UNIVERSIDADE FEDERAL DO CEARÁ

REDES DE COMPUTADORES

SISTEMAS DISTIBUÍDOS

RELATÓRIO DO PROJETO

Título do Projeto: Calculadora do Ciclista

Aluno : Mateus Allen, Alberto Luiz, Rafael Lima

Matrícula : 476735, 428423, 428453

Professor : Marcos Dantas

Quixadá - CE, 7 de abril de 2021

Introdução e serviços remotos

O projeto da *Calculadora do ciclista* tem como principal objetivo a identificação da sua saúde, se você esta de acordo com o seu peso, qual o quadro da bicicleta da mais para sua altura e por último a identificação dos seus batimentos cardíacos se está normal ou precisa ir ao médico. Diante disso, o nosso projeto foi desenvovido na linguagem de programção em *Java* e *JavaScripts - Nodes.js*, o cliente é em Java (o cliente é que vai inserir seus dados), já o servido é em Javascripts, pois já trata de forma mais adequada os dados e a entrega de pacotes na rede e não precisamos nos preocupar com threads, pois o o servidor criado em JavaScript ja trata isso mantendo o desempenho na execução dos processos, que no caso, utilizamos o protocolo UDP.

1º Serviço remoto: Peso Nesse serviço remoto o usuário deve informar a sua altura e peso, pois estamos realizando o calcúlo do IMC (Índice de Massa Corporal) para identificarmos se você esta no peso adequado ou melhor ainda normal. A *figura 1* mostra como ficou o código para o usuário inserir os dados e serem processados para serem calculados e depois apresnetados para o cliente.

```
{
    double dados[] = new double[2];
    System.out.println("------------------CALCULO DE IMC------------------");
    System.out.print("informe sua altura: ");
    dados[0] = Double.parseDouble(teclado.nextLine());
    System.out.print("informe seu peso: ");
    dados[1] = Double.parseDouble(teclado.nextLine());
    System.out.println(calculadoraCiclista.IMC(dados[0], dados[1]));
    break;
}
```

Figura 1 – Código para informa os dados

2º Serviço remoto: Altura da bike Nesse serviço remoto temos só uma informação na qual o usuário deverá inserir, que é a sua altura , para pedermo identic-lá e apresentar para os clientes o quadro da bicicleta perfeito de acordo com a sua altura. O quadro da bicicleta significa o tamanho a bike certa para você sem a necessidades de ficar na dúvida qual bicicleta perfeita, apartir da sua altura pode identicar o quador da mesma, e o importante é que o cliente não precisa ta direto procurando o bike idela, simplesmente inserindo a sua altura o nosso serviço identica para você.

```
{
    double dados[] = new double[1];
    System.out.println("-----------------QUADRO IDEAL DA BIKE-----------------");
    System.out.print("informe sua altura: ");
    dados[0] = Double.parseDouble(teclado.nextLine());
    System.out.println(calculadoraCiclista.alturaBike(dados[0]));
    break;
}
```

Figura 2 – Código para informa os dados

3º Serviço remoto: Batimentos cardíacos

Esse nosso último serviço é de grande importância para todas as classes de pessoas, identificando se você esta com os batimentos adequados ou se precisa de acompanhamento médico, o usuário apenas irá inserir o quanto de pulsação ocorreu em 10 segundos, apenas irá informar quantas pulsações aconteceu nos segundos estabelecidos pelo nossso serviço. O usuário pode identificar as suas pulsações no pulso precionando com o dedo indicador e maior de todos, ou no pescoço, que pra alguns é mais preciso e fácil de identificar.

```
{
    int dados[] = new int[1];
    System.out.println("-----------------BATIMENTOS CARDIACOS-----------------");
    System.out.println("DICA: CONTE O NUMERO DE PULSAÇÕES POR 10 SEGUNDOS");
    System.out.print("Informe o numero de pulsações: ");
    dados[0] = Integer.parseInt(teclado.nextLine());
    System.out.println(calculadoraCiclista.batimentosCardiacos(dados[0]));
    break;
}
```

Figura 3 – Código para informa os dados

Métodos remotos

Podemos identificar na *figura 4* como os dados serão recebidos de acordo com os métods da *linha 2* até a *linha 7*, e depois passa para o construtor a serem contruidos e apresentados.

```
1   class Message {
2       messageType; // int
3       requestId; // int
4       objectReference; // String
5       methodId; // String
6       methodId; // Srting
7       arguments; // Array ou Object
8
9       constructor(msg) {
10          const json = JSON.parse(msg);
11          this.messageType = json.messageType;
12          this.requestId = json.requestId;
13          this.objectReference = json.objectReference;
14          this.methodId = json.methodId;
15          this.methodId = json.methodId;
16          this.arguments = json.arguments;
17      }
18  }
19
20  module.exports = Message;
```

