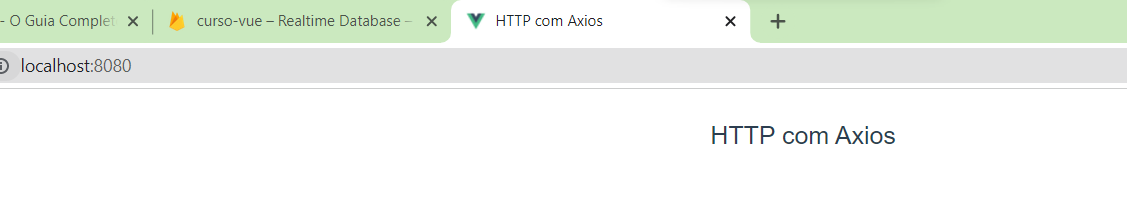
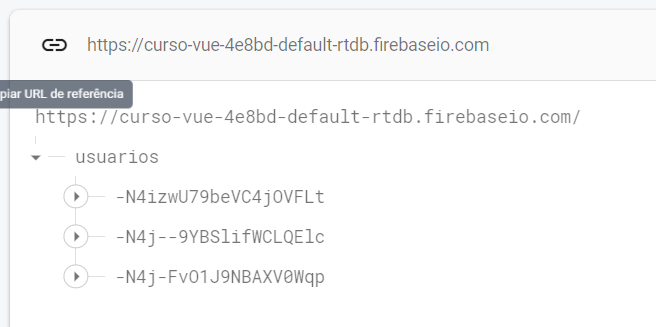
**Conectando com Servidor via HTTP**

1. Vamos utilizar o axios para integrar nossa aplicação com o backend ou seja fazer requisições, vamos utilizar também o firebase como backend na nuvem.

**Aula 236 – Configurando o Firebase**

1. Entre na página <https://console.firebase.google.com/?hl=pt-br>
2. E clique em adicionar projeto.
3. Crie um nome para o novo projeto.
4. Após criado clique em realtime database
5. Clique em criar banco de dados
6. Selecione o local EUA.
7. Clique em iniciar no modo teste.
8. Iremos clicar em regras e vamos alterar as regras para não precisarmos autenticar para realizar alterações no banco de dados.
9. Clique em regras e insira true para .read e true para .write.
10. 
11. Clique em publicar
12. Copie a url do banco de dados criado:
13. <https://curso-vue-4e8bd-default-rtdb.firebaseio.com/>

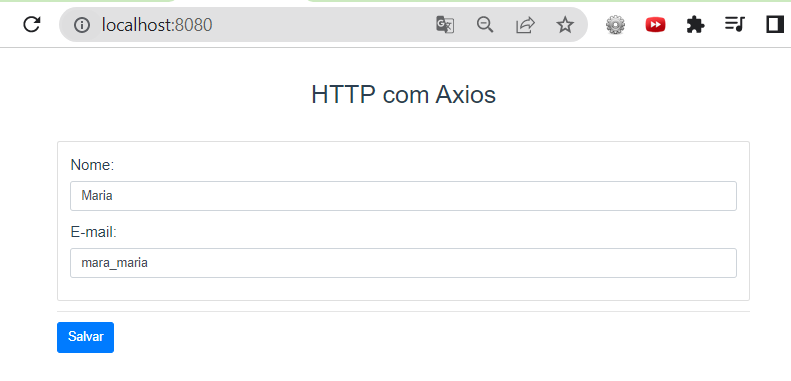
**Aula 237 – Cofiguração global do Axios**

1. Baixe os recursos nesta aula
2. Agora instale o axios com : npm i –- save axios.
3. Será instalado também as dependências que estão faltando.
4. Execute npm run serve para testar.
5. 
6. Existem várias formas de instalar o axios no projeto.
7. Neste projeto iremos registrar de forma local, então iremos dentro da página de plugin e iremos criar um novo arquivo chamado axios.js e iremos realizar importação do vue e o axios.
8. Vamos configurar a url da base de dados com
9. Axios.defaults.baseURL = ‘’
10. Passando o url no banco de dados que criamos.
11. Desta forma configuramos uma propriedade global.
12. import Vue from 'vue'
13. import axios from 'axios'
14. axios.defaults.baseURL= 'https://curso-vue-4e8bd-default-rtdb.firebaseio.com/'
15. Vue.use({
16. install(Vue){
17. Vue.prototype.$http = axios
18. }
19. })
20. Agora vamos no main.js e vamos registrar o plugin importando para o main.
21. import '@babel/polyfill'
22. import Vue from 'vue'
23. import App from './App.vue'
24. import './plugins/bootstrap-vue'
25. import './plugins/axios'
26. Vue.config.productionTip = false
27. **new** Vue({
28. render: *h* => h(App),
29. }).$mount('#app')
30. Dentro de App.
31. Vamos criar o método created() e chamar nele o instância global do axios, chamando o método .post para gravar um usuário.
32. Sendo que o padrão do firebase é utilizar o .json no final.
33. created(){
34. *this*.$http.post('usuarios.json',{
35. nome: 'Maria',
36. email: 'maria\_maria@gmail.com'
37. }).then(*res*=> console.log(*res*))
38. }
39. Ao atualizar a página do firebase o registro já será salvo automaticamente.
40. 

