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 uma página web, elaborado no framework nodejs com sockets io, com a habilidade de trabalhar com programação concorrente e assíncrona.

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

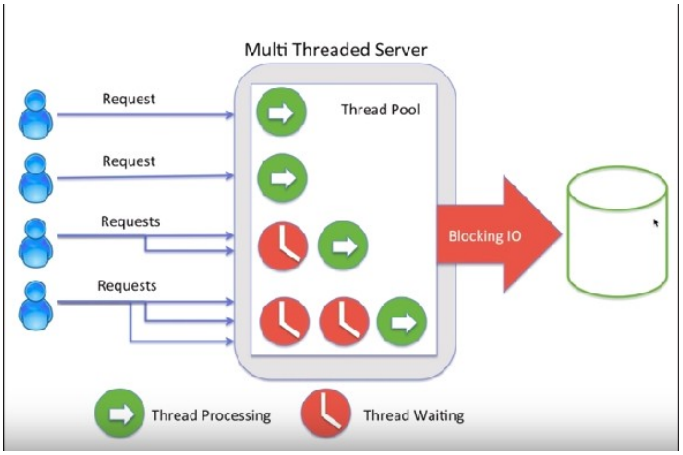


Figura1: Problema de espera de conclusão de threads de usuários em um servidor web.

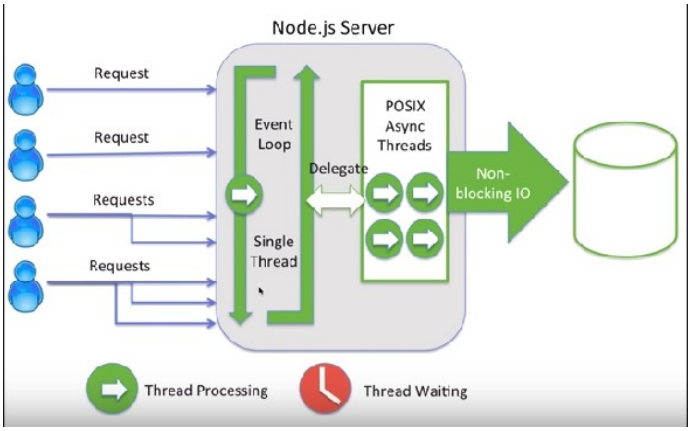


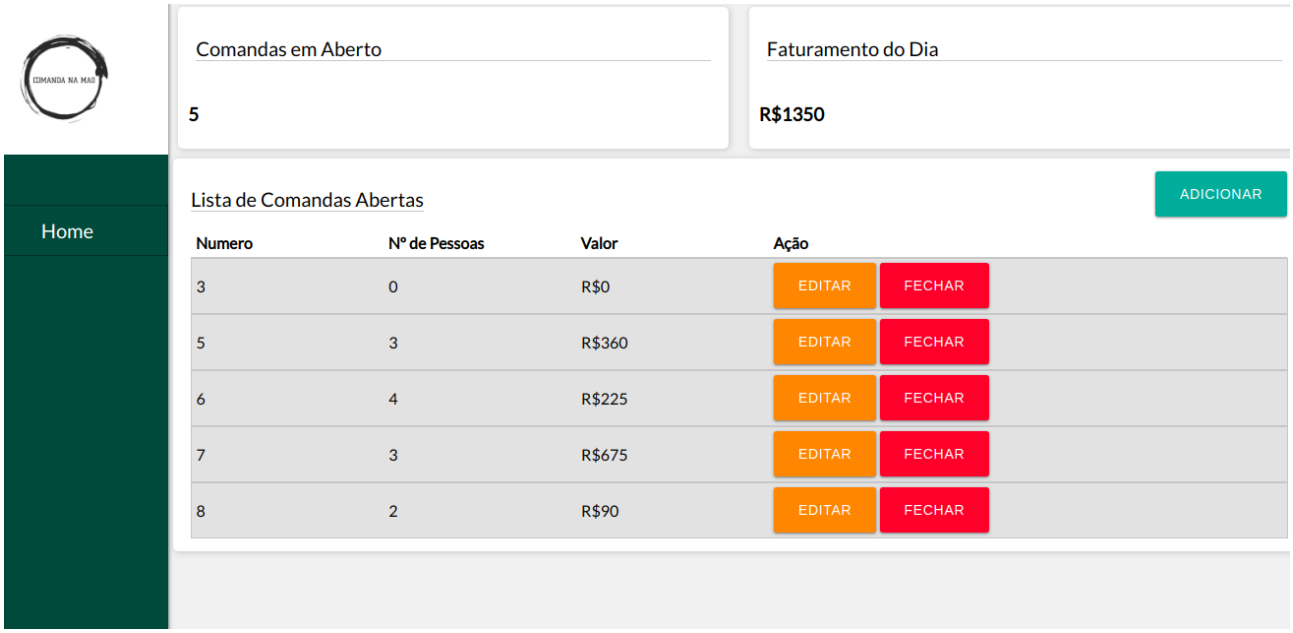
Figura2: Solução do nodejs com sockets io, para tratar Threads.

