**PROJETO STOCK TRADER**

1. Neste projeto iremos simular um ambiente de compra e vendas de ações, vamos utilizar o vuetify para nos auxiliar na criação do css.
2. Entre na pasta onde ficará o projeto e digite vue create stock-trader e selecionar quais as dependências que serão criadas no inicio do projeto depois disso instalamos o vuetify porém a instalação deve ser na pasta raiz do projeto e náo em src. Instalamos com vue add vuetify.

**Aula 305 – Criando os primeiros componentes**

1. na pasta src crie outra pasta chamada componentes onde iremos criar nossos componentes.
2. Crie o componente Header.vue que será o cabeçalho da aplicação.
3. Crie o componente Home.vue que será a pagina inicial.
4. Dentro da pasta components crie outra pasta chamada portfolio.
5. Destro desta pasta crie Portfolio.vue e Stock.vue
6. Crie outra pasta dentro da pasta componentes, chamada stocks.
7. E dentro dela crie os componentes Stock.vue e Stocks.vue
8. O componente Portfolio irá mostrar todas as ações que o usuário tem em sua carteira de ações.
9. O componente Stocks.vue irá mostrar todas as ações disponíveis no mercado.
10. O componente Stock.vue da pasta portfolio será utilizado para vender a ação.\
11. O componente Stock.vue da pasta stocks será utilizado para comprar ações.

**Aula 306 – Configurando as rotas do projeto**

1. Vamos instalar a dependência do vue-router com : npm i –save vue-router.
2. Vamos configurar as rotas, no arquivo router.js vamos importar os componentes que serão utilizados para navegar no sistema.
3. Home.vue que vai mostrar as mensagem do inicio e saldo do usuário.
4. Portfolio vai mostrar as ações compradas pelo usuário.
5. Stocks ira mostrar as ações cadastradas no sistema.
6. import Vue from 'vue'
7. import Router from 'vue-router'
8. import Home from '../components/Home'
9. import Portfolio from '../components/portfolio/Portfolio'
10. import Stocks from '../components/stocks/Stocks'
11. Vamos registrar o Router
12. Vue.use(Router)
13. Exportamos a nova instância de Router utilizando o parâmetro mode: ‘history’ para não aparecer os hashs na url.
14. E também iremos utilizar o parâmetro routes:[] onde serão passados as rotas.
15. export default **new** Router({
16. mode: 'history',
17. routes:[
18. {path: '/', component: Home},
19. {path: '/portfolio', component: Portfolio},
20. {path: '/stocks', component: Stocks}
21. ]
22. })
23. Vamos agora configurar o App.vue
24. Vamos registrar alguns componentes do Vuetify, podemos consultar os componentes deste framework em <http://vuetifyjs.com>
25. Dentro de App.vue vamos adicionar um <v-app>, dentro dele é onde serão inseridos todos os componentes.
26. Vamos inserir um <v-content> que será nosso conteúdo dentro de dele iremos definir um <v-container> onde ficarão nossos componentes, dentro dele iremo inserir o <router-view> que irá mostrar os componentes.
27. Para testar devemos ainda importar e registrar o router dentro do arquivo main.js.
28. import Vue from 'vue'
29. import App from './App.vue'
30. import './registerServiceWorker'
31. import router from './router/router'
32. import store from './store/store'
33. import vuetify from './plugins/vuetify'
34. Vue.config.productionTip = false
35. **new** Vue({
36. router,
37. store,
38. vuetify,
39. render: *h* => h(App)
40. }).$mount('#app')
41. O próximo passo será inserir um cabeçalho na aplicação e nele terá os links para navegação entre os componentes.
42. Aula 307 – Adicionando Cabeçalho e Navegação
43. Vamos importar para o arquivo App.js o componente Cp-Header.
44. <script>
45. /\* eslint-disable \*/
46. import Header from './components/Cp-Header.vue'
47. export default {
48. components:{ Header }
49. }
50. </script>
51. Depois vamos inserir o <Header> antes do <v-content> dessa forma ele será mostrado antes de todas as rotas.
52. <template>
53. <v-app>
54. <Header></Header>
55. <v-content>
56. Agora vamos abrir nosso componente Cp-Header.vue e vamos configurá-lo conforme modelo do vuetify.
57. Podemos ver vários modelos de Toobars na página do vuetify.
58. Dentro do template vamos inserir um <v-toobar app>
59. Informando que ele será utilizando um <v-app>
60. Vamos inserir um titulo com <v-toolbar-title> os itens do toolbar com <v-toolbar-item> dentro do item vamos adicionar alguns botões com <v-btn>, dentro do <v-btn> iremos utilizar a propriedade to=’ ‘ informando a rota que será executada, desta forma não será necessário utilizar o <router-link>.
61. Vamos inserir então um <v-btn> para a rota principal , outro para a rota ‘/portfolio’ outro para ‘/stocks’ e assim por diante.
62. Vamos inserir um espaçamento com <v-spacer>
63. Depois vamos inserir um novo <v-toolbar-items>
64. E dentro dele vamos inserir um botão para finalizar o dia <v-btn flat>
65. E um menu com <v-menu> que terá um <template> e dentro um <v-btn flat com slot=’activator’>Salvar & Carregar</v-bnt>
66. Que irá abrir as opções cadastradas no menu.
67. Essas opções serão cadastradas em um v-list
68. <v-list> dentro do v-list terá um v-list-item e dentro do <v-list-tile> terá o <v-list-tile-title> que terá o texto da chamada.
69. <template>
70. <v-app-bar
71. *app*
72. >
73. <v-toolbar-title *class*="headline text-uppercase mr-4">
74. <span>Stock</span>
75. <span *class*="font-weight-light">Trader</span>
76. </v-toolbar-title>
77. <v-toolbar-items>
78. <v-btn *text* *to*="/">Início</v-btn>
79. <v-btn *text* *to*="/portfolio">Portfólio</v-btn>
80. <v-btn *text* *to*="/stocks">Ações</v-btn>
81. </v-toolbar-items>
82. <v-spacer></v-spacer>
83. <v-toolbar-items>
84. <v-btn *text*>Finalizar Dia</v-btn>
85. <v-menu *offset-y*>
86. <template *v-slot:activator*="{ on }">
87. <v-btn *text* *v-on*="on">Salvar & Carregar</v-btn>
88. </template>
89. <v-list>
90. <v-list-item>
91. <v-list-item-title>Salvar Dados</v-list-item-title>
92. </v-list-item>
93. <v-list-item>
94. <v-list-item-title>Carregar Dados</v-list-item-title>
95. </v-list-item>
96. </v-list>
97. </v-menu>
98. </v-toolbar-items>
99. </v-app-bar >
100. </template>
101. <script>
102. export default{
104. }
105. </script>
106. <style>
107. </style>

