



GOVERNO DO ESTADO DO  
**AMAZONAS**

**EST**

Escola Superior  
de Tecnologia da UEA

**UEA**  
UNIVERSIDADE  
DO ESTADO DO  
AMAZONAS

**UNIVERSIDADE DO ESTADO DO AMAZONAS**  
**ESCOLA SUPERIOR DE TECNOLOGIA**

**LUÍS ANTÔNIO VALOIS MIRANDA**

**MONITORAMENTO DE PARÂMETROS AMBIENTAIS DE UM LEITO  
HOSPITALAR UTILIZANDO ESP32**

**MANAUS – AM**  
**2019**

**LUÍS ANTÔNIO VALOIS MIRANDA**

**MONITORAMENTO DE PARÂMETROS AMBIENTAIS DE UM LEITO  
HOSPITALAR UTILIZANDO ESP32**

Projeto de pesquisa desenvolvido durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro Eletricista

Orientador: Prof. Dr. Fábio de Sousa Cardoso

MANAUS

2019

**Universidade do Estado do Amazonas – UEA**  
**Escola Superior de Tecnologia - EST**

*Reitor:*

**Cleinaldo de Almeida Costa**

*Vice-Reitor:*

**Cleto Cavalcante de Souza Leal**

*Diretor da Escola Superior de Tecnologia:*

**Ingrid Sammyne Gadelha Figueiredo**

*Coordenador do Curso de Engenharia Elétrica:*

**Walfredo Lucena da Costa Filho**

*Banca Avaliadora composta por:*

**Prof. Fábio de Sousa Cardoso (Orientador)**

**Prof. Karlo Homero Ferreira Santos**

**Prof. Almir de Oliveira Costa Júnior**

*Data da defesa: 19/12/2019.*

## **CIP – Catalogação na Publicação**

Miranda, Luís Antônio Valois

Monitoramento de parâmetros ambientais de um leito hospitalar utilizando ESP32/ Luís Antônio Valois Miranda; [orientado por] Fábio de Sousa Cardoso. – Manaus: 2019.

44 f. p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2019.

1. ESP32. 2. *Message Queuing Telemetry Transport* (MQTT). 3. *Internet of Things* (IoT).

I. Fábio de Sousa Cardoso

LUÍS ANTÔNIO VALOIS MIRANDA

MONITORAMENTO DE PARÂMETROS AMBIENTAIS DE UM LEITO HOSPITALAR  
UTILIZANDO ESP32

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Nota obtida: \_\_\_\_\_ (\_\_\_\_\_)

Aprovado em \_\_\_\_/\_\_\_\_/\_\_\_\_.

Área de concentração: Sistemas Embarcados

BANCA EXAMINADORA

---

Orientador: Fábio de Sousa Cardoso, Dr.

---

Avaliador: Karlo Homero Ferreira Santos, Me.

---

Avaliador: Almir de Oliveira Costa Júnior, Me.

Manaus  
2019

## DEDICATÓRIA

Dedico este trabalho aos meus familiares: meus pais, para meu irmão e para minha tia.

## **AGRADECIMENTOS**

Agradeço a Deus pela sabedoria inspirada e pelas bênçãos concedidas, por esta etapa concluída e pelo dom da Perseverança. Grato a Ele também pela Sabedoria inspirada.

Agradeço também aos meus pais, Cláudia e José, minha tia Letícia, e familiares, pelo apoio sempre concedido, mesmo nos momentos de dificuldades, sempre foram uma Fortaleza.

Agradeço ao meu irmão, André, que posso considerar meu melhor amigo, meu imenso agradecimento.

Ao professor Fábio Cardoso pela paciência e pela compreensão das minhas limitações, agradeço sempre pela ajuda concedida durante este período.

Agradeço aos amigos da GE HealthCare, que me instruíram enquanto estagiário, me ajudaram e orientaram: Alexandre, Edmilson e Zenildo. Espero em breve retornar ao time.

E por fim, agradeço aos amigos na UEA, em especial por terem me ajudado e não permitido desistir nos momentos mais difíceis; Cassiana e Paulo. Aos demais amigos e colegas da EST meu muitíssimo obrigado.

## RESUMO

Este trabalho tem o objetivo de apresentar o processo do desenvolvimento de um protótipo para coleta de parâmetros ambientais presentes em um leito hospitalar, como por exemplo, temperatura e umidade. Possui como objetivo a otimização da comunicação entre leito-enfermaria ou leito-acompanhante, visando o conforto do paciente e diminuição de sofrimento do mesmo, numa tentativa de humanização hospitalar. Algumas funções disponíveis são as coletas de temperatura e umidade, botão de emergência e sensor de impacto. Utilizou-se o ESP32, da fabricante ESPRESSIF, como microcontrolador, substituindo-se o Arduino e o Raspberry Pi inicialmente propostos. O ESP32 possibilita a conexão com Wi-Fi e a criação de uma nuvem, ou seja, os dados coletados pelos sensores trafegam pela nuvem e são impressos a um usuário final, utilizando os conceitos de Internet das Coisas (*Internet of Things* - IoT). Para a comunicação entre o ESP32 e a saída utiliza-se um protocolo de mensagens leve, baseado em TCP/IP, conhecido por *Message Queuing Telemetry Transport* - MQTT, onde se precisa ter um servidor cadastrado para fazer a interface de dados coletados. Por fim, os dados coletados aparecem no monitor, ou no aplicativo de celular, melhorando a comunicação e podendo trazer mais conforto e bem-estar ao paciente.

Palavras Chave: ESP32 – *Message Queuing Telemetry Transport* (MQTT) - *Internet of Things* (IoT).

## ABSTRACT

This paper aims to present the process of developing a prototype to collect environmental parameters present in a hospital bed, such as temperature and humidity. Its objective is to optimize communication between the ward or accompanying bed, aiming at patient comfort and reducing their suffering, in an attempt to hospital humanization. Some available functions are temperature and humidity collections, emergency button and impact sensor. The ESP32 from ESPRESSIF was used as a microcontroller, replacing the originally proposed Arduino and Raspberry Pi. ESP32 makes it possible to connect to Wi-Fi and create a cloud, meaning that data collected by sensors travels across the cloud and is printed to an end user using the concepts of Internet of Things (IoT). For communication between ESP32 and outbound, a lightweight TCP / IP based messaging protocol known as Message Queuing Telemetry Transport (MQTT) is used, where a registered server is required to interface the collected data. Finally, the data collected appears on the monitor or mobile application, improving communication and may bring more comfort and well-being to the patient.

Key Words: ESP32 – *Message Queuing Telemetry Transport (MQTT)* - *Internet of Things (IoT)*.



## LISTA DE FIGURAS

Figura 1 - Arquitetura Básica de um “nó” de sensor.....	16
Figura 2 - Alegoria de um Sistema IoT.....	17
Figura 3 - Módulo Microcontrolador ESP32.....	18
Figura 4 - Pinagem do ESP 32.....	20
Figura 5 - Ilustração de comunicação com protocolo MQTT.....	21
Figura 6 - IDE do Arduino configurada para programação no ESP32.....	22
Figura 7 - IDE do Arduino.....	23
Figura 8 - Funcionamento resumido do protótipo.....	24
Figura 9 - Diagrama completo do funcionamento do protótipo.....	25
Figura 10 - Circuito de monitoramento.....	27
Figura 11 - Sensor de Temperatura DHT11.....	28
Figura 12 - DHT11 Pinagem.....	29
Figura 13 - KY-031 sensor de impacto.....	29
Figura 14 - Pinagem do KY-031.....	29
Figura 15 - Servidor ( <i>Broker</i> ) MQTT – CloudMQTT.....	30
Figura 16 - MQTT Dash – Aplicativo.....	31
Figura 17 - Bibliotecas utilizadas no código.....	32
Figura 18 - Código fonte.....	32
Figura 19 - Função para conectar ao Wi-Fi.....	33
Figura 20 - Loop de duas strings declaradas.....	34
Figura 21 - Implementação do Circuito em Protoboard.....	35
Figura 22 - Monitor Serial do ESP32.....	36
Figura 23 - Aplicação final na tela de usuário.....	37

## LISTA DE TABELAS

Tabela 1 - Comparativo de tecnologias de Microcontroladores.....	19
Tabela 2 - Principais parâmetros de código para comunicação MQTT .....	21
Tabela 3 - Parâmetros do servidor ( <i>broker</i> ) para utilizar no código.....	31

## LISTA DE SIGLAS

ADC - *Analog to Digital Converter*;  
DAC – *Digital to Analog Converter*;  
ECG – *Eletrocardiograma*;  
GPIO - *General Purpose Input/Output*;  
IoT - *Internet of Things*;  
LED - *Light Emitting Diode*;  
MQTT - *Message Queuing Telemetry Transport*;  
NTC - *Negative Temperature Coefficient*;  
PWM- *Pulse Width Modulation*;  
RAM - *Random Access Memory*;  
RF - *Radio Frequency*;  
RFID - *Radio Frequency Identification*;  
RSSF - *Rede De Sensores Sem Fio*;  
SSL - *Secure Sockets Layer*;  
TLS - *Transport Layer Security*;  
TCP / IP - *Transmission Control Protocol / Internet Protocol*;  
USB – *Universal Serial Bus*;

## SUMÁRIO

<b>INTRODUÇÃO</b> .....	13
<b>1. REFERENCIAL TEÓRICO</b> .....	15
1.1. REDE DE SENSORES SEM FIO (RSSF).....	15
1.2. <i>INTERNET OF THINGS</i> (IoT).....	16
1.3. ESP32 .....	18
1.4. <i>MESSAGE QUEUING TELEMETRY TRANSPORT</i> (MQTT).....	20
1.5. AMBIENTE DE PROGRAMAÇÃO .....	21
<b>2. METODOLOGIA</b> .....	24
2.1. DIAGRAMA EM BLOCOS .....	24
2.2. FUNCIONAMENTO .....	25
2.3. ESQUEMA DO CIRCUITO .....	26
<b>3. IMPLEMENTAÇÃO</b> .....	28
3.1. ESP32 .....	28
3.2. DHT11 .....	28
3.3. COLISÃO.....	29
3.4. LEDS E BOTÕES .....	30
3.5. <i>BROKER</i> UTILIZADO .....	30
3.6. APLICAÇÃO .....	31
3.7. DESENVOLVIMENTO DO CÓDIGO .....	31
3.8. IMPLEMENTAÇÃO FÍSICA.....	34
<b>4. TESTES E RESULTADOS OBTIDOS</b> .....	36
<b>CONCLUSÃO</b> .....	38
<b>REFERÊNCIAS BIBLIOGRÁFICAS</b> .....	39
<b>APÊNDICE – CÓDIGO FONTE</b> .....	41

## INTRODUÇÃO

Presencia-se nos últimos tempos a massificação das internações em leitos de enfermarias ou em apartamentos nos hospitais. Isso tem feito com que os pacientes fiquem dispersos. Nem sempre se tem uma boa comunicação com a enfermaria. Os cuidadores também por vezes ficam sem informações prévias do que acontece com os pacientes.

