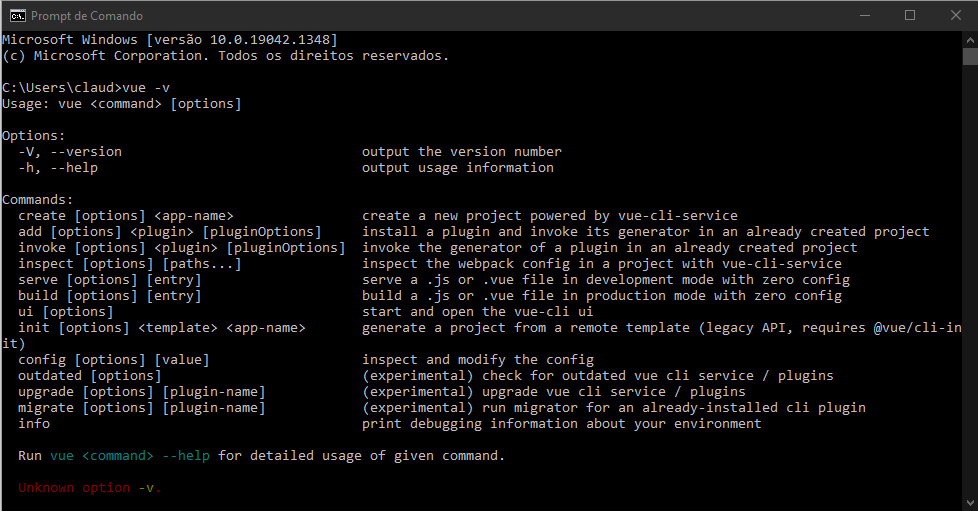
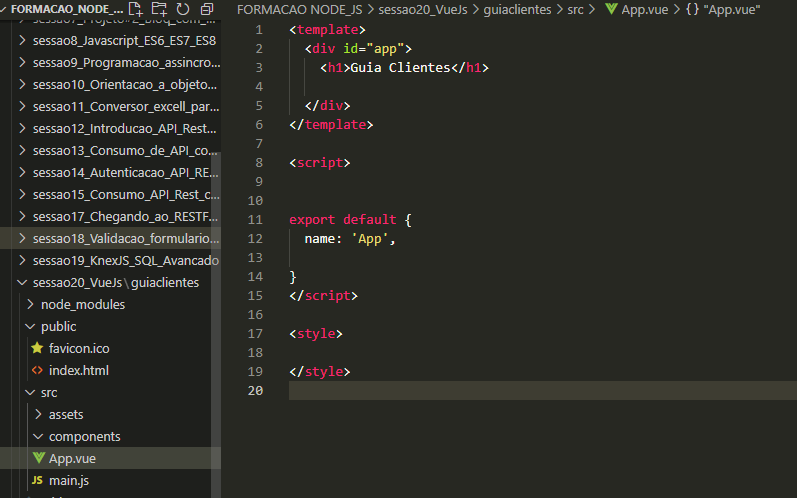
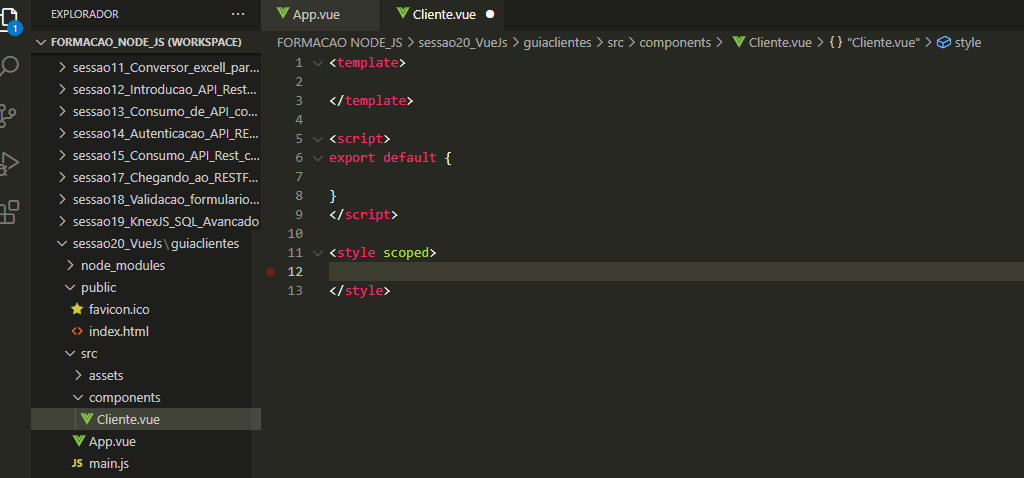
TRABALHANDO COM O FRAMEWORK VUEJS

