

**FACULDADE CATÓLICA SALESIANA CURSO DE ENGENHARIA DA
COMPUTAÇÃO COM ÊNFASE EM ENGENHARIA DE SISTEMAS
EMBARCADOS**

ADRIEL DE SOUZA MEDEIROS

PROJETO 2 – SENSORIAMENTO CLIMÁTICO

Macaé – RJ
Setembro/2021

SUMÁRIO

1. Introdução	6
2. Objetivos do projeto	6
3. Hardware do Sistema Embarcado	8
3.1 Componentes utilizados	8
3.2 Estrutura do circuito	9
3.3 Código do Arduino	10
3.1.1 Declarações iniciais	11
3.1.2 Setup	11
3.1.3 Função do sensor DHT 11	12
3.1.4 Função do sensor LDR	12
3.1.5 Função do sensor MQ-2	12
3.1.6 Função do sensor de chuva	13
3.1.7 Função do Higrômetro	13
3.1.8 Loop	14
4. Supervisório do Sistema Embarcado	14
4.1 Desenvolvimento do Sistema Supervisório	15
4.1.1 Banco de Dados	15
4.1.2 Biblioteca jSerialComn	17
4.1.3 Interface no JAVAFX	17
4.1.4 Código em JAVA	18
4.1.4.1 Primeiros passos	18
4.1.4.2 Classe	20
4.1.4.3 Funções relacionadas ao banco de dados	22
4.1.4.4 Função do botão de conectar	24
4.1.4.5 Função do botão de desconectar	27
4.1.4.6 Função de preenchimento da ListView	27
5. Testes Gerais	28
6. Conclusão	36

LISTA DE FIGURAS

Figura 1 Sensor DHT11	7
Figura 2 Sensor Higrômetro Shield	7
Figura 3 Sensor de gás MQ-2	7
Figura 4 Sensor de chuva	8
Figura 5 Sensor LDR	8
Figura 6 Estrutura do circuito vista pela lateral	9
Figura 7 Estrutura do circuito vista de cima	10
Figura 8 Declarações iniciais do código Arduino	11
Figura 9 Função Setup	11
Figura 10 Função do sensor DHT 11	12
Figura 11 Função do sensor LDR	12
Figura 12 Função do sensor de gás MQ-2	13
Figura 13 Função do sensor de chuva	13
Figura 14 Função do higrômetro shield	14
Figura 15 Função loop	14
Figura 16 Tabela do banco de dados	16
Figura 17 Estrutura dos dados no banco de dados	16
Figura 18 Interface desenvolvida no JAVAFOX	17
Figura 19 Bibliotecas utilizadas	18
Figura 20 Importações iniciais	19
Figura 21 Declarações iniciais	19
Figura 22 Inicialize do controler	20
Figura 23 Função carregarPortas	20
Figura 24 Classe Registro	21
Figura 25 Função getConexao	22
Figura 26 Função getPreparedStatement	22
Figura 27 Função getDateTime	22
Figura 28 Função gravar	23
Figura 29 Função consultarDados	23
Figura 30 Função btnConectarAction	25
Figura 31 Função btnDesconectarAction	27

Figura 32 Função preencherLista_____	28
Figura 33 Estrutura final do circuito em funcionamento _____	28
Figura 34 Estrutura final do sensor DHT11 _____	29
Figura 35 Realização de teste no sensor LDR_____	29
Figura 36 Realização de teste no sensor de gás MQ-2 _____	30
Figura 37 realização de teste em caso de chuva leve no sensor de chuva ____	30
Figura 38 Realização de teste com o caso de chuva intensa no sensor de chuva _____	31
Figura 39 Realização de teste do higrômetro com areia _____	31
Figura 40 Realização de teste do higrômetro com barro vermelho _____	32
Figura 41 Realização de teste do higrômetro com terra preta _____	32
Figura 42 Sistema supervisorio ao ser iniciado_____	33
Figura 43 Sistema supervisorio com a ComboBox para apresentação das portas conectadas _____	34
Figura 44 Sistema supervisorio ao realizar a conexão com a porta selecionada _____	35
Figura 45 Sistema supervisorio ao ser desconectado da porta selecionada_	36

LISTA DE TABELAS

<i>Tabela 1 Componentes utilizados na construção do circuito</i>	9
--	---

1. Introdução

Um sistema embarcado é um sistema microprocessado no qual o computador é completamente encapsulado ou dedicado ao dispositivo ou sistema que ele controla. Ao contrário dos computadores de uso geral, como os computadores pessoais, os sistemas embarcados executam um conjunto de tarefas predefinidas, geralmente com requisitos específicos. De modo geral, tais sistemas não podem alterar suas funções durante o uso. Caso seja necessário mudar o propósito, deve-se reprogramar todo o sistema.

Os sistemas embarcados estão ficando mais baratos, mais fáceis de acessar, requerem menos consumo de energia e, além de serem mais compactos, também têm capacidades de processamento mais fortes. Com esse poder de processamento cada vez maior, o mundo em que vivemos se tornará cada vez mais micro conectado, onde não apenas os computadores podem acessar a Internet, mas também os objetos ao nosso redor. Até um futuro próximo, todos nós viveremos em um mundo onde a fronteira entre realidade e virtualidade é muito tênue, senão imperceptível. Esse progresso e o progresso que já estão diretamente relacionados aos sistemas embarcados.

Durante este trabalho, será exposto o exemplo de aplicação de um sistema embarcado realizar o monitoramento climático com diferentes sensores avaliando diferentes variáveis climáticas de forma automatizada. Sendo apresentado e explicado todo processo de montagem e programação do sistema distribuído, além do processo de comunicação e construção de um sistema supervisorio utilizando a linguagem JAVA.

2. Objetivos do projeto

Desenvolver um sistema completo (Embarcado + Supervisorio) para monitorar e registrar dados climáticos.

- Sistema embarcado

Os sensores utilizados são:

DHT11 (temperatura e umidade ambiente)

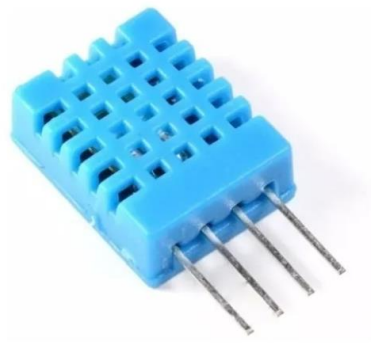


Figura 1 Sensor DHT11

Higrômetro shield (umidade do solo)

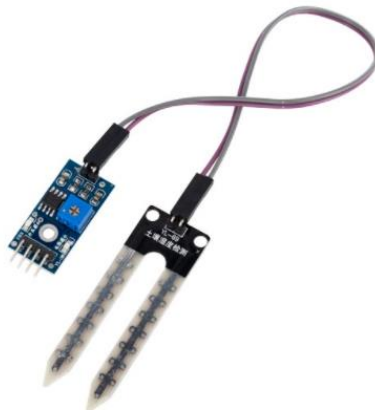


Figura 2 Sensor Higrômetro Shield

Sensor de gás MQ-2 (Qualidade do ar, detectando fumaça de queimada)



Figura 3 Sensor de gás MQ-2

Sensor de chuva



Figura 4 Sensor de chuva

LDR como sensor de Luz



Figura 5 Sensor LDR

- Sistema supervisorio

Registrar os dados dos sensores em tabela do banco de dados, a cada um ciclo de tempo (Ex.: A cada 1 minuto). Além de interface para o usuário apresentado o status da conexão, e o histórico de registro dos dados de todos os sensores.

3. Hardware do Sistema Embarcado

3.1 Componentes utilizados

Os componentes utilizados pelos circuitos para realizar as tarefas deste relatório estão listados na tabela abaixo:

Componentes	Quantidade
Arduino Uno R3	1
Placa de ensaio pequena	1
Fonte de energia externa	1

Sensor DHT 11	1
Sensor LDR	1
Resistor 10 kΩ	1
Sensor de gás MQ-2	1
Sensor de chuva	1
Higrômetro Shield	1

Tabela 1 Componentes utilizados na construção do circuito

3.2 Estrutura do circuito

O circuito utilizado neste projeto foi desenvolvido diretamente em sua forma física devido a não existência dos sensores necessários no simulador tinkercad. Além do fato de ser necessário a conexão do sistema embarcado com um computador, situação impossível ao se utilizar o simulador. Sendo assim, o resultado da estrutura do circuito é apresentado nas Figuras 6 e 7.