Figura 4 – Construção dos dados ao ser repassados pro lado servidor

Classes que o dados são enviados

* Temos a classe *IMC* que vai realizar o cálculo da dos dados inseridos pelo usuário, que será a altura e o peso, assim para identifcar a sua massa muscular.

```
class IMC {
    altura; // double
    peso; // double

    constructor(altura, peso) {
        this.altura = parseFloat(altura);
        this.peso = parseFloat(peso);
    }

    calcIMC() {
        const imc = this.peso / (this.altura * this.altura);
        let resposta = null;

        if (imc <= 18.5) {
            resposta = "VOCE ESTÁ ABAIXO DO PESO IDEAL";
        } else if (imc >= 18.6 && imc <= 24.9) {
            resposta = "PESO IDEAL PARABÉNS!!!";
        } else if (imc >= 25 && imc <= 29.9) {
            resposta = "SOBREPESO, COMECE A PEDALAR DE FORMA LEVE";
        } else if (imc >= 30 && imc <= 34.9) {
            resposta = "OBESIDADE GRAU 1, PROCURE PEDALAR COM FREQUÊNCIA";
        } else if (imc >= 35 && imc <= 39.9) {
            resposta = "OBESIDADE GRAU 2, COMECE A PEDALAR CON FREQUENCIA SEGUIDA DA AJUDA DE UM ESPECIALISTA";
        } else if (imc >= 40) {
            resposta = "OBESIDADE MORBIDA, COMECE A PEDALAR CON FREQUENCIA SEGUIDA DA AJUDA DE UM ESPECIALISTA, VOCE CORRE RISCO DE VIDA";
        }
        return [ resposta ];
    }
}
```

Figura 5 – IMC

* Temos a classe *Bike* que vai identificar para o usuário a bicicleta ideal para sua compra, apenas inserindo a sua altura o nosso serviço já identica o quadro perfeito para o cliente.

```
30
31  class Bike {
32      altura; // double
33
34      constructor(altura) {
35          this.altura = parseFloat(altura);
36      }
37
38      alturaBike() {
39          let resposta;
40
41          if (this.altura >= 1.65 && this.altura <= 1.71) {
42              resposta = "Quadro ideal para sua altura é a S -15 ou 16-";
43          } else if (this.altura >= 1.72 && this.altura <= 1.76) {
44              resposta = "Quadro ideal para sua altura é a M -17 ou 18-";
45          } else if (this.altura >= 1.77 && this.altura <= 1.82) {
46              resposta = "Quadro ideal para sua altura é a L -19-";
47          } else if (this.altura >= 1.83 && this.altura <= 1.90) {
48              resposta = "Quadro ideal para sua altura é a XL -21-";
49          } else {
50              resposta = "Altura invalida";
51          }
52          return [ resposta ];
53      }
54  }
```

Figura 6 – BIke

* Temos a classe *Health* que calcula para o usuário o se os seus batimentos cardíacos então normais ou se precisa ir ao médico se consultar, apenas irá inserir o total de batimentos em 10 segundos estabelcidos pelo nosso serviço.

```
55
56  class Health {
57      batimentosTotal; // int
58
59      constructor(batimentosTotal) {
60          this.batimentosTotal = parseInt(batimentosTotal);
61      }
62
63      batimentosCardiacos() {
64          const resultado = (this.batimentosTotal * 6);
65          let resposta;
66
67          if (resultado < 60) {
68              resposta = "Você está com bradicardia procure um médico";
69          } else if (resultado > 100) {
70              resposta = "Você está com taquicardia procure um médico";
71          } else {
72              resposta = "Batimentos normal";
73          }
74          return [ resposta ];
75      }
76  }
77
78  module.exports = {
79      IMC,
80      Bike,
81      Health
82  };
```

