animation e transition**

1. Caso o tempo de transição das duas animações sejam diferentes, basta utilizarmos o atributo type na tag <transition> passando para o type qual a transição que irá mandar.
2. Para realizarmos a mistura dos dois tipos de animação basta utilizarmos as duas animações nas classes .slide-enter-active{} e .slide-leave-active{}
3. *.slide-enter-active*{
4. animation: slide-in 2s ease;
5. transition: opacity 2s;
6. }
7. *.slide-leave-active*{
8. animation: slide-out 2s ease;
9. transition: opacity 2s;
10. }

**Aula 216 – V-show**

1. Podemos utilizar o v-show da mesma forma que utilizamos o v-if.
2. <transition *name*="slide" *appear* >
3. <b-alert *variant*="info" *show* *v-show*="exibir">{{msg}}</b-alert>
4. </transition>

**Aula 217 – Configurando animação no carregamento do** **componente.**

1. Para podermos carregar a transição de um componente ao iniciar a aplicação devemos utilizar o atributo appear na tag <transition>, para exemplificar vamos utilizar na transition de slide.
2. <transition *name*="slide" *appear* >
3. <b-alert *variant*="info" *show* *v-show*="exibir">{{msg}}</b-alert>
4. </transition>
5. E passar a variável exibir para true.
6. data() {
7. return {
8. msg: 'Uma mensagem de informação para o usuário!',
9. exibir: true
10. }

**Aula 218 – Usando nomes diferentes de classes CSS**

1. É possível utilizar outro padrão de nomenclatura das classes do vue, sem problema nenhum, para isso.
2. Vamos encontrar a biblioteca na página <https://animate.style/>
3. Que possui várias animações. Vamos copiar o link do cdn e colar na nossa página index.html.
4. <link
5. rel="stylesheet"
6. href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css"
7. />
8. Vamos duplicar a última transition do template e retirar o nome e appear.
9. dentro da tag transition vamos incluir as classes enter-active-class e leave-active-class e iremos utilizar uma classe da biblioteca animate.
10. <transition
11. *enter-active-class*="animated bounce"
12. *leave-active-class*="animated shake"  >
13. <b-alert *variant*="info" *show* *v-show*="exibir">{{msg}}</b-alert>
14. </transition>
15. Passe o valor da variável exibir para false.

**Aula 219 – Usando nomes e atributos dinâmicos**

1. Os nomes dos atributos da tag <transition> podem ser passados dinamicamente utilizando bind. Para exemplificar vamos duplicar a tag transition do slide.
2. <transition :*name*="tipoAnimacao" >
3. <b-alert *variant*="info" *show* *v-show*="exibir">{{msg}}</b-alert>
4. </transition>
5. Dentro de data() vamos criar uma variável chamada tipoAnimacao que terá um valor inicialmente de ‘fade’.
6. data() {
7. return {
8. msg: 'Uma mensagem de informação para o usuário!',
9. exibir: false,
10. tipoAnimacao: 'fade'
11. }
12. },
13. Agora no template vamos inserir uma tag <b-select> e dentro dela duas tags <option> com as opções de value ‘fade’e ‘slide’.
14. Na tag b-select iremos utilizar um v-model apontando para tipoAnimacao.
15. Na tag transition que foi duplicada vamos incluir um bind na propriedade :name passando tipoAnimacao para ela.
16. <b-select *v-model*="tipoAnimacao" *class*="mb-4">
17. <option *value*="fade">Fade</option>
18. <option *value*="slide">Slide</option>
19. </b-select>
20. <transition :*name*="tipoAnimacao" >
21. <b-alert *variant*="info" *show* *v-show*="exibir">{{msg}}</b-alert>
22. </transition>
23. C