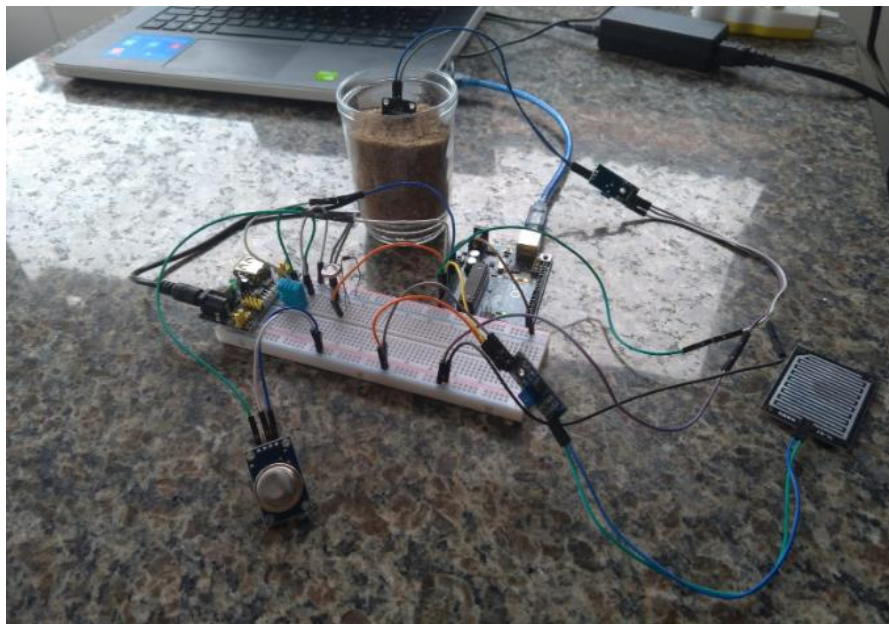


Figura 6 Estrutura do circuito vista pela lateral

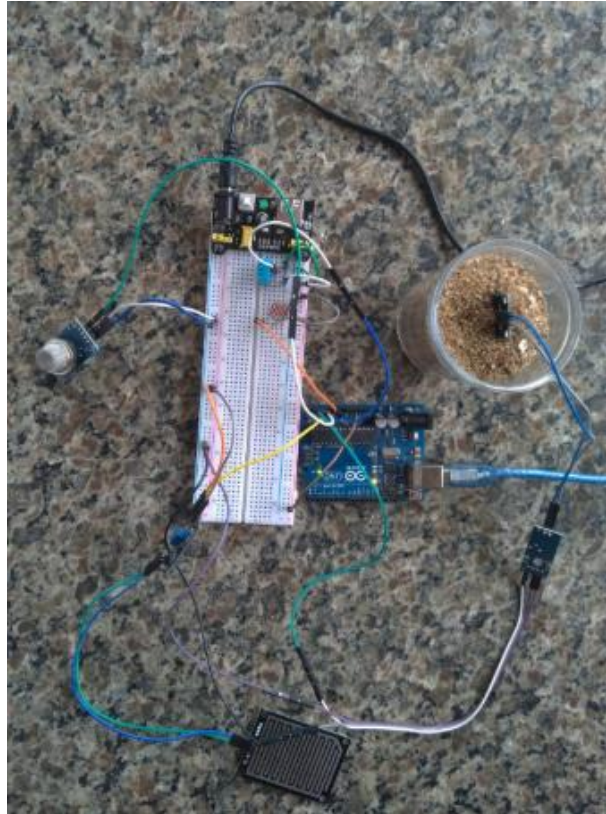


Figura 7 Estrutura do circuito vista de cima

3.3 Código do Arduino

Por se tratar de um circuito que contém muitos componentes, cada um dos componentes foi dividido em uma função que realiza a leitura e impressão do valor analógico do mesmo no monitor serial. Resultando assim na chamada dessas funções na função loop, o que torna o código muito mais legível.

Os comentários contidos no código foram mantidos por se tratar do tratamento dos dados lidos dos sensores para impressão de resultado final no monitor serial do Arduino, sendo de extrema importância caso não seja adotado um sistema supervisorio, e foi o caso dos primeiros passos desse projeto, tendo em vista que a construção do mesmo foi iniciada pela montagem do circuito. Mas ao desenvolver o sistema supervisorio foi decidido que seria melhor apenas ler o valor analógico e trata-lo no sistema supervisorio. Os dados de cada sensor foram somados a um “,” que auxilia no sistema supervisorio para a aplicação do método split, está parte pertinente ao desenvolvimento do supervisorio é mais detalhada no capítulo 4.

3.1.1 Declarações iniciais

Inicialmente, no código Arduino, foi feito a inclusão da biblioteca necessária para a utilização do sensor DHT11. Em seguida foi feito o instanciamento de variáveis que armazenam o valor referente a entrada analógica que cada sensor foi conectado. Conforme pode ser observado na Figura 8.

```
#include "DHT.h"

// DHT 11 -----
#define DHTPIN A0
#define DHTTYPE DHT11
const int pinoDHT11 = A0;
DHT dht(DHTPIN, DHTTYPE);
// -----

//LDR -----
const int pinoLDR = A1;
// -----

// MQ-2 -----
#define MQ_analog A2
// -----

//Sensor de chuva -----
const int pinoSensorChuva = A3;
// -----

// Higrômetro -----
const int pinoSensorHigrometro = A4;
// -----
```

Figura 8 Declarações iniciais do código Arduino

3.1.2 Setup

Na função setup (Figura 9), foi realizada a inicialização de cada uma das portas que os sensores foram conectados, além da inicialização da porta serial. E por último a realização de um delay de 2 segundos para garantir que o processo ocorra de forma segura.

```
void setup() {
  Serial.begin(2000000);
  dht.begin(); //DHT 11
  pinMode(pinoLDR, INPUT); // LDR
  pinMode(MQ_analog, INPUT); // MQ-2
  pinMode(pinoSensorChuva, INPUT); //Sensor de chuva
  pinMode(pinoSensorHigrometro, INPUT); //Higrômetro
  delay(2000);
}
```

Figura 9 Função Setup

3.1.3 Função do sensor DHT 11

A função do sensor DHT11 (Figura 10) utiliza da biblioteca importada para realizar a leitura do valor da umidade do ar e a temperatura ambiente no momento, printando este dado no monitor serial.

```
void sensor_DHT11() {  
  // Serial.print("Umidade: ");  
  Serial.print((String)dht.readHumidity()+"");  
  // Serial.print("%");  
  // Serial.print(" / Temperatura: ");  
  Serial.print((String) (dht.readTemperature())+"");  
  // Serial.println("°C");  
}
```

Figura 10 Função do sensor DHT 11

3.1.4 Função do sensor LDR

A função do sensor LDR (Figura 12) realiza a leitura do valor analógico do sensor e realiza o print do valor no monitor serial.

```
void sensor_LDR() {  
  int valor_analogico = analogRead(pinoLDR);  
  Serial.print((String)valor_analogico+"");  
  // if(valor_analogico > 350){  
  //   Serial.println("Noite");  
  // }  
  // else{  
  //   Serial.println("Dia");  
  // }  
}
```

Figura 11 Função do sensor LDR

3.1.5 Função do sensor MQ-2

A função do sensor MQ-2 (Figura 12) realiza a leitura do valor analógico do sensor e realiza o print do valor no monitor serial.

```

void sensor_MQ_2(){
    int valor_analogico = analogRead(MQ_analog);
    Serial.print((String)valor_analogico+");");
    // if(valor_analogico >100){
    //     Serial.println("Indicio de incêndio");
    // }else{
    //     Serial.println("sem indicio de incêndio");
    // }
}

```

Figura 12 Função do sensor de gás MQ-2

3.1.6 Função do sensor de chuva

A função do sensor de chuva (Figura 13) realiza a leitura do valor analógico do sensor e realiza o print do valor no monitor serial.

```

void sensor_de_chuva(){
    int valor_analogico = analogRead(pinoSensorChuva);
    Serial.print((String)valor_analogico+");");
    // if(valor_analogico<900 && valor_analogico>300){
    //     Serial.println("Chuva leve");
    // }else if(valor_analogico<300){
    //     Serial.println("Chuva intensa");
    // }else{
    //     Serial.println("Não chove no momento");
    // }
}

```

Figura 13 Função do sensor de chuva

3.1.7 Função do Higrômetro

A função do sensor higrômetro shield (Figura 14) realiza a leitura do valor analógico do sensor e realiza o print do valor no monitor serial.

```

void sensor_higrometro() {
    int valor_analogico = analogRead(pinoSensorHigrometro);

    Serial.print( (String)valor_analogico+"");
    //
    // if(valor_analogico > 0 && valor_analogico < 400){
    //     Serial.println("umido");
    // }
    // if (valor_analogico > 400 && valor_analogico < 800){
    //     Serial.println("com umidade moderada");
    // }
    // if (valor_analogico > 800 && valor_analogico < 1024){
    //     Serial.println("seco");
    // }
}

```

Figura 14 Função do higrômetro shield

3.1.8 Loop

A função loop (Figura 15) acabou tendo como tarefa final apenas a chamada das funções de cada um dos sensores. Além de realizar no final das chamadas a um delay que define de quanto em quanto tempo o monitoramento climático vai ser realizado pelos sensores.

```

void loop() {
    // Serial.println("----- Sensor DHT 11 -----");
    sensor_DHT11();
    // Serial.println("----- Sensor LDR -----");
    sensor_LDR();
    // Serial.println("----- Sensor MQ-2 -----");
    sensor_MQ_2();
    // Serial.println("----- Sensor de chuva -----");
    sensor_de_chuva();
    // Serial.println("---- Sensor de Humidade do solo -----");
    sensor_higrometro();
    // Serial.println("\n\n\n\n");
    delay(1000);
}

```

Figura 15 Função loop

4. Supervisório do Sistema Embarcado

O sistema supervisorio tem como principal objetivo a aquisição, supervisão, controle e apresentação dos dados obtidos por um sistema embarcado, facilitando assim a visualização e análise desses dados obtidos. Tendo em vista que os sistemas embarcados podem muitas vezes se encontrarem em locais de difícil acesso, sendo o sistema supervisorio um excelente software para monitoramento desse sistema embarcado.

