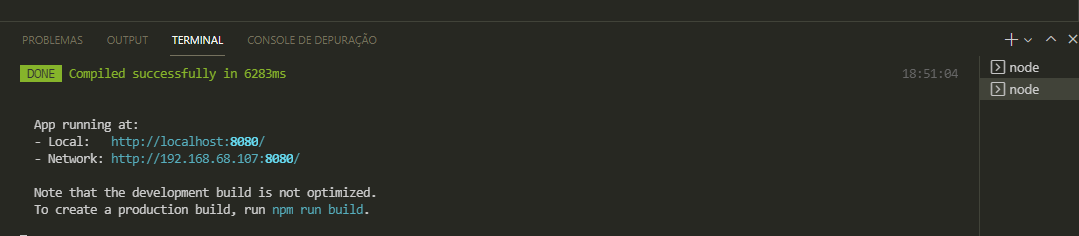
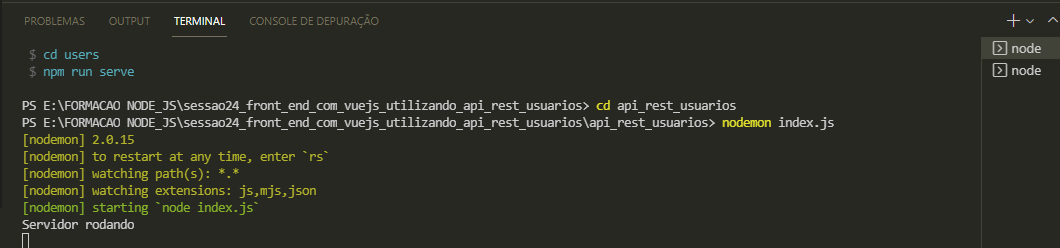
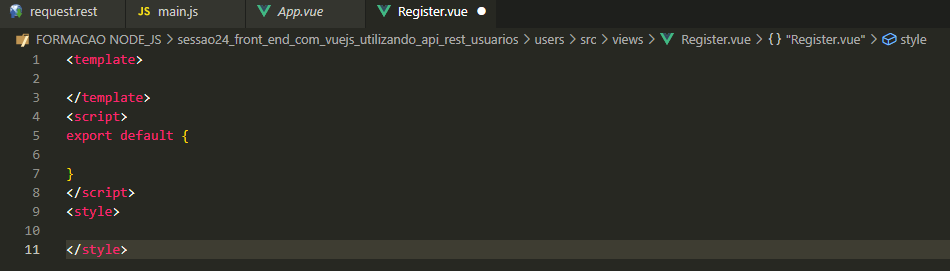
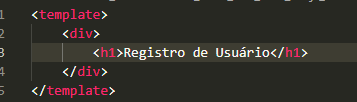
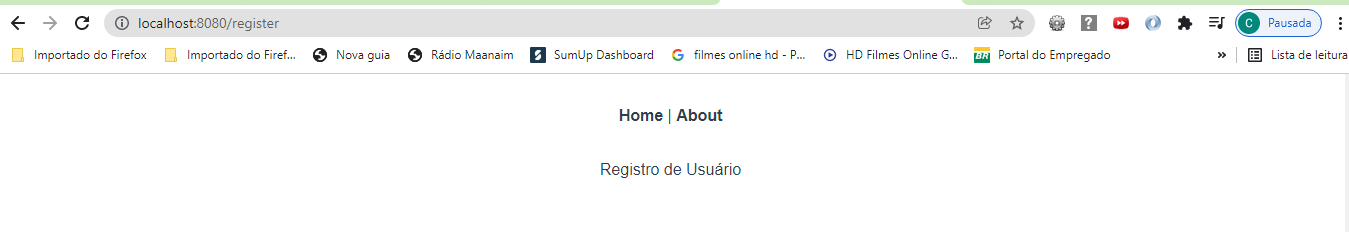
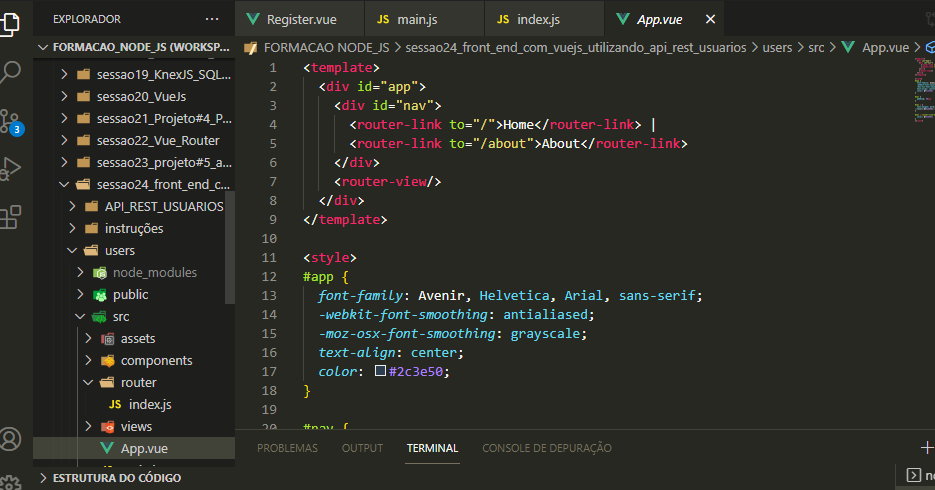
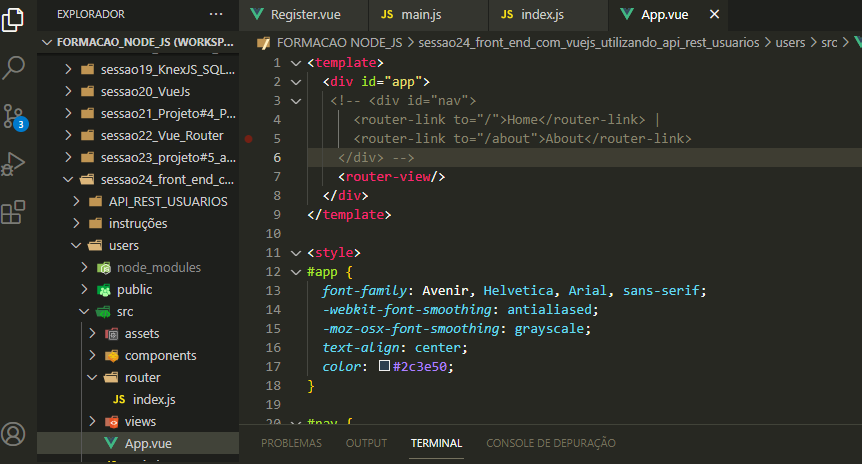
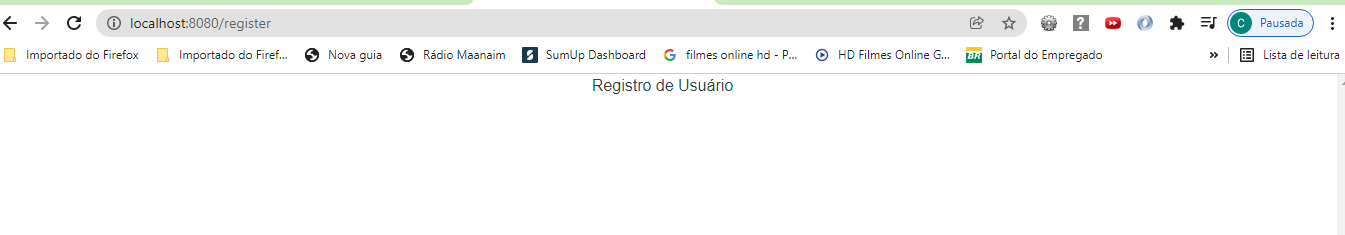
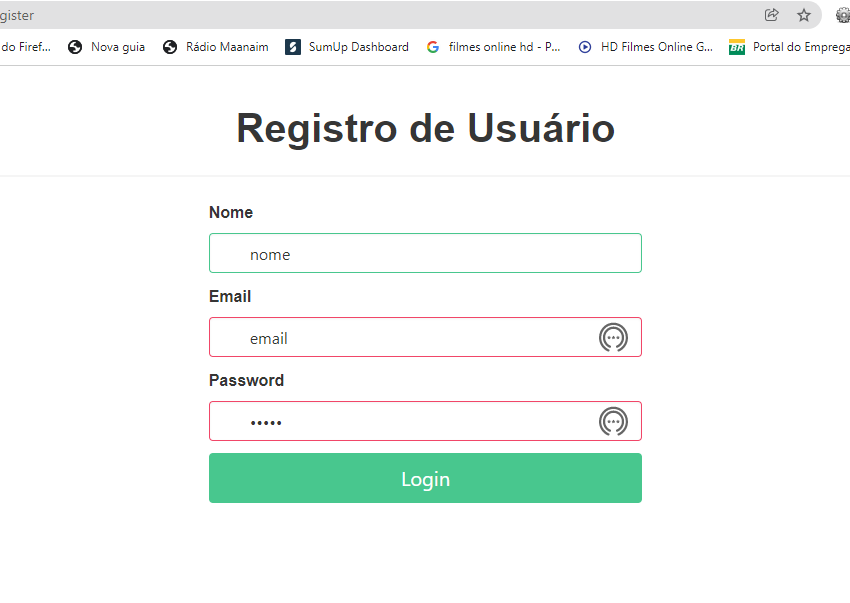
**SISTEMA DE GESTÃO DE USUÁRIOS COM FRONTEND VUEJS**

