

**UNIVERSIDADE FEDERAL FLUMINENSE**

**MARCELO MARQUES DA ROCHA**

**DESENVOLVIMENTO OPEN-SOURCE PARA  
A INTERNET DAS COISAS  
(ARQUITETURAS PARA INTERFACES WEB E MÓVEL)**

**Niterói**

**2018**

**MARCELO MARQUES DA ROCHA**

**DESENVOLVIMENTO OPEN-SOURCE PARA  
A INTERNET DAS COISAS  
(ARQUITETURAS PARA INTERFACES WEB E MÓVEL)**

Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Computação.

**Orientador:  
Luciano Bertini**

**NITERÓI  
2018**

Ficha catalográfica automática - SDC/BEE

R672d Rocha, Marcelo Marques da  
Desenvolvimento Open-Source para a Internet das Coisas  
(Arquiteturas para Interfaces Web e Móvel) / Marcelo Marques  
da Rocha ; Luciano Bertini, orientador. Niterói, 2018.  
148 f. : il.

Trabalho de Conclusão de Curso (Graduação em Tecnologia  
de Sistemas de Computação)-Universidade Federal Fluminense,  
Escola de Engenharia, Niterói, 2018.

1. Internet das Coisas. 2. Residências Inteligentes. 3.  
Interface Web e Móvel. 4. Sistemas Open-source. 5. Produção  
intelectual. I. Título II. Bertini, Luciano, orientador. III.  
Universidade Federal Fluminense. Escola de Engenharia.  
Departamento de Ciência da Computação.

CDD -

**MARCELO MARQUES DA ROCHA**

**DESENVOLVIMENTO OPEN-SOURCE PARA  
A INTERNET DAS COISAS  
(ARQUITETURAS PARA INTERFACES WEB E MÓVEL)**

Trabalho de Conclusão de Curso subme-  
tido ao Curso de Tecnologia em Siste-  
mas de Computação da Universidade  
Federal Fluminense como requisito par-  
cial para obtenção do título de Tecnólogo  
em Sistemas de Computação.

Niterói, \_\_\_\_ de \_\_\_\_\_ de 2018.

Banca Examinadora:

---

Prof. Luciano Bertini, D.Sc. – Orientador  
UFF – Universidade Federal Fluminense

---

Prof. Alessandro Copetti, D.Sc. – Avaliador  
UFF – Universidade Federal Fluminense

Dedico este trabalho aos meus pais (Luís e Vera) ao meu irmão Márcio (meu primeiro professor) à minha esposa Rachel e à minha filha Luiza.

## **AGRADECIMENTOS**

Ao meu Deus, que permitiu que eu chegasse até aqui. “Todas as coisas foram feitas por Ele, e sem Ele nada do que foi feito se fez”. (João 1:3).

A meu Orientador Luciano Bertini pelo estímulo e direcionamento durante a pesquisa neste trabalho.

À Marta Zanetti, coordenadora do pólo de Angra dos Reis, por toda ajuda desde o vestibular.

Aos Colegas de curso pelo incentivo e troca de experiências.

A todos os meus familiares e amigos pelo apoio e colaboração.

“O único aprendizado que influencia significativamente o comportamento é a aprendizagem autoapropriada e autodescoberta, verdade que foi assimilada na experiência”.

Carl R. Rogers (1969)

## **RESUMO**

A IoT (Internet das Coisas) é uma tendência tecnológica atual na qual tudo pode estar conectado na Internet, desde interruptores de luz, torradeiras, ou qualquer outro objeto. Essa tecnologia pretende mudar a vida das pessoas, suas casas, cidades e até a indústria. A Internet das coisas é vista como um componente fundamental para a 4<sup>a</sup> revolução industrial. Apesar das empresas privadas desenvolverem soluções para esta tecnologia, ela é dominada e impulsionada por tecnologias open-source. Como exemplo dessas tecnologias existem, na parte de hardware, o Arduino, o Raspberry Pi e o NodeMcu, que são placas de baixo custo que permitem a implementação da computação embarcada com a comunicação necessária para a IoT. Na parte de software existem o protocolo MQTT, o servidor Mosquitto (*MQTT broker*), a linguagem Lua e o Node-RED. Este trabalho tem como objetivo apresentar essas tecnologias. Pretende mostrar os protocolos usados no processo de comunicação entre os dispositivos. Também vai falar sobre a linguagem utilizada na criação dos firmwares e da ferramenta de criação da aplicação de controle de IoT web móvel. Mostrará algumas possibilidades de arquiteturas para Internet das coisas e por último será implementado um protótipo (hardware e software) de um sistema de uma Casa inteligente.

**Palavras-chaves:** Internet das Coisas, Casas inteligentes, MQTT, Mosquitto.

## **ABSTRACT**

IoT (Internet of Things) is a current technological trend in which everything can be connected in the Internet, from light switches, toasters, or any other object. This technology aims to change the lives of people, their homes, cities, and even the industry. The Internet of things is seen as a fundamental component of the 4th industrial revolution. Although private companies develop solutions for this technology, it is dominated and driven by open-source technologies. As an example of such technologies there are, in the hardware part, Arduino, Raspberry Pi, and NodeMcu, which are low cost boards that allow the implementation of embedded computing with the necessary communication for the IoT. In the software part, there are the MQTT protocol, the Mosquitto server (MQTT broker), the Lua language, and the Node-RED. This work aims to present these technologies. It aims to show the protocols used in the communication process between the devices, and also will talk about the language used in the creation of *firmwares*, and the creation tool for mobile web IoT control applications. It will show some possibilities for Internet of Things architectures, and finally, a hardware and software prototype of a Smart House will be implemented.

**Key words:** Internet of Things, Smart Houses, MQTT, Mosquitto.

## **LISTA DE CÓDIGOS-FONTE**

Código 1: Controlando o <i>led</i> com o Raspberry Pi usando Python.....	54
Código 2: Código do objeto msg.....	62
Código 3: Importação das bibliotecas para o ThingSpeak.....	70
Código 4: Configuração do canal ThingSpeak.....	70
Código 5: ThingSpeak - Inicialização.....	71
Código 6: Loop principal para leitura do sensor e publicação no ThingSpeak.....	72
Código 7: Definição dos pinos do DHT11.....	75
Código 8: Configuração do acesso à rede, host e campos do canal ThingSpeak....	76
Código 9: Configuração da velocidade das portas seriais.....	76
Código 10: Envio dos dados lidos do sensor DHT11 para o ThingSpeak.....	77
Código 11: Função que faz a leitura do sensor DHT11.....	78
Código 12: Função que envia os comandos AT para o Esp8266.....	78
Código 13: Importando a biblioteca WiFi para o Esp8266.....	84
Código 14: Configurando e conectando ao WiFi.....	86
Código 15: Função loop() - Leitura do sensor e envio dos dados para o T. Speak...	87
Código 16: Executando automaticamente com init.lua.....	89
Código 17: Módulo credenciais.lua.....	90
Código 18: Módulo tabela.lua.....	90
Código 19: Módulo conecta_wifi.lua.....	92
Código 20: Módulo thingspeak_http.lua - Declarando variáveis.....	93
Código 21: Configurando o timer para o envio de dados.....	94
Código 22: Postando no ThingSpeak - Cliente HTTP.....	94
Código 23: Parâmetros de conexão - MQTT <i>broker</i> .....	102
Código 24: Configurando os pinos do node-quarto.....	103
Código 25: Lendo o estado do sensor de movimento.....	105
Código 26: Criando o objeto cliente MQTT.....	106
Código 27: Conectando ao servidor MQTT parte 1.....	107
Código 28: Conectando ao servidor MQTT parte 2.....	108

Código 29: Conectando ao servidor MQTT parte 3.....	109
Código 30: Recebendo uma mensagem MQTT parte 1.....	110
Código 31: Recebendo uma mensagem MQTT parte 2.....	110
Código 32: Publicando mensagens MQTT.....	111
Código 33: Iniciando os serviços HTTP e Telnet.....	112
Código 34: Código fonte - Servidor HTTP parte 1.....	145
Código 35: Código fonte - Servidor HTTP parte 2.....	146
Código 36: Código fonte - Servidor HTTP parte 3.....	147
Código 37: Código fonte - Servidor Telnet.....	148

## LISTA DE ILUSTRAÇÕES

Figura 1: MQTT Publish / Subscribe.....	27
Figura 2: A arquitetura do MQTT-SN.....	32
Figura 3: IDE do Arduino - Código Blink.....	36
Figura 4: Arduino YÚN.....	37
Figura 5: Família Esp8266.....	38
Figura 6: Tamanho do Esp01.....	39
Figura 7: Versão do <i>firmware</i> utilizada no Esp8266.....	41
Figura 8: Descrição dos pinos do NodeMcu.....	43
Figura 9: Lista de módulos disponíveis para o <i>firmware</i> Lua.....	45
Figura 10: Raspberry Pi - Model B.....	50
Figura 11: Raspberry Pi 1 model B - Pinos de GPIO.....	51
Figura 12: Nomenclatura dos pinos GPIO - <i>BOARD</i> e <i>BCM</i> .....	52
Figura 13: Circuito de exemplo.....	52
Figura 14: Esquema do Raspberry PI.....	53
Figura 15: Node-RED - Nodes.....	61
Figura 16: Flow - Controlando o <i>led</i> através do Twitter.....	63
Figura 17: Node twitter-in - Configuração.....	64
Figura 18: Configuração - rpi-gpio out.....	64
Figura 19: Ethernet Shield W5100.....	65
Figura 20: Sensor LM35.....	66
Figura 21: Esquema eletrônico - Arduino/Ethernet/LM35.....	66
Figura 22: LM35 - Modo 1.....	67
Figura 23: LM35 - Modo 2.....	67
Figura 24: Criando um canal no ThingSpeak.....	68
Figura 25: Configuração do canal ThingSpeak.....	69
Figura 26: Channel ID e Write API Key.....	69
Figura 27: Instalando a biblioteca ThingSpeak na IDE do Arduino.....	70
Figura 28: Sensor DHT11 e seus pinos.....	73

Figura 29: Esquema eletrônico - Arduino / DHT11 / Esp8266 (AT).....	74
Figura 30: Monitor serial - Comandos AT.....	79
Figura 31: Configurando o Esp8266 na IDE do Arduino.....	80
Figura 32: Instalando a placa Esp8266 na IDE do Arduino.....	81
Figura 33: Selecionando a placa Esp8266.....	81
Figura 34: Selecionado o modelo do Esp8266 correto.....	82
Figura 35: Selecionado a porta do Esp8266.....	83
Figura 36: Esquema eletrônico - Esp8266/DHT11 - Programando em C++.....	84
Figura 37: Casa Inteligente - Cenário.....	96
Figura 38: Servidor Raspberry Pi - IP estático.....	97
Figura 39: Configurando o DMZ.....	98
Figura 40: Configurando o DNS dinâmico no roteador.....	98
Figura 41: Tipos de nós (NodeMcu) da rede.....	99
Figura 42: Esquema eletrônico do NodeMcu - Tipo 1.....	100
Figura 43: Esquema eletrônico do NodeMcu - Tipo 2.....	100
Figura 44: Esquema eletrônico do NodeMcu - Tipo 3.....	101
Figura 45: Compilando código Lua com o auxílio da IDE Esplorer.....	104
Figura 46: Acessando o servidor HTTP do nó NodeMcu.....	112
Figura 47: Saída do servidor Telnet.....	113
Figura 48: Interface gráfica da aplicação (Node-RED) - Visão dos controles.....	114
Figura 49: Menu da aplicação no NodeRED.....	115
Figura 50: Seção de controle da Iluminação da sala.....	116
Figura 51: Ícone do Android.....	116
Figura 52: Seção de controle do ar-condicionado central.....	116
Figura 53: Seção de controle do Umidificador de ar.....	117
Figura 54: Seção de gráficos - Temperatura, umidade e nível da cisterna.....	118
Figura 55: Interface gráfica - Modo de operação.....	119
Figura 56: Interface gráfica da aplicação Node-RED - Rede de nós NodeMcu.....	119
Figura 57: Esquema do Node-RED - Conexão com a cisterna.....	120
Figura 58: Configuração do function-node Switch.....	121
Figura 59: Tweet de aviso de cisterna em nível crítico.....	122
Figura 60: Esquema do Node-RED - Publicando em tópicos.....	122
Figura 61: Node-RED (Flow) - node-sala.....	124
Figura 62: Raspberry Pi - Servidor MQTT.....	125

Figura 63: Protótipo dos nós NodeMcu do tipo 1.....	125
Figura 64: Protótipo do nó NodeMcu do tipo 2.....	126
Figura 65: Protótipo do nó NodeMcu do tipo 3.....	126
Figura 66: Protótipo completo da rede de nós NodeMcu.....	127
Figura 67: Foto da aplicação rodando no Tablet Samsung.....	127
Figura 68: Foto da aplicação rodando em um celular modelo LG X cam.....	128
Figura 69: Verificando a porta associada ao Esp8266.....	135
Figura 70: Progresso do upload do <i>firmware</i> para a placa.....	136
Figura 71: NodeMcu - PyFlasher.....	137
Figura 72: Conectando o NodeMcu à IDE Esplorer.....	138
Figura 73: Conexão da IDE com o NodeMcu.....	138
Figura 74: IDE Esplorer - Funções.....	139
Figura 75: IDE Esplorer - Aba AT-based.....	140
Figura 76: Verificando o status do servidor Mosquitto.....	141
Figura 77: Criando um tópico de teste no Mosquitto.....	142
Figura 78: Publicação de teste no servidor Mosquitto.....	142
Figura 79: Tela de log do serviço Node-RED.....	144

## **LISTA DE TABELAS**

Tabela 1: Arduinos - Especificações técnicas.....	36
Tabela 2: Lista de SDKs para Esp8266.....	39
Tabela 3: Lista de comandos AT .....	41
Tabela 4: Relação das GPIO com os pinos do Esp8266.....	85

## **LISTA DE GRÁFICOS**

Gráfico 3.1: Linguagens de <i>script</i> mais usadas em jogos.....	48
Gráfico 3.2: Plotando gráficos usando o MATLAB.....	57
Gráfico 3.3: Gráfico da temperatura - LM35 - ThingSpeak.....	72
Gráfico 3.4: Gráficos de temperatura e umidade - ThingSpeak (Comandos AT).....	79
Gráfico 3.5: Leitura do DHT - Esp8266 programado em C++.....	87

