**Comunicação entre componentes**

**Aula 119 – Comunicação entre componentes**

1. Baixe o conteúdo desta aula para podermos realizar as tarefas.
2. Instale as dependências digitando : npm i
3. Vemos que a aplicação possui três componentes Usuário, UsuárioEditar e UsuárioInfo, nós iremos realizar a comunicação entre estes componentes.
4. No componente Usuario vamos criar um botão abaixo do primeiro parágrafo chamando a função click que irá receber a função ‘Alterar Nome’.
5. Vamos dentro de export Default utilizar a propriedade data() criando dentro dela a variável nome inicialmento com o valor ‘Pedro’.
6. Vamos ainda adicionar a propriedade methods criando o método ‘alterarNome’ dentro do método iremos passar o valor ‘Ana’ para a variável ‘nome’.
7. <template>
8. <div *class*="container">
9. <h1>Componente Usuário</h1>
10. <p>Esse é um componente muito legal!</p>
11. <button @*click*='alterarNome'>Alterar Nonme</button>
12. <hr>
13. <div *class*="componentes">
14. <app-usuario-info />
15. <app-usuario-editar />
16. </div>
17. </div>
18. </template>
19. <script>
20. import AppUsuarioInfo from './UsuarioInfo'
21. import AppUsuarioEditar from './UsuarioEditar'
22. export default {
23. components: { AppUsuarioInfo, AppUsuarioEditar },
24. data() {
25. return {
26. nome: 'Pedro'
27. }
28. },
29. methods: {
30. alterarNome(){
31. *this*.nome = 'Ana'
32. }
33. },
34. }
35. </script>
36. <style *scoped*>
37. *.container* {
38. background-color: #27363b;
39. color: #fff;
40. padding: 10px;
41. }
42. *.container* hr {
43. margin: 20px 10px;
44. }
45. *.componentes* {
46. display: flex;
47. }
48. *.componentes* > \* {
49. margin: 10px;
50. }
51. </style>
52. No componente UsuarioInfo iremos adicionar outro parágrafo e interpolar a variável nome que foi declarada no componente Usuário.
53. Porém para passarmos valores para outros componentes temos que definir uma propriedade chamada props onde iremos indicar quais são as variáveis que serão recebidas pelo componente passadas para o template.
54. <template>
55. <div *class*="componente">
56. <h2>As Informações de Usuário</h2>
57. <p>Vários detalhes...</p>
58. <p>Nome do Usuário: <strong>{{nome}}</strong></p>
59. </div>
60. </template>
61. <script>
62. export default {
63. props:[
64. 'nome'
65. ]
66. }
67. </script>
68. <style *scoped*>
69. *.componente* {
70. flex: 1;
71. background-color: #ec485f;
72. color: #fff;
73. }
74. </style>
75. Agora no componente Usuario vamos adicionar a variável nome no carregamento do componente no template.
76. <div *class*="componentes">
77. <app-usuario-info *nome*='Pedro' />
78. <app-usuario-editar />
79. </div>
80. A comunicação entre componentes ‘pai’ e ‘filho’ se dá através de props que são criadas para passarmos valores entre componentes.
81. No componente filho nós dizemos quais são as props que esperamos receber e no componente pai nós passamos essa propriedade.
82. Ainda no componente Usuário nós iremos utilizar a diretiva v-bind:nome para passarmos o valor da ‘variavel nome’ para o template, pois da forma que estava a variável era estática.
83. <div *class*="componentes">
84. <app-usuario-info *v-bind*:*nome*='nome' />
85. <app-usuario-editar />
86. </div>
87. Desta forma conseguimos também alterar o nome através do botão no template.

**Aula 123 – Usando props no componente filho**

1. No componente UsuarioInfo vamos criar um methodo chamado inverterNome retornando o nome de forma reversa. Para isso iremos chamar o nome utilizando o comando split para converter ele em um array de strings vamos utilizar também a função reverse do array, já que agora ele é um array, para inverter a ordem dos elementos e finalmente iremos utilizar a função join que irá juntar as strings do array em um único elemento.
2. Após isso vamos substituir a variável nome no template para o método que criamos inverterNome().
3. <template>
4. <div *class*="componente">
5. <h2>As Informações de Usuário</h2>
6. <p>Vários detalhes...</p>
7. <p>Nome do Usuário: <strong>{{inverterNome()}}</strong></p>
8. </div>
9. </template>
10. <script>
11. export default {
12. props:[
13. 'nome'
14. ],
15. methods: {
16. inverterNome(){
17. return *this*.nome.split('').reverse().join('')
18. }
19. },
20. }
21. </script>
22. <style *scoped*>
23. *.componente* {
24. flex: 1;
25. background-color: #ec485f;
26. color: #fff;
27. }
28. </style>
29. C