1. Entre no endereço : nodejs.org e instale o node js
2. Instalar o visual estúdio code e o plugin vetur
3. Instalar o Vue CLI.
4. Abra o prompt do Windows no modo administrador, desative o firewaw e antivírus e digite no c:\ do prompt
5. Npm install -g @vue/cli
6. Para ver se o vue foi instalado digite no prompt : vue -v
7. Tem que aparecer a tela abaixo.



1. Após a instalação vá para a pasta onde será criado o novo projeto
2. Digite :
3. - vue create “nome do projeto”
4. O vue irá criar uma pasta com o nome do projeto
5. Após criação do projeto vá na pasta do projeto e digite :
6. - ‘npm run serve’ para iniciar o servidor de desenvolvimento do vue.
7. O projeto possui várias pastas, porém.
8. Temos basicamente 3 pastas importantes que são a public onde está o arquivo index.html
9. onde vue será injetado
10. EX:
11. <!DOCTYPE *html*>
12. <html *lang*="">
13. <head>
14. <meta *charset*="utf-8">
15. <meta *http-equiv*="X-UA-Compatible" *content*="IE=edge">
16. <meta *name*="viewport" *content*="width=device-width,initial-scale=1.0">
17. <link *rel*="icon" *href*="*<*%= BASE\_URL %>favicon.ico">
18. <title>*<*%= htmlWebpackPlugin.options.title %></title>
19. </head>
20. <body>
21. <noscript>
22. <strong>We're sorry but *<*%= htmlWebpackPlugin.options.title %> doesn't work properly without JavaScript enabled. Please enable it to continue.</strong>
23. </noscript>
24. <div *id*="app"></div>
25. <!-- built files will be auto injected -->
26. </body>
27. </html>
28. Dentro da pasta src temos as pastas assets onde serão adicionados os vídeos, imagens que serão utilizadas no projeto.
29. Na pasta componentes onde serão guardados os componentes, podendo ser criadas outras pastas.
30. Dentro da pasta compontens é onde está o componente app.vue que é responsável por mostar a página inicial do projeto. E este arquivo está sendo inserido dentro da página index.html do projeto
31. O componente App.vue é o componente raiz da aplicação tudo que for feito na aplicação será realizado em cima deste componente.
32. O arquivo main.js cria uma nova instância Vue e joga sobre uma tag Html e monta o aplicativo sobre a tag app
33. **Criando componentes:**
34. Para melhorar a organização de uma aplicação todos os elementos da aplicação serão criados como componentes. Ex: menu, rodapé, conteúdo principal, etc...
35. E cada componente no VueJs será um arquivo. Cada componente no vue é dividido em três partes
36. Html, Css e JavaScript
37. Todo arquivo terminado em vue é um arquivo de componente.
38. Quando criamos um componente, podemos chamar qualquer compontente dentro de outro compontente bastando digitar a tag <”nome do componente” ..... >
39. Para criar uma aplicação limpa basta ir no aplicativo App.vue e limpar os códigos criados pelo vue o aquivo ficará assim:
40. 
41. Para criar um novo componente basta ir na pasta components e criar um novo arquivo vue
42. Vamos criar o componente Cliente.vue
43. O novo componente deverá ter a seguinte estrutura basica
44. 
