

FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS  
CURSO DE GRADUAÇÃO EM SISTEMA DE INFORMAÇÃO

SISTEMA PARA PREVENÇÃO A INCÊNDIO MARÍTIMO UTILIZANDO  
HARDWARES E LINGUAGENS DE PROGRAMAÇÃO LIVRES

POR:  
FELLIPE AZEVEDO PORFIRIO MAIA

MACAÉ  
2018

FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS  
CURSO DE GRADUAÇÃO EM SISTEMA DE INFORMAÇÃO

FELLIPE AZEVEDO PORFIRIO MAIA

SISTEMA PARA PREVENÇÃO A INCÊNDIO MARÍTIMO UTILIZANDO  
HARDWARES E LINGUAGENS DE PROGRAMAÇÃO LIVRES

Trabalho Final apresentado ao curso de graduação em Sistema de Informação, da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), para obtenção do grau de BACHAREL em Sistema de Informação.

Professor Orientador: Irineu Lima

MACAÉ/RJ

2018

FELLIPE AZEVEDO PORFIRIO MAIA

SISTEMA PARA PREVENÇÃO A INCÊNDIO MARÍTIMO UTILIZANDO  
HARDWARES E LINGUAGENS DE PROGRAMAÇÃO LIVRES

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistema de Informação, da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), para obtenção do grau de BACHAREL em Sistema de Informação.

Aprovada em \_\_\_\_ de \_\_\_\_\_ de 20\_\_\_\_

BANCA EXAMINADORA

---

Prof. Irineu de Azevedo Lima Neto  
Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)  
1º Examinador

---

Prof. Anselmo Costa  
Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)  
2º Examinador

## **DEDICATÓRIA**

Gostaria de dedicar este trabalho ao meu irmão Raphael A. P. Maia (*in memorian*), que com pesar não estará comigo fisicamente nesta e futuras conquistas, mas estará sempre em meu coração. Dedico também este trabalho à minha mãe Fátima A. P. Maia, mulher lutadora, forte e que sempre vibra com as minhas vitórias e nunca mediu esforços por minha educação.

## **AGRADECIMENTO**

Agradeço a Verônica S. Rocha, que esteve comigo nestes últimos anos me ajudando a dedicar mais tempo para as atividades e estudo das matérias, pois sem ela não estaria concluindo neste ano a faculdade. E ao meu orientador, Irineu A. L. Neto, pela paciência na orientação e incentivo que tornaram possível a conclusão desta monografia. A todos os meus professores da faculdade, que foram essenciais na minha trajetória acadêmica. Ao Curso de Sistema de Informação da FeMASS e às pessoas com quem convivi nesses espaços ao longo desses anos.

## **EPÍGRAFE**

Não é a linguagem de programação que define o  
programador, mas sim sua lógica.  
David Ribeiro Guilherme

## RESUMO

O sistema de combate a incêndio é um dos elementos da segurança em unidades marítimas, onde há sempre o risco devido à presença de produtos químicos, entre outros produtos inflamáveis. Para monitorar e prevenir incêndios, há normas nacionais e internacionais para instalação de rede de sensores. No geral, os equipamentos mínimos exigidos são: detectores de fumaça, gases, temperatura e luminosidade, geralmente atrelados a equipamentos proprietários de alto custo e de pouca mão de obra disponível para manutenção. Logo, justifica o presente projeto utilizando *hardware* livre e linguagem *open source*, visando a redução de custo e maior disponibilidade de mão de obra para manutenção. A elaboração do estudo de caso proposto teve como objetivo estabelecer uma rede de sensores úteis ao combate a incêndio, além de um sistema supervisório para registro dos dados adquiridos e geração de relatórios, visando auxiliar um operador em tomadas de decisões. O supervisório foi projetado para trabalhar em conjunto com o Arduino e seus sensores de campo estabelecidos, e a comunicação entre estes será feita por *shield Ethernet* e protocolo UDP entre cliente-servidor devido às distâncias entre os pontos de monitoração. O estudo de caso desenvolvido é simulado para testes de uma situação real, determinando o contexto de possível aplicabilidade. Os dados gerados nos testes foram gravados em um banco de dados e posteriormente exibidos no *dashboard* desenvolvido para relatório. Os resultados avaliados demonstraram a viabilidade do projeto na captura dos dados monitorados, permitindo discussões a respeito e necessidades de aprimoramento do protótipo.

Palavras-chave: Sistema de combate a incêndio, Supervisório, Arduino, *dashboard*.

## ABSTRACT

The firefighting system is one of the elements of safety in maritime units, where there is always the risk because the presence of chemicals, and other flammable products. There are national and international standards for sensor network installation to monitor and prevent fires. In general, the minimum equipment required is: smoke, gas, temperature and light detectors, usually tied to high-cost, proprietary equipment and generally is not easy to find qualified people to labor with its maintenance. Therefore, it justifies the present project using free hardware and open source language, aiming a cost reduction and greater availability of labor for maintenance. The purpose of the proposed case study is to establish a network of sensors useful for firefighting, as well as a supervisory system for recording acquired data and generating reports, assisting an operator in decision making. The supervisory computer will work in conjunction with the Arduino and its established field sensors, and the network communication will be done by Ethernet shield and client-server UDP protocol due to the distances between the monitoring points. The case study developed is simulated to test a real situation, determining the context of possible applicability. The data generated in the tests were recorded in a database and later displayed in the dashboard developed for reporting. The results were evaluated and demonstrated the feasibility of the project in the capture of the monitored data, allowing discussions about it and the needs of the prototype improvement.

Keywords: Firefighting system, Supervisory, Arduino, *dashboard*.



## LISTA DE FIGURAS

FIGURA 1: ARDUINO UNO.....	20
FIGURA 2: FUNÇÕES CARACTERÍSTICAS DOS DIFERENTES TERMOPARES.....	22
FIGURA 3: SENSOR DE RADIAÇÃO LUMINOSA COMO OPÇÃO PARA DETECÇÃO DE CHAMAS. ....	23
FIGURA 4: CARACTERÍSTICAS SENSITIVAS DO SENSOR MQ-6, ENTRE CONCENTRAÇÃO DE GÁS NO AMBIENTE E A RELAÇÃO DE RESISTÊNCIA ESPERADA.....	23
FIGURA 5: EXEMPLO DE <i>SHIELD ETHERNET</i> VIA PAR TRANSADO. ....	24
FIGURA 6: DIAGRAMA DO CIRCUITO ELETRÔNICO.....	26
FIGURA 7: CÓDIGO QUE DEFINE OS PINOS DE COMUNICAÇÃO.....	26
FIGURA 8: CÓDIGO DE LEITURA DO SENSOR DE TEMPERATURA (TERMOPAR).....	27
FIGURA 9: COMANDOS DE LEITURA DO PINO ANALÓGICO .....	27
FIGURA 10: FUNÇÃO DE LEITURA MÉDIA DO SENSOR DE LUMINOSIDADE NO ARDUINO. ....	28
FIGURA 11: COMANDOS DA BIBLIOTECA MQ2.H PARA SENSOR DE FUMAÇA NO ARDUINO. ÊNFASE PARA O COMANDO <i>READSMOKE</i> . ....	28
FIGURA 12: TRECHO DE CÓDIGO ARDUINO PARA INICIALIZAÇÃO DA PLACA <i>ETHERNET</i> PARA FINS DE COMUNICAÇÃO EM REDE. ....	29
FIGURA 13: CÓDIGO DE CONFIGURAÇÃO DE IP E ATRIBUTOS DE CONEXÃO À REDE <i>ETHERNET</i> NO ARDUINO.....	29
FIGURA 14: CÓDIGO DE RECEBIMENTO DE MENSAGEM .....	30
FIGURA 15: CÓDIGO PARA SOLICITAR A COMUNICAÇÃO COM O SERVIDOR .....	31
FIGURA 16: CONFIGURANDO OS SENSORES DE LUZ E FUMAÇA .....	31
FIGURA 17: CONSTRUÇÃO E ENVIO DA <i>STRING</i> . ....	32
FIGURA 18: LEITURA DOS SENSORES.....	32
FIGURA 19: ENVIO DOS DADOS PARA O SERVIDOR.....	33
FIGURA 20: INICIANDO UM THREAD DA CLASSE "SERVIDORSOCKET" .....	34
FIGURA 21: PARTE DO CÓDIGO DO MÉTODO "RUN" .....	35
FIGURA 22: PARTE DO CÓDIGO DO MÉTODO "RUN" .....	35
FIGURA 23: PARTE DO CÓDIGO DO MÉTODO "RUN" .....	36
FIGURA 24: PÁGINA DO <i>DASHBOARD</i> . ....	37
FIGURA 25: HTML DO ARQUIVO INDEX.....	37
FIGURA 26: CÓDIGO DA FUNÇÃO "SELECTGRAFIC" – PARA GERAÇÃO DE GRÁFICO DE PROPRIEDADE DE SENSOR DESEJADA NA PÁGINA <i>DASHBOARD</i> .....	38
FIGURA 27: CÓDIGO DA CLASSE "CHARTBEAN" MÉTODO "BUSCARDADOS".....	38
FIGURA 28: PARTE DO CÓDIGO DE BUSCA DA CLASSE "DAO DADOS" – CONSULTA SQL FORMATADA PARA OBTENÇÃO DE VALOR MÉDIO DE PROPRIEDADES DE SENSORES CONFORME O INTERVALO DE DATA E LOCAL DESEJADOS. ....	39
FIGURA 29: PARTE DO CÓDIGO DE BUSCA DA CLASSE "DAO DADOS" – FILTRAGEM DE DADOS CONSULTADOS PARA USO EM VARIÁVEIS ÚTEIS À GERAÇÃO DE GRÁFICO DE PROPRIEDADES NA PÁGINA <i>INDEX</i> . ....	39
FIGURA 30: CÓDIGO PARA RETORNAR A LISTA DE TEMPO .....	40
FIGURA 31: SCRIPT PÓS PROCESSAMENTO DA PÁGINA "INDEX" .....	40

FIGURA 32: JAVASCRIPT PARA CRIAÇÃO DO GRÁFICO NA PÁGINA <i>INDEX</i> .....	41
FIGURA 33: CÓDIGO RESPONSÁVEL DE INICIAR A CONSTRUÇÃO DO SCRIPT .....	41
FIGURA 34: ARDUINO LOCALIZADO EM DOIS AMBIENTES DISTINTOS, COMO SALA E COZINHA DE UMA EMBARCAÇÃO. ....	42
FIGURA 35: DIAGRAMA EM BLOCO DAS CONEXÕES ENTRE OS COMPONENTES DA REDE.....	43
FIGURA 36: LEITURA VIA PORTA SERIAL DOS ARDUINOS. ....	43
FIGURA 37: TERMINAL DO NETBEANS COM OS DADOS RECEBIDOS VIA <i>SOCKET</i> .....	44
FIGURA 38: GRÁFICO DE TEMPERATURA .....	45
FIGURA 39: GRÁFICO DE LUMINOSIDADE .....	46
FIGURA 40: GRÁFICO DE FUMAÇA .....	46

## LISTA DE TABELAS

TABELA 1: PARES DE JUNÇÕES MAIS COMUNS EM TERMOPARES .....	21
TABELA 2: PRINT DA TABELA DO BANCO .....	36
TABELA 3: ESTRUTURADA APÓS A CONSULTA EM BANCO DE DADOS NO COMPUTADOR SUPERVISÓRIO, MOSTRANDO DADOS RECÉM-CAPTURADOS PELOS SENSORES INSTALADOS NOS AMBIENTES DISTINTOS. ....	44

## LISTA DE ABREVIATURAS E SIGLAS

Atmega	- Fabricante dos microcontroladores utilizados nos Arduinos.
CC	- <i>Creative Commons Attribution Share-Alike</i> (Creative Commons Atribuição-Compartilha Igual).
CI	- Circuito Integrado.
CLPs	- Controlador Lógico Programável
<i>Dashboard</i>	- Uma tela composta de uma ou mais camadas, sob a forma de um painel, com instrumentos virtuais onde se associam variáveis a serem monitoradas.
DHCP	- <i>Dynamic Host Configuration Protocol</i> (Protocolo de configuração dinâmica de hosts)
DNS	- <i>Domain Name System</i> (Sistema de nomes de domínio)
<i>Frameworks</i>	- Um conjunto de conceitos usado para resolver um problema de um domínio específico
GLP	- <i>Gnu General Public License</i> (Licença Pública Geral Gnu)
IDE	- <i>Integrated Development Environment</i> (Ambiente de desenvolvimento integrado)
IP	- Internet Protocol (Protocolo de Internet)
LGPL	- <i>Library Gnu General Public License</i> (Licença Pública Geral da Biblioteca Gnu)
MAC	- Media Access Control (Controle de acesso de mídia ou Endereço único de hardware)
<i>Offshore</i>	- O que se destina a atender às necessidades de atividades marítimas de uma empresa.
<i>Open source</i>	- Licença de Software Aberto
ppm	- Parte Por Milhão
SCADA	- <i>Supervisory Control and Data Acquisition</i> (Sistemas de Supervisão e Aquisição de Dados)
<i>Shields</i>	- Componente eletrônico construído para exercer uma atividade específica
SPI	- <i>Serial Peripheral Interface</i> (Interface Periférica Serial)
TCP/IP	- <i>Transmission Control Protocol/Internet Protocol Suite</i> (Protocolo de Controle de Transmissão / Internet Protocol Suite)
Tripulantes	- O que se destina a atender às necessidades de atividades marítimas de uma embarcação.
UDP	- <i>User Datagram Protocol</i> (Protocolo de datagrama do usuário)

## SUMÁRIO

1.	INTRODUÇÃO .....	16
1.1.	OBJETIVOS .....	16
1.2.	JUSTIFICATIVA .....	17
1.3.	METODOLOGIA DE PESQUISA .....	17
2.	REFERENCIAL TEÓRICO .....	17
2.1.	SISTEMA SUPERVISÓRIO.....	18
2.2.	CONSIDERAÇÕES SOBRE REGRAS DE ALERTAS NA PREVENÇÃO A INCÊNDIO .....	19
2.3.	PROCEDIMENTO DE COMBATE A INCÊNDIO .....	19
2.4.	ARDUINO .....	19
2.5.	SENSORES E <i>SHIELDS</i> ARDUINO .....	20
2.5.1.	TERMOPAR: .....	21
2.5.2.	SENSOR DE RADIAÇÃO LUMINOSA COMO DETECTOR DE CHAMAS: .....	22
2.5.3.	DETECTOR DE GÁS: .....	23
2.5.4.	COMUNICAÇÃO ENTRE SENSORES DE CAMPO E SUPERVISÓRIO .....	24
3.	ESTUDO DE CASO .....	24
3.1.	SITUAÇÃO PROBLEMA EM UM CENÁRIO TIPO MINIMUNDO .....	25
3.2.	PROPOSIÇÃO DO SISTEMA DE HARDWARE .....	25
3.3.	SUPERVISÓRIO E COMUNICAÇÃO DE DADOS .....	33
3.4.	COMUNICAÇÃO DE DADOS.....	33
3.5.	DASHBOARD .....	36
4.	SIMULAÇÃO E TESTES .....	42
4.1.	PREPARAÇÃO DO AMBIENTE DE SIMULAÇÃO .....	42
4.2.	SIMULAÇÃO DE CENÁRIO REAL.....	43
4.3.	ANÁLISE DOS GRÁFICOS VIA PÁGINA <i>DASHBOARD</i> .....	45
5.	CONSIDERAÇÕES FINAIS.....	47
6.	REFERÊNCIAS BIBLIOGRÁFICAS .....	49
7.	ANEXOS .....	52

<b>ANEXO A – CÓDIGO RETIRADO DO SITE BAELDUNG .....</b>	<b>52</b>
<b>ANEXO B – CÓDIGO RETIRADO DO SITE SYSTEMBASH.....</b>	<b>53</b>
<b>8. APÊNDICE .....</b>	<b>54</b>
<b>APÊNDICE A – CÓDIGO DO ARDUINO.....</b>	<b>54</b>
<b>APÊNDICE B – DIAGRAMA DE CLASSES DO SERVIDOR SOCKET .....</b>	<b>57</b>
<b>APÊNDICE C – CLASSE DADOS DO SERVIDORSOCKET .....</b>	<b>58</b>
<b>APÊNDICE D – DIAGRAMA DE CLASSES DO SERVIDOR SOCKET .....</b>	<b>59</b>
<b>APÊNDICE E – PAGINA <i>INDEX</i> .....</b>	<b>60</b>
<b>APÊNDICE F – CLASSE FACTORYCALENDARIO .....</b>	<b>62</b>
<b>APÊNDICE G – CLASSE FACTORYCHARJSDATA .....</b>	<b>63</b>
<b>APÊNDICE H – CLASSE FACTORYDATASETS.....</b>	<b>64</b>

## 1. INTRODUÇÃO

A atividade de exploração e produção de petróleo (E&P) em plataformas marítimas tem vários riscos associados à atividade e ao ambiente em que estão, um desses riscos é o de incêndio ou explosão (GUAIANO, 2014). Neste contexto, há o agravante da evacuação da plataforma no caso de incêndio, logo os tripulantes necessitam ser capazes de extinguir ou retardar o incêndio, permitindo uma evacuação segura (GUAIANO, 2014; BRASIL, 1978). Uma das formas de se detectar com rapidez o incêndio é através dos sensores e botoeira do sistema de combate a incêndio (FSS, 2010).