**Aula 308 – Criando o componente de Ações(Stocks)**

1. Para definirmos um layout na aplicação, devemos ter um <v-container> dentro dele um <v-layout> e dentro dele um <v-flex>.
2. Mesmo que sejam em componentes diferentes.
3. Então no componente App teremos o <v-container>, no componente Cp-Stocks iremos utilizar o <v-layout> e no componente Cp-Stock iremos utilizar o <v-flex>
4. Vamos abrir então o componente Cp-Stock e incluir a tag <v-flex> com a class pr-3 e pb-3 que são padding right e padding bottom.
5. Dentro do <v-flex> iremos criar um <v-cad> com as classes green darken-3 white—text esse <v-cad será responsável pelo título do componente e iremos criar outro v-card responsável pelo corpo do componente.
6. No v-card do título teremos o nome da Ação e o preço da ação.
7. No outro <v-card> teremos a quantidade que o usuário irá selecionar e o botão que ele irá utilizar para comprar a ação.
8. Dentro deste <v-card> iremos colocar um <v-container> com a propriedade fill-heigth para o conteúdo preencher toda a área do <v-card>.
9. Dentro do <v-container> vamos inserir um <v-text-field com label=’Quantidade’ type=’number’. O <v-container> terá também um <v-btn com as classes green darken-3 white—text esse botão terá o texto ‘Comprar’.
10. No <v-flex> do componente iremos utilizar os atributos xs12 (para telas de celular) md6(telas tablet) lg4(telas notebook)
11. <template>
12. <v-flex *class*="pr-3 pb-3" *xs12* *md6* *lg4*>
13. <v-card *class*="green darken-3 white--text">
14. <v-card-title *class*="headline">
15. <strong>Nome <small>(Preco: PREÇO)</small></strong>
16. </v-card-title>
17. </v-card>
18. <v-card>
19. <v-container *fill-height*>
20. <v-text-field *label*="Quantidade" *type*="number"></v-text-field>
21. <v-btn *class*="green darken-3 white--text">Comprar</v-btn>
22. </v-container>
23. </v-card>
24. </v-flex>
25. </template>
26. <script>
27. export default{
29. }
30. </script>
31. <style>
32. </style>
34. Após definir o conteúdo do componente Cp-Stock, vamos abrir o componente Cp-Stocks, vamos importar o componente Cp-Stock e registrar.
35. Vamos definir alguns dados para serem carregados.
36. No template vamos definir um <v-layout row wrap>, dentro do <v-layout> vamos inserir o componente <Stock> utilizando um v-for para carregar todos os dados do array stocks ou seja todas as ações.
37. <template>
38. <v-layout *row* *wrap*>
39. <Stock v-for="stock in stocks" :*key*="stock.id" :*stock*="stock" />
40. </v-layout>
41. </template>
42. <script>
43. import Stock from './Cp-Stock.vue'
44. export default{
45. components: { Stock },
46. data() {
47. return {
48. stocks: [
49. {id: 1, name: 'BMW', price: 110},
50. {id: 2, name: 'Google', price: 200},
51. {id: 3, name: 'Apple', price: 280},
52. {id: 4, name: 'Twitter', price: 83},
53. ]
54. }
55. },
56. }
57. </script>
58. <style>
59. </style>