# SUMÁRIO

<u>RESUMO</u> .....	8
<u>ABSTRACT</u> .....	9
<u>LISTA DE CÓDIGOS-FONTE</u> .....	10
<u>LISTA DE ILUSTRAÇÕES</u> .....	12
<u>LISTA DE TABELAS</u> .....	15
<u>LISTA DE GRÁFICOS</u> .....	16
<u>1</u> <u>Introdução</u> .....	19
<u>2</u> <u>Fundamentação Teórica</u> .....	23
<u>2.1</u> <u>IoT (Internet Of Things)</u> .....	23
<u>2.1.1</u> <u>Possíveis cenários com a IoT</u> .....	24
<u>2.2</u> <u>MQTT</u> .....	25
<u>2.2.1</u> <u>O Padrão Publish/Subscribe</u> .....	26
<u>2.2.2</u> <u>Tópicos/Aassinaturas</u> .....	28
<u>2.2.3</u> <u>Qualidade de serviço</u> .....	29
<u>2.2.4</u> <u>Retenção de mensagens</u> .....	30
<u>2.2.5</u> <u>Sessão limpa / Conexões duradouras</u> .....	30
<u>2.2.6</u> <u>Wills (Testamento)</u> .....	31
<u>2.3</u> <u>MQTT-SN (MQTT for Sensor Networks)</u> .....	31
<u>2.3.1</u> <u>Arquitetura do MQTT-SN</u> .....	31
<u>2.4</u> <u>Mosquitto (MQTT Broker)</u> .....	33
<u>3</u> <u>Componentes e Arquiteturas</u> .....	35
<u>3.1</u> <u>Componentes (Hardware e software)</u> .....	35
<u>3.1.1</u> <u>Arduino</u> .....	35
<u>3.1.2</u> <u>Esp8266</u> .....	37
<u>3.1.2.1</u> <u>Comandos AT</u> .....	40
<u>3.1.2.2</u> <u>NodeMcu (Firmware Lua)</u> .....	43
<u>3.1.2.3</u> <u>Linguagem Lua</u> .....	46

<u>    3.1.2.3.1 eLua (Embedded Lua).....</u>	48
<u>3.1.3 Raspberry Pi.....</u>	50
<u>3.1.4 ThingSpeak (Uma plataforma open source para IoT).....</u>	55
<u>    3.1.4.1 Coleta de dados.....</u>	56
<u>    3.1.4.2 Análise dos dados.....</u>	57
<u>    3.1.4.3 Associe uma ação (Trigger an action).....</u>	58
<u>3.1.5 Node-RED.....</u>	58
<u>3.2 Arquiteturas.....</u>	65
<u>    3.2.1 Arduino com shield ethernet.....</u>	65
<u>    3.2.2 Arduino com WiFi via Esp8266 (comandos AT).....</u>	72
<u>    3.2.3 Esp8266 (programando o <i>firmware</i> com a IDE do Arduino).....</u>	80
<u>    3.2.4 NodeMcu + ThingSpeak (usando Lua).....</u>	88
<u>4 Implementando uma Casa Inteligente.....</u>	95
<u>    4.1 Preparando a infraestrutura da rede.....</u>	97
<u>    4.2 Os esquemas eletrônicos dos nós NodeMcu.....</u>	99
<u>    4.3 Analisando o funcionamento do código-fonte.....</u>	101
<u>    4.4 Construindo a aplicação móvel com o Node-RED.....</u>	113
<u>    4.5 Protótipo construído.....</u>	125
<u>CONCLUSÕES E TRABALHOS FUTUROS.....</u>	129
<u>REFERÊNCIAS BIBLIOGRÁFICAS.....</u>	131
<u>APÊNDICE A – Esp8266 (fazendo o upload do <i>firmware</i> Lua).....</u>	134
<u>APÊNDICE B – Usando a IDE Esplorer.....</u>	138
<u>APÊNDICE C – Instalando e configurando o servidor Mosquitto.....</u>	141
<u>APÊNDICE D – Instalando o Node-RED.....</u>	143
<u>APÊNDICE E – Código do servidor HTTP.....</u>	145
<u>APÊNDICE F – Código do servidor Telnet.....</u>	148

# 1 INTRODUÇÃO

Este trabalho apresenta um estudo de tecnologias para a IoT (Internet das Coisas) que consiste na integração de tudo e qualquer coisa através da Internet. Mas antes de dar detalhes sobre essas tecnologias é preciso definir o que seria Internet das Coisas. O termo vem de um conceito bem mais antigo e amplo, o da computação ubíqua<sup>1</sup>.

Segundo Silva et al (2015), o termo ubiquidade tem relação direta com o significado da palavra onipresente, que quer dizer: algo que está em todo lugar. Apesar de onipresente essa tecnologia deve ser praticamente imperceptível, ou seja, deve passar despercebida no nosso cotidiano. As características de um cenário ubíquo são:

- Invisibilidade
- Pró-atividade
- Sensibilidade ao contexto
- Interfaces naturais
- Descentralização

A IoT é um termo moderno e que aparece como a possibilidade de implementação das ideias da computação ubíqua. Um cenário de IoT traz consigo muitas das características citadas acima. Das casas inteligentes até dispositivos vestíveis (*wearables*),<sup>2</sup> a IoT está presente na vida das pessoas.

São muitas as tecnologias que estão surgindo para atender o mercado gerado pela IoT e a maioria delas vêm do mundo open-source. Desde o hardware necessário para se implantar um ambiente de IoT até o software que embarca esse hardware com inteligência, desde os protocolos de comunicação entre esses dispositivos até interfaces web móveis, tudo isso é fornecido e impulsionado pela filosofia open-source.

O objetivo deste trabalho é revelar a existência destes componentes de hardware e software, suas definições e aplicações, e como eles podem ser interconectados formando arquiteturas para a construção de produtos de IoT.

---

<sup>1</sup> Onipresente.

<sup>2</sup> Vestíveis.

Toda a ideia do trabalho começa na apresentação do protocolo de comunicação utilizado pelos dispositivos com o intuito de trocar mensagens. O protocolo responsável por essa troca é o MQTT. Ele é um protocolo leve, ideal para dispositivos com limitações de processamento, memória e para redes com pequena largura de banda. O MQTT usa o modelo *publish/subscribe*<sup>3</sup>, que permite que dois dispositivos troquem mensagens sem mesmo saber um da existência do outro. Uma das características do modelo *pub/sub* é o desacoplamento, que possibilita que dois dispositivos troquem mensagens, independente do tempo e do espaço. Para que o MQTT funcione entra em cena o servidor Mosquitto, que é também chamado de MQTT *broker*.

Este trabalho é construído em cima dos seguintes dispositivos de hardware:

- **Arduino** – Hardware open-source que pode se conectar com sensores, atuadores, *shields*<sup>4</sup> de rede com fio e sem fio. Serão apresentadas ao todo cinco arquiteturas, e o Arduino será usado nas duas primeiras, onde será também o centro do processamento.
- **Esp8266** – Dispositivo que serve como *shield WiFi* para Arduino mas também pode funcionar de maneira independente. Possui portas para a conexão com sensores e já vem com a capacidade de se conectar às redes *wireless*.
- **Raspberry Pi** – É um computador completo capaz de funcionar com vários sistemas operacionais. Entre eles a distribuição GNU/Linux Raspbian que é uma versão da distribuição Debian otimizada para o Pi. Também pode se conectar a sensores e atuadores. Neste trabalho ele é usado como um servidor rodando o Mosquitto e o Node-RED.
- **NodeMcu** – É um kit de desenvolvimento pra protótipos de IoT. Usa uma linguagem de programação chamada Lua que permite que com poucas linhas de código seja criada uma aplicação de IoT.

Na parte de software são abordados os comandos AT, que servem para a comunicação entre o Arduino e o Esp8266, e são mostrados exemplos de códigos escritos nas linguagens C++ e Lua. Para encerrar a parte de software, há uma descrição do Node-RED e de seus principais tipos de componentes. O Node-RED é uma linguagem visual baseada em fluxos (*flows*) criada pela IBM a fim de facilitar a criação de aplicações IoT.

---

<sup>3</sup> Publica/Assina.

<sup>4</sup> Placas que adicionam novas funcionalidades ao Arduino.

Na parte das arquiteturas, após a apresentação de todos esses conceitos de hardware e software, são apresentadas cinco possibilidades de se construir aplicações de IoT. Cada arquitetura é uma combinação progressiva desses componentes e para cada uma delas, o trabalho mostra os esquemas eletrônicos em *protoboard*<sup>5</sup> e traz o código fonte explicado detalhadamente.

A quinta arquitetura é a implementação de um protótipo de uma Casa Inteligente. É nesse ponto do trabalho, que tudo o que foi abordado anteriormente, converge para a construção da aplicação web e dos códigos dos *firmwares*<sup>6</sup> dos dispositivos. Todos os esquemas eletrônicos são mostrados em detalhes e os fontes dos *firmwares* dissecados linha por linha.

A aplicação da Casa Inteligente implementada faz uso de 6 nós NodeMcu, esses nós são posicionados nos diversos cômodos da casa. Há um NodeMcu na sala, no quarto, no banheiro, na garagem, na cozinha e na cisterna. Todos eles, com exceção do que fica na cisterna, têm a capacidade de detectar movimento e controlar a iluminação do ambiente.

A casa é monitorada/controlada pelos nós NodeMcu. A Temperatura e umidade do ar são aferidas e seus valores visualizados através de gráficos na aplicação. É possível automatizar o funcionamento do ar-condicionado e do umidificador baseado em valores determinados na interface da aplicação. O nível de água da cisterna é monitorado constantemente e se esse nível baixar de um valor especificado, a aplicação twitta o aviso de alerta. Há um gauge<sup>7</sup> que mostra a quantidade de água disponível.

Os nós detectam o movimento e notificam o usuário. Há duas opções globais na aplicação que são importantes, a primeira permite que em caso de movimento detectado um e-mail seja enviado para o usuário informando em que cômodo foi detectada a presença, a segunda opção, é uma opção que foca na acessibilidade, ela permite que os avisos de detecção de movimento e o aviso da cisterna em nível crítico sejam falados através de um sintetizador de voz.

É possível simular o controle do ar-condicionado e do umidificador de ar através de dois *leds*, também é possível simular o controle da iluminação dos cômodos através de um terceiro *led*. Toda a implementação necessária para esses controles, foi realizada. So-

---

<sup>5</sup> Uma placa de ensaio ou matriz de contato. É uma placa com furos e conexões condutoras para montagem de circuitos elétricos experimentais.

<sup>6</sup> Software responsável pelo funcionamento do dispositivo.

<sup>7</sup> Instrumento de medição.

mente não foi instalada completamente em uma casa devido a dificuldades práticas que não são escopo deste trabalho, como a passagem de cabos por conduítes por exemplo.

Há uma seção que analisa os códigos-fonte dos *firmwares* e a partir desse detalhamento são abordados conceitos importantes sobre como é feita a conexão ao ThingSpeak, que é uma plataforma open-source para IoT e que permite coletar, analisar e atuar sobre os dados vindos dos dispositivos. Uma outra seção vai mostrar como fazer a conexão com o MQTT *broker* e como se publica e assina um tópico. Também serão vistos alguns detalhes importantes sobre programação Lua para o NodeMcu, como o uso dos timers, a importância da indentação na organização do código Lua e o uso das funções *callback*. Será explicado também todo o processo de configuração da infraestrutura de rede necessária para o funcionamento do projeto e como torná-lo acessível via Internet. Também poderá ser visto como funciona a interface web (Node-RED) e como conectar seus componentes. Este trabalho foi subdividido nos seguintes capítulos:

O capítulo 2 vai apresentar o embasamento teórico sobre o protocolo de comunicação e o uso do servidor de mensagens.

O capítulo 3 vai mostrar os componentes e suas arquiteturas. Todo hardware e software necessário para a implementação de uma Casa Inteligente será apresentado neste capítulo.

O capítulo 4 será a aplicação de todos os conceitos e ferramentas vistos nos capítulos anteriores. Será mostrado todo o processo de configuração, programação e construção do protótipo de uma Casa Inteligente. Ao final podem ser vistas algumas fotos do protótipo (6 nós NodeMcu) montado em bancada.

O trabalho conta ainda com seis apêndices que trazem detalhes técnicos sobre a instalação e uso das ferramentas: esptool.py, Pyflasher, Esplorer. Eles explicam como fazer a gravação dos *firmwares* Lua e AT, e, por fim, trazem os códigos-fonte dos servidores HTTP e Telnet que rodam em alguns dos dispositivos.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresentará conceitos importantes para o desenvolvimento de aplicações de IoT. Ele começa com uma definição do que vem a ser a IoT e seu impacto na sociedade e na indústria, também apresenta um dos protocolos de comunicação mais indicados para utilização em projetos de Internet das coisas.

### 2.1 IOT (INTERNET OF THINGS)

É muito fácil ter um panorama do impacto da tecnologia sobre o mundo atual. A roda tornou possível mover e transportar coisas e pessoas. Ela mudou tudo desde a agricultura até as políticas governamentais. Casas e empresas foram iluminadas por lâmpadas – mudando completamente o jeito como arquitetos projetam estruturas e cidades inteiras são estruturadas. O automóvel trouxe uma rápida viagem de um ponto a outro para as pessoas – redefinindo a maneira como as pessoas vivem e trabalham. O computador introduziu um mundo digital com dados que podem ser armazenados e compartilhados de uma maneira nova e fora do comum.

O entendimento do que significa “Internet das coisas”<sup>8</sup> é fácil, todos sabem o que é a Internet e também o que são “coisas”. Podemos então dizer que “Internet das coisas” são coisas na Internet. O termo Internet das coisas possui uma relação com um termo mais antigo chamado computação ubíqua, ou onipresente. Segundo Silva et al (2015), o termo ubiquidade tem relação direta com o significado da palavra onipresente, que quer dizer: algo que está em todo lugar.

A Internet das coisas está estreitamente relacionada a valor, conhecimento e oportunidades que surgem com a interconexão do mundo digital e físico. Dependendo da situação, “valor” pode ter vários significados. Se em uma empresa de entregas é preciso monitorar remotamente as frotas de veículos, o “valor” obtido com a IoT seria uma possível otimização.

---

<sup>8</sup> Segundo a Wikipédia Internet das coisas (IoT) é uma rede de objetos físicos ou “coisas” embarcadas com eletrônica, software, sensores, e conectividade para que essa seja capaz de alcançar um grande valor e serviço, trocando dados com o fabricante (inventor, produtor), operador, e/ou outros dispositivos conectados.

mização da rota, melhor aproveitamento do combustível ou maior eficiência no cumprimento dos horários de entrega. Um outro exemplo de “valor” obtido com a IoT seria na área da saúde, imaginando-se a necessidade de se monitorar o estado de saúde de um paciente ou mesmo de um parente. Neste caso, o “valor” significaria manter a saúde e até mesmo salvar a vida de uma pessoa.