Na resolução deste projeto, foi desenvolvido um sistema supervisorio capaz de conectar e desconectar da porta serial do Arduino através de uma interface computacional, além de apresentar os resultados de leitura de sensores climáticos nesta interface. Os detalhes da construção deste software são abordados nos tópicos seguintes.

4.1 Desenvolvimento do Sistema Supervisorio

No desenvolvimento deste sistema supervisorio se fez necessário o uso de algumas tecnologias, sendo elas: PostgreSQL, Linguagem de programação JAVA, biblioteca jSerialComn e a biblioteca JAVAFOX.

4.1.1 Banco de Dados

O banco de dados utilizado para armazenar as informações lidas da porta serial do Arduino do sistema embarcado desenvolvido neste trabalho foi o PostgreSQL 11, juntamente com a interface PgAdmin4 para modelagem do banco e visualização dos dados salvos no mesmo.

Para este projeto, fez-se necessário a utilização de um banco simples, contendo uma tabela e sete colunas (Figura 16). Sendo utilizada para salvar os 6 dados lidos pelos 5 sensores, além do registro de data e hora da conferência dos dados climáticos.

registro ✕

General Columns Advanced Constraints Parameters Security SQL

Inherited from table(s)

Columns							+
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	
	umidade	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	temperatura	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	ldr	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	mq_2	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	chuva	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	higrometro	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	
	data_hora	character varying			<input type="button" value="No"/>	<input type="button" value="No"/>	

i
?
✕ Cancel
↺ Reset
💾 Save

Figura 16 Tabela do banco de dados

A forma como os dados foram salvos no banco pode ser observado na Figura 17.

	umidade character varying	temperatura character varying	ldr character varying	mq_2 character varying	chuva character varying	higrometro character varying	data_hora character varying
67	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:39
68	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:40
69	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:42
70	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:43
71	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:44
72	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:46
73	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:47
74	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:48
75	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:50
76	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:51
77	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:52
78	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:54
79	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:55
80	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:56
81	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:57
82	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:04:59
83	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:01
84	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:02
85	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:03
86	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:04
87	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:05
88	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:06
89	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:09
90	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:10
91	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:11
92	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:12
93	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:13
94	42.00	31.00	Noite	não existe indicio de in...	Não chove	seco	20/09/2021 21:05:14

Figura 17 Estrutura dos dados no banco de dados

4.1.2 Biblioteca jSerialComn

A biblioteca jSerialComn destina-se a ser uma biblioteca JAVA que fornece de uma maneira independente acessar portas seriais padrão sem exigir bibliotecas externas, código nativo ou qualquer outra ferramenta. Tendo maior facilidade de uso com relação às outras alternativas encontradas para fazer ao acesso as portas seriais encontradas atualmente, e contendo ainda um suporte aprimorado para intervalos de tempo e a capacidade de abrir várias portas simultaneamente.

4.1.3 Interface no JAVA FX

Para o desenvolvimento da interface do sistema supervisor foi utilizada a biblioteca JAVA FX do JAVA. Contendo esta interface uma ComboBox para listagem das portas seriais conectadas no computador, um botão para realizar a desconexão da porta, um botão para conexão com a porta selecionada, além de uma ListView para apresentar os dados vindo do banco. O resultando final da interface pode ser visto na Figura 18.

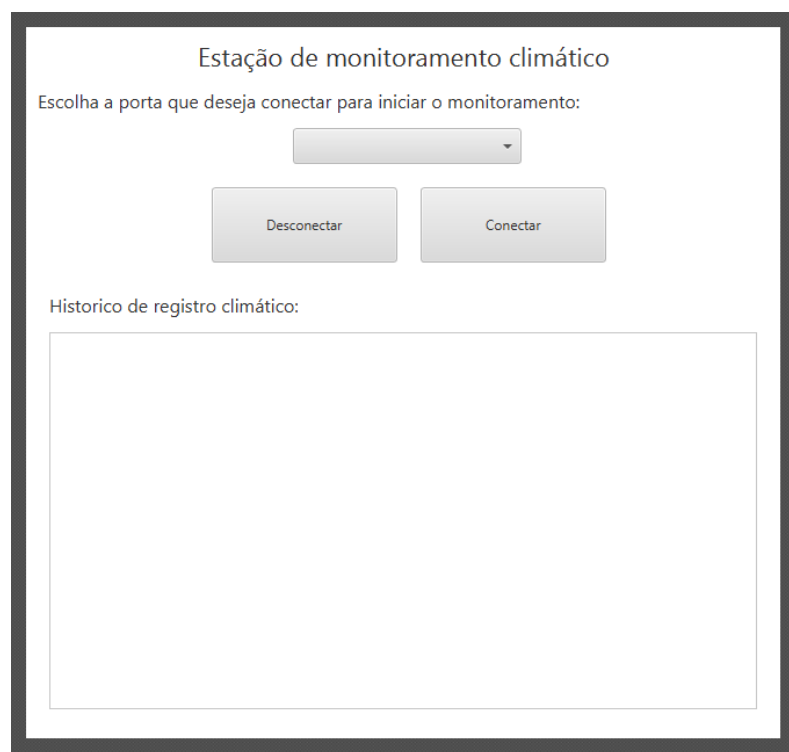


Figura 18 Interface desenvolvida no JAVA FX

4.1.4 Código em JAVA

No desenvolvimento deste supervisor foi utilizada a linguagem de programação JAVA, explorando sua biblioteca jSerialComn para leitura da porta serial do Arduino, e o JAVAFX para criação de interface.

4.1.4.1 Primeiros passos

Inicialmente, foi construído um projeto javaFX utilizando o JAVA 8, e importada a biblioteca jSerialComn no projeto, além da importação do driver JDBC do PostgreSQL, sendo este driver necessário para a comunicação com o banco de dados, conforme mostra a Figura 19.

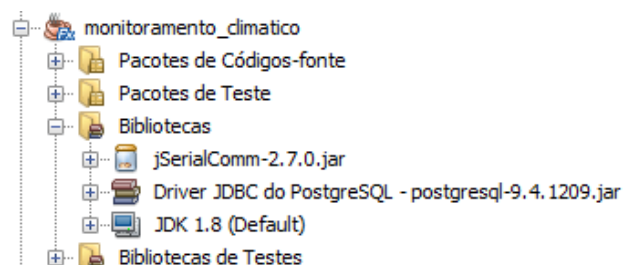


Figura 19 Bibliotecas utilizadas

Dentro do código, foi feita as importações necessárias de funcionalidades tanto do JAVA, como do JAVAFX, e da utilização do banco de dados, que são necessárias no projeto (Figura 20). Além da declaração da ComboBox, o Button de conectar e o de desconectar, a ListView do tipo Registro, a SerialPort, a Thread e um objeto da classe registro (Figura 21)

```

1 package monitoramento_climatico;
2
3 import com.fazecast.jSerialComm.SerialPort;
4 import java.io.InputStream;
5 import java.net.URL;
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.PreparedStatement;
9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import java.sql.Statement;
12 import java.text.DateFormat;
13 import java.text.SimpleDateFormat;
14 import java.util.ArrayList;
15 import java.util.Date;
16 import java.util.List;
17 import java.util.ResourceBundle;
18 import javafx.collections.FXCollections;
19 import javafx.collections.ObservableList;
20 import javafx.fxml.FXML;
21 import javafx.fxml.Initializable;
22 import javafx.scene.control.Button;
23 import javafx.scene.control.ComboBox;
24 import javafx.scene.control.ListView;

```

Figura 20 Importações iniciais

```

28 @FXML
29 private ComboBox cbPortas;
30
31 @FXML
32 private Button btnConectar;
33
34 @FXML
35 private Button btnDesconectar;
36
37 @FXML
38 private ListView lstRegistros;
39
40 private SerialPort porta;
41
42 Thread thread;
43
44 Registro r = new Registro();