**Aula 309 – Adicionando o Botão Comprar**

1. No componente Cp-Stock nós iremos criar uma props: chamada stock. Dentro do template iremos interpolar o atributo stock.name e o preço sock.price.
2. Vamos ainda criar uma variável chamada quantity e atribuir essa variável em um v-model.number dentro do campo input do template.
3. Vamos definir um método chamado byStock e vamos associar ao vento click do botão.
4. O método terá um variável chamada order que será um objeto que terá os atributos stockId, stockPrice e stockQuantity esses atributos irão capturar os dados da variável stock.
5. Dentro deste método iremos ainda definir que no final da execução a variável quantity seja setada para zero.
6. No template dentro do button iremos utilizar um bind da propriedade disabled para que quando a quantity for menor ou igual a zero o botão seja desabilitado, ainda dentro da propriedade disabled vamos verificar se o número não é inteiro utilizando a função !Number.isInteger(quantity) e neste caso o botão também ficará desabilitado
7. <template>
8. <v-flex *class*="pr-3 pt-3" *xs12* *md6* *lg4*>
9. <v-card *class*="green darken-3 white--text">
10. <v-card-title *class*="headline">
11. <strong>{{stock.name}} <small>(Preco: {{stock.price}})</small></strong>
12. </v-card-title>
13. </v-card>
14. <v-card>
15. <v-container *fill-height*>
16. <v-text-field *label*="Quantidade" *type*="number"
17. *v-model*.*number*="quantity"></v-text-field>
18. <v-btn *class*="green darken-3 white--text"
19. :*disabled* = 'quantity <=0 || !Number.isInteger(quantity)'
20. @*click*="byStock()">Comprar</v-btn>
21. </v-container>
22. </v-card>
23. </v-flex>
24. </template>
25. <script>
26. export default{
27. props:{
28. stock:{type: *Object*, required: true}
29. },
30. data() {
31. return {
32. quantity: 0
33. }
34. },
35. methods: {
36. byStock(){
37. const order = {
38. stockId: *this*.stock.id,
39. stockPrice: *this*.stock.price,
40. quantity: *this*.quantity
41. }
42. console.log(order)
43. *this*.quantity = 0
44. }
45. },
46. }
47. </script>
48. <style>
49. </style>

**Aula 310 - Configurando o Vuex no projeto**

1. Caso o projeto não tenha sido configurado com o vuex deste o início, podemos realizar a instalação com : npm i –save vuex.
2. Após instalação vamos criar uma pasta chamada store dentro de src. Dentro desta pasta criamos o arquivo store.js. dentro deste arquivo iremos importar u Vue e o Vuex
3. Iremos registrar o Vuex no Vue.
4. Iremos criar uma instância de Vuex.
5. Vamos criar dois módulos um para stocks e outro para portfólio.
6. Então dentro da pasta store vamos criar uma pasta chamada modulo e os arquivos stocksModulo.js e portfolioModulo.js.
7. Dentro do arquivo stocksModulo.js vamos exportar um state que terá um array vazio chamado stocks. Vamos ter também um mutations com um método chamado setStocks que recebe como parâmetro o state e o stocks alterarando o array stocks.
8. Iremos definir duas actions,
9. A action byStock que recebe como parâmetro um commit e a order definida lá no Cp-Stock.
10. O outro action será o initStock que recebe um commit e tem por finalidade inicializar as ações que tem no projeto.
11. Essas ações estão sendo carregadas diretamente através do data() do componente Cp-Stocks.
12. export default{
13. state:{
14. stocks: []
15. },
16. mutations:{
17. setStocks(*state*, *stocks*){
18. *state*.stocks = *stocks*
19. }
20. },
21. actions:{
22. buyStock({*commit*}, *order*){
23. commit()
24. },
25. initStocks({*commit*}){
26. commit('setStocks', )
27. }
28. }
29. }
30. Para levarmos esses dados para dentro do Vuex, iremos criar dentro de src vamos criar uma pasta chamada data.
31. Vamos criar uma arquivo chamado stocksData.js, vamos copiar o array de dados do componente cp-stocks e colar no arquivo stockData.js
32. export default{
33. [
34. {id: 1, name: 'BMW', price: 110},
35. {id: 2, name: 'Google', price: 200},
36. {id: 3, name: 'Apple', price: 280},
37. {id: 4, name: 'Twitter', price: 83},
38. ]
40. }
41. Feito isso vamos agora apagar o método data() dentro de cp-stocks.vue
42. <script>
43. import Stock from './Cp-Stock.vue'
44. export default{
45. components: { Stock },
46. }
47. </script>
48. No modulo stocksModulo.js vamos importar o arquivo dos dados stocksData.js. e passar stocksModulo como parâmetro no método initStocks.
49. import dataStocks from '../../data/stocksData'
50. export default{
51. state:{
52. stocks: []
53. },
54. mutations:{
55. setStocks(*state*, *stocks*){
56. *state*.stocks = *stocks*
57. }
58. },
59. actions:{
60. buyStock({*commit*}, *order*){
61. commit()
62. },
63. initStocks({*commit*}){
64. commit('setStocks', dataStocks)
65. }
66. }
67. }
68. Ainda dentro do modulo stocksModulo iremos ter um getters responsável por obter todas as ações cadastradas dentro do state.
69. getters:{
70. getStocks(*state*){
71. return *state*.stocks
72. }
73. }
74. Agora temos que registrar esse módudo dentro do arquivo store.js.
75. Vamos importar esse módulo e registrar ele dentro do atributo modules:{}.
76. import Vue from 'vue'
77. import Vuex from 'vuex'
78. import stocksModule from '../store/modulos/stocksModulo'
79. Vue.use(Vuex)
80. export default **new** Vuex.Store({
81. modules:{ stocksModule }
82. })
83. Após isso temos que importar e registrar o store dentro do arquivo main.js
84. import Vue from 'vue'
85. import App from './App.vue'
86. // import './registerServiceWorker'
87. import router from './router/router'
88. import store from './store/store'
89. import vuetify from './plugins/vuetify'
90. Vue.config.productionTip = false
91. **new** Vue({
92. router,
93. store,
94. vuetify,
95. render: *h* => h(App)
96. }).$mount('#app')
97. Vamos criar dentro do componente App.vue um método created() para que quando o componente for criado as ações sejam carregadas automaticamente.
98. Dentro deste método iremos chamar a action initStore. Para chamar uma action utilizamos this.$store.dispatch()
99. export default {
100. components:{ Header },
101. created(){
102. *this*.$store.dispatch('initStocks')
103. }
104. }
105. </script>
106. Feito isso vamos no componente cp-stocks.vue e vamos criar uma propriedade computada que irá chamar o getter getStocks para carregar as ações no componente.
107. computed:{
108. stocks(){
109. return *this*.$store.getters.getStocks
110. }
111. }