O Socket.IO é uma biblioteca JavaScript para aplicativos da Web em tempo real. Permite comunicação bidirecional em tempo real entre clientes da web e servidores. Ele tem duas

partes: uma biblioteca do lado do cliente que é executada no navegador e uma biblioteca do lado do servidor para o Node.js.

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

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



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.

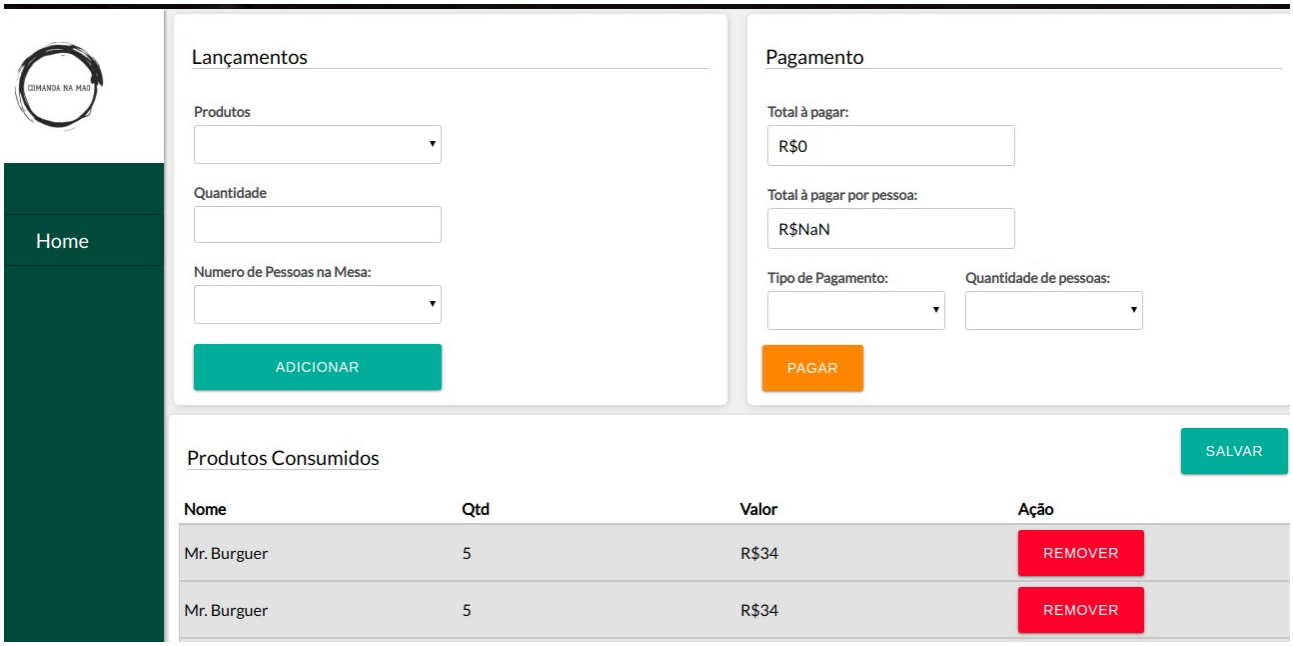
A lista no centro da tela, contêm um link para fechar uma mesa, e editá-la, indo para a página de front do garçom.