1. Copie a pasta API\_RES\_USUARIOS do projeto anterior para dentro da pasta deste novo projeto
2. Vamos primeiramente criar um projeto Vue com o router chamado usuer digite:
3. Vue create users.
4. Selecione a opção ‘Manually select features’ para selecionar o que será adicionado à aplicação vue,
5. Navegue até Router e digite a barra de espaço para seleciona-lo.
6. Tecle enter
7. Tecle enter
8. A próxima tela pergunta se vc quer instalar o history mode. O history mode cria rotas mais amigáveis, porém quando for fazer o deploy será necessário alterar configurações do apache dependendo do servidor.
9. Mas vamos utilizar o history mode assim mesmo.
10. Digite Y
11. Pressione Enter na primeira opção,
12. Pressione Enter na primeira opção
13. Pressione Enter na primeira opção
14. Digite N e tecle Enter
15. Mantenha o servidor rodando na pasta da api.
16. Abra outro terminal e instale o Bulma como framework css na pasta users.
17. Digite npm install bulma
18. Abra o arquivo main.js na pasta src e importe o bulma
19. 
20. Na pasta users instale também o axios para podermos realizar as consultas na nossa api.
21. Npm install axios
22. Vamos agorar rodar o projeto vue dentro da pasta user, digite npm run serve.
23. teremos a API rodando dentro de um terminal e o projeto vue rodando em outro.
24. Terminal com o servidor do vue rodando
25. 
26. Terminal com a API rodando
27. 

**Criando view Register**

1. Na pasta views crie um arquivo chamado Register.vue que será a página de cadastro.
2. Vamos começando definindo a configuração básica de uma página vue
3. 
4. Dentro do template crie uma div
5. Cire um h1 com a mensagem Registro de usuário
6. 
7. Vamos abrir o arquivo index.js na pasta router
8. Vamos importar o componente Register.vue e criar uma nova rota. Com o path ‘/register’ o name ‘Register’ e componente Register.
9. Agora para testar basta ir na página e digitar a url register.
10. 
11. Como podemos ver está aparencendo o link Home e About na página,
12. Abra o arquivo App.vue e podemos ver que eles estão fora do <router-view> eles estão dentro de um <router-link> por isso estão aparencendo como um menu.
13. 
14. Vamos comentar então os <router-link>.
15. 
16. A página ficará da seguinte forma.
17. 
18. Vamos desenvolver um pequeno formulário no componente Register.vue
19. Vamos na página do bulma para para copiar os códigos dos inputs que vamos utilizar.
20. <template>
21. <div class="content is-medium">
22. <h1 style="padding-top:40px">Registro de Usuário</h1>
23. <hr>
24. <div class="columns is-centered">
25. <div class="column is-one-third">
26. <div class="field">
27. <label class="label" style="text-align:left">Nome</label>
28. <div class="control has-icons-left has-icons-right">
29. <input class="input is-success" type="text" name="name" placeholder="Nome do usuário" value="">
30. <span class="icon is-small is-left">
31. <i class="fas fa-user"></i>
32. </span>
33. <span class="icon is-small is-right">
34. <i class="fas fa-check"></i>
35. </span>
36. </div>
37. </div>
38. <div class="field">
39. <label class="label" style="text-align:left">Email</label>
40. <div class="control has-icons-left has-icons-right">
41. <input class="input is-danger" type="email" name="email" placeholder="Email do usuário" value="">
42. <span class="icon is-small is-left">
43. <i class="fas fa-envelope"></i>
44. </span>
45. <span class="icon is-small is-right">
46. <i class="fas fa-exclamation-triangle"></i>
47. </span>
48. </div>
49. </div>
50. <div class="field">
51. <label class="label" style="text-align:left">Password</label>
52. <div class="control has-icons-left has-icons-right">
53. <input class="input is-danger" type="password" name="password" placeholder="\*\*\*\*\*\*" value="">
54. <span class="icon is-small is-left">
55. <i class="fas fa-lock"></i>
56. </span>
57. <span class="icon is-small is-right">
58. <i class="fas fa-exclamation-triangle"></i>
59. </span>
60. </div>
61. </div>
62. <div class="field">
63. <p class="control">
64. <button class="button is-success is-medium is-fullwidth">
65. Login
66. </button>
67. </p>
68. </div>
69. </div>
70. </div>
71. </div>
72. </template>
73. Vamos criar um data() que irá retornar um json com os campos do formulário.
74. <script>
75. export default {
76. data(){
77. return{
78. name:'',
79. password:'',
80. email: ''
81. }
82. }
83. }
84. Agora vamos fazer um databynd nos campos do formulário com o v-model e nome do campo.
85. <input class="input is-success" type="text" placeholder="Nome do usuário" v-model="name">
86. <input class="input is-danger" type="email" placeholder="Email do usuário" v-model="email">
87. <input class="input is-danger" type="password" placeholder="\*\*\*\*\*\*" v-model="password">
88. Vamos criar um método chamado register() dentro do json de methods, que no inicio irá fazer um console.log das variáveis
89. methods:{
90. register(){
91. console.log(this.name);
92. console.log(this.email);
93. console.log(this.password);
94. }
95. }
96. Agora vamos chamar o envento @click no botão passando o método register().
97. <button class="button is-success is-medium is-fullwidth" @click="register">
98. Login
99. </button>
100. A página ficará assim:
101. 

