**Rotas em uma aplicação VueJS**

1. Nesse capitulo iremos estudar sobre o Vue Ruter que é um plugin que irá controlar a navegação entre vários componentes.

**Aula 251 – Instalando o vue-router.**

1. vamos instalar a versão 3.0.2 do vue-router, para isso digite:
2. npm i --save vue-router@3.0.2 -E
3. serão instaladas todas as outras dependências.
4. Incluímos no arquivo index.html referências para o animate.css e fonte.
5. Criamos a pasta components e o componente inicio.vue
6. Criado a pasta usuário dentro de components e criado o componente Usuario.vue
7. Dentro da pasta src criado o arquivo router.js para configurarmos as rotas da aplicação.
8. No arquivo router.js
9. Importamos o Vue , importamos o vue-router e os dois componentes criados.
10. E registramos o plugin dentro do Vue.
11. E vamos exportar uma função construtora Router com objetos de configuração do vue-router.
12. Cada objeto que estiver dentro do array será responsável por mapear uma rota.
13. Configuramos o path: informando qual a rota que será aplicada e qual o componente será carregado.
14. import Vue from 'vue'
15. import Router from 'vue-router'
16. import Inicio from './components/Inicio'
17. import Usuario from './components/usuario/Usuario'
18. Vue.use(Router)
19. export default **new** Router({
20. routes: [{
21. path: '/',
22. component: Inicio
23. },{
24. path: '/usuario',
25. component: Usuario
26. }]
27. })
28. para funcionar devemos registrar o router no arquivo main.js importando e registrando dentro da instância do Vue.
29. import './style.css'
30. import Vue from 'vue'
31. import App from './App.vue'
32. import router from './router'
33. Vue.config.productionTip = false
34. **new** Vue({
35. router,
36. render: *h* => h(App),
37. }).$mount('#app')
38. Após realizar a configuração do vue-router temos que indicar onde os componentes serão exibido o conteúdo que será carregado pelo vue-router. Por isso vamos inserir no arquivo app.vue
39. A tag <router-view> e será neste ponto onde o vue-router irá carregar os componentes que foram importados no arquivo de rotas router.js.
40. <template>
41. <div *id*="app">
42. <h1>Rotas com VueRouter</h1>
43. <router-view />
44. </div>
45. </template>
46. C

**Aula 253 – Entendendo os modos de Rotas (hash vs history).**

1. Modo hash toda a requisição feita ao servidor passa obrigatoriamente pelo arquivo index.html e é onde é carregado o arquivo app.js.
2. Para mantermos o hash aparecendo na url do navegador, devemos fazer uma configuração no arquivo router.js informando o mode: ‘hash’.
3. export default **new** Router({
4. mode:'hash',
5. routes: [{
6. path: '/',
7. component: Inicio
8. },{
9. path: '/usuario',
10. component: Usuario
11. }]
12. })

Para retirar o hash da url basta utilizar o ‘history’ no mode, export default **new** Router({

    mode:'history',

    routes: [{

        path: '/',

        component: Inicio

    },{

        path: '/usuario',

        component: Usuario

    }]

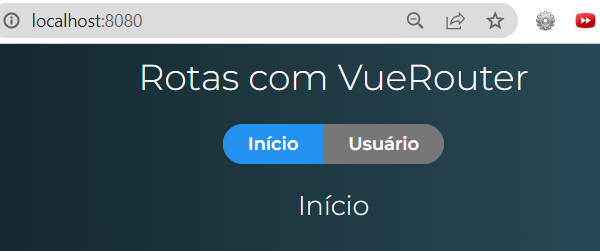
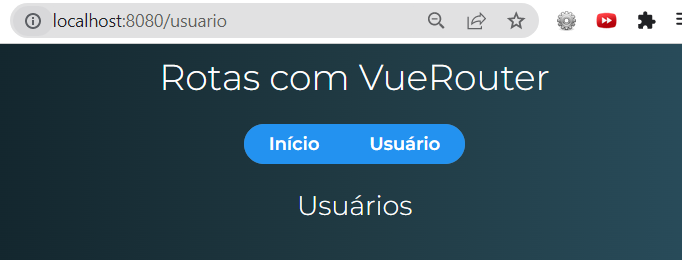
})

1. porém é necessário realizar configurações no servidor.
2. Para o modo history funcionar corretamente.

**Aula 254 – Navegando com Router links**

1. Iremos criar um menu para navegarmos entre os componentes sem ter que digitar a url.
2. Para isso vamos criar dentro de comonents uma nova pasta chamada template e dentro dela um novo arquivo chamado menu.vue.
3. Que terá a estrutura de um componente e em template uma tag nav e classe menu.
4. Dentro da tag nav teremos os links, porém, devemos utilizar o <router-link> utilizando a propriedade to: para informar os valores de path que estão configurados no arquivo router.js.
5. <template>
6. <nav *class*="menu">
7. <router-link *to*="/">Início</router-link>
8. <router-link *to*="/usuario">Usuário</router-link>
9. </nav>
10. </template>
11. Agora vamos importar o componente Menu.vue para o arquivo app.vue e registra-lo.
12. Vamos inserir o componente Menu após o h1 e antes do router-view.
13. Porém para ficar mais dinâmico, criei uma variável do tipo array no arquivo App.vue, chamada ‘itensMenu’ que possui como atributos ‘title: texto que irá aparecer no menu’ e ‘tag: caminho que foi configurado no arquivo router.js’.
14. <div *id*="app">
15. <h1>Rotas com VueRouter</h1>
16. <Menu :*itensMenu*="itensMenu" */*>
17. <router-view />
18. </div>
19. No arquivo App.js passo esse array para o componente Menu.
20. <div *id*="app">
21. <h1>Rotas com VueRouter</h1>
22. <Menu :*itensMenu*="itensMenu" */*>
23. <router-view />
24. </div>
25. no menu tenho um props do tipo array que irá receber o conteúdo do array.
26. export default{
27. props:{
28. itensMenu: {type: *Array*, required: true}
29. }
30. }
31. No componente Menu tenho também um v-for que irá percorrer os itensMenu e irá criar os roter-links para cada item.
32. <router-link v-for="item in itensMenu" :*key*="item.tag"
33. :*to*="item.tag"
34. >{{item.title}}</router-link>
35. C