O termo Internet das Coisas foi criado em 1999 por Kevin Ashton e desde então tem sido atribuído a vários jargões tecnológicos. Ao longo dos anos, tem-se ouvido os termos “Internet das Coisas”, “Internet de Tudo”, “M2M” (Máquina-para-Máquina), “Computação Física”, “Computação Ubíqua”, e assim por diante. (DARNELL, 2015)

Independente de como o termo pode ser rotulado, a Internet das coisas é a prática de conectar e monitorar “coisas” ou dispositivos de maneira remota e autônoma. Visa a integração dos dispositivos físicos com a Internet, a fim de coletar dados trocados pelos dispositivos, analisar esses dados, extraer conhecimento a respeito do ambiente onde esses dispositivos estão posicionados e atuar sobre eles.

### 2.1.1 Possíveis cenários com a IoT

A IoT pode aparecer no dia a dia de diversas maneiras e em diversas áreas. Sua presença pode ser percebida em cenários futurísticos da vida cotidiana mas existem também outras possibilidades para IoT na indústria, assistência a saúde e etc.

O relógio despertador toca, ao acordar a pessoa percebe que está a 10 minutos atrasada. O relógio checou o horário de saída dos trens na Internet e viu que o trem atrasaria 10 minutos. Com isso a pessoa pôde ter uns minutos a mais de descanso.

Na cozinha, uma luz piscando faz lembrar que é a hora de tomar os comprimidos. Se o paciente esquece, a tampa do frasco de remédio fica online e envia um e-mail para o médico para que ele fique ciente.

Ao se preparar para sair de casa, percebe-se um brilho no canto dos olhos. O cabo do guarda-chuvas está aceso, que significa que ele checou o relatório do tempo que prevê que vai chover.

Como se pode ver, há inúmeros exemplos de Internet das Coisas, e o que é comum em todos os cenários é o uso da Internet para enviar, receber, ou comunicar informação. Em cada caso, o dispositivo que estava conectado à Internet não era um computador, tablet, ou telefone móvel mas um objeto, uma coisa.

Pessoas fazem uso da Internet grandemente através da *World Wide Web*, que é um conjunto de aplicações que rodam na Internet. Na Internet das Coisas, em comparação, dispositivos eletrônicos autônomos trocam informação uns com os outros sobre a Internet. Mas estes dispositivos não tem uma aplicação equivalente a um navegador web para usar. Estas ferramentas e serviços estão ainda no início do seu desenvolvimento. A comunicação entre estes objetos é feita usando a pilha de protocolos TCP/IP (que é o coração da Internet) e alguns protocolos mais específicos, que funcionam na camada de aplicação da pilha TCP/IP. É possível citar o protocolo HTTP que é a base do modelo cliente-servidor usado para Web e também o protocolo MQTT que é leve, eficiente e ideal para conectar pequenos dispositivos a redes restritas.

## 2.2 MQTT

O MQTT<sup>9</sup> é um protocolo de transporte de mensagens do tipo *publish/subscribe*. É aberto, leve, simples e projetado para ser de fácil implementação. É ideal para sensores e pequenos dispositivos móveis, é otimizado para redes TCP/IP não confiáveis ou de alta latência. Essas características o torna perfeito para uso em muitas situações, incluindo ambientes restritos, como por exemplo, comunicação de Máquina para Máquina (M2M) e Internet das coisas (IoT). Há também o MQTT-SN, uma variação do protocolo destinada a redes que não sejam baseadas em TCP/IP, como ZigBee<sup>10</sup>. Os termos *publish/subscribe* serão explicados na seção 2.2.1

MQTT minimiza a largura de banda usada e requerida para a comunicação; ao mesmo tempo, uma alta confiabilidade é alcançada para a transmissão dos dados. Estes requisitos são especialmente predominantes em redes de sensores, *Machine-to-Machine* (M2M), telemedicina, monitoramento de pacientes e Internet das coisas. Nestas aplicações, os dispositivos conectados estão “sempre ligados” e comunicam-se constantemente uns com os outros.

Desde 2013, o OASIS padronizou o MQTT como o protocolo para Internet das coisas. O OASIS é um consórcio sem fins lucrativos que dirige o desenvolvimento, conver-

---

<sup>9</sup> Message Queue Telemetry Transport – Transporte de Telemetria da Fila de Mensagens.

<sup>10</sup> Zigbee designa um conjunto de especificações para a comunicação sem-fio entre dispositivos eletrônicos, com ênfase na baixa potência de operação, na baixa taxa de transmissão de dados e no baixo custo de implementação.

gência e adoção de padrões abertos. O OASIS MQTT Technical Committee está produzindo um padrão para o MQTT compatível com MQTT V3.1, unido a requisitos para aprimoramentos, exemplos de uso documentados, melhores práticas e orientação para uso dos tópicos MQTT com mecanismos de registro e descoberta comumente disponíveis. O padrão suporta mensagens bidirecionais para lidar uniformemente com sinais e comandos, entrega de mensagens determinísticas, níveis básicos de QoS, cenários sempre/às vezes conectados, baixo acoplamento e escalabilidade para suportar um grande número de dispositivos. Os candidatos para aprimoramentos incluem prioridade e expiração de mensagem, tipo da carga útil da mensagem, solicitação/resposta e expiração de assinatura.

Outro aspecto importante é que o MQTT é extremamente fácil de implementar no lado do cliente. Isso se adapta perfeitamente aos dispositivos restritos com recursos limitados, como os microcontroladores. Na verdade, este foi um dos principais objetivos quando o MQTT foi inventado.

O MQTT foi inventado por Andy Stanford-Clark (IBM) e Arlen Nipper (Arcom, agora Cirrus Link) em 1999, quando o objetivo deles era criar um protocolo para conectar oleodutos, em uma conexão via satélite, que tivesse o mínimo de consumo de bateria e que usasse uma mínima largura de banda. (HIVEMQ, 2018)

Eles especificaram os seguintes objetivos, que o futuro protocolo deveria ter:

- Implementação simples
- Serviço de entrega de dados de qualidade
- Leve e com largura de banda eficiente
- Dados agnósticos
- Consciência de sessão contínua

Essas metas ainda são o núcleo do MQTT. Apenas o foco mudou, de sistemas embarcados proprietários para a Internet de coisas.

### 2.2.1 O Padrão Publish/Subscribe

O padrão *publish/subscribe* (publica/assina) é bem diferente do modelo cliente-servidor. No modelo cliente-servidor, um cliente se conecta diretamente com outro nó da rede. No modelo pub/sub, quem envia uma mensagem é chamado de *publisher* e quem

recebe uma mensagem é chamado de *subscriber*, nesse modelo, *publisher* e *subscriber* não sabem da existência um do outro, há um desacoplamento. Essa “ponte” é feita por um terceiro componente chamado *broker*, ele é um programa intermediário que direciona cada publicação para o nó interessado na mensagem. O *broker* conhece ambos, o *publisher* e o *subscriber*. Um exemplo de MQTT *broker* é o Mosquitto, um *message-broker* de código aberto que implementa o protocolo MQTT. A Figura 1 mostra como funciona a interação entre o *publisher*, o *subscriber* e o *broker*.

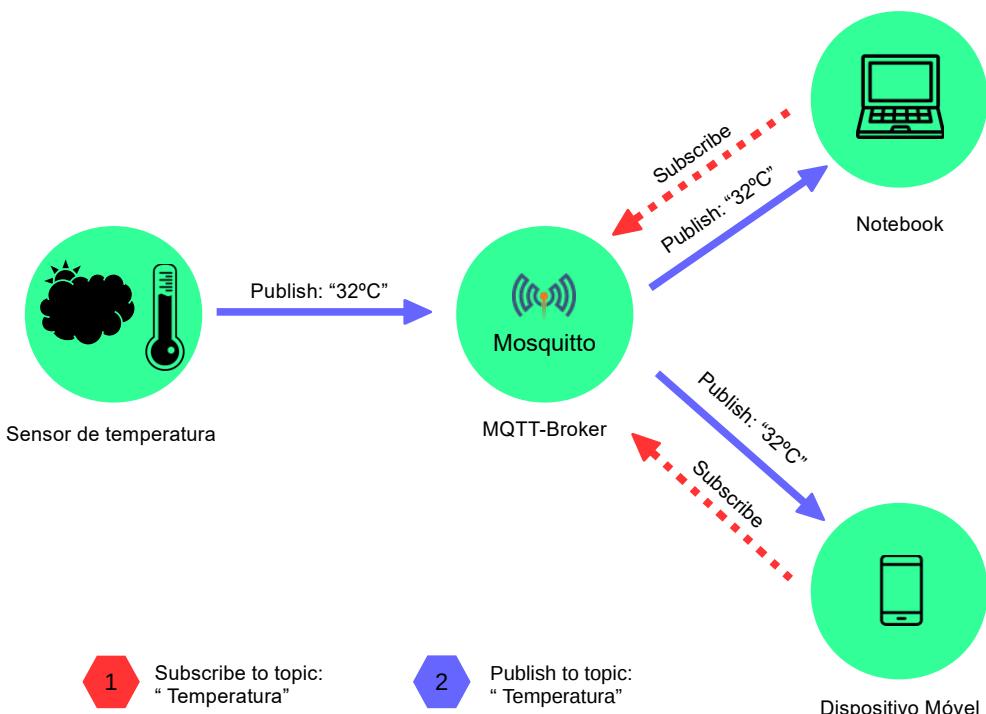


Figura 1: MQTT Publish / Subscribe  
Fonte: (HIVEMQ, 2018)

O principal aspecto no modelo *pub/sub* é o desacoplamento do *publisher* e do *subscriber*, que pode ser diferenciado três dimensões: Espaço, Tempo e Sincronização.

- **Desacoplamento espacial:** O *Publisher* e *subscriber* não precisam se conhecer, por endereço de ip ou porta. Só precisam estar conectados ao *broker*.
- **Desacoplamento temporal:** O *Publisher* e *subscriber* não precisam rodar simultaneamente.
- **Desacoplamento de sincronização:** As operações em ambos os componentes não são interrompidas durante a publicação ou recepção.

Através de um mecanismo de filtragem, é possível definir que certas mensagens sejam somente entregues a clientes específicos. O desacoplamento é a característica mais importante do modelo pub/sub.

## 2.2.2 Tópicos/Assinaturas

Na prática, mensagens em MQTT são publicadas em tópicos. E não há a necessidade de configurar qualquer tópico, basta publicar nele que já é o suficiente. Tópicos são tratados como uma hierarquia, usando uma barra (/) como separador. É semelhante a um sistema de arquivos. Por exemplo, vários computadores podem publicar suas temperaturas dos seus discos rígidos no seguinte tópico, com seu próprio nome do computador e o nome do seu disco rígido.

- sensores/NOME\_COMPUTADOR/temperatura/DISCO\_NOME

Clientes podem receber mensagens assinando aos tópicos. Uma assinatura pode ser feita a um tópico explícito, neste caso somente as mensagens para aquele tópico serão recebidas, ou a assinatura pode ser feita usando wildcards<sup>11</sup> (coringas). Dois coringas estão disponíveis, + ou #.

O + pode ser usado como um coringa para um nível simples da hierarquia. Ele pode ser usado com o tópico acima para obter informações sobre todos os computadores e hds como segue:

- sensores+/temperatura/+

O # pode ser usado como um coringa para todos os níveis restantes da hierarquia. Isto significa que ele deve ser o último caractere em uma assinatura. Com um tópico do tipo “a/b/c/d”, as seguintes assinaturas são válidas:

- a/b/c/d

---

<sup>11</sup> Wildcards são símbolos usados para substituir ou representar um ou mais caracteres.

- #
- a/#
- a/b/#
- a/b/c/#
- +/b/c/#

Os níveis de tópicos de comprimento zero são válidos, o que pode levar a um comportamento um pouco incompreensível. Por exemplo, um tópico de “a// tópico” concordaria corretamente com uma assinatura de “a/+/*tópico*”. Da mesma forma, níveis de tópicos de comprimento zero podem existir no início e no final de uma sequência de tópicos, então “/a/tópico” combinaria com uma assinatura de “+/a/ tópico”, “#” ou “/#”, e um tópico “a/tópico/” corresponderia a uma assinatura de “a/tópico/+” ou “a/tópico/#”.

### 2.2.3 Qualidade de serviço

O protocolo MQTT define três níveis de qualidade de serviço (QoS). O QoS define o quanto rígido, o *broker/client*, será na tentativa de garantir que a mensagem seja recebida. Mensagens podem ser enviadas em qualquer nível de QoS, e clientes podem tentar assinar aos tópicos em qualquer nível de QoS. Isto significa que o cliente escolhe o máximo QoS em que ele receberá uma mensagem. Por exemplo, se uma mensagem é publicada em QoS 2 e um cliente assinou o tópico com um QoS 0, a mensagem será entregue ao cliente com QoS 0. Se um outro cliente assinou o mesmo tópico, mas com o QoS 2, ele receberá a mesma mensagem mas com QoS 2. Um outro exemplo, se um cliente assina um tópico com QoS 2 e uma mensagem é publicada com QoS 0, o cliente receberá a mensagem com QoS 0.

Altos níveis de QoS são mais confiáveis, mas envolvem uma maior latência<sup>12</sup> e requerem maior largura de banda.

- QoS 0: O *broker/client* entregará a mensagem uma vez, sem confirmação.
- QoS 1: O *broker/client* entregará a mensagem pelo menos uma vez, com um pedido de confirmação.

---

<sup>12</sup> Tempo de resposta entre o envio da mensagem e a confirmação do seu recebimento.

- QoS 2: O *broker/client* entregará a mensagem exatamente uma vez usando um *handshaking*<sup>13</sup> de quatro passos.

#### 2.2.4 Retenção de mensagens

Todas as mensagens podem ser configuradas como “retidas”. Isto significa que o *broker* manterá a mensagem mesmo depois dela ter sido enviada para todos os assinantes do tópico. Se um novo cliente assina este tópico em que a mensagem foi configurada como retida, a mensagem será enviada a este cliente. Isto é um mecanismo importante. Se um tópico é atualizado com pouca frequência, sem o mecanismo de reter a mensagem, um novo cliente que assinasse este tópico ficaria muito tempo sem receber uma atualização. Com uma mensagem retida, o cliente receberia uma atualização instantânea.

#### 2.2.5 Sessão limpa / Conexões duradouras

Na conexão, um cliente define o *flag* da “sessão limpa”, que às vezes também é conhecido como o sinalizador de “início limpo”. Se a sessão limpa for configurada como falsa, a conexão será tratada como durável. Isso significa que quando o cliente se desconectar, todas as assinaturas permanecerão e todas as mensagens subsequentes de níveis QoS 1 ou 2 serão armazenadas até que o cliente se conecte novamente no futuro. Se a sessão limpa for verdadeira, todas as assinaturas serão removidas para o cliente quando ele for desconectado.