Figura 7 – Health

Dados transmitidos

Aqui é onde ocorre todo o tratamento dos dados inseridos nas classes abaixo. Iremos explicar como cada classe irá tratar de tal informação até ser apresentado ao cliente e ao servidor só quando a mensagem for enviada ela será tratada com ID, arguments e outros (onde nós chamamos de *sendRequest e sendResponse* que acontece entre o cliente e servidor.

```java
package cliente;

import org.json.JSONArray;
import org.json.JSONObject;

public class RequestMsg {
    private int messageType = 0;
    private int requestId;
    private String methodId;
    private String objectReference;
    private String[] arguments;

    public RequestMsg(String objectReference, String methodId, String[] arguments, int requestId) {
        this.methodId = methodId;
        this.objectReference = objectReference;
        this.arguments = arguments;
        this.requestId = requestId;
    }

    public int getRequestId() {
        return requestId;
    }

    public void setRequestId(int requestId) {
        this.requestId = requestId;
    }

    public String toString() {
        JSONArray array = new JSONArray();
        JSONObject json = new JSONObject();
        json.put("messageType", this.messageType);
        json.put("requestId", this.requestId);
        json.put("methodId", this.methodId);
        json.put("objectReference", this.objectReference);
        for (String args: this.arguments) {
            array.put(args);
        }
        json.put("arguments", array);
        return json.toString();
    }
}
```

Figura 8 – Request - dados transmitidos

Cleinte UDP Na classe do cliente passamos os parâmetros necessários para a conexão como o servidor, a porta e o ip e depois a mensagem a ser enviada, o cliente empacota mensagem onde inserimos um tamanho real da mesagem, pois sem um valor de tamanho exato não conecta e da erro entre as duas classes cliente e servidor. Resolvemos fazer em o cliente em Java pois só vai fazer a requisição, já no servidor não é isso, ire exeplicar mais adiante. Portanto, tivemos todos a questão de tratamento de erro de conectivdade, no caso da mensagem não ser enviada no lado cliente e podemos ver esse exemplo muito importante que pode ser visto na *figura 8*.

```java
package cliente;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPCliente {

    private DatagramSocket socket;

    public void sendRequest(String request, int porta, String ip) throws IOException {
        socket = new DatagramSocket();
        byte[] dados = request.getBytes();
        InetAddress host = InetAddress.getByName(ip);
        int servePorta = porta;
        DatagramPacket mensagem = new DatagramPacket(dados, dados.length, host, servePorta);
        socket.send(mensagem);
        System.out.println("mensagem enviada");
    }
    public String sendResponse() throws IOException {
        byte[] buffer = new byte[1000];
        DatagramPacket resposta = new DatagramPacket(buffer, buffer.length);
        socket.setSoTimeout(5000);
        socket.receive(resposta);
        System.out.println("mensagem recebida");
        String result = new String(resposta.getData());
        return result;
    }
    public void close() {
        socket.close();
    }
}
```

Figura 9 – Cliente UDP

Servidor Java Script Já no servidor temos não temos que inserir threads, pois a própria linguagem já tratat do desempenho das mensagens quando são entregues desempacotadas e entregues ao cliente, onde teremos todo o sistema de como os dados serão tratados passando pelas outras classes que fazem parte de um requesição positiva e de grande segurança em relação aos dados. E essa troca de mensagens acontece de forma pergunta e reposta tanto para o lado do cliente como do lado servidor, só que no lado do cliente é uma forma mais diferente, pois mostra o que o usuário realmente quer, lá no servidor temos o tratamento das mesmas, com o ID da mensagem o argumento e entre outros.

```javascript
const Despachante = require('./Despachante');
const Message = require('./Message');

class UDPServe {
    despachante = new Despachante();

    constructor(socket, porta) {
        this.socket = socket;
        this.porta = porta;
    }

    getRequest(msg, rinfo) {
        const message = new Message(msg.toString());
        message.arguments = this.despachante.invoke(message);
        message.messageType = 1;
        console.log(JSON.stringify(message, null, 4));
        this.sendReply(JSON.stringify(message), rinfo.address, rinfo.port);
    }

    sendReply(reply, clientHost, clientPort) {
        this.socket.send(reply, clientPort, clientHost, (error) => error && console.error(error));
    }
}

module.exports = UDPServe;
```

Figura 10 – Servidor Java Script

Comunicação e transparência

Comunicação Nessa classe onde ocorre a conexão é estabelecida e pode ser utilizadas para envia e receber as mesagens que foram enviada. Empacotadas no lado cliente e desempacotadas no ladeo serviço.