**Aula 255 – Estilizando o link ativo do Menu**

1. Para aplicar os estilos que foram configurados nas classes css do arquivo style.css vamos envolver o <router-link> do componente Menu.vue, dentro de uma ul e li.
2. <ul>
3. <li v-for="item in itensMenu" :*key*="item.tag">
4. <router-link
5. :*to*="item.tag"
6. >{{item.title}}</router-link>
7. </li>
8. </ul>
9. Mas existe uma outra forma de alteramos a tag que envolve o <router-link>.
10. Para isso vamos utilizar o atributo tag dentro de <li> e passar para ela ‘li’, temos também que colocar o texto do <router-link> entre duas tag <a> </a>. O resultado final será o mesmo.
11. <nav *class*="menu">
12. <ul>
13. <router-link v-for="item in itensMenu" :*key*="item.tag"
14. :*to*="item.tag"
15. *tag*="li"
16. ><a>{{item.title}}</a>
17. </router-link>
18. </ul>
19. </nav>
20. 
21. Existe uma propriedade no router-link que define se o link está ativo ou não, essa propriedade é class-active. No arquivo style.css definimos uma classe chamada active que muda o estilo do link quando está ativo. Vamos então definir essa propriedade no router-link para aplicamos este estilo.
22. 
23. Porem existe um erro quando clicamos no outro link ‘usuario’, os dois ficam selecionados.
24. 
25. Para resolvermos esse erro basta utilizar a propriedade exact dentro do router-link, e desta forma ele irá aplicar a propriedade somente se o link for exatamento o que foi passado na propriedade to=.
26. <nav *class*="menu">
27. <ul>
28. <router-link v-for="item in itensMenu" :*key*="item.tag"
29. :*to*="item.tag"
30. *tag*="li"
31. *active-class*="active"
32. *exact*
33. ><a>{{item.title}}</a>
34. </router-link>
35. </ul>
36. 

**Aula 256 – Navegação via código (Navegação imperativa)**

1. Para fazer este tipo de navegação iremos utilizar código Javascript e não somente o <router-link>.
2. Para verificar isso vamos abrir o componente Usuario e incluir um botão para voltar na página anterior.
3. <template>
4. <div *class*="usuario">
5. <h2>Usuários</h2>
6. <hr>
7. <button *sucesso*>Voltar</button>
8. </div>
9. </template>
10. Porem estamos passando um atributo chamado ‘sucesso’para o botão esse atributo foi definido no arquivo style.css e está entre [] neste arquivo.
11. [*sucesso*] {
12. color: #FFF;
13. background-color: #47af55;
14. }
15. Desta forma aplicamos um estilo através de um atributo e não somente apartir de uma classe definida.
16. 
17. Agora vamos criar um método chamado irParaInicio() chamando a variável global $router. Vamos passar para a variável $router, através do método push, o path e ele irá navegar para o componente que possui o path associado.
18. export default{
19. methods:{
20. irParaInicio(){
21. *this*.$router.push('/')
22. }
23. }
24. }
25. Vamos inserir este método no evento @click do botão.
26. <button @*click*="irParaInicio" *sucesso*>Voltar</button>
27. Outra possibilidade também seria ao invés de passar ‘/’ para o método .push(), poderíamos passar um json com o path já definido.
28. methods:{
29. irParaInicio(){
30. // this.$router.push('/')
31. *this*.$router.push({ path: '/'})
32. }
33. }

**Aula 257 – Passando parâmetros para as Rotas.**

1. Para atribuirmos parâmetros em nossa url, devemos incluir no path do arquivo router.js os parâmetros utilizando ‘:’com o nome do paramentro.
2. export default **new** Router({
3. mode:'history',
4. routes: [{
5. path: '/',
6. component: Inicio
7. },{
8. path: '/usuario/:id',
9. component: Usuario
10. }]
11. })
12. Para nossa navegação continuar funcionando vamos subsituir na variável tag do array itensMenu o valor ‘/usuario’ por ‘/usuário/10’. Veremos que irá funcionar normalmente.
13. 

**Aula 258 – Lendo e usando parâmetros de rotas**

1. Estamos passando o parâmetro diretamente na variável tag do array. Para verificar o funcionamento de passagem de parâmetro vamos inserir no componente <Usuario />.
2. Uma tag <p> com um Strong com a palavra Código dentro de <p> iremos interpolar a variável id que em data(), recebe o parâmetro ‘id’ configurado no arquivo router.js.
3. <div *class*="usuario">
4. <h2>Usuários</h2>
5. <hr>
6. <p><strong>Código:</strong> {{id}}</p>
7. <button @*click*="irParaInicio" *sucesso*>Voltar</button>
8. </div>
9. Em data() teremos então o id recebendo o parâmetro.
10. data() {
11. return {
12. id: *this*.$route.params.id
13. }
14. },
15. Vamos alterar o array de itensMenu e incluir mais doi itens.
16. Quando clicamos no item do menu a variável id que está no template de <Usuario> para resolvermos isso iremos utilizar a propriedade watch:
17. Dentro de watch iremos monitorar a propriedade $route que possui dois parâmetros to e from, iremos setar a variável this.id com to.params.id isso fará com que o parâmetro id da url seja passado automaticamente para this.id.
18. watch:{
19. $route(*to*, *from*){
20. *this*.id = *to*.params.id
21. }
22. }