# Para corrigir um erro no console : de Unchecked runtime.lastError: The message port closed before a response was received' chrome issue – Tive que desabilitar a extensão “Norton Safe Web”

**Aula 124 – Valindando Props**

1. A validadção das prpos é importante para que seja possível verificar no console qual o erro que está ocorrendo. No componente UsuarioInfo iremos alterar props nome de um array para um json informando qual o tipo da propriedade, sendo que é possível utilizar mais de um tipo na propriedade.
2. export default {
3. props:{
4. nome: *String*
5. }
6. Temos a possibilidade também de alterar o props para um objeto, passando para ele mais de um atributo, inclusive de required que neste caso será exibido a mensagem de atributo requerido caso a props não seja passada.
7. export default {
8. props:{
9. nome:{
10. type: *String*,
11. required: true
12. }
13. Para vermos isso funcionando vamos apagar a props nome do template <UsuarioInfo> no componente Usuario.
14. <div *class*="componentes">
15. <*UsuarioInfo*  />
16. <*UsuarioEditar* />
17. </div>
18. Dessa forma será gerado o erro no console:
19. vue.runtime.esm.js?2b0e:587 [Vue warn]: Missing required prop: "nome"
20. found in
21. ---> <UsuarioInfo> at src/components/UsuarioInfo.vue
22. <AppUsuario> at src/components/Usuario.vue
23. <App> at src/App.vue
24. <Root>
25. Temos também a possibilidade de incluir a propriedade default no objeto declarado na props, isso para que caso não seja informado a props no template, será mostrado o objeto default.
26. props:{
27. nome:{
28. type: *String*,
29. //required: true
30. default: 'Anônimo'
31. }
32. Vamos omitir a props no template para vermos funcionar na prática.
33. <div *class*="componentes">
34. <*UsuarioInfo*  />
35. <*UsuarioEditar* />
36. </div>
37. 
38. Vemos que foi mostrado o nome do usuário como Anônimo.

**Aula 125 – Validando props 02**

1. Ainda no componente UsuarioInfo temos a possibilidade também de declarar uma function dentro do atributo default na variável nome da props.
2. Faremos um exemplo onde caso não seja fornecido a props nome para o componente filho, será retornado um array de 10 posições preenchidas com o zero juntados com uma vírgula entre eles.
3. props:{
4. nome:{
5. type: *String*,
6. //required: true
7. //   default: 'Anônimo'
8. default: function(){
9. return *Array*(10).fill(0).join(',')
10. }
11. }
12. // nome: String
13. }
14. Agora vamos comentar a declaração da variável nome do template UsuarioInfo do componente Usuario.
15. <div *class*="componentes">
16. <!-- <UsuarioInfo v-bind:nome="nome" /> -->
17. <*UsuarioInfo* />
18. <*UsuarioEditar* />
19. </div>
20. O resultado é:
21. 

**Aula 126 – Comunicação Indireta com Eventos personalizados**

1. A comunicação entre o componente pai com o filho se dá através das props. E a comunicação entre o filho e o pai se dá através de eventos personalizados.
2. Para visualizar melhor vamos inserir no componente Pai(Usuario) um novo parágrafo interpolando a variável nome.

<p>Esse é um componente muito legal!</p>

        <p>O nome é <strong>{{nome}}</strong></p>

1. Agora no componente filho, UsuarioInfo, nós vamos adicionar um novo botão para chamar o novo método reiniciarNome. Esse método irá voltar com o nome original.
2. methods: {
3. inverterNome(){
4. return *this*.nome.split('').reverse().join('')
5. },
6. reiniciarNome(){
7. return *this*.nome = 'Claudisnei'
8. }
9. },
10. Porém desta forma o nome será reiniciado somente no componente filho.
11. 