---

<sup>13</sup> É um processo automático de negociação que dinamicamente define os parâmetros de um canal de comunicação estabelecido entre duas entidades.

### 2.2.6 Wills (Testamento)

Quando um cliente se conecta a um *broker*, ele pode informar ao *broker* que ele possui um testamento. Esta é uma mensagem que deseja que o *broker* envie quando o cliente se desconecta de forma inesperada. A mensagem de testamento tem um tópico, QoS e o status de mensagem retida, exatamente como qualquer outra mensagem.

## 2.3 MQTT-SN (MQTT FOR SENSOR NETWORKS)

Conforme em Standford-Clark e Truong (2018), o MQTT-SN foi projetado para ser o mais próximo possível do MQTT, mas foi adaptado às peculiaridades de um ambiente de comunicação sem fio com banda baixa de conexão, alta taxa de falhas de conexão, mensagens curtas, etc. O MQTT-SN é otimizado para implementação em dispositivos de baixo custo e com limitação de armazenamento e capacidade de processamento. Foi desenvolvido para rodar em cima da camada APS ZigBee, ao contrário do MQTT que exige uma camada básica de TCP/IP (e isso é muito complexo para dispositivos muito simples e de baixo custo) o MQTT-SN é projetado de tal maneira que ele é agnóstico em relação às camadas de serviços de rede. Qualquer rede que ofereça uma transferência de dados bidirecional entre qualquer nó e em particular um *gateway* deve ser capaz de suportar MQTT-SN.

### 2.3.1 Arquitetura do MQTT-SN

A arquitetura do MQTT-SN pode ser vista na Figura 2. Ela é composta por três componentes: clientes MQTT-SN, *gateways* MQTT-SN e *forwarders* (encaminhadores) MQTT-SN.

Clientes MQTT-SN conectam-se a um *broker* MQTT através de um *gateway* MQTT-SN usando o protocolo MQTT-SN. Um *gateway* MQTT-SN pode ou não ser integrado com um *broker* MQTT. No caso de um *gateway* independente, o protocolo MQTT é usado entre

o *broker* MQTT e o *gateway* MQTT-SN. Sua principal função é a tradução entre MQTT e MQTT-SN.

Os clientes MQTT-SN também podem acessar um *gateway* através de um *forwarder* caso o *gateway* não esteja diretamente ligado à rede. O *forwarder* simplesmente encapsula os quadros MQTT-SN que ele recebe no lado sem fio e os encaminha inalterados para o *gateway*; na direção oposta, ele desencapsula os quadros que recebe do *gateway* e os envia para os clientes, inalterados também. Dependendo de como um *gateway* realiza a conversão de protocolo entre MQTT e MQTT-SN, podemos diferenciar entre dois tipos de *gateway*, ou seja, *gateway* transparentes e agregados.

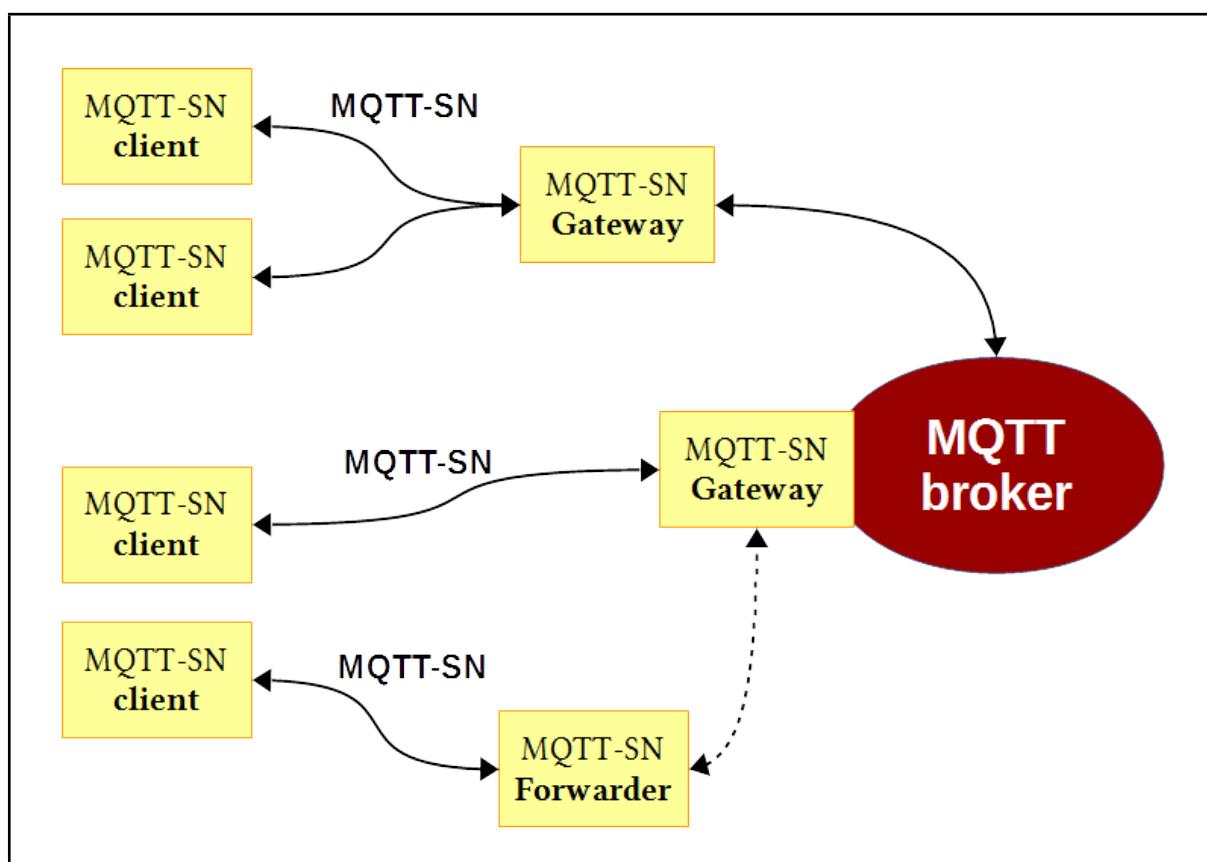


Figura 2: A arquitetura do MQTT-SN  
Fonte: (STANFORD-CLARK e TRUONG, 2013)

## 2.4 MOSQUITTO (MQTT BROKER)

Mosquitto é um projeto da divisão de IoT do Eclipse<sup>14</sup>. Ele fornece uma implementação leve do servidor de protocolos MQTT e MQTT-SN, escrito em C. (LIGHT, 2017)

A motivação para escrevê-lo em C é permitir que o servidor seja executado em máquinas que não têm capacidade para executar uma JVM (Máquina virtual Java). Os sensores e atuadores, que são muitas vezes fontes e destinos das mensagens MQTT e MQTT-SN, podem vir a ser muito pequenos e de baixa potência. Isso também se aplica às máquinas incorporadas às quais estão conectadas, é onde o Mosquitto poderia ser executado.

No IBM AlphaWorks<sup>15</sup>, em 2008, foi lançada a Really Small Message Broker (RSMB). Seu objetivo era ser um servidor MQTT simples e elementar. Seu código foi fechado e lançado sob uma licença somente de avaliação. Nos próximos dois anos, um seguimento pequeno mas entusiasmado aderiu ao projeto. Em 2010, Roger Light<sup>16</sup> aprendeu sobre MQTT e RSMB em uma apresentação de Andy Stanford-Clark<sup>17</sup>. Roger então decidiu criar o Mosquitto para ser uma alternativa de código aberto ao RSMB.

A partir daí, o Mosquitto passou a ter um seguimento entusiasmado próprio e cresceu incluindo funções não disponíveis no RSMB. Ele passou a fazer parte de várias distribuições Linux. O RSMB teve algumas capacidades do MQTT-SN adicionadas, mas que não foram lançadas fora da IBM.

Normalmente, a implementação atual do Mosquitto de Roger Light tem um executável de tamanho em torno de 120kB que consome cerca de 3MB de RAM com 1000 clientes conectados. Há relatos de testes bem-sucedidos com 100.000 clientes conectados com modestas taxas de mensagens.

O servidor Mosquitto, além de aceitar conexões de aplicativos de cliente MQTT, possui o recurso *bridge* (ponte) que permite conectar-se a outros servidores MQTT, incluindo outras instâncias do Mosquitto. Tornando possível que redes de servidores MQTT se-

---

<sup>14</sup> <https://iot.eclipse.org/>

<sup>15</sup> IBM AlphaWorks é uma comunidade da Web para desenvolvedores que podem visualizar e colaborar em tecnologia emergente dos laboratórios de pesquisa da IBM e transformá-los em produtos comerciais.

<sup>16</sup> Professor assistente na Universidade de Nottingham.

<sup>17</sup> Engenheiro britânico, pesquisador em tecnologia da informação, especializado em telemetria e no padrão de mensagens *publish/subscribe*.

jam construídas, passando mensagens MQTT de qualquer local na rede para qualquer outra rede, dependendo da configuração das pontes.

O Eclipse Mosquitto é um MQTT *broker*. Ele é licenciado pela EPL/EDL que implementa os protocolos versão 3.1 e 3.1.1. O termo “*broker*”, em alguns textos, tem sido substituído por “*server*”. O projeto Mosquitto fornece uma pequena implementação do servidor de protocolos MQTT e MQTT-SN. Essa pequena implementação possui as seguintes funções:

1. São incluídas somente as funções necessárias (a compilação condicional pode ser usada para omitir funções desnecessárias para uma aplicação particular).
2. As funções são codificadas da forma mais eficiente possível.

O servidor tem as seguintes características, que não estão descritas na especificação do protocolo MQTT:

1. Um MQTT bridge, permite ao Mosquitto se conectar a outros servidores MQTT.
2. Comunicação de forma segura usando SSL/TLS.
3. Autenticação de usuário – Podendo restringir a um usuário o acesso a alguns tópicos.

O Mosquitto tem a capacidade de traduzir e transferir mensagens entre MQTT e MQTT-SN, ele é capaz de atuar como um *gateway* entre dispositivos que se comunicam com um dos dois protocolos.

Ele mantém atualizado com qualquer mudança de especificação sobre os protocolos MQTT ou MQTT-SN, como por exemplo, o resultado da padronização da especificação OASIS MQTT. O Mosquitto adere às especificações e padrões do protocolo o mais próximo possível, portanto, pode ser usado em testes de conformidade. Qualquer comportamento não padronizado incluído é opcional.

## 3 COMPONENTES E ARQUITETURAS

### 3.1 COMPONENTES (HARDWARE E SOFTWARE)

#### 3.1.1 Arduino

O Arduino pode ser usado para o desenvolvimento de objetos que podem interagir com o meio físico, através de vários tipos de entradas como *switches* ou sensores, sendo assim capaz de controlar vários tipos de elementos como *leds*, motores DC, motores de passo, mecanismos, entre outras saídas. O Arduino foi projetado com o intuito de facilitar o uso da eletrônica em projetos multidisciplinares. O hardware é programado em uma linguagem baseada no Wiring<sup>18</sup> (sintaxe + bibliotecas), similar ao C++ com algumas simplificações e modificações. Também conta com uma IDE baseada no Processing<sup>19</sup>.

A IDE do Arduino inclui um editor de código fonte com características tais como destaque na sintaxe, indentação automática e é capaz de compilar e fazer o upload para a placa com apenas um clique. Não é preciso editar makefiles ou executar utilitários em linha de comando.

A IDE do Arduino vem com uma biblioteca C/C++ chamada "Wiring" (do projeto que tem o mesmo nome), que torna mais fácil muitas operações comuns de entrada e saída. Os programas para Arduino são escritos em C/C++, embora os usuários somente necessitem definir duas funções para fazer um programa executável:

---

<sup>18</sup> É um framework de código aberto para microcontroladores. <http://wiring.org.co/>

<sup>19</sup> É um sketchbook de software flexível e uma linguagem para aprender a codificar dentro do contexto das artes visuais. <https://processing.org/>

**setup()** - Essa função é executada somente uma vez, no momento em que a placa é inicializada. Pode ser usada como uma área para inicializar variáveis, criar objetos e definir os parâmetros para estes objetos.

**loop()** - Função que é a chamada repetidamente. É o mainloop do código Arduino. Ele fica ativo até que a placa seja desligada.

```

Blink | Arduino 1.8.5
Arquivo Editar Sketch Ferramentas Ajuda
Blink
1 void setup() {
2   pinMode(LED_BUILTIN, OUTPUT);
3 }
4
5 // the loop function runs over and over again forever
6 void loop() {
7   digitalWrite(LED_BUILTIN, HIGH);
8   delay(1000);
9   digitalWrite(LED_BUILTIN, LOW);
10  delay(1000);
11 }

```

1 Arduino/Genuino Uno em /dev/ttyACM0

Figura 3: IDE do Arduino - Código Blink  
Fonte: (O autor, 2018)

A IDE do Arduino (Figura 3) usa, para compilar os programas, o kit de ferramentas GNU e a AVR libc, e para fazer o upload dos programas para a placa, o AVRDUDE.

Na Tabela 1 pode-se observar um comparativo entre três placas com diferentes recursos e aplicações:

Nome	Processador	Tensão de operação	CPU speed	Analog In/Out	Digital IO/PWM	EE-PROM [kb]	SRAM [kb]	Flash [kb]	USB	UART
<b>Uno</b>	Atmega328P	5V / 7-12V	16 MHz	6/0	14/6	1	2	32	Comum	1
<b>Due</b>	ATSAM3X8E	3.3V / 7-12V	84 MHz	12/2	54/12	-	96	512	2 Micro	4
<b>Yún</b>	Atmega32U4 AR9331 Linux	5V	16 MHz 400 MHz	12/0	20/7	1	2.5 15MB	32 64MB	Micro	1

Tabela 1: Arduinos - Especificações técnicas  
Fonte: (O autor, 2018)

O Arduino YÚN (Figura 4) é a placa ideal para conectar dispositivos e, de uma maneira geral, projetos de IoT. É uma combinação do poder de ter um Linux instalado com a facilidade de uso do Arduino. O seu processador suporta uma distro Linux que se baseia no OpenWrt<sup>20</sup> chamada Linino OS. Seu grande diferencial está na sua capacidade de integrar com o Linux sendo possível criar scripts shell e python para interações robustas.



