



UNIVERSIDADE FEDERAL DE SÃO PAULO

Departamento de Ciência e Tecnologia

Bacharelado em Ciência da Computação

DOUGLAS DINIZ LANDIM RA 76681 DDLANDIM@UNIFESP.BR

SISTEMA DE PEDIDOS WEB POR THREADS

Proposta de projeto final da disciplina de

Paradigmas de Programação

Prof. Dr. Vinícius Veloso de Melo

2º Semestre de 2018

São José dos Campos, Novembro de 2018.

1. DESCRIÇÃO

O sistema apresentado consiste em um sistema web, elaborado no framework Vue.js elaborado com com linguagem javascript executada em um interpretador nodejs, que trabalha com funções nativas de programação concorrente com promisses.

Node.js é um interpretador de código JavaScript com o código aberto, focado em migrar o Javascript do lado do cliente para servidores.

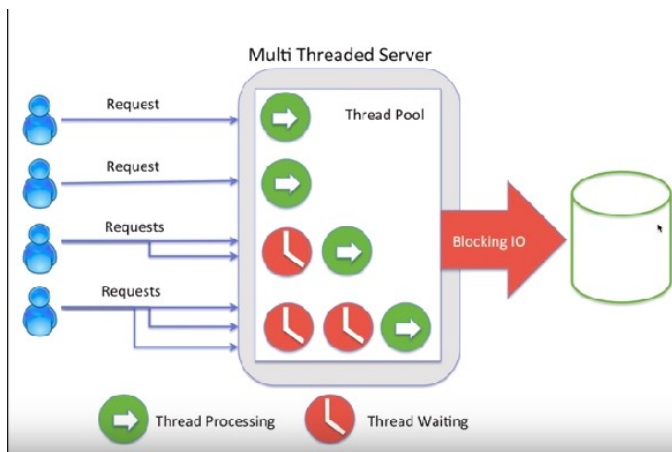


Figura1: Problema de espera de conclusão de threads de usuários em um servidor web.

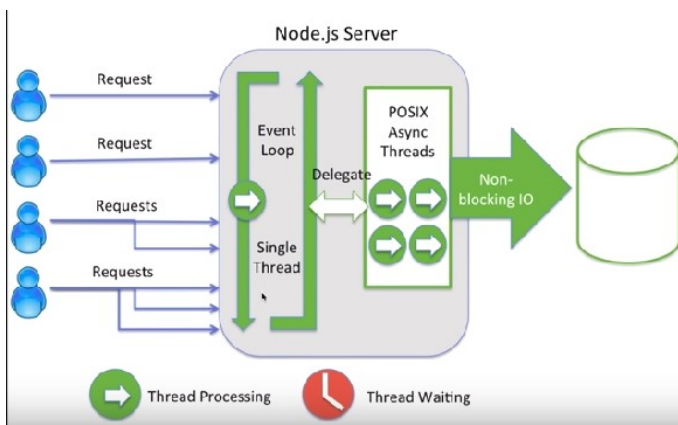
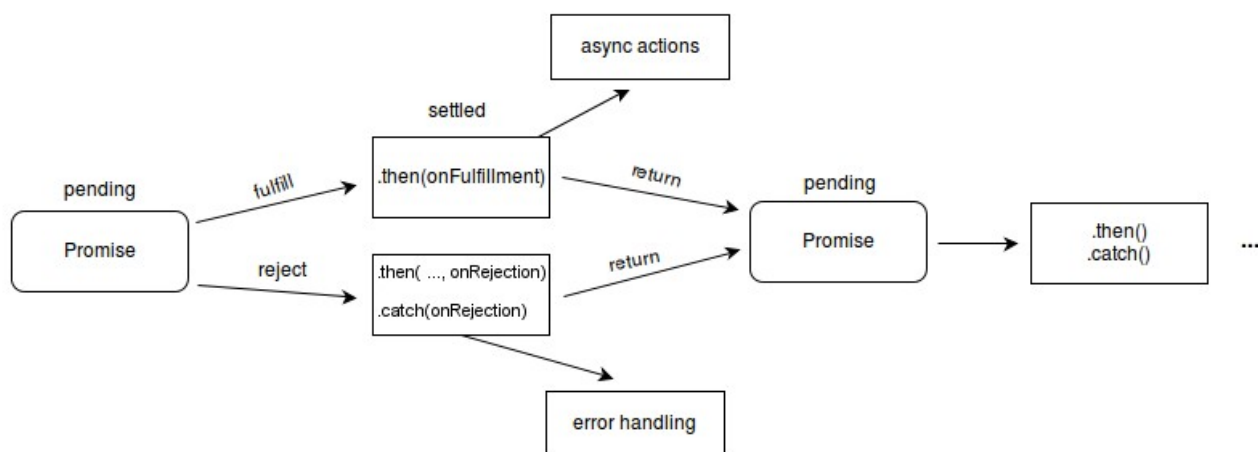