```

Figura 21 Declarações iniciais

O “initialize” do “controler” realiza a chamada das funções preencherLista, carregarPortas e a getConexao (Figura 22).

```

48  @Override
49  public void initialize(URL url, ResourceBundle rb) {
50      preencherLista();
51      carregarPortas();
52      try {
53          getConexao();
54      } catch (SQLException ex) {
55          Logger.getLogger(FXMLDocumentController.class.getName()).log(Level.SEVERE, null, ex);
56      }
57  }

```

Figura 22 Initialize do controler

A função `carregarPortas` realiza um “for” buscando as portas que possuem conexão no computador (Figura 23), salvando-as para preencher a `ComboBox` com o nome dessas portas. Já as outras funções chamadas no “initialize” serão explicadas nos tópicos seguintes.

```

59  private void carregarPortas() {
60      SerialPort[] portNames = SerialPort.getCommPorts();
61
62      for (SerialPort portName : portNames) {
63          cbPortas.getItems().add(portName.getSystemPortName());
64      }
65  }

```

Figura 23 Função `carregarPortas`

4.1.4.2 Classe

A classe registro foi desenvolvida para realizar a manipulação e controle dos dados que vão ser salvos e consultados do banco de dados. Dentro de sua estrutura encontra-se a declaração das variáveis referentes as colunas do banco de dados, ou métodos “get” e “set” de cada uma delas, além do método de retorno no formato desejado para salvar na `ListView`. A estrutura da classe registro pode ser observada na Figura 24.

```

1 package monitoramento_climatico;
2
3 public class Registro {
4
5     private String umidade;
6     private String temperatura;
7     private String ldr;
8     private String mq_2;
9     private String chuva;
10    private String higrometro;
11    private String data_hora;
12
13    public String getUmidade() {
14        return umidade;
15    }
16
17    public void setUmidade(String umidade) {
18        this.umidade = umidade;
19    }
20
21    public String getTemperatura() {
22        return temperatura;
23    }
24
25    public void setTemperatura(String temperatura) {
26        this.temperatura = temperatura;
27    }
28
29    public String getLdr() {
30        return ldr;
31    }
32
33    public void setLdr(String ldr) {
34        this.ldr = ldr;
35    }
36
37    public String getMq_2() {
38        return mq_2;
39    }
40
41    public void setMq_2(String mq_2) {
42        this.mq_2 = mq_2;
43    }
44
45    public String getChuva() {
46        return chuva;
47    }
48
49    public void setChuva(String chuva) {
50        this.chuva = chuva;
51    }
52
53    public String getHigrometro() {
54        return higrometro;
55    }
56
57    public void setHigrometro(String higrometro) {
58        this.higrometro = higrometro;
59    }
60
61    public String getData_hora() {
62        return data_hora;
63    }
64
65    public void setData_hora(String data_hora) {
66        this.data_hora = data_hora;
67    }
68
69    @Override
70    public String toString() {
71        return "-----
72            + "\t \t \t \t Histórico de Monitoramento Climático \n"
73            + "Umidade do ar: " + umidade + "%" + "\n"
74            + "Temperatura: " + temperatura + "°C" + "\n"
75            + "Atualmente é " + ldr + "\n"
76            + "A condição do ar indica que " + mq_2 + "\n"
77            + "No momento " + chuva + "\n"
78            + "O solo está " + higrometro + "\n"
79            + "Data e hora do registro das condições climáticas: " + data_hora + "\n"
80            + "-----
81    }
82 }
83

```

Figura 24 Classe Registro

4.1.4.3 Funções relacionadas ao banco de dados

As funções relacionadas ao banco de dados são: getConexao, getPreparedStatement, getDateTime, gravar e consultarDados.

As funções getConexao (Figura 25), e getPreparedStatement (Figura 26) são as responsáveis por estabelecer a conexão com o banco de dados.

```
128 |  
129 |  
130 |  
131 |  
132 |  
protected Connection getConexao() throws SQLException {  
    String url = "jdbc:postgresql://" + "localhost" + ":" + "5432" + "/" + "monitoramentoclimatico";  
    Connection conn = DriverManager.getConnection(url, "postgres", "postgres");  
    return conn;  
}
```

Figura 25 Função getConexao

```
134 |  
135 |  
136 |  
137 |  
138 |  
139 |  
140 |  
141 |  
142 |  
protected PreparedStatement getPreparedStatement(boolean chavePrimaria, String sql) throws Exception {  
    PreparedStatement ps = null;  
    if (chavePrimaria) {  
        ps = getConexao().prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);  
    } else {  
        ps = getConexao().prepareStatement(sql);  
    }  
    return ps;  
}
```

Figura 26 Função getPreparedStatement

Já a função getDateTime (Figura 27), é responsável por disponibilizar a data e hora atual no formato correto para salva-la no banco quando necessário.

```
144 |  
145 |  
146 |  
147 |  
148 |  
private String getDateTime() {  
    DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");  
    Date date = new Date();  
    return dateFormat.format(date);  
}
```

Figura 27 Função getDateTime

A função gravar (Figura 28), quando chamada, realiza a gravação no banco de dados dos dados recebidos dos sensores, além da data e hora atual do sistema. Nesta função que está sendo feito os testes que inicialmente ocorria no código do Arduino, para com base no valor definido registrar se é noite ou dia, existe indicio de incêndio ou não, se chove intensamente, moderado ou não chove,

além de dizer se o solo está úmido, seco ou com umidade moderada.

```
150 public void gravar(String umidade, String temperatura, String ldr, String mq_2, String chuva, String higrometro) throws Exception {
151     String sql = "insert into registro(umidade, temperatura, ldr, mq_2, chuva, higrometro, data_hora) values (?, ?, ?, ?, ?, ?, ?)";
152     PreparedStatement ps = getPreparedStatement(false, sql);
153
154     if ((Integer.parseInt(ldr) > 350) {
155         ldr = "Noite";
156     } else {
157         ldr = "Dia";
158     }
159
160     if ((Integer.parseInt(mq_2)) > 100) {
161         mq_2 = "existe indicio de incêndio";
162     } else {
163         mq_2 = "não existe indicio de incêndio";
164     }
165
166     if ((Integer.parseInt(chuva)) < 900 && (Integer.parseInt(chuva)) > 300) {
167         chuva = "Chove leve";
168     } else if ((Integer.parseInt(chuva)) < 300) {
169         chuva = "Chove intensamente";
170     } else {
171         chuva = "Não chove";
172     }
173
174     if ((Integer.parseInt(higrometro)) > 0 && (Integer.parseInt(higrometro)) < 400) {
175         higrometro = "umido";
176     } else if ((Integer.parseInt(higrometro)) > 400 && (Integer.parseInt(higrometro)) < 800) {
177         higrometro = "com umidade moderada";
178     } else if ((Integer.parseInt(higrometro)) > 800 && (Integer.parseInt(higrometro)) < 1024) {
179         higrometro = "seco";
180     }
181
182     ps.setString(1, umidade);
183     ps.setString(2, temperatura);
184     ps.setString(3, ldr);
185     ps.setString(4, mq_2);
186     ps.setString(5, chuva);
187     ps.setString(6, higrometro);
188     ps.setString(7, getDateTime());
189     ps.executeUpdate();
190 }
```

Figura 28 Função gravar

A função consultarDados (Figura 29), quando chamada, realiza a busca dos dados atuais contidos no banco de dados e os retorna.

```
192 public List<Registro> consultarDados() throws Exception {
193     String sql = "SELECT * FROM registro order by data_hora desc";
194     PreparedStatement ps = getPreparedStatement(false, sql);
195
196     ResultSet rs = ps.executeQuery();
197
198     List<Registro> registros = new ArrayList<Registro>();
199     while (rs.next()) {
200         Registro registro = new Registro();
201         registro.setUmidade(rs.getString("umidade"));
202         registro.setTemperatura(rs.getString("temperatura"));
203         registro.setLdr(rs.getString("ldr"));
204         registro.setMq_2(rs.getString("mq_2"));
205         registro.setChuva(rs.getString("chuva"));
206         registro.setHigrometro(rs.getString("higrometro"));
207         registro.setData_hora(rs.getString("data_hora"));
208         registros.add(registro);
209     }
210     return registros;
211 }
```

