**Semana OmniStack – 10° Edição**

*Funcionalidades Avançadas*

# Real-Time

Quando um dev se cadastrar na web, atualiza automaticamente no mobile sem a necessidade do recarregamento.

## Protocolo WebSocket

Protocolo que permite o backend fornecer informações ao frontend sem ser a partir de uma requisição HTTP.

Comando: yarn add socket.io

Index.js

const http = require('http')

const { setupWebsocket } = require('./websocket')

//servidor http fora do express para trabalhar com ele diretamente

//extrai o servidor http do express

const server = http.Server(app)

//chama a função criada enviando o servidor

setupWebsocket(server)

//definição da porta da aplicação

server.listen(3333)

websocket.js

//faz com que o servidor fique escutando também o protocolo websocket

const socketio = require('socket.io')

const parseStringAsArray = require('./utils/parseStringAsArray')

const calculateDistance = require('./utils/calculateDistance')

//salvamento das conexões

//idel que seja em um BD

//realizado na memória do node

const connections = []

let io

//exporta o primeiro método

//faz as primeiras configurações para o servidor aceitar as requisições no formato websocket

*exports*.setupWebsocket = (*server*) => {

    io = socketio(server)

    //toda vez que receber uma conexão

    //listener .on

    io.on('connection', *socket* => {

        //pega o parâmetros recebidos

        const { latitude, longitude, techs } = socket.handshake.query

        connections.push({

            id: socket.id,

            coordinates: {

                latitude: *Number*(latitude),

                longitude: *Number*(longitude),

            },

            techs: parseStringAsArray(techs),

        })

    })

}

//faz o filtro das conexões

*exports*.findConnections = (*coordinates*, *techs*) => {

    return connections.filter(*connection* => {

        //calcula a distancia. < 10km

        //some: retorna se pelo menos 1 é true

        return calculateDistance(coordinates, connection.coordinates) < 10

            && connection.techs.some(*item* => techs.includes(item))

    })

}

//pra quem quer mandar a msg

//o tipo da msg

//e o valor

*exports*.sendMessage = (*to*, *message*, *data*) => {

    //percorre cada um dos destinatários

    to.forEach(*connection* => {

        //envia a msg ao socketid

        //qual o tipo da mensagem e o valor

        io.to(connection.id).emit(message, data)

    })

}

devController.js

const { findConnections, sendMessage } = require('../websocket')

//filtrar as conexões procurando por aquelas que satisfaçam aos filtros (coordenadas e tecnologias)

            const sendSocketMessageTo = findConnections(

                { latitude, longitude },

                techsArray

            )

            sendMessage(sendSocketMessageTo, 'new-dev', dev)

### Parte Mobile

Comando: yarn add socket.io-client

Services/socket.js

//configuração do websocket no mobile

import socketio from 'socket.io-client'

//passa qual é o endereço do backend

const socket = socketio('http://192.168.56.1:3333', {

    //não deixa fazer a conexão automaticamente

    autoConnect: false,

})

function subscribeToNewDevs(*subscribeFunction*) {

    //escuta o evento

    socket.on('new-dev', subscribeFunction)

}

//faz a conexão

function connect(*latitude*, *longitude*, *techs*) {

    //manda como opções os parâmetros para o backend

    socket.io.opts.query = {

        latitude,

        longitude,

        techs,

    }

    socket.connect()

}

function disconnect() {

    if(socket.connected) {

        socket.disconnect()

    }

}

export { connect, disconnect, subscribeToNewDevs }

main

//importa o socket pois e a pagina que precisa se comunicar com o backend qnd uma nova inserção for feita

import { connect, disconnect, subscribeToNewDevs } from '../services/socket'

//sempre que a variável devs mudar executa a função

    useEffect(() => {

        subscribeToNewDevs(*dev* => setDevs([...devs, dev]))

    }, [devs])

function setupWebsocket() {

        //desconecta a conexão anterior para não ficar com conexões sobrando

        disconnect()

        const { latitude, longitude } = currentRegion

        //envia para o servidor os parâmetros

        connect(

            latitude,

            longitude,

            techs,

        )

    }

//carregar usuários: carrega toda vez que o usuário clica no botão

    async function loadDevs() {

        const { latitude, longitude } = currentRegion

        try {

            const response = await api.get('/search', {

                params: {

                    latitude,

                    longitude,

                    techs

                }

            })

            setDevs(response.data.devs)

        } catch(err) {

*console*.error(err);

        }

        //chama o real time apenas qnd ocorre a pesquisa/passa a escutar apenas qnd a pesquisa é feita

        setupWebsocket()

    }