**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. }