**Aula 311 – Adicionando o módulo portfolio ao Vuex**

1. O módulo portfólio terá as ações compradas e o saldo em conta.
2. Essas informações faram parte do estado correspondente do state do modulo portfolio.
3. Na pasta módulos vamos criar um arquivo chamado portfolio.js. nele teremos um state que terá funds sendo o valor do saldo em conta e um array de stocks que serão as ações do portfolio.
4. Vamos criar uma mutation que será o byStock que receberá como parâmetro o state e o objeto que representa a ordem de compra.
5. O mesmo que foi definido no método byStock() do componente Cp-Sotck.vue. vamos ao invés de utilizar o objeto como parâmetro, vamos fazer um destruct pegando os atributos diretamente na definição do método.
6. xport default{
7. state{
8. funds: 10000,
9. stocks: []
10. },
11. mutations:{
12. buyStock(state,{ stockId, quantity, stockPrice}) {
13. }
14. }
15. }
16. Podemos ver que o mutation está no modulo portfolio pois ele irá alterar o array de stocks do portfólio, porem a action que dispara o commit está no modulo stocksModulo, porém o estado é modificado somente dentro do módulo do portfólio.
17. Então dentro do mutation buyStocks iremos verificar se o stock que estamos comporando já existe dentro do array stock desta forma se o stock já existir iremos somente acrescentar a sua quantidade e não inserir novamente outro stock igual.
18. Para fazer iremos criar uma variável que irá receber um elemento do array, nessa verificação iremos utilizar o método find de um array verificando se o o id do elemento que estamos procurando está dentro do array stocks.
19. mutations:{
20. buyStock(state,{ stockId, quantity, stockPrice}) {
21. const record = state.stocks.find(*element* => *element*.id == stockId)
22. if(record){
23. record.quantity =+ quantity
24. }else{
25. state.stocks.push({
26. id: stockId,
27. quantity: quantity
28. })
29. }
30. }
31. }
32. O próximo passo é ajustar o saldo para refletir a compra.
33. state.funds -= stockPrice \* quantity
34. agora iremos fazer uma mutation para vender a ação.
35. sellStock(*state*,{ *stockId*, *quantity*, *stockPrice*}){
37. }
38. Para fazer a venda iremos fazer a busca para verificar se existe a ação no portfolio.
39. Depois iremos verificar se quantidade de ações existente é maior que a quantidade que se quer vender e neste caso as ações do portfólio será subtraida
40. Caso contrário iremos vender todas as ações e remover a ação do portfólio.
41. Para remover a ação utilizamos o método splice do array. Passando o índice do elemento no array e a quantidade de elemento que se quer excluir apartir deste índice.
42. sellStock(*state*,{ *stockId*, *quantity*, *stockPrice*}){
43. const record = *state*.stocks.find(*element* => *element*.id == *stockId*)
44. if(record.quantity > *quantity*){
45. record.quantity -= *quantity*
46. }   else{
47. *state*.stocks.splice(*state*.stocks.indexOf(record), 1)
48. }
49. }
50. Agora temos que ajustar o saldo
51. }
52. *state*.funds =+ *stockPrice* \* *quantity*
53. Agora temos que criar a action responsável por vender a ação.
54. actions:{
55. sellStock({*commit*},*order*){
56. commit('sellStock', *order*)
57. }
58. }
59. Agora temos que definir os getters.
60. Vamos criar o getter sotckPortfolio que será responsável por mostrar todas as ações do meu portfólio.
61. Vamos utilizar o método map do array para transformar um array que tem somente dois atributos que são stockId e quantity em um array com todos os atributos.
62. Os parâmetros de entra do getter stockPortfolio será o state mais a lista de todos os getters da aplicação para podermos recuperar o nome e o preço da ação.
63. Vamos utilizar o método map de um array para localizar a ação do array stocks dentro deste array iremos criar uma variável record que irá receber o getter getStocks do módulo stocksModulo. Iremos utilizar nesse getter o método find para localizar a ação no array stocks comparando o stock.id com o element.id. Uma vez encontrado iremos retornar os dados completos da ação.
64. stockPortfolio(*state*, *getters*){
65. return *state*.stocks.map(*stock* => {
66. const record = *getters*.stocks.find(*element* => *element*.id == *stock*.id)
67. return{
68. id: *stock*.id,
69. quantity: *stock*.quantity,
70. name: record.name,
71. price: record.price
72. }
73. })
74. }
75. Teremos outro getter que irá retornar o saldo dentro do portfólio.
76. funds(*state*){
77. return *state*.funds
78. }