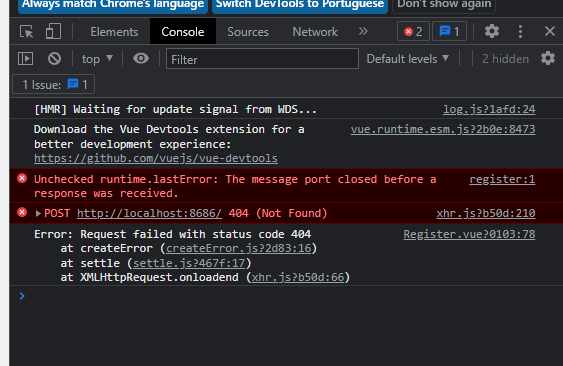
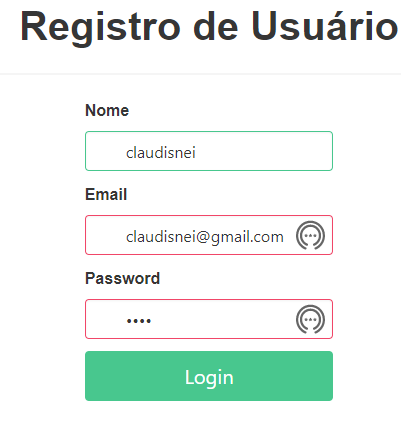
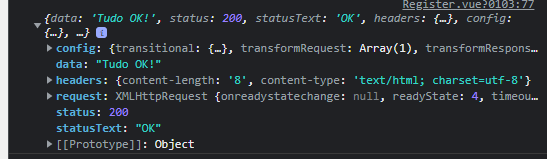
**Requisição da formulário de Registro para a API**

1. Faremos a requisição utilizando o axios.
2. Vamos importar o axios dentro do script do componente Register.vue
3. <script>
4. import axios from 'axios';
5. no método register vamos utilizar o axios que fará uma requisição do tipo get para uma rota da API. <http://localhost:8686/> e nos dará uma resposta ou um erro.
6. methods:{
7. register(){
8. axios.get('http://localhost:8686/').then((*res*)*=>*{
9. console.log(*res*);
10. }).catch(*err=>*{
11. console.log(*err*);
12. });
13. }
14. }
15. Ocorreu um erro devido ao cors, pois é uma política de segurança para troca de informações entre servidores. O projeto vue está rodando em uma porta diferente da API.

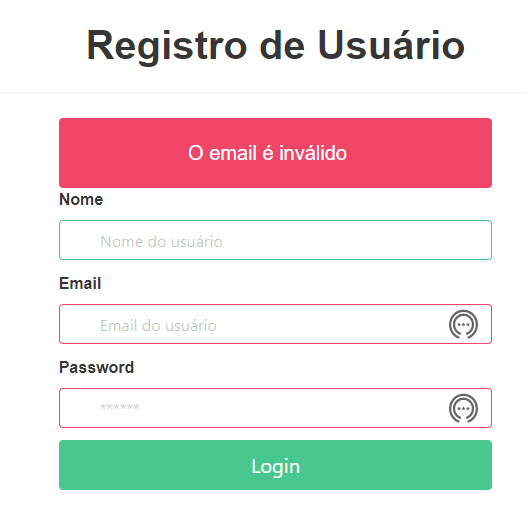
**Instalando a biblioteca cors na API**

1. Para resolver este erro temos que ir até a pasta onde está rodando a API, parar o servidor e instalar a biblioteca chamada cors. – npm install cors -- save
2. Essa biblioteca funciona tipo um middleware do express
3. Vamos importar essa biblioteca para o arquivo index.js do projeto API\_REST\_USERS
4. *var* cors = require('cors');
5. vamos solicitar que o app utilize o cors.
6. app.use(cors());
7. rode novamente o servidor e faça o teste na view register.

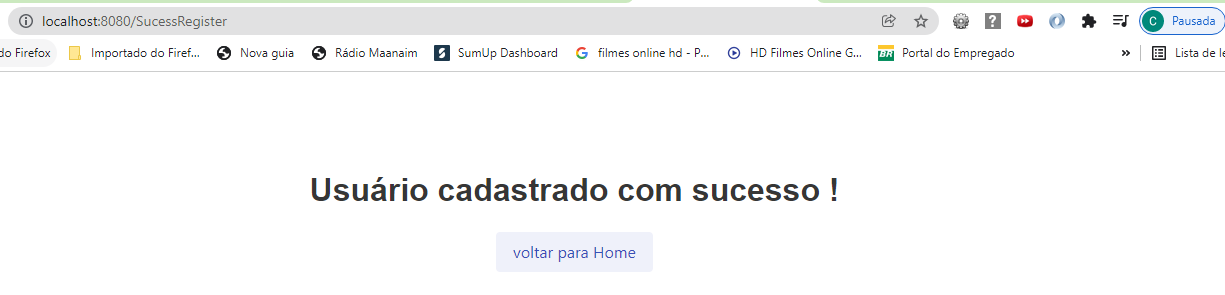
**Enviando dados para o servidor**

1. Vamos alterar a rota no método register para enviar os dados dos inputs.
2. methods:{
3. register(){
4. axios.post('http://localhost:8686/user',{
5. name: this.name,
6. password: this.password,
7. email: this.email;
8. }).then((*res*)*=>*{
9. console.log(*res*);
10. }).catch(*err=>*{
11. console.log(*err*);
12. });
13. }
14. }
15. Ao clicarmos no botão salvar aparece um erro que não informa nada
16. 
17. Vamos alterar a mensagem de erro para melhorar a resposta ao usuário.
18. }).catch(*err=>*{
19. *var* errMens = *err*.response.data.err;
20. console.log(errMens);
21. });
22. Desta forma o erro será mais amigável.
23. 
24. Agora vamos criar um usuário para testar
25. 
26. O retorno no console foi.
27. 

