



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 javascript que usa funções nativas de programação concorrente com promises.

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

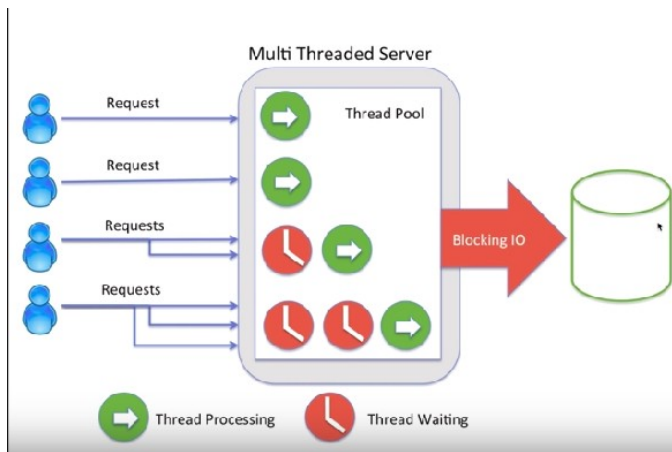


Figura1: Problema de espera de conclusão de threads de usuários em um servidor web.

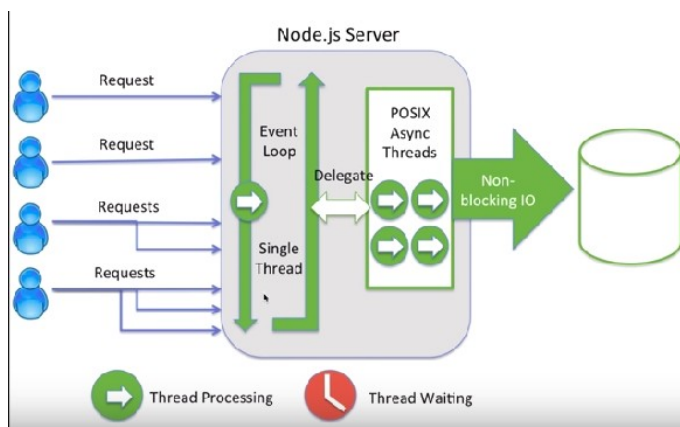


Figura2: Solução do nodejs com sockets io, para tratar Threads.

Uma promise é o resultado final de uma operação assíncrona. É um marcador de posição no qual o valor de resultado bem-sucedido ou motivo da falha se materializará.

As promissas fornecem uma alternativa mais simples para executar, compor e gerenciar operações assíncronas quando comparadas a abordagens tradicionais baseadas em retorno de chamada.

Eles também permitem que você lide com erros assíncronos usando abordagens semelhantes a try / catch síncrono.

A API primária de uma promessa 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

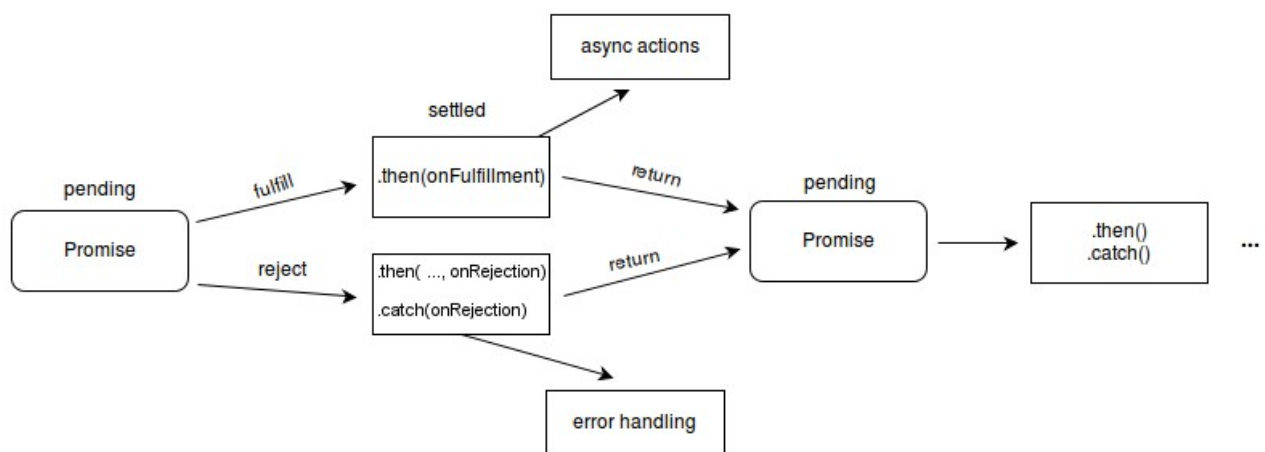


imagem:

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

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	EDITAR FECHAR
5	3	R\$360	EDITAR FECHAR
6	4	R\$225	EDITAR FECHAR
7	3	R\$675	EDITAR FECHAR
8	2	R\$90	EDITAR FECHAR