**Criando Evento personalizado –**

1. Para podermos alterar o nome no componente Pai, temos que modificar o método reiniciarNome no componente Filho emitindo um evento que iremos chamar de ‘nomeMudou’ juntamente com o novo nome que acabou de ser reiniciado.
2. methods: {
3. inverterNome(){
4. return *this*.nome.split('').reverse().join('')
5. },
6. reiniciarNome(){
7. *this*.nome = 'Claudisnei'
8. *this*.$emit('nomeMudou', *this*.nome)
9. }
10. },
11. Agora temos que ir no componente Pai e incluir uma nova diretiva v-on com o nome do evento que estamos esperando escutar.
12. Desta forma toda vez que o evento ‘nomeMudou’ for disparado, o componente Pai irá ouvir o evento e saberá que a variável nome foi alterada.
13. O valor que está sendo passado juntamente com o evento pode ser capturado utilizando o $event que pode ser um objeto com vários valores ou um valor propriamente dito.
14. <div *class*="componentes">
15. <*UsuarioInfo*
16. *v-bind:nome*="nome"
17. *v-on:nomeMudou*=" nome = $event"
18. />
19. <!-- <UsuarioInfo /> -->
20. <*UsuarioEditar* />
21. </div>
22. Agora quando clicamos no botão reinicar nome no componente filho UsuarioInfo, o nome é alterado no componente filho e também no componente Pai.
23. Temos a possibilidade de passar um objeto com o valor antigo e o valor novo no evento que estamos emitindo no componente filho.
24. Para isso criamos uma constante dentro do evento passando o valor antigo para ela, então após isso criamos o objeto dentro do evento com o valor antigo e o atual.
25. reiniciarNome(){
26. const antigo = *this*.nome
27. *this*.nome = 'Claudisnei'
28. *this*.$emit('nomeMudou', {
29. novo: *this*.nome,
30. antigo
31. })
32. }
33. No componente Pai, para acessarmos o objeto, basta utilizarmos o $event. e qual o parâmetro que se deseja acessar.
34. <*UsuarioInfo*
35. *v-bind:nome*="nome"
36. *v-on:nomeMudou*=" nome = $event.novo + '-' + $event.antigo"
37. />

**Aula 127 – Comunicação indireta com call-back**

1. No componente pai, Usuario, vamos criar uma nova função chamada reiniciar nome e essa função irá pegar o this.nome e setar como Claudisnei.
2. methods: {
3. alterarNome(){
4. *this*.nome = 'Ana'
5. },
6. reiniciarNome(){
7. *this*.nome = 'Claudisnei'
8. }
9. },
10. Agora podemos pegar essa função e passa-la como parâmetro no componente filho, UsuarioInfo.
11. Para isso criamos o atributo ‘reiniciarFn’ e passamos para ela a função reiniciarNome.
12. <*UsuarioInfo*
13. *v-bind:nome*="nome"
14. *v-on:nomeMudou*=" nome = $event.novo + '-' + $event.antigo"
15. *v-on:reiniciarFn*="reiniciarNome()"
16. />
17. Agora temos que ir no componente filho e criar a propriedade ‘reiniciarFn’ do tipo ‘Function’, essa nova props irá receber a nova função passada pelo componente pai.
18. props:{
19. nome:{
20. type: *String*,
21. //required: true
22. default: 'Anônimo'
23. //  default: function(){
24. //      return Array(10).fill(0).join(',')
25. //  }
26. },
27. reiniciarFn: *Function*
28. // nome: String
29. }
30. Para testar vamos criar outro botão no template para utilizarmos a função que foi passada como props pelo componente pai.
31. <template>
32. <div *class*="componente">
33. <h2>As Informações de Usuário</h2>
34. <p>Vários detalhes...</p>
35. <p>Nome do Usuário: <strong>{{inverterNome()}}</strong></p>
36. <button *v-on:click*="reiniciarNome()">Reiniciar Nome</button>
37. <button *v-on:click*="reiniciarFn()">Reiniciar Nome (CallBack)</button>
38. </div>
39. </template>