Figura 29 Função consultarDados

4.1.4.4 Função do botão de conectar

A ação do clique no botão conectar chama o método `btnConectarAction` (Figura 30), que começa realizando o preenchimento da `ListView` com os dados mais recentes do banco de dados, e em seguida é feita a atribuição do valor do `SerialPort`, começando com um `getCommPort` da porta selecionada na `ComboBox`, e em seguida um `setComPortTimeouts` com o parâmetro `TIMEOUT_READ_SEMI_BLOCKING`, além de 0 milissegundos nos dois parâmetros subsequentes, que são referentes ao tempo limite de leitura de semi-bloqueio. Após esta etapa é feito o `setBaudRate` para ajustar a velocidade de acordo com a configurada na IDE do Arduino, como parâmetro foi passado o valor mais alto possível, 2000000, pois foi o que apresentou o melhor desempenho de comunicação com a porta serial. Após estas etapas, é declarado um `InputStream` que recebe como valor o `getInputStream` da porta selecionada.


```

69 @FXML
70 private void btnConectarAction() throws SQLException {
71     preencherLista();
72
73     porta = SerialPort.getCommPort(cbPortas.getSelectionModel().getSelectedItem().toString());
74     porta.setComPortTimeouts(SerialPort.TIMEOUT_READ_SEMI_BLOCKING, 0, 0);
75     porta.setBaudRate(2000000);
76     InputStream in = porta.getInputStream();
77
78     thread = new Thread() {
79         public void run() {
80             int availableBytes = 0;
81             do {
82                 result = ""; //Zerando o valor da variavel para a proxima leitura
83                 try {
84                     availableBytes = porta.bytesAvailable();
85                     if (availableBytes > 0) {
86                         byte[] buffer = new byte[1024];
87                         int bytesRead = porta.readBytes(buffer, Math.min(buffer.length, porta.bytesAvailable()));
88                         String response = new String(buffer, 0, bytesRead);
89                         result = response;
90                     }
91                     Thread.sleep(60000); // 1 min
92                     in.close();
93                     if (result.length() > 20 && result.length() < 50) {
94                         String array[] = new String[6];
95                         array = result.split(",");
96                         String umidade = array[0];
97                         String temperatura = array[1];
98                         String ldr = array[2];
99                         String mq_2 = array[3];
100                         String chuva = array[4];
101                         String higrometro = array[5];
102                         gravar(umidade, temperatura, ldr, mq_2, chuva, higrometro);
103                     }
104                 } catch (Exception e) {
105                     e.printStackTrace();
106                 }
107             } while (availableBytes > -1);
108         }
109     };
110     porta.openPort();
111     cbPortas.setDisable(true);
112     thread.start();
113     btnConectar.setDisable(true);
114     btnDesconectar.setDisable(false);
115 }

```

Figura 30 Função btnConectarAction

Após essas declarações, uma Thread é criada para ser executada durante todo o tempo que o software está em execução. Dentro desta Thread inicialmente é feito a declaração de uma variável do tipo String para armazenamento do valor lido da porta serial a cada consulta do valor dos sensores.

Dentro da função principal da Thread, é feito inicialmente a declaração da variável que armazena o valor referente a quantidade de bytes que estão sendo lidos pela porta serial. Esta variável é utilizada para garantir a execução do “do while” que ocorre logo em seguida a sua declaração, caso essa variável possua valor maior que -1,

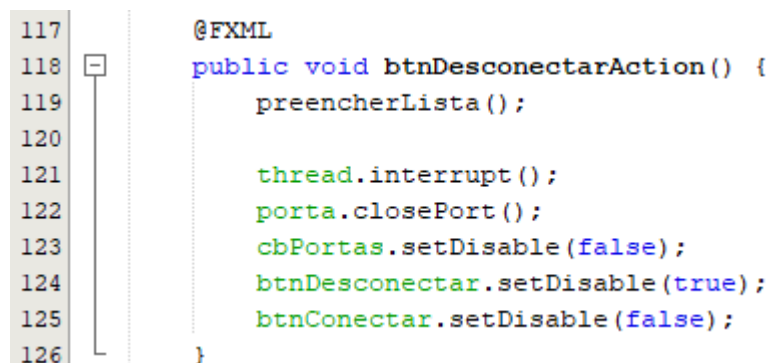
significa que a comunicação com a porta serial está sendo feita, e o “do while” fica sendo executado, caso contrário, o “do while” é encerrado.

Dentro do “do while”, primeiro o valor da variável que armazena o valor lido da porta serial é igualado a zero e a variável de armazenamento da quantidade de bytes é setada com o bytesAvailable da porta selecionada, e em seguida um “if” confirma se este valor é maior do que 0. Com a condição do “if” atendida, um array de byte é criado recebendo o valor de 1024, e em seguida é criada uma variável inteira chamada “bytesRead” que recebe a leitura dos bytes vindos da porta serial. Após esta etapa, a string vinda da porta serial é atribuída a uma variável chamada “response”, e em seguida o valor de response é atribuído a variável string criada no início da Thread, sendo esta atribuição necessária para poder utilizar o valor lido da porta serial fora do “if” atual. Em seguida o Thread.sleep com o atributo 60000 (1 minuto) é chamado e o InputStream é fechado. Ainda dentro da Thread, e no “do while”, o valor da variável que armazena a mensagem vinda da porta serial é testado, mas especificamente o seu tamanho, para saber se ele é maior que 20 e menor que 50, este teste garante que a mensagem que vai ser gravada não se trata de um bug gerado nos primeiros momentos de execução onde a comunicação serial ainda está sendo estabelecida. Com esse teste aprovado, um array de string de 6 posições é criado para armazenar o valor de cada um dos sensores vindo do monitor serial, inicialmente todos os valores fazem parte de uma única string, separados por “;”, ao se aplicar a função “split” esses valores são separados e armazenado cada um deles a cada uma das posições do array. Cada um dos valores dos sensores referentes a cada uma das posições do array são atribuídos a variáveis que possuem o

nome mais intuitivo, relacionado ao sensor que este valor pertence, e essas variáveis passadas como parâmetro para a função de gravação dos dados no banco. Todo este ciclo é encerrado quando o availableBytes obtém o valor maior que -1, finalizando assim o loop. Fora da Thread a porta selecionada recebe um openPort, a comboBox é desabilitada, a Thread é iniciada, o botão de conectar é desabilitado para evitar o duplo clique do usuário e o botão de desconectar é habilitado.

4.1.4.5 Função do botão de desconectar

O botão de desconectar chama a função btnDesconectarAction (Figura 31), que por sua vez, realiza o preenchimento da ListView com os dados mais recentes do banco de dados, realiza um interrupt na Thread, e fecha a porta conectada. Sendo que essa ação faz com que a variável availableBytes receba o valor de -1, parando assim o loop que acontece na Thread contida na função do botão de conectar. Após isso, a ComboBox e o botão de conectar são habilitados novamente, e o botão de desconectar é desabilitado para impedir o duplo clique pelo usuário.



```
117
118 ☐
119
120
121
122
123
124
125
126
```

```
@FXML
public void btnDesconectarAction() {
    preencherLista();

    thread.interrupt();
    porta.closePort();
    cbPortas.setDisable(false);
    btnDesconectar.setDisable(true);
    btnConectar.setDisable(false);
}
```

Figura 31 Função btnDesconectarAction

4.1.4.6 Função de preenchimento da ListView

A função de preenchimento da ListView (Figura 32) inicia declarando uma lista de Registro chamada registros, que vai ser utilizada para armazenar os registros do banco. Em seguida essa lista recebe o valor retornado do chamado

da função consultarDados, que retorna os dados do banco de Dados. Após esta etapa um ObservableList de Registro chamado data é criado recebendo FXCollections.observableArrayList da lista de registros. Por fim, a ListView é setada com os valores contidos no ObservableList data.

```
213 private void preencherLista() {  
214     List<Registro> registros;  
215     try {  
216         registros = consultarDados();  
217         ObservableList<Registro> data = FXCollections.observableArrayList(registros);  
218         lstRegistros.setItems(data);  
219     } catch (Exception e) {  
220         e.printStackTrace();  
221     }  
222 }  
223 }  
224 }
```

Figura 32 Função preencherLista

5. Testes Gerais

No desenvolvimento deste projeto foram utilizadas algumas técnicas para conseguir simular alterações de valores detectados por cada um deles. Além de definição de um tempo pequeno entre cada leitura, para se alcançar assim um maior volume de dados em um menor tempo de execução. O circuito final montado e ligado pode ser observado na Figura 33.



Figura 33 Estrutura final do circuito em funcionamento

O resultado final da montagem do sensor de temperatura pode ser observado na Figura 34.