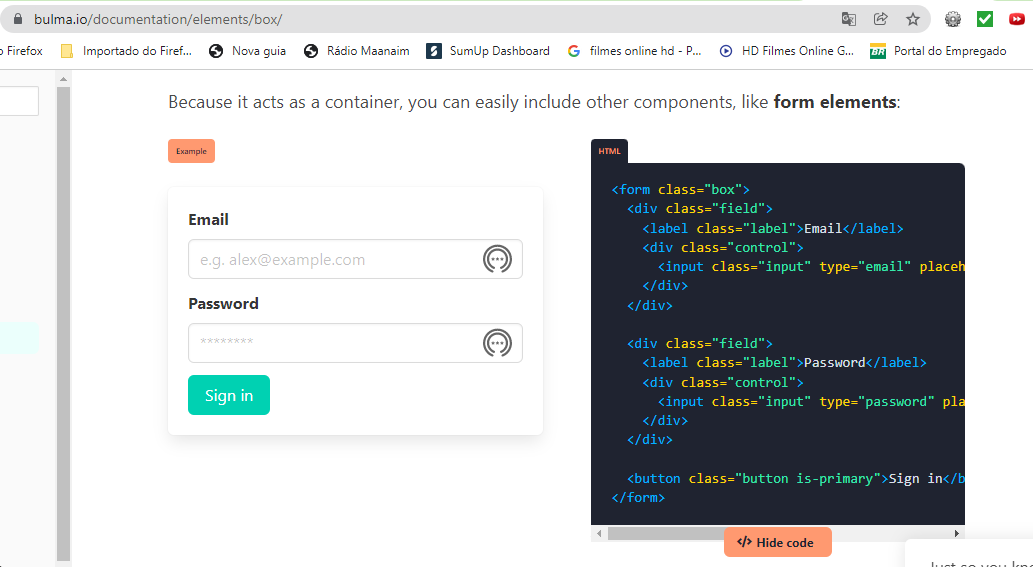
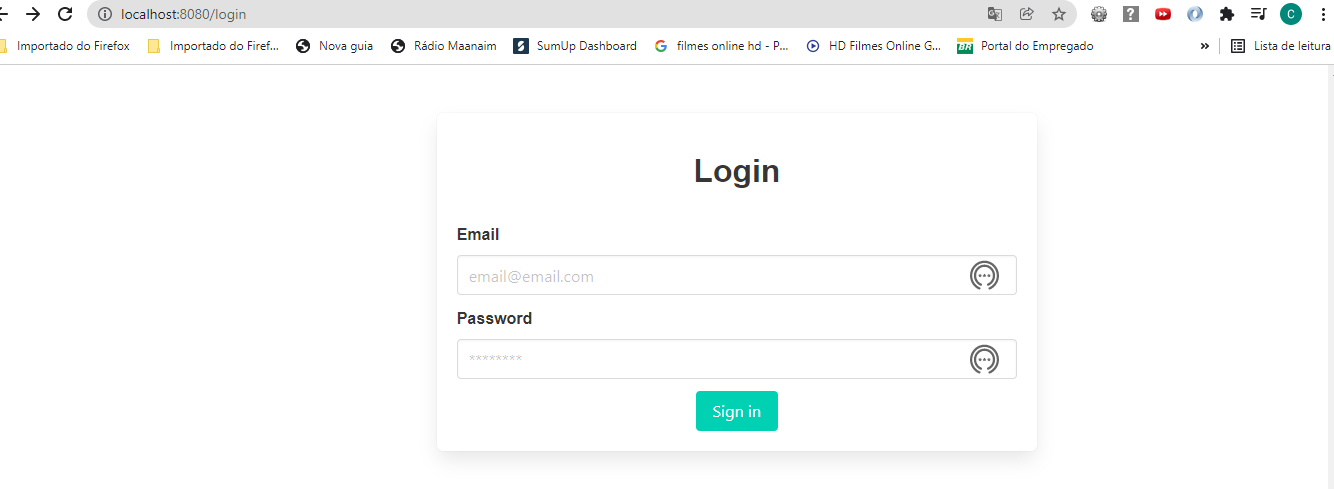
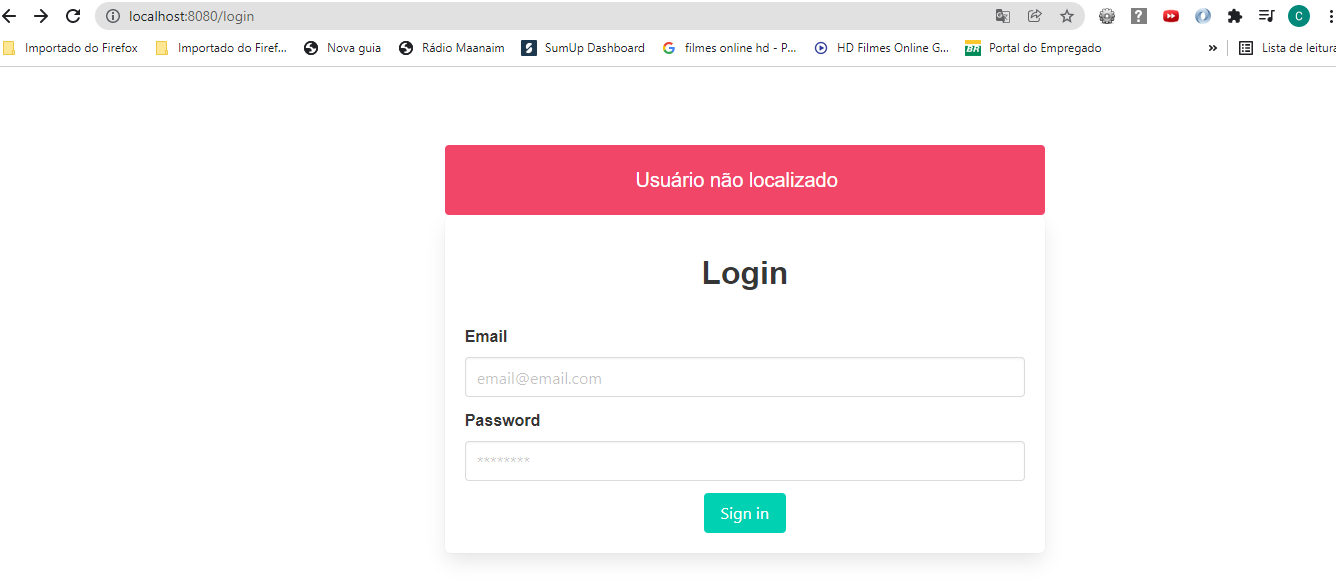
**Tratando a Resposta para o usuário**