Figura 4: Arduino YÚN  
Fonte: (CURVELLO, 2014)

### 3.1.2 Esp8266

O módulo WiFi Esp8266 é um SoC,<sup>21</sup> com uma pilha de protocolo TCP/IP integrada, que permite que qualquer microcontrolador, conectado a ele, tenha acesso a uma rede WiFi. O Esp8266 é capaz de rodar um aplicativo ou servir como um *shield* de acesso à redes sem fio. Cada módulo Esp8266 vem pré-programado com um *firmware* do conjunto de comandos AT<sup>22</sup>, tornando possível, simplesmente conectá-lo a um Arduino (ou qualquer outro microcontrolador) e obter a mesma capacidade Wi-Fi, oferecida pelo WiFi *Shield* do Arduino.

Essa foi a primeira aplicação planejada pelos fabricantes do Esp8266, servir como *shield* WiFi para a placa Arduino. O Esp8266 é uma placa extremamente econômica com uma enorme e crescente comunidade.

<sup>20</sup> O OpenWrt é um sistema operacional GNU/Linux voltado para dispositivos embarcados.

<sup>21</sup> Um SoC (System-On-Chip) é um microchip com todos os circuitos eletrônicos e peças necessárias para um determinado sistema, como um smartphone. E tudo em um único circuito integrado.

<sup>22</sup> O conjunto de comandos AT, oficialmente conhecido como o conjunto de comandos AT padrão Hayes, é o conjunto padrão de instruções para configurar e controlar modems.

Este módulo tem uma capacidade de processamento e armazenamento poderosa o suficiente que permite que ele seja integrado com sensores e outros dispositivos específicos da aplicação por meio de seus GPIOs<sup>23</sup> com o mínimo de desenvolvimento inicial e carga mínima durante o tempo de execução. Seu alto grau de integração *on-chip* permite que o circuito externo mínimo, incluindo o módulo *front-end*, seja projetado para ocupar uma área mínima da PCB<sup>24</sup>.

Há uma fonte quase ilimitada de informações disponíveis para o Esp8266, todas as quais foram fornecidas pelo incrível apoio da comunidade.

A família de modelos do Esp8266 é muito grande. São versões que variam em poder de processamento, quantidade de memória, quantidade de GPIOs e etc. A Figura 5 pode dar uma idéia dos modelos disponíveis.



Figura 5: Família Esp8266  
Fonte: (SINGH, 2016)

O Esp01 é o modelo menor e mais barato. Este módulo, é o mais conhecido, ele é realmente pequeno e custa muito pouco, em torno de 5 dólares. Contudo o número de GPIOs disponíveis é bastante limitado (apenas dois pinos). Também é bastante trabalho-so plugá-lo em uma *protoboard* padrão. Para a gravação do *firmware* ele depende de um conversor USB/Serial como o FTDI FT232 por exemplo. Ele não possui um canal de leitu-

<sup>23</sup> General purpose input and output – São as conexões de propósitos gerais (entradas ou saídas).

<sup>24</sup> Uma placa de circuito impresso (PCB) suporta mecanicamente e liga eletricamente componentes eletrônicos utilizando trilhas condutoras.

ra analógica, sendo assim impossível o seu uso em projetos que fazem uso de sensores analógicos. A Figura 6 mostra o módulo Esp01.

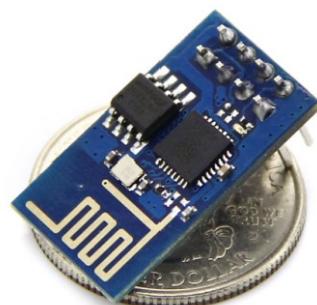


Figura 6: Tamanho do Esp01  
Fonte: (PENNINKHOF, 2015)

Há vários SDKs para o Esp8266. A Tabela 2 mostra alguns deles.

SDK	Descrição
NodeMcu	Firmware baseado em Lua
Arduino	Arduino like programming
MicroPython	Programação Python
Esp8266 BASIC	Interpretador BASIC open source para IoT
Zbasic for Esp8266	Um subconjunto do Microsoft Visual BASIC
Espruino	JavaScript SDK (emula Node.js)
Mongoose <i>firmware</i>	Cloud service
Esp-Open-SDK	Implementação open source do SDK padrão
Esp-Open-RTOS	Baseado no FreeRTOS
Zerynth	Programação Python

Tabela 2: Lista de SDKs para Esp8266  
Fonte: (O autor, 2018)

### 3.1.2.1 Comandos AT

O conjunto de comandos AT, oficialmente conhecido como o conjunto de comandos AT padrão Hayes<sup>25</sup>, é o conjunto padrão de instruções para configurar e controlar modems. Este conjunto foi desenvolvido pela Hayes Microcomputer Products. Hayes foi um pioneiro no campo dos modems, e seu padrão é usado em sua totalidade ou em parte por quase todos os fabricantes de modems para uso com computadores pessoais. Os comandos são sequências curtas de caracteres ASCII<sup>26</sup>. Todas as strings de comando (ou seja, sequências de caracteres) devem ser precedidas pelas letras AT, uma abreviação de “attention” (atenção), representando o nome do conjunto.

Geralmente, há alguma variação no conjunto de comandos de acordo com o fabricante do modem. Isto é particularmente verdadeiro para os comandos proprietários. Assim, é melhor consultar o conjunto de comandos para um modelo específico de modem, em vez de confiar em qualquer listagem genérica de comandos.

Um dos comandos AT mais conhecidos é o ATZ, que normalmente aparece na inicialização do modem. Essa string é uma combinação de AT, que inicia os comandos, e Z, que redefine o modem para seu estado padrão. O comando ATDT representa a sequência de inicialização seguida por D, que representa discagem, e T, que representa discagem por tom (em oposição a P para discagem por pulso). Os modems são configurados enviando-lhes comandos AT do computador nas mesmas linhas seriais usadas pelo computador para enviar dados ao modem.

O módulo WiFi Esp8266 é um chip integrado projetado para comunicação com o mundo da Internet através de sinais de rádio WiFi. Tem processamento e memória integrados que permitem a integração com a eletrônica através de seus GPIOs. O Esp8266, por exemplo, pode ser usado para conectar projetos com um Arduino à Internet. Em sua configuração padrão, é inicializado no modo de modem serial. Neste modo, é possível se comunicar com ele usando um conjunto de comandos AT. A seguir serão apresentados alguns dos comandos AT conhecidos que o Esp8266 suporta.

---

<sup>25</sup> Dennis C. Hayes (nascido em 1950) foi o fundador de produtos de comunicação para microcomputador. Fabricante de modems, foi pioneiro por introduzir um conjunto de comandos de comunicação apelidado de comandos Hayes, o que foi, posteriormente, utilizado na maioria dos modems produzidos até hoje.

<sup>26</sup> ASCII (do inglês American Standard Code for Information Interchange; “Código Padrão Americano para o Intercâmbio de Informação”).

A Tabela 3 exibe uma lista com alguns comandos AT separados por camadas. Essa lista não contém todos os comandos. Para obter uma lista completa é preciso consultar a documentação referente ao *firmware* AT instalado no Esp8266. É importante verificar se os comandos são compatíveis com a versão do *firmware* que está sendo usada no Esp8266. Os comandos da Tabela 3 são compatíveis com a versão do *firmware* mostrada na Figura 7. A imagem mostra a saída do terminal conectado ao Esp8266 após o comando AT+GMR.

```
AT+GMR
AT version:1.1.0.0(May 11 2016 18:09:56)
SDK version:1.5.4(baaeaebb)
Ai-Thinker Technology Co. Ltd.
Jun 13 2016 11:29:20
OK
```

Figura 7: Versão do *firmware* utilizada no Esp8266

Fonte: (O autor, 2018)

Básico	Camada WiFi	Camada TCP/IP
AT	AT+CWMODE	AT+CIPSTATUS
AT+RST	AT+CWJAP	AT+CIPSTART
AT+GMR	AT+CWLAP	AT+CIPSEND
AT+GSLP	AT+CWQAP	AT+CIPCLOSE
ATE	AT+CWSAP	AT+CIFSR
	AT+CWLIF	AT+CIPMUX
	AT+CWDHCP	AT+CIPSERVER
	AT+CIPSTAMAC	AT+CIPMODE
	AT+CIPAPMAC	AT+CIPSTO
	AT+CIPSTA	AT+CIUPDATE
	AT+CIPAP	+IPD

Tabela 3: Lista de comandos AT

Fonte: (O autor, 2018)

Alguns comandos podem não funcionar pelas seguintes razões, o *firmware* pode ser de uma versão mais antiga, antes do comando ter sido implementado ou o comando pode ter sido descontinuado em versões de *firmware* mais recentes. Um exemplo de um comando descontinuado é o comando AT+UART que era usado para modificar a velocidade de comunicação serial do módulo. Este comando foi substituído pelos comandos

AT+UART\_CUR e AT+UART\_DEF. O primeiro modifica a velocidade do módulo mas não salva a configuração na memória flash, sendo assim, após um *reset* a velocidade volta à velocidade original que veio gravada no *firmware*, o segundo comando tem a mesma função porém salvando a configuração na memória flash, garantindo assim, que ela permanecerá a mesma mesmo após uma reinicialização do módulo.

A seguir uma breve definição de alguns dos comandos da Tabela 3.

1. **AT** – Verifica se o módulo está conectado corretamente e o seu funcionamento. O módulo responderá com uma confirmação.
2. **AT + RST** – Dá um *reset* na placa. É uma boa prática executar um *reset* antes ou depois de qualquer programação.
3. **AT + GMR** – Mostra a versão do *firmware* instalada no Esp8266.
4. **AT + CWLAP** – Detecta e mostra os pontos de acesso disponíveis na área de alcance junto com suas respectivas intensidades de sinal.
5. **AT + CWJAP = "SSID", "PASSWORD"** - Conecta o Esp8266 ao SSID especificado no comando AT mencionado.
6. **AT + CIFSR** – Mostra o endereço IP obtido pelo Esp8266.
7. Se o usuário quiser se desconectar de qualquer ponto de acesso, pode usar o seguinte comando **AT + CWJAP = "", "**
8. **AT + CWMODE = 1** – Define o modo WiFi. Pode ser configurado no modo *Station*, *AP* ou *Station/AP*. Deve estar sempre configurado para o Modo 1 se o módulo for usado como um nó.

### 3.1.2.2 NodeMcu (*Firmware* Lua)

O NodeMcu, é um *firmware* de código aberto e um kit de desenvolvimento, que facilita a criação de um protótipo de produto de IoT, com apenas algumas linhas de script Lua. O lema do NodeMcu é “Conecte ‘coisas’ facilmente”.

É um *firmware* interativo, baseado em Lua, para o SoC WiFi Esp8266 (modelo Esp12E). Baseado no projeto eLua e construído sobre o SDK Non-OS da Espressif para Esp8266. Também é uma placa de hardware de código aberto que, ao contrário dos módulos Esp8266 WiFi, inclui um chip CP2102<sup>27</sup> (conversor USB/Serial), para programação e depuração, se encaixa facilmente nas *protoboards* padrão e pode, simplesmente, ser alimentada, via sua porta micro USB. O NodeMcu apresenta seis GPIOs extras em relação ao Esp12E.

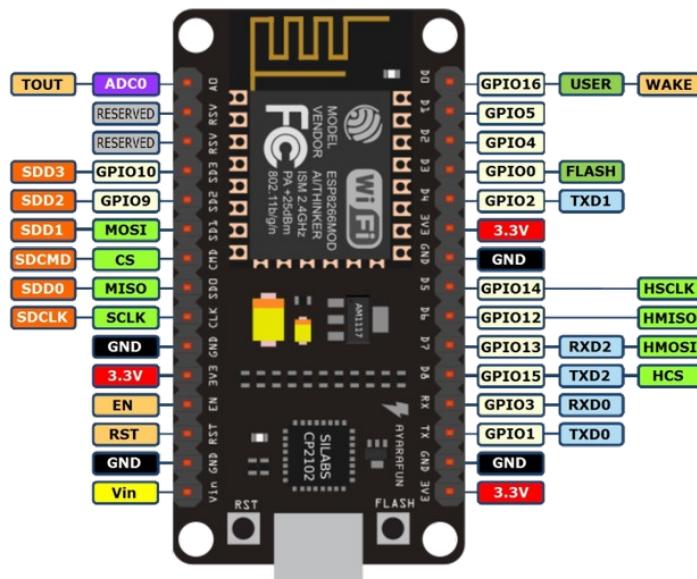


Figura 8: Descrição dos pinos do NodeMcu  
Fonte: (VEIGA, 2017)

Esse kit de desenvolvimento baseado no Esp8266 integra GPIO, pwm, i<sup>2</sup>C, ADC e 1-wire, tudo em uma placa. As portas GPIO do NodeMcu e suas respectivas funções são mostradas na Figura 8.

<sup>27</sup> O módulo conversor USB TTL CP2102 é utilizado para comunicação entre o computador e dispositivos como microcontroladores, módulos e outros equipamentos que utilizam comunicação via interface serial.

Algumas de suas principais características :

- Open-source
- Interativo
- Programável
- Barato
- Simples
- Inteligente
- WiFi

Dentre todas essas características interessantes, há três que se destacam:

- i. **Arduino *like programming*** – É possível programar como se programa um Arduino, mas, interativamente usando scripts em Lua. Através da IDE Esplorer, que será vista mais a frente, é possível enviar para o NodeMcu scripts em Lua e até mesmo comandos AT. É possível também depurar código através do monitor de porta serial dentro da IDE.
- ii. **Node.js *style network API*** – O modelo de programação é similar ao Node.js<sup>28</sup>, só que em linguagem Lua. É um modelo assíncrono e orientado a eventos. Muitas funções, portanto, possuem parâmetros para funções *callback*<sup>29</sup>, permitindo associar funções de *callback* a eventos. Esses eventos podem ser gerados por timers por exemplo. Também é possível associar *callbacks* a funções de comunicação MQTT e HTTP. Diferente de outros modelos, como no Arduino em que existe uma função principal que fica em loop infinito, na programação em Lua no NodeMcu não<sup>30</sup> existe o looping principal, como dito acima é uma programação assíncrona não bloqueante.
- iii. **Lowest cost WiFi** – Custo muito baixo, em torno de US\$ 5,00.

---

<sup>28</sup> Node.js é uma plataforma construída sobre o motor JavaScript do Google Chrome. Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos.

<sup>29</sup> Uma função *callback* é um pedaço de código que é passado como parâmetro para alguma outra função. É esperado que o método execute o código do argumento em algum momento.

<sup>30</sup> É possível usar um loop infinito no NodeMcu porém não é recomendado. Essa prática poderia causar mal funcionamento, devido ao seu próprio modelo assíncrono.