O sistema de prevenção a incêndio tem que ser capaz de monitorar o funcionamento e o ambiente, em especial, onde o calor gerado pelos equipamentos pode iniciar a ignição de certos vapores de combustíveis ou gases inflamáveis, gerando uma sequência de eventos no sistema para isolar ou extinguir o princípio de incêndio (CONTECH, 2016; CONTECH, 2017).

O sistema de prevenção a incêndio é um dos mais importantes da embarcação marítima, pois segundo NBR 17240:2010 (ABNT, 2010) e FSS (2010), a rede de sensores deve estar em todos os locais da plataforma, desde a área operacional até as acomodações da embarcação. Um exemplo é de uma planta de processo de separação do óleo cru, que tem seus próprios sensores e painel de controle, pois em caso de perda de controle pode ocorrer uma catástrofe. Assim, os sensores do sistema de incêndio localizados na planta captariam as alterações das variáveis medidas, disparando o alarme a depender do sensor detector da anomalia (FSS, 2010; BRASIL, 1978).

Uma rede de prevenção a incêndio contém diversos sensores, alguns desses são: detectores de fumaça, gases, temperatura e luminosidade (FSS, 2010). A malha de rede de sensores, os controladores lógicos e o sistema supervisório têm um alto custo de implementação e manutenção, pois são geralmente aplicadas tecnologias proprietárias (PALMIERE, 2016).

### 1.1. Objetivos

Este trabalho tem o objetivo geral de desenvolver um exemplo de malha de prevenção a incêndio e um sistema supervisório, na proposição de um estudo de caso. Para tal construção, serão utilizados *hardwares* e linguagens de programação livres para redução de

custos, livre difusão de conhecimentos acessíveis para a geração de mão-de-obra qualificada, e a minimização de uso de tecnologias proprietárias para o suporte do sistema.

Os objetivos específicos são:

- Elaborar um estudo de caso para utilização de *hardware* da plataforma Arduino com o controlador Atmega (controlador para comunicação de sensores de campo), aplicando a um contexto simulado de plataforma marítima de E&P.
- Efetuar um levantamento dos tipos de sensores úteis para prevenção a incêndios, averiguando a viabilidade de simulação para alarmes e avisos desejáveis.
- Criar um sistema supervisor utilizando a linguagem Java e *frameworks* para monitorar e controlar a malha de prevenção a incêndio.

## 1.2. Justificativa

Justifica-se o contexto da indústria de E&P em Macaé e o mercado de trabalho regional que demanda a inserção de profissionais de Sistemas de Informação em projetos de sistemas de automação e monitoramento. Logo, os sistemas de supervisão e controle são de grande importância, e o seu bom funcionamento é primordial para a segurança dos colaboradores *offshore*. Além disso, o conhecimento gerado acerca do tema e emprego de tecnologias livres contribui para a justificativa deste projeto.

## 1.3. Metodologia de pesquisa

Está fundamentada em pesquisa bibliográfica em livros, monografias, artigos e fóruns conhecidos sobre o tema a fim de obter conhecimento sobre o funcionamento de equipamentos e procedimentos do sistema de prevenção a incêndios em unidades marítimas. A pesquisa será fundamental para a determinação de sensores apropriados, averiguando-se a precisão de leitura de dados, utilizando o controlador em Arduino para a construção da malha de exemplo, em uma experimentação a ser proposta.

## 2. REFERENCIAL TEÓRICO



Apresenta-se um estudo bibliográfico de referência para elaborar um estudo de caso e solução de sistema para combate a incêndio. Este estudo seguirá algumas normas ou resoluções para prevenção a incêndios em embarcação marítima, definidas para a construção do modelo de ambiente em maquete e criação do sistema supervisorio, discutidas na sequência.

A resolução MSC.98(73) elaborada pelo Comitê de Segurança Marítima é responsável por descrever os requisitos necessários para a prevenção a incêndio (FSS, 2010). Segunda tal resolução, os sistemas de monitoramento marítimo devem ser compostos pelos sensores: detector de calor, fumaça ou outros detectores de gases e chamas. Porém, o sensor de chamas tem que ter sua utilização associada ao sensor de calor ou detector de fumaça. A construção do sistema deve utilizar-se de material que resista às condições do ambiente marítimo, prevenindo os efeitos da corrosão.

A norma NR-23 (BRASIL,1978) determina a necessidade de instalação de botoeira para o acionamento manual do sistema de incêndio, e a MSC.98(73) (FSS 2010) o acionamento dos alarmes sonoros e visuais no passadiço, e caso não tenha sido verificado em 2 minutos, o acionamento é estendido a todos os recintos da embarcação, considerando ambas as formas de acionamento manual ou por detecção.

## **2.1. Sistema Supervisorio**

O sistema supervisorio ou SCADA (*Supervisory Control and Data Acquisition*) tem a função de monitorar as informações dos sensores de campo, que são geridos por sistemas controladores (por exemplo, CLPs - controladores lógicos programáveis, sistemas microcontroladores – p. ex., Atmega em kits Arduino, ou sistemas microprocessadores – p. ex., arquiteturas Raspberry). Os dados de sensores são enviados via barramento de comunicação para o computador, onde se encontra o sistema supervisorio, para que sejam registrados no banco de dados como forma de histórico e exibição (JUNQUEIRA, 2003).

A interface gráfica de sistemas supervisorios pode ser construída de forma personalizada frente aos objetivos do monitoramento dos sensores. Para tal, esquemas de ligações e desenhos documentais devem demonstrar os fluxos corretos das informações, permitindo modelagem exata do mapa de ativos dos sensores de campo (JURIZATO, PEREIRA, 2002-2003).

## **2.2. Considerações sobre regras de alertas na prevenção a incêndio**

O sistema supervisório tem como parte importante a sua composição de regras de alertas. Um sistema sem regras ou com regras mal definidas pode gerar um grande congestionamento de informações (avalanche de alertas), o que não significa que a informação será de qualidade e que auxiliará o operador a tomar a decisão mais assertiva para a situação apresenta (ARAÚJO, 2010; ALMEIDA, 2010).

Como forma de reduzir as falhas humanas, iniciativas de estudos resultaram, por exemplo, nas normas internacionais EEMUA 191, ISA 18.2 que determinam a estrutura básica para a elaboração das regras de alertas a fim de melhorar a comunicação do sistema com o operador, utilizando o sistema de monitoramento (Sistemas supervisório ou Painel de Controle) com o objetivo de apontar a atenção do operador para casos mais críticos e melhorar a segurança da instalação (ARAÚJO, 2010).

## **2.3. Procedimento de combate a incêndio**

O procedimento é um passo-a-passo que orienta os colaboradores responsáveis pelas atividades de como proceder em várias situações de rotina de trabalho ou emergências. Cada instalação contém uma complexidade que deve ser observada para que possa ser criado um procedimento adequado para cada cenário observado, e este documento deve estar impresso no passadiço. Em caráter adicional, um sistema de apoio à decisão pode ser utilizado (SOLAS, 2014; GUAIANO, 2014). Entretanto, as normas nacionais e internacionais têm certas regras que devem ser seguidas, como as NBR 17240:2010 (ABNT, 2010) e FSS (2010) que determinam a localização, distribuição e os tipos de sensores que devem ser empregados no sistema de combate a incêndios.

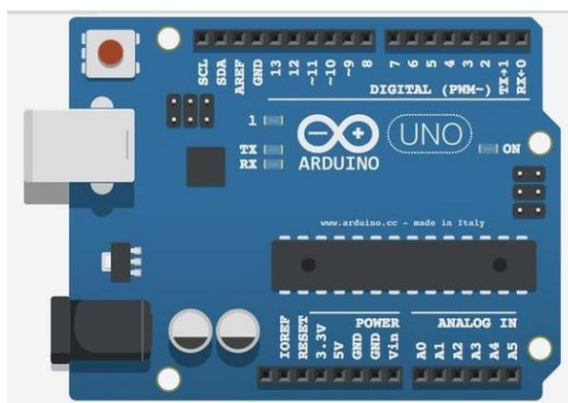
## **2.4. Arduino**

Projetado na Itália no ano de 2005, Arduino foi idealizado para ser uma plataforma livre (*open source*), onde os esquemas elétricos e codificações estão disponíveis para consulta ou reprodução, o que possibilita a qualquer pessoa estudar, aprender e otimizar o projeto para

suas necessidades (STEVAN JUNIOR; SILVA, 2015). Por ser uma plataforma livre, o Arduino está sob a licença "CC" (*Creative Commons Attribution Share-Alike*), com o seu diagrama disponível para download. O código do ambiente de desenvolvimento (IDE) e as bibliotecas em C/C++ estão sob licenças similares, sendo a IDE GLP (*Gnu General Public License*) e as Bibliotecas LGPL (*Library Gnu General Public License*) que permite ser usado em código de terceiros (ARDUINO.CC, 2018). Esses são os principais motivos do Arduino ser tão diversificado.

A plataforma Arduino (Figura 1) pode ser utilizada por diversas áreas do conhecimento e incontáveis tipos de projetos, podendo ser impressão 3D, robótica, engenharia de transportes, engenharia agrônômica, musical, moda e tantas outras (MULTILOGICA-SHOP, 2018).

**Figura 1: Arduino Uno.**



Fonte: <https://www.portalgsti.com.br/cursos/curso-aprenda-arduino/>

## 2.5. Sensores e *shields* arduino

O Arduino tem a capacidade de efetuar leituras do ambiente e transformar propriedades-alvo em sinal analógico ou digital, dependendo dos sensores que são utilizados, efetuando assim o processamento de dados conforme uma lógica prévia carregada para a memória do microcontrolador, permitindo ações através de atuadores que executam ações no meio físico (MULTILOGICA-SHOP, 2018).

Os sensores e atuadores são utilizados para o monitoramento das grandezas e atuar no meio, alguns sensores são dispostos em placas de circuitos com componentes eletrônicos adicionais ao correto funcionamento, conhecidos como *shields* para Arduino (KOMIDO,

REIS, GALLO, 2016). Nas subseções seguintes serão tratados sensores comumente empregados na prevenção a incêndios:

### 2.5.1. Termopar:

O termopar utiliza dois tipos distintos de liga metálica soldados em uma das extremidades desta, e quando em contato com o calor produz uma quantidade baixa de tensão onde é convertido por um circuito para valores legíveis para um controlador (STEVAN JUNIOR; SILVA, 2015). Pode ser composto por vários tipos distintos de liga metálica, como poder ser observado na Tabela 1.

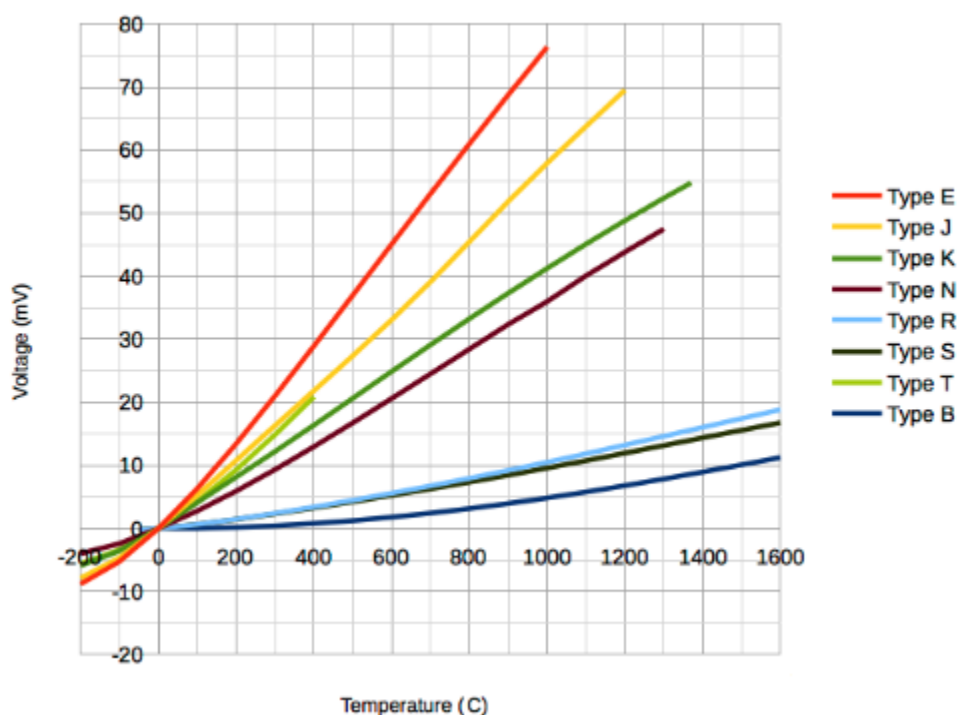
**Tabela 1: Pares de Junções mais comuns em termopares**

Referencia	Descrição	Gama (°C)
B	Platina 30% Ródio/Platina 6% Ródio	[0,1800]
E	Cromel <sup>®</sup> / Constantan <sup>®</sup>	[-200,1000]
J	Ferro/ Constantan <sup>®</sup>	[-200,900]
K	Cromel <sup>®</sup> / Alumel <sup>®</sup>	[-200,1300]
N	Nirosil <sup>®</sup> / Nisil <sup>®</sup>	[-200,1300]
R	Platina/ Platina 13% Ródio	[0,1400]
S	Platina/ Platina 10% Ródio	[0,1400]
T	Cobre/ Constantan <sup>®</sup>	[-200,400]

Fonte: adaptado de (STEVAN JUNIOR, SILVA, 2015)

Uma das possibilidades de uso é o termopar de referência K, que é composto por uma placa de cromo e alumínio que tem uma faixa de temperatura de atuação para leitura de – 200 até 1300° C, e por sua linearidade como pode ser observado na Figura 2.

**Figura 2: Funções Características dos Diferentes Termopares**



Fonte: [https://www.engineeringtoolbox.com/thermocouples-d\\_496.html](https://www.engineeringtoolbox.com/thermocouples-d_496.html)

### 2.5.2. Sensor de radiação luminosa como detector de chamas:

Sensor de radiação luminosa (Figura 3) detecta a intensidade luminosa de uma fonte de luz, como a proveniente de um incêndio, por exemplo, podendo ser de dois tipos: 1) os resistivos, que alteram a resistência elétrica do sensor dependendo intensidade da luz sobre o mesmo, necessitando de um circuito eletrônico para converter em um sinal legível; e 2) os fotovoltaicos, que convertem energia luminosa em energia elétrica e não necessitam de circuito para conversão de energia (STEVAN JUNIOR; SILVA, 2015).