45. Vamos adicionar uma tag h1 no template
46. <h1>Cliente aqui</h1>
47. Para usar o componente basta importar este componente no arquivo que queremos utilizar.
48. Vamos importar o componente Cliente para o aquivo App.vue
49. Dentro da tag script informamos o nome do componente e de onde ele será importado, porém temos que informar também que queremos utilizar este novo compontente, conforme abaixo:
50. <script>
51. import Cliente from './components/Cliente'
52. export default {
53. name: 'App',
54. components:{
55. Cliente
56. }
58. }
59. </script>
60. Depois disso temos que inserir o componente no template.
61. <template>
62. <div id="app">
63. <h1>Guia Clientes</h1>
64. <cliente/>
65. </div>
66. </template>
67. <script>
68. import Cliente from './components/Cliente'
69. export default {
70. name: 'App',
71. components:{
72. Cliente
73. }
75. }
76. </script>
77. <style>
78. </style>
79. Agora basta fazer as alterações no componente que estas serão aplicadas dentro do App.vue
80. <template>
81. <div id="cliente">
82. <h1 id="cliente-nome">Cliente aqui</h1>
83. <h3>Descrição do Cliente bla bla bla bla bla</h3>
84. </div>
85. </template>
86. <script>
87. export default {
89. }
90. </script>
91. <style scoped>
92. #cliente{
93. *color*: blue;
94. *background-color*: beige;
95. *max-width*: 600px;
96. *height*: 100px;
97. }
98. #cliente-nome{
99. *color*: green;
100. }
101. </style>
102. **Reatividade**
103. Reatividade é quando vc altera o conteúdo de seus dados e o vue atualiza os dados da página.
104. Reatividade utilizando a interpolação
105. Basta utilizar chaves duplas onde será inserido funções criadas no script como abaixo:
106. <template>
107. <div id="cliente">
108. <h1 id="cliente-nome">Cliente</h1>
109. <h3>Nome do cliente: {{nome}}</h3>
110. <p>Descrição do cliente: {{descricao}}</p>
112. </div>
113. </template>
114. <script>
115. export default {
116. data(){
117. return {
118. nome : "Joâo da Silva",
119. numero: "",
120. email: "",
121. idade: 80,
122. descricao: "Lorem ipsum lorem ipson"
123. }
124. }
126. }
127. </script>
128. **Data bind**
129. **On way data bind.**
130. Sempre que formos utilizar uma variável criada na área de data() em um campo de formulário, devemos utilizar ‘:’ na frente do campo value do formulário.
131. <input type="text" :value="nome">
132. <hr>
133. <input type="text" :value="descricao">
134. Os dados serão exibidos dentro do campo do formulário porém se forem alterados a alteração não ocorrerá na base de dados. Ou seja os dados serão exibidos somente para leitura
135. **Two way data bind**
136. Para realizamos alterações tanto no formulário quanto na variável original, devemos utilizar o v-model no lugar de value no campo do formulário.
137. As alterações realizadas no formulário serão refletidas nas variáveis
138. <h3>Two way data bind</h3>
139. <input type="text" v-model="nome">
140. <input type="text" v-model="descricao">
141. **Props**
142. As props vão permitir que se passem os dados de um componente durante a criação deles, a props é uma promessa que no componente terá as variáveis criadas na props.