**Aula 260 – Parâmetros de rotas via PROPS**

1. Veremos a possibilidade de monitorar um parâmetro de uma rota de um componente utilizando props.
2. Vamos criar uma props chamada id no componente Usuario.
3. Sendo que o nome da props é igual ao nome do parâmetro da rota.
4. <script>
5. export default{
6. props: ['id'],
7. Agora temos que ir no arquivo router.js, na rota que possui o parâmetro chamado ‘id’ e incluir um atrbuto chamado props passando para ele o valor de true.
8. Significando que todos os parâmetros desta rota serão passados para o componente na forma de props.
9. xport default **new** Router({
10. mode:'history',
11. routes: [{
12. path: '/',
13. component: Inicio
14. },{
15. path: '/usuario/:id',
16. component: Usuario,
17. props: true
18. }]
19. })

**Aula 261 – Configurando rotas filhas – rotas aninhadas**

1. Para exemplificar o uso de rotas aninhadas, vamos criar mais alguns componentes.
2. Vamos criar o componente UsuarioLista.vue, nesse componente vamos inserir uma <ul> com uma lista de usuários.
3. <template>
4. <div *class*="usuario-lista">
5. <h3>Lista de Usuários</h3>
6. <ul *class*="lista">
7. <li *class*="item-lista">Usuário 1</li>
8. <li *class*="item-lista">Usuário 2</li>
9. <li *class*="item-lista">Usuário 3</li>
10. </ul>
11. </div>
12. </template>
13. <script>
14. export default{
16. }
17. </script>
18. <style>
19. </style>
20. Vamos criar outro componente chamado UsuarioDetalhe
21. Nesse componente vamos utilizar somente um h3
22. <template>
23. <div *class*="usuario-detalhe">
24. <h3>Usuário Detalhe</h3>
25. </div>
26. </template>
27. Outro componente será chamado UsarioEditar
28. <template>
29. <div *class*="usuario-editar">
30. <h3>Usuario Editar</h3>
31. </div>
32. </template>
33. Para criarmos as rotas aninhadas, precisamos importar para dentro de router.js os componentes que criamos.
34. Dentro da rota do componente usuário iremos inserir uma propriedade chamada children e nela iremos inserir um array passando as rotas aninhadas.
35. Vamos alterar a rota do usuário retirando o parâmetro :id, uma vez que ele estará em uma outra rota filha.
36. import Vue from 'vue'
37. import Router from 'vue-router'
38. import Inicio from './components/Inicio'
39. import Usuario from './components/usuario/Usuario'
40. import UsuarioDetalhe from './components/usuario/UsuarioDetalhe'
41. import UsuarioLista from './components/usuario/UsuarioLista'
42. import UsuarioEditar from './components/usuario/UsuarioEditar'
43. Vue.use(Router)
44. export default **new** Router({
45. mode:'history',
46. routes: [{
47. path: '/',
48. component: Inicio
49. },{
50. path: '/usuario',
51. component: Usuario,
52. props: true,
53. children: [
54. { path: '', component: UsuarioLista},
55. { path: ':id', component: UsuarioDetalhe, props: true},
56. { path: ':id/editar', component: UsuarioEditar, props: true}
57. ]
58. }]
59. })
60. Agora vamos realizar algumas alterações no componente Usuario.
61. Vamos retirar template o parágrafo que contem a variável 'id' e a props também ‘id’.
62. <template>
63. <div *class*="usuario">
64. <h2>Usuários</h2>
65. <hr>
66. <button @*click*="irParaInicio" *sucesso*>Voltar</button>
67. </div>
68. </template>
69. <script>
70. export default{
71. methods:{
72. irParaInicio(){
73. // this.$router.push('/')
74. *this*.$router.push({ path: '/'})
75. }
76. },
78. }
79. </script>
80. <style>
81. </style>
82. e passar para o componente UsuarioDetalhe.
83. <template>
84. <div *class*="usuario-detalhe">
85. <h3>Usuário Detalhe</h3>
86. <p><strong>Código:</strong> {{id}} </p>
88. </div>
89. </template>
90. <script>
91. export default{
92. props: ['id'],
93. }
94. </script>
95. <style>
96. Lembrando que no arquivo router.js foi definido na rota children de Usuario uma rota com o componente UsuarioDetalhe e nesta rota há o parâmetro :id que será passado para o componente como props.
97. Mas para podermos exibir os componentes que estão aninhados no componente Usuario, devemos incluir no template do componente Usuario uma tag <router-view> onde serão renderizados os componentes que estão nas rotas ‘filhas’ ou aninhadas.
98. <template>
99. <div *class*="usuario">
100. <h2>Usuários</h2>
101. <hr>
102. <router-view />
103. <button @*click*="irParaInicio" *sucesso*>Voltar</button>
104. </div>
105. </template>
106. Como podemos ver nas rotas aninhas, quando o path for ‘ ‘ o componente que será carregado será UsuarioLista.
107. Dentro deste componente iremos substiruir as tags <li> por <router-link> e nesses router-links iremos incluir no parâmetro to= o path que será carregado na url, neste caso será por exemplo ‘/usuário/1’ assim será carregado o componente UsuarioDetalhe pois foi passado um parâmetro na url. Desta forma ao clicarmos sobre o link, o componente UsuarioDetalhe será carregado automaticamente.
108. UsuarioLista:
109. <template>
110. <div *class*="usuario-lista">
111. <h3>Lista de Usuários</h3>
112. <ul *class*="lista">
113. <router-link
114. *to*="/usuario/1"
115. *tag*="li"
116. *active-class*="active"
117. *exact*><a>Usuario 1</a></router-link>
118. <router-link
119. *to*="/usuario/2"
120. *tag*="li"
121. *active-class*="active"
122. *exact*><a>Usuario 2</a></router-link>
123. <router-link
124. *to*="/usuario/3"
125. *tag*="li"
126. *active-class*="active"
127. *exact*><a>Usuario 3</a></router-link>
128. </ul>
129. </div>
130. </template>
131. <script>
132. export default{
134. }
135. </script>
136. <style>
137. </style>
138. Aula 262 – Navegando para Rotas Aninhadas
139. Já foi feito esta implementação na aula anterior.