**Aula 238 – Criando instâncias do axios.**

1. Vamos comentar a linha da instância global que criamos do axios, para podermos criar várias instâncias com várias url’s de banco de dados, caso necessário.
2. Dentro da instância do Vue iremos comentar alterar a linha do axios, incluindo um objeto que terá as informações básicas do banco.
3. Desta forma podermos definir quantas instâncias quisermos, basta trocar em prototype.$http o $http pelo nome da nova instância.

**Aula 239 – Criando Formulário**

1. Nesta aula iremos criar um formulário para podermos inserir registros no banco de dados.
2. No campo data() teremos um json chamado usuário com as chaves nome e email.
3. data() {
4. return {
5. usuario:{
6. nome:'',
7. email:''
8. }
9. }
10. },
11. No template iremos utilizar b-card do bootstrap.
12. E dentro do b-card iremos utilizar um b-form-group com dois b-form-input.
13. <template>
14. <div *id*="app" *class*="container">
15. <h1>HTTP com Axios</h1>
16. <b-card>
17. <b-form-group *label*="Nome:">
18. <b-form-input
19. *type*="text"
20. *size*="lg"
21. *v-model*="usuario.nome"
22. *placeholder*="Informe o nome">
23. </b-form-input>
24. </b-form-group>
25. <b-form-group *label*="E-mail:">
26. <b-form-input
27. *type*="email"
28. *size*="lg"
29. *v-model*="usuario.email"
30. *placeholder*="Informe o email">
31. </b-form-input>
32. </b-form-group>
33. </b-card>
34. </div>
35. </template>
36. Vamos adicionar um botão para podermos submeter os dados do formulário.
37. <hr>
38. <b-button
39. *variant*="primary"
40. *size*="lg"
41. @*click*="salvar">Salvar</b-button>
42. 

**Aula 240 – Enviando POST**

1. No método salvar() vamos configurar o post do axios passando o nome do banco de dados e os dados que serão salvos. O método post é uma promisse por isso possui um .then que irá retornar uma res.
2. Na resposta podemos chamar uma outra função,por exemplo para limpar os campos do formulário.
3. salvar(){
4. *this*.$http.post('usuarios.json', *this*.usuario).then(*resp* =>{
5. *this*.usuario.nome = ''
6. *this*.usuario.email = ''
7. })
8. }

**Aula 241 – Enviando via GET**

1. Nesta aula iremos consultar os dados e mostrar na tela.
2. Vamos criar mais um atributo em data() chamado usuários que será um array com todos os usuários do banco.
3. data() {
4. return {
5. usuario:{
6. nome:'',
7. email:''
8. },
9. usuarios:[]
10. }
11. Vamos também incluir mais um botão ao formulário que tá um método obterUsuarios.
12. <b-button
13. *variant*="success"
14. *size*="lg"
15. *class*="ml-2"
16. @*click*="obterUsuarios">Obter Uuário</b-button>
17. Vamos definir o método obterUsuarios() que terá uma chamada http do tipo get do axios.
18. Neste método pegaremos os dados do banco de dados e iremos salvar no array usuários.
19. obterUsuarios(){
20. *this*.$http.get('usuarios.json').then(*resp* => {
21. *this*.usuarios = *resp*.data
22. console.log(*this*.usuarios) })
23. }
24. Vamos criar um b-list-group e um b-list-group-item para utilizarmos um v-for e imprimir na tela todos os registros do banco de dados.
25. <hr>
26. <b-list-group>
27. <b-list-group-item
28. v-for="(usuario, i) in usuarios" :*key*="usuario.i">
29. <span>Usuario: {{usuario.nome}}</span><br>
30. <span>email: {{usuario.email}}</span><br>
31. <span>ID: {{i}}</span><br>
32. </b-list-group-item>
33. </b-list-group>

**Aula 242 – Acessando o Axios localmente**

1. Para acessar o axios localmente, basta realizar a importação do axios para o arquivo app.js
2. E utilizar o axios nas requisições, porém será necessário passar a url completa do banco de dados para funcionar.
3. import axios from 'axios'
4. //AXIOS LOCALMENTE
5. axios.get('https://curso-vue-4e8bd-default-rtdb.firebaseio.com/usuarios.json').then(*resp* => {
6. *this*.usuarios = *resp*.data
7. console.log(*this*.usuarios) })