**Figura 3: Sensor de radiação luminosa como opção para detecção de chamas.**

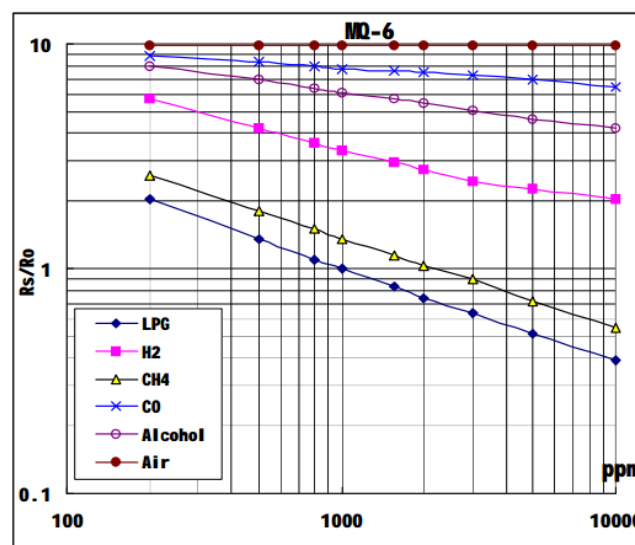


<http://tecnodomas.blogspot.com/2015/08/sensor-de-chamas-com-arduino.html>

### 2.5.3. Detector de gás:

O sensor de detector de gás mais difundido na arquitetura do Arduino é o modelo MQ-X, que utiliza um componente eletroquímico que em contato com o elemento gasoso a ser medido, irá conduzir menos ou mais corrente elétrica (Figura 4) (PINTO, 2016). Para medir esse sinal, é necessário utilizar um circuito eletrônico para converter um sinal em mV para um sinal legível ao microcontrolador através de uma porta analógica, variando de 0 a 5 V (PINTO, 2016).

**Figura 4: Características sensitivas do sensor MQ-6, entre concentração de gás no ambiente e a relação de resistência esperada.**



Fonte: <https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-6.pdf>

#### 2.5.4. Comunicação entre sensores de campo e supervisor

Para a comunicação entre o controlador com sistemas supervisórios a fim de transmitir os dados do monitoramento, pode-se utilizar tipos de comunicações diferentes, por exemplo, *Ethernet* via par trançado ou WIFI, comunicação serial (RS-232 ou RS-485), para a comunicação do Arduino pode ser utilizar o *shield Ethernet* (Figura 5) que se conecta via RJ-45, podendo se conectar a rede utilizando a arquitetura TCP/IP (*Transmission Control Protocol/Internet Protocol Suite*) através de IP (Protocolo de Internet) para envio e recebimento dos dados lidos, comandos enviados pelo servidor e alertas disseminados na rede (DUARTE, 2003). O protocolo UDP (*User Datagram Protocol*) é formado pelo cabeçalho e mensagem, a proposta deste protocolo é transmissão de pacotes de forma contínua mesmo que alguns pacotes se percam, cheguem desordenado ou corrompido, nestes casos de falha são simplesmente ignorados, caso o desenvolvedor não crie regras para contorná-los, logo o UDP não necessita de conexão constante (COMER, 2015). O sistema supervisório poderá ser configurado para registro das informações recebidas e assim alimentar o banco de dados de informações de histórico, permitindo a tomada de decisão posterior (JUNQUEIRA, 2003).

**Figura 5: Exemplo de *shield Ethernet* via par trançado.**



<https://www.robocore.net/loja/produtos/modulo-ethernet-enc28j60.html?newlang=english>

### 3. ESTUDO DE CASO

Neste capítulo será apresentado o cenário de estudo a ser simulado para prevenção a incêndios em formato minimundo, permitindo modelar o sistema, objeto deste estudo.

### 3.1. Situação problema em um cenário tipo minimundo

O cenário a ser considerado é uma plataforma *offshore*, necessitando de um modelo de sistema baseado em sensores para detecção de incêndio que possa ser replicado e instalado em qualquer sala de convivência dos funcionários. Assim, tais sensores deverão monitorar o ambiente sobre parâmetros de temperatura, luminosidade e fumaça. O conjunto de sensores deverão propiciar a avaliação do risco ou princípio de incêndio.

Todos os sensores deverão fazer uma leitura do ambiente em tempo definido, a cada 5 segundos, e após essa leitura, uma *string* contendo os dados formatados mais o nome do local no qual estão localizados é enviada para um sistema supervisorio, responsável pela captação e armazenamento em um banco de dados.

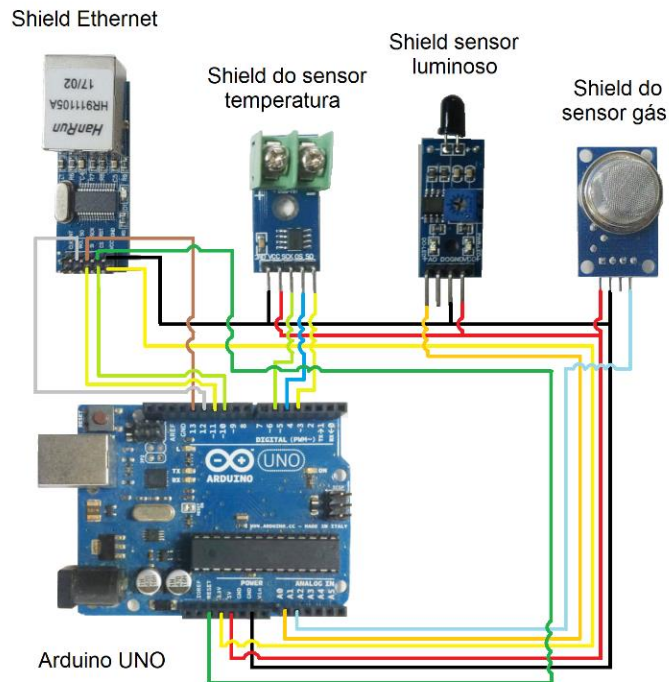
O sistema supervisorio deverá permitir a consulta dos dados cadastrados, para que um operador possa obter um *dashboard* de informações de sensores, formatadas por elementos gráficos, e geração de relatórios.

### 3.2. Proposição do sistema de hardware

Para suprir a necessidade do minimundo e as premissas propostas, de *hardware* livre e baixo custo no projeto, foi determinada a utilização de uma placa baseada em Arduino Uno com processador Atmega 328P, usada para leitura dos sensores e processamento dos dados, permitindo posterior envio para o computador supervisorio utilizando a placa complementar (denominada *shield*) de *Ethernet* para comunicação via rede de computadores TCP/IP, o protocolo de comunicação foi o UDP, como pode ser visto na seção 2.5.6,

O esquema abaixo, Figura 6, demonstra as ligações efetuadas de todos os componentes que foram utilizados no projeto.



**Figura 6: Diagrama do circuito eletrônico.**

**Imagem produzida pelo autor**

O sensor de temperatura utiliza o termopar tipo K para efetuar a leitura da temperatura do ambiente, apresentado na seção 2.5.1 e exibido na Figura 6, que possui o CI MAX 6875, responsável em converter o sinal analógico gerado pelo sensor em um sinal digital via protocolo SPI.

A biblioteca “max6675.h” deve ser usada ao desenvolver o programa Arduino para funcionamento da leitura da temperatura. Como mostra linha 20 da Figura 7, um objeto do tipo MAX6675 é iniciado onde necessita receber 3 parâmetros de entrada que são as portas digitais utilizadas no Arduino para a comunicação com o *shield*, o “SCLK” - define o pino responsável pela sincronia da comunicação (*clock*), segundo parâmetro “CS” - define o pino que ativa o envio dos dados do CI para o Arduino, e o terceiro parâmetro “MISO” - é o pino responsável por receber o valor da leitura do sensor no Arduino.

**Figura 7: Código que define os pinos de comunicação**

```

19 //MAX6675(int8_t SCLK, int8_t CS, int8_t MISO);
20 MAX6675 thermocouple(6, 5, 4);
21

```

**Imagem produzida pelo autor**

Para efetuar a leitura do sensor de temperatura (MAX6675), será utilizada a função “*readCelsius*” contida no objeto criado, e o retorno dessa função é um *double* na unidade Celsius, conforme mostra a Figura 8.

**Figura 8: Código de leitura do sensor de temperatura (termopar)**

```
double readCelsius(void);
double readFahrenheit(void);
```

**Imagem produzida pelo autor**

O sensor de luminosidade (Figura 6 e Seção 2.5.2) gera um valor entre 0 a 20 mA, o CI LM393 converte o valor gerado para um valor analógico entre 0 e 5 Volts, permitindo o Arduino interpretar o valor recebido em um pino analógico de entrada (*analog in*), traduzindo-o em uma variação decimal de 0 a 1023 (neste caso, o valor 1023 representa ausência de luz, ambiente escuro). Para ser possível a captação do valor de luminosidade pela programação, é utilizado o comando nativo “*analogRead*” que lê o sinal analógico recebido, como pode ser visto na Figura 9.

**Figura 9: Comandos de leitura do pino analógico**

```
14 | #define sensorLuz A0
    | ...
167 | int leitura=analogRead(sensorLuz);
```

**Imagem produzida pelo autor**

Para uma leitura mais precisa do sensor de luminosidade, foi desenvolvida uma função, na Figura 10, cujo objetivo é minimizar os picos de valores calculando a média das leituras definidas via parâmetros de número de leituras e intervalo de tempo.

**Figura 10: Função de leitura média do sensor de luminosidade no Arduino.**

```

162 double Luz(int num_leituras, int tempo){
163     int i;
164     double res=0;
165     for (i=0;i<num_leituras;i++)
166     {
167         int leitura=analogRead(sensorLuz);
168         res+=(double) (1023-leitura);
169         delay(tempo);
170     }
171     res/=num_leituras;
172     return res;
173 }

```

Imagem produzida pelo autor

O sensor de detecção de fumaça (na Figura 6, seção 2.5.3) tem a capacidade de medir a concentração em um ambiente usando unidade ppm (parte por milhão). Para efetuar a leitura do sensor, foi utilizada a biblioteca “MQ2.h” compatível para o referido dispositivo, compondo o programa que está sendo executado no Arduino. Assim, a Figura 11 traz os principais comandos úteis da referida biblioteca aplicada. Um objeto contendo o pino de entrada do sensor é instanciado, permitindo executar o comando de leitura “*readSmoke*”, cujo retorno da função é do tipo “*float*” e representa a concentração de fumaça em ppm.

**Figura 11: Comandos da biblioteca MQ2.h para sensor de Fumaça no Arduino. Ênfase para o comando *readSmoke*.**

```

void begin();
MQ2(int pin);
float* read(bool print);
float readLPG();
float readCO();
float readSmoke();

```

Imagem produzida pelo autor

Após leitura dos sensores aplicados, é possível formatar os dados capturados em uma *string* para envio pela rede utilizando protocolo de comunicação UDP, conforme descrito na seção 2.5.6, utilizando a placa (*shield*) *Ethernet*, que no modelo empregado utiliza o CI ENC28J60 da Microchip e biblioteca “*EtherCard.h*” para Arduino. A escolha do protocolo UDP é devido a uma limitação do CI empregado no *shield*. Para utilizar a placa *Ethernet* é necessário executar uma sequência de configurações e variáveis pré-definidas, para a

construção de um pacote de dados no momento da transmissão. Para iniciar o funcionamento do *shield* é definido qual pino irá controlar a comunicação via protocolo SPI com o Arduino, o endereço MAC e o tamanho máximo do pacote. O estabelecimento da comunicação *Ethernet* pode ser visualizado na Figura 12.

**Figura 12: Trecho de código Arduino para inicialização da placa *Ethernet* para fins de comunicação em Rede.**

```

16 #define SS 10
...
36 byte Ethernet::buffer[256];
...
64 Serial.print("Placa de rede iniciando...");
65 if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0){
66     Serial.println( "Failed to access Ethernet controller");
67 }else{
68     Serial.println("Placa de rede iniciado");
69 }

```

Imagem produzida pelo autor

Com a placa *Ethernet* iniciada, são definidos os IPs que serão utilizados para a comunicação entre os componentes da rede. Neste projeto é utilizado IP fixo que é definido através da função “*staticSetup*” da biblioteca de “*EtherCard.h*”, esta função utiliza variáveis configuradas previamente os “*myip*”, “*gwip*”, “*dns*” e “*mask*” que são variáveis estáticas que contém o IP do “*host*”, do *Gateway*, controlador DNS e máscara de subrede, respectivamente, conforme a Figura 13.

**Figura 13: Código de configuração de IP e atributos de conexão à rede *Ethernet* no Arduino.**

```

27 // ethernet interface ip address
28 static byte myip[] = { 10, 0, 0, 101 };
29 // gateway ip address
30 static byte gwip[] = { 10, 0, 0, 1 };
31 // mask
32 static byte mask[] = { 255, 255, 255, 0 };
33 //dns
34 static byte dns[] = { 10, 0, 0, 1 };
...
73 Serial.println("ip Statico definido");
74 //ether.staticSetup(myip, gwip,dns,mask);
75 ether.staticSetup(myip,gwip,dns,mask);

```

Imagem produzida pelo autor

Após a configuração inicial dos parâmetros do *shield*, a função “*udpServerListenOnPort*” da biblioteca “*EtherCard.h*” é executada, que passa como parâmetro a porta do *firewall* utilizada pelo servidor e a função “*udpSerialPrint*”, ela é utilizada pela biblioteca para entregar os dados recebidos do servidor ao programa executado no Arduino, conforme mostra a Figura 14.

**Figura 14: Código de recebimento de mensagem**

```

48 bool recebido = false;
49 //bool udpSerialPrint(word port, byte ip[4], const char *data, word len) {
50 void udpSerialPrint(word port, byte ip[4], char *data, word len) {
51     IPAddress src(ip[0], ip[1], ip[2], ip[3]);
52     recebido = true;
53 }
...
86 ether.udpServerListenOnPort(&udpSerialPrint, portServe);

```

**Imagem produzida pelo autor**

Após as configurações anteriores, é iniciada uma tentativa de comunicação com o servidor. Este procedimento é definido neste projeto a fim de verificar a comunicação inicial, tratado na Figura 15, de modo que para efetuar a comunicação foi criado um loop para conexão entre cliente (Arduino) e servidor (computador supervisor) até que haja a confirmação de conexão por parte do servidor. Assim, uma mensagem previamente definida solicitando a conexão através da palavra-comando “CONNECT::” necessita ser enviada pela função “*sendUdp*” da biblioteca do *shield Ethernet*, utilizando os parâmetros de entrada da função, que são: mensagem a ser enviada; o tamanho da mensagem, porta da rede utilizada no Arduino; IP do servidor; e a porta utilizada no servidor (que neste caso é a mesma no Arduino). Quando o servidor envia uma resposta, a função “*packetReceive*” da biblioteca “*EtherCard.h*” verifica o recebimento do pacote no Arduino, e a função “*packetLoop*” da mesma biblioteca e retorna a posição da mensagem contida no pacote. Em consequência disso, a função “*udpSerialPrint*”, como pode ser visto na Figura 14, é executada alterando o status da variável “recebido” para interrupção do loop, dando continuidade na execução do código.