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:

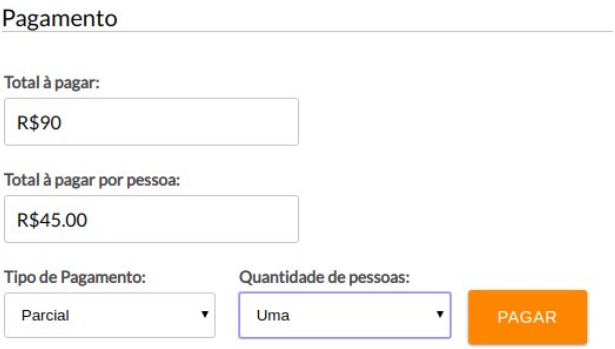


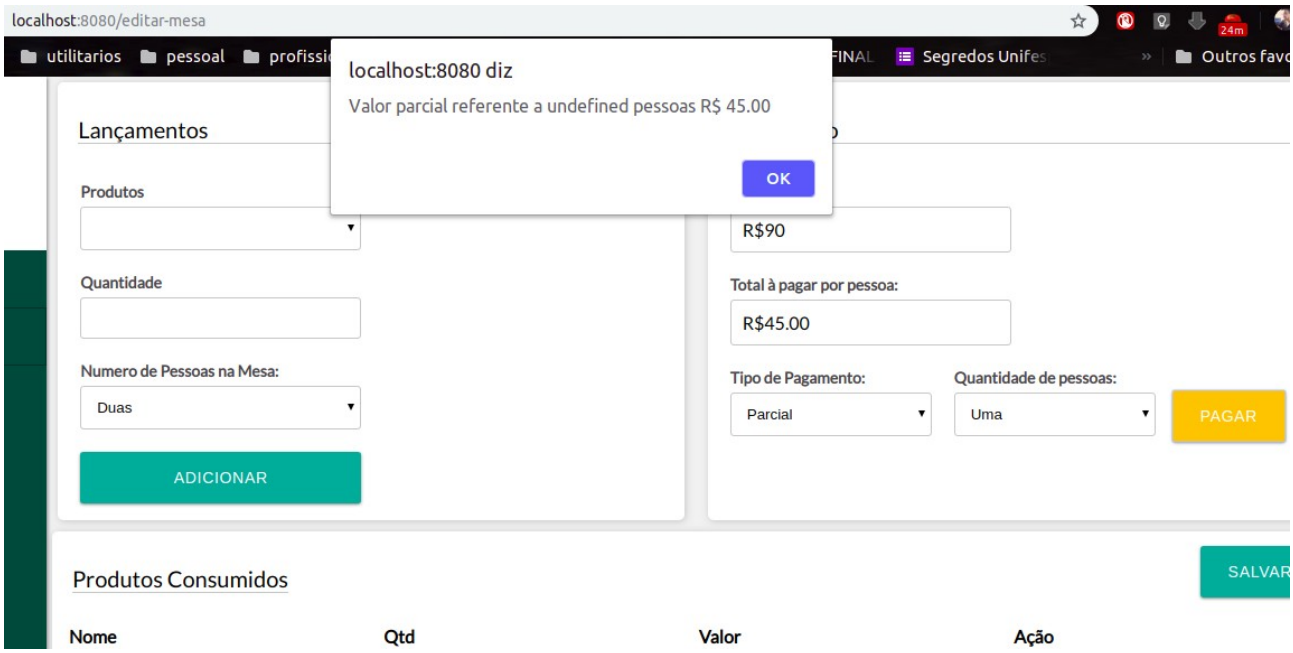
O bloco de lançamentos é manipulável.
Uma lista de produtos é obtida do banco de dados (json-server), com a lista de produtos e preços carregados por cada um.

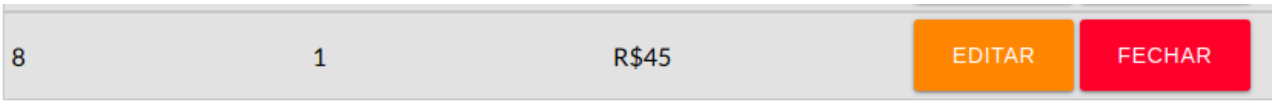
A informação de quantidade é apenas um multiplicador de produtos pedidos.