Figura2: Solução do nodejs para tratar threads em espera.

As promissas em javascript fornecem uma alternativa mais simples para executar, compor e gerenciar operações assíncronas quando comparadas a abordagens tradicionais baseadas em retorno de chamada. Elas também lidam com erros assíncronos usando abordagens semelhantes a try / catch síncrono em java.

A API primária de uma promise é o método then, que registra retornos de chamada para receber o valor eventual ou a razão pela qual a promessa não pode ser atendida.

Um aspecto poderoso das promessas é permitir que você transforme os valores futuros retornando um novo valor da função de retorno de chamada transmitida para o momento



Fonte:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

Já o framework Vue.js é uma estrutura JavaScript de código aberto para criar interfaces com o usuário e aplicativos de página única, oferecendo recursos escaláveis e métodos de transformar muitos recursos de um sistema front-end em objetos, facilitando assim o reuso de código e aperfeiçoando a velocidade de desenvolvimento.

Referencia: <https://www.devmedia.com.br/vue-js-tutorial/38042>

<https://github.com/vuejs>

História:

O Vue foi criado por Evan You depois de trabalhar para o Google usando o AngularJS em vários projetos. Mais tarde, ele resumiu seu processo de pensamento: "Eu imaginei: e se eu pudesse simplesmente extrair a parte que eu realmente gostei sobre o Angular e construir algo realmente leve?" [6] O Vue foi originalmente lançado em fevereiro de 2014.

<https://en.wikipedia.org/wiki/Vue.js>

INSTALAÇÃO DO PROJETO:

Sistema operacional homologado:

Linux Ubuntu, 16.

Código fonte:

<https://github.com/ddlandim/cs-unifespbr-paradigmas/tree/master/projetoFinal/restaurante-controle-comandas-vuejs>

Estrutura de arquivos do sistema:

- **/json-server** *contêm um único arquivo: info.json com um banco de dados inicial com exemplos de pedidos e produtos cadastrados para a demonstração do sistema.
- **/restaurante-gestao** *pasta principal do projeto

Instalação dos gerenciadores de pacotes:

- **sudo apt-get install npm**
- **sudo apt-get install yarn**

Instalação do banco de dados:

vá até o diretório - /json-server, abra um terminal e instale primeiramente o json-server:

\$ sudo npm install -g json-server

em seguida inicie o arquivo de banco de dados:

json-server info.json

mantenha o terminal aberto para o servidor ficar em execução.

Instalação do projeto:

vá até o diretório -/restaurante-gestao, abra e instale o interpretador nodejs:

\$ sudo apt-get install nodejs (comando para ubuntu e <node> para outras distribuições linux).

Em seguida instale as dependências do projeto:

- \$ sudo npm install

E por fim execute o projeto:

- npm run dev

APRESENTAÇÃO DO SISTEMA:

A página web tem 2 versões: Frente de Caixa, e Frente de garçom.

Versão frente de caixa, link: <http://localhost:8080/visao-geral>

Numero	Nº de Pessoas	Valor	Ação
3	0	R\$0	<button>EDITAR</button> <button>FECHAR</button>
5	3	R\$360	<button>EDITAR</button> <button>FECHAR</button>
6	4	R\$225	<button>EDITAR</button> <button>FECHAR</button>
7	3	R\$675	<button>EDITAR</button> <button>FECHAR</button>
8	2	R\$90	<button>EDITAR</button> <button>FECHAR</button>

Na frente de caixa, o front-end é exibido trazendo uma visão geral de todas as mesas do estabelecimento, com o logotipo do estabelecimento, carregado no diretório:

`/restaurante-gestao/src/assets/img/logo.png`

No topo esquerdo da tela principal, o bloco Comandas em Aberto, exibe o número de mesas em aberto no restaurante.