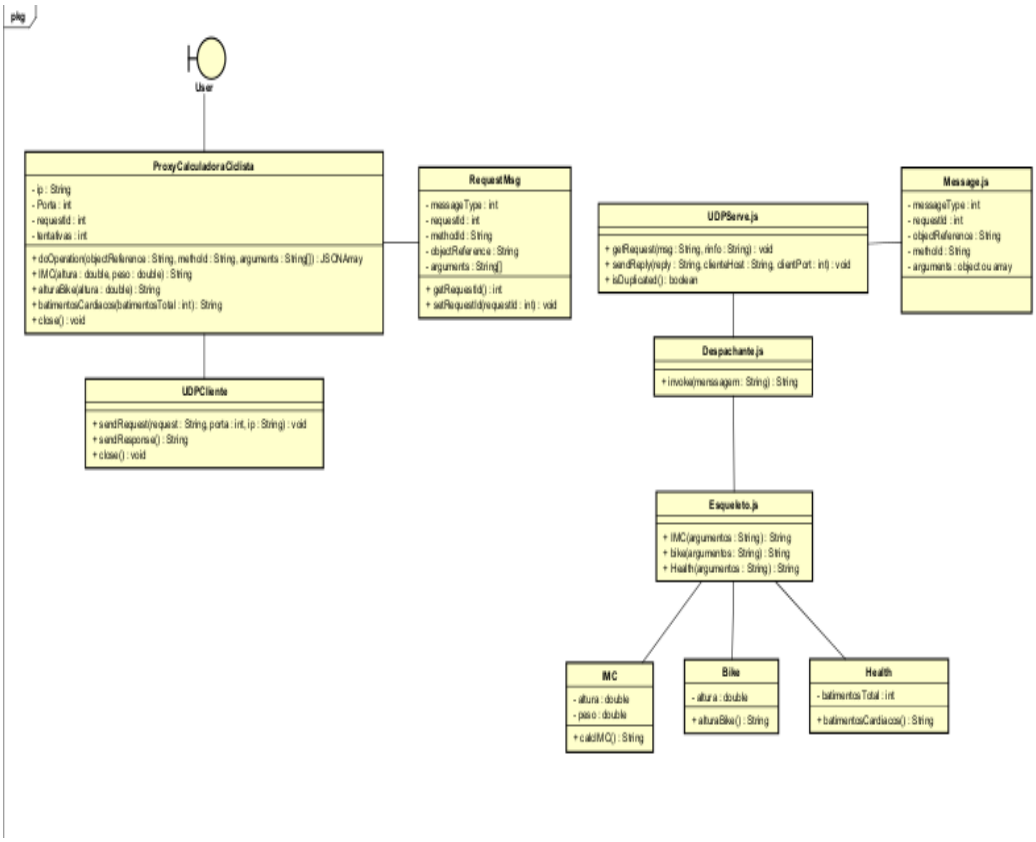

Figura 11 – Modelo UML para representar a classe Comunicação

Serviço Para completar sobre esse assunto, os tipos de serviço do nosso sistema ja foi descrito. Mas, para fixar mais o entdimento, é sobre o que o nosso serviço ter a oferecer ao cliente que são três serviços, o IMC, a Bike ideal e os seus batimentos cardíacos.

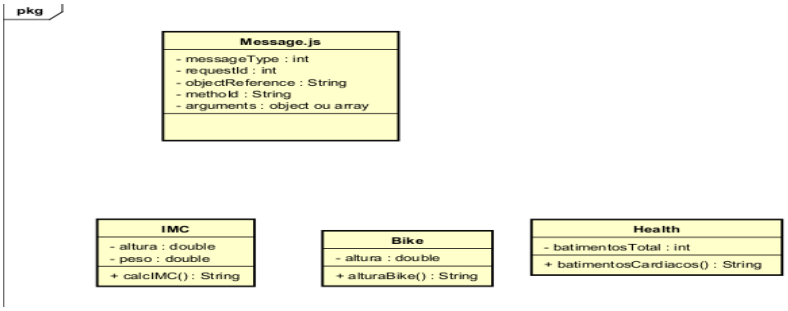

Figura 12 – Modelo UML para representar as classe Serviço

Transparência Nas classe abaixo podemos perceber que elas não são visiveis para o usuário, ficam ocultas, poisnão tem necessidade de o usuário quere saber isso, mas sim o que o serviço oferece e de ótima qualidade, essas classes é como se fosse um único sistema para o usuário.