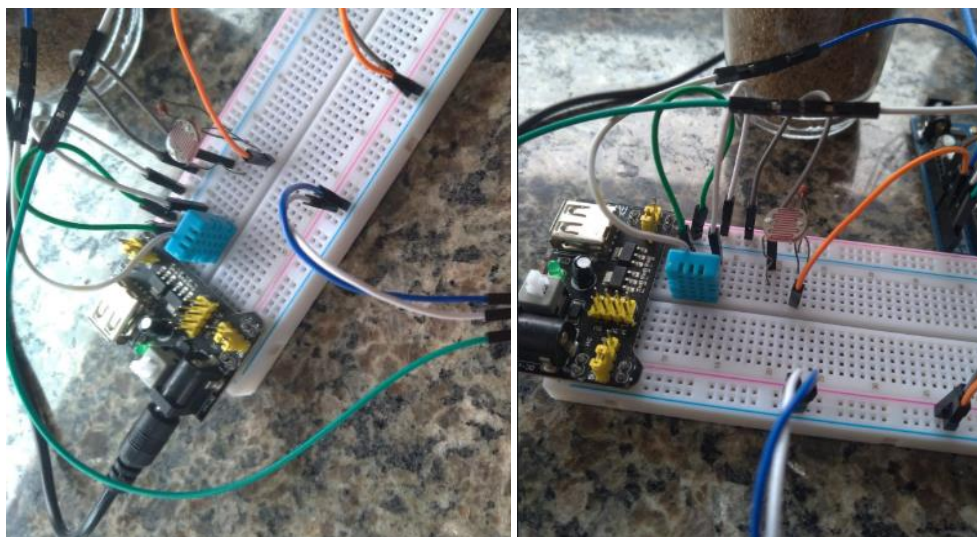


Figura 34 Estrutura final do sensor DHT11

Para simular situação de dia e noite com o sensor LDR, foi inserido o dedo sobre o sensor para impedir a chegada de luz até o mesmo, e simular assim a situação de dia e noite. O resultado final do circuito com sua simulação pode ser observado na Figura 35.

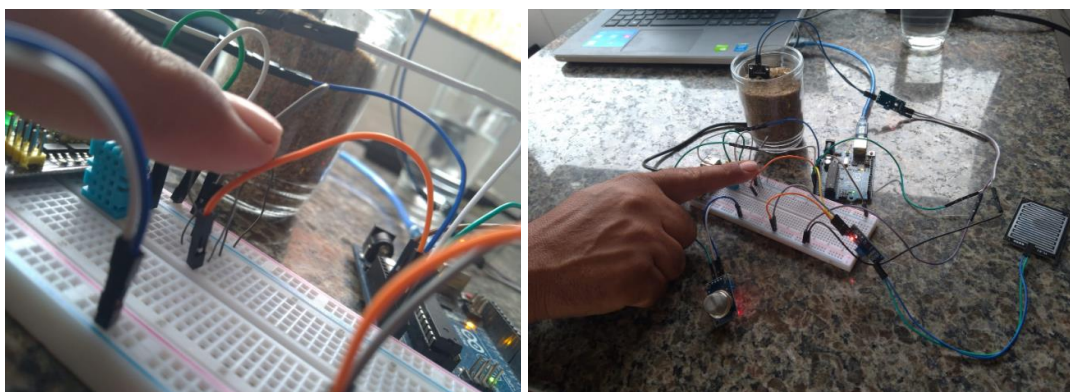


Figura 35 Realização de teste no sensor LDR

Com o sensor MQ-2, fez-se necessário a utilização de um isqueiro, onde com a mão foi criado um campo livre de vento, e pressionado o botão para liberar o gás do isqueiro próximo ao sensor, sem que o vento o espalhasse rapidamente. Conforme pode ser observado pela Figura 36.

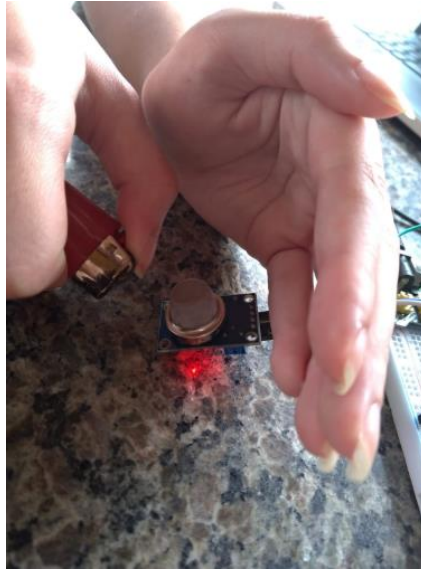


Figura 36 Realização de teste no sensor de gás MQ-2

Além do caso de não chover, ou seja, sensor seco, outros dois casos foram simulados, o de chuva leve (Figura 37), e chuva intensa (Figura 38).

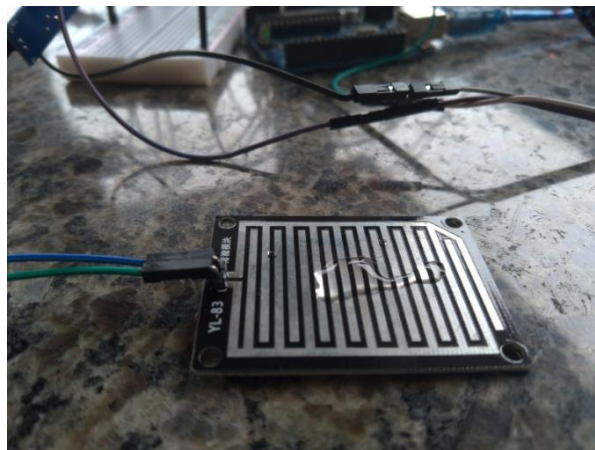


Figura 37 realização de teste em caso de chuva leve no sensor de chuva

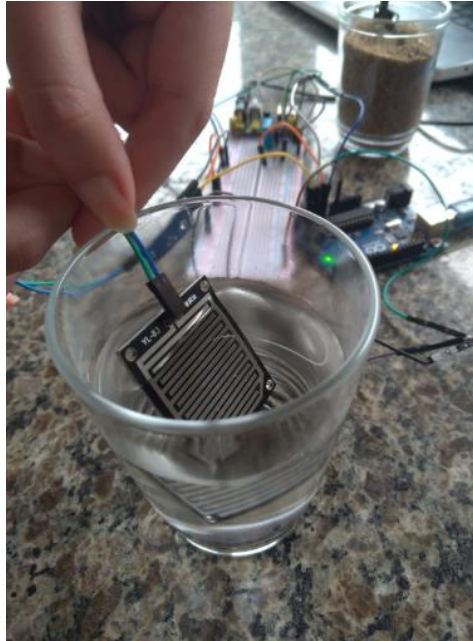


Figura 38 Realização de teste com o caso de chuva intensa no sensor de chuva

Os testes com o higrômetro shield foram um pouco mais trabalhosos. Ao se realizar o teste com areia (Figura 39), a leitura analógica não chegava abaixo de 500.

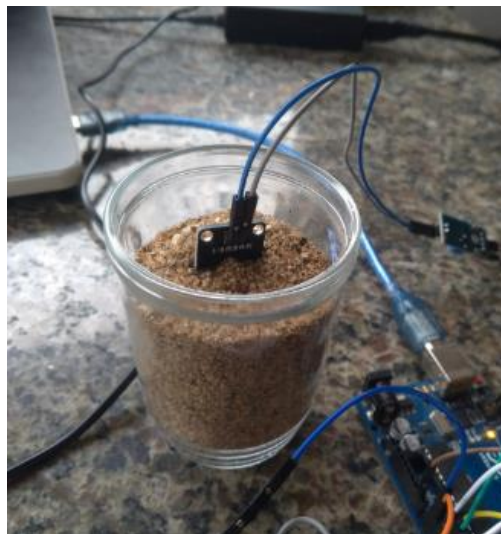


Figura 39 Realização de teste do higrômetro com areia

Os testes do higrômetro shield com barro vermelho (Figura 40) apresentaram um resultado relativamente melhor que a areia, mas mesmo assim não chegou abaixo de 430.

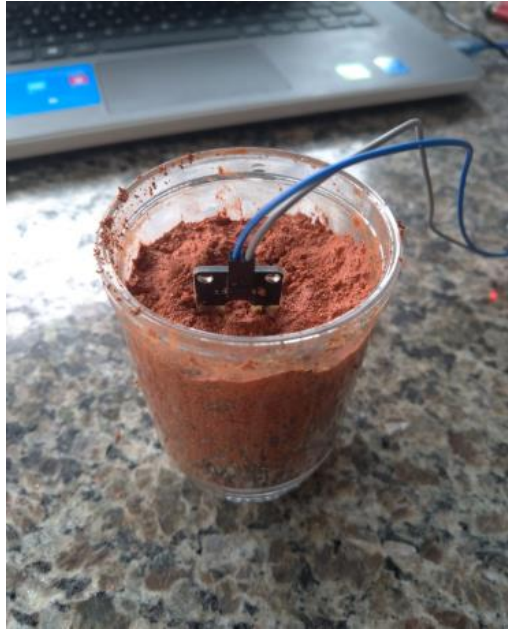


Figura 40 Realização de teste do higrômetro com barro vermelho

O tipo de solo que possibilitou alcançar um valor bem baixo na leitura analógica foi a terra preta (Figura 41), a mesma utilizada em jardins. Este tipo de solo ao ser umedecido possibilitou uma leitura mínima de 90.