143. Devemos criar as props dentro do campo export default onde teremos o nome do campo e o tipo de variável do campo.
144. //COMPONENTE CLIENTE
145. export default {
146. data(){
147. return {
148. // nome : "Joâo da Silva",
149. numero: "2",
150. email: "jsssd@gmail.com",
151. idade: 80,
152. descricao: "Lorem ipsum lorem ipson"
153. }
154. },
155. props: {
156. nome: *String*
157. }
159. }
160. Após isso iremos adicionar as variáveis criadas dentro do componente principal
161. //COMPONENTE PRINCIPAL APP
162. <template>
163. <div id="app">
164. <h1>Guia Clientes</h1>
165. <cliente nome= "Pedro"/>
166. <cliente nome="João"/>
167. <cliente nome="Maria"/>
168. </div>
169. </template>
170. **Diretivas condicionais V-if, V-else, V-else-if**
171. È um atributo que não existe no html normal, o v-model é uma diretiva, o vueJs possui as diretivas v-if, v-else, v-else-if eles servem para esconder ou exibir um elemento html apartir de uma condição. Vamos colocar uma condição na idade para exibir o campo idade conforme a condição. O v-else deve ser usado embaixo do v-if. O v-else-if pode ser utilizado sem limites.
172. //código do componente cliente
173. <template>
174. <div id="cliente">
175. <h4>Nome: {{cliente.nome}}</h4>
176. <hr>
177. <p>Email: {{cliente.email}}</p>
178. <p v-if="showIdade == true">Idade: {{cliente.idade}}</p>
179. <p v-else >O Usuário escondeu a idade</p>
180. </div>
181. </template>
182. <script>
183. export default {
184. data(){
185. return {
187. }
188. },
189. props: {
190. //tipo objeto
191. cliente: *Object*,
192. showIdade: *Boolean*
193. }
194. //código do componente Pai App
195. <template>
196. <div id="app">
197. <h1>Guia Clientes</h1>
198. <input type="text" v-model="clienteVictor.nome" />
199. <cliente :cliente="clienteVictor" :showIdade="false"/>
200. <cliente :cliente="clienteVictor" :showIdade="true"/>
201. <cliente :cliente="clienteVictor" :showIdade="false"/>
202. <cliente :cliente="clienteVictor" :showIdade="true"/>
203. </div>
204. </template>
205. <script>
206. import Cliente from './components/Cliente';
207. export default {
208. name: 'App',
209. components:{
210. Cliente
211. },
212. data(){
213. return {
214. clienteVictor: {
215. nome: "Victor Lima",
216. email: "Victor@lima",
217. idade: 23
218. }
219. }
220. }
222. }
223. </script>
224. **Classes condicionais**
225. Podemos utilizar comparação de valores para utilizar uma determinada classe, basta criar uma variável dentro do campo data(), podendo ser booleana. Dentro da tag html fazemos a condição para determinar qual classe será utilizada. No exemplo abaixo criamos uma variável booleana isPremium = true. Antes criamos uma classe cliente-premium que contem um formato diferente de cliente. Se a variável isPremium for false as propriedades dos campos será da classe cliente se não será da classe cliente-premium.
226. //código do componente pai App
227. <template>
228. <div :class="{'cliente': !isPremiun,'cliente-premium':isPremiun}">
229. <h4>Nome: {{cliente.nome}}</h4>
230. <hr>
231. <p>Email: {{cliente.email}}</p>
232. <p v-if="showIdade == true">Idade: {{cliente.idade}}</p>
233. <p v-else >O Usuário escondeu a idade</p>
234. </div>
235. </template>
236. <script>
237. export default {
238. data(){
239. return {
240. isPremiun: true
241. }
242. },
243. **Diretiva condicional v-for**
244. Utilizado para evitar a repetição de códigos. É semelhante ao foreach e para isso devemos ter um array. Como exemplo iremos criar um array clientes e dentro dele vamos criar três clientes, dentro do template criamos uma div que fará um loop no array e irá exibir os códigos que foram criados dentro da div, para cada cliente no array, chamamos o componente cliente e utilizamos uma prop :cliente passando cada cliente para ela. Podemos utilizar a propriedade index do array para mostrar o item do array.