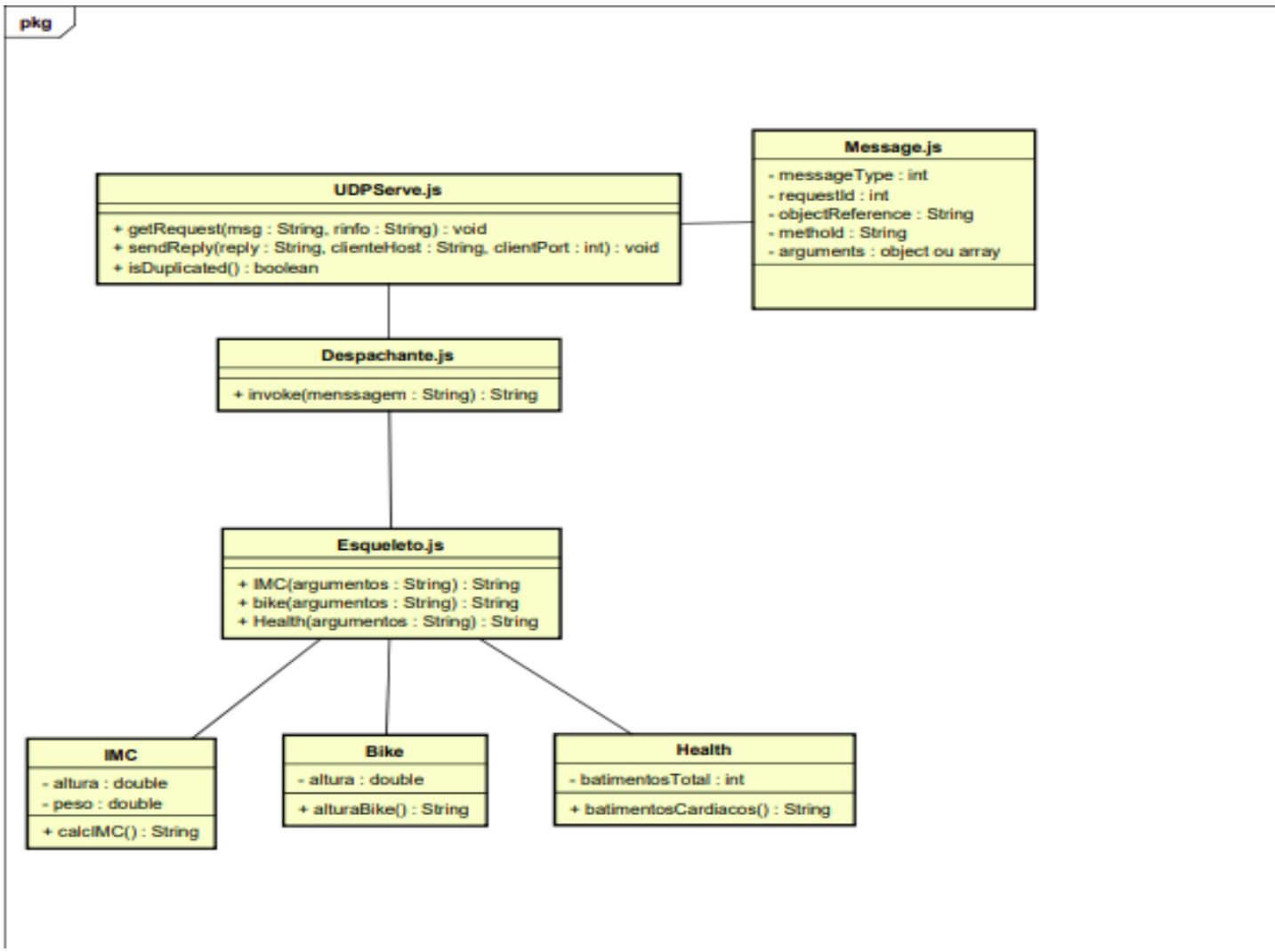

Figura 13 – Modelo UML para representar as classe de Transparência

Modelos de falhas

No nosso serviço podemos identificar as falhas de mensagens duplicadas e do timeout. No entanto, foi solucionado a o timeout, quando o cliente exceder mais de que 5 requesições o serviço da erro de timeout e apresenta no console para o cliente e demora 5 segundos. A questão das mensagens duplicadas foi resolvida da seguinte maneira: criou o objeto e o nome desse objeto é o IP do cliente que esta conctado e cria uma variável que guarda o último ID que foi enviado e tem um vetor que armazena a mensagem por completo onde guarda o histórico de mensagens, quando chega uma mensagem nova, verifico se o ID que chegou novo é igual ao ID que esta no histórico, se for igual mensagem é duplicada, senão adiciono essa mensagem ao vetor de mensagens e atualizo o ID parra o ID novo. E para o tratamento de exceções utilizamos o *try e catch*, é de grande importência para quando ocorrer algum problema com o comando no bloco a execução desviará para o catch, pois eles tratam possiveís erros de conversão.