Figura 41 Realização de teste do higrômetro com terra preta

A leitura mais completa na terra preta se deu pelo fato do sensor depender dos sais minerais da terra para fazer a análise, não apenas a quantidade de água no solo. Com isso, a terra preta, por ser mais rica, possibilitou alcançar valores mais baixos.

O sistema supervisorio em execução é apresentado nas imagens que estão a seguir (Figura: 42, 43, 44 e 45), contendo nelas a ilustração de como ele se comporta durante sua execução. Com os dados atuais do banco sendo apresentados na interface, controle das portas conectadas no computador e o controle dos botões para evitar o uso indevido do mesmo pelo usuário.

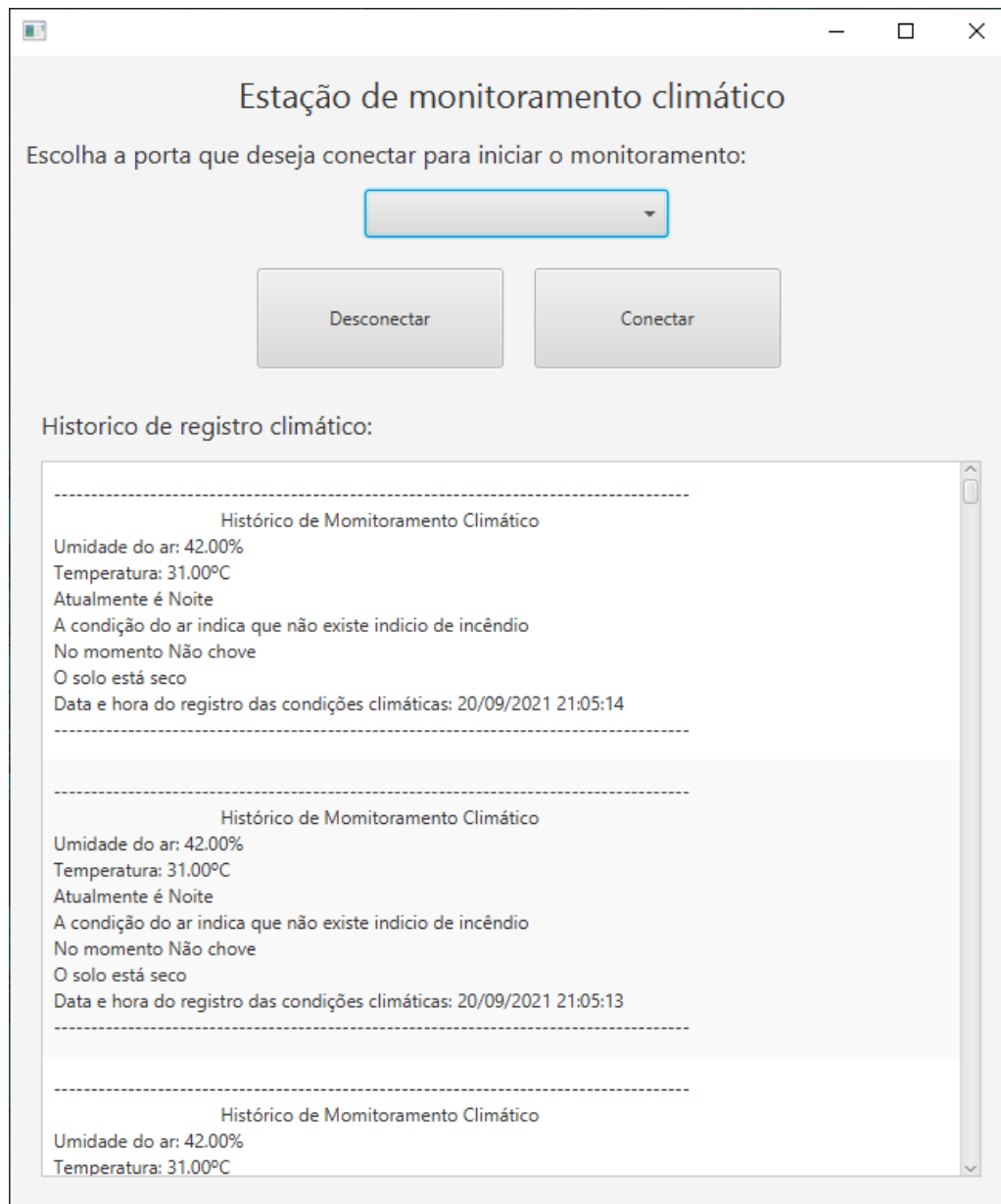


Figura 42 Sistema supervisorio ao ser iniciado

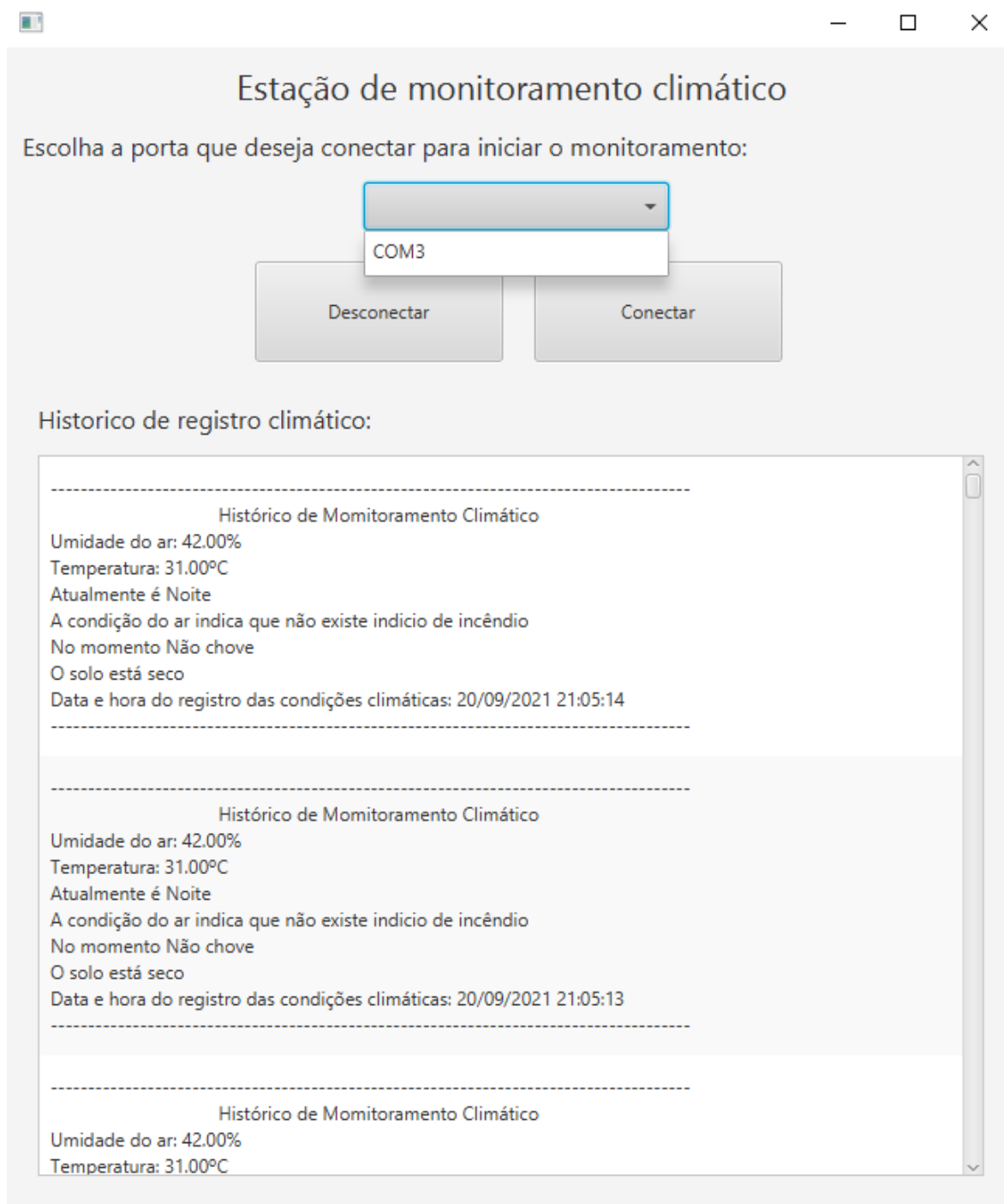


Figura 43 Sistema supervisorio com a ComboBox para apresentação das portas conectadas