No topo direito da tela principal, o bloco Faturamento do Dia, exibe a soma de todos os pedidos realizados desde a abertura do sistema.

A lista no centro da tela, contém um link para fechar uma mesa, e editá-la, indo para a página de front do garçom.

O botão adicionar adiciona novas mesas.

Quando um produto (imagem) for arrastado para esta área da tela, uma segunda thread que ficará lendo esta lista de produtos dessa área, gravará as imagens desta em uma pasta de saída.

Logo será possível acessar uma pasta de usuário e ver através das imagens gravadas ali, quais itens ele fez o pedido.

A front-end do garçom, localizado no link, <http://localhost:8080/editar-mesa>, é visualizada conforme imagem abaixo:

Lançamentos

Produtos:

Quantidade:

Numero de Pessoas na Mesa:

ADICIONAR

Pagamento

Total à pagar:

Total à pagar por pessoa:

Tipo de Pagamento:

Quantidade de pessoas:

PAGAR

Produtos Consumidos **SALVAR**

Nome	Qtd	Valor	Ação
Mr. Burguer	5	R\$34	REMOVER
Mr. Burguer	5	R\$34	REMOVER

O bloco de lançamentos é manipulável.

Uma lista de produtos é obtida do banco de dados (json-server), com a lista de produtos e preços carregados por cada um.

A informação de quantidade é apenas um multiplicador de produtos pedidos.

Número de pessoas na mesa é um valor inteiro, útil para o momento do pagamento.

No bloco de pagamento é possível escolher pagamento total ou parcial.

Se escolhida a opção total, todo o total de pedidos da mesa será cobrado, independente da quantidade de pessoas presentes na mesa

Se escolhida a opção parcial, e selecionado a quantidade de pessoas que irão fazer o pagamento parcial, uma informação do valor cobrado por pessoa neste momento será exibido. O valor referente será debitado da mesa, deixando-o aberta com os participantes restantes.

Pagamento

Total à pagar:

Total à pagar por pessoa:

Tipo de Pagamento:

Quantidade de pessoas:

PAGAR

localhost:8080/editar-mesa

utilitarios | pessoal | profissio

localhost:8080 diz
Valor parcial referente a undefined pessoas R\$ 45.00

OK

Lançamentos

Produtos

Quantidade

Numero de Pessoas na Mesa:

ADICIONAR

R\$90

Total à pagar por pessoa:

Tipo de Pagamento:

Quantidade de pessoas:

PAGAR

PRODUTOS CONSUMIDOS

SALVAR

Nome	Qtd	Valor	Ação
8	1	R\$45	<div>EDITAR</div> <div>FECHAR</div>

Lançamentos

Produtos

Quantidade

Numero de Pessoas na Mesa:

ADICIONAR

Pagamento

Total à pagar:

Total à pagar por pessoa:

Tipo de Pagamento:

Quantidade de pessoas:

PRODUTOS CONSUMIDOS

Nome	Qtd	Valor	Ação
Mr. Bacon	2	R\$45	<div>REMOVER</div>

2. EXPLORANDO O CÓDIGO:

Na raiz do projeto, é visualizado a página index.html carregada pelo navegador, na maior parte de projetos web esta página contém a principal lista de arquivos e páginas a serem carregadas, mas como o nodejs usa uma thread para renderizar o front end, usando uma hierarquia definida no arquivo de rotas, esta página index.html contém apenas as seguintes informações:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>restaurante-gestao</title>
  </head>
  <body>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