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. Burger	5	R\$34	REMOVER
Mr. Burger	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:

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 profissio

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:

/restaurante-gestao/src/components/VisaoGeral.vue

```
1  import axios from 'axios';
2  let endpoint = "http://localhost:3000"
3  export default {
4    name: 'Visao Geral',
5    methods:{
6      adicionaComanda () {
7        this.$router.push({name:'Editar Mesa'})
8      },
9      editarComanda(comanda){
10       this.$router.push({name: 'Editar Mesa' , params: comanda})
11     },
12     deletaComanda(comanda,index){
13       this.comandas.find(e => {
14         if(e.id == comanda.id){
15           this.comandas.splice(index,1);
16           axios.delete(`${endpoint}/comandas/${comanda.id}`)
17             .then(response => {})
18             .catch(e => {
19               this.errors.push(e)
20             })
21         }
22       });
23     },
24     chamaComandas(){
25       axios.get(`${endpoint}/comandas`)
26         .then(response => {
27           this.comandas = response.data
28           if(this.comandas.length>1){
29             this.faturamento = this.comandas.reduce(function (soma, m) {
30               return soma + m.total_valor;
31               console.log(m.total_valor)
32             }, 0)
33           }else{
34             this.faturamento = this.comandas[0].total_valor
35           }
36         }).catch(e => { console.log(` error ==> ${e}`)})
37
38     },
39   },
40   mounted(){
41     this.chamaComandas();
42   },
43   created() {
44     this.chamaComandas();
45   },
46   data() {
47     return {
48       comandas:[],
49       faturamento: ''
50     };
51   },
52 };
53
```

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

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

deletaComanda: passa o id da comanda para o backend a qual será excluída 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 componente ser renderizado, aí 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 lógica 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 está 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

```
1 import axios from 'axios';
2 let endpoint = "http://localhost:3000"
3 export default {
4   created(){
5     this.chamaProdutos()
6     if(this.$route.params.id){
7       console.log(this.$route.params)
8       this.mesa = this.$route.params
9     }
10  },
11  methods: {
12    chamaProdutos(){
13      axios.get(`${endpoint}/produtos`)
14        .then(response => {
15          this.produtos = response.data
16        }).catch(e => { console.log(` error ==> ${e}`)})
17    },
18    adicionaProduto() {
19      if (this.produto.quantidade) {
20        this.mesa.pedidos.push(JSON.parse(JSON.stringify(this.produto)))
21        this.atualizaPedidos();
22      }
23    },
24    atualizaPedidos(){
25      this.mesa.total_valor = this.mesa.pedidos.reduce(function (soma, m) {
26        return soma + (m.valor * m.quantidade);
27      }, 0)
28      this.mesa.total_valor_pessoa = (this.mesa.total_valor / this.mesa.num_pessoas).toFixed(2)
29    },
30    salvarLancamentos(){
31      if(this.mesa.id){
32        axios.put(`${endpoint}/comandas/${this.mesa.id}`, this.mesa).then(response =>{
33          this.$router.push({name: 'Visao Geral'})
34          console.log(response)
35        }).catch( e => {console.log(e)})
36      }else{
37        axios.post(`${endpoint}/comandas`, this.mesa)
38          .then(response => {})
39          .catch(e => {
40            this.errors.push(e)
41          })
42        this.$router.push({name: 'Visao Geral'})
43      }
44    },
45  },
46 }
```

```

45     deleteComanda(comanda){
46         axios.delete(`${endpoint}/comandas/${comanda.id}`)
47         .then(response => {})
48         .catch(e => {
49             this.errors.push(e)
50         })
51         this.$router.push({name: 'Visao Geral'})
52     },
53     removePedido(index, pedido){
54         this.mesa.pedidos.splice(index, 1)
55         this.atualizaPedidos();
56     },
57     pagaConta() {
58         if (this.pagamento.tipo == 1) {
59             alert(`Pagamento total da Mesa R$ ${this.mesa.total_valor.toFixed(2)}`)
60             this.deleteComanda(this.mesa)
61         } else {
62             let valorParcialTotal = this.mesa.total_valor_pessoa * this.pagamento.quantidade_pagantes
63             alert(
64                 `Valor parcial referente a ${this.pagamento.quantidade_pessoas} pessoas R$ ${valorParcialTotal.toFixed(2)}`
65             )
66             this.mesa.num_pessoas = this.mesa.num_pessoas - this.pagamento.quantidade_pagantes
67             this.mesa.total_valor = this.mesa.total_valor - valorParcialTotal
68             this.mesa.total_valor_pessoa = (this.mesa.total_valor / this.mesa.num_pessoas).toFixed(2)
69         }
70         this.salvarLancamentos();
71     }
72 }

```

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`

```

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                 },

```