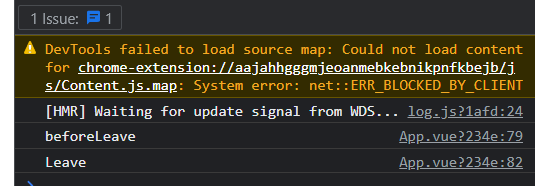
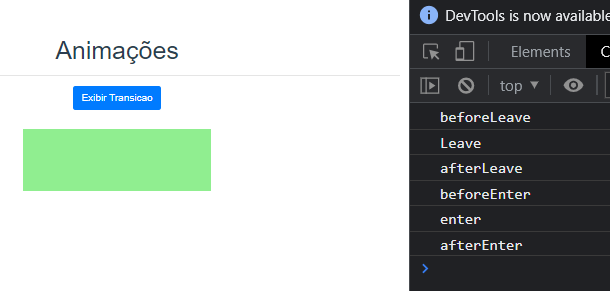
**Aula 220 – Transicionar entre múltiplos elementos**

1. Vamos duplicar o último <transition> e acrescentar mais um alert com a variant warning, devemos alterar também o v-show para v-if e v-else.
2. Porém para funcionar a transição entre eles devemos inserir mais um atributo nas tags alert, que será o key passando como valor info e warning.
3. <transition :*name*="tipoAnimacao" >
4. <b-alert *variant*="info" *show* v-if="exibir" *key*="info">{{msg}}</b-alert>
5. <b-alert *variant*="warning" *show* v-else *key*="warning">{{msg}}</b-alert>
6. </transition>
7. Mas dessa forma ainda aparece um erro pois os elementos não estão com a ordem de entrada e saída bem definidos, para resolver isso vamos incluir um atributo em transition chamado mode passando para ele ‘out-in’ou seja primeiramente será mostrado o elemento que está saindo para depois mostrar o elemento que está entrando.

**Aula 221 – Escutando a eventos de transição Hooks**

1. Até o momento nós trabalhamos com transições somente no CSS, apartir desta aula iremos começar a trabalhar também com javaScritp, podendo ser separadas ou ao mesmo tempo.
2. Para isso podemos utilizar alguns métodos de ciclo de vida ou gatilhos em Js, chamados, before-enter, enter, after-enter, enter-cancelled , before-leave, leave, after-leave, leave-cancelled.