O arquivo na raiz do projeto, **package.json** também é carregado ao dar o comando **npm install** para carregar as bibliotecas principais do projeto:

```
{
  "name": "restaurante-gestao",
  "version": "1.0.0",
  "description": "A Vue.js project",
  "author": "ddlandim <ddlandim@unifesp.br>",
  "private": true,
  "scripts": {
    "dev": "webpack-dev-server --inline --progress --config build/webpack.dev.conf.js",
    "start": "npm run dev",
    "lint": "eslint --ext .js,.vue src",
    "build": "node build/build.js"
  },
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^1.2.2",
    "@fortawesome/free-solid-svg-icons": "^5.2.0",
    "@fortawesome/vue-fontawesome": "^0.1.1",
    "axios": "^0.18.0",
    "fontawesome": "^4.7.2",
    "i": "^0.3.6",
    "npm": "^6.3.0",
    "vue": "^2.5.2",
    "vue-router": "^3.0.1"
  },
  "devDependencies": {
    "autoprefixer": "^7.1.2",
    "babel-core": "^6.22.1",
    "babel-eslint": "^8.2.1",
    "babel-helper-vue-jsx-merge-props": "^2.0.3",
    "babel-loader": "^7.1.1",
    "babel-plugin-syntax-jsx": "^6.18.0",
    "babel-plugin-transform-runtime": "^6.22.0",
    "babel-plugin-transform-vue-jsx": "^3.5.0",
    "babel-preset-env": "^1.3.2",
    "babel-preset-stage-2": "^6.22.0",
```


/router/index.js

Neste arquivo é carregado as rotas e hierarquia de páginas do projeto:

```
import Vue from 'vue';
import Router from 'vue-router';
import DashboardLayout from '@/components/DashboardLayout';
import VisaoGeral from '@/components/VisaoGeral';
import EditarMesa from '@/components/Mesa/EditarMesa/EditarMesa';

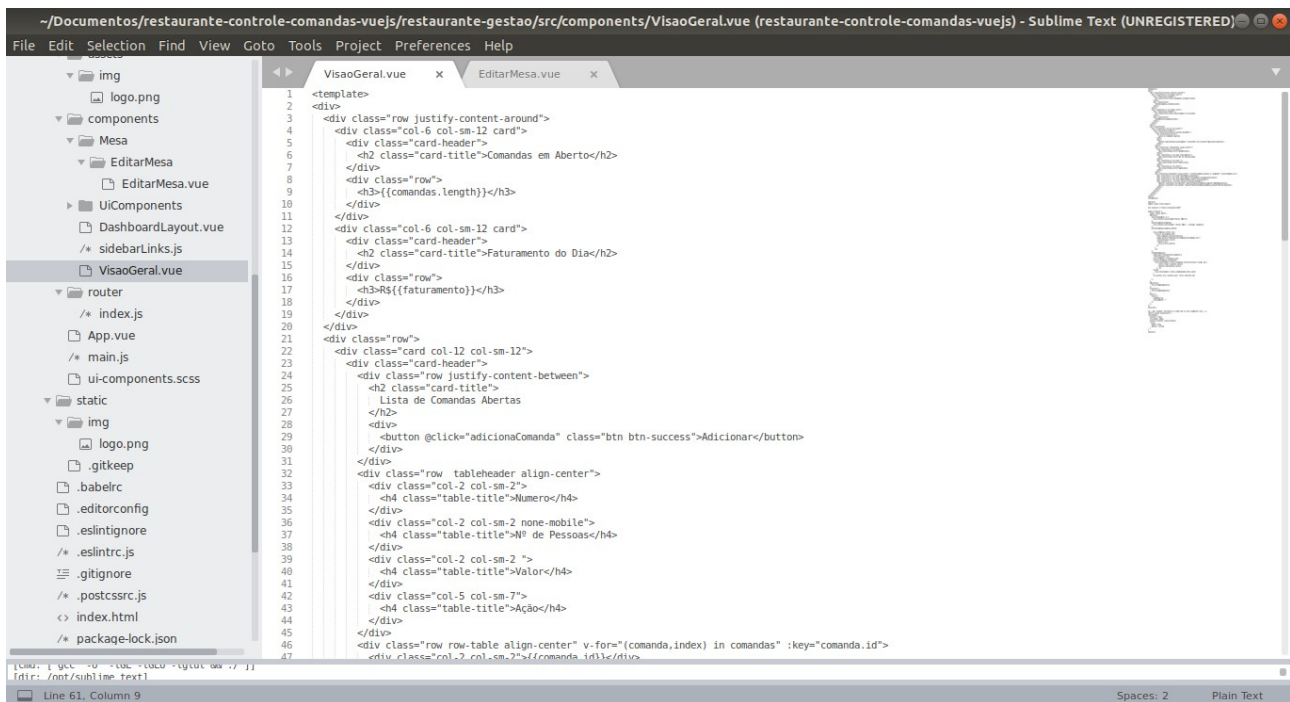
Vue.use(Router);

const router = new Router({
  mode: 'history',
  routes: [
    {
      path: '/',
      component: DashboardLayout,
      redirect: '/visao-geral',
      children: [
        {
          component: VisaoGeral,
          name: 'Visao Geral',
          path: '/visao-geral',
        },
        {
          component: EditarMesa,
          name: 'Editar Mesa',
          path: '/editar-mesa',
        }
      ]
    }
  ]
})

export default router
```