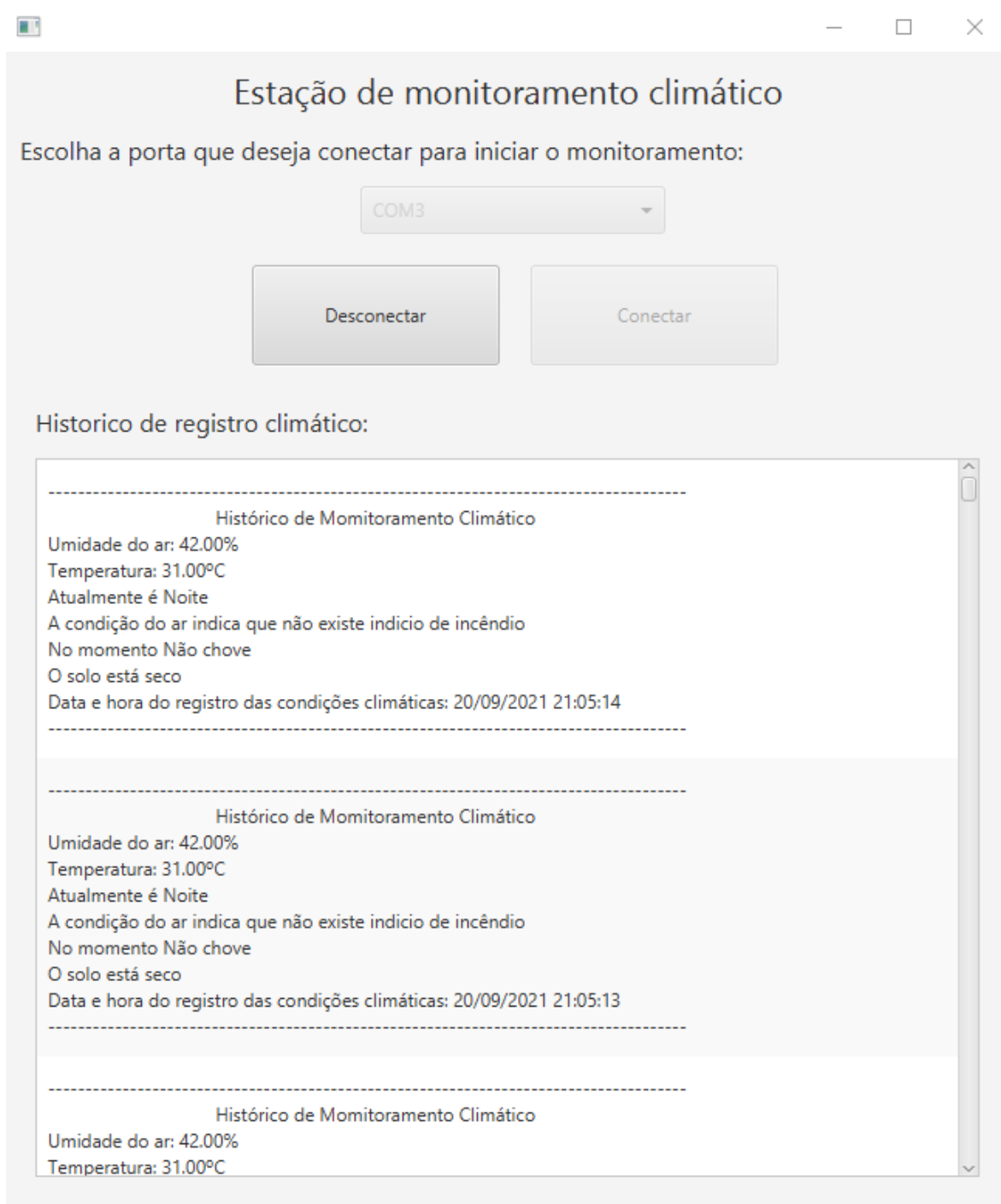


Figura 44 Sistema supervisorio ao realizar a conexão com a porta selecionada

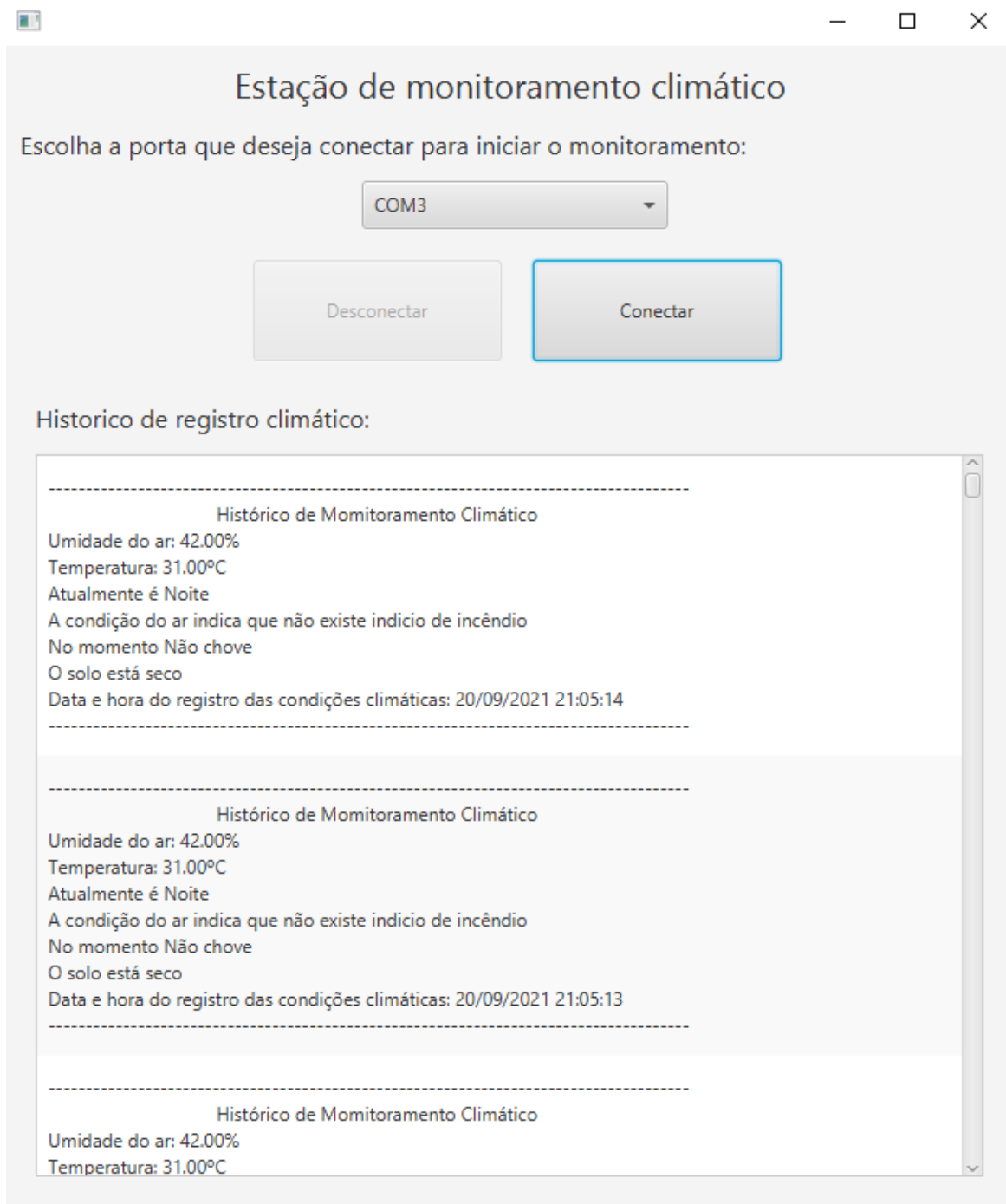


Figura 45 Sistema supervisorio ao ser desconectado da porta selecionada

6. Conclusão

Os sistemas embarcados podem ser aplicados em diversas situações, e vem se tornando mais popular a cada dia. Este avanço e aumento de popularidade faz com que os sistemas embarcados sejam implantados no dia a dia das pessoas, sem mesmo que consigam perceber, como por exemplo o controle antibloqueio das rodas do carro.

Este projeto abordou uma das funcionalidades dos sistemas embarcados, que é a sua utilização, composto por sensores, para fazer o monitoramento climático. Esta é uma solução relativamente simples e barata, com um impacto muito grande em situações que dependem dos dados das variáveis climáticas do local.

Todo o desenvolvimento do sistema embarcado, como do sistema supervisor, foi detalhado ao decorrer deste documento, possibilitando assim a percepção de como ocorre todo o processo de criação deste produto. Sendo este, um trabalho acadêmico que o que estimula o aluno a aplicar seus conhecimentos teóricos no desenvolvimento de soluções para os problemas do mundo real. Amadurecendo assim a mentalidade do aluno a levar seus conhecimentos para resolução de problemas reais, e criação de produtos e soluções para o mercado.