**Figura 15: Código para solicitar a comunicação com o servidor**

```

87  strcpy(message, "CONNECT::");
88  int TRUE = 1;
89  while(TRUE) {
90      ether.sendUdp(message, sizeof(message), portServe, ipServe, portServe);
91      timer = millis() + 5000;
92      Serial.println("Aguardando resposta!");
93      while(millis() < timer){
94          ether.packetLoop(ether.packetReceive());
95          if(recebido){
96              Serial.println("Conexão aceita!");
97              TRUE = 0;
98              recebido = false;
99              break;
100         }
101     }
102 }

```

**Imagem produzida pelo autor**

As duas últimas configurações necessárias são do sensor de luminosidade e sensor de fumaça, como pode ser visto na Figura 16, respectivamente. Para configurar o sensor de luminosidade, é utilizado o comando nativo “*pinMode*” que determina a configuração do pino para entrada de valor e em seguida é testada a leitura com outro comando nativo o “*analogRead*”. Já o sensor de fumaça é necessário utilizar um objeto da biblioteca “MQ2.h”, iniciado previamente. Esse objeto tem uma função chamada de “*begin*” que inicia o funcionamento do sensor.

**Figura 16: Configurando os sensores de Luz e Fumaça**

```

14  #define sensorLuz A0
    ...
22  MQ2 mq2(sensorFunmaca);
    ...
106  pinMode(sensorLuz, INPUT);
107  Serial.println("Luz "+analogRead(sensorLuz));
108
109  Serial.println("Calibrando.....");
110  mq2.begin();
111  Serial.println("Calibrado.");
112

```

**Imagem produzida pelo autor**

Após a configuração do programa, pode-se iniciar a transmissão dos parâmetros lidos pelos sensores. A fim de facilitar o entendimento da mensagem sem a necessidade de conhecimento prévio do código, foi optado por enviar junto aos dados a descrições dos mesmos. Na Figura 17 mostra como foi feita a construção da *string*.

**Figura 17: Construção e envio da *string*.**

```

124 timer = millis() + 300;
125 strcpy(message, "MSG::NAME::");
126 strcat(message, nome);
127 strcat(message, "TEMP::");
128 dtostrf((media_temp/cont_leitura), 6, 2, auxChar);
129 strcat(message, auxChar);
130 strcat(message, "LUZ::");
131 dtostrf((media_luz/cont_leitura), 8, 2, auxChar);
132 strcat(message, auxChar);
133 strcat(message, "FUM::");
134 dtostrf((media_fum/cont_leitura), 6, 2, auxChar);
135 strcat(message, auxChar);
136 strcat(message, "END\0");
137 Serial.println(message);

```

Imagem produzida pelo autor

A leitura dos sensores ocorre no intervalo de tempo de captura (“*else*” – Linha 149 – Figura 18) para composição de médias e posterior montagem da *string*, conforme tratado na Figura 17.

**Figura 18: Leitura dos sensores**

```

149 }else{
150     cont_leitura++;
151     media_temp += thermocouple.readCelsius();
152     media_luz += Luz(5, 80);
153     media_fum += mq2.readSmoke();
154 }

```

Imagem produzida pelo autor

**Figura 19: Envio dos dados para o servidor**

```

140 ether.sendUdp(message, sizeof(message), portServe, ipServe, portServe );
141 for(int i =0;i<100;i++){
142     message[i] = '\0';
143 }
144 cont_leitura = 0.0;
145 media_temp = 0.0;
146 media_luz = 0.0;
147 media_fum = 0.0;

```

**Imagem produzida pelo autor**

Com a mensagem pronta para o envio, é executada a função “*sendUdp*” do objeto “*ether*” (vide Figura 19), da biblioteca do *shield Ethernet* (Figura 15). Após o envio, as variáveis são zeradas para iniciar um novo ciclo de envio, que neste caso foi definido após 300 milissegundos (Figura 17 - linha 124). A mensagem enviada é recebida pelo servidor que irá tratar os dados (na seção 3.3). O Código completo do Arduino estará no Apêndice A.

### 3.3. Supervisório e comunicação de dados

Esta seção tratará a comunicação entre o Arduino e o servidor (computador supervisorio), explicando as principais funções inerentes. O supervisorio terá dois programas em execução: 1) responsável pela captação dos dados em comunicação com o Arduino via *socket* UDP, desenvolvido em linguagem Java; e, 2) responsável por disponibilizar uma página de “*dashboard*” exibindo os dados dos sensores já cadastrados no banco de dados, cuja funcionalidade foi desenvolvida em linguagem Java e *frameworks* JSF (para construção da página *web*) e Bootstrap (para estilos). Os detalhes serão discutidos a seguir.

### 3.4. Comunicação de dados

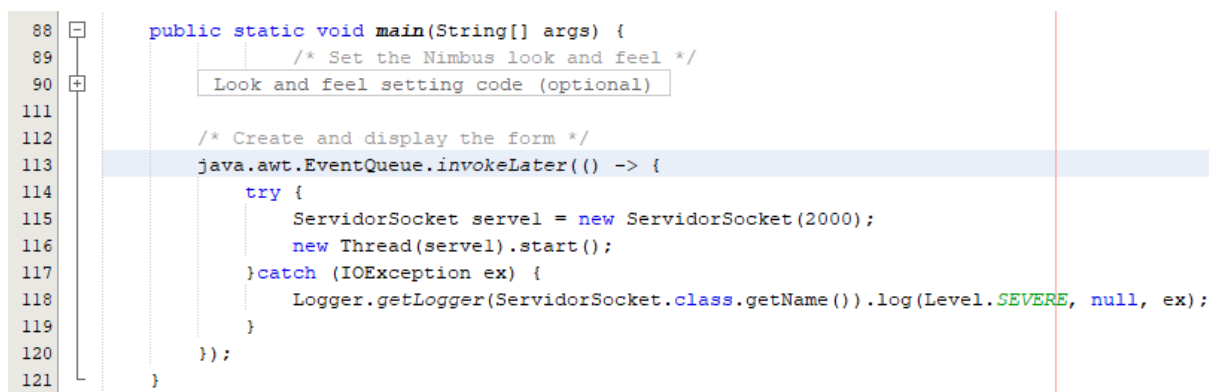
O diagrama de classe (Apêndice B) representa a organização da ferramenta criada para efetuar a leitura dos pacotes de dados transmitidos e armazená-los no banco de dados, neste estudo, implementados em supervisorio. Uma dessas classes criadas tem a responsabilidade de capturar os dados que chegam a servidor (*servidorsocket*). Essa classe foi criada com base no código contido no site Baeldung (BAELDUNG, 2017, Anexo A), e no site



Systembash (DEVE, 2008; Anexo B), foram feitas alterações com o suporte da documentação do Java (ORACLE, 2017) para suprir a necessidade do projeto.

A fim de monitorar de forma constante o recebimento das *strings* com os dados dos sensores, foi utilizado o recurso *implement* “*Runnable*” que permite iniciar uma *thread* da classe “*ServidorSocket*”. A implementação do método “*run*” é uma exigência do “*Runnable*”, pois é chamado quando se inicia a *thread*.

**Figura 20: Iniciando um thread da classe “*ServidorSocket*”**



```

88 public static void main(String[] args) {
89     /* Set the Nimbus look and feel */
90     Look and feel setting code (optional)
111
112     /* Create and display the form */
113     java.awt.EventQueue.invokeLater(() -> {
114         try {
115             ServidorSocket servel = new ServidorSocket(2000);
116             new Thread(servel).start();
117         } catch (IOException ex) {
118             Logger.getLogger(ServidorSocket.class.getName()).log(Level.SEVERE, null, ex);
119         }
120     });
121 }

```

**Imagem produzida pelo autor**

Quando a aplicação é iniciada, a função “*main*”, na Figura 20, cria o objeto do tipo “*ServidorSocket*” e inicia uma *thread* desta classe, onde por consequência executa a função “*run*” do “*ServidorSocket*”.

Para iniciar o servidor é necessário iniciar a classe “*DatagramaSocket*” que recebe um objeto do tipo “*IntSocketAddress*”, contendo o IP do servidor e porta que será utilizada para a comunicação (Figura 21). Quando entra dentro do *while* (linha 80 – Figura 21), os dois primeiros comandos deste são responsáveis por adquirir os pacotes enviados dos Arduinos (dotados de sensores em ambientes diferentemente monitorados). O comando “*new Datagramapacket*” cria um objeto que irá conter todos os paramentos do pacote, incluindo a mensagem. Para carregar esse objeto, o comado “*serve.receive*” deve ser executado, porém a execução do método “*run*” é interrompido até que a execução do método “*receive*” seja finalizada, e isso só ocorre quando um pacote é recebido pelo servidor.

Figura 21: Parte do código do método “run”

```

76 |         try {
77 |             InetAddress intSocketAddress = new InetAddress("10.0.0.100", this.port);
78 |             serve = new DatagramSocket(intSocketAddress);
79 |
80 |             while (TRUE) {
81 |                 this.reader = new DatagramPacket(this.bufReader, this.bufReader.length);
82 |                 serve.receive(reader);

```

Imagem produzida pelo autor

Assim que o pacote é recebido pelo objeto “serve” (linha 82 – Figura 21), dados são carregados no “reader”. Para facilitar a manipulação da mensagem, é convertido de *byte* para *string*, utilizando o método construtor da classe (Figura 22, linha 85).

Quando o pacote recebido pelo servidor é uma solicitação de um Arduino, é enviado uma resposta com a mensagem contendo “CONNECTED::”. Para enviar essa mensagem, primeiramente tem que ser gerado o pacote para a transmissão deste. A classe “DatagramPacket” é responsável por criar esse pacote. Para isso, é instanciado um objeto de mesma classe, sendo que um dos parâmetros recebido no método construtor é o objeto do tipo “InetSocketAddress”, que foi retirado do pacote que solicitou a conexão (Figura 22, linha 88) e para finalizar o pacote é necessário adicionar a mensagem ao mesmo, neste caso utilizado o método “setData” contido no objeto criado acima. Com o pacote construído, ele é enviado através do método “send” do objeto *serve*.

Figura 22: Parte do código do método “run”

```

85 |         String message = new String(this.reader.getData());
86 |         if(message.length() > 0){
87 |             if(message.startsWith("CONNECT::")){
88 |                 this.write = new DatagramPacket(bufWriter, bufWriter.length, this.reader.getSocketAddress());
89 |                 String t = "CONNECTED::";
90 |                 write.setData(t.getBytes());
91 |                 serve.send(write);
92 |             }else

```

Imagem produzida pelo autor

Caso a mensagem recebida seja contendo os dados, o mesmo é decomposto em parte, a fim de converte-los em número e adicionando-se ao objeto do tipo Dados (Apêndice C) para ser entregue ao método estático “persistir” da classe “DaoDados” que irá persistir no banco de dados, na Figura 23.

Figura 23: Parte do código do método “run”

```

65 | if(message.startsWith("MSG:")){
66 |     Double temp = Double.NaN;
67 |     String t = message.substring(message.lastIndexOf("TEMP:") + 6, message.indexOf("LUZ:"));
68 |     if(!"NaN".equals(t.replace(" ", ""))){
69 |         temp = Double.valueOf(message.substring(message.lastIndexOf("TEMP:") + 6, message.indexOf("LUZ:")));
70 |     }
71 |     Dados dados = new Dados(
72 |         message.substring(message.lastIndexOf("NAME:") + 6, message.indexOf("TEMP:")),
73 |         temp,
74 |         Double.valueOf(message.substring(message.lastIndexOf("LUZ:") + 5, message.indexOf("FUM:"))),
75 |         Double.valueOf(message.substring(message.lastIndexOf("FUM:") + 5, message.lastIndexOf("END")))
76 |     );
77 |     DaoDados.persistir(dados);
78 | }

```

Imagem produzida pelo autor

O banco de dados utilizado foi o Postgres, de modo que os dados são salvos em uma única tabela nomeada de “DADO” (Tabela 2), pois a complexidade é de simples geração de registro de dados (*log* dos sensores), não necessitando de múltiplas tabelas para tal fim.

Tabela 2: Print da Tabela do banco

	data_leitura timestamp without time zone	nome character varying (30)	temperatura numeric	luminoso numeric	fumaca numeric
1	2018-11-20 01:00:00.042	Sala Comum	29	3.2	0
2	2018-11-20 01:00:00.292	Cozinha	27.5	60.2	3
3	2018-11-20 01:00:00.828	Sala Comum	28.5	2.8	0
4	2018-11-20 01:00:01.079	Cozinha	28	60	3
5	2018-11-20 01:00:01.615	Sala Comum	28.5	1.8	0
6	2018-11-20 01:00:01.867	Cozinha	27.75	60	3
7	2018-11-20 01:00:02.403	Sala Comum	28.5	2.6	0
8	2018-11-20 01:00:02.654	Cozinha	28	60	3
9	2018-11-20 01:00:03.189	Sala Comum	28.75	3.2	0

Imagem produzida pelo autor

### 3.5. Dashboard

Com os dados contidos no banco de dados, pode-se visualizá-los no *dashboard* através de gráficos de linhas, instalado no servidor de aplicação Java (o supervisor, neste estudo). O resultado da construção do *dashboard* pode ser visualizado na Figura 24.

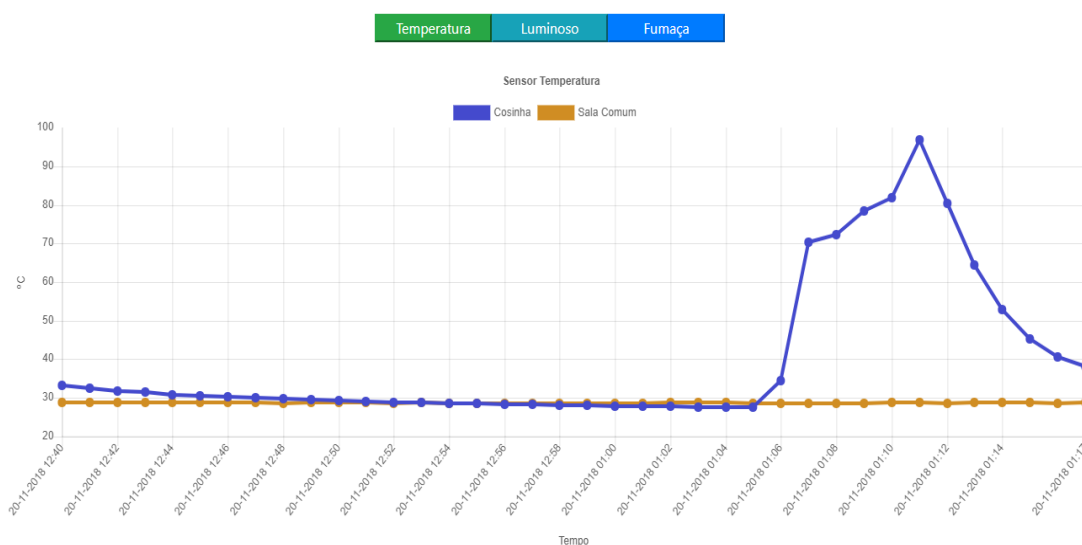
Figura 24: Página do *dashboard*.

Imagem produzida pelo autor

Para a construção dos gráficos foi utilizada a *tag* “*canvas*”, na Figura 25 (linhas 21 a 23). Assim, o gráfico de linhas na página é construído com base no *framework* de interface, chamado “*chartjs*” (Chart.js, 2013). Quando a página é carregada pelo navegador, o evento “*onload*” executa duas funções: “*buscarDados*” - que é um método da classe “*ChartBean*” (Apêndice D), e o “*selectGrafic*” - uma função criada em javascript localizado no final da página index (Figura 26 e Apêndice E).