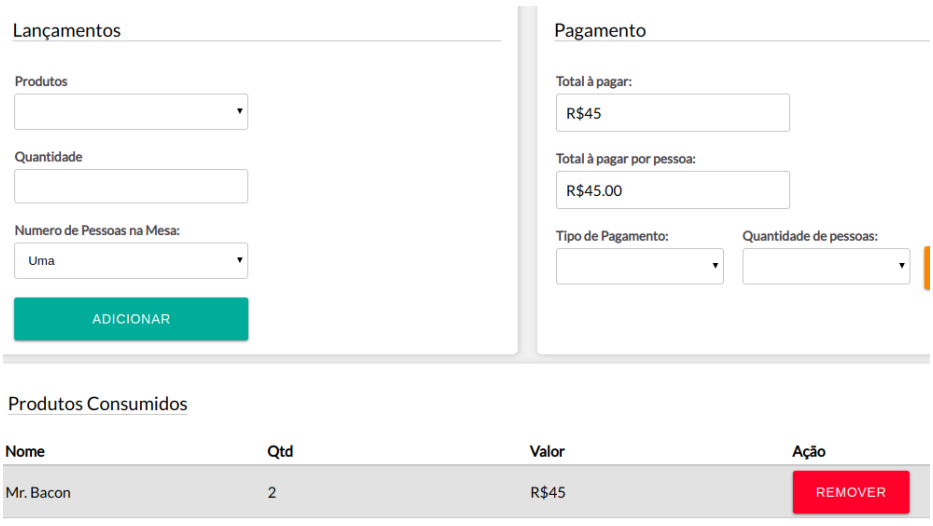
Número de pessoas na mesa é um valor inteiro, útil para o momento do pagamento.

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 quantia 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:
R$90
Total à pagar por pessoa:
R$45.00
Tipo de Pagamento:
Parcial
Quantidade de pessoas:
Uma
PAGAR


localhost:8080/editar-mesa
utilitarios
pessoal
Segredos Unifes
Outros favo
localhost:8080 diz
Valor parcial referente a undefined pessoas R$ 45.00
OK
Lançamentos
Produtos
Quantidade
Numero de Pessoas na Mesa:
Duas
ADICIONAR
R$90
Total à pagar por pessoa:
R$45.00
Tipo de Pagamento:
Parcial
Quantidade de pessoas:
Uma
PAGAR
Produtos Consumidos
SALVAR
Nome
Qtd
Valor
Ação


8
1
R$45
EDITAR
FECHAR


Lançamentos
Produtos
Quantidade
Numero de Pessoas na Mesa:
Uma
ADICIONAR
Pagamento
Total à pagar:
R$45
Total à pagar por pessoa:
R$45.00
Tipo de Pagamento:
Quantidade de pessoas:
Produtos Consumidos
Nome
Qtd
Valor
Ação
Mr. Bacon
2
R$45
REMOVER

2. PÁGINA VISÃO GERAL / CÓDIGO FONTE:

```
import axios from 'axios';
let endpoint = "http://localhost:3000"
export default {
  name: 'Visao Geral',
  methods:{
    adicionaComanda () {
      this.$router.push({name:'Editar Mesa'})
    },
    editarComanda(comanda){
      this.$router.push({name: 'Editar Mesa' , params: comanda})
    },
    deletaComanda(comanda,index){
      this.comandas.find(e => {
        if(e.id == comanda.id){
          this.comandas.splice(index,1);
          axios.delete(`${endpoint}/comandas/${comanda.id}`)
          .then(response => {})
          .catch(e => {
            this.errors.push(e)
          })
        }
        });
    },
    chamaComandas(){
      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)
      }else{
        this.faturamento = this.comandas[0].total_valor
      }
      }).catch(e => { console.log(` error ==> ${e}`)})

    },
  },
  mounted(){
    this.chamaComandas();
  },
  created() {
    this.chamaComandas();
  },
  data() {
    return {
      comandas:[],
      faturamento: ''
    };
  },
};
```

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

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

chamaComandas: um get em todas as comandas do bd, 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 funçao <axios.get> é executado o operador <.then> que executa uma logica dentro dele, o .then é o retorno da promisse 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.

```
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)
          }).cath( 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 ira 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

```
"comandas": [
  {
    "num_pessoas": 2,
    "pedidos": [
      {
        "nome": "Mr. Burguer",
        "valor": 34,
        "quantidade": "5"
      },
      {
        "nome": "Mr. Burguer",
        "valor": 34,
        "quantidade": "5"
      },
      {
        "nome": "Mr. Bacon",
        "valor": 45,
        "quantidade": "5"
      },
      {
        "nome": "Mr. Bacon",
        "valor": 45,
        "quantidade": "5"
      },
```