245. <template>
246. <div id="app">
247. <h1>Guia Clientes</h1>
248. <input type="text" v-model="clienteVictor.nome" />
250. <div v-for="(cliente,index) in clientes" :key="cliente.id">
251. <h4> {{index + 1}} </h4>
252. <cliente :cliente="cliente"/>
253. <hr>
254. <h4>Edição:</h4>
255. <input type="text" v-model="cliente.nome">
256. <input type="text" v-model="cliente.email">
257. </div>
258. </div>
259. </template>
260. <script>
261. import Cliente from './components/Cliente';
262. export default {
263. name: 'App',
264. components:{
265. Cliente
267. },
268. data(){
269. return {
270. clienteVictor: {
271. nome: "Debora Sobrinho",
272. email: "debora@s",
273. idade: 40
274. },
275. clientes: [
276. {
277. id: 2,
278. nome: "Sthefany S alves",
279. email: "sth@alves",
280. idade: 18
281. },
282. {
283. id: 3,
284. nome: "Carolina S Alves",
285. email: "Carol@alves",
286. idade: 14
287. },
288. {
289. id: 5,
290. nome: "Claudisnei Bello",
291. email: "cba@alves",
292. idade: 48
293. },
294. {
295. id: 8,
296. nome: "Debora Sobrinho",
297. email: "debora@s",
298. idade: 40
299. }
300. ]
301. }
302. }
304. }
305. </script>
306. **Eventos**
307. No componente temos que criar um elemento que possui eventos exemplo: um botão, um input etc..
308. No componente Cliente, vamos criar um botão para que ao clicar, seja chamado um método para mudar a cor do cliente, alterando entre as classes cliente e cliente-premium pela mudança da variável isPremium.
309. . Para selecionar o tipo de evento digitamos @ que no vuejs aparecerá os eventos disponíveis, vamos selecionar o evento @click.
310. Vamos criar um método chamado mudarCor dentro do campo exports do componente, após isso vamos inserir o nome do método mudarCor dentro do evento click. Lembrando que para utilizarmos uma variável criada dentro do campo data() em um componente, temos que utilizar a palavra this.
311. Importante lembrar que todos os eventos retornam um dado, para isso utilizamos $event dentro do campo de () do método
312. //código do componente Cliente
313. <template>
314. <div :class="{'cliente': !isPremiun,'cliente-premium':isPremiun}">
315. <h4>Nome: {{cliente.nome}}</h4>
316. <hr>
317. <p>Email: {{cliente.email}}</p>
318. <p v-if="showIdade == false">Idade: {{cliente.idade}}</p>
319. <p v-else >O Usuário escondeu a idade</p>
320. <button @click="mudarCor($event)">Mudar a cor</button>
321. </div>
322. </template>
323. <script>
324. export default {
325. data(){
326. return {
327. isPremiun: true
328. }
329. },
330. props: {
331. //tipo objeto
332. cliente: *Object*,
333. showIdade: *Boolean*
334. },
335. methods: {
336. mudarCor: *function*(*$event*){
337. console.log(*$event*);
338. this.isPremiun = !this.isPremiun
339. }
340. }
342. }
343. </script>
344. <style scoped>
345. .cliente{
346. *background-color*: #ECE5E3;
347. *max-width*: 600px;
348. *height*: 180px;
349. *padding*: 1%;
350. *margin-top*: 2%;
351. }
352. .cliente-premium{
353. *background-color*: black;
354. *color*: yellow;
355. *max-width*: 600px;
356. *height*: 180px;
357. *padding*: 1%;
358. *margin-top*: 2%;
359. }
361. </style>
362. **Formulário**
363. Para criar um formulário vamos criar três inputs para acrescentar o array clientes os dados de um novo cliente.
364. Vamos criar também um botão que irá chamar no evento click o método cadastrarUsuário.
365. O método cadastrarUsuário realizará um push no array clientes com os dados das variáveis criadas e que foram inseridas nos inputs com v-model.