1. Para melhorar a visualização do erro para o usuário, vamos criar dentro do data() uma variável chamada error e inicialmente ela será undefined.
2. data(){
3. return{
4. name:'',
5. password:'',
6. email: '',
7. error: undefined
8. }
9. Vamos criar na área do template do componente Register.vue uma div que terá um parágrafo a div terá um v-if que irá verificar se o error é undefined, se não for o erro será exibido.
10. <template>
11. <div class="content is-medium">
12. <h1 style="padding-top:40px">Registro de Usuário</h1>
13. <hr>
14. <div class="columns is-centered">
16. <div v-if="error !=undefined">
17. <p>{{error}}</p>
18. </div>
19. Vamos agora passar a mensagem de erro para a variável error.
20. methods:{
21. register(){
22. axios.post('http://localhost:8686/user',{
23. name: this.name,
24. password: this.password,
25. email: this.email
26. }).then(*res* *=>*{
27. console.log(*res*);
28. }).catch(*err=>*{
29. *var* errMens = *err*.response.data.err;
30. this.error = errMens;
31. });
32. }
33. }
34. Vamos agora melhorar a exibição da mensagem utilizando um quandro notification do bulma
35. <div v-if="error !=undefined">
36. <div class="notification is-danger">
37. <p>{{error}}</p>
38. </div>
39. </div>
40. O erro aparecerá da seguinte forma.
41. 
42. Quando o cadastro for realizado com sucesso iremos encaminhar o usuário para uma página com uma mensagem de usuário cadastrado com sucesso.
43. Iremos utilizar o vue router para isso.
44. Então vamos na promisse do método register e quando a requisição ocorre com sucesso, vamos chamar o método push do router passando para a variável name o name da rota que será utilizada.
45. }).then(*res* *=>*{
46. console.log(*res*);
47. this.$router.push({name: 'SucessRegister'});

**Pagina de mensagem de sucesso**

1. Na pasta views crie um novo arquivo chamado SucessRegister.vue
2. Neste arquivo crie a configuração básica de um componente vue.
3. Dentro da área de template crie um h1 com a mensagem de usuário cadastrado com sucesso
4. Crie também um botão com o label de voltar para Home, no botão inclua o evento click e passe para ele o método ‘home’.
5. Na área de script
6. Dentro da área de methods crie o método chamado home, este método será utilizado para que quando o usuário clicar no botão ele seja redirecionado para a página home.
7. Neste método chame o router com seu método push e passe para a variável name o Home
8. Código do componente SucessRegister
9. <template>
10. <div>
11. <h1 class="title is-3" style="padding-top:100px">Usuário cadastrado com sucesso !</h1>
12. <button @click="home" class="button is-link is-light">voltar para Home</button>
13. </div>
14. </template>
15. <script>
16. export default {
17. data(){
18. return{
19. }
20. },
21. methods:{
22. home(){
23. this.$router.push({name: 'Home'});
24. }
25. }
26. }
27. </script>
28. <style>
29. </style>
30. Para isso funcionar temos que ir no arquivo index.js da pasta router
31. Insira as informações da rota para a página SucessRegister.
32. {
33. path: '/SucessRegister',
34. name: 'SucessRegister',
35. component: SucessRegister
36. },
37. Agora quando o usuário realizar um cadastro será encaminha para a página
38. 

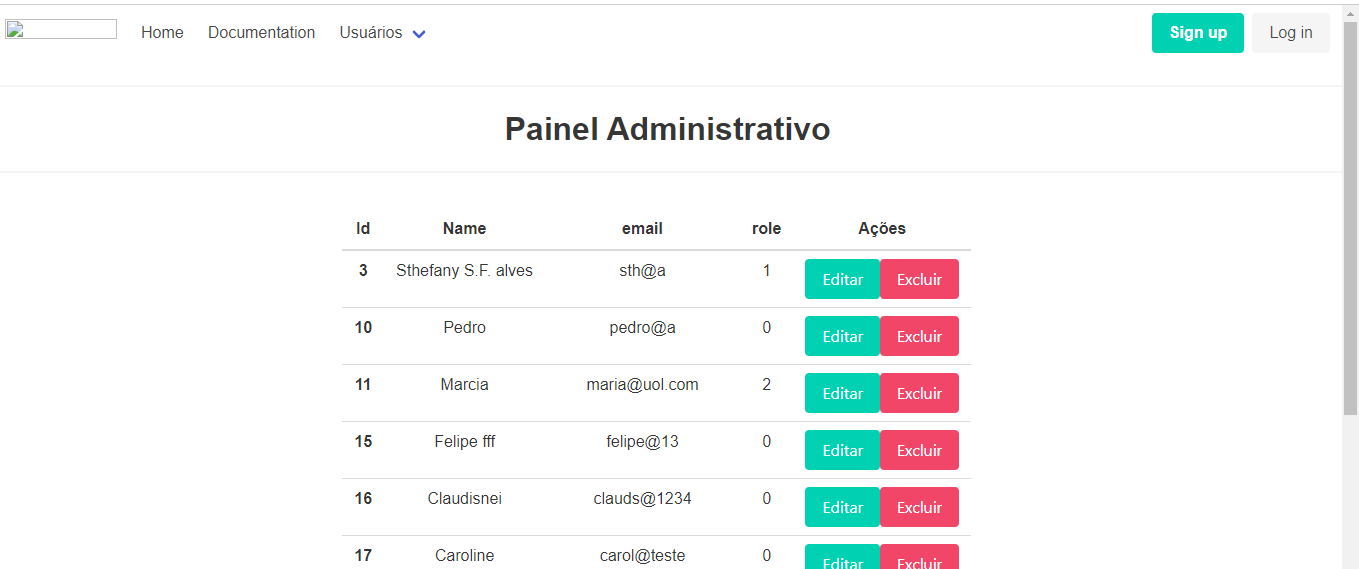
**Criando a página de Login**

1. Na pasta views crie o componente Login.vue com a estrutura básica de um componente vue.
2. <template>
4. </template>
5. <script>
6. export default {
7. data(){
8. return{
9. }
10. },
11. methods:{
12. }
14. }
15. </script>
16. <style>
17. </style>
18. Na página do bulma copíe o código de um form de login.
19. 
20. Na área do template vamos inserir o html da página de login
21. <template>
22. <div class="content is-medium">
23. <div class="columns is-centered">
24. <div class="column is-one-third" style="padding-top:60px">
25. <div class="box" style="width:600px; padding-top:30px">
26. <h1 class="title is-3" style="padding:10px">Login</h1>
27. <div class="field">
28. <label class="label textLabel">Email</label>
29. <div class="control">
30. <input class="input" type="email" placeholder="email@email.com">
31. </div>
32. </div>
33. <div class="field">
34. <label class="label textLabel">Password</label>
35. <div class="control">
36. <input class="input" type="password" placeholder="\*\*\*\*\*\*\*\*">
37. </div>
38. </div>
39. <button class="button is-primary">Sign in</button>
40. </div>
41. </div>
42. </div>
43. </div>
45. </template>
46. Para podermos visualizar a página vamos configurar uma rota no arquivo index.js da pasta router.
47. Vamos incluir os dados abaixo:
48. {
49. path: '/login',
50. name: 'Login',
51. component: Login
52. },
53. Agora basta digitar a url <http://localhost:8686/login> que a página abrirá.
54. Faça alguns ajustes para aparência configure uma largura para o box e paddings.
55. A página ficará desta forma:
56. 
57. Vamos agora configurar as variáveis email e password para que quando o usuário preencher os inputs e clicar em Sing in, ele consiga ir para a página Home.
58. Na área de script dentro export default e dentro de return da propriedade data() crie as variáveis email e password e error.
59. export default {
60. data(){
61. return{
62. email: '',
63. password: '',
64. error: undefined
65. }
66. Agora vá nos inputs da página e crie um v-model com as variáveis criadas.
67. <input class="input" type="email" v-model="email" placeholder="email@email.com">
68. <input class="input" type="password" v-model="password" placeholder="\*\*\*\*\*\*\*\*">
69. Na parte superior antes do título login adicione um box de notificação onde será passado o erro através de um v-if. Se ocorrer erro o box mostrará a mensagem.
70. <div class="column is-one-third" style="padding-top:60px">
71. <div style="width:600px; padding-top:30px" v-if="error !=undefined">
72. <div class="notification is-danger">
73. <p>{{error}}</p>
74. </div>
75. </div>
76. Dentro da propriedade methods crie um método chamado login().
77. Onde será chamado o axios com o verbo post com uma requisição para a rota login de nossa api.
78. Para facilitar foi criado uma pasta config com um arquivo Config.js para armazenar as variáveis a aplicação, neste arquivo está salvo o host da api.
79. Este método irá pegar os dados passados na página nos campos input e passará para rota login da api. Se o usuário existir ele será encaminhado para a página uma página específica, caso contrário será enviado um erro para o box no topo da página.
80. methods:{
81. login(){
82. axios.post(config.hostApi+'login',{
83. email: this.email,
84. password: this.password
85. }).then(*res=>*{
86. console.log(*res*);
87. this.$router.push({name: 'SucessRegister'});
88. }).catch(*err=>*{
89. //var msErro = err.response
90. this.error = *err*.response.data.err;
92. })
93. }
94. Se ocorre um erro será mostrado como abaixo.
95. 