**Aula 263 – Tornando o router-link mais dinâmico**

1. Vamos criar um botão para sair da tela de detalhe para a tela editar.
2. Porém ao invés de criarmos um botão nós iremos utilizar o <router-link> utilizando a propriedade tag dele.
3. Iremos realizar um bind da propriedade to= para que a alteração do path seja dinâmico.
4. Dentro de to= temos a referência da props ‘id’ que está sendo passada para o componente via router.js.
5. <router-link
6. *tag*="button"
7. *primario*
8. :*to*="`/usuario/${id}/editar`" >
9. Editar
10. </router-link>
11. Porém se quisermos podemos também utilizar a variável global $route pegando dela o parâmetro desejado neste caso ‘id’, irá funcionar da mesma forma.
12. <router-link
13. *tag*="button"
14. *primario*
15. :*to*="`/usuario/${$route.params.id}/editar`" >
16. Editar
17. </router-link>

**Aula 264 – Criando links com Rotas nomeadas**

1. Iremos mostrar que é possível navegar através de um atributo chamado nome e não somente apartir de path.
2. No arquivo router.js vamos então dentro da rota aninhada do componente UsuarioEditar, inserir a propriedade name: ‘editarUsuario’
3. children: [
4. { path: '', component: UsuarioLista},
5. { path: ':id', component: UsuarioDetalhe, props: true},
6. { path: ':id/editar', component: UsuarioEditar,
7. props: true, name: 'editarUsuario'}
8. ]
9. No componente UsuarioDetalhe vamos alterar o atributo :to do <router-link> incluindo a propriedade name: ‘editarUsuario’ e params: { id: $route.params.id}. desta forma o link para o componente UsuarioDetalhe será através do nome dado na rota, passando também o parâmetro id na rota. Para passar o parâmetro podemos ao invés de utilizar o $route.params.id, podemos utilizar somente a props que já está sendo enviada na própria rota.
10. <router-link
11. *tag*="button"
12. *primario*
13. :*to*="{name: 'editarUsuario', params: { id } }" >
14. Editar
15. </router-link>

**Aula 265 – Usando parâmetros da Query**

1. Query sâo os parâmetros que são passados após um sinal de interrogação apartir de uma requisição.
2. Para exemplificar vamos no componente UsuarioDetalhe e dentro do atributo :to do <router-link> vamos incluir a propriedade query: dentro de query podemos passar um json com os parâmetros que queremos enviar via requisição get.
3. Neste caso iremos inserir query: { completo: true, língua: ‘pt-br’}.
4. <router-link
5. *tag*="button"
6. *primario*
7. :*to*="{name: 'editarUsuario', params: { id },
8. query: { completo: true, lingua: 'pt-br'} }" >
9. Editar
10. </router-link>
11. Esses parâmetros farão parte da url do componente UsuarioEditar quando clicarmos no <router-link>
12. No componente UsuarioEditar podemos pegar os parâmetros que foram passado na url através de $route.query. e o nome do parâmetro passado.
13. Vamos inserir no componente UsuarioDetalhe mais dois parágrofos e vamos pegar os parâmetros através de query.
14. <template>
15. <div *class*="usuario-editar">
16. <h3>Usuario Editar</h3>
17. <hr>
18. <p><strong>Editando usuário:</strong> {{id}} </p>
19. <p><strong>Completo:</strong> {{ $route.query.completo ? 'Sim' : 'Não'}} </p>
20. <p><strong>Lingua:</strong> {{ $route.query.lingua }} </p>
21. </div>
22. </template>
23. 

**Aula 266 – Múltiplos router-views – Router views nomeados**

1. No componente App.vue vamos apagar <Menu> e no lugar iremos colocar um <router-view name: ‘Menu’ />
2. Teremos assim dois router-view um nomeado e outro sem nome.
3. <template>
4. <div *id*="app">
5. <h1>Rotas com VueRouter</h1>
6. <!-- <Menu :itensMenu="itensMenu" /> -->
7. <router-view *name*="Menu" />
8. <router-view />
9. </div>
10. </template>
11. Agora temos que ir no router.js, vamos comentar o atributo component do path name:’Inicio’. E no lugar vamos incluir componentes: {} passando em que router-view será renderizado cada componente.
12. routes: [{
13. name: 'inicio',
14. path: '/',
15. // component: Inicio
16. components: {
17. default: Inicio,
18. menu: Menu
19. }
20. },{
21. Temos que importar o componente Menu para o arquivo router.js.
22. No arquivo App.vue vamos retirar a importação do componente Menu uma vez que ele será passado via <router-view>.
23. Vamos criar um componente chamado MenuAlt que será o componente Menu porém em inglês.
24. <template>
25. <nav *class*="menu">
26. <ul>
27. <!-- <router-link v-for="item in itensMenu" :key="item.tag"
28. :to="item.tag"
29. tag="li"
30. active-class="active"
31. exact
32. ><a>{{item.title}}</a>
33. </router-link> -->
34. <router-link
35. :*to*="{ name: 'inicio'}"
36. *tag*="li"
37. *active-class*="active"
38. *exact*
39. ><a>Home</a>
40. </router-link>
41. <router-link
42. :*to*="'/usuario'"
43. *tag*="li"
44. *active-class*="active"
45. *exact*
46. ><a>User</a>
47. </router-link>
48. </ul>
49. </nav>
50. </template>
51. <script>
52. export default{
53. props:{
54. itensMenu: {type: *Array*, required: true}
55. }
56. }
57. </script>
58. <style>
59. </style>
60. No arquivo router.js vamos alterar o path: ‘/usuario’ comentando o component Usuario e inserindo um component default: Usuario um menu: MenuAlt e um menuInferior: MenuAlt
61. path: '/usuario',
62. // component: Usuario,
63. components:{
64. default: Usuario,
65. menu: MenuAlt,
66. menuInferior: MenuAlt
67. },

**Aula 267 – Redirecionamento**

1. Vamos mostrar como fazer com que ao entrar em uma url o browser redirecione para outra url automaticamente.
2. Para isso vamos no arquivo router.js
3. E no final do array de rotas vamos adicionar um path.
4. },{
5. path: '/redirecionar',
6. redirect: '/usuario'
7. }]
8. Desta forma quando digitarmos /redirecionar na url, automaticamente a página será direcionada para /usuário.