366. //código do app
367. <h3>Cadastro:</h3>
368. <input type="text" placeholder="nome" v-model="nomeField"><br>
369. <input type="email" placeholder="email" v-model="emailField"><br>
370. <input type="number" placeholder="idade" v-model="idadeField"><br>
371. <button @click="cadastrarUsuario">Cadastrar</button>
372. data(){
373. return {
374. //variáveis temporárias para salvar os dados no array clientes
375. nomeField:"",
376. emailField:"",
377. idadeField: 0,
378. clientes: [
379. {
380. id: 2,
381. nome: "Sthefany S alves",
382. email: "sth@alves",
383. idade: 18
384. },
385. {
386. id: 3,
387. nome: "Carolina S Alves",
388. email: "Carol@alves",
389. idade: 14
390. },
391. {
392. id: 5,
393. nome: "Claudisnei Bello",
394. email: "cba@alves",
395. idade: 48
396. },
397. {
398. id: 8,
399. nome: "Debora Sobrinho",
400. email: "debora@s",
401. idade: 40
402. }
403. ]
404. }
405. },
406. methods:{
407. cadastrarUsuario: *function*(){
408. this.clientes.push({nome: this.nomeField, email: this.emailField, idade: this.idadeField, id: *Date*.now()}),
409. this.nomeField= "",
410. this.idadeField="",
411. this.emailField=""
412. }
413. }
415. }
416. **Validação de formulários**
417. Vamos criar um campo do tipo small no inicio do formulário que irá aparecer em vermelho caso o usuário não digite o nome corretamente.
418. Vamos criar uma variável deuErro do tipo boolean como false
419. No campo small vamos inserir o v-show que irá mostrar o campo small se a variável deuErro for true.
420. Na método cadastrarUsuario vamos incluir um if que verificará se o nome está dentro da regra se não estiver, a variável deuErro será true e aparecerá a mensagem para o usuário
421. <template>
422. <div id="app">
423. <h1>Guia Clientes</h1>
424. <h3>Cadastro:</h3>
425. <small id="nomeErro" v-show="deuErro">O nome é invalido!, tente novamente</small><br>
426. <input type="text" placeholder="nome" v-model="nomeField"><br>
427. <input type="email" placeholder="email" v-model="emailField"><br>
428. <input type="number" placeholder="idade" v-model="idadeField"><br>
429. <button @click="cadastrarUsuario">Cadastrar</button>
430. <div v-for="(cliente,index) in clientes" :key="cliente.id">
431. <h4> {{index + 1}} </h4>
432. <cliente :cliente="cliente"/>
433. <hr>
434. </div>
435. </div>
436. </template>
437. <script>
438. import Cliente from './components/Cliente';
439. export default {
440. name: 'App',
441. components:{
442. Cliente
443. },
444. data(){
445. return {
446. //variáveis temporárias para salvar os dados no array clientes
447. nomeField:"",
448. emailField:"",
449. idadeField: 0,
450. //variavel para verificar se o usuário digitou corretamente o nome do usuário
451. deuErro: false,
452. clientes: [
453. {
454. id: 2,
455. nome: "Sthefany S alves",
456. email: "sth@alves",
457. idade: 18
458. },
459. {
460. id: 3,
461. nome: "Carolina S Alves",
462. email: "Carol@alves",
463. idade: 14
464. },
465. {
466. id: 5,
467. nome: "Claudisnei Bello",
468. email: "cba@alves",
469. idade: 48
470. },
471. {
472. id: 8,
473. nome: "Debora Sobrinho",
474. email: "debora@s",
475. idade: 40
476. }
477. ]
478. }
479. },
480. methods:{
481. cadastrarUsuario: *function*(){
482. //verificando se o campo é vazio
483. if(this.nomeField =="" || this.nomeField==" "|| this.nomeField < 3){
484. //console.log("Erro de validação")
485. this.deuErro= true;
486. }else{
487. this.deuErro=false;
488. this.clientes.push({nome: this.nomeField, email: this.emailField, idade: this.idadeField, id: *Date*.now()}),
489. this.nomeField= "",
490. this.idadeField="",
491. this.emailField=""
492. }
493. }
494. }
496. }
497. </script>
498. <style>
499. #nomeErro{
500. *color*: red;
501. }
502. </style>
503. **Emissão de eventos**