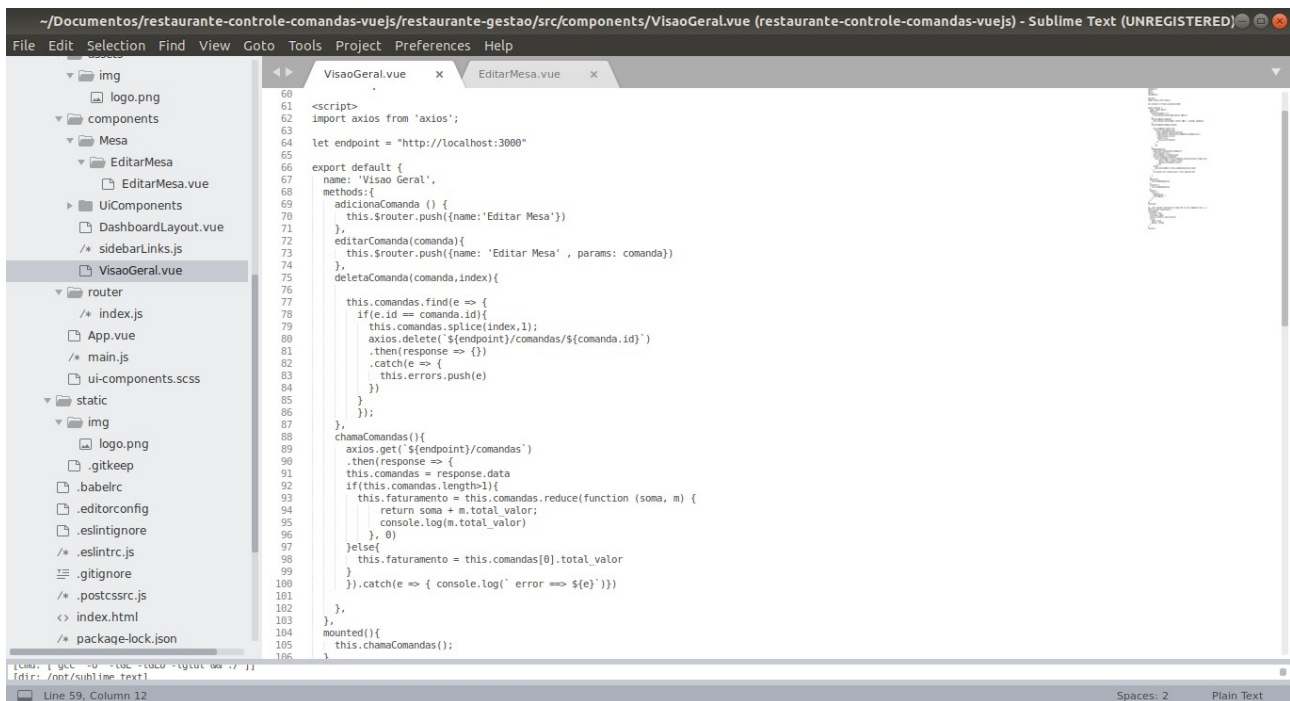
/src/components/VisaoGeral.vue

Da linha 1 à 59 é carregado o template html do projeto:



```
1 <template>
2 <div>
3   <div class="row justify-content-around">
4     <div class="col-6 col-sm-12 card">
5       <div class="card-header">
6         <h2 class="card-title">Comandas em Aberto</h2>
7       </div>
8       <div class="row">
9         <h3>{{comandas.length}}</h3>
10      </div>
11    </div>
12    <div class="col-6 col-sm-12 card">
13      <div class="card-header">
14        <h2 class="card-title">Faturamento do Dia</h2>
15      </div>
16      <div class="row">
17        <h3>R${faturamento}</h3>
18      </div>
19    </div>
20  </div>
21  <div class="row">
22    <div class="card col-12 col-sm-12">
23      <div class="card-header">
24        <div class="row justify-content-between">
25          <h2 class="card-title">
26            Lista de Comandas Abertas
27          </h2>
28          <div>
29            <button @click="adicionaComanda" class="btn btn-success">Adicionar</button>
30          </div>
31        </div>
32        <div class="row tableheader align-center">
33          <div class="col-2 col-sm-2">
34            <h4 class="table-title">Numero</h4>
35          </div>
36          <div class="col-2 col-sm-2 none-mobile">
37            <h4 class="table-title">Nº de Pessoas</h4>
38          </div>
39          <div class="col-2 col-sm-2">
40            <h4 class="table-title">Valor</h4>
41          </div>
42          <div class="col-5 col-sm-7">
43            <h4 class="table-title">Ação</h4>
44          </div>
45        </div>
46        <div class="row row-table align-center" v-for="(comanda,index) in comandas" :key="comanda.id">
47          <div class="col-2 col-sm-2">{{comanda.id}}</div>
```

A partir da linha 61 temos a implementação do javascript:



```
60
61 <script>
62 import axios from 'axios';
63
64 let endpoint = "http://localhost:3000"
65
66 export default {
67   name: 'Visao Geral',
68   methods: {
69     adicionaComanda () {
70       this.$router.push({name: 'Editar Mesa'})
71     },
72     editarComanda(comanda){
73       this.$router.push({name: 'Editar Mesa' , params: comanda})
74     },
75     deleteComanda(comanda, index){
76
77       this.comandas.find(e => {
78         if(e.id == comanda.id){