**Aula 312 – Trabalhando com ações do portfólio**

1. Para podermos utilizar os módulos temos que registra-los no arquivo store.js
2. import Vue from 'vue'
3. import Vuex from 'vuex'
4. import stocksModule from '../store/modulos/stocksModulo'
5. import portfolioModule from '../store/modulos/portfolioModulo'
6. Vue.use(Vuex)
7. export default **new** Vuex.Store({
8. modules:{ stocksModule, portfolioModule }
9. })
10. Agora iremos implementar a logica para compra de ações, para isso vamos no componente cp-Stock no methodo byStock e vamos chamar a action que foi implementada no modulo stockModulo.
11. *this*.$store.dispatch('buyStock',order)
12. Temos agora que ir no modulo stockModulo e comitar o método ‘buyStock’ e order.
13. buyStock({ *commit* }, *order*){
14. commit('byStock',*order*)
15. },
16. Vamos agora criar o componente para a ação do portfólio.
17. No arquivo cp-stock-portfolio vamos copiar todo o conteúdo do componente cp-stock e realizar algumas alterações, primeiro vamos altera a cor de green para blue, depois vamos alterar o texto do botão para ‘Vender’ e o método para ‘sellStock’.
18. O nome do método será alterado para sellStock e vamos alterar o dispatcth alterando o parâmetro para ‘sellStock’.
19. <template>
20. <v-flex *class*="pr-3 pt-3" *xs12* *md6* *lg4*>
21. <v-card *class*="blue darken-3 white--text">
22. <v-card-title *class*="headline">
23. <strong>
24. {{stock.name}}
25. <small>
26. (Preco: {{stock.price}} | Qtde: {{stock.id}})
27. </small>
28. </strong>
29. </v-card-title>
30. </v-card>
31. <v-card>
32. <v-container *fill-height*>
33. <v-text-field *label*="Quantidade" *type*="number"
34. *v-model*.*number*="quantity"></v-text-field>
35. <v-btn *class*="blue darken-3 white--text"
36. :*disabled* = 'quantity <=0 || !Number.isInteger(quantity)'
37. @*click*="sellStock()">Vender</v-btn>
38. </v-container>
39. </v-card>
40. </v-flex>
41. </template>
42. <script>
43. export default{
44. props: ['stock'],
45. data() {
46. return {
47. quantity: 0
48. }
49. },
50. methods: {
51. sellStock(){
52. const order = {
53. stockId: *this*.stock.id,
54. stockPrice: *this*.stock.price,
55. quantity: *this*.quantity
56. }
57. *this*.$store.dispatch('sellStock',order)
59. // eslint-disable-next-line
60. console.log(order)
61. *this*.quantity = 0
62. }
63. },
64. }
65. </script>
66. <style>
67. </style>

**Aula 313 – Conectando o Portfólio ao Vuex**

1. No componente Cp-Portfólio vamos importar o { mapGetters } para podermos utilizar todos os getters do Vuex e o componente cp-Stock-Portfolio.
2. Vamos declarar o componente.
3. Vamos utilizar também uma propriedade computada que irá mapear o mapGetters.
4. Vamos alterar o nome da propriedade getter ‘stockPorfólio’ para stocks para podermos utilizar a mesma estrutura do componente cp-stocks.
5. <template>
6. <v-layout *row* *wrap*>
7. <Stock v-for="stock in stocks" :*key*="stock.id" :*stock*="stock" />
8. </v-layout>
9. </template>
10. <script>
11. import { mapGetters } from 'vuex'
12. import Stock from './Cp-Stock-Portfolio'
13. export default{
14. components:{ Stock },
15. computed:{
16. ...mapGetters({
17. stocks: 'stockPortfolio'
18. })
19. }
20. }
21. </script>
22. <style>
23. </style>