**Aula 243 – Interceptando requisições**

1. Podemos interceptar as requisições e respostas de forma global no axios.
2. Podemos configurar isso diretamente no plugin que criamos.
3. Utilizamos o método .interceptor.request.use da instância global $http que criamos. Neste método iremos criar uma função que irá retornar as configurações de interceptação de requisição, além de podermos passar alterar as configurações caso necessário.
4. Vue.prototype.$http.interceptors.request.use(*config* => {
6. //exemplo de alteração de configuração.
7. if(*config*.method === 'post'){
8. *config*.method = 'put'
9. }
10. console.log(*config*.method)
11. return *config*
12. })
13. Nessa configuração toda requisição do tipo ‘post’ será convertida para ‘put’.
14. Podemos realizar um tratamento ainda no nosso interceptor, para em caso de erro ele cancelar a requisição.
15. Vue.prototype.$http.interceptors.request.use(*config* => {
17. //exemplo de alteração de configuração.
18. // if(config.method === 'post'){
19. //     config.method = 'put'
20. // }
21. console.log(*config*.method)
22. return *config*
23. }, *error*=> *Promise*.reject(*error*))

**Aula 244 – Interceptando respostas**

1. Vamos utilizar o método .interceptors.response.use e utilizar uma arrow function para configurar a resposta.
2. Nesse caso iremos capturar a reposta criando um array e preenchendo com os dados que foram interceptados. Após isso retornamos o array que criamos como resposta.
3. Vue.prototype.$http.interceptors.response.use(*res* =>{
4. const array = []
5. for(let chave in *res*.data){
7. // ... é chamado operador spredin poderia ser
8. //substituido por nome: res.data[chave].nome
9. // e email: res.data[chave].email
10. array.push({ id: chave, ...*res*.data[chave]})
11. }
12. *res*.data = array
13. return *res*
14. },*error* => *Promise*.reject(*error*))
15. Temos que realizar algumas alterações no App.js para exibir o array que configuramos.
16. <b-list-group>
17. <b-list-group-item
18. v-for="(usuario, id) in usuarios" :*key*="usuario.id">
19. <span>Usuario: {{usuario.nome}}</span><br>
20. <span>email: {{usuario.email}}</span><br>
21. <span>ID: {{usuario.id}}</span><br>
22. </b-list-group-item>
23. </b-list-group>

**Aula 245 – Adicionando Headers globais**

1. É muito comum passarmos token em autenticação que são utilizados no cabeçalho ‘Autrhorization, podemos fazer de forma global.
2. Para isso basta setarmos o método .headers.common
3. axios.defaults.headers.common['Authorization'] = 'abc123'
4. axios.defaults.headers.get['Accepts'] =  'application`/json'
5. mas no nosso projeto estamos utilizando a forma local então configuramos da seguinte maneira.
6. Vue.prototype.$http = axios.create({
7. baseURL: 'https://curso-vue-4e8bd-default-rtdb.firebaseio.com/',
8. headers: {
9. 'Authorization': 'abc123'
10. }
11. })
12. Podemos também incluir o header dentro de um componente.
13. obterUsuarios(){
14. *this*.$http.get('usuarios.json').then(*resp* => {
15. *this*.usuarios = *resp*.data
16. console.log(*this*.usuarios)
18. //configurando de forma local o header 'Autrhorization'
19. *this*.$http.defaults.headers.common['Authorization'] = 'abc123'
20. })