**Aula 222 – Entendendo animação em JavaScript**

1. Iremos exemplificar o funcionamento dos métodos de ciclo de vida do Javascript para animações que foram citados na aula anterior.
2. Para começar iremos criar outra variável chamada exibir2 passando true para ela.
3. data() {
4. return {
5. msg: 'Uma mensagem de informação para o usuário!',
6. exibir: false,
7. exibir2: true,
8. tipoAnimacao: 'fade'
9. }
10. },
11. No template no final vamos incluir um <hr> e um botão que ficará chamando o evento @click alternando os valores de exibir2 entre true e false para exibir ou não a div .caixa dentro de uma tag <transition>.
12. <hr>
13. <b-button @*click*="exibir2 = !exibir2" *variant*="success">Exibir Transicao</b-button>
15. <transition>
16. <div *class*="caixa"></div>
17. </transition>
18. Vamos configurar os atributos css da classe .caixa.
19. *.caixa*{
20. height: 100px;
21. width: 300px;
22. margin: 30px auto;
23. background-color: lightgreen;
24. }
25. Agora vamos atribuir um v-if a div. Caixa para exibir ou não a div .caixa conforme clicamos no botão.
26. <transition>
27. <div v-if="exibir2" *class*="caixa"></div>
28. </transition>
29. Após isso vamos definir os métodos hooks que serão chamados na transition com os respectivos métodos que eles irão executar, primeiramente os de entrada e depois os de saída.
30. <transition
31. @*before-enter*='beforeEnter'
32. @*enter*='enter'
33. @*after-enter*='afterEnter'
34. @*enter-cancelled*='enterCancelled'
35. @*before-leave*='beforeLeave'
36. @*leave*='leave'
37. @*after-leave*='afterLeave'
38. @*leave-cancelled*='leaveCancelled'   >
39. <div v-if="exibir2" *class*="caixa"></div>
40. </transition>
41. Feito isso vamos agora criar os métodos citados dentro dos hooks.
42. O primeiro método beforeEnter irá receber somente o elemento que será animado.
43. O método enter irá receber o elemento e uma função que será executada quando sua animação for concluída, isso porque o vue não sabe quando a animação oi concluída se vc não informar e desta forma o vue não irá passar para os outros métodos como o after-enter e after-leave.
44. A principio vamos comentar a função.
45. methods: {
46. beforeEnter(*el*){
47. console.log('beforeEnter')
48. },
49. enter(*el*, *done*){
50. console.log('enter')
51. // done()
52. },
53. afterEnter(*el*){
54. console.log('afterEnter')
55. },
56. enterCancelled(){
57. console.log('enterCancelled')
58. },
59. beforeLeave(*el*){
60. console.log('beforeLeave')
61. },
62. Leave(*el*, *done*){
63. console.log('Leave')
64. // done()
65. }
66. afterLeave(*el*){
67. console.log('afterLeave')
68. }
69. leaveCancelled(){
70. console.log('LeaveCancelled')
71. }
72. Agora vamos executar e verificar as mensagens geradas no console.
73. 
74. Vemos que o método afterLeave não foi chamado pois não foi definido o final da transição ou a função de fim não foi passada para Leave.
75. Se descomentarmos a função done() veremos que todos os métodos irão funcionar.
76. methods: {
77. beforeEnter(*el*){
78. console.log('beforeEnter')
79. },
80. enter(*el*, *done*){
81. console.log('enter')
82. done()
83. },
84. afterEnter(*el*){
85. console.log('afterEnter')
86. },
87. enterCancelled(){
88. console.log('enterCancelled')
89. },
90. beforeLeave(*el*){
91. console.log('beforeLeave')
92. },
93. leave(*el*, *done*){
94. console.log('Leave')
95. done()
96. },
97. afterLeave(*el*){
98. console.log('afterLeave')
99. },
100. leaveCancelled(){
101. console.log('LeaveCancelled')
102. }
103. },
104. C
105. 

**Aula 223 – Excluindo CSS da animação**

1. Para evitar que qualquer classe css interfira na execução da animação 100% Javascript, devemos utilizar dentro da tag <transition> um bind com a propriedade ‘:css’ passando false.
2. <transition :*css*="false" :*name*="tipoAnimacao"
3. @*before-enter*='beforeEnter'
4. @*enter*='enter'
5. @*after-enter*='afterEnter'
6. @*enter-cancelled*='enterCancelled'
7. @*before-leave*='beforeLeave'
8. @*leave*='leave'
9. @*after-leave*='afterLeave'
10. @*leave-cancelled*='leaveCancelled'   >
11. <div v-if="exibir2" *class*="caixa"></div>
12. </transition>