**Aula 314 – Exibindo o Saldo**

1. Vamos exibir o valor do saldo do cabeçalho e na tela inicial.
2. No componente Home vamos começar definir o template.
3. Vamos inserir uma div e dentro dela um h1, dentro do h1 vamos inserir a classe display-3 que dará um tamanho ao texto e a classe font-weight-light para dar um ‘peso’ leve ao texto e uma margem em baixo com a classe ‘mb-4’, podemos consultar sobre tipografia na página do vuetify. Vamos inserir também um v-sheet que é semelhante ao v-card, porém o v-card tem uma sombra e o v-sheet não, dentro do v-sheet vamos inserir a propriedade ‘:elevation=6’ para dar um aspecto de sombra vamos inserir a classe ‘pa-2’ que dará um espaçamento em todos os lados o ‘a’ é de all.
4. Vamos setar a cor do v-sheet como primary.
5. Como não encontramos um componente que tenha uma mensagem, vamos criar um com esse objetivo.
6. Para isso vamos inserir um v-icon com o texto Info vamos inserir a classe ‘white—text’ e ‘mr-3’ para dar uma margem na direita,
7. Vamos inserir um span com um texto vamos inserir a classe ‘headline’ e White—text para o texto ficar branco e font-weight-light para ficar uma fonte mais leve.
8. Fora do v-sheet vamos inserir um v-divider com a class ‘my-4’ que irá aplicar uma margem superior e inferior do tipo 4.
9. Logo após isso vamos inserir um parágrafo com um Strong com o valor do saldo.
10. Após feito essa configuração vamos copiar o v-sheet alterando a mensagem a cor iremos alterar para success, vamos escurecer um pouco a cor com a classe ‘darken-1’
11. E vamos incluir uma margem superior com a classe ‘mt-3’
12. Agora vamos melhorar o parágrafo do saldo, vamos inserir uma classe ‘display-1’.
13. No parágrafo do fundo vamos interpolar uma variável ‘funds’ e criar dentro de export default uma variável computada funds() que irá retorar o getter de funds que foi criado no modulo ‘portfolioModulo’.
14. export default{
15. state: {
16. funds: 10000,
17. stocks: []
18. },
19. mutations:{
20. buyStock(*state*,{ *stockId*,*quantity*, *stockPrice* }) {
21. const record = *state*.stocks.find(*element* => *element*.id == *stockId*)
22. // console.log('Record: '+record)
23. if(record){
24. record.quantity += *quantity*
25. }else{
26. *state*.stocks.push({
27. id: *stockId*,
28. quantity: *quantity*
29. })
30. }
32. *state*.funds -= *stockPrice* \* *quantity*
33. },
34. sellStock(*state*,{ *stockId*, *quantity*, *stockPrice*}){
35. const record = *state*.stocks.find(*element* => *element*.id == *stockId*)
36. if(record.quantity > *quantity*){
37. record.quantity -= *quantity*
38. }   else{
39. *state*.stocks.splice(*state*.stocks.indexOf(record), 1)
40. }
41. *state*.funds += *stockPrice* \* *quantity*
42. }
43. },
44. actions:{
45. sellStock({ *commit* }, *order*){
46. commit('sellStock', *order*)
47. }
48. },
49. getters:{
50. stockPortfolio(*state*, *getters*){
51. return *state*.stocks.map(*stock* => {
52. const record = *getters*.getStocks.find(*element* => *element*.id == *stock*.id)
53. return{
54. id: *stock*.id,
55. quantity: *stock*.quantity,
56. name: record.name,
57. price: record.price
58. }
59. })
60. },
61. funds(*state*){
62. return *state*.funds
63. }
64. }
65. }
66. Para finalizar iremos também configurar a exibição do saldo no header.
67. No componente cp-Header vamos inserir dentro do segundo v-toolbar-itens um v-layout com ‘align-center’ dentro do v-layout vamos inserir um span com a class ‘text-uppercase’ para o texto ficar com letras maiúsculas uma classe ‘grey—text’ para o texto ficar na cor cinza e um ‘text—darken2’ para o texto ficar um pouco mais escuro.
68. Vamos inserir dentro do span a palavra saldo e interpolar uma variável funds.
69. Agora temos que criar uma propriedade computed dentro de export default. Essa propriedade será funds() e retornará o getter funds criando dentro de portfolioModulo.

**Aula 315 – Adicionando Validações**

1. **Validando saldo insuficiente no componente Cp-Stock.**
2. Vamos fazer algumas validações, primeiramente vamos fazer uma validação para que agente só consiga comprar um número de ações de acordo com o fundo.
3. Vamos entrar dentro da pasta stocks no componente cp-stock.
4. Vamos criar uma propriedade computada chamada funds que irá mapear a o saldo.
5. E outra propriedade será insufficientFunds() que irá determinar se a quantidade saldo é suficiente ou não.
6. computed:{
7. funds(){
8. return *this*.$store.getters.funds
9. },
10. insufficientFunds(){
11. return *this*.quantity \* *this*.stock.price > *this*.funds
12. }
13. },
14. Agora vamos incluir essa validação no v-btn do componente de modo que o botão comprar só ficará habilitado se o resultado da propriedade computada insufficientFunds() for false.
15. Vamos também colocar uma validação no label do botão para que se o insufficientFunds for true o label mude para ‘Insuficiente’
16. <v-btn  *class*="green darken-3 white--text"
17. :*disabled*="insufficientFunds || quantity <= 0 || !Number.isInteger(quantity)"
18. @*click*="buyStock()">
19. {{ insufficientFunds ? 'Insuficiente' : 'Comprar'}}
20. </v-btn>
21. Vamos inserir também uma validação no label do campo v-text-field para que caso insufficientFunds seja true, seja aplicado o formato de erro no input.
22. <v-text-field
23. *label*="Quantidade"
24. *type*="number"
25. :*error*="insufficientFunds"
26. *v-model*.*number*="quantity">
27. </v-text-field>
28. **Validando quantidade de ações no componente Cp-Stock-Portfolio.**
29. Vamos criar uma propriedade comuputada que irá verificar se a quantidade de ações setadas é maior que a quantidade de ações do portfolio.
30. computed:{
31. insufficientQuantity(){
32. return *this*.quantity > *this*.stock.quantity
33. }
34. },
35. Agora vamos inserir dentro do v-btn na propriedade :disabled a condição de insufficientQuantity de maneira que caso a quantidade digitada for maior que a quantidade do portfolio, o botão ficará desabilitado.
36. Vamos incluir também uma lógica para alterar o label do botão, caso a quantidade seja insuficiente o label será alterado para 'Insuficiente'.
37. <v-btn *class*="blue darken-3 white--text"
38. :*disabled* ='insufficientQuantity || quantity <=0 || !Number.isInteger(quantity)'
39. @*click*="sellStock()">
40. {{ insufficientQuantity ? 'Insuficiente' : 'Vender'}}
41. </v-btn>
42. Vamos inserir também uma validação no label do campo v-text-field para que caso insufficientQuantity seja true, seja aplicado o formato de erro no input, de maneira que ele fique vermelho.
43. <v-text-field
44. *label*="Quantidade"
45. *type*="number"
46. :*error*="insufficientQuantity"
47. *v-model*.*number*="quantity">
48. </v-text-field>