**Aula 246 – Implementando CRUD**

1. Vamos inserir mais uma variável chamada id que receberá o id do usuário selecionado, a principio terá o valor de null.
2. data() {
3. return {
4. usuario:{
5. nome:'',
6. email:''
7. },
8. id: null,
9. usuarios:[]
10. }
11. },
12. Vamos criar um método limpar() para limpar os campos do formulário e setar o id como null.
13. limpar(){
14. *this*.usuario.nome=''
15. *this*.usuario.email=''
16. *this*.id= null
17. },
18. Vamos incluir no template no b-list-group-item dois botões um que será para carregar e outro para excluir.
19. O botão carregar terá um método chamado carregar(id)
20. E botão excluir terá o método chamado excluir(id).
21. <b-list-group-item
22. v-for="(usuario, i) in usuarios" :*key*="usuario.i">
23. <span>Usuario: {{usuario.nome}}</span><br>
24. <span>email: {{usuario.email}}</span><br>
25. <span>ID: {{i}}</span><br>
26. <b-button *variant*="warning" *size*="lg"
27. @*click*="carregar(i)">Carregar</b-button>
28. <b-button *variant*="danger" *size*="lg"
29. @*click*="excluir(i)" *class*="ml-2">Excluir</b-button>
30. </b-list-group-item>
31. O método carregar(id) irá passar o id do list-item para a variável this.id, então iremos carregar a variável this.usuario com os dados do usuários[id].
32. Desta forma os dados serão carregados automaticamente nos campos do formulário.
33. carregar(*id*){
34. *this*.id = *id*
35. *this*.usuario = { ...*this*.usuarios[*id*]}
36. },
37. Ao carregar os dados nos campos do formulário, podemos alterar os dados e salvar no banco de dados.
38. Criamos uma lógica no método salvar() em que é verificado se a variável this.id está setada com um valor ou se está null.
39. Criamos então duas variáveis chamadas método e finalUrl para podermos definir qual método será utilizado o ‘patch’ que realiza alterações nos dados do firebase ou ‘post’que cria um novo elemento no banco de dados.
40. A variável método irá receber ‘patch’ ou ‘post’
41. Conforme o if ternário.
42. const metodo = *this*.id ? 'patch' : 'post'
43. a variável finalUrl irá receber o final da url para podermos setar a url do banco de dados.
44. const finalUrl = *this*.id ? `/${*this*.id}.json` : '.json'
45. após definido o final da url, utilizamos o método global $http.
46. salvar(){
47. // //configuração global do axios
48. // this.$http.post('usuarios.json', this.usuario).then(() =>{
49. //  this.limpar()
50. // })
51. //axios localmente
52. // axios.post('https://curso-vue-4e8bd-default-rtdb.firebaseio.com/usuarios.json', this.usuario).then(resp =>{
53. //  this.usuario.nome = ''
54. //  this.usuario.email = ''
55. // })
56. //utilizando o mesmo método para salvar um registro novo
57. //ou alterar um já existente.
58. const metodo = *this*.id ? 'patch' : 'post'
59. const finalUrl = *this*.id ? `/${*this*.id}.json` : '.json'
60. *this*.$http[metodo](`/usuarios${finalUrl}`, *this*.usuario)
61. .then(()=>{*this*.limpar()})
62. },
63. Temos também o método excluir(id) onde utilizamos .delete do axios para apagar um usuário de id específico do array de usuários.
64. excluir(*id*){
65. *this*.$http.delete(`/usuarios/${*id*}.json`).then(()=>{
66. *this*.limpar()
67. })
68. }

**Aula 247 – Exibindo mensagens**

1. Iremos definir um novo array chamado mensagens, acrescentando o array no método limpar().
2. Esse array terá dois parâmetros, o texto e o tipo sendo o texto a mensagem do erro e o tipo será passado como bind para :variant de um b-alert fazendo com que o b-alert mude conforme a mensagem de erro.
3. Para isso vamos inserir um b-alert no inicio do template. Vamos setar o b-alert com show dismissible para aparecer o ‘x’ para fechar a mensagem.
4. Iremos utilizar um v-for que terá uma mensagem contida no array mensagens.
5. Como dissemos faremos um bind da propriedade :variante passando o tipo da mensagem para ele.
6. Faremos também uma interpolação com mensagem.texto para passar o conteúdo da mensagem para o alert
7. <b-alert *show* *dismissible*
8. v-for="mensagem in mensagens"
9. :*key*="mensagem.texto"
10. :*variant*="mensagem.tipo"
11. >{{mensagem.texto}}</b-alert>
12. <b-card>
13. Estas mensagem devem ser inseridas no tratamento dos métodos, por isso iremos incluir uma lógica para inserir estas mensagems.
14. No método salvar() temos a opção de utilizar os métodos patch ou post do axios, conforme o valor da constante método. Então faremos iremos verificar qual o método que está sendo aplicado e inserir uma mensagem correspondente no array mensagens.
15. if(metodo === 'patch'){
16. *this*.mensagens.push({
17. texto: 'Usuário alterado com sucesso !',
18. tipo: 'warning'
19. })
20. }else{
21. *this*.mensagens.push({
22. texto: 'Usuário cadastrado com sucesso !',
23. tipo: 'success'
24. })
25. }
26. Desta forma se alterarmos um usuário ou criarmos um novo a mensagem será gerada automaticamente.
27. No método de exclusão iremos fazer da mesma forma, porém iremos utilizar um tratamento de erro com catch passando para o array de mensagem um erro personalizado.
28. excluir(*id*){
29. *this*.$http.delete(`/usuarios/${*id*}.json`).then(()=>{
30. *this*.limpar()
31. *this*.mensagens.push({
32. texto: 'Usuário apagado !',
33. tipo: 'info'
34. })
35. }).catch(*err* =>{
36. *this*.mensagens.push({
37. texto: 'Erro ao excluir usuário !',
38. tipo: 'danger'
39. })
40. })
41. }