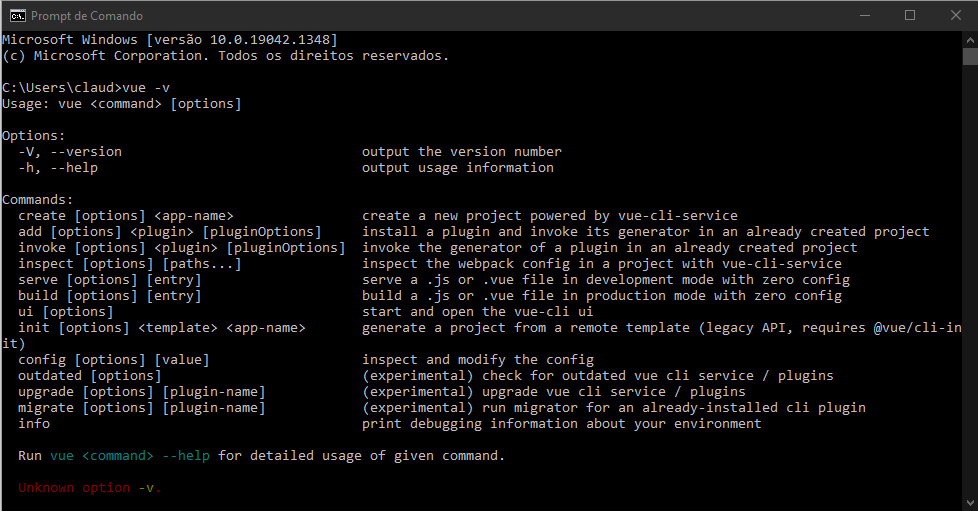
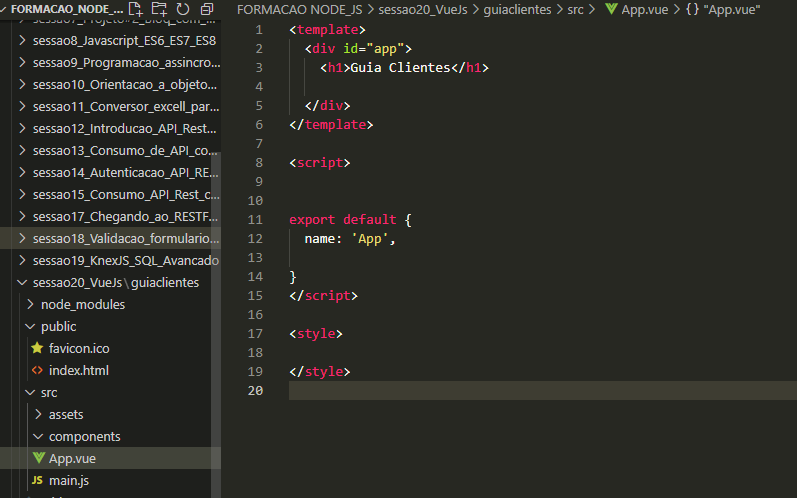
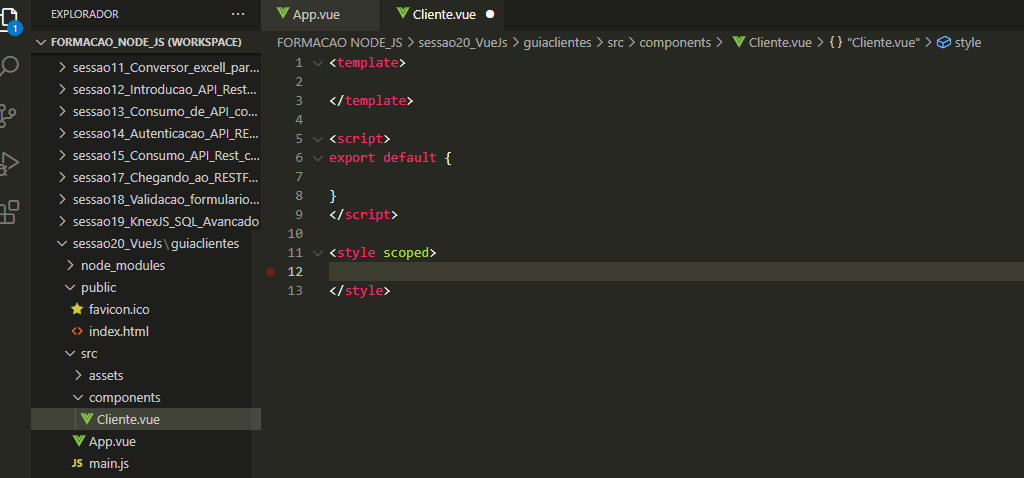
TRABALHANDO COM O FRAMEWORK VUEJS

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.