**Aula 268 – Configurando rota Pega Tudo**

1. Podemos configurar nossas rotas para que ao digitarmos uma rota qualquer que não seja suportada, a página seja encaminhada para uma rota padrão.
2. Basta configuramos outra rota conforme a aula anterior, porém no path: iremos passar ‘\*’ e em redirect passamos por exemplo a rota raiz ‘/’.
3. ,{
4. path: '\*',
5. redirect: '/'
6. }]

**Aula 269 – Animando Transições de rotas**

1. Vamos mostrar como realizar animações entre rotas.
2. No arquivo App.vue vamos inserir o <router-view> não nomeado entre duas tags <transition>
3. <transition>
4. <router-view />
5. </transition>
6. Como realizamos a configuração do plugin Animate.css na página index.html.
7. Podemos descobrir quais as animações em <https://github.com/animate-css/animate.css?files=1>.
8. Na tag transition vamos inserir enter-active-class=”animated rubberBand” que é a classe que será executada na entrada do componente.
9. Vamos inserir também a classe ‘leave-active-class=”animated rollOut” essa classe será executada na saída do componente.
10. Vemos que após “animated” temos a animação que será executada.
11. <transition *mode*="out-in"
12. *enter-active-class*="animated rubberBand"
13. *leave-active-class*="animated rollOut">
14. <router-view />
15. </transition>