Figura 25: HTML do arquivo index

```

13 <h:body onload="#{chartBean.buscarDados()} selectGrafic(0)">
14   <div class="container col-12">
15     <div class="row offset-4 col-4" style="margin-top: 5px">
16       <button class="btn-success col-4" onclick="selectGrafic(0)">Temperatura</button>
17       <button class="btn-info col-4" onclick="selectGrafic(1)">Luminoso</button>
18       <button class="btn-primary col-4" onclick="selectGrafic(2)">Fumaça</button>
19     </div>
20     <div class="row" style="padding-left: 3%;padding-right: 3%">
21       <canvas class="my-4 w-100" id="temperatura" width="800" height="320"></canvas>
22       <canvas class="my-4 w-100" id="luminoso" width="800" height="320"></canvas>
23       <canvas class="my-4 w-100" id="fumaca" width="800" height="320"></canvas>
24     </div>
25   </div>

```

Imagem produzida pelo autor

Conforme a Figura 25, a função “*selectGrafic*” (destacada na Figura 26), dependendo do valor de entrada, irá ocultar os gráficos não desejados, exibindo assim somente o gráfico-alvo no navegador, utilizando o comando condicional “*switch*” e alterando o atributo “*hidden*” para “*false*” das tags “*canvas*”.

**Figura 26: Código da função "selectGrafic" – para geração de gráfico de propriedade de sensor desejada na página *dashboard*.**

```

84 <script>
85     function selectGrafic(sele) {
86         switch(sele) {
87             case 0:
88                 document.getElementById("temperatura").hidden = false;
89                 document.getElementById("luminoso").hidden = true;
90                 document.getElementById("fumaca").hidden = true;
91             break;
92             case 1:
93                 document.getElementById("temperatura").hidden = true;
94                 document.getElementById("luminoso").hidden = false;
95                 document.getElementById("fumaca").hidden = true;
96             break;
97             case 2:
98                 document.getElementById("temperatura").hidden = true;
99                 document.getElementById("luminoso").hidden = true;
100                 document.getElementById("fumaca").hidden = false;
101             break;
102         }
103     }
104 </script>

```

Imagem produzida pelo autor

Quando o evento “onload” é chamado na Figura 25, o método “buscarDados” da classe “ChartBean” é invocado, na Figura 27, executando o método estático da classe “DaoDados” para buscar a lista de dados dos sensores no banco de dados.

**Figura 27: Código da classe "ChartBean" método "buscarDados"**

```

29 public void buscarDados () {
30     try {
31         this.listLocal = DaoDados.pesquisar();
32     } catch (SQLException ex) {
33         Logger.getLogger(ChartBean.class.getName()).log(Level.SEVERE, null, ex);
34     } catch (ClassNotFoundException ex) {
35         Logger.getLogger(ChartBean.class.getName()).log(Level.SEVERE, null, ex);
36     }
37 }

```

Imagem produzida pelo autor

Na classe “DaoDados” é executado o método “pesquisar”, na Figura 28, retornando uma lista de locais contidos no banco de dados (ou seja, locais registrados onde estejam instalados os conjuntos de sensores). A *string* de pesquisa faz um agrupamento por data e locais, com isso, se faz uma média dos valores de sensores com a função “AVG” do próprio banco de dados em consulta SQL (linhas 33 a 39 – Figura 28).

**Figura 28:** Parte do código de busca da classe "DaoDados" – consulta SQL formatada para obtenção de valor médio de propriedades de sensores conforme o intervalo de data e local desejados.

```

31 public static List pesquisar() throws SQLException, ClassNotFoundException{
32     String sql =
33         "select TO_CHAR(d.data_leitura, 'YYYY-MM-DD HH24:MI:00') as tempo, \n" +
34         "        d.nome, CAST(AVG(d.temperatura) as decimal(7,2)) as temperatura, \n" +
35         "        CAST(AVG(d.luminoso) as decimal(7,2)) as luminoso, \n" +
36         "        CAST(AVG(d.fumaca) as decimal(9,4)) as fumaca \n" +
37         "        from dado d\n" +
38         "        GROUP BY TO_CHAR(DATA_LEITURA, 'YYYY-MM-DD HH24:MI:00'), d.nome\n" +
39         "        ORDER BY tempo, nome";
40
41     PreparedStatement stm = Conexao.getConexao().prepareStatement(sql);
42
43     ResultSet rs = stm.executeQuery();

```

Imagem produzida pelo autor

Após carregar os dados na variável Lista, foi adicionado em uma variável estática local de nome “listTempo” as datas retiradas do banco, convertendo a *string* em um *calendar*, na Figura 29 – linha 65, utilizando um método “toDBCcalendar” estático da classe “FactoryCalendario” (Apêndice F). Essa lista estática pode ser buscada utilizando o método “getLestRead” que retornará as datas previamente salvas (Figura 30).

**Figura 29:** Parte do código de busca da classe "DaoDados" – filtragem de dados consultados para uso em variáveis úteis à geração de gráfico de propriedades na página *index*.

```

46 while(rs.next()){
47     Local l = new Local(rs.getString("nome"));
48     if(lista.contains(l)){
49         l = lista.get(lista.indexOf(l));
50     }else{
51         lista.add(l);
52     }
53     l.add(
54         new Dados(
55             rs.getString("tempo") , rs.getDouble("temperatura"),
56             rs.getDouble("luminoso"), rs.getDouble("fumaca")
57         )
58     );
59     if(!stringTime.contains(rs.getString("tempo"))){
60         stringTime.add(rs.getString("tempo"));
61     }
62 }
63 DaoDados.listTempo = new ArrayList<>();
64 for(String s: stringTime){
65     DaoDados.listTempo.add(FactoryCalendario.toDBCcalendar(s));
66 }
67
68 return lista;
69 }

```

Imagem produzida pelo autor

Figura 30: Código para retornar a lista de tempo

```

25     private static List<Calendar> listTempo = null;
26
27     public static List<Calendar> getLestRead() {
28         return DaoDados.listTempo;
29     }

```

Imagem produzida pelo autor

Quando terminar a execução na Classe “DaoDados”, retornará o objeto do tipo *List* contendo os dados que são salvos na variável “listaLocal” (Figura 27), sendo assim, continua a construção da página “index”. Existem três *scripts* que são responsáveis por criar os gráficos, conforme tratado no Apêndice E. Cada *script* contém uma variável que é buscada pelo ID da tag “canvas”.

Para construção das *strings* que contém os dados que serão exibidos no gráfico, foram criadas duas classes, a “*FactoryCharjsData*” (Apêndice G) e a “*FactoryDatasets*” (Apêndice H), que necessitam criar a *string* de acordo com padrão do *framework*. O processo final da construção da *string* pode ser visualizado na Figura 31.

Figura 31: Script pós processamento da página "index"

```

28     <script>
29         var ctx = document.getElementById("temperatura");
30         var myChart = new Chart(ctx, {
31             type: 'line',
32             data: {
33                 labels: ["20-11-2018 12:40", "20-11-2018 12:41", "20-11-2018 12:42", "20-11-2018 12:43", "20-11-2018 12:44",
34                     "20-11-2018 12:45", "20-11-2018 12:46", "20-11-2018 12:47", "20-11-2018 12:48", "20-11-2018 12:49", "20-11-2018 12:50",
35                     "20-11-2018 12:51", "20-11-2018 12:52", "20-11-2018 12:53", "20-11-2018 12:54", "20-11-2018 12:55", "20-11-2018 12:56",
36                     "20-11-2018 12:57", "20-11-2018 12:58", "20-11-2018 12:59", "20-11-2018 01:00", "20-11-2018 01:01", "20-11-2018 01:02",
37                     "20-11-2018 01:03", "20-11-2018 01:04", "20-11-2018 01:05", "20-11-2018 01:06", "20-11-2018 01:07", "20-11-2018 01:08",
38                     "20-11-2018 01:09", "20-11-2018 01:10", "20-11-2018 01:11", "20-11-2018 01:12", "20-11-2018 01:13", "20-11-2018 01:14",
39                     "20-11-2018 01:15", "20-11-2018 01:16", "20-11-2018 01:17"],
40                 datasets: [{
41                     label: 'Cosinha',
42                     fill: false,
43                     data: [33.27, 32.47, 31.85, 31.39, 30.79, 30.49, 30.22, 30.07, 29.8, 29.59, 29.31, 29.12, 28.87, 28.69, 28.65, 28.52,
44                         28.37, 28.25, 28.15, 27.98, 27.89, 27.86, 27.69, 27.59, 27.6, 27.46, 34.49, 70.23, 72.19, 78.35, 81.82, 96.83, 80.31,
45                         64.42, 52.93, 45.32, 40.55, 38.14],
46                     lineTension: 0,
47                     backgroundColor: '#A4DC94',
48                     borderColor: '#A4DC94',
49                     borderWidth: 4,
50                     pointBackgroundColor: '#A4DC94'
51                 }],
52                 label: 'Sala Comum',
53                 fill: false,
54                 data: [28.7, 28.7, 28.68, 28.67, 28.68, 28.68, 28.73, 28.66, 28.61, 28.68, 28.72, 28.72, 28.56, 28.69, 28.65, 28.64,
55                     28.52, 28.6, 28.64, 28.58, 28.63, 28.62, 28.68, 28.76, 28.67, 28.62, 28.64, 28.62, 28.6, 28.63, 28.67, 28.69, 28.65,
56                     28.75, 28.7, 28.69, 28.65, 28.66],
57                 lineTension: 0,
58                 backgroundColor: '#5A2B38',
59                 borderColor: '#5A2B38',
60                 borderWidth: 4,
61                 pointBackgroundColor: '#5A2B38'
62             }
63         });

```

Imagem produzida pelo autor

Para dar início a construção do *script*, da Figura 31, é chamado o método “*factoryCharjsDataTemp*” da classe “*ChartBean*” na página “*index*” como pode ser visto na Figura 32 - linha 37.

**Figura 32: JavaScript para criação do gráfico na página *index*.**

```

33 <script>
34     var ctx = document.getElementById("temperatura");
35     var myChart = new Chart(ctx, {
36         type: 'line',
37         data: #{chartBean.factoryCharjsDataTemp},
38         options: {
39             responsive: true,
40             title: {display: true, text: ' Sensor Temperatura'},
41             tooltips: {mode: 'index', intersect: false},
42             hover: {mode: 'nearest', intersect: true},
43             scales: {
44                 xAxes: [{display: true, scaleLabel: {display: true, labelString: 'Tempo'}}],
45                 yAxes: [{display: true, scaleLabel: {display: true, labelString: '°C'}}]
46             }
47         }
48     });
49 </script>

```

Imagem produzida pelo autor

Na Figura 33, o método “*getFactoryChartjsDataTemp*”, cria o objeto do tipo “*FactoryChartjsData*” que necessita passar no método construtor uma lista com as datas que serão exibidas no eixo x do gráfico, e com ajuda do “*for*” os dados são passados um a um para dentro do objeto. Quando executa o comando “*getCharjsData*” ele retornará a *string* com todos os dados necessários e no padrão correto para a exibição do gráfico.

**Figura 33: Código responsável de iniciar a construção do script**

```

39 public String getFactoryCharjsDataTemp() {
40     FactoryCharjsData charjsData;
41     charjsData = new FactoryCharjsData(DaoDados.getLestRead());
42     for(Local l : this.listLocal){
43         charjsData.addFactoryDatasets(l.getNome(), l.getListTemp());
44     }
45     return charjsData.getCharjsData();
46 }
47

```

Imagem produzida pelo autor

Após a estruturação da página *index* (*dashboard*), iniciaram-se os testes, conforme discutidos.



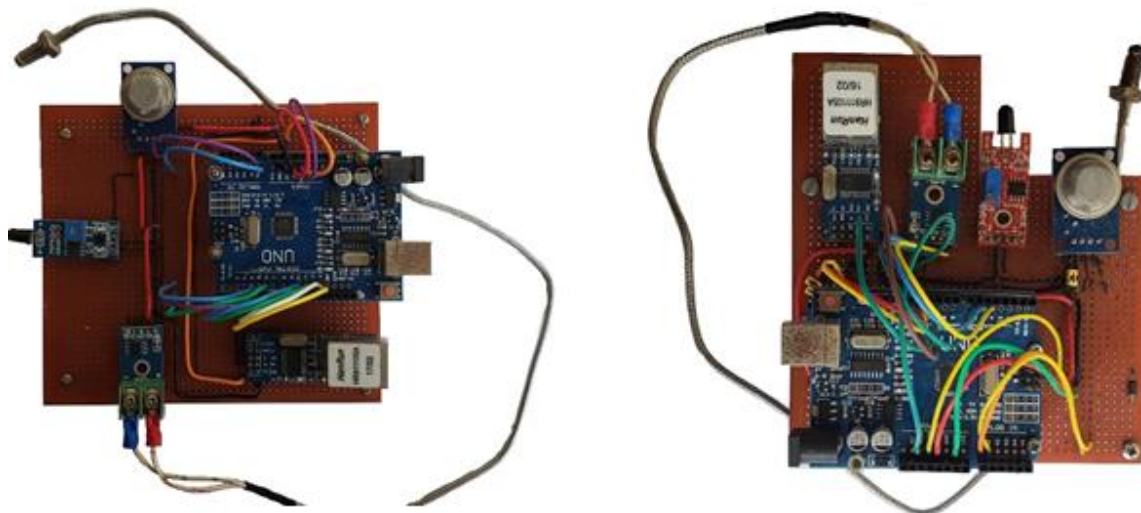
## 4. SIMULAÇÃO E TESTES

Após a construção do sistema, descrito no Capítulo 3, este capítulo almeja realizar simulação e testes em um ambiente controlado, sendo assim possível estimular a mudança de valores dos sensores de prevenção a incêndio de forma controlada e segura.

### 4.1. Preparação do ambiente de simulação

O cenário de simulação consiste em dois ambientes-alvo de monitoramento contra incêndio, sendo instalados módulos Arduino, cada qual com seu conjunto de sensores de temperatura, luminosidade e fumaça, além da placa *Ethernet* para comunicação com o computador supervisor. A Figura 34 mostra os referidos protótipos instalados nos respectivos ambientes (elaborada conforme diagrama eletrônico na Figura 6 – seção 3.2).

**Figura 34: Arduino localizado em dois ambientes distintos, como sala e cozinha de uma embarcação.**



**Imagem produzida pelo autor**

Uma rede local (Intranet) foi estabelecida para efetuar a comunicação entre os *hosts* (protótipos Arduino e o computador supervisor), utilizando um roteador convencional com a função DHCP desabilitada, pois os *hosts* foram configurados adequadamente em uma mesma rede local utilizando-se faixa de IP estático, conforme anteriormente tratado na seção

3.3.1. O diagrama de conexão pode ser visto na Figura 35, que exemplifica a comunicação entre os componentes.

**Figura 35: Diagrama em bloco das conexões entre os componentes da rede.**

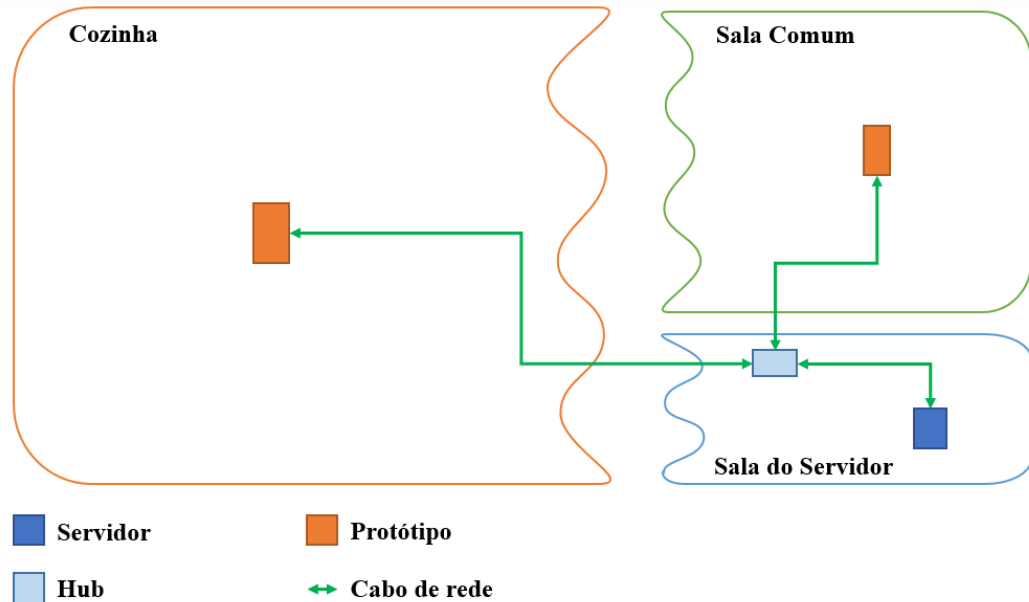


Imagem produzida pelo autor

## 4.2. Simulação de cenário real

A simulação foi iniciada pela certificação de comunicação entre os *hosts* utilizando uma comunicação Serial tipo cliente-servidor (observe que a ideia é garantir que há comunicação entre os sensores de campo monitorados pelos Arduinos e o computador supervisor, para uma simples conferência – Figura 36, mas os registros que alimentam o banco não usufruem desta comunicação, mas sim a comunicação *Ethernet* descrita no Capítulo 3).