**Aula 129 – Comunicação entre Componentes Irmãos**

1. Iremos realizar comunicação entre os componentes Irmãos, UsuarioInfo e UsuarioEditar.
2. Vamos criar outra props no componente Pai, será a props idade.
3. data() {
4. return {
5. nome: 'Claudisnei',
6. idade: '49'
7. }
8. },
9. Vamos criar um novo parágrafo e interpolar a idade.
10. <p>Idade: <strong>{{idade}}</strong></p>
11. Iremos também passar a props idade para os componentes filhos.
12. <*UsuarioInfo*
13. *v-bind:nome*="nome"
14. *v-on:nomeMudou*=" nome = $event"
15. *v-bind:reiniciarFn*="reiniciarNome"
16. :*idade*="idade"
17. />
18. <*UsuarioEditar*
19. :*idade*="idade"
20. />
21. C
22. Agora temos que ir nos componentes filhos e declarar as props que está sendo enviada pelo componente Pai.
23. No componente UsuarioEditar:
24. xport default {
25. props:{
26. idade: *Number*
27. }
29. }
30. No componente UsuarioInfo:
31. export default {
32. props:{
33. nome:{
34. type: *String*,
35. //required: true
36. default: 'Anônimo'
37. //  default: function(){
38. //      return Array(10).fill(0).join(',')
39. //  }
40. },
41. reiniciarFn: *Function*,
42. // nome: String
43. idade: *Number*
44. }
45. Após declarar as props nos componentes filhos, vamos inserir nos templates dos componentes filhos um parágrafo, interpolando a variável idade. Veremos que a idade irá aparecer nos componentes filhos.
46. <p>Idade: <strong>{{idade}}</strong></p>
47. Para alterarmos a variável idade através dos componentes filhos, vamos criar no componente filho UsuarioEditar um método ‘alterarIdade’ onde iremos emitir um evento e um novo valor para a variável idade.
48. methods: {
49. alterarIdade(){
50. *this*.$emit('idadeMudou', 35)
51. }
52. },
53. inserir no componente UsuarioEditar, um botão com uma diretiv v-on:click passando para ele o método ‘alterarIdade’.
54. <button @*click*="alterarIdade">Alterar Idade</button>
55. C
56. Temos que ir no componente pai e nas tags dos componentes filhos temos que inserir uma diretiva v-on: com o nome do evento que está sendo emitido pelo componente filho, neste caso, ‘idadeMudou’. Desta forma quando a idade for alterada no componente filho, será emitido o evento com a nova idade e o componente Pai irá passar essa idade para todos os componentes.
57. <*UsuarioInfo*
58. *v-bind:nome*="nome"
59. *v-on:nomeMudou*=" nome = $event"
60. @*idadeMudou*=" idade = $event"
61. *v-bind:reiniciarFn*="reiniciarNome"
62. :*idade*="idade"
63. />
64. <*UsuarioEditar*
65. :*idade*="idade"
66. @*idadeMudou*=" idade = $event"
67. />

**Aula 130 – Usando Event Bus para comunicação entre componentes irmãos.**

1. Vamos criar um novo arquivo chamado barramento.js na pasta src onde iremos importar o Vue e exportar uma instância do Vue.
2. import Vue from 'Vue'
3. export default **new** Vue()
4. Agora vamos importar o arquivo barramento no componente UsuarioEditar e dentro do método ‘alterarIdade()’ nós vamos chamar a instância de barramento passando para ele o $emit com o nome do evento e valor da variável.
5. import barramento from '@/barramento'
6. export default {
7. props:{
8. idade: *Number*
9. },
10. methods: {
11. alterarIdade(){
12. *this*.idade = 35
13. //this.$emit('idadeMudou', this.idade )
14. barramento.$emit('idadeMudou', *this*.idade)
15. }
16. },
17. No componente UsuarioInfo nós iremos importar também o arquivo barramento utilizando o endereço absoluto.
18. Ainda neste componente, vamos utilizar o método de ciclo de vida created() onde iremos escutar o evento que foi emitido ‘idadeMudou’.
19. created() {
20. barramento.$on('idadeMudou', *idade* =>{
21. *this*.idade = *idade*
22. })
23. },
24. Vamos agora retirar a diretiva v-on:idadeMudou da tag do componente editar no componente pai, para vermos funcionando. Desta forma a idade será alterada no componente UsuarioInfo através do barramento.js.
25. <*UsuarioEditar*
26. :*idade*="idade"
28. />
29. C
30. Aula 132 – Adicionando métodos no Event Bus
31. Nesta aula iremos adicionar métodos no arquivo barramento.js para alterar a idade emitindo um evento e o valor da nova variável ‘idade’.
32. Vamos criar outro método que ira escutar o evento ‘idadeMudou’ que recebe um call-back.
33. export default **new** Vue({
34. methods: {
35. alterarIdade(*idade*){
36. *this*.$emit('idadeMudou', *idade*)
37. },
38. quandoIdadeMudar(*callback*){
39. *this*.$on('idadeMudou', *callback*)
40. }
41. },
42. })
43. Para vermos funcionando iremos retirar o .$emit do barramento no componente UsuarioEditar e vamos chamar o método alterarIdade passando a idade para ele.
44. methods: {
45. alterarIdade(){
46. *this*.idade = 35
47. //this.$emit('idadeMudou', this.idade )
48. // barramento.$emit('idadeMudou', this.idade)
49. barramento.alterarIdade(*this*.idade)
50. }
51. No componente UsuarioInfo nós iremos chamar o método ‘quandoIdadeMudar’ e passaremos a função call-back.