**Aula 316 – Criando filtros para mostrar valores monetários**

1. O ideal é criarmos uma pasta para inserir os arquivos de filtros.
2. Vamos porém criar um único filtro dentro do arquivo main.js.
3. O filtro é uma função que recebe um valor e retorna outro.
4. Vamos utilizar o método value.toLocaleString() que colocará os pontos no local correto.
5. Vue.filter('currency', *value*=>{
6. return 'R$' + *value*.toLocaleString()
7. })
8. Agora basta inserirmos esse filtro onde teremos essa finalidade. Vamos abrir o componente Cp-Header e inserir o filtro.
9. <span *class*="text-uppercase grey--text text--darken2">
10. Saldo: {{funds || currency}}
11. </span>
12. Vamos abrir também o componente Cp-Home
13. <p *class*="display-1">
14. <strong>Seu Saldo :</strong>{{ funds | currency}}
15. </p>
16. Vamos inserir também no componente Cp-Stock
17. <v-card-title *class*="headline">
18. <strong>{{stock.name}} <small>(Preco: {{stock.price | currency}})</small></strong>
19. </v-card-title>
20. e Cp-Stock-Portfolio.
21. <small>
22. (Preco: {{stock.price | currency}} | Qtde: {{stock.quantity}})
23. </small>

**Aula 317 – Finalizando Dia – Alterando preço das ações**

1. Vamos criar uma função para finalizar o dia para variar o preço da ação.
2. Vamos criar dentro do modulo stockModulo uma nova mutation que será responsável por randomizar o preço das ações do array que foi importado.
3. Essa mutation irá receber o state e faremos um forEach para o array stocks dentro do forEach iremos alterar o preço aredondando a multiplicação entre o preço atual e um fator que será o resultado da soma de 1 mais um valor aleatório que varia de 0 a 9.9 subtraindo de 0.45.
4. randomizeStocks(*state*){
5. *state*.stocks.array.forEach(*stock* => {
6. *stock*.price = Math.round(*stock*.price \* (1 + Math.random() - 0.45))
7. });
8. }
9. E também uma action que fará a ação de mudar o preço.
10. randomizeStocks({*commit*}){
11. commit('randomizeStocks')
12. }
13. Dentro do componente Cp-Header vamos importar o mapAction do vuex.
14. import { mapActions } from 'vuex'
15. Dentro do methods vamos realizar um spread do mapActions capturando o action ‘randomizeStrocks’ e um método chamado endDay().
16. Dentro do método endDay() iremos utilizar o método do mapAction, neste caso , this.randomizeStocks()
17. methods:{
18. ...mapActions(['randomizeStocks']),
19. endDay(){
20. *this*.randomizeStocks()
21. }
22. }
23. Feito isso iremos incluir no botão Finalizar Dia o método endDay().
24. <v-btn @*click*="endDay()" *text*>Finalizar Dia</v-btn>

**Aula 318 – Animando a Transição de Rotas**

1. No componente App vamos colocar o <router-view> entre a tag de transition.
2. Devemos passar para a tag transition qual o tipo de transição pelo name, além de utilizar o mode com ‘out-in’ pois dessa forma primeiro uma transição termina e depois outra inicia.
3. <transition *name*="slide" *mode*="out-in">
4. <router-view></router-view>
5. </transition>
6. Agora temos que configurara dentro de <style> o @keyFrames slide-in utilizando from onde a transformação irá utilizar um translateY(-30px) com opacidade 0. E indo para (to) transformando translateY(0px) com opacidade 1.
7. Temos que configurar também o slide-out com os valores invertidos.
8. <style>
9. @keyframes *slide-in*{
10. from{ transform: translateY(-30px); opacity: 0;}
11. to { transform: translateY(0px); opacity: 1;}
12. }
13. @keyframes *slide-out*{
14. from{ transform: translateY(0px); opacity: 1;}
15. to { transform: translateY(-30px); opacity: 0;}
16. }
17. </style>
18. Temos ainda que configurar a classe .slide-enter-active
19. Utilizando o tipo de animação o tempo de transição e o algoritmo utilizado. Isso será na entrada da transição.
20. Na saída da transição utilizamos a classe .slide-leave-active com os parâmetros similares porém com slide-out.
21. *.slide-enter-active*{
22. animation: slide-in 0.3s ease;
23. }
24. *.slide-leave-active*{
25. animation: slide-out 0.3s ease;
26. }

**Aula 319 – Configurando o axios & Firebase**

1. Vamos para a aplicação e instalar o axios com : npm i – save axios.
2. Após instalar o axios iremos configura-lo de forma global, para isso vamos criar dentro da pasta plugin o arquivo axios.js
3. Vamos importar o Vue e o axios e dentro da instância Vue vamos utilizar o método install
4. import Vue from 'vue'
5. import axios from 'axios'
6. Vue.use({
7. install(Vue){
8. Vue.prototype.$http = axios.create({
9. baseURL: '?'
10. })
11. }
12. })
13. Feito isso vamos entrar no Firebase e criar um novo projeto
14. Com o nome stock-trader iremos configur como real time.
15. Dentro do menu Criação selecione Realtime Database, clique em Criar Banco de dados
16. Selecione Iniciar no modo de teste e clique em ativar.
17. Copie o link <https://stock-trader-2f245-default-rtdb.firebaseio.com> e cole no campo baseUrl do arquivo axios.js
18. import Vue from 'vue'
19. import axios from 'axios'
20. Vue.use({
21. install(Vue){
22. Vue.prototype.$http = axios.create({
23. baseURL: 'https://stock-trader-2f245-default-rtdb.firebaseio.com/'
24. })
25. }
26. })
27. Temos ainda que acessar as regras do Database do Firebase e alterar as permissões para true, porém essa configuração não deve ser realizada para ambiente de produção