**Figura 36: Leitura via porta serial dos Arduinos.**

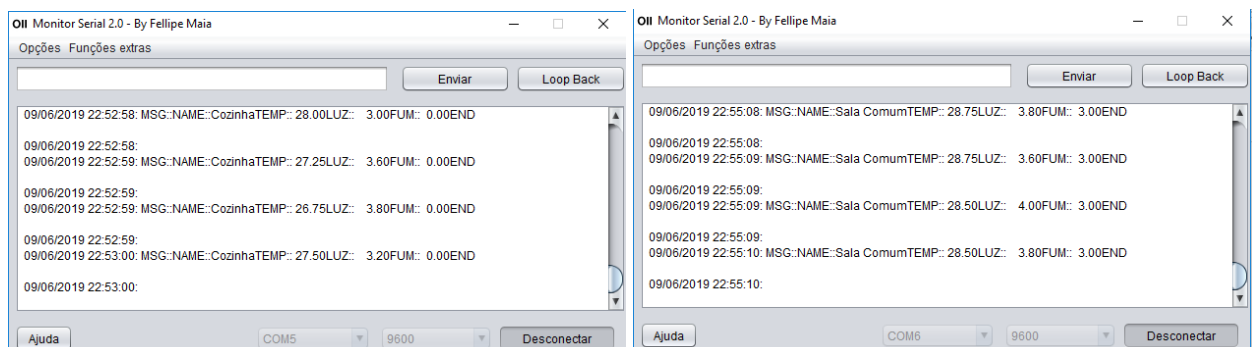
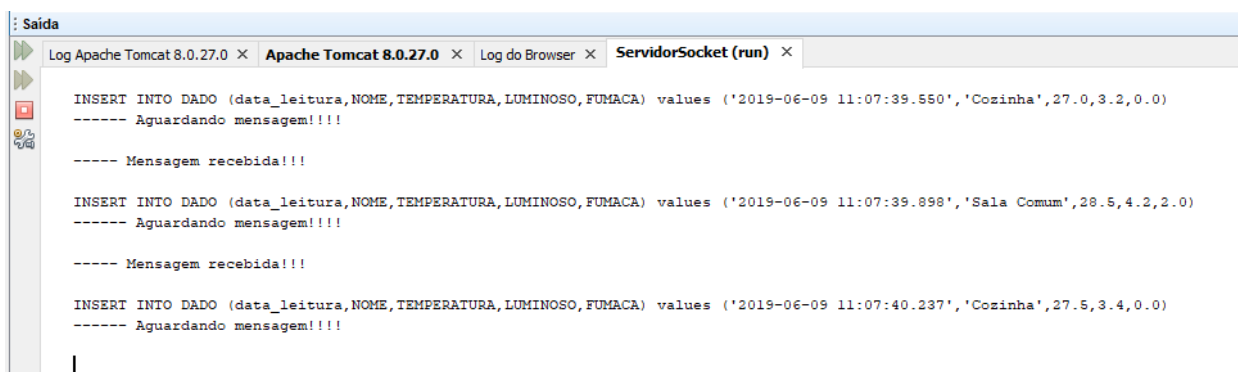


Imagem produzida pelo autor

A conexão *Ethernet* é verificada através do terminal no servidor. Durante a execução da aplicação “ServidorSocket” desenvolvida e tratada na seção 3.3.1. O resultado da comunicação pode ser visto na Figura 37. O terminal exibe os dados recebidos pelo *socket* e salvando-os no banco de dados.

**Figura 37: Terminal do Netbeans com os dados recebidos via *socket*.**



```

Saída
Log Apache Tomcat 8.0.27.0 x Apache Tomcat 8.0.27.0 x Log do Browser x ServidorSocket (run) x

INSERT INTO DADO (data_leitura,NOME,TEMPERATURA,LUMINOSO,FUMACA) values ('2019-06-09 11:07:39.550','Cozinha',27.0,3.2,0.0)
----- Aguardando mensagem!!!!

----- Mensagem recebida!!!

INSERT INTO DADO (data_leitura,NOME,TEMPERATURA,LUMINOSO,FUMACA) values ('2019-06-09 11:07:39.898','Sala Comum',28.5,4.2,2.0)
----- Aguardando mensagem!!!!

----- Mensagem recebida!!!

INSERT INTO DADO (data_leitura,NOME,TEMPERATURA,LUMINOSO,FUMACA) values ('2019-06-09 11:07:40.237','Cozinha',27.5,3.4,0.0)
----- Aguardando mensagem!!!!
  
```

**Imagem produzida pelo autor**

Durante a execução dos testes, o equipamento nomeado de “Cozinha” (Arduino com sensores instalados no ambiente cozinha) foi exposto a uma fonte de energia térmica a fim de simular uma situação real, as alterações das leituras dos sensores foram registradas e salvas no banco de dados. A Tabela 3 mostra os dados consultados no banco de dados via SGBD (Sistema de gerenciamento de Banco de Dados). Os resultados dos relatórios sobre os dados coletados no SGBD serão abordados na seção 4.3.

**Tabela 3: estruturada após a consulta em banco de dados no computador supervisorio, mostrando dados recém-capturados pelos sensores instalados nos ambientes distintos.**

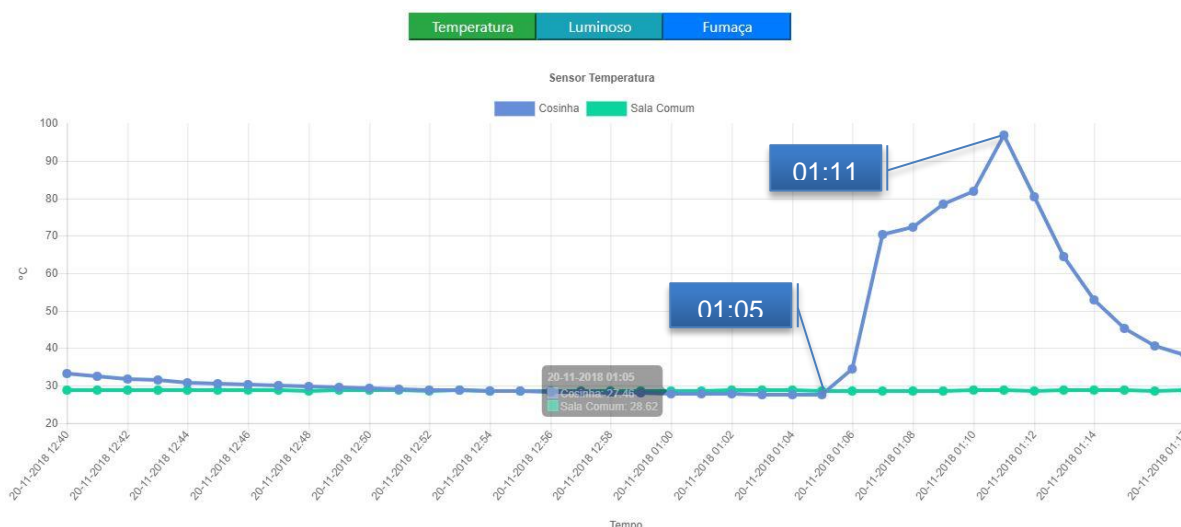
	data_leitura timestamp without time zone	nome character varying (30)	temperatura numeric	luminoso numeric	fumaca numeric
1	2018-11-20 01:00:00.042	Sala Comum	29	3.2	0
2	2018-11-20 01:00:00.292	Cozinha	27.5	60.2	3
3	2018-11-20 01:00:00.828	Sala Comum	28.5	2.8	0
4	2018-11-20 01:00:01.079	Cozinha	28	60	3
5	2018-11-20 01:00:01.615	Sala Comum	28.5	1.8	0
6	2018-11-20 01:00:01.867	Cozinha	27.75	60	3
7	2018-11-20 01:00:02.403	Sala Comum	28.5	2.6	0
8	2018-11-20 01:00:02.654	Cozinha	28	60	3
9	2018-11-20 01:00:03.189	Sala Comum	28.75	3.2	0

**Imagem produzida pelo autor**

### 4.3. Análise dos gráficos via página *Dashboard*

Com os dados de sensores armazenados no banco de dados, a página *dashboard* permite verificação destes no computador supervisor. Para cada opção de propriedade disponível (temperatura, luminosidade e fumaça), será possível acessar um gráfico de registros para os ambientes aplicados (Figura 37 a Figura 39). No exemplo de teste, a variação da temperatura nos dois ambientes monitorados são constantes até 01:05, como pode ser percebido na Figura 38. Após este horário, ocorreu uma alteração dos valores de temperatura, tendo o valor mais alto registrado as 01:11 (~97° C), porém ainda não considerado um valor para disparo de alarme (segundo a norma internacional FSS, 2010 – discutido no Capítulo 2), considerado o local cozinha ser comum de ocorrer elevação da temperatura, o que não isenta de investigação o aumento repentino da temperatura registrada, supondo que no referido horário não haja a utilização de fogões e fornos, quando a cozinha deveria apresentar os parâmetros de um ambiente comum.

**Figura 38: Gráfico de Temperatura**



**Imagem produzida pelo autor**

Além dos valores de temperatura, há o valor de luminosidade que é útil para a detecção de incêndio (sensor de apoio ao julgamento da ocorrência de incêndio, pois conforme a norma marítima MSC.98(73) (FSS, 2010), este não pode ser o único valor a ser considerado para acionar o alarme). Além do aumento brusco da temperatura, observa-se ter

ocorrido uma elevação também brusca dos valores do sensor de luminosidade (Figura 39) em relação à luminosidade média do ambiente por determinação a partir do histórico do local.

**Figura 39: Gráfico de Luminosidade**



Imagem produzida pelo autor

No gráfico de concentração da propriedade fumaça, na Figura 40, pode-se considerar a ocorrência de um incêndio na cozinha, pois o nível de concentração de fumaça encontrado foi ~2000 ppm as 01:11, sendo que em um ambiente sem fumaça teria valor aproximadamente de 0 ppm. Já o pico negativo visualizado as 01:09, não foi possível determinar a causa, um provável ruído à detecção da concentração abrupta durante os testes com fumaça.

**Figura 40: Gráfico de fumaça**

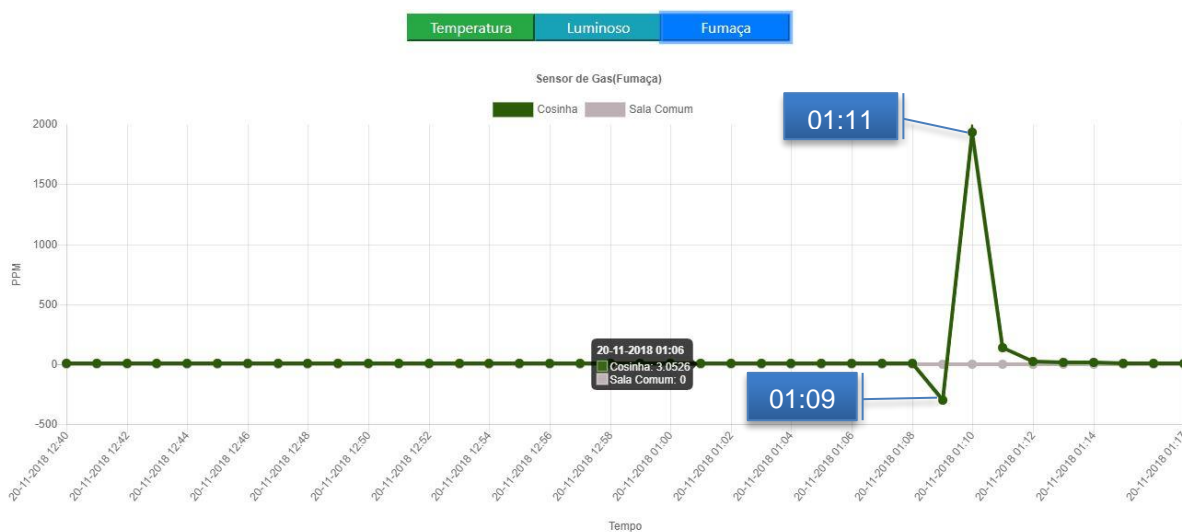


Imagem produzida pelo autor

Os testes realizados provam a detecção dos parâmetros-alvo para julgamento da ocorrência de incêndio. No entanto, este estudo limita-se ao registro e geração de relatórios das propriedades de interesse entre sensores de campos (com Arduinos) e o computador supervisor, não cabendo estipular regras para geração de alarmes, mas sim fornecer insumos para tal.

## 5. CONSIDERAÇÕES FINAIS

O presente trabalho sintetizou um estudo sobre a criticidade, parâmetros e sensores úteis à detecção e prevenção a incêndios em ambiente marítimos. Assim, o planejamento de sistemas que facilitem a detecção, monitoramento e alarmes aumentam a segurança de embarcações e trabalhadores envolvidos, especialmente para o contexto de produção de petróleo *offshore*, o que justifica a importância deste na área, passível de extensão a outros ambientes comumente sujeitos a riscos de incêndios por produtos químicos inflamáveis e sistemas com alta pressurização.

A utilização do Arduino para elaboração de um protótipo de sensores de campo, úteis à detecção de incêndio, e comunicação com o sistema supervisor, mostrou-se eficaz nas simulações controladas no contexto deste estudo, para possíveis locais de convivência em qualquer unidade marítima, como salas, dormitórios, cozinha, corredores, entre outros. Os sensores utilizados neste projeto foram definidos com base na documentação da norma internacional MSC.98(73) (FSS, 2010) e suas atualizações, para as propriedades de interesse do ambiente: temperatura, luminosidade e concentração de fumaça. Esses dados foram lidos pela aplicação desenvolvida neste projeto, utilizando a linguagem Java e estabelecimento de comunicação *socket* via UDP entre sensores de campo com Arduinos e o computador supervisor. No supervisor, os dados registrados em banco de dados foram apresentados por aplicação *dashboard* desenvolvido em linguagem Java e *frameworks* JSF2.2, Bootstrap e ChartJs.

Como propostas para trabalhos futuros, recomendam-se:

- Melhorar o sistema supervisor pela implementação de regras de alertas e alarmes que permitam a tomada de decisão assistida e/ou automática;

- Melhorar a programação do protótipo de sensores com Arduino, permitindo obtenção de resposta do computador supervisor para o refinamento de propriedades em aquisição no campo;
- Aplicar o sistema proposto neste estudo em um ambiente real para avaliação.

## 6. REFERÊNCIAS BIBLIOGRÁFICAS

ABNT, ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 17240: Sistemas de detecção e alarme de incêndio - Projeto, instalação, comissionamento e manutenção de sistemas de detecção e alarme de incêndio - Requisitos**. Rio de Janeiro: [s.i], 2010. 54 p. Disponível em: <<http://www.segmafire.com.br/wp-content/uploads/sites/179520/2017/06/NBR-17240-2010-Substituindo-NBR-9441-Alarme.pdf>>. Acesso em: 09 jul. 2018.