Outrossim, ambientes hospitalares necessitam de constante monitoramento, para se garantir conforto aos pacientes dispostos nos leitos. Na maioria das situações, apenas se dispõe de um botão para chamar um colaborador na enfermaria, e nada além disso. Trata-se de um sistema bastante simples, com poucas interações. A Agência Nacional de Vigilância Sanitária (ANVISA), que é responsável pela fiscalização das unidades hospitalares, explana no manual de tecnovigilância (ANVISA, 2010), as prováveis falhas de acompanhamento em situações adversas, que podem ocorrer em ambientes hospitalares, considerando a falta de acompanhamento da temperatura e umidade, desde a acomodação de pacientes, até o armazenamento de medicamentos e vacinas. Aumenta-se também o interesse pela comercialização de sensores, e se alia ao uso clínico, integrando-se ao conceito de Internet das Coisas (Internet of Things - IoT), onde se propõe que sistemas sejam formados por diversos dispositivos capazes de realizar a comunicação entre si, chamado também como comunicação entre máquinas (machine-to-machine), conforme (ASTHON, 1999).

Portanto, este trabalho tem o intuito de desenvolver uma plataforma de rede de sensores, que enviam as informações coletadas a um servidor, disponibilizando a informação final a um usuário, em aplicações disponíveis para desktop e smartphone. Este trabalho tem como meta trazer mais conforto e qualidade de vida aos enfermos e convalescentes, possibilitando a dinamização da comunicação entre leito-enfermaria e leito-cuidador. Atualmente existem alguns projetos de implementação de monitoramento, como o projeto de (WHITCHURCHEM ET AL, 2017), em países de terceiro mundo onde não se é disponível os equipamentos hospitalares, e o projeto em questão tem como finalidade monitorar alguns parâmetros como Eletrocardiograma (ECG), Oximetria, entre outros, através destes sensores conectados do paciente a um RaspBerry Pi, e então, através de uma plataforma de *Internet of Things* (IoT), seria disponibilizado numa nuvem aos médicos e enfermeiros.

Neste projeto será utilizado como microcontrolador, substituindo o Arduino e o RaspBerry Pi pelo ESP32 (conforme explicação adiante), da fabricante chinesa ESPRESSIF, possuindo plataformas de Wi-Fi e *Bluetooth* integradas, sendo um grande fator de otimização em custo e em nível de circuitos.

Para organização deste trabalho, estão dispostos 4 capítulos onde se encontram o conteúdo dele, além das referências e anexo:

Capítulo 1 – Referencial Teórico: Tem como objetivo explicar a teoria utilizada neste trabalho, bem como conceitos e a teoria do hardware empregado;

Capítulo 2 – Metodologia: Nesta etapa será mostrado como se desenvolveu o diagrama de blocos e circuito do projeto, bem como a proposta inicial de construção;

Capítulo 3 – Implementação: Serão mostrados materiais utilizados, bem como a montagem do protótipo, e pequena explicação do código;

Capítulo 4 – Resultados: Nesta etapa temos os resultados coletados durante o desenvolvimento do projeto.

## 1. REFERENCIAL TEÓRICO

Existem alguns modelos propostos por outros estudos até então no que se diz respeito à monitoramentos de ambientes, onde podem ser ambientes industriais, ambientes domésticos, ambiente rural (campo). Alguns modelos foram apresentados anteriormente, como o modelo de (WHITCHURCHEM et al., 2017). Outros modelos, como o modelo de (KELLY, 2013), utilizando-se o hardware de maneira customizada para monitorar o ambiente de uma residência, coletando-se informações de temperatura, luminosidade e consumo energético, sendo depois expostos a uma plataforma na web para consumo.

Neste trabalho será focado o monitoramento de um ambiente de um leito hospitalar. Serão explicados alguns conceitos a serem utilizados no escopo deste trabalho.

### 1.1. REDE DE SENSORES SEM FIO (RSSF)

Primeiramente tem-se a implementação de uma Rede de Sensores Sem Fio (RSSF), onde os sensores estão dispostos num arranjo específico, e por fim, estes enviam as informações coletadas a um *gateway* (para realizar a interface com as outras redes e protocolos). Na literatura são chamados de “Nós” Estes Nós por sua vez possuem um protocolo específico de comunicação.

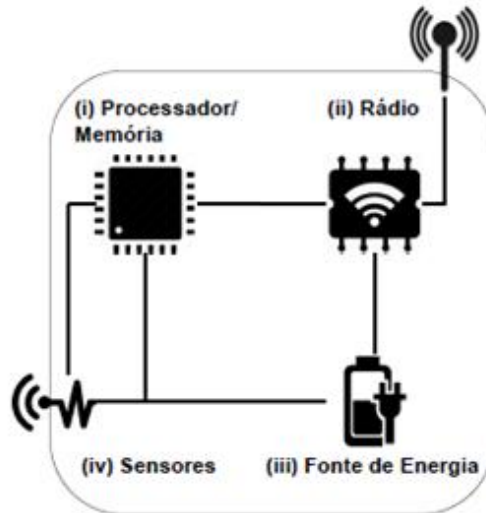
Segundo (CANTENHEDE E SILVA, 2014) estes sensores possuem várias aplicações disponíveis; aplicações industriais, domésticas, para o campo, área de saúde, entre outros. A aplicação a ser utilizada para este trabalho será para a área de saúde (*HealthCare*). Uma rede de sensores possui algumas características, segundo (CANTENHEDE E SILVA, 2014);

- Endereçamento de sensores;
- Agregação de dados;
- Mobilidade dos sensores;
- Restrições dos dados coletados;
- Quantidade de sensores;
- Limitação da energia disponível;
- Auto-organização da rede;
- Tarefas colaborativas;
- Capacidade de responder a consultas;

Segundo (LOUREIRO et al., 2014), as características básicas de um sensor são: o transceptor, memória, processador, sensor e bateria. A unidade sensorial coleta as informações analógicas físicas do ambiente, e depois converte para informação digital através de um

conversor analógico-digital. No processamento se tem o controle do nó, que é formado por uma memória e uma unidade de armazenamento. Na unidade transceptora se tem a comunicação com os outros sensores (nós), onde se converte a informação de bits para Rádio Frequência (RF), e enviase a outros sensores. Na unidade de energias tem se a alimentação do sensor.

Figura 1– Arquitetura Básica de um “nó” de sensor.



Fonte: SANTOS, SILVA, (ca 2010)

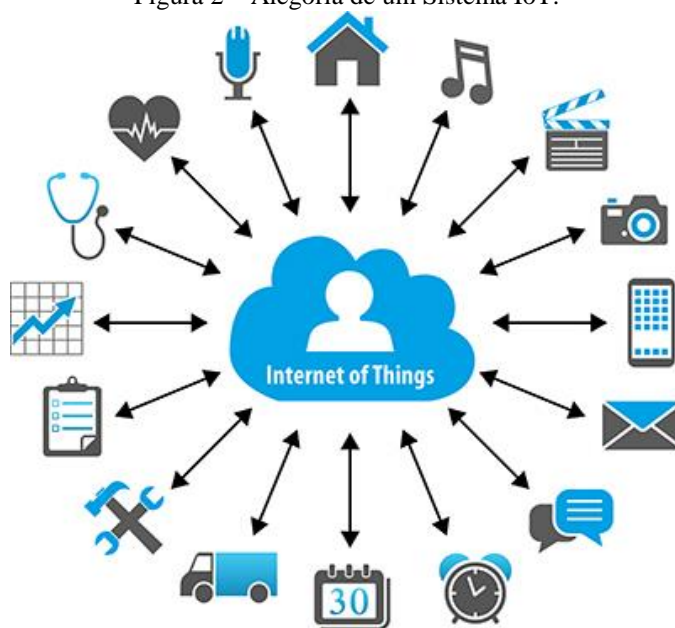
## 1.2. INTERNET OF THINGS (IoT)

O conceito de Internet das Coisas (*Internet of Things* – IoT) surgiu em 1999, num artigo no RFID Journal (ASHTON, 1999), onde se propunha a interligação da cadeia de suprimentos de uma empresa utilizando *Radio Frequency Identification* (RFID) – Identificação por Rádio Frequência.

A internet das coisas aparece-nos como um agrupamento de várias áreas pouco ou muito similares, como sistemas embarcados, microeletrônica, comunicação e sensoriamento, conforme (CANTENHEDE E SILVA, 2014). A Internet das Coisas trata-se então da continuação da internet como já se conhece, porém, fazendo com que objetos (coisas) do quotidiano tenham a facilidade de se conectar com a Internet.



Figura 2 – Alegoria de um Sistema IoT.



Fonte: Libelium, 2019.

Os objetos (coisas) inteligentes possuem então capacidade de comunicação e processamento. Fazendo um paralelo com outros objetos, não só mais os computadores estão conectados à Internet, mas também tablets, TVs, smartphones, consoles de jogos, webcams, etc., onde se aumenta a cada dia a lista. Outrossim, uma gama de possibilidades de aplicações surge (ex: cidades inteligentes (*Smart Cities*), saúde (*HealthCare*), casas inteligentes (*SmartHome*)) e desafios emergem (regulamentações, segurança, padronizações).

A arquitetura de um sistema de objetos conectados na Internet deve ser bastante flexível, onde tem-se muitos dispositivos com diferentes arquiteturas, podendo serem simples ou sofisticados. O modelo básico apresenta 3 camadas, conforme (LOUREIRO et al., 2014):

- Camada de Percepção – Representa o mundo físico, aos quais os sensores captam as informações;
- Camada de Rede - Na camada de rede ocorre a comunicação, roteamento, serviços de gerenciamento que devem ser realizados;
- Camada de Aplicação – Informar os resultados de coletas e serviços realizados aos clientes;

O protocolo padrão utilizado continua o IPv4, onde se possibilita estar conectado com a Internet. O protocolo IPv6 também pode ser utilizado na escassez de possibilidades de endereços IPv4, porém, o tamanho do pacote de informações seria maior, necessitando de uma correção de protocolo para envio e recebimento de informações. Neste projeto, continuará a usar-se o habitual, no caso, o IPv4.

### 1.3. ESP32

Conforme o próprio fabricante (ESPRESSIF 2018), o ESP32 é um microcontrolador de baixo custo e que tem integrado interfaces Wi-Fi e Bluetooth. É da mesma família do ESP8266, desenvolvidos pela empresa chinesa ESPRESSIF. Ou seja, além de permitir a alocação dos sensores nos pinos, os dados são enviados, por intermédio de um servidor, a uma nuvem, onde a saída pode ser monitorada por meio de um aplicativo, ou sendo vista como uma página na Web.

Figura 3 – Módulo Microcontrolador ESP32.



Fonte: FilipeFlop, 2019.

O módulo possui a saída de tensão em 3,3 volts, sendo esta a alimentação utilizada nos sensores presentes, e possui também 30 pinos. O código é enviado da IDE do Arduino (explicado em detalhes posteriormente) por um cabo micro USB, (ESPRESSIF, 2018), além de dois botões; *Enable* (En) e *Boot*. Na Figura 4 dispõe-se da Pinagem do ESP32. Os pinos *General Purpose Input/Output* (GPIO) são responsáveis por receber ou enviar informações digitais a outros pinos. Os pinos ADC (*Analog to Digital Converter*) permitem também receber ou enviar informações a outros pinos, porém de entrada ou saída analógicas.

Abaixo, na Tabela 1, temos o comparativo entre tecnologias empregadas em microcontroladores:

Tabela 1 – Comparativo de tecnologias de Microcontroladores.

	ESP32	ESP8266	ARDUINO UNO
Cores	2	1	1
Arquitetura	32 bits	32 bits	8 bits
clock	160 MHz	80 MHz	16 MHz
Wi-Fi	Sim	Sim	Não
Bluetooth	Sim	Sim	Não
RAM	512 Kb	160 Kb	2 Kb
FLASH	16 Mb	16 Mb	32 Kb
GPIO	32	17	14
ADC	18	1	6
DAC	2	0	0

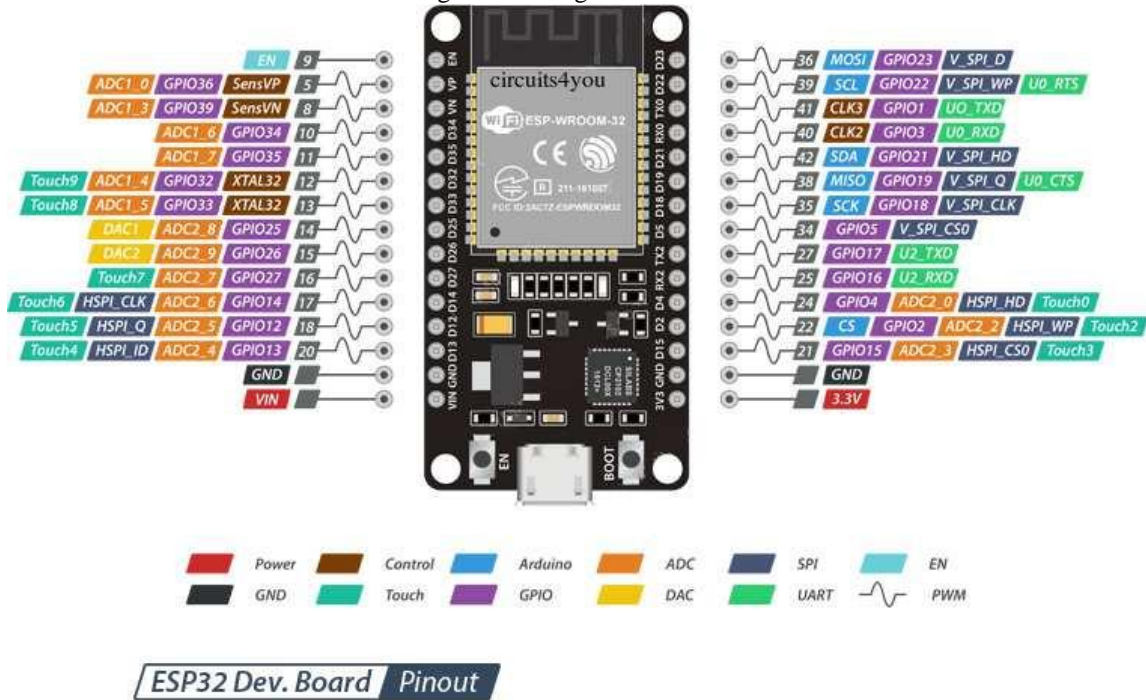
Fonte: Koyonage, 2017.

Conforme (KOYANAGE, 2017), algumas características do ESP32 são:

- Chip com WiFi embutido: padrão 802.11 B/G/N, operando na faixa de 2.4 a 2.5GHz;
- Modos de operação: Client, Access Point, Station + Access Point;
- Microprocessador dual core Tensilica Xtensa 32-bit LX6;
- Clock ajustável de 80MHz até 240MHz;
- Tensão de operação: 3.3 VDC;
- Corrente máxima por pino é de 12mA;
- GPIO com função PWM;
- Possui Bluetooth v4.2 (Bluetooth Low Energy);

Mesmo sendo uma tecnologia recente, percebe-se o quão robusto é o ESP32, mostrando muitas vantagens para substituir o Arduino, e microcontroladores da própria família, como o ESP8266. Outra característica interessante do ESP32 é o LED *on board*, que é representado pelo GPIO2, que é o LED vermelho. Outro LED *on board* é o da placa quando conectada, representado pelo LED azul.

Figura 4 - Pinagem do ESP 32



Fonte: Circuits4You.com, 2018.

#### 1.4. MESSAGE QUEUING TELEMETRY TRANSPORT (MQTT)

*Message Queuing Telemetry Transport* (MQTT), refere-se a um protocolo de mensagens leves para pilha de redes de protocolo TCP/IP. Criado pela IBM em 1999, este protocolo deve lidar com alta latência, baixa largura de banda e instabilidade na comunicação. Conforme (YUAN, 2017), a flexibilidade do MQTT possibilita o suporte a diversos cenários de aplicativo para dispositivos e serviços de Internet das Coisas – IoT.

O protocolo MQTT tem dois tipos de objetos na rede: um servidor *broker* e inúmeros clientes. O *broker* recebe todas as mensagens dos clientes e roteia na nuvem. Por outro lado, o cliente é tudo aquilo que pode interagir com o *broker* pra receber as mensagens. A comunicação é feita então pelo seguinte passo a passo, conforme (YUAN, 2017) :

1. O cliente se conecta ao *broker*. Ele pode assinar qualquer "tópico" de mensagem no *broker*. Essa conexão pode ser uma conexão simples ou uma conexão criptografada para mensagens sensíveis;
2. O cliente publica as mensagens em um tópico, enviando a mensagem e o tópico ao *broker*;
3. Em seguida, o *broker* encaminha a mensagem a todos os clientes que assinam esse tópico;

Alguns parâmetros são interessantes serem explicados na Tabela 2:

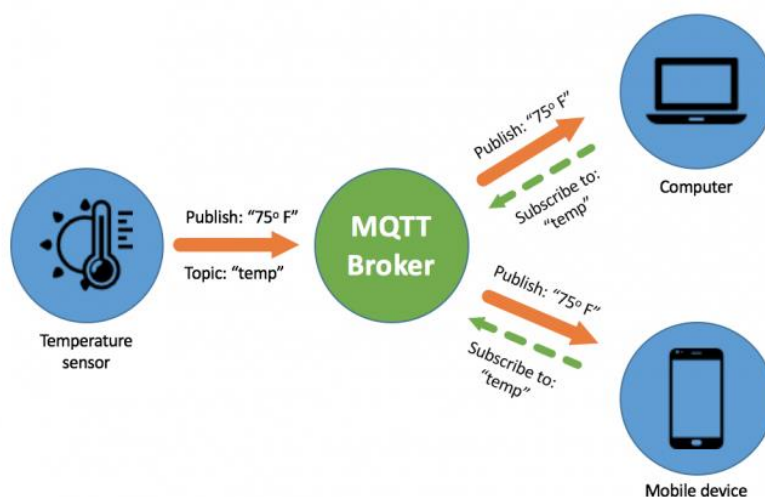
Tabela 2 – Principais parâmetros de código para comunicação MQTT

Parâmetro	Descrição
<i>publish</i>	Enviar informação do broker para a saída
<i>subscribe</i>	Retorno de informação da saída para o broker
<i>username</i>	Usuário fornecido pelo servidor
<i>password</i>	Senha fornecida pelo servidor
<i>client.setServer</i>	Instância o cliente
<i>WiFiClient</i>	Cria o objeto
<i>PubSubClient client</i>	Publicar ao cliente

Fonte: O próprio autor.

Abaixo ilustra-se como se dá a comunicação com o protocolo MQTT:

Figura 5 – Ilustração de comunicação com protocolo MQTT.



Fonte: Pplware, 2019.

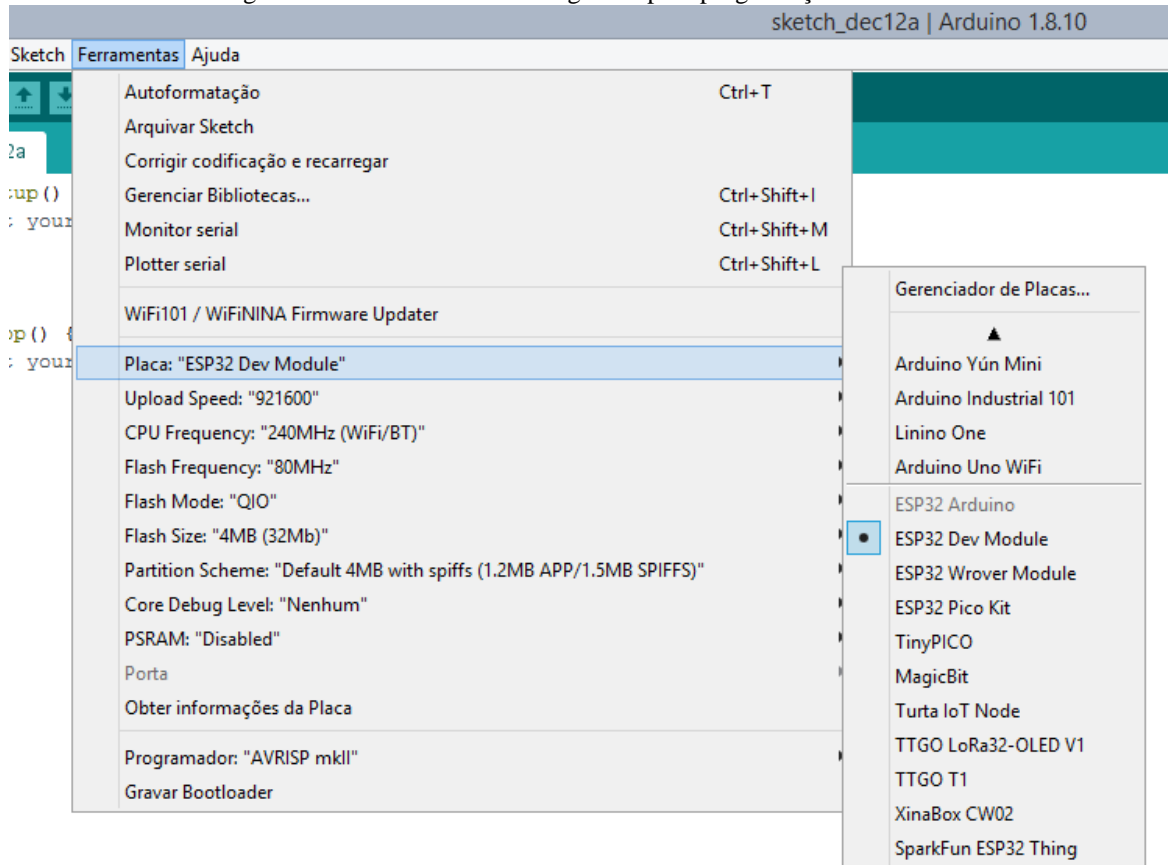
O servidor irá informar alguns parâmetros necessários (usuário, senha e porta) para se ter a comunicação entre *broker* e cliente, que podem ser vistos mais adiante na Tabela 3, e serão explanados mais adiante.

### 1.5. AMBIENTE DE PROGRAMAÇÃO

Há várias formas possíveis de programação para o ESP32, como *Espressif IoT Development Framework* ou *MicroPython*, porém pela familiaridade de uso, optou-se pela IDE do Arduíno. Apenas é necessário instalar o *driver* do ESP32, e sua biblioteca também, e alocar na pasta da biblioteca da IDE do Arduíno. Depois, tem a opção de “Placa” no menu “Ferramentas”, e selecionar “Esp32 Dev Module”, ao invés de selecionar qualquer tipo de placa

de Arduino. Basicamente a programação é similar ao Arduino, adicionando-se apenas alguns comandos específicos para a comunicação com o broker, visto na tabela

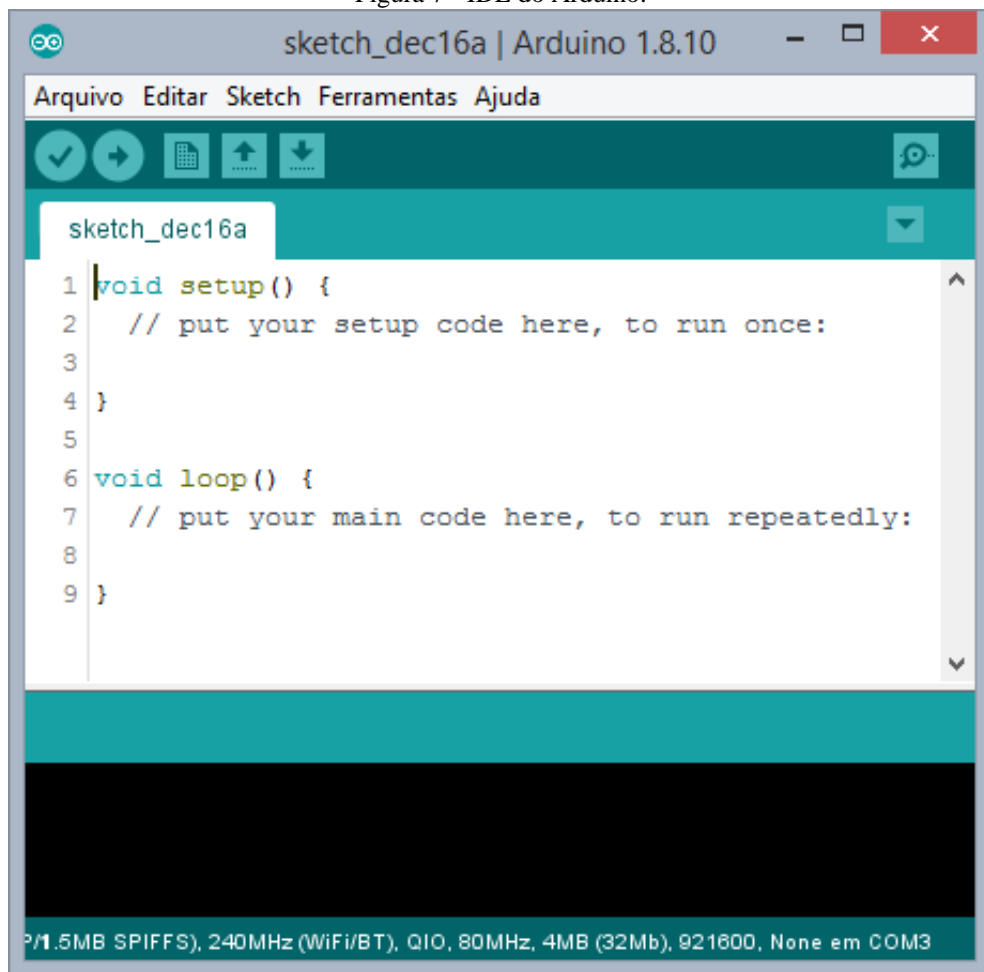
Figura 6 - IDE do Arduino configurada para programação no ESP32.



Fonte: O próprio autor.

A IDE do Arduino, conforme (ARDUINO, 2019) consiste em um ambiente baseado em C, com o código dividido em *setup* e *loop*. Em *setup* temos a inicialização de pinos em entrada e saída e a inicialização das variáveis. Em *loop* teremos a repetição do código enquanto a placa estiver ligada. Previamente antes desses dois parâmetros, deve-se inserir no código as bibliotecas necessárias para a compilação do mesmo. A vantagem da IDE do Arduino é ser “ecclética”, permitindo trabalhar com MQTT, mas também outras linguagens, como HTML.

Figura 7 - IDE do Arduino.



Fonte: O próprio autor.

## 2. METODOLOGIA

O presente trabalho trata-se de uma Pesquisa Aplicada, ou seja, tem objetivo a utilização de informações disponíveis para criar novas tecnologias e métodos, mas também comprovar ou rejeitar hipóteses sugeridas em modelos teóricos (RODRIGUES, 2007). A pesquisa Aplicada tem como objetivo a pesquisa exploratória, onde tem como finalidade proporcionar maior familiaridade com o problema, tornando o mesmo mais explícito, ou então, construir hipóteses (GIL, 1987). Será utilizado como método de abordagem o hipotético-dedutivo. A elaboração seguirá o procedimento monográfico. A coleta de dados será disposta através de documentação indireta, onde os dados são qualitativos.

É importante frisar o embasamento técnico utilizado, oriundo das disciplinas vistas durante o curso de graduação em Engenharia Elétrica. Algumas matérias são a base para este trabalho, tais como Comunicações Digitais, Tópicos de Arduino, Linguagem de Programação I e II, Microcontroladores e Redes de comunicações de Dados.

Coletou-se dados também em artigos e sites para embasamento teórico deste trabalho, focando em artigos e sites explanando a tecnologia do ESP32 e artigos que explorassem outros tipos de monitoramento, para servir de comparativo com o então trabalho. Buscou-se compreender o funcionamento do protocolo MQTT, assim como aplicações práticas. Determinou-se o *broker* e a aplicação a ser utilizada no *smartphone*.

Com todo conhecimento técnico e teórico adquirido, começou-se então a selecionar os materiais a serem utilizados, e se iniciou a construção do algoritmo.

### 2.1. DIAGRAMA EM BLOCOS

Figura 8- Funcionamento resumido do protótipo.

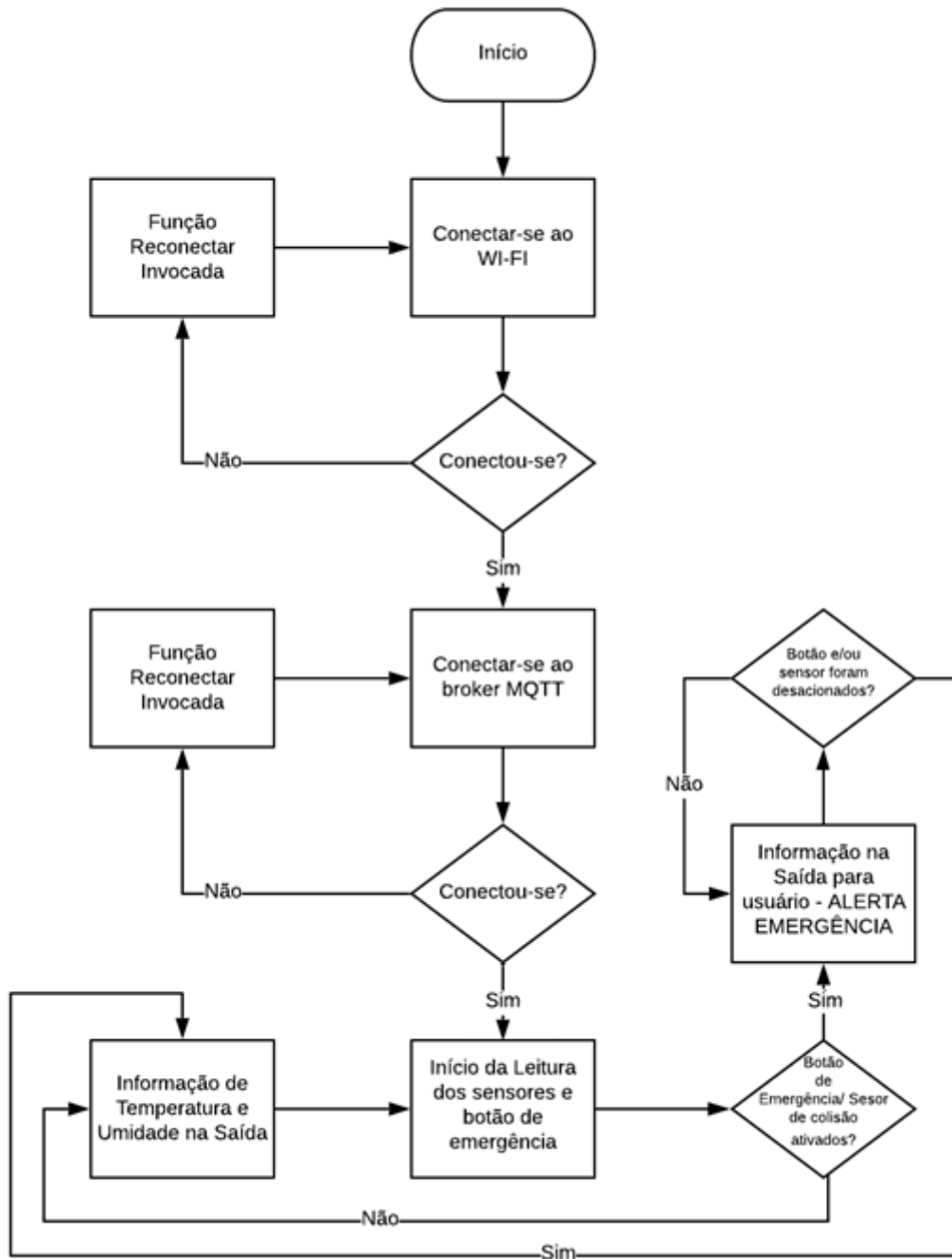


Fonte: O próprio autor.



## 2.2. FUNCIONAMENTO

Figura 9 - Diagrama completo do funcionamento do protótipo



Fonte: O próprio autor

Para a primeira parte deste trabalho, propunha-se construir a rede de sensores sem fio com o Arduino de Microcontrolador, juntamente com o RapBerry Pi para ser o servidor da rede sem fio – este executaria a função de se criar a nuvem com os dados. Sugeriu-se também outros

protocolos para a transmissão sem fio, como o *Bluetooth*, que traçaria as informações dos sensores.

Otimizando a execução deste trabalho, convencionou-se substituir ambos microcontroladores citados acima por apenas um, o ESP32, citado em seções anteriores deste trabalho, onde o próprio além de servir de interface para os sensores, possui também capacidade de comunicação sem fio, pois possui interface Wi-Fi e *Bluetooth*. Outras características são baixo consumo de energia, design robusto, entre outros, conforme o site do fabricante.

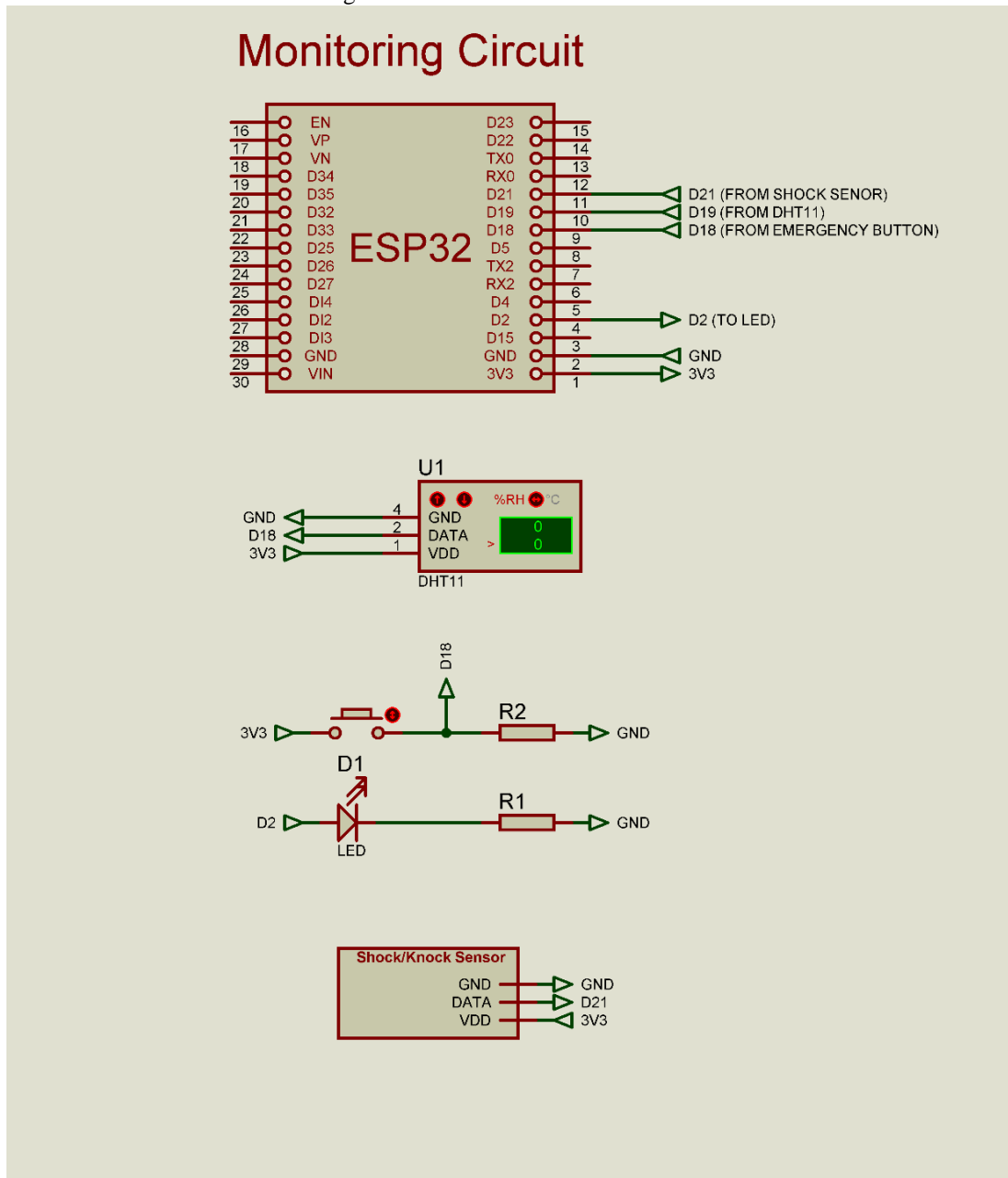
Resumidamente, o funcionamento será feito da seguinte maneira: os sensores irão executar a leitura dos parâmetros ambientais (temperatura e umidade), e mais outros dois parâmetros, como sensor de colisão e o botão de emergência. Estes estarão conectados ao ESP32, que será o microcontrolador que fará a interface entre os sensores e a comunicação com o servidor, através do protocolo MQTT. O servidor informará alguns parâmetros importantes, que serão utilizados no código e na aplicação de saída para o usuário.

Um detalhe deste projeto é o botão de emergência, que só é desativado no local do leito, evitando ser desarmado por alguém remotamente, e conseqüentemente, o paciente ser ignorado.

### 2.3. ESQUEMA DO CIRCUITO

Na figura abaixo encontra-se o circuito, elaborado a partir do *software* de simulação de circuitos eletrônicos, *Proteus 8 Professional*. Um detalhe importante é; sendo o ESP32 uma plataforma recente, se fez necessário construir no simulador o módulo do ESP e o módulo do sensor de colisão. Na Figura 10, dispõe-se o próprio ESP, os sensores DHT11 e KY-031 (foram estes selecionados e serão melhores explicados no próximo item), além do circuito do botão de emergência, bem como os pinos em que estão conectados.

Figura 10 - Circuito de monitoramento



Fonte: Próprio do autor.

### 3. IMPLEMENTAÇÃO

Nesta seção serão descritos os materiais utilizados e como se implementou:

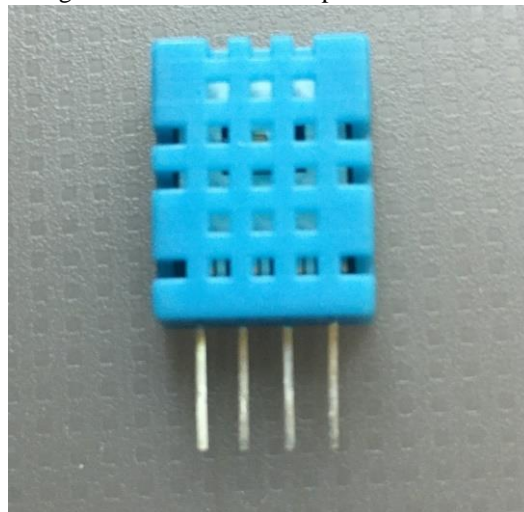
#### 3.1. ESP32

Conforme descrito anteriormente, utilizou-se o microcontrolador o ESP32, otimizando o circuito e eliminando também custos. Ligou-se os sensores aos pinos de GPIO do ESP, onde se pode consultar na Figura 10.

#### 3.2. DHT11

O sensor modelo DHT11 faz leitura em tempo real da temperatura e umidade relativa do ar. Este sensor trabalha com um componente de medida de umidade relativa do tipo resistivo e um componente de medida de temperatura do tipo *Negative Temperature Coefficient* - NTC, deste conforme (JUNIOR et al, 2014). A vantagem deste sensor é a comunicação com microcontroladores, tipo Arduino, ou o próprio ESP32.

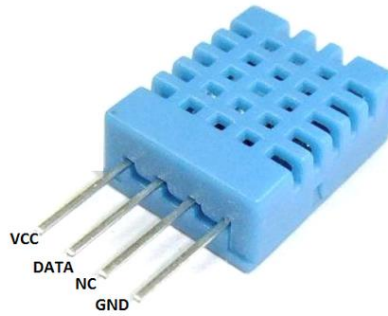
Figura 11 - Sensor de Temperatura DHT11.



Fonte: O próprio autor.

Na sequência, a pinagem do mesmo:

Figura 12 - DHT11 Pinagem.



Fonte: IndianMart, 2019.

### 3.3. COLISÃO

Para verificar colisões presentes no ambiente, utilizou-se o sensor KY-031, também com interface disponível para o Arduino e o próprio ESP32. Abaixo, temos a Figura 12:

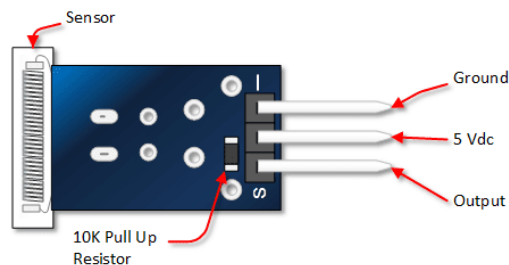
Figura 13 - KY-031 sensor de impacto.



Fonte: IndianMart, 2019.

Na Figura 14 temos a pinagem do sensor de Impacto:

Figura 14 - Pinagem do KY-031



Fonte: Mikrokontroler Palembang, 2016.

### 3.4. LEDS E BOTÕES

Alocou-se um botão de emergência ao ESP32, simulando o paciente pressionando o mesmo para situações de emergência, e a resposta terá saída do LED, que permanecerá piscando até o botão ser novamente pressionado.

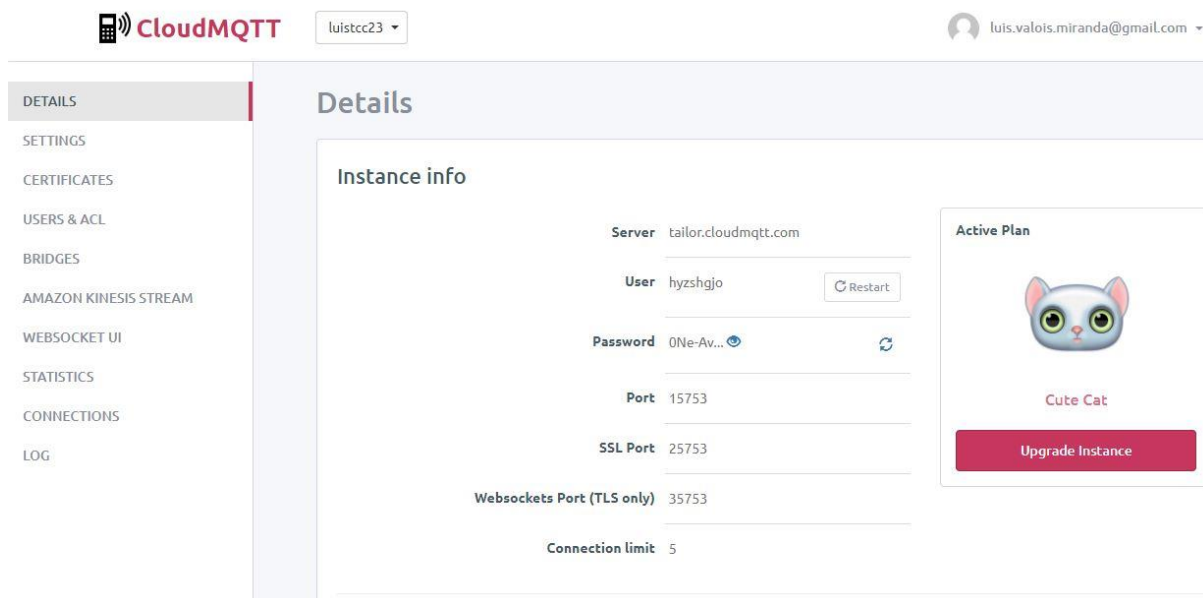
### 3.5. BROKER UTILIZADO

Conforme descrito inicialmente, para se ter a comunicação com o protocolo MQTT é necessário se ter um servidor, o *broker*. Por isso, deve-se escolher um servidor *broker* para receber as mensagens. O servidor utilizado neste trabalho é o *CloudMQTT*.

Este servidor disponibiliza alguns tipos de pacotes de serviços para servidores e clientes, sendo escolhido o plano de um *broker* e cinco clientes. Este tipo de plano é gratuito, e é suficiente para o desenvolvimento deste trabalho, sendo planos com mais de cinco clientes ser necessário comprar o pacote.

Na figura abaixo encontramos a página de cadastro do *broker*, onde temos as informações necessárias para uso no código para estabelecer a comunicação, fornecidas pelo servidor:

Figura 15 - Servidor (*Broker*) MQTT – CloudMQTT.



Fonte: CloudMQTT, 2019.

Na tabela 3 temos as informações fornecidas na figura acima, onde as informações com asterisco (\*) foram utilizadas no código:

Tabela 3 - Parâmetros do servidor (broker) para utilizar no código.

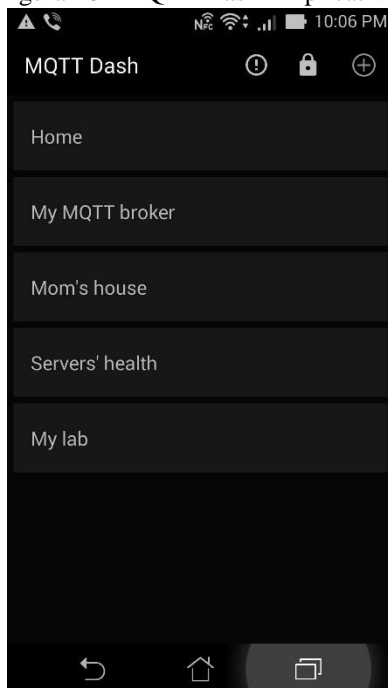
<i>Server (*)</i>	tailor.cloudmqtt.com
<i>User (*)</i>	hyzshgjo
<i>Password (*)</i>	0Ne-AvkK1xU5
<i>Port (*)</i>	15753
<i>SSL Port</i>	25753
<i>Websockets Port (TLS Only)</i>	35753
<i>Connection limit</i>	5

Fonte: O próprio autor.

### 3.6. APLICAÇÃO

Para aplicação utilizou-se um aplicativo já existente para sistemas *Android*, o *MQTT Dash*, onde cria-se o ambiente com os parâmetros a serem lidos a partir do *broker*. É um aplicativo grátis, sendo de fácil uso e customizável conforme a necessidade a ser empregado.

Figura 16 - MQTT Dash – Aplicativo.



Fonte: Apkpure, 2019.

### 3.7. DESENVOLVIMENTO DO CÓDIGO

O código completo utilizado no desenvolvimento deste trabalho encontra-se no Anexo.

Para se desenvolver o código, utilizou-se a IDE do Arduino mencionada anteriormente. Adicionaram-se algumas bibliotecas especiais, como a biblioteca do sensor de temperatura e umidade, a biblioteca Wi-Fi e a biblioteca própria do protocolo MQTT.

Figura 17- Bibliotecas utilizadas no código.

```
4 #include <DHT.h>
5 #include <WiFi.h>
6 #include <PubSubClient.h>
```

Fonte: O próprio autor.

Definiu-se então os pinos correspondentes aos sensores, LED e botão de emergência. Definiu-se também os tópicos a serem enviados para o broker. Definiu-se os parâmetros da rede Wi-Fi e os parâmetros necessários para o broker, fornecidos no site do servidor cadastrado. Algumas variáveis auxiliares foram declaradas também.

Figura 18 - Código fonte. Nesta parte temos a definição do que será enviado ao broker, nas linhas 16 até 19. Na linha 26 e 27 temos os dados do Wi-Fi, e da linha 30 até 34, os dados fornecidos pelo broker MQTT. Com posse desses dados, pode ser feita a conexão do ESP32 ao servidor.

```
14 #define DHTTYPE DHT11 //sensor em utilização: DHT22
15
16 #define TOPICO_PUBLISH_TEMPERATURA "topico_sensor_temperatura"
17 #define TOPICO_PUBLISH_UMIDADE "topico_sensor_umidade"
18 #define TOPICO "topico_led"
19 #define TOPICO_COLISAO "topico_colisao"
20
21
22 /* Variaveis, constantes e objetos globais */
23 DHT dht (DHTPIN, DHTTYPE);
24
25 //informações da rede WIFI
26 const char* ssid = "VIVOFIBRA-FE3F"; //SSID da rede WIFI
27 const char* password = "72231FFE3F"; //senha da rede wifi
28
29 //informações do broker MQTT
30 const char* mqttServer = "tailor.cloudmqtt.com";//server
31 const char* mqttUser = "hyzshgjo"; //user
32 const char* mqttPassword = "0Ne-AvkK1xU5"; //password
33 const int mqttPort = 15753; //port
34 const char* mqttTopicSub ="LuisTCC"; //tópico que sera assinado
35
```

Fonte: O próprio autor

Para otimizar o código, várias funções foram criadas, cada um com um objetivo específico. Criou-se as funções de leitura de temperatura e umidade, que retornam o valor lido pelo sensor na função principal. Criou-se a função de conectar-se ao Wi-Fi, onde será feita a conexão com um IP disponível na rede. Criou-se a função de conectar o servidor MQTT, e as funções de “reconnectMQTT” e “reconnectWiFi” caso tenha falha de conexão, será refeita a conexão no Wi-Fi e ao broker.



Figura 19 - Função para conectar ao Wi-Fi. Verifica-se que é impresso no Monitor Serial do Arduino o IP dinâmico na rede local escolhido pelo código. Dispõe-se também de função para conectar ao servidor MQTT (vide apêndice B para mais detalhes).

```
98 void reconnectWiFi(void)
99 {
100     //se já está conectado a rede WI-FI, nada é feito.
101     //Caso contrário, são efetuadas tentativas de conexão
102     if (WiFi.status() == WL_CONNECTED)
103         return;
104
105     WiFi.begin(ssid, password); // Conecta na rede WI-FI
106
107     while (WiFi.status() != WL_CONNECTED)
108     {
109         delay(100);
110         Serial.print(".");
111     }
112
113     Serial.println();
114     Serial.print("Conectado com sucesso na rede ");
115     Serial.print(ssid);
116     Serial.println("IP obtido: ");
117     Serial.println(WiFi.localIP());
118 }
```

Fonte: O próprio autor.

Depois tem as duas funções já presentes no IDE do Arduino, setup e loop. No setup chama-se as funções acima, define-se os sensores como entrada, e o LED como saída. E no loop que temos as informações retornadas das funções e enviadas para o broker, onde o usuário poderá efetuar a leitura dos dados na aplicação.

Figura 20 - No loop temos duas strings declaradas, que receberão os valores retornados das funções de leitura de temperatura e umidade – e serão enviadas ao *broker*.

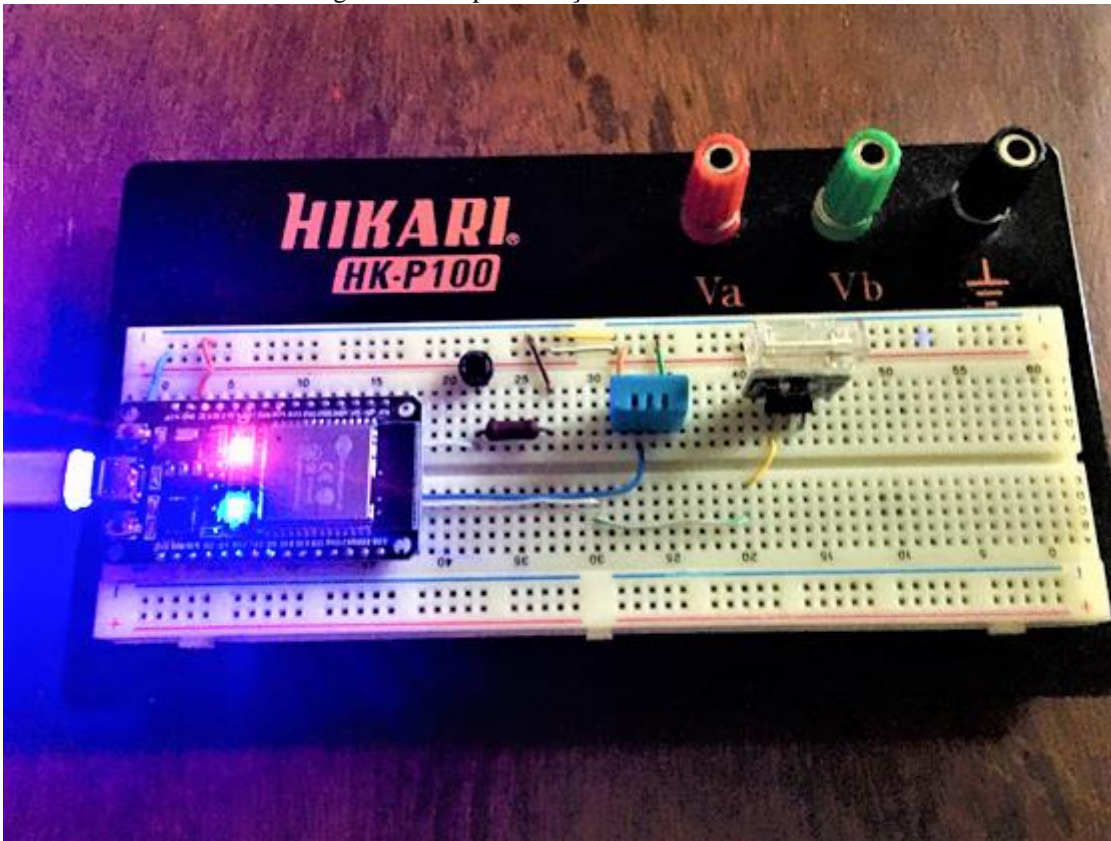
```
170 void loop()
171 {
172     char temperatura_str[10] = {0};
173     char umidade_str[10]     = {0};
174
175     /* garante funcionamento das conexões WiFi e ao broker MQTT */
176     VerificaConexoesWiFiMQTT();
177
178
179     while( i > 5000){
180
181         /* Compose as strings a serem enviadas pro dashboard (campos texto) */
182         sprintf(temperatura_str,"%fC", faz_leitura_temperatura());
183         sprintf(umidade_str,"%f", faz_leitura_umidade());
184
185         /* Envia as strings ao dashboard MQTT */
186         client.publish(TOPICO_PUBLISH_TEMPERATURA, temperatura_str);
187         client.publish(TOPICO_PUBLISH_UMIDADE, umidade_str);
```

Fonte: O próprio autor.

### 3.8. IMPLEMENTAÇÃO FÍSICA

Por fim, executou-se inicialmente a montagem em protoboard, conforme pode-se ver na Figura 20. O código é então compilado e carregado ao ESP, onde para realmente enviado, deve-se pressionar o botão “*Boot*” para ser efetivamente carregado. Após isso, o circuito começa a realizar as leituras de umidade e temperatura e iniciar a varredura nos pinos do botão de emergência e no sensor de colisão. Inicia-se o funcionamento do projeto propriamente dito.

Figura 21 - Implementação do Circuito em Protoboard.

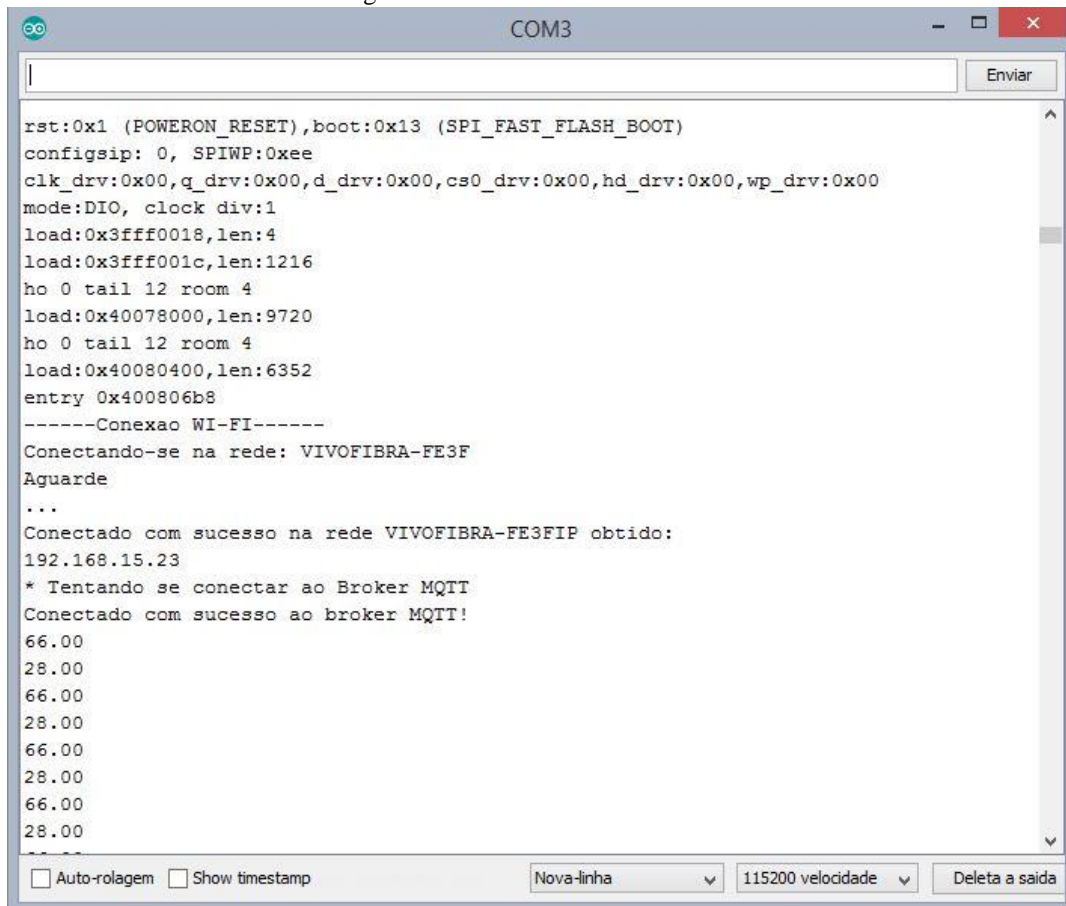


Fonte: O próprio autor.

#### 4. TESTES E RESULTADOS OBTIDOS

Para efeito de testes, selecionou-se a saída serial (*Serial.print*) para se plotar a saída Serial de temperatura e umidade dos sensores. Pode-se ver que todos os parâmetros inicialmente são zerados, assim como os pinos iniciam em nível baixo. Pode-se ver também a conexão ao Wi-Fi, onde é alocado um IP disponível dentro da rede para o ESP32. Após a conexão ao Wi-Fi temos a conexão ao *broker* MQTT. Por fim, após as conexões bem-sucedidas tanto no Wi-Fi quanto ao *broker*, as informações começam a ser enviadas para o broker, e temos a saída em temperatura e umidade. Na figura 23 temos a aplicação final sendo exibida pelo aplicativo, onde as informações de temperatura e umidade são atualizadas a cada 10 segundos.

Figura 22 - Monitor Serial do ESP32.



```
COM3
Enviar

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
-----Conexao WI-FI-----
Conectando-se na rede: VIVOFIBRA-FE3F
Aguarde
...
Conectado com sucesso na rede VIVOFIBRA-FE3FIP obtido:
192.168.15.23
* Tentando se conectar ao Broker MQTT
Conectado com sucesso ao broker MQTT!
66.00
28.00
66.00
28.00
66.00
28.00
66.00
28.00
66.00
28.00
...
 Auto-rolagem  Show timestamp
Nova-linha ▼ 115200 velocidade ▼ Deleta a saída
```

Fonte: O próprio autor.

No momento em que apareceu na tela, o sensor de colisão não havia sido acionado. Mas o botão de emergência foi acionado, e pode-se ver a saída da aplicação na figura 23:

Figura 23 – Aplicação final na tela de usuário.



Fonte: O próprio autor.

Por fim, verifica-se o funcionamento pleno nas informações de temperatura e umidade, porém o sensor de colisão e botão de emergência apresentam certa intermitência de funcionamento, funcionando bem ora sim ora não. Alguns momentos quando o botão é pressionado, o aplicativo tem uma resposta mais lenta na resposta. Mas no geral, tem-se uma resposta razoavelmente satisfatória, podendo enxergar o funcionamento do envio de mensagens no protocolo MQTT. Propõe-se a investigação para se verificar o mau funcionamento futuro, seja por falha de sensor, de lógica no código, ou de outro parâmetro qualquer.

## CONCLUSÃO

Este projeto propôs a elaboração de um protótipo capaz de realizar a verificação dos parâmetros ambientais em um leito hospitalar, otimizando a comunicação entre o paciente e o cuidador ou o paciente e a enfermaria. Aplicou-se os conhecimentos adquiridos durante as pesquisas executadas, juntamente com o conteúdo ministrado na graduação.

Apresentaram-se no conteúdo deste trabalho, o microcontrolador ESP32, bem com suas características e se percebe quão robusta é esta tecnologia, que apesar de nova, vem ganhando mercado mundo afora com soluções em IoT. Uma otimização alcançada foi a substituição do Arduino com o Raspberry Pi inicialmente propostos, pelo ESP32, sendo uma alternativa mais econômica e realizando o que duas plataformas realizariam, em apenas um microcontrolador.

Construiu-se o protótipo com os sensores de temperatura, umidade e colisão foram implementados, juntamente com o botão de emergência. Implementou-se o código fonte na IDE do Arduino e foi feita a compilação e envio do código. Pelo Monitor Serial consegue-se ver a conexão com Wi-Fi e com o servidor são bem-sucedidas. Começa então o envio, por meio do MQTT dos parâmetros coletados pelos sensores, e é visto na aplicação.

Pode-se verificar o funcionamento da comunicação entre o *broker* e o cliente, onde a informação na nuvem esteve presente na aplicação, ao usuário de saída. A resposta ao usuário foi relativamente satisfatória, onde o aplicativo utilizado suportou parcialmente o recebimento de informações do servidor.

Para desenvolvimento deste trabalho futuramente, propõe-se a implementação de um aplicativo próprio para o sistema *Android*. Propõe-se também como melhorias a criação de um ambiente gráfico, uma página na Web, onde se teria a informação impressa num navegador, bem como a disposição do protótipo numa placa de circuito impresso, para se ter um produto.

## REFERÊNCIAS BIBLIOGRÁFICAS

AGÊNCIA NACIONAL DE VIGILÂNCIA SANITÁRIA - ANVISA (2010). Manual de Tecnovigilância abordagens de vigilância sanitária de produtos para a saúde comercializados no Brasil. site: <http://tinyurl.com/manualTecnovigilancia>. Online; acessado 25-maio-2019;

ARDUINO (?). What is an Arduino. Disponível em <<https://www.arduino.cc/en/Guide/Introduction>>. Acessado em 25 de Maio de 2019;

ASTHON, Kevin "That 'Internet of Things' Thing". RFID Journal (1999);

APKPURE. MQTT Dash (IoT, Smart Home). 2019. Disponível em <<https://apkpure.com/br/mqtt-dash-iot-smart-home/net.routix.mqttdash>>. Acessado em 06 de dezembro de 2019;

CANTANHEDE, R. F., SILVA, C. E. Uma Proposta de Sistema de IoT para Monitoramento de Ambiente Hospitalar. Instituto Metrópole Digital - Universidade Federal do Rio Grande do Norte (UFRN), 2014;

CIRCUITS4YOU.COM. ESP32 DevKit ESP32-WROOM GPIO Pinout. 2018. Disponível em <<https://circuits4you.com/2018/12/31/esp32-devkit-esp32-wroom-gpio-pinout/>>. Acessado em 01 de dezembro de 2019;

CLOUDMQTT. Instances. 2019. Disponível em <<https://api.cloudmqtt.com/console/82646997/details>>. Acessado em 01 de dezembro de 2019;

ESPRESSIF. ESP32: A diferente Iot and Power Performance. 2018. Disponível em <<https://www.espressif.com/en/products/hardware/esp32/overview>> Acessado em 02 de dezembro de 2019;

FILIFELOP. Módulo WiFi ESP32 Bluetooth. (?). Disponível em <<https://www.filieflop.com/produto/modulo-wifi-esp32-bluetooth/>>. Acessado em 05 de dezembro de 2019;

GIL, Antônio Carlos. Como elaborar projetos de pesquisa. - 4. ed. São Paulo, Atlas, 2002;

HASSANALIERAGH, M., PAGE, A., SOYATA, T., et al., "Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges". In: Services Computing (SCC), 2015 IEEE International Conference on, pp. 285-292, IEEE, 2015;

INDIANMART. Dht11 Temperature and Humidity Sensor. 2019. Disponível em Disponível em <<https://www.vidadesilicio.com.br/sensor-de-vibracao-batidas-ky-031>> Acessado em 10 de dezembro de 2019;

KELLY, S. D. T., Towards the implementation of IoT for environmental condition monitoring in homes", IEEE Sensors Journal, v. 13, n. 10, pp. 3846-3853, 2013;

KONAYAGE, Fernando. Introdução ao ESP32. 2017. Disponível em <<https://www.fernandok.com/2017/11/introducao-ao-esp32.html>> Acessado em 10 de dezembro de 2019;

JUNIOR, Amauri Marcos Costa de Moraes. Sistema de monitoramento de temperatura e umidade relativa do ar – SIMTU-AR. Universidade Federal do Rio Grande do Norte, Natal, 2017;

LIBELIUM. 50 Sensor Applications for a Smarter World. 2019. Disponível em <[http://www.libelium.com/resources/top\\_50\\_iot\\_sensor\\_applications\\_ranking/#show\\_infographic](http://www.libelium.com/resources/top_50_iot_sensor_applications_ranking/#show_infographic)>. Acessado em 05 de dezembro de 2019;

LOUREIRO, A. F. et al. Redes de Sensores Sem Fio. Departamento de Ciências da Computação, Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais (2014);

MIKROKONTROLER PALEMBANG. KY-031 Knock Impact Sensor Module. 2016. Disponível em < [http://mikrokontrolerpalembang.blogspot.com/2016/11/ky-031-knock-impact-sensor-module\\_2.html](http://mikrokontrolerpalembang.blogspot.com/2016/11/ky-031-knock-impact-sensor-module_2.html)>. Acessado em 10 de dezembro de 2019;

PPLWARE MQTT: Protocolo de comunicação para pequenos dispositivos móveis. 2019. Disponível em < <https://pplware.sapo.pt/tutoriais/networking/mqtt-protocolo-de-comunicacao/>>. Acessado em 10 de dezembro de 2019;

PROTEUS 8 PROFESSIONAL. LabCenter Eletronics 1989-2018. Release 8.7 SP3 (Build 25561) with Advanced Simulation;

RASPBERRY PI FOUNDATION. What is a Raspberry Pi. Disponível em <<https://www.raspberrypi.org/help/videos/#what-is-a-raspberry-pi>>. Acessado em 25 de maio de 2019;

RODRIGUES, William Costa. Metodologia Científica. FAETEC/IST. Paracambi, Rio de Janeiro. 2007;

SANTOS, B. P., SILVA, A. M. L et al. Internet das Coisas: da Teoria à Prática. Universidade Federal de Minas Gerais, Belo Horizonte, Minas Gerais. (ca 2010);

WHITCHURCHEM, A. K. et al. Connected Health: Open source IoT patient monitor. Disponível em <<https://hackaday.io/project/25380-connected-health-open-source-iot-patient-monitor>>. Acessado em 25 de maio de 2019;

YUAN, Michael. Conhecendo o MQTT. 2017. Disponível em <<https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html> >. Acesso em 2 de dezembro de 2019;



## APÊNDICE – CÓDIGO FONTE

```
//Luís Antônio Valois Miranda ---> TCC2
//Criar ambiente de sensoriamento num leito hospitalar

#include <DHT.h>
#include <WiFi.h>
#include <PubSubClient.h>

#define DHTPIN      19      //GPIO que está ligado o pino de dados do sensor
#define botao      18 // PIN 34
#define LED        2
#define pinoSensor 21

//#define DHTTYPE DHT11
#define DHTTYPE DHT11 //sensor em utilização: DHT22

#define TOPICO_PUBLISH_TEMPERATURA "topico_sensor_temperatura"
#define TOPICO_PUBLISH_UMIDADE    "topico_sensor_umidade"
#define TOPICO                    "topico_led"
#define TOPICO_COLISAO             "topico_colisao"

/* Variaveis, constantes e objetos globais */
DHT dht(DHTPIN, DHTTYPE);

//informações da rede WIFI
const char* ssid = "iPhone de Luis"; //SSID da rede WIFI
const char* password = "nhhjhhjjj"; //senha da rede wifi

//informações do broker MQTT
const char* mqttServer = "tailor.cloudmqtt.com";//server
const char* mqttUser = "hyzshgjo"; //user
const char* mqttPassword = "0Ne-AvkK1xU5"; //password
const int mqttPort = 15753; //port
const char* mqttTopicSub ="LuisTCC"; //tópico que sera assinado

int valor;
int i=0;
int cont;
int j=1;

//Variáveis e objetos globais
WiFiClient espClient; // Cria o objeto espClient
PubSubClient client(espClient); // Instancia o Cliente MQTT passando o
objeto espClient

// Prototypes
float faz_leitura_temperatura(void);
float faz_leitura_umidade(void);
void initWiFi(void);
void initMQTT(void);
void reconnectMQTT(void);
void reconnectWiFi(void);
void VerificaConexoesWiFiMQTT(void);

//const char *TOPICO ="ledstatus";

//Leitura de Temperatura do Sensor
```

```

float faz_leitura_temperatura(void)
{
    float t = dht.readTemperature();
    float result;

    if (! (isnan(t)) )
        result = t;
    else
        result = -99.99;

    return result;
}

//Leitura de Humidade do Sensor
float faz_leitura_umidade(void)
{
    float h = dht.readHumidity();
    float result;

    if (! (isnan(h)) )
        result = h;
    else
        result = -99.99;

    return result;
}

void initWiFi(void)
{
    delay(10);
    Serial.println("-----Conexao WI-FI-----");
    Serial.print("Conectando-se na rede: ");
    Serial.println(ssid);
    Serial.println("Aguarde");

    reconnectWiFi();
}

void reconnectWiFi(void)
{
    //se já está conectado a rede WI-FI, nada é feito.
    //Caso contrário, são efetuadas tentativas de conexão
    if (WiFi.status() == WL_CONNECTED)
        return;

    WiFi.begin(ssid, password); // Conecta na rede WI-FI

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }

    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(ssid);
    Serial.println("IP obtido: ");
    Serial.println(WiFi.localIP());
}

```

```

void initMQTT(void)
{
    client.setServer(mqttServer, mqttPort); //informa qual broker e porta
    deve ser conectado
}

void reconnectMQTT(void)
{
    while (!client.connected())
    {
        Serial.println("* Tentando se conectar ao Broker MQTT");
        if (client.connect("8266Client", mqttUser, mqttPassword ))
        {
            Serial.println("Conectado com sucesso ao broker MQTT!");
        }
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}

void VerificaConexoesWiFIEMQTT(void)
{
    if (!client.connected())
        reconnectMQTT(); //se não há conexão com o Broker, a conexão é
refeita

        reconnectWiFi(); //se não há conexão com o WiFi, a conexão é refeita
}

void setup()
{
    Serial.begin(115200);

    // Inicializacao do sensor de temperatura
    dht.begin();

    // Inicializa a conexao wi-fi
    initWiFi();

    // Inicializa a conexao ao broker MQTT
    initMQTT();

    pinMode(botao, INPUT);
    pinMode(LED, OUTPUT);

    pinMode(pinoSensor, INPUT);
}

void loop()
{
    char temperatura_str[10] = {0};
    char umidade_str[10] = {0};

    /* garante funcionamento das conexões WiFi e ao broker MQTT */
    VerificaConexoesWiFIEMQTT();
}

```

```

while( i > 5000){

/* Compose as strings a serem enviadas pro dashboard (campos texto) */
sprintf(temperatura_str,"%0.2fC", faz_leitura_temperatura());
sprintf(umidade_str,"%0.2f", faz_leitura_umidade());

/* Envia as strings ao dashboard MQTT */
client.publish(TOPICO_PUBLISH_TEMPERATURA, temperatura_str);
client.publish(TOPICO_PUBLISH_UMIDADE, umidade_str);

/* keep-alive da comunicação com broker MQTT */
client.loop();

Serial.print(faz_leitura_umidade());
i=0;
}
i++;

// faz a leitura do pino D1 (no nosso caso, o botão está ligado nesse
pino)

valor = digitalRead(botao);

// checa se o botão está pressionado
if(valor == HIGH) {
j++;
cont = j%2;
}

if (cont==1){
digitalWrite(LED, HIGH);

client.publish(TOPICO, "ALERTA EMERGÊNCIA");}
else {
digitalWrite(LED, LOW); }

Serial.println(cont);
Serial.println(j);

if(digitalRead(pinoSensor) == HIGH){
Serial.println("Batida detectada");
client.publish(TOPICO_COLISAO, "ALERTA COLISAO");
}

}

```