**Salvando o tokenJwt no localStorage**

1. Nométodo login vamos criar um item dentro de localStorage chamado token e vamos passar o tokenJwt passado pela resposta da requisição.
2. Com esse recurso poderemos verificar se o usuário e do tipo admin.
3. login(){
4. axios.post(config.hostApi+'login',{
5. email: this.email,
6. password: this.password
7. }).then(*res=>*{
8. console.log(*res*);
9. localStorage.setItem('token',*res*.data.tokenJwt);
10. this.$router.push({name: 'Home'});
11. }).catch(*err=>*{
13. this.error = *err*.response.data.err;
14. })
15. }

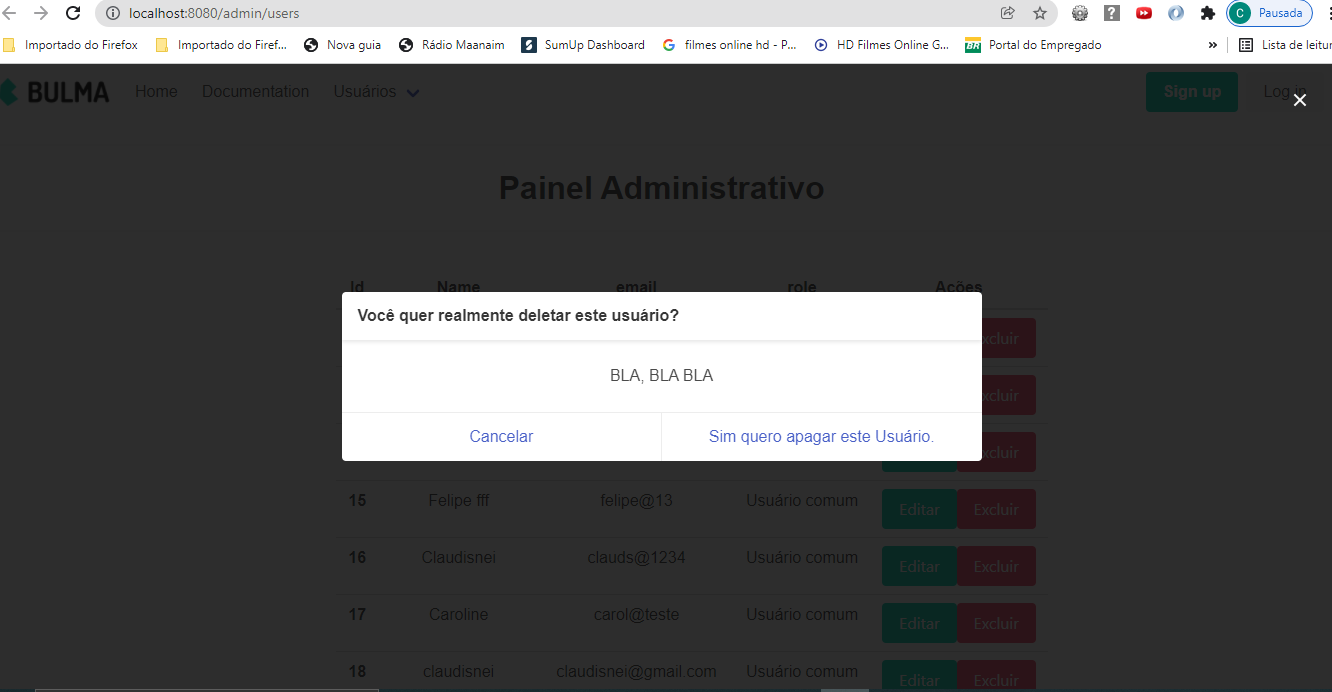
**Criando a página de gestão de Usuários**