79           this.comandas.splice(index, 1);
80           axios.delete(`${endpoint}/comandas/${comanda.id}`)
81             .then(response => {})
82             .catch(e => {
83               this.errors.push(e)
84             })
85         }
86       });
87     },
88     chamaComandas(){
89       axios.get(`${endpoint}/comandas`)
90         .then(response => {
91           this.comandas = response.data
92           if(this.comandas.length>1){
93             this.faturamento = this.comandas.reduce(function (soma, m) {
94               return soma + m.total_valor;
95             }, 0)
96             console.log(m.total_valor)
97           }else{
98             this.faturamento = this.comandas[0].total_valor
99           }
100         })
101         .catch(e => { console.log(' error ==> ${e}')})
102     },
103   },
104   mounted(){
105     this.chamaComandas();
106   }
107 }
```

A biblioteca **axios**, é importada para fazer requisições para api

adicionaComanda : chama a rota editar comanda que é usada tambem para criar comandas

editarComanda: chama a rota editar comanda passando um parametro com objeto comanda que é a comanda que sera editada na rota

deleteComanda: passa o id da comanda para o backend a qual sera excluida no banco de dados.

chamaComandas: um get em todas as comandas do banco de dados e constroi a lista de comandas do visão geral

mounted: monta coisas antes do comonente ser renderizado, ai eu chamo as comandas antes para que não tenha delay

chamaComandas: PROGRAMAÇÃO CONCORRENTE

3. PROGRAMAÇÃO CONCORRENTE E FUNCIONAL:

```
axios.get(`${endpoint}/comandas`)
  .then(response => {
    this.comandas = response.data
    if(this.comandas.length>1){
      this.faturamento = this.comandas.reduce(function (soma, m) {
        return soma + m.total_valor;
        console.log(m.total_valor)
      }, 0)
```

Após ser chamada a função <axios.get> é executado o operador <.then> que executa uma logica dentro dele, o .then é o retorno da promise que é a parte paralela da função, o restante da função fora do .then continua sendo executado independente do que esta acontecendo dentro do .then

Logo todo bloco executado dentro de um operador .then é executado seja feito em paralelo com o que acontece no js, e o valor return deste bloco é repassado para a função que executou o .then.