504. //códigos do cliente
505. <button @click="emitirEventoDelete">Deletar</button>
506. emitirEventoDelete: *function*(){
507. //console.log("Emitindo um filho");
508. //utilizamos este atributo para emitir um evento que será chamado meDelete para o app Pai
509. //podemos enviar qualque dados pelo evento
510. this.$emit("meDelete", {idCliente:this.cliente.id,  curso:"Formação nodeJS", emPromocao: true, component: this});
511. },

//códigos do app Pai

1. <cliente :cliente="cliente" @meDelete="deletarUsuario($event)" />
2. deletarUsuario: *function*(*$event*){
3. // console.log("Recebendo evento!");
4. // console.log($event);
5. // console.log($event.component.idCliente);
6. // $event.component.testar();
7. *var* id = *$event*.idCliente;
8. *var* novoArray = this.clientes.filter(*cliente* *=>* *cliente*.id != id)
9. this.clientes = novoArray;
11. }
12. **Filters**
13. Os filtros são utilizados para filtrar informações conforme informado nos parâmetros, sendo parecidos com os métodos. Utilizamos filters() dentro da clausula export defauts. O filtro recebe um valor e deve retornar alguma coisa. Neste caso vamos receber o conteúdo de um email e passar o texto todo para maiúscula. Para aplicar o filtro devemos ir no local onde estamos bydando a variável e inserir um pipe “|” e o filtro neste caso “processarEmail”.
14. <p>Email: {{cliente.email | processarEmail}}</p>
15. filters:{
16. processarEmail: *function*(*value*){
17. return *value*.toUpperCase();
18. }
19. }
20. **Computed properties**
21. Propriedades que são geradas dinamicamente por exemplo um id para o usuário, juntando o email+nome em letra maiúscula. São como métodos que retornam valor.
22. computed:{
23. idEspecial: *function*(){
24. return (this.cliente.email + this.cliente.nome + this.cliente.id).toUpperCase();
26. }
27. }
28. <hr>
29. <h4>Id Especial: {{idEspecial}}</h4>
30. Vamos ordernar os nomes pela ordem alfabética.
31. Para isso vamos instalar a biblioteca lodash.
32. No terminal na pasta do projeto digite : npm i – save lodash.
33. Essa biblioteca possui varias ferramentas como filtros e ordenação.
34. Após a instalação basta ir no componente que possui os dados e importar a biblioteca:
35. Digite : import \_ from ‘lodash’;
36. Vamos criar uma computed para ordernar o array pelo nome do cliente.
37. cumputed:{
38. orderClientes:*function*(){
39. return \_orderBy(this.clientes,['nome'],['asc']);
40. }
41. }
42. Depois disso vamos substituir o array não ordenado clientes no v-for pelo array ordenado orderClientes.
43. <div v-for="(cliente,index) in orderClientes" :key="cliente.id">
44. <h4> {{index + 1}} </h4>
45. <cliente :cliente="cliente" @meDelete="deletarUsuario($event)" />
46. <hr>
47. </div>
48. Para mais informações podemos acessar a página da documentação do lodash: lodash.com.
49. **Utilizando frameworks css no VueJs**
50. O framework mais utilizado é o Buma, pois o bootstrap pode dar interferência no VueJs, porém podemos utilizar também a biblioteca de componentes BootstrapVue.
51. Neste projeto vamos então instalar o Bulma, na pasta do projeto digite:
52. “npm install Bulma”
53. Após a instalação vamos no arquivo main.js e importamos o css do bulma.
54. import "../node\_modules/bulma/css/bulma.css"
55. para testar, vamos na página de documentação do bulma “bulma.io” e vamos copiar o exemplo de um botão para o projeto para ver ser está funcionando.
56. //código inserido no componente principal App
57. <!--Codigo de teste do bulma no projeto-->
58. <div class="buttons">
59. <button class="button is-primary is-light">Primary</button>
60. <button class="button is-link is-light">Link</button>
61. </div>
62. <!--Final do cógido de teste do bulma-->
63. para mais informações o endereço do bulma é “bulma.io”