ALMEIDA, A. L. d. **Gerenciamento de Alarmes em Plataformas Marítimas de Produção de Hidrocarbonetos: Metodologia e Estudo de Caso**. 68 f. Dissertação (Mestrado) - Curso de Pos-graduação em Ciência e Engenharia de Produção, Universidade Federal do Rio Grande do Norte, Natal, 2010. Disponível em: <<http://repositorio.ufrn.br:8080/jspui/handle/123456789/12941>>. Acesso em: 17 maio 2018.

ARAÚJO, E. V. **Gerenciamento de Alarmes em plantas Industriais: Conceitos Normas e Estudo de Caso Em UM Forno de Reaquecimento de Blocos**. 2010. 100 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 2010. Disponível em: <<http://www.bibliotecadigital.ufmg.br/dspace/handle/1843/BUBD-8GQFV7>>. Acesso em: 25 jun. 2018.

ARDUINO.CC, **FREQUENTLY Asked Questions**. Disponível em: <<https://www.arduino.cc/en/Main/FAQ#toc3>>. Acesso em: 26 jun 2018.

BAELDUNG. **A Guide To UDP In Java**. 2017. Disponível em: <<https://www.baeldung.com/udp-in-java>>. Acesso em: 08 set. 2018.

BRASIL, MINISTÉRIO DO TRABALHO E EMPREGO. **NR 23 - Proteção Contra Incêndios. Portaria nº 3.214, de 8 de junho de 1978**. Brasília: Ministério do Trabalho e Emprego, Disponível em: <<http://www.camara.gov.br/sileg/integras/839945.pdf>>. Acesso em: 08 jul. 2018.

Chart.js, **Gráficos JavaScript simples e flexíveis para designers e desenvolvedores**. 17 Março 2013. Disponível em: <<https://www.chartjs.org/>>. Acesso em: 10 nov. 2018

COMER, Douglas. **Interligação de Redes com TCP/IP: Princípios, Protocolos e Arquitetura**. 6. ed. Rio de Janeiro: Elsevier, 2015. Disponível em: <[https://books.google.com.br/books?id=F1\\_jBwAAQBAJ&printsec=frontcover&hl=pt-BR&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](https://books.google.com.br/books?id=F1_jBwAAQBAJ&printsec=frontcover&hl=pt-BR&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)>. Acesso em: 14 maio 2019.



CONTECH, **Proteção contra incêndios: Cuidado nunca é demais. Tudo o que precisamos de segurança contra incêndio e como fornecer produtos, sistemas e soluções de segurança contra incêndio.** 2016. Disponível em: <<http://contechind.com.br/blog/2016/09/13/protecao-contraincendios-cuidado-nunca-e-demais/>> Acesso em 26 mar. 2018

CONTECH. **As plataformas petroleiras e suas necessidades de automação de sistemas contra incêndio.** 2017. Disponível em: <<http://contechind.com.br/blog/2017/05/09/as-plataformas-petroleiras-e-suas-necessidades-de-automacao-de-sistemas-contraincendio/>>. Acesso em: 03 jul. 2018.

DEVE. **A Simple Java UDP Server and UDP Client.** 2008. Disponível em: <<https://systembash.com/a-simple-java-udp-server-and-udp-client/>>. Acesso em: 10 out. 2018.

DUARTE, O. C. M. B. **IP Security.** 2003. Disponível em: <[https://www.gta.ufrj.br/grad/03\\_1/ip-security/paginas/main.html](https://www.gta.ufrj.br/grad/03_1/ip-security/paginas/main.html)>. Acesso em: 27 jun. 2018.

GUAIANO, O. **Acidentes em Plataformas de petróleo. SOBRASA (Simpósio Brasileiro de Emergências Aquáticas)**, Rio Janeiro: 2014. 13 slides, color. Disponível em: <<http://www.sobrasa.org/simposio-brasileiro-de-emergencias-aquaticas-19-de-agosto-de-2014-rio-de-janeiro/>>. Acesso em: 28 jun. 2018.

JURIZATO, L. A.; PEREIRA, P. S. R. **SISTEMAS SUPERVISÓRIOS. Network Technologies: Nova Odessa. Nova Odessa**, p. 105-114. maio 2002-2003. Disponível em: <<http://centralmat.com.br/Artigos/Mais/sistemasSupervisorios.pdf>>. Acesso em: 25 jun. 2018.

JUNQUEIRA, G. S. **Análise das possibilidades de utilização de sistemas supervisórios no planejamento e controle de produção.** 2003. 131 f. Dissertação (Mestrado) - Curso de Engenharia de Produção, Escola de Engenharia de São Carlos, São Carlos, 2003. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/18/18140/tde-01082005-224411/en.php>>. Acesso em: 03 jul. 2018.

KOMIDO, D. T. M; REIS, D. H.; GALLO, W. N. **MONITORAMENTO DE ATIVOS DE REDE COM USO DE GSM E SMS SOBRE PLATAFORMA ARDUINO.** RE3C-Revista Eletrônica Científica de Ciência da Computação, v. 11, n. 1, 2016. Disponível em <<http://revistas.unifenas.br/index.php/RE3C/article/view/164>> Acesso em: 03 jul. 2018.

MULTILOGICA-SHOP. **Arduino Guia Iniciante 2.0.** Santo André, 2018 Disponível em: <[https://multilogica-shop.com/download\\_guia\\_arduino](https://multilogica-shop.com/download_guia_arduino)>. Acesso em: 28 jun. 2018.

ORACLE. Java™ Platform, **Standard Edition 8:: API Specification**. 2017. Disponível em: <<https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html>>. Acesso em: 15 out. 2018.

PALMIERE, S. E. **CLP versus Microcontrolador**. 2016. Disponível em: <<https://www.embarcados.com.br/clp-versus-microcontrolador/>>. Acesso em: 09 jul. 2018.

PINTO, A. C. V. **Desenvolvimento de Sistema de Monitoramento de Gás LP Com Alarme por SMS**. 2016. 52 f. Monografia (Especialização) - Curso de Engenharia de Controle e Automação, Universidade Federal de Ouro Preto, Ouro Preto, 2016. Disponível em: <<http://www.monografias.ufop.br/handle/35400000/275>>. Acesso em: 27 jun. 2018.

FSS, **International Code For Fire Safety Systems - FSS**. 2010. Disponível em: <[https://www.ccaimo.mar.mil.br/sites/default/files/fss\\_consolidado\\_com\\_emd\\_jul2010\\_2.pdf](https://www.ccaimo.mar.mil.br/sites/default/files/fss_consolidado_com_emd_jul2010_2.pdf)>. Acesso em: 17 maio 2018.

SOLAS, **CONVENÇÃO INTERNACIONAL PARA SALVAGUARDA DA VIDA HUMANA NO MAR**. SOLAS 1974/1988: Artigos da Convenção Internacional para Salvaguarda da Vida Humana no Mar, 1974. [S.i.: S.n.], 2014. 506 p. Disponível em: <<https://www.ccaimo.mar.mil.br/solas>>. Acesso em: 17 maio 2018.

STEVAN JUNIOR, S. L.; SILVA, R. A. **Automação e Instrumentação Industrial com Arduino: Teoria e Projetos**. São Paulo: Érica, 2015.

## 7. ANEXOS

### ANEXO A – CÓDIGO RETIRADO DO SITE BAEILDUNG

```
1  public class EchoServer extends Thread {
2
3      private DatagramSocket socket;
4      private boolean running;
5      private byte[] buf = new byte[256];
6
7      public EchoServer() {
8          socket = new DatagramSocket(4445);
9      }
10
11     public void run() {
12         running = true;
13
14         while (running) {
15             DatagramPacket packet
16                 = new DatagramPacket(buf, buf.length);
17             socket.receive(packet);
18
19             InetAddress address = packet.getAddress();
20             int port = packet.getPort();
21             packet = new DatagramPacket(buf, buf.length, address, port);
22             String received
23                 = new String(packet.getData(), 0, packet.getLength());
24
25             if (received.equals("end")) {
26                 running = false;
27                 continue;
28             }
29             socket.send(packet);
30         }
31         socket.close();
32     }
33 }
```

## ANEXO B – CÓDIGO RETIRADO DO SITE SYSTEMBASH

### UDPServer.java:

```
import java.io.*;
import java.net.*;

class UDPServer
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true)
        {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String( receivePacket.getData());
            System.out.println("RECEIVED: " + sentence);
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

## 8. APÊNDICE

### APÊNDICE A – CÓDIGO DO ARDUINO

```
#include <EtherCard.h>
#include <IPAddress.h>
#include <MQ2.h>
#include "max6675.h"

void variarNome();

#define load_Res 10
#define sensorLuz A0
#define sensorFunmaca A1
#define SS 10

//MAX6675(int8_t SCLK, int8_t CS, int8_t MISO);
MAX6675 thermocouple(6, 5, 4);

MQ2 mq2(sensorFunmaca);

// ethernet mac address - must be unique on your network
static byte mymac[] = { 0x74, 0x69, 0x69, 0x30, 0x2D, 0x33 };
// ethernet interface ip address
static byte myip[] = { 10, 0, 0, 101 };
// gateway ip address
static byte gwip[] = { 10, 0, 0, 1 };
// mask
static byte mask[] = { 255, 255, 255, 0 };
//dns
static byte dns[] = { 10, 0, 0, 1 };

byte Ethernet::buffer[256]; // tcp/ip send and receive buffer

static byte ipServe[4] = { 10, 0, 0, 100 };
char nome[20];
char mensage[100];
char auxChar[10];
unsigned int portServe = 2000;
static uint32_t timer;
double PPMFunmaca;
char dados;

bool recebido = false;
//bool udpSerialPrint(word port, byte ip[4], const char *data, word len)
{
void udpSerialPrint(word port, byte ip[4], char *data, word len) {
    IPAddress src(ip[0], ip[1], ip[2], ip[3]);
    recebido = true;
}
```

```

}

void setup() {
  strcpy(nome, "Cozinha");
  //===== Serial

  Serial.begin(9600);
  Serial.println("Serial iniciado");

  //===== Placa de Rede

  Serial.print("Placa de rede iniciando...");
  if (ether.begin(sizeof Ethernet::buffer, mymac, SS) == 0){
    Serial.println("Failed to access Ethernet controller");
  }else{
    Serial.println("Placa de rede iniciado");
  }

  //===== Configurando IP

  Serial.println("ip Statico definido");
  //ether.staticSetup(myip, gwip,dns,mask);
  ether.staticSetup(myip,gwip,dns,mask);

  ether.copyIp(ether.hisip,ipServe);

  Serial.println("Ip's");
  ether.printIp("IP: ", ether.myip);
  ether.printIp("GW: ", ether.gwip);
  ether.printIp("DNS: ", ether.dnsip);
  ether.printIp("SRV: ", ether.hisip);
  ether.printIp("MAC: ", ether.mymac);

  ether.udpServerListenOnPort(&udpSerialPrint, portServe);
  strcpy(mensaje, "CONECT::");
  int TRUE = 1;
  while(TRUE){
    ether.sendUdp(mensaje, sizeof(mensaje), portServe, ipServe,
portServe);
    timer = millis() + 5000;
    Serial.println("Aguardando resposta!");
    while(millis() < timer){
      ether.packetLoop(ether.packetReceive());
      if(recebido){
        Serial.println("ConexÃ£o aceita!");
        TRUE = 0;
        recebido = false;
        break;
      }
    }
  }
}

//===== Luz

```

```

pinMode(sensorLuz, INPUT);
Serial.println("Luz"+analogRead(sensorLuz));

//===== Fumaça

Serial.println("Calibrando.....");
mq2.begin();
//PPMFunmaca = resistencia(50,500)/air_factor;
//Serial.print("PPM: ");
//Serial.println(PPMFunmaca);
Serial.println("Calibrado.");

Serial.println("\nFim das configurações\n");
}

float cont_leitura = 0.0;
float media_temp = 0.0, media_luz = 0.0, media_fum = 0.0;

void loop() {
    if (millis() > timer) {
        timer = millis() + 300;
        strcpy(message,"MSG::NAME::");
        strcat(message,nome);
        strcat(message,"TEMP::");
        dtostrf((media_temp/cont_leitura),6,2,auxChar);
        strcat(message,auxChar);
        strcat(message,"LUZ::");
        dtostrf((media_luz/cont_leitura),8,2,auxChar);
        strcat(message,auxChar);
        strcat(message,"FUM::");
        dtostrf((media_fum/cont_leitura),6,2,auxChar);
        strcat(message,auxChar);
        strcat(message,"END\0");
        Serial.println(message);

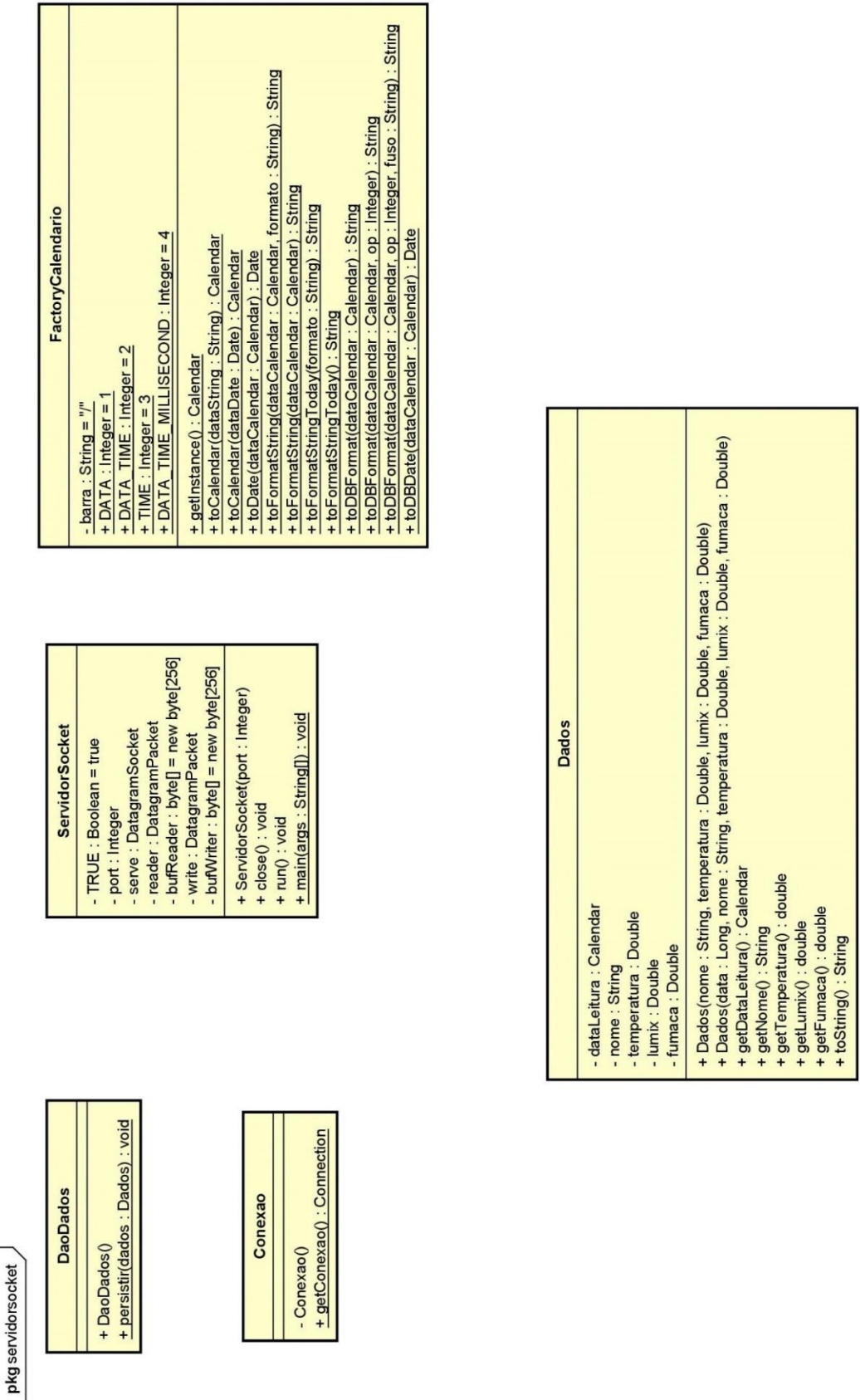
        //static void sendUdp (char *data,uint8_t len,uint16_t sport,
uint8_t *dip, uint16_t dport);
        ether.sendUdp(message, sizeof(message), portServe, ipServe,
portServe );
        for(int i =0;i<100;i++){
            message[i] = '\0';
        }
        cont_leitura = 0.0;
        media_temp = 0.0;
        media_luz = 0.0;
        media_fum = 0.0;