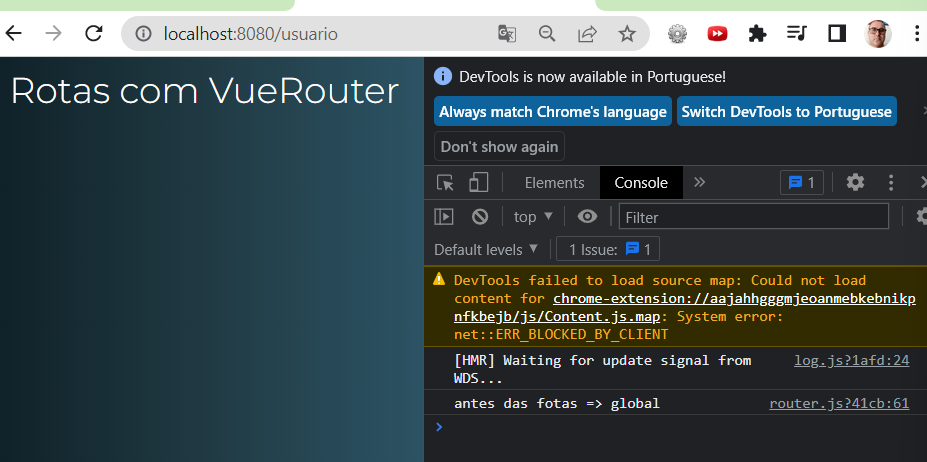
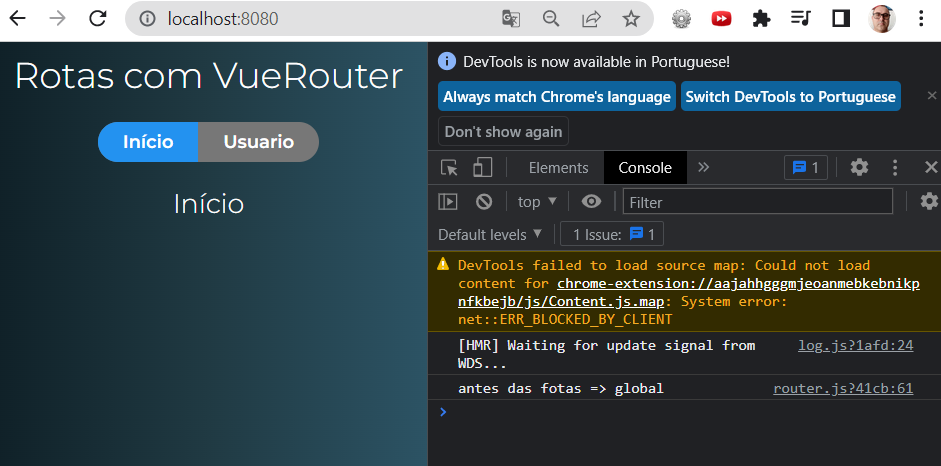
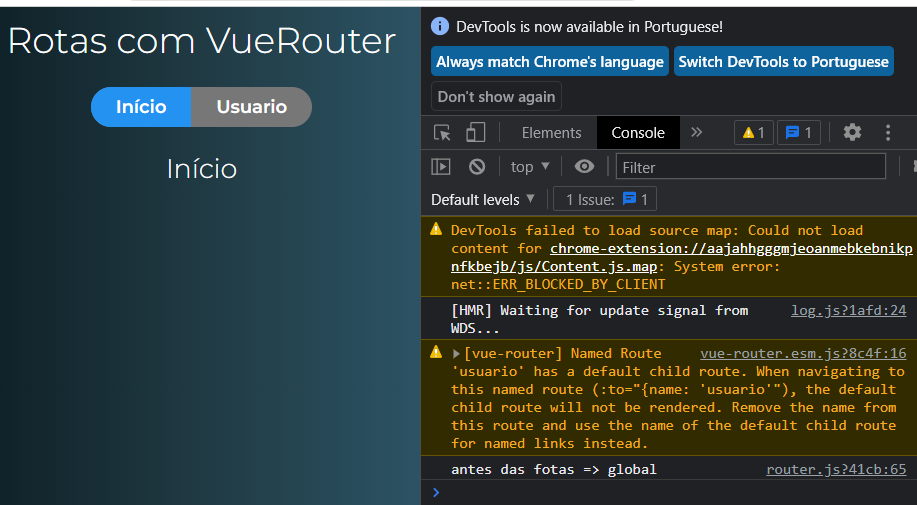
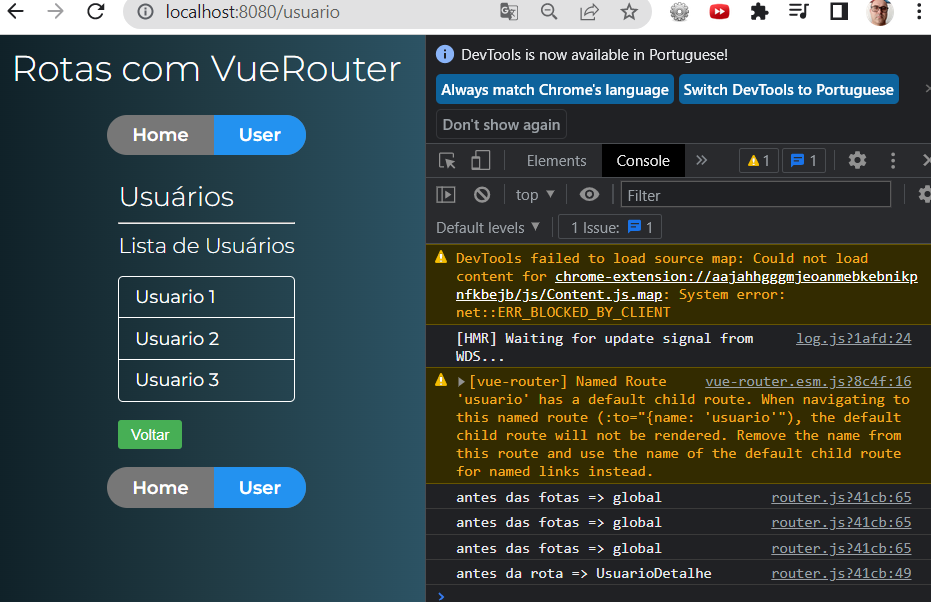
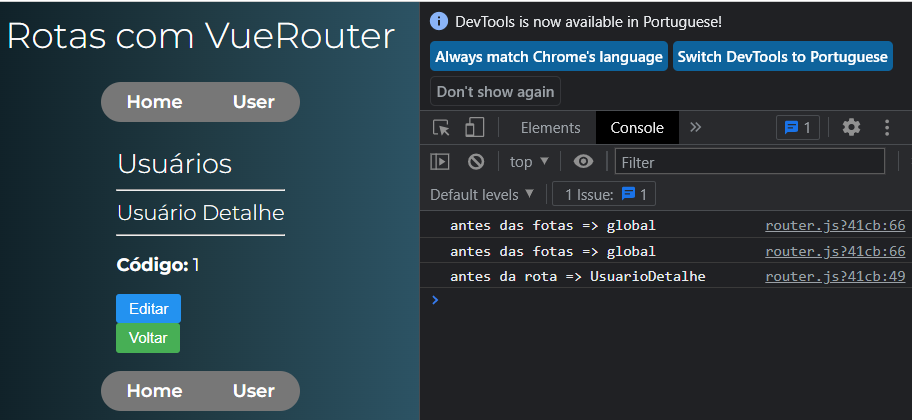
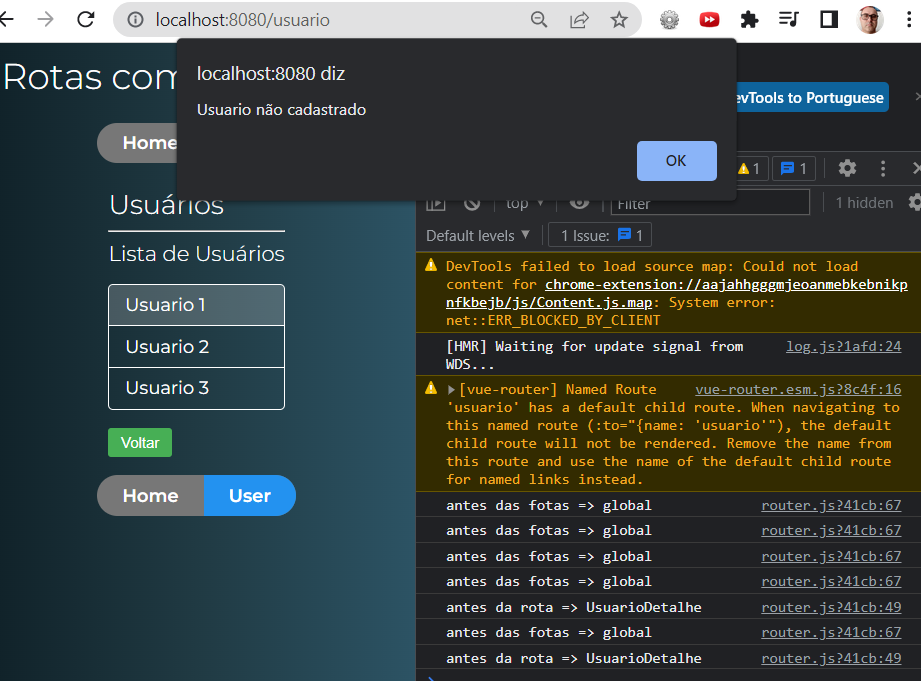
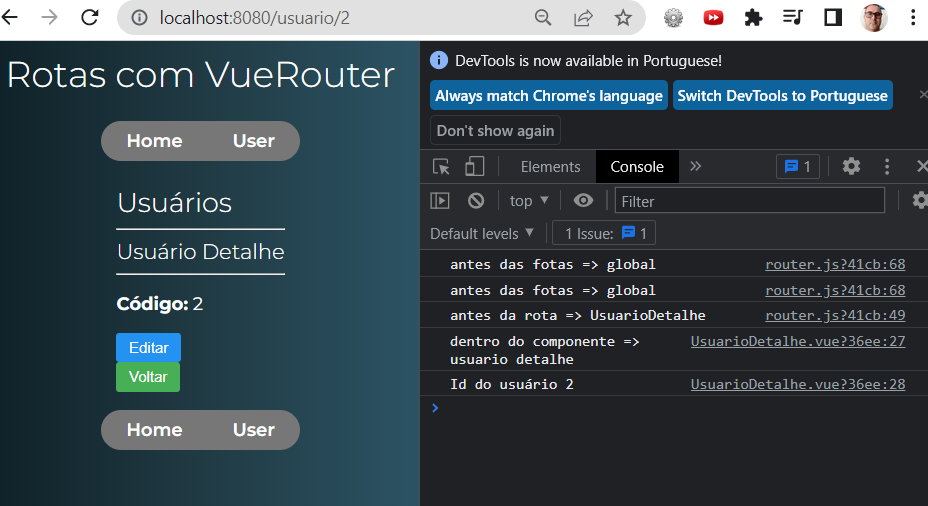
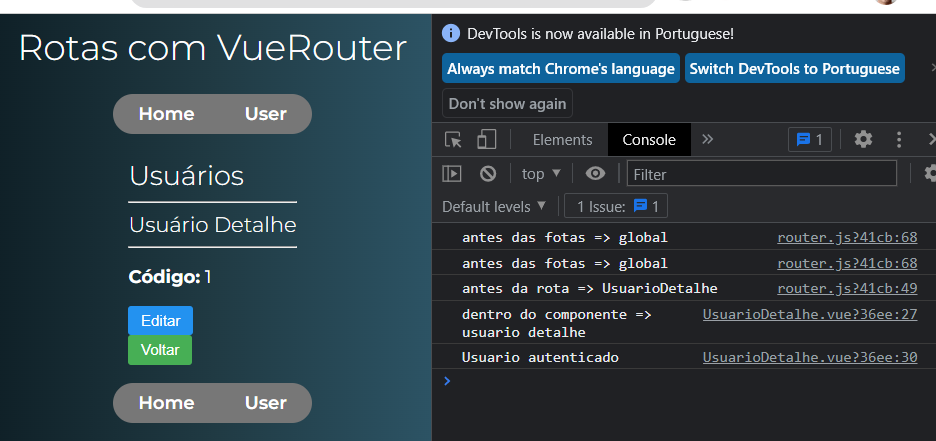
**Aula 270 – Passando Fragmento hash**