**Aula 133 – Desafio**

1. Neste desafio nós teremos um app que possui dois componentes filhos, ‘usuárioLista’ e ‘usuárioDetalhe’.
2. O componente ‘Pai’ possui um array com 5 registros.
3. export default {
4. name: "app",
5. components: { UsuarioLista, UsuarioDetalhe },
6. data() {
7. return {
8. usuarios: [
9. { id: 1, nome: 'Ana', idade: 31 },
10. { id: 2, nome: 'Carlos', idade: 22 },
11. { id: 3, nome: 'Daniel', idade: 40 },
12. { id: 4, nome: 'Ema', idade: 17 },
13. { id: 5, nome: 'Lia', idade: 28 }
14. ]
15. }
16. }
17. }
18. No componente componente filho ‘usuario-lista’ criamos uma diretiva v-bind passando o array.
19. Para podermos criar uma comunicação entre os componentes filhos, nós iremos criar um arquivo chamado barramento.js, este arquivo terá um método chamado setSelectUser que receberá por parâmetro o usuário selecionado, este método irá emitir um evento chamado ‘selectUser’ e o próprio usuário que foi recebido por parâmetro.
20. methods: {
21. setSelectUser(*UserSelect*){
22. *this*.$emit('selectUser', *UserSelect*)
23. },
24. Teremos também outro método chamado onSelectUser que irá escutar o evento ‘selectUser’ e irá passar uma função call-back, que foi recebida por parâmetro.
25. onSetSelectUser(*callback*){
26. *this*.$on('selectUser',*callback*)
27. }
28. No componente filho ‘usuárioLista’ recebemos o array usuário via props e importamos o arquivo barramento.
29. Temos um método chamado ‘setSelectUser’ que recebe via parâmetro o usuário selecionado via click.
30. Este método utiliza o método barramento.setSelectUser que recebe o usuário selecionado via click.
31. script>
32. import barramento from '@/barramento'
33. export default {
34. props: { usuarios: *Array* },
35. methods: {
36. setSelectUser(*selectUser*){
37. barramento.setSelectUser(*selectUser*)
38. }
39. },
40. }
41. </script>
42. No template do componente temos uma tabela com um v-for dentro do <tr> que percorre todos os registros do array usuários e para cada usuário é criado um <tr>.
43. Dentro do <tr> temos também uma diretiva v-on:click que recebe o método ‘setSelectUser’, este método irá receber por parâmetro o usuário da linha que foi clicada.
44. <template>
45. <div *class*="usuario-lista">
46. <table>
47. <thead>
48. <tr>
49. <th>#ID</th>
50. <th>Nome</th>
51. </tr>
52. </thead>
53. <tbody>
54. <tr v-for="usuario in usuarios" :*key*="usuario.id"
55. *v-on:click*="setSelectUser(usuario)">
56. <td>{{ usuario.id }}</td>
57. <td>{{ usuario.nome }}</td>
58. </tr>
59. </tbody>
60. </table>
61. </div>
62. </template>
63. Ao clicarmos na linha ou seja no <tr> o usuário selecionado é passado para método setSelectUser do arquivo barramento, desta forma o outro componente irmão poderá receber este usuário.
64. O componente ‘usuarioDetalhe’ possui uma variável chamada usuário que recebe na inicialização o valor de ‘null’
65. Este componente importa o arquivo barramento.
66. Criamos também o métoco created() que será executado toda vez que a tela for renderizada.
67. No método created() utilizaremos o método onSetSelectUser do arquivo barramento que irá passar o valor de usuário para a variável usuário.
68. <script>
69. import barramento from '@/barramento'
70. export default {
71. data() {
72. return {
73. usuario: null
74. }
75. },
76. created() {
77. barramento.onSetSelectUser(*usuario*=>{
78. *this*.usuario = *usuario*
79. })
80. },
81. }
82. </script>
83. Neste componente temos então o template que na inicialização verifica através da diretiva v-if se a variável usuário está vazia, neste caso será apresentado a mensagem ‘Usuário não selecionado’.
84. Através da diretiva v-else interpolamos os dados do array usuário e mostramos na tela, lembrando que o usuário foi recebido via click lá no componente irmão.
85. <template>
86. <div *class*="usuario-detalhe">
87. <h2 v-if="!usuario">Usuário não selecionado!</h2>
88. <div v-else>
89. <h2>ID: {{ usuario.id }}</h2>
90. <h2>Nome: {{ usuario.nome }}</h2>
91. <h2>Idade: {{ usuario.idade }}</h2>
92. </div>
93. </div>
94. </template>