1. Vamos criar o componente Users.vue com na pasta views.
2. <template>
3. <div>
4. <h1 class="title is-1">
5. Painel Administrativo
6. </h1>
7. </div>
9. </template>
10. <script>
11. export default {
12. data(){
13. return{
14. }
15. },
16. methods:{
17. }
18. }
19. </script>
20. <style scoped>
21. </style>
22. Vamos agora criar a rota para acesso ao painel administrativo
23. {
24. path: '/admin/users',
25. name: 'Users',
26. component: Users
27. },
28. Vamos proteger esta rota de forma que para se logar seja necessário ter o token salvo no localStorage. Para isso utilizar uma propriedade dentro da rota.
29. Vamos na rota Users e configuramos a propriedade beforEnter com a lógica requerida para acesso.
30. {
31. path: '/admin/users',
32. name: 'Users',
33. component: Users,
34. beforeEnter: (*to*, *from*, *next*)*=>*{
35. if(localStorage.getItem('token')!=undefined){
36. next();
37. }else{
38. next('/login');
39. }
40. }
41. },
42. Ou seja se o token estiver salvo no localstorage o usuário conseguirá acessar a rota Users, caso contrário será direcionado para rota ‘/login’.
43. Porem desta forma é possível burlar as regras e entrar na página. Para evitar isso temos que criar um método de validação do token no backend.
44. Vamos criar um método chamado validade no controller HomeController que terá somente uma mensagem de tudo ok.
45. async validate(*req*, *res*){
46. *res*.send("Ok");
47. }
48. Vamos no arquivo de rotas da API routes.js e vamos inserir a rota
49. //rota para acesso do painel administrativo
50. router.post('/validate',AdminAuth ,HomeController.validate)
51. que utilizará o middleware de autenticação criado para autenticar os usuário admin.
52. Vamos agora importar o axios no arquivo index.js da pasta router.
53. Vamos criar também uma variável req onde iremos setar o cabeçalho Authorization que é necessário para o middleware adminAuth realizar a autenticação.
54. *var* req = {
55. headers: {
56. Authorization: "Bearer "+ localStorage.getItem('token')
57. }
58. }
60. Agora passamos a variável req junto da requisição do axios.
61. }
62. axios.post(config.hostApi+'validate',{},req).then(*res=>*{
63. console.log(*res*);
64. next()
65. }).catch(*err=>*{
66. console.log(*err*.response.data);
67. next('/');
68. });
69. }else{
70. next('/login');
71. }
72. }
73. },
74. O middleware irá comparar o token passado pela variável req com o token salvo no banco de dados. E verificar se é tipo admin. Se for admin o usuário será direcionado para a pagina /Amin/users se não será direcionado para pagina home /
75. Como essa verificação será realizada em toda rota que o acesso seja somente para usuário admin, vamos criar uma função dentro do arquivo index.js para fazer essa verificação e somente passar a função para a propriedade beforeEnter.
76. A rota ficará assim
77. {
78. path: '/admin/users',
79. name: 'Users',
80. component: Users,
81. beforeEnter: AdminAuth
82. },
83. E afunção.
84. *function* AdminAuth(*to*, *from*, *next*){
85. if(localStorage.getItem('token')!=undefined){
86. *var* req = {
87. headers: {
88. Authorization: "Bearer "+ localStorage.getItem('token')
89. }
90. }
91. axios.post(config.hostApi+'validate',{},req).then(*res=>*{
92. console.log(*res*);
93. next()
94. }).catch(*err=>*{
95. console.log(*err*.response.data);
96. next('/');
97. });
98. }else{
99. next('/login');
100. }
101. }
102. Melhorando a página de gestão administrativa
103. Na área do template vamos criar uma tabela que terá os dados dos usuário.
104. No body do formulário vamor criar um v-for que irá receber um array dos usuários “user” e para cada usuário será criada uma linha com os dados e dois botões um de Editar e outro de excluir.
105. <template>
106. <div >
107. <div>
108. <nav class="navbar" role="navigation" aria-label="main navigation">
109. <div class="navbar-brand">
110. <a class="navbar-item" href="">
111. <img src="https://bulma.io/images/bulma-logo.png" width="112" height="28">
112. </a>
113. <a role="button" class="navbar-burger" aria-label="menu" aria-expanded="false" data-target="navbarBasicExample">
114. <span aria-hidden="true"></span>
115. <span aria-hidden="true"></span>
116. <span aria-hidden="true"></span>
117. </a>
118. </div>
119. <div id="navbarBasicExample" class="navbar-menu">
120. <div class="navbar-start">
121. <a class="navbar-item">Home</a>
122. <a class="navbar-item">Documentation</a>
123. <div class="navbar-item has-dropdown is-hoverable">
124. <a class="navbar-link">Usuários</a>
125. <div class="navbar-dropdown">
126. <a class="navbar-item" >Listar Usuário</a>
127. <a class="navbar-item">Listar</a>
128. <a class="navbar-item">Editar</a>
129. <hr class="navbar-divider">
130. <a class="navbar-item">Sair</a>
131. </div>
132. </div>
133. </div>
134. <div class="navbar-end">
135. <div class="navbar-item">
136. <div class="buttons">
137. <a class="button is-primary"><strong>Sign up</strong></a>
138. <a class="button is-light">Log in</a>
139. </div>
140. </div>
141. </div>
142. </div>
143. </nav>
144. <hr>
145. <h1 class="title is-3">
146. Painel Administrativo
147. </h1>
148. <hr>
149. </div>
150. <div class="column is-centered is-offset-one-quarter">
151. <table class="table">
152. <thead>
153. <tr>
154. <th>Id</th>
155. <th>Name</th>
156. <th>email</th>
157. <th>role</th>
158. <th>Ações</th>
159. </tr>
160. </thead>
161. <tbody >
162. <tr v-for="user in users" :key="user.id">
163. <th>{{user.id}}</th>
164. <td>{{user.name}}</td>
165. <td>{{user.email}}</td>
166. <td>{{user.role}}</td>
167. <td>
168. <button class="button is-primary">Editar</button>
169. <button class="button is-danger">Excluir</button>
170. </td>
171. </tr>
172. </tbody>
173. </table>
174. </div>
176. </div>
177. </template>
178. Na área de script na propriedade export defaut vamos criar um data() para retornar um array de usuários chamado users.
179. Vamos criar o método created() que irá criar uma variável req que armazenará o cabeçalho Authorization com o token do usuário.
180. Ainda dentro do método created vamos utilizar o axios para realizar uma requisição do tipo get para a rota /user junto com a variável req.
181. Essa requisição na API irá retornar no res.data a lista de usuário que será salva dentro da variável users que criamos dentro de data().
182. Essa variável users será usada dentro do v-for.
183. export default {
184. data(){
185. return{
186. users: []
187. }
188. },
190. created(){
191. *var* req = {
192. headers: {
193. Authorization: "Bearer "+ localStorage.getItem('token')
194. }
195. }
196. axios.get(config.hostApi+'user',req).then(*res=>*{
197. console.log(*res*.data)
198. this.users = *res*.data;
199. }).catch(*err=>*{
200. console.log(*err*);
201. });
202. }
203. }
204. Ao realiazar o login com um usuário que é do tipo admin será aberto a seguinte página.
205. 

**Aplicando filtros na listagem de usuário**

1. Dentro da área do script no arquivo Users.vue, vamos inserir uma propriedade chamada filters.
2. Nela vamos criar um método chamado processRole que irá receber uma variável de sistema value. Dentro do método vamos simplesmente verificar se value é do tipo 1 ou 0 e retornar o tipo de usuário correspondente.
3. //essa propriedade possibilita a criação de filtros que serão utilizados no interface com o usuário
4. filters:{
5. processRole: *function*(*value*){
6. if(*value* == 0){
7. return "Usuário comum"
8. }else if(*value* == 1){
9. return "Admin"
10. }
11. }
12. }
13. Após criado o método dentro do filtro, vamos até o campo da tabela que se quer mudar, neste caso, user.role e vamos inserir o filtro. Para isso inserirmos após o campo, um “pipe”( | ) e em seguinda o nome do filtro criado.
14. <td>{{user.role | processRole}}</td>
15. Desta forma será exibido o tipo de usuário “literal” ao invés de 1 ou 0.
16. 