    }else{
        cont_leitura++;
        media_temp += thermocouple.readCelsius();
        media_luz += Luz(5,80);
        media_fum += mq2.readSmoke();
    }
}

double Luz(int num_leituras, int tempo){
    int i;
    double res=0;
    for (i=0;i<num_leituras;i++)
    {
        int leitura=analogRead(sensorLuz);
        res+=(double) (1023-leitura);
        delay(tempo);
    }
    res/=num_leituras;
    return res;
}

```

APÊNDICE B – DIAGRAMA DE CLASSES DO SERVIDOR SOCKET





## APÊNDICE C – CLASSE DADOS DO SERVIDOR SOCKET

```

package servidorsocket.model;

import java.io.Serializable;
import java.util.Calendar;

/**
 *
 * @author fmaia
 */
public class Dados implements Serializable {
    private final Calendar dataLeitura;
    private final String nome;
    private final Double temperatura;
    private final Double lumix;
    private final Double fumaca;

    public Dados(String nome, Double temperatura, Double lumix, Double
fumaca) {
        this.dataLeitura = Calendar.getInstance();
        this.nome = nome;
        this.temperatura = temperatura;
        this.lumix = lumix;
        this.fumaca = fumaca;
    }

    public Dados(Long data, String nome, Double temperatura, Double lumix,
Double fumaca) {
        this.dataLeitura = Calendar.getInstance();
        this.dataLeitura.setTimeInMillis(data);
        this.nome = nome;
        this.temperatura = temperatura;
        this.lumix = lumix;
        this.fumaca = fumaca;
    }

    public Calendar getDataLeitura() {
        return (Calendar) dataLeitura.clone();
    }

    public String getNome() {
        return nome;
    }

    public double getTemperatura() {
        return temperatura;
    }

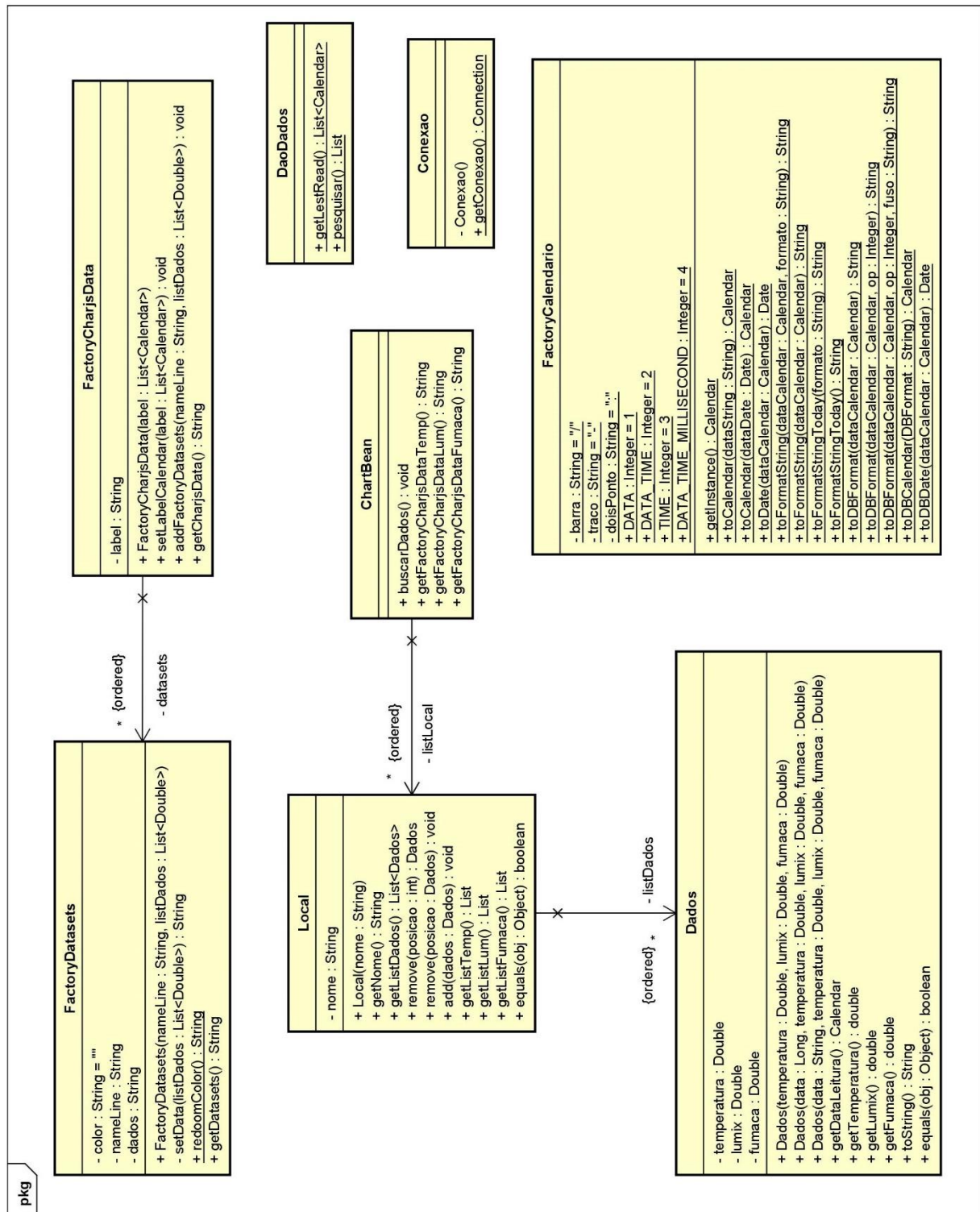
    public double getLumix() {
        return lumix;
    }

    public double getFumaca() {
        return fumaca;
    }

    @Override
    public String toString() {
        return "Dados{" + "nome=" + nome + ", temperatura=" + temperatura
+ ", lumix=" + lumix + ", fuma\u00e7a=" + fumaca + '}';
    }
}

```

# APÊNDICE D – DIAGRAMA DE CLASSES DO SERVIDOR SOCKET



## APÊNDICE E – PAGINA INDEX

```

1  <?xml version='1.0' encoding='UTF-8' ?>
2  <!DOCTYPE html
3  | PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
5  <h:head>
6      <title>Facelet Title</title>
7      <meta charset="utf-8">
8      </meta>
9      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
10     </meta>
11     <meta name="description" content="">
12     </meta>
13     <meta name="author" content="">
14     </meta>
15     <link href="./css/bootstrap.min.css" rel="stylesheet">
16     </link>
17 </h:head>
18 <h:body onload="#{chartBean.buscarDados()} selectGrafic(0)">
19     <div class="container col-12">
20         <div class="row offset-4 col-4" style="margin-top: 5px">
21             <button class="btn-success col-4" onclick="selectGrafic(0)">Temperatura</button>
22             <button class="btn-info col-4" onclick="selectGrafic(1)">Luminoso</button>
23             <button class="btn-primary col-4" onclick="selectGrafic(2)">Fumaça</button>
24         </div>
25         <div class="row" style="padding-left: 3%;padding-right: 3%">
26             <canvas class="my-4 w-100" id="temperatura" width="800" height="320"></canvas>
27             <canvas class="my-4 w-100" id="luminoso" width="800" height="320"></canvas>
28             <canvas class="my-4 w-100" id="fumaca" width="800" height="320"></canvas>
29         </div>
30     </div>
31     <!-- script's -->
32     <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
33         integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5SmXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous">
34     </script>
35     <script src="./js/bootstrap.min.js"></script>
36     <!-- Graphs -->
37     <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.1/Chart.min.js"></script>
38     <script>
39         var ctx = document.getElementById("temperatura");
40         var myChart = new Chart(ctx, {
41             type: 'line',
42             data: #{ chartBean.factoryCharjsDataTemp },
43             options: {
44                 responsive: true,
45                 title: { display: true, text: ' Sensor Temperatura' },
46                 tooltips: { mode: 'index', intersect: false },
47                 hover: { mode: 'nearest', intersect: true },
48                 scales: {
49                     xAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Tempo' } }],
50                     yAxes: [{ display: true, scaleLabel: { display: true, labelString: '°C' } } ]
51                 }
52             }
53         });
54     </script>
55     <script>
56         var ctx = document.getElementById("luminoso");
57         var myChart = new Chart(ctx, {
58             type: 'line',
59             data: #{ chartBean.factoryCharjsDataLum },
60             options: {
61                 responsive: true,
62                 title: { display: true, text: 'Sensor Luminoso' },
63                 tooltips: { mode: 'index', intersect: false },
64                 hover: { mode: 'nearest', intersect: true },
65                 scales: {
66                     xAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Tempo' } }],
67                     yAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Lumix' } } ]
68                 }
69             }
70         });
71     </script>

```

```

55     <script>
56         var ctx = document.getElementById("luminoso");
57         var myChart = new Chart(ctx, {
58             type: 'line',
59             data: #{ chartBean.factoryCharjsDataLum },
60             options: {
61                 responsive: true,
62                 title: { display: true, text: 'Sensor Luminoso' },
63                 tooltips: { mode: 'index', intersect: false },
64                 hover: { mode: 'nearest', intersect: true },
65                 scales: {
66                     xAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Tempo' } }],
67                     yAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Lumix' } }]
68                 }
69             }
70         });
71     </script>
72     <script>
73         var ctx = document.getElementById("fumaca");
74         var myChart = new Chart(ctx, {
75             type: 'line',
76             data: #{ chartBean.factoryCharjsDataFumaca },
77             options: {
78                 responsive: true,
79                 title: { display: true, text: 'Sensor de Gas(Fumaça)' },
80                 tooltips: { mode: 'index', intersect: false },
81                 hover: { mode: 'nearest', intersect: true },
82                 scales: {
83                     xAxes: [{ display: true, scaleLabel: { display: true, labelString: 'Tempo' } }],
84                     yAxes: [{ display: true, scaleLabel: { display: true, labelString: 'PPM' } }]
85                 }
86             }
87         });
88     </script>
89     <script>
90         function selectGrafic(sele) {
91             switch (sele) {
92                 case 0:
93                     document.getElementById("temperatura").hidden = false;
94                     document.getElementById("luminoso").hidden = true;
95                     document.getElementById("fumaca").hidden = true;
96                     break;
97                 case 1:
98                     document.getElementById("temperatura").hidden = true;
99                     document.getElementById("luminoso").hidden = false;
100                     document.getElementById("fumaca").hidden = true;
101                     break;
102                 case 2:
103                     document.getElementById("temperatura").hidden = true;
104                     document.getElementById("luminoso").hidden = true;
105                     document.getElementById("fumaca").hidden = false;
106                     break;
107             }
108         }
109     </script>
110 </h:body>
111
112 </html>

```

## APÊNDICE F – CLASSE FACTORYCALENDARIO

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package servidorsocket.factory;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

/**
 *
 * @author FMaia
 */
public class FactoryCalendario {
    private static final String barra = "/";
    private static final String traco = "-";
    private static final String doisPonto = ":";

    public static final Integer DATA = 1;
    public static final Integer DATA_TIME = 2;
    public static final Integer TIME = 3;
    public static final Integer DATA_TIME_MILLISECOND = 4; //millisecond

    public static Calendar getInstance(){
        Calendar hoje = Calendar.getInstance();
        return hoje;
    }

    public static Calendar toCalendar(String dataString){
        String dataVetor[] = dataString.split(barra);
        Calendar dataCalendar = Calendar.getInstance();
        dataCalendar.set(Integer.parseInt(dataVetor[2]),
            Integer.parseInt(dataVetor[1])-
1, Integer.parseInt(dataVetor[0]));
        return dataCalendar;
    }

    public static Calendar toCalendar(Date dataDate){
        if(dataDate == null)
            return null;
        Calendar dataCalendar = Calendar.getInstance();
        dataCalendar.setTime(dataDate);
        return dataCalendar;
    }

    public static Date toDate(Calendar dataCalendar){
        if(dataCalendar == null)
            return null;
        /*Date dataDate = new Date();
        dataDate.setTime(dataCalendar.getTimeInMillis());
        return dataDate;*/
        return dataCalendar.getTime();
    }

    public static String toFormatString(Calendar dataCalendar, String formato){
        if(dataCalendar == null)
            return "";
        SimpleDateFormat dataFormat = new SimpleDateFormat(formato);
        return dataFormat.format(dataCalendar.getTime());
    }

    public static String toFormatString(Calendar dataCalendar){

```

## APÊNDICE G – CLASSE FACTORYCHARJSDATA

```

package servidorsocket.factory;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

/**
 *
 * @author fmaia
 */
public class FactoryCharjsData {

    private List<FactoryDatasets> datasets = new ArrayList<>();
    private String label;

    public FactoryCharjsData(List<Calendar> label) {
        this.setLabelCalendar(label);
    }

    public void setLabelCalendar(List<Calendar> label){
        String Label = "";
        for(Calendar c: label){
            Label = Label +"\n"+
                FactoryCalendario.toFormatString(c, "dd-MM-yyyy
hh:mm")+
                "\n, ";
        }
        this.label = Label.substring(0, Label.length()-2);
    }

    public void addFactoryDatasets(String nameLine,List<Double>
listDados){
        this.datasets.add(new FactoryDatasets(nameLine, listDados));
    }

    public String getCharjsData(){
        String data =
            "{\n" +
            "    labels: [\"+this.label+\"],\n" +
            "    datasets: [\";
            for(FactoryDatasets d: this.datasets){
                data += d.getDatasets();
                data += ", ";
            }
            data = data.substring(0, data.length()-1);
            data += "]\n}";
        return data;
    }
}

```

## APÊNDICE H – CLASSE FACTORYDATASETS

```

package servidorsocket.factory;

import java.util.List;
import java.util.Random;

/**
 *
 * @author fmaia
 */
public class FactoryDatasets {

    private String color = "";
    private String nameLine;
    private String dados;

    public FactoryDatasets(String nameLine, List<Double> listDados) {
        this.nameLine = nameLine;
        this.dados = this.setData(listDados);
        this.color = redoomColor();
    }

    private String setData(List<Double> listDados){
        String Label = "";
        for(Double d: listDados){
            Label = Label + d+", ";
        }
        return Label.substring(0, Label.length()-2);
    }

    public static String redoomColor(){
        Random ra = new Random();
        int r, g, b;
        r=ra.nextInt(255);
        g=ra.nextInt(255);
        b=ra.nextInt(255);
        return String.format("#%02X%02X%02X", r, g, b);
    }

    public String getDatasets() {
        return
            "{\n" +
            "    label: '"+nameLine+"',\n" +
            "    fill: false,\n" +
            "    data: ["+dados+"],\n" +
            "    lineTension: 0,\n" +
            "    backgroundColor: '"+color+"',\n" +
            "    borderColor: '"+color+"',\n" +
            "    borderWidth: 4,\n" +
            "    pointBackgroundColor: '"+color+"'\n" +
            "    }";
    }
}

```