**Aula 320 – Salvando Dados (Requisição PUT)**

1. Como já configuramos o axios com o banco de dados real time, vamos entrar no componente Cp-Header pois é nele que temos os botões para salvar os dados e Carregar os dados.
2. Então em methods vamos criar outro método chamados saveData() que terá as constantes funds, stocksPortfolio e getStocks que são getters que criamos nos módulos portfolioModulo e stocksModulo.
3. saveData(){
4. const{ funds, stockPortfolio, getStocks} = *this*.$store.getters
5. }
6. Vamos chamar o método put do $http passando para ele o formato e os valores ou arrays que serão salvos.
7. saveData(){
8. const{ funds, stockPortfolio, getStocks} = *this*.$store.getters
9. *this*.$http.put('data.json',{ funds, stockPortfolio, stocks: getStocks})
10. }
11. Após isso vamos no v-list-title onde está localizado ‘Salvar Dados’ no menu e incluímos o evento @click chamando o método saveData
12. <v-list>
13. <v-list-item @*click*="saveData">
14. <v-list-item-title>Salvar Dados</v-list-item-title>
15. </v-list-item>
16. <v-list-item >
17. <v-list-item-title>Carregar Dados</v-list-item-title>
18. </v-list-item>
19. </v-list>

**Aula 321 – Obtendo dados – requisição GET**

1. Para fazermos isso teremos que ir no módulo de portfolio e realizar alterações.
2. Iremos criar uma mutations para setar o saldo e as ações do portfolio.
3. A mutations será setPortfolio e receberá como parâmetro o state e o stockPortfolio (ambos são getters).
4. No método faremos iremos acessar o state e iremos realizar uma consulta o getters funds passando o valor para o state.funds
5. Iremos fazer o mesmo com stock realizando a consulta ao getter stockPortfolio.
6. setPortfolio(*state*, *stockPortfolio*){
7. *state*.funds = *stockPortfolio*.funds
8. *state*.stocks = *stockPortfolio*.stockPortfolio ? *stockPortfolio*.stockPortfolio:[]
9. }
10. Temos um método similar no modulo stocksModulo que seta também os valores para o array stocks do state.
11. setStocks(*state*, *stocks*){
12. *state*.stocks = *stocks*
13. },
14. Agora que temos duas mutations que irão alterar os state dos dois módulos temos que ter uma action que irá disparar essas mutations. Para fazermos isso iremos criar um arquivo chamado action.js dentro da pasta store, para criar a action neste arquivo, ou seja, fora dos dois módulos onde serão realizadas as alterações.
15. Dentro do arquivo actions.js iremos criar um método chamado loadData que terá como parâmetro um commit que será responsável por acessar todas as mutations que essa actions ira alterar.
16. Iremos acessar os dados externos através de uma requisição do tipo get utilizando uma função assíncrona do Vue.
17. Esta função utiliza o prototype chamando o método $http.get .
18. Dentro desta função esperamos receber uma resposta que será atribuído a uma constante ‘data’.
19. Se data existir iremos realizar o um commit passando a mutation ‘setStocks’ ,que está dentro do módulo stockModulo, passando para a mutation o resultado de data.stocks
20. Faremos outro commit passando também a mutation ‘setPortfolio’, definida dentro do modulo portfolioModulo, passando para a mutation dois resultados data.funds e data.stockPortolio. Esses resultados serão atribuídos diretamente nos getters do modulo portfolioModulo.
21. import Vue from 'vue'
22. export default{
23. loadData({*commit*}){
24. Vue.prototype.$http.get('data.json').then(*resp* =>{
25. const data = *resp*.data
26. if(data){
27. commit('setStocks',data.stocks)
28. commit('setPorfolio', {
29. funds: data.funds,
30. stockPortfolio: data.stockPortfolio
31. })
32. }
33. })
34. }
35. }
36. Feito isso temos que importar dentro de store pois ele ainda não conhece esses métodos que criamos.
37. Vamos fazer a importação dentro do arquivo store.js
38. import Vue from 'vue'
39. import Vuex from 'vuex'
40. import actions from './actions'
41. import stocksModule from '../store/modulos/stocksModulo'
42. import portfolioModule from '../store/modulos/portfolioModulo'
43. Vue.use(Vuex)
44. export default **new** Vuex.Store({
45. actions,
46. modules:{ stocksModule, portfolioModule }
47. })
48. Temos ainda que passar o método criado para o menu ‘Carregar Dados’ do componente cp-Header.
49. Agora dentro de methods temos que fazer mais um mapeamento incluindo a action ‘loadData()’ dentro do spread mapActions.
50. Temos também que criar o método que irá chamar essa action.
51. Porém não pode ter o mesmo nome da action.
52. Vamos criar então o método loadDataLocal que irá chamar a action loadData.
53. methods:{
54. ...mapActions(['randomizeStocks','loadData']),
55. endDay(){
56. *this*.randomizeStocks()
57. },
58. saveData(){
59. const{ funds, stockPortfolio, getStocks} = *this*.$store.getters
60. *this*.$http.put('data.json',{ funds, stockPortfolio, stocks: getStocks})
61. },
62. loadDataLocal(){
63. *this*.loadData()
64. }
65. }
66. Agora vamos incluir o evento @click no menu Carregar Dados
67. <v-list-item  @*click*="loadDataLocal">
68. <v-list-item-title>Carregar Dados</v-list-item-title>
69. </v-list-item>
70. Projeto finalizado.