Isso faz com que o nodejs não se preocupe com o que esta acontecendo na thread lançada e continue trabalhando em paralelo.

/restaurante-gestao/src/components/Mesa/EditarMesa/EditarMesa.vue

```
import axios from 'axios';
let endpoint = "http://localhost:3000"
export default {
  created(){
    this.chamaProdutos()
    if(this.$route.params.id){
      console.log(this.$route.params)
      this.mesa = this.$route.params
    }
  },
  methods: {
    chamaProdutos(){
      axios.get(`${endpoint}/produtos`)
        .then(response => {
          this.produtos = response.data
        }).catch(e => { console.log(` error => ${e}`)})
    },
    adicionaProduto() {
      if (this.produto.quantidade) {
        this.mesa.pedidos.push(JSON.parse(JSON.stringify(this.produto)))
        this.atualizaPedidos();
      }
    },
    atualizaPedidos(){
      this.mesa.total_valor = this.mesa.pedidos.reduce(function (soma, m) {
        return soma + (m.valor * m.quantidade);
      }, 0)
      this.mesa.total_valor_pessoa = (this.mesa.total_valor / this.mesa.num_pessoas).toFixed(2)
    },
    salvarLancamentos(){
      if(this.mesa.id){
        axios.put(`${endpoint}/comandas/${this.mesa.id}`, this.mesa).then(response =>{
          this.$router.push({name: 'Visao Geral'})
          console.log(response)
        }).catch( e => {console.log(e)})
      }else{
        axios.post(`${endpoint}/comandas`, this.mesa)
          .then(response => {})
          .catch(e => {
            this.errors.push(e)
          })
        this.$router.push({name: 'Visao Geral'})
      }
    },
    deletaComanda(comanda){
      axios.delete(`${endpoint}/comandas/${comanda.id}`)
        .then(response => {})
        .catch(e => {
          this.errors.push(e)
        })
      this.$router.push({name: 'Visao Geral'})
    },
    removePedido(index, pedido){
      this.mesa.pedidos.splice(index, 1)
      this.atualizaPedidos();
    },
    pagaConta() {
      if (this.pagamento.tipo == 1) {
        alert(`Pagamento total da Mesa R$ ${this.mesa.total_valor.toFixed(2)}`)
        this.deletaComanda(this.mesa)
      } else {
        let valorParcialTotal = this.mesa.total_valor_pessoa * this.pagamento.quantidade_pagantes
        alert(
          `Valor parcial referente a ${this.pagamento.quantidade_pessoas} pessoas R$ ${valorParcialTotal.toFixed(2)}`
        )
        this.mesa.num_pessoas = this.mesa.num_pessoas - this.pagamento.quantidade_pagantes
        this.mesa.total_valor = this.mesa.total_valor - valorParcialTotal
        this.mesa.total_valor_pessoa = (this.mesa.total_valor / this.mesa.num_pessoas).toFixed(2)
      }
      this.salvarLancamentos();
    }
  }
}
```

adicionaProduto: adiciona um produto a comanda

salvarLancamentos: A função `axios.put` atualiza um objeto no banco de dados, após sua execução o front irá redirecionar para a tela Visão geral, porem o js continua executando a linha de comando independente dessa ação da mesma forma que a outra função de calcular a soma do pedido.

4. ESTRUTURA DOS OBJETOS PEDIDOS, RETORNADOS PELO BACKEND:

Diretorio: `json-server/info.json`

```
28   "comandas": [  
29     {  
30       "num_pessoas": 2,  
31       "pedidos": [  
32         {  
33           "nome": "Mr. Burguer",  
34           "valor": 34,  
35           "quantidade": "5"  
36         },  
37         {  
38           "nome": "Mr. Burguer",  
39           "valor": 34,  
40           "quantidade": "5"  
41         },  
42         {  
43           "nome": "Mr. Bacon",  
44           "valor": 45,  
45           "quantidade": "5"  
46         },  
47         {  
48           "nome": "Mr. Bacon",  
49           "valor": 45,  
50           "quantidade": "5"  
51         },  
52       ]  
53     }  
54   ]  
55 }
```