Existem muitos módulos de software disponíveis para o NodeMcu, por isso, não há mais um *firmware* padrão para download. É possível customizar um *firmware* com os módulos necessários para o projeto e assim fazer o *upload*<sup>31</sup> do *firmware* para o dispositivo. Alguns dos módulos (bibliotecas) disponíveis podem ser vistos na Figura 9.

<input type="checkbox"/> ADC <a href="#">[link]</a>	<input type="checkbox"/> end user setup <a href="#">[link]</a>	<input type="checkbox"/> perf <a href="#">[link]</a>	<input type="checkbox"/> Switec <a href="#">[link]</a>
<input type="checkbox"/> ADS1115 <a href="#">[link]</a>	<input checked="" type="checkbox"/> file <a href="#">[link]</a>	<input type="checkbox"/> PWM <a href="#">[link]</a>	<input type="checkbox"/> TCS34725 <a href="#">[link]</a>
<input type="checkbox"/> ADXL345 <a href="#">[link]</a>	<input type="checkbox"/> gdbstub <a href="#">[link]</a>	<input type="checkbox"/> RC (no docs) <a href="#">[link]</a>	<input type="checkbox"/> TM1829 <a href="#">[link]</a>
<input type="checkbox"/> AM2320 <a href="#">[link]</a>	<input checked="" type="checkbox"/> GPIO <a href="#">[link]</a>	<input type="checkbox"/> rfswitch <a href="#">[link]</a>	<input checked="" type="checkbox"/> timer <a href="#">[link]</a>
<input type="checkbox"/> APA102 <a href="#">[link]</a>	<input type="checkbox"/> HDC1080 <a href="#">[link]</a>	<input type="checkbox"/> rotary <a href="#">[link]</a>	<input type="checkbox"/> TSL2561 <a href="#">[link]</a>
<input type="checkbox"/> bit <a href="#">[link]</a>	<input type="checkbox"/> HMC5883L <a href="#">[link]</a>	<input type="checkbox"/> RTC fifo <a href="#">[link]</a>	<input type="checkbox"/> U8G <a href="#">[link]</a>
<input type="checkbox"/> Bloom filter <a href="#">[link]</a>	<input type="checkbox"/> HTTP <a href="#">[link]</a>	<input type="checkbox"/> RTC mem <a href="#">[link]</a>	<input checked="" type="checkbox"/> UART <a href="#">[link]</a>
<input type="checkbox"/> BME280 <a href="#">[link]</a>	<input type="checkbox"/> HX711 <a href="#">[link]</a>	<input type="checkbox"/> RTC time <a href="#">[link]</a>	<input type="checkbox"/> UCG <a href="#">[link]</a>
<input type="checkbox"/> BME680 <a href="#">[link]</a>	<input type="checkbox"/> I <sup>2</sup> C <a href="#">[link]</a>	<input type="checkbox"/> Si7021 <a href="#">[link]</a>	<input type="checkbox"/> websocket <a href="#">[link]</a>
<input type="checkbox"/> BMP085 <a href="#">[link]</a>	<input type="checkbox"/> L3G4200D <a href="#">[link]</a>	<input type="checkbox"/> Sigma-delta <a href="#">[link]</a>	<input checked="" type="checkbox"/> WiFi <a href="#">[link]</a>
<input type="checkbox"/> CoAP <a href="#">[link]</a>	<input type="checkbox"/> MCP4725 <a href="#">[link]</a>	<input type="checkbox"/> SJSON <a href="#">[link]</a>	<input type="checkbox"/> WiFi monitor <a href="#">[link]</a>
<input type="checkbox"/> color utils <a href="#">[link]</a>	<input type="checkbox"/> mDNS <a href="#">[link]</a>	<input type="checkbox"/> SNTP <a href="#">[link]</a>	<input type="checkbox"/> WPS <a href="#">[link]</a>
<input type="checkbox"/> Cron <a href="#">[link]</a>	<input type="checkbox"/> MQTT <a href="#">[link]</a>	<input type="checkbox"/> Somfy <a href="#">[link]</a>	<input type="checkbox"/> WS2801 <a href="#">[link]</a>
<input type="checkbox"/> crypto <a href="#">[link]</a>	<input checked="" type="checkbox"/> net <a href="#">[link]</a>	<input type="checkbox"/> SPI <a href="#">[link]</a>	<input type="checkbox"/> WS2812 <a href="#">[link]</a>
<input type="checkbox"/> DHT <a href="#">[link]</a>	<input checked="" type="checkbox"/> node <a href="#">[link]</a>	<input type="checkbox"/> SQLite 3 <a href="#">[link]</a>	<input type="checkbox"/> WS2812 effects <a href="#">[link]</a>
<input type="checkbox"/> DS18B20 <a href="#">[link]</a>	<input type="checkbox"/> 1-Wire <a href="#">[link]</a>	<input type="checkbox"/> struct <a href="#">[link]</a>	<input type="checkbox"/> XPT2046 <a href="#">[link]</a>
<input type="checkbox"/> encoder <a href="#">[link]</a>	<input type="checkbox"/> PCM <a href="#">[link]</a>		

Figura 9: Lista de módulos disponíveis para o *firmware* Lua

Fonte: (NODEMCU-BUILD, 2017)

É possível destacar alguns dos módulos dessa lista e suas respectivas funções.

- **ADC** – Serve para trabalhar como conversões AD/DA,
- **DHT** – Torna fácil trabalhar com a família de sensores de umidade e temperatura DHT11 e DHT22.
- **PWM** – Um módulo para trabalhar com sinais do tipo PWM.
- **MQTT** – Para trabalhar com o protocolo MQTT.
- **HTTP** – Trabalha com serviços HTTP.
- **UART** – Comunicação serial.

<sup>31</sup> O APÊNDICE A contém uma explicação detalhada do processo de customização do *firmware* e também do processo de *upload*.

### 3.1.2.3 Linguagem Lua

A linguagem Lua foi projetada, implementada e desenvolvida no Brasil. Criada na PUC-RJ. Os seus criadores são: Roberto Ierusalimschy, Waldemar Celes e Luiz Henrique Figueiredo. Lua é um software de exportação sendo usado em vários projetos por todo o mundo.

Lua é, hoje, amplamente usada em todas as áreas que podem se beneficiar de uma linguagem de *script* simples, extensível, portável e eficiente, como sistemas embarcados, dispositivos móveis e, é claro, jogos. (IERUSALIMSCHY, 2015, pag. V)

O objetivo principal de Lua é de se integrar a linguagens como C/C++. Lua não busca fazer o que outras linguagens como C já fazem muito bem, como computação de alto desempenho, interagir direto com o hardware e etc. O que Lua vem a oferecer são coisas mais complicadas de fazer em C. Lua se mantém distante do hardware, possui estruturas dinâmicas, ausência de redundâncias e facilidade para teste e depuração. Lua, como toda boa linguagem de *script* tem um ambiente seguro, ou seja, não é possível acessar serviços não autorizados pelo programa hospedeiro (pelo programa onde ela está acoplada), também tem gerenciamento automático de memória, facilidades na manipulação de cadeias de caracteres e outros tipos de estruturas com tamanho dinâmico.

Um dos pontos mais fortes de Lua é a ideia de oferecer meta-mecanismos para implementar recursos, em vez de fornecê-los já prontos na linguagem. Apesar de Lua não ser uma linguagem puramente orientada a objetos, ela permite que se implemente classes e herança. Com esse conceito a linguagem permanece pequena e permite o uso de vários paradigmas de programação como: programação procedural; programação orientada a objetos; programação funcional; programação orientada a dados e descrição de dados. Sua tipagem dinâmica possibilita um alto grau de polimorfismo.

Lua é rápida e tem uma reputação merecida de desempenho. “*Benchmarks*<sup>32</sup> independentes mostram-na como uma das mais rápidas linguagens de *script*”. (IERUSALIMSCHY, 2015, pag. V)

---

<sup>32</sup> Padrão, ou um conjunto de padrões, usado como ponto de referência para avaliar o desempenho ou o nível de qualidade.

Ela, executa *bytecodes*<sup>33</sup> em uma “máquina virtual, baseada em registradores”<sup>34</sup> e possui gerenciamento automático de memória. É possível dividir os usuários de Lua em três grandes grupos: os que a utilizam embarcada em uma aplicação, os que a utilizam de maneira autossuficiente, e aqueles que a utilizam combinado com C. Lua é usada em várias aplicações industriais:

- Adobe Lightroom (Um milhão de linhas de código Lua)
- Sistemas embarcados
  - Middleware Ginga (programas para Tv digital)
  - Roteadores (Cisco)
  - Impressoras (Oliveti)
  - Dispositivos M2M (Sierra wireless)
- Jogos – World of Warcraft e Angry Birds e muitos outros
- Nmap (linguagem de *script* embutida na aplicação)
- VLC media player

Lua é uma linguagem simples e pequena. A distribuição completa da linguagem, isto é, código-fonte, manual, e mais os binários, cabe em um disquete. Ela é capaz de rodar em todos os tipos de Unix e Windows, e na maioria das plataformas existentes como Symbian, Nintendo DS, PSP, PS3 (PPE e SPE), Android, iOS, IBM z/OS e outros. Também roda em microprocessadores embarcados como Arm e Rabbit. Pode-se estender facilmente programas em Lua usando funções escritas em C e também é possível estender outras aplicações usando código Lua.

É software livre de código aberto, distribuída sob uma licença muito liberal conhecida como licença MIT. A licença MIT foi criada pelo Instituto de Tecnologia de Massachusetts (MIT). Ela é uma licença permissiva utilizada tanto em software livre quanto em software proprietário. A licença é permissiva e seu texto é bem mais explícito ao tratar dos direitos que estão sendo transferidos, afirmando que qualquer pessoa que obtém uma cópia do software e seus arquivos de documentação associados pode lidar com eles sem

---

<sup>33</sup> É um estágio intermediário entre o código-fonte (escrito numa linguagem de programação específica) e a aplicação final, sendo sua vantagem a dualidade entre a portabilidade e a ausência de pré-processamento típico dos compiladores.

<sup>34</sup> É uma implementação de máquina virtual em que a estrutura, onde os operandos são armazenados, é baseada em registradores.

restrição, incluindo sem limitação, os direitos a usar, copiar, modificar, mesclar, publicar, distribuir, vender cópias do software. As condições impostas para tanto são apenas manter o aviso de *copyright* e uma cópia da licença em todas as cópias do software. Por isso, Lua pode ser usada para quaisquer propósitos, incluindo propósitos comerciais, sem qualquer custo ou burocracia. Basta fazer o download e usá-la.

Como dito anteriormente, Lua é um produto de exportação e tem importância global. É a única linguagem de programação de impacto desenvolvida fora do primeiro mundo. Tem uma grande importância nas áreas de desenvolvimento de jogos, sistemas embarcados e IoT. O Gráfico 3.1 mostra um ranking das linguagens de *script* mais usadas no desenvolvimento de jogos.

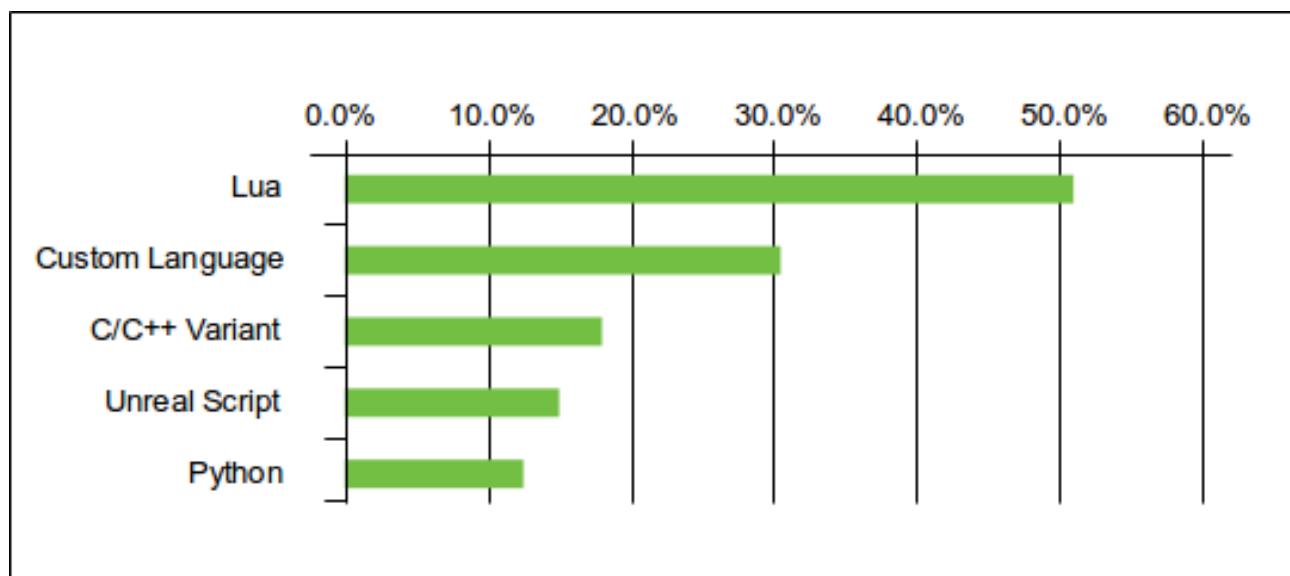


Gráfico 3.1: Linguagens de *script* mais usadas em jogos  
Fonte: (DELOURA, 2009)

### 3.1.2.3.1 eLua (Embedded Lua)

Como foi falado no capítulo 3.1.2.2, o NodeMcu roda uma versão de eLua (*Embedded Lua*). Ela é uma implementação completa da linguagem Lua para sistemas embarcados e adiciona recursos específicos a estes ambientes, trazendo um nível de portabilidade sem precedentes ao mundo do software embarcado. eLua segue o mesmo princípio mini-

malista e funcional de Lua, alinhada com a filosofia KISS, *Keep It Small and Simple* (mantenha-o pequeno e simples).