1. No componente UsuarioEditar vamos inserir uma div com um h3=Curso Vue.
2. <div *class*="usuario-editar">
3. <h3>Usuario Editar</h3>
4. <hr>
5. <p><strong>Editando usuário:</strong> {{id}} </p>
6. <p><strong>Completo:</strong> {{ $route.query.completo ? 'Sim' : 'Não'}} </p>
7. <p><strong>Lingua:</strong> {{ $route.query.lingua }} </p>
8. <div *id*="rodape">
9. </div>
10. </div>
11. Vamos inserir um estilo no componente para forçar que o rodapé fique na base da tela.
12. <style>
13. *#rodape*{
14. margin-top: 1000px;
15. }
16. </style>
17. Se digitarmos no final da url /#/rodape, veremos que a que o rodapé será selecionado automaticamente.
18. Para inserirmos o hash (‘#’) devemos abrir o componente usuário detalhe e no router-link devemos incluir o atributo hash no parâmetro :to=.
19. <router-link
20. *tag*="button"
21. *primario*
22. :*to*="{  name: 'editarUsuario',
23. params: { id },
24. query: { completo: true, lingua: 'pt-br'},
25. hash: '#rodape' }" >
26. Editar
27. </router-link>
28. Mas para o hash funcionar, devemos implentar uma função.

**Aula 271 – Controlando comportamento de rolagem (scroll)**

1. No arquivo router.js iremos inserir logo após o mode, a função scrollBehavior(){}
2. scrollBehavior(){
3. return { x: 0, y: 1000}
4. },
5. Porém esta não é a forma correta.
6. A forma correta é verificar se há algum hash e se existir selecionar o hash.
7. scrollBehavior( *to* ){
8. //return { x: 0, y: 1000}
9. if(*to*.hash){
10. return {
11. selector: *to*.hash
12. }
13. }
14. Podemos ainda utilizar um parâmetro chamado savedPosition para retornar a página para uma posição específica, ou voltar para 0 caso não tenha nenhuma posição salva.
15. scrollBehavior( *to*, *from*, *savedPosition* ){
16. //return { x: 0, y: 1000}
17. if(*savedPosition*){
18. return *savedPosition*
19. }else if(*to*.hash){
20. return {
21. selector: *to*.hash
22. }
23. }else {
24. return { x:0, y:0}
25. }
26. },