**Aula 224 – Criando animação em Javascript**

1. Vamos trabalhar com a div .caixa que foi criada anteriormente, alterando sua largura, saindo de 0 até 300.
2. Para começarmos vamos criar outra variável chamada larguraBase com valor 0.
3. data() {
4. return {
5. msg: 'Uma mensagem de informação para o usuário!',
6. exibir: false,
7. exibir2: false,
8. tipoAnimacao: 'slide',
9. larguraBase: 0
10. }
11. },
12. E quando clicarmos no botão a largura será incrementada até chegar a 300px, para isso iremos precisar de um temporizador.
13. Iremos alterar a função de entrada enter() criando uma variável rodada que recebera um valor inicial de 1 essa variável será utilizada para aumentar ou diminuir a escala de aumento da largura.
14. Vamos criar também uma variável chamada temporizador com setInterval() com um time de 20 milisegundos.
15. enter(*el*, *done*){
16. let rodada = 1
17. const temporizador = setInterval(() => {
18. const novaLargura = *this*.larguraBase + rodada \* 10
20. }, 20);
21. },
22. Agora dentro da função setInterval() vamos criar uma variável chamada novaLargura que irá receber o valor de this.larguraBase + rodada \* 10
23. Para alterar a largura do elemento iremos utilizar o el passado via parâmetro na função enter()
24. Para isso iremos chamar .style do el() passando .width com o valor da nova largura concatenando com a unidade.
25. enter(*el*, *done*){
26. let rodada = 1
27. const temporizador = setInterval(() => {
28. const novaLargura = *this*.larguraBase + rodada \* 10
29. *el*.style.width = `${novaLargura}px`
30. }, 20);
31. },
32. Temos que incrementar a variável rodada e quando chegar em 30 deveremos limpar o temporizador e finalizar a função com done(), para deixar claro que a função terminou e será possível chamar as outras funções do ciclo de vida.
33. Porém devemos garantir que a largura seja setada para 0 antes de aumenta-la, para isso utilizaremos o método do ciclo de vida beforeEnter().
34. beforeEnter(*el*){
35. *this*.larguraBase = 0
36. *el*.style.width = `${*this*.larguraBase}px`
37. },
38. enter(*el*, *done*){
39. let rodada = 1
40. const temporizador = setInterval(() => {
41. const novaLargura = *this*.larguraBase + rodada \* 10
42. *el*.style.width = `${novaLargura}px`
43. rodada++
44. if(rodada > 30) {
45. clearInterval(temporizador)
46. done()
47. }
48. }, 20);
49. },
50. Para realizar a diminuição da largura nós iremos utilizar o método leave() ou seja ao sair, para que quando clicarmos novamente, a largura diminua até 0.
51. Mas para isso temos que garantir que a largura base tenha no inicio do método, largura de 300px, para isso iremos utilizar o método beforeLeave() onde passaremos a largura para 300px.
52. beforeLeave(*el*){
53. *this*.larguraBase = 300
54. *el*.style.width = `${*this*.larguraBase}px`
55. },
56. leave(*el*, *done*){
57. let rodada = 1
58. const temporizador = setInterval(() => {
59. const novaLargura = *this*.larguraBase - rodada \* 10
60. *el*.style.width = `${novaLargura}px`
61. rodada++
62. if(rodada > 30) {
63. clearInterval(temporizador)
64. done()
65. }
66. }, 20);
67. },
68. Para realizarmos um reuso de código sem gerar repetições, podemos criar um método chamado animar que terá três parâmetros, el, done, negativo.
69. E a única coisa que será alterado será o sinal da nova largura. Para isso iremos utilizar um método ternário para calcular com o sinal de + ou -.
70. methods: {
71. animar(*el*, *done*, *negativo*){
72. let rodada = 1
74. const temporizador = setInterval(() => {
75. const novaLargura = *this*.larguraBase +
76. (*negativo* ? -rodada \* 10 : rodada \* 10)
77. *el*.style.width = `${novaLargura}px`
78. rodada++
79. if(rodada > 30) {
80. clearInterval(temporizador)
81. done()
82. }
83. }, 20);
84. },
85. beforeEnter(*el*){
86. *this*.larguraBase = 0
87. *el*.style.width = `${*this*.larguraBase}px`
88. },
89. enter(*el*, *done*){
90. *this*.animar(*el*, *done*, false)
91. },
92. afterEnter(*el*){
93. console.log('afterEnter')
94. },
95. enterCancelled(){
96. console.log('enterCancelled')
97. },
98. beforeLeave(*el*){
99. *this*.larguraBase = 300
100. *el*.style.width = `${*this*.larguraBase}px`
101. },
102. leave(*el*, *done*){
103. *this*.animar(*el*, *done*, true)
104. },
105. afterLeave(*el*){
106. console.log('afterLeave')
107. },
108. leaveCancelled(){
109. console.log('LeaveCancelled')
110. }
111. },