**Criando um Modal para confirmar a deleção de usuário**

1. Na página do bulma, copie o código exemplo de um modal
2. E cole dentro da área do template do componente Users.vue.
3. Volte no bulma e copie o código de um card com opção de botões de salve, edit e cancel.
4. E cole dentro do conteúdo do modal.
5. Edite o código Html para ficar como abaixo:
6. <!--modal da confirmação de deleção-->
7. <div class="modal is-active">
8. <div class="modal-background"></div>
9. <div class="modal-content">
10. <!-- Any other Bulma elements you want -->
12. <!--Card com as opções de confirmar e cancelar exclusão-->
13. <div class="card">
14. <header class="card-header">
15. <p class="card-header-title">
16. Você quer realmente deletar este usuário?
17. </p>
18. </header>
19. <div class="card-content">
20. <div class="content">
21. BLA, BLA BLA
22. </div>
23. </div>
24. <footer class="card-footer">
25. <a href="#" class="card-footer-item">Cancelar</a>
26. <a href="#" class="card-footer-item">Sim quero apagar este Usuário.</a>
27. </footer>
28. </div>
29. <!--final do card-->
31. </div>
32. <button class="modal-close is-large" aria-label="close"></button>
33. </div>
34. <!--final do modal-->
35. A tela ficará desta forma.
36. 
37. Para exibir e ocultar o modal, basta incluirmos na class modal a propriedade is-active.

1. Vamor criar uma variável chamada showModal que será true ou false.
2. data(){
3. return{
4. users: [],
5. showModal = false
6. }
7. },
8. Então vamos byndar uma variável chamada :class no modal e passando a classe modal com a condição true e também a classe is-active com base no valor da variável showModal, quando a variável modal for alterada o modal será exibido.
9. Vamos criar um método chamado hideModal que irá basicamente ocultar o modal, passando o valor false para a variável showModal.
10. methods:{
11. hideModal(){
12. this.showModal = false
13. }
14. },
15. Vamos agora inserir esse método no botão fechar do modal.
16. <button class="modal-close is-large" aria-label="close" @click="hideModal()"></button>
17. Vamos fazer o mesmo para o link cancelar dentro do card
18. <footer class="card-footer">
19. <a href="#" class="card-footer-item" @click="hideModal()">Cancelar</a>
20. <a href="#" class="card-footer-item">Sim quero apagar este Usuário.</a>
21. </footer>
22. C
23. Criamos uma variável userid para que quando o usuário clicar no botão deletar o id do usuário sera passado para essa variável.
24. data(){
25. return{
26. users: [],
27. showModal : false,
28. userid: -1
29. }
30. },
31. Vamos agora criar um método chamado showModalUser, que irá capturar o id do usuário que está sendo apagado passando para a variável useriid e também exibir o modal quando o usuário clicar no botão para deletar.
32. showModalUser(*id*){
33. console.log("Id do user: "+ *id*);
34. this.showModal = true;
35. this.userid = *id*;
36. }
37. Vamos inserir este método no botão excluir da tabela.
38. <button class="button is-danger"  @click="showModalUser(user.id)" >Excluir</button>
39. Agora vamos criar um novo método chamado userDelete que irá realizar uma requisição com o axios deletando o usuário do id gravado no userid.
40. userDelete(){
41. *var* req = config.tokenLocalStorage;
42. axios.delete(config.hostApi+'user/'+this.userid, req).then(*res=>*{
43. this.showModal = false;
44. //Possibilidade de aplicação de filtro
45. //this.users = this.users.filter(u => u.id != this.userid);
46. console.log(*res*);
47. //console.log(this.users)
48. }).catch(*err=>*{
49. this.showModal = false;
50. console.log(*err*);
51. });
52. //consultando o banco de dados e atualizando a relação de usuários
53. axios.get(config.hostApi+'user', req).then(*res=>*{
54. console.log(*res*.data);
55. }).catch(*error=>*{
56. console.log(*error*);
57. });
58. }
59. Vamos inserir no link de deleção do card o método userDelete para efetivar a exclusão do usuário
60. <footer class="card-footer">
61. <a href="#" class="card-footer-item" @click="hideModal()">Cancelar</a>
62. <a href="#" class="card-footer-item" @click="userDelete()">Sim quero apagar este Usuário.</a>
63. </footer>

**EDIÇÃO DO USUÁRIO**

1. Iremos criar uma página para o usuário editar informações do usuário.
2. Na pasta de views vamos criar uma copia do componente Register alterando o nome para UserEdit.
3. Faça algumas alterações no html conforme abaixo:
4. <template>
5. <div class="content is-medium">
6. <h1 style="padding-top:40px">Edição de Usuário</h1>
7. <hr>
8. <div class="columns is-centered">
9. <div class="column is-one-third">
10. <div v-if="error !=undefined">
11. <div class="notification is-danger">
12. <p>{{error}}</p>
13. </div>
14. </div>
15. <div class="field">
16. <label class="label" style="text-align:left">Nome</label>
17. <div class="control has-icons-left has-icons-right">
18. <input class="input is-success" type="text" placeholder="Nome do usuário" v-model="name">
19. <span class="icon is-small is-left">
20. <i class="fas fa-user"></i>
21. </span>
22. <span class="icon is-small is-right">
23. <i class="fas fa-check"></i>
24. </span>
25. </div>
26. </div>
27. <div class="field">
28. <label class="label" style="text-align:left">Email</label>
29. <div class="control has-icons-left has-icons-right">
30. <input class="input is-success" type="email" placeholder="Email do usuário" v-model="email">
31. <span class="icon is-small is-left">
32. <i class="fas fa-envelope"></i>
33. </span>
34. <span class="icon is-small is-right">
35. <i class="fas fa-exclamation-triangle"></i>
36. </span>
37. </div>
38. </div>
39. <div class="field">
40. <p class="control">
41. <button class="button is-success is-medium is-fullwidth" @click="update">
42. Salvar
43. </button>
44. </p>
45. </div>
46. </div>
48. </div>
49. </div>
50. </template>
51. No arquivo de rotas index.js vamos importar o componente UserEdit e criar uma rota para acessar a página do componente.
52. {
53. path: '/admin/user/edit/:id',
54. name: 'UserEdit',
55. component: UserEdit,
56. beforeEnter: AdminAuth
57. },
58. C
59. Na propriedade data() vamos criar as variáveis que serão necessárias na componente.
60. data(){
61. return{
62. name:'',
63. email: '',
64. id: -1,
65. error: undefined
66. }
67. },
68. C
69. Vamos criar o método created() que irá pegar o id do usuário que foi passado por parâmetro e utilizar o axios para fazer uma requisição buscando o usuário do id passado.
70. created(){
71. //pegando o id do usuário passadi oeka requisição
72. *var* userId = this.$route.params.id;
73. //verificando se o id passado pelo usuário é valido
74. axios.get(config.hostApi+'user/'+ userId, config.tokenLocalStorage).then(*res=>*{
75. console.log(*res*);
76. //pegando os valores que foram buscados no banco de dados pelo axios
77. this.name = *res*.data.user.user.name;
78. this.email = *res*.data.user.user.email;
79. this.id = *res*.data.user.user.id;
80. }).catch(*err=>*{
81. console.log(*err*.response);
82. //caso não encontre o usuário, o cliente será redirecionado para a página home
83. this.$router.push({name: 'Users'});
84. })
86. },
87. Caso o usuário não exista, o cliente será redirecionado para o componente Users.
88. Caso exista, os dados do usuário que estão na requisição serão passados para as variáveis e a página irá exibir o usuário dentro dos campos input.
89. 