**Aula 272 – Protegendo rotas**

1. O vue-router disponibiliza interceptors detectar o momento que iremos entrar ou sair de uma rota.
2. Vamos inserir um botão no componente UsuarioEditar
3. Existem três formas para interceptar a navegação
4. A primeira maneira de realizar o controle é alterando o router.js defindo o router como uma constante e exportando ele no final, isso para podermos declarar de forma global uma função de ciclo de vida beforeEach().
5. Ou seja antes de cada navegação a função será chamada.
6. Dentro da função para realizar um teste vamos inserir um console.log e veremos que antes de toda rota a mensagem será exibida.
7. Porém temos que inserir next() pois se não a rota não será completada.
8. const router = **new** Router({
9. mode:'history',
10. scrollBehavior( *to*, *from*, *savedPosition* ){
11. //return { x: 0, y: 1000}
12. if(*savedPosition*){
13. return *savedPosition*
14. }else if(*to*.hash){
15. return {
16. selector: *to*.hash
17. }
18. }else {
19. return { x:0, y:0}
20. }
21. },
22. routes: [{
23. name: 'inicio',
24. path: '/',
25. // component: Inicio
26. components: {
27. default: Inicio,
28. menu: Menu
29. }
30. },{
31. path: '/usuario',
32. // component: Usuario,
33. components:{
34. default: Usuario,
35. menu: MenuAlt,
36. menuInferior: MenuAlt
37. },
38. props: true,
39. children: [
40. { path: '', component: UsuarioLista},
41. { path: ':id', component: UsuarioDetalhe, props: true},
42. { path: ':id/editar', component: UsuarioEditar,
43. props: true, name: 'editarUsuario'}
44. ]
45. },{
46. path: '/redirecionar',
47. redirect: '/usuario'
48. },{
49. path: '\*',
50. redirect: '/'
51. }]
52. })
53. router.beforeEach((*to*, *from*, *next*) => {
54. console.log('antes das fotas => global')
56. })
57. export default router
58. 
59. Vemos que as rotas não foram executadas, porém, quando incluímos next() a rota principal é executada.
60. router.beforeEach((*to*, *from*, *next*) => {
61. console.log('antes das fotas => global')
62. next()
63. })
64. 
65. Vamos definir que a rota deverá ser ‘/usuario’
66. Para isso iremos inserir a rota dentro do parênteses de next() e verificar através de to.path se na url ‘/usuario’ já foi setado, se não foi, a única rota permitida será ‘/usuario’.
67. router.beforeEach((*to*, *from*, *next*) => {
68. console.log('antes das fotas => global')
69. if(*to*.path !== '/usuario'){
70. next('/usuario')
71. }else{
72. next()
73. }
75. })
76. 
77. Desta forma ‘travamos’ a navegação e somente a rota ‘/usuario’ será permitida.
78. Vamos comentar o código dentro de beforeEach() para podermos testar as outras possibilidades.
79. router.beforeEach((*to*, *from*, *next*) => {
80. console.log('antes das fotas => global')
81. // if(to.path !== '/usuario'){
82. //     next('/usuario')
83. // }else{
84. next()
85. // }
87. })
88. Outra possibilidade é incluirmos o atributo beforeEnter: dentro de uma rota específica, passando os mesmos parâmetrod de beforeEach.
89. Vamos inserir esse atributo dentro da rota aninhada do componente UsuarioDetalhe exibindo a mensagem ‘antes da rota => usuarioDetalhe’.
90. children: [
91. { path: '', component: UsuarioLista},
92. { path: ':id', component: UsuarioDetalhe, props: true,
93. beforeEnter: (*to*,*from*,*next*)=>{
94. console.log('antes da rota => UsuarioDetalhe')
95. }},
96. { path: ':id/editar', component: UsuarioEditar,
97. props: true, name: 'editarUsuario'}
98. ]
99. Veremos que todas as rotas serão permitidas porém não conseguiremos navegar para a rota de usuário detalhe.
100. 
101. 
102. Vemos que a rota UsuarioDetalhe foi chamada, porém não navegou para ela, somente a mensagem foi mostrada.
103. Agora vamos inserir next() dentro de beforeEnter:
104. { path: ':id', component: UsuarioDetalhe, props: true,
105. beforeEnter: (*to*,*from*,*next*)=>{
106. console.log('antes da rota => UsuarioDetalhe')
107. next()
108. }},
109. Vemos que desta forma a rota foi completada para usuárioDetalhe.
110. 
111. Podemos além de enviar mensagem para o console, gerar um alert na tela.
112. 
113. Outra maneira também de protegermos nossas rotas é utilizar um método de ciclo de vida dentro de export default() do componente.
114. O método utilizado será o beforeRouteEnter() que possui os mesmos parâmetros de beforeEach() e beforeEnter:
115. Então dentro do componente UsuarioDetalhe, vamos inserir este método.
116. export default{
117. props: ['id'],
118. beforeRouteEnter (*to*, *from*, *next*) {
119. console.log('dentro do componente => usuario detalhe')
120. }
121. }
122. Veremos que agora não será possível entrar na rota do usuário detalhe, a mensagem no console é exibida porém a rota não é completada.
123. 
124. Caso seja necessário podemos acessar os dados do componente dentro de next() pois quando a rota é completada, dentro de next() podemos acessar a nova instância do componente.
125. export default{
126. props: ['id'],
127. beforeRouteEnter (*to*, *from*, *next*) {
128. console.log('dentro do componente => usuario detalhe')
129. next(*vm* =>{ console.log(`Id do usuário ${*vm*.id}`)})
130. }
131. }
132. 
133. Podemos por exemplo criar um variável de autenticação e caso ela seja false a rota não será completada .
134. export default{
135. props: ['id'],
136. beforeRouteEnter (*to*, *from*, *next*) {
137. console.log('dentro do componente => usuario detalhe')
138. // next(vm =>{ console.log(`Id do usuário ${vm.id}`)})
139. const autenticado = true
140. autenticado ? next(console.log('Usuario autenticado')) : next(false)
141. }
142. }
143. 
144. Caso não esteja autenticado.
145. export default{
146. props: ['id'],
147. beforeRouteEnter (*to*, *from*, *next*) {
148. console.log('dentro do componente => usuario detalhe')
149. // next(vm =>{ console.log(`Id do usuário ${vm.id}`)})
150. const autenticado = false
151. autenticado ? next(console.log('Usuario autenticado')) : next(false, console.log('Usuario não autenticado'))
152. }
153. }
154. 