Alguns aspectos sobre eLua:

- eLua roda sobre o “*bare-metal*”<sup>35</sup>. Não existe um sistema operacional entre os programas e o microcontrolador. Embora ela ofereça algumas características de um SO, como diferentes tipos de sistema de arquivos, um shell de comandos e etc.
- Como em Lua um programa roda em uma variedade de plataformas e arquiteturas.
- eLua não é um conjunto limitado de Lua com o intuito de caber no ambiente embarcado. Ao contrário disso, ela oferece as mesmas características que o ambiente Lua para *desktop*, mas o complementa com características específicas de uso em sistemas embarcados e ao mesmo tempo descarta a necessidade de ter um sistema operacional rodando nos microcontroladores.
- Oferece diferentes “sabores” de uma implementação completa de Lua, pode-se escolher uma versão de implementação que só trabalha com números inteiros ou com números em ponto flutuante. Há um esforço muito grande em tornar Lua mais “*embedded-friendly*” ampliando recursos que permitam um menor consumo de memória e uma melhor performance.
- eLua não é uma aplicação para um sistema operacional ou para um microcontrolador específico. eLua é independente e suporta uma grande quantidade de microcontroladores.
- eLua não é um software comercial, ela usa a mesma licença que Lua, a licença MIT.

eLua traz consigo a oportunidade de programar microcontroladores com recursos encontrados somente em grandes e caros *frameworks* de desenvolvimento.

- Tipagem dinâmica
- Multitarefa colaborativa ou multitarefa não preemptiva
- Múltiplos modelos de gerenciamento de memória
- Funções de primeira classe

---

<sup>35</sup> A expressão “Bare Metal” (metal nu ou metal puro, em inglês) serve para descrever ambientes de TI em que o sistema operacional é instalado diretamente no hardware, em vez de uma camada de sistema hospedando diversas máquinas virtuais, como é realizado em ambientes virtualizados.

### 3.1.3 Raspberry Pi

Um Raspberry Pi é um computador do tamanho aproximado de um cartão de crédito, originalmente projetado para a educação, inspirado pelo BBC Micro<sup>36</sup> de 1981. O objetivo de seu criador, Eben Upton, era criar um dispositivo de baixo custo que melhorasse as habilidades de programação e compreensão de hardware para alunos do nível pré-universitário. Mas graças ao seu pequeno tamanho e preço acessível, foi rapidamente adotado por desenvolvedores e entusiastas da eletrônica para projetos que necessitavam mais que um simples microcontrolador (tal como Arduino).

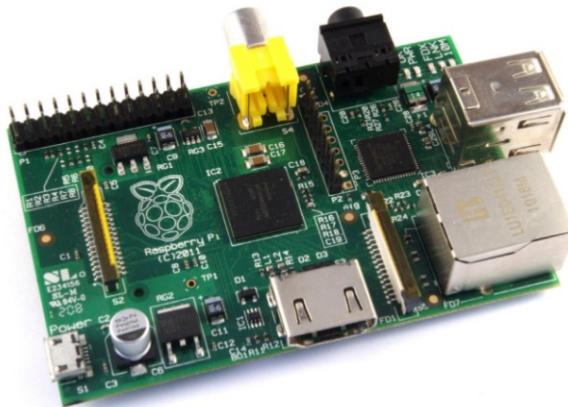


Figura 10: Raspberry Pi - Model B  
Fonte: (UPTON e HALFACREE, 2014)

Apesar de ser mais lento que um *desktop* ou um *notebook* ele é um computador completo. É capaz de rodar vários sistemas operacionais. Existem diversas distribuições Linux pra ele como Raspbian, Pidora e Arch Linux. Ele pode funcionar como um servidor web ou um servidor FTP e etc. O grande diferencial do Pi está no seu custo e no seu processamento que consome pouquíssima energia.

Segundo Upton e Halfacree (2014, p. 15), o hardware do Raspberry Pi é aberto, com exceção do chip principal, o SoC (System on a Chip) Broadcom BCM2835. Esse chip é responsável pelas principais funções da placa: CPU, gráficos, memória, controlador de USB, etc.

---

<sup>36</sup> BBC Microcomputer System é uma série de microcomputadores e periféricos construídos pela Acom Computers Ltd para o BBC Computer Literacy Project (Um projeto de educação por computador).

Muitos dos projetos feitos com o Raspberry Pi também estão abertos e bem documentados podendo ser reconstruídos e modificados. O sistema escolhido para rodar no Raspberry Pi foi o GNU/Linux. Já existem várias distribuições otimizadas pra rodar nele.

Há diversos modelos de Raspberry Pi, variando em capacidade de memória e recursos de processamento, também há diferenças na parte física de acesso aos pinos GPIO.

São estes os modelos disponíveis hoje:

- Raspberry Pi 3 model B
- Raspberry Pi 2 model B
- Raspberry Pi 1 model B e B+
- Raspberry Pi 1 model A e A+
- Raspberry Pi ZERO W
- Raspberry Pi ZERO

O modelo usado neste trabalho é o Raspberry Pi 1 model B (Figura 10). Uma característica poderosa do Raspberry Pi são seus pinos de GPIO que ficam junto à lateral da placa, próximo ao plug amarelo de saída de vídeo.



Figura 11: Raspberry Pi 1 model B - Pinos de GPIO  
Fonte: (CHAUHAN, 2017)

Estes pinos (Figura 11) são a interface entre o Pi e o mundo exterior. São entradas e saídas digitais. É possível ler seus estados, ligado e desligado (entrada) ou através de software ligar e desligar pinos (saída). Dos 26 pinos, 17 são de propósito geral (entradas ou saídas), os outros são pinos de tensão (VCC) e pinos de referência 0V (ground). A Figura 12 mostra as duas nomenclaturas dos pinos que podem ser escolhidas para trabalhar dentro do programa. **GPIO.BOARD**, adota a numeração que está indicada dentro dos círculos. **GPIO.BCM** (Broadcom SOC channel), adota a numeração indicada dentro dos retângulos. O Raspberry Pi é ideal para conectar e controlar dispositivos físicos sobre a Internet.



Figura 12: Nomenclatura dos pinos GPIO - BOARD e BCM  
Fonte: (HAWKINS, 2018)

Não é o objetivo deste trabalho usar o Raspberry Pi como dispositivo que controlará os atuadores e ler os sensores, porém é o objetivo citar maneiras de se fazer IoT usando software e hardware de código aberto. Então, abaixo, pode-se ver um exemplo de como acender um *led* usando o Raspberry Pi.

Ignorando o Raspberry Pi por um momento, pode ser construído um circuito simples. Uma bateria é ligada a uma fonte de luz e uma chave (o resistor, no circuito (Figura 13), está ali para limitar a corrente elétrica e proteger o *led*).

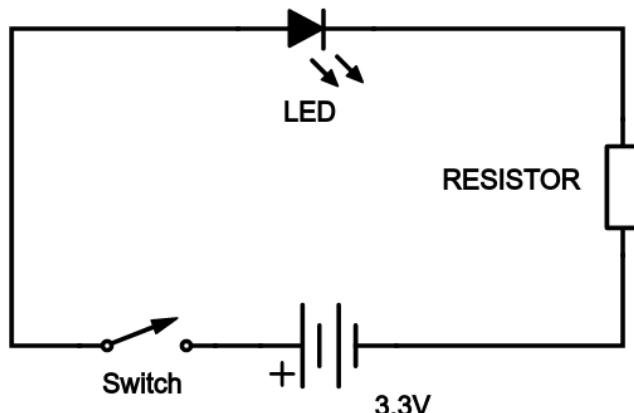


Figura 13: Circuito de exemplo  
Fonte: (O autor, 2018)

Quando um pino de GPIO é usado como saída, o Raspberry Pi substitui a bateria e a chave no diagrama acima. Cada pino pode ligar ou desligar, ou ir a nível lógico alto ou baixo (em termos computacionais). Quando o pino está em nível alto ele passa a ter um potencial de 3.3 volts; quando ele está em nível lógico baixo ele fica com um potencial de

zero volt. A próxima imagem mostra o mesmo circuito montado no Raspberry Pi. Esquema da Figura 14 foi criado com o software Fritzing<sup>37</sup>.

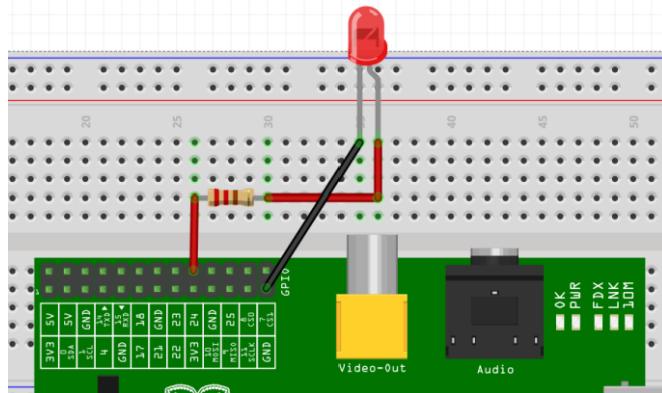


Figura 14: Esquema do Raspberry PI  
Fonte: (O autor, 2018)

O próximo passo é escrever um programa que faça este controle do pino de saída, que faça o pino ir para o nível lógico alto ou baixo. A linguagem que será usada neste exemplo é a linguagem Python, que já vem instalada no Raspbian.

O Código 1, quando executado, faz com que o *led* acenda e apague em intervalos de 1s. A linha 1 do código importa o módulo RPI.GPIO e cria uma “Alias”<sup>38</sup> para que seja possível se referir ao módulo apenas com o nome de GPIO, este dá acesso às portas GPIO sendo possível setar as suas propriedades. A linha 2 importa o módulo que possibilita trabalhar com o tempo no Python, dar uma pausa de 1s por exemplo. A linha 4 seleciona o modo como a GPIO será referenciada, neste caso escolhendo o modo BCM, já citado anteriormente. A linha 5 define que o pino rotulado como 24 será um pino de saída.

Uma questão prática importante, pode ocorrer na interrupção do código, quando se utiliza o barramento GPIO, neste caso específico, acendendo e apagando o *led*. O programa, pode sofrer uma interrupção de três maneiras: a primeira é que ele faça o que foi proposto e encerre naturalmente; a segunda é que o usuário force o encerramento através das teclas “control + c”; a terceira maneira, é que ocorra algum erro e o programa termine abruptamente. Nos casos 2 e 3 é possível que a GPIO usada fique no seu estado corrente, fique “presa”, ou melhor fique “em uso”. Nessa condição, se o programa for executado novamente, é possível que apareça a seguinte mensagem: “*RuntimeWarning: This chan-*

<sup>37</sup> Fritzing – É um software de design para automação e eletrônica direcionado a designers, artistas e todo aquele que tem interesse em computação física e prototipagem.

<sup>38</sup> Apelido.

*nel is already in use, continue anyway".* A maneira correta de prevenir este erro é proteger o laço principal através de um bloco *try/except/finally*, como mostrado no código.

```

1 import RPi.GPIO as GPIO
2 import time
3
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(24, GPIO.OUT)
6
7 try:
8
9     while(1):
10         # acende o led
11         GPIO.output(24, 1)
12         time.sleep(1)
13         # apaga o led
14         GPIO.output(24, 0)
15         time.sleep(1)
16
17 except:
18
19     print "O programa foi interrompido..."
20
21 finally:
22
23     GPIO.cleanup()

```

Código 1: Controlando o *led* com o Raspberry Pi usando Python

A linha 7 dá início ao bloco *try*. A linha 9 define um laço infinito, fazendo com que o *led* pisque por um tempo indeterminado. A linha 10 é apenas um comentário. A linha 11 coloca o pino 24 com nível lógico alto (1). A linha 12 faz com que o programa pause por 1s. A linha 13 tem outro comentário. A linha 14 leva o pino 24 para nível lógico baixo (0). A linha 15 espera por mais 1s reiniciando o laço novamente. Se houver qualquer tipo de interrupção, qualquer tipo de exceção, essa exceção será capturada e a execução será direcionada para a linha 19, logo após o comando *except*. A partir daí a linha 19 imprime o texto que está entre aspas e o programa segue a executar o que está depois do *finally* (linha 21), executando então, na linha 23, a função *GPIO.cleanup()* que é responsável por “limpar” as portas que foram usadas, configuradas no código.

Há duas maneiras erradas de tentar evitar essa situação, uma delas é chamar a função GPIO.cleanup() no início do código. Isso não funciona pois a função limpa as configurações de entrada e saída (dos pinos) que foram feitas no código, o que não surte qualquer efeito nos pinos que estão “presos”, pois foram configurados anteriormente, no código que foi interrompido. A segunda maneira, apelidada de “*chicken’s way out*<sup>39</sup>”, é chamar a função GPIO.setwarnings(False) no início do código. Ela desabilita os avisos (advertências), o que esconde o problema mas não o resolve. O jeito correto e seguro de tratar o problema é como explicado acima, usando o tratamento de exceções.

### 3.1.4 ThingSpeak (Uma plataforma open source para IoT)

“ThingSpeak é uma plataforma de serviço de IoT analítica que permite coletar, visualizar e analisar fluxos de dados ao vivo na nuvem”. (MATHWORKS, 2018). É possível enviar dados para o ThingSpeak a partir de dispositivos, criar visualizações instantâneas de dados ao vivo e enviar alertas usando serviços da web como Twitter e Twilio<sup>40</sup>. Com o MATLAB analytics dentro do ThingSpeak, pode-se escrever e executar o código MATLAB para realizar pré-processamento, visualizações e análises. O ThingSpeak permite que engenheiros e cientistas criem protótipos e sistemas IoT sem configurar servidores ou desenvolver software para Web. As principais capacidades do ThingSpeak incluem:

- Configuração dos dispositivos para enviar dados para o ThingSpeak usando uma API REST<sup>41</sup> ou MQTT.
- Coleta de dados a partir de dispositivos e fontes de terceiros.
- Obtenção de visualizações instantâneas de dados de sensores ao vivo ou históricos de dados coletados.
- Pré-processamento e análise de dados coletados utilizando o MATLAB integrado.
- Execução de análises IoT automaticamente com base em horários ou eventos.

---

<sup>39</sup> O jeito amedrontado, assustado de se fazer algo.

<sup>40</sup> O Twilio é uma plataforma de comunicações em nuvem para chamadas telefônicas e mensagens SMS.

<sup>41</sup> Representational State Transfer (REST), em português, Transferência de Estado Representacional, é um estilo de arquitetura que define um conjunto de restrições e propriedades baseados em HTTP.

- Atualização de dados e de comunicação usando serviços de terceiros como Twilio ou Twitter.

A Internet das Coisas fornece acesso a uma ampla gama de dispositivos integrados e serviços Web. ThingSpeak é uma plataforma IoT que permite coletar, armazenar, analisar, visualizar e atuar em dados de sensores ou atuadores, como Arduino, Raspberry Pi, BeagleBone Black e outros equipamentos. Por exemplo, com o ThingSpeak, é possível criar aplicativos de log de sensores, aplicativos de localização e uma rede social de coisas com atualizações de status, para que se possa ter um termostato doméstico com controle próprio com base em sua localização atual.

O elemento principal da atividade ThingSpeak é o canal, que contém campos de dados, campos de localização e um campo de status. Depois de criar um canal ThingSpeak, você pode gravar dados no canal, processar e visualizar os dados com o código MATLAB e reagir aos dados com tweets e outros alertas. O fluxo de trabalho típico ThingSpeak permite:

1. Criar um canal e coleto dados
2. Analisar e visualizar os dados
3. Atualizar os dados usando qualquer um dos vários aplicativos

#### 3.1.4.1 Coleta de dados

Sensores ou coisas, detectam dados e tipicamente atuam localmente. O ThingSpeak permite que sensores, instrumentos e sites enviem dados para a nuvem onde ele está armazenado em um canal privado ou público. Por padrão, o ThingSpeak armazena dados em canais privados, porém, os canais públicos podem ser usados para compartilhar dados com outros usuários. Uma vez que os dados estão em um canal ThingSpeak, pode-se analisá-lo e visualizá-lo, calcular novos dados ou interagir com redes sociais, serviços da Web e outros dispositivos.

### 3.1.4.2 Análise dos dados

É possível analisar e visualizar dados com o MATLAB. Armazenar dados na nuvem fornece acesso fácil aos dados coletados. Usando ferramentas analíticas on-line, pode-se explorar e visualizar dados, descobrir relacionamentos, padrões e tendências. Calculando novos dados, visualizando-os em gráficos, tabelas e medidores, como mostra o gráfico 3.2. Mais detalhes práticos sobre o uso do ThingSpeak, como taxa de amostragem e contas de usuário, serão vistos na seção 3.2.1.

O ThingSpeak fornece acesso ao MATLAB a fim de ajudar a entender os dados. Sendo possível:

- Converter, combinar e calcular novos dados.
- Programar cálculos para executar em determinados horários.
- Compreender visualmente os relacionamentos em dados usando funções de traçado integradas.
- Combinar dados de múltiplos canais para construir uma análise mais sofisticada.

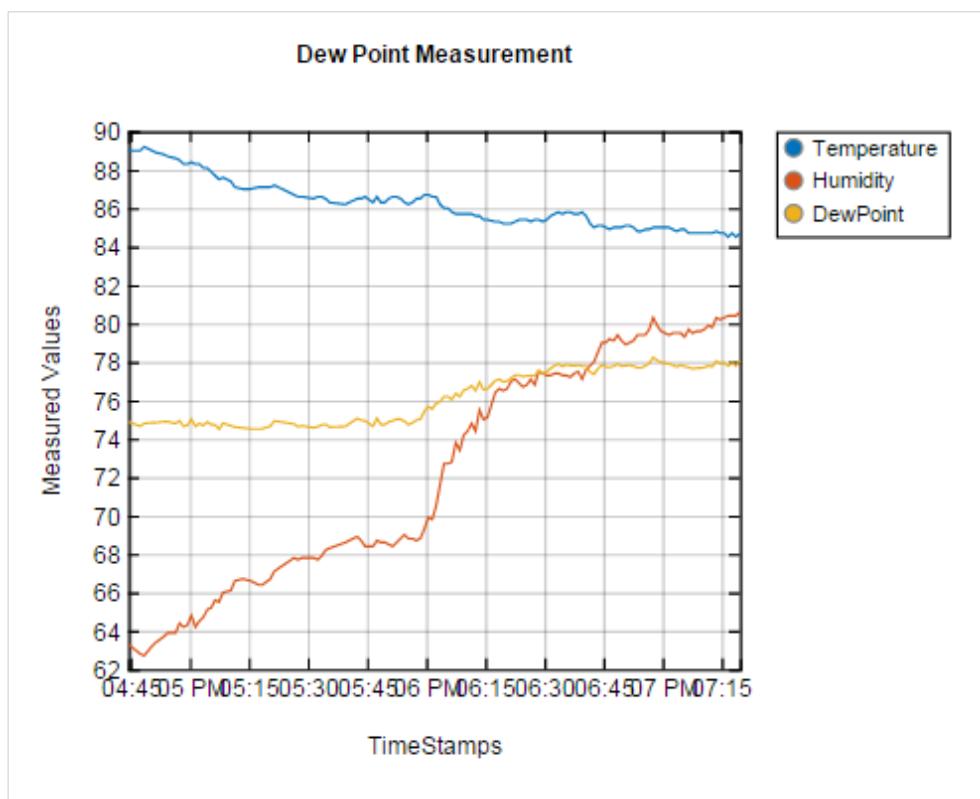


Gráfico 3.2: Plotando gráficos usando o MATLAB  
Fonte: (MATHWORKS, 2018)

### 3.1.4.3 Associe uma ação (Trigger an action)

Atuar sobre os dados pode ser algo tão simples como receber um tweet quando a temperatura que está sendo medida vai acima de 70°C. Também é possível configurar uma ação mais complexa, como ligar um motor quando o nível da água em um tanque de água cair abaixo de um especificado limite. Pode-se controlar dispositivos de controle remoto, como bloqueios de portas operados por bateria, usando o aplicativo TalkBack.

ThingSpeak fornece ferramentas que permitem a comunicação do dispositivo a todas essas ações e muito mais. Permitindo:

- Reagir aos dados – dados brutos e novos dados, que são calculados quando eles entram em um canal.
- Enfileirar comandos para que um dispositivo os execute.

O ThingSpeak oferece todas as ferramentas necessárias para a implementação de uma Casa Inteligente, sendo possível por exemplo, monitorar a temperatura do ambiente e ao mesmo tempo controlar o funcionamento do refrigerador de ar. Também se pode controlar dispositivos através de palavras-chave em *tweets* ou até mesmo “tweetar” uma informação importante coletada de um sensor. Pode-se automatizar ações e tarefas baseadas em tempo usando aplicativo TimeControl<sup>42</sup>. Tudo isso via web, através do computador, *tablet* ou do *smartphone*.

### 3.1.5 Node-RED

O Node-RED é uma poderosa ferramenta de programação para Internet das coisas. Ela é baseada na conexão de componentes gráficos, que na verdade são blocos de códigos que podem enviar, processar e receber informações de diversas fontes e se comunicar usando diversos tipos de protocolos. Esses blocos de códigos são conhecidos como nodes. A conexão destes nodes gera fluxos (*flows*) de informação. Os nodes podem ser definidos como nodes de entrada (*input-nodes*), nodes de processamento (*function-*

---

<sup>42</sup> Permite executar uma ação em um horário específico ou em horários periódicos previamente agendados.

*nodes*) e nodes de saída (*output-nodes*). “Originalmente desenvolvida como um projeto de código aberto na IBM no final de 2013, para suprir a necessidade de rapidamente conectar hardware e dispositivos a serviços web e outros softwares, como uma espécie de ‘cola’ para IoT.” (LEA, 2016)

O Node-RED é uma ferramenta baseada em fluxos que possui um editor que pode ser acessado eu um navegador web. É muito simples conectar esses fluxos de informação, encadeando entradas e saídas, processando e transformando os dados através desse fluxos. Existe uma grande quantidade de nodes na paleta de opções. Após conectar esses nodes é possível executar a aplicação com só um clique.

Estas são suas características principais:

- Editor de fluxos baseado em navegador – Basta um navegador web para começar a conectar o nodes e criar fluxos.
- Construído em Node.js – Aproveitando ao máximo o modelo de I/O direcionada a evento não bloqueante. O Node-RED é ideal para rodar em hardwares de baixo custo tais como Raspberry Pi e também na nuvem.
- Fácil compartilhamento – Os fluxos criados em Node-RED são armazenados usando JSON<sup>43</sup> e podem ser importados e exportados com facilidade permitindo o compartilhamento com outras pessoas.

Com o Node-RED rodando, pode-se ter acesso a seu editor através de um navegador web no endereço <http://localhost:1880>. É possível também acessar o editor usando um outro navegador web em outra máquina, se for conhecido o endereço ip da máquina em que está rodando o Node-RED.

O processo de construção de uma aplicação no Node-RED é feito através de fluxos (*flows*) que contém *nodes* (nós).

A Figura 15 mostra alguns do *nodes* que estão disponíveis na instalação padrão do Node-RED, com exceção dos *nodes* da aba Raspberry Pi que somente estão disponíveis quando o Node-RED é instalado em um Raspberry Pi. Estes *nodes* são: *input-nodes*, *output-nodes*, *function-nodes*, *dashboard-nodes*, *social-nodes*, *network-nodes* e *raspberry pi-nodes*.

**Input-nodes** – Estes *nodes* desencadeiam o início de um fluxo que cria o pacote inicial de informação. Normalmente esses *nodes* são conectados a um evento externo

---

<sup>43</sup> JSON (JavaScript Object Notation – Notação de objetos JavaScript) – É uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para máquinas, é fácil de interpretar e gerar.

como uma requisição HTTP, uma mensagem MQTT, a chegada de uma informação na porta serial e até mesmo um botão. É possível identificar esses *nodes* pois eles têm apenas um ponto de conexão no lado direito, o conector de saída, isto é, não têm ponto de conexão de entrada. Sendo assim não é possível conectar dois *input-nodes*.

**Output-nodes** – Enviaam informação para o meio externo à aplicação. É possível identificá-los pela ausência do conector de saída do lado direito. Com eles é possível enviar uma informação para um MQTT *broker*, um byte pela porta serial, tocar um arquivo de áudio e etc.

**Function-nodes** – Sua função é o processamento do pacote de informação recebido, estes *nodes* têm ambos conexão de entrada e conexão de saída. Ao receber uma mensagem na entrada, ele processa a informação (executa uma função) e com isso é capaz de modificá-la e enviá-la por sua conexão de saída.

**Raspberry Pi-nodes** – Estes *nodes* são específicos para o uso com o Node-RED rodando dentro do Raspberry, eles já vêm instalados por padrão nesse caso. Com estes *nodes* é possível interagir com o Raspberry e com seus pinos de GPIO.

**Dashboards-nodes** – A criação da interface com o usuário fica fácil com o uso destes *nodes*. São botões, caixas de texto, switches, date picker e objetos para a exibição de gráficos tipo linha e também gauges.

**Social-nodes** – Torna possível o enviar e receber e-mails e também a interação com o Twitter.

**Storage-nodes** – Permite a interação com arquivos do sistema hospedeiro. É possível ler o conteúdo de um arquivo, seja ele um String ou buffer binário. Pode-se também gravar o conteúdo de *msg.payload* para um arquivo.

**Network-nodes** – Dentro dessa categoria existe o *node* chamado *ping* que possibilita enviar um *ping* para um endereço ip e receber como retorno o tempo de resposta em ms, este valor vem dentro do objeto *msg.payload*.

O pacote de informação que “flui” através dos *nodes* é um simples objeto JavaScript que se chama *msg*. Este objeto pode ser estendido por outros *nodes* sendo possível adicionar, modificar e remover propriedades dinamicamente. Algumas propriedades merecem um destaque especial:

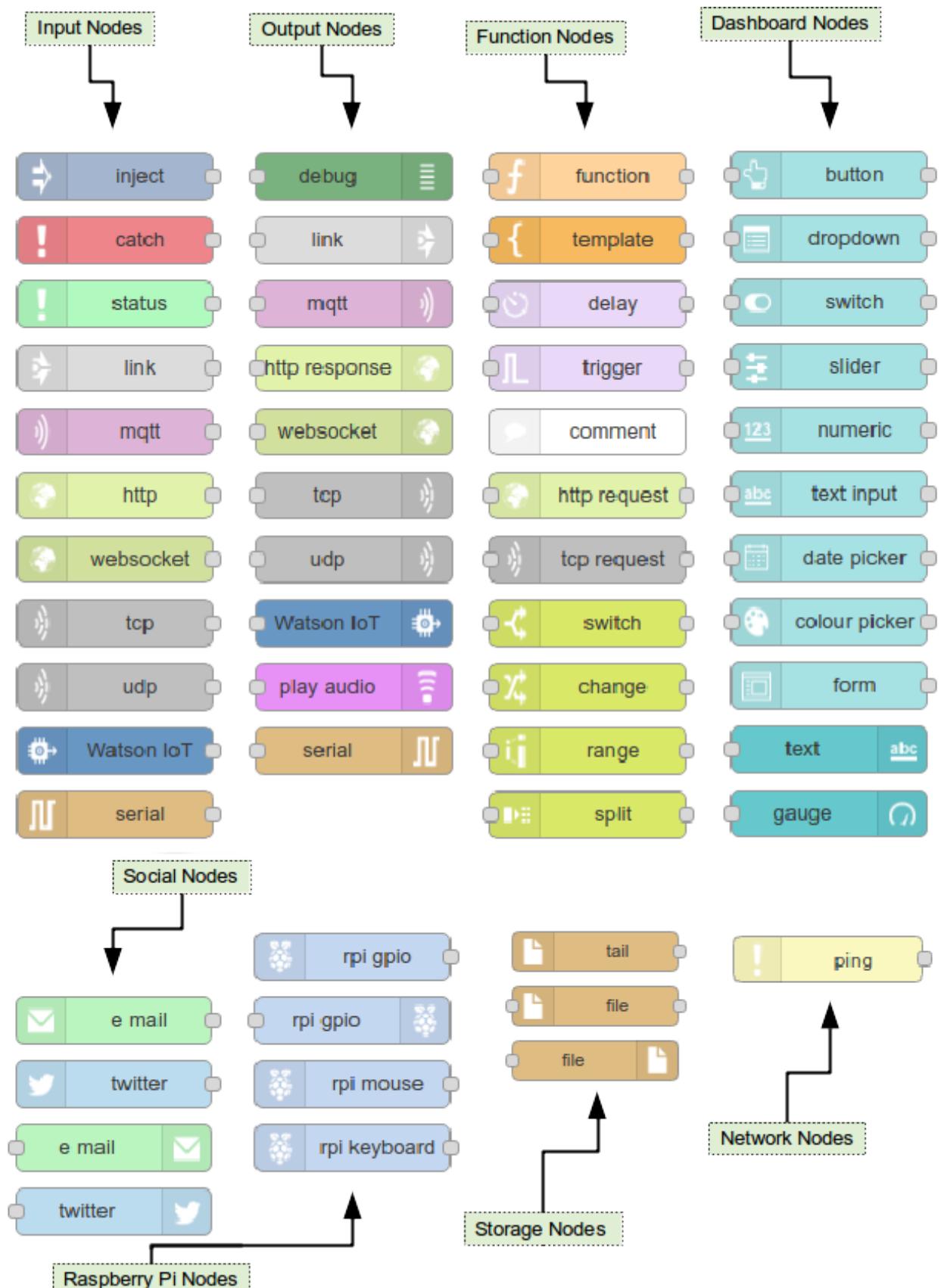


Figura 15: Node-RED - Nodes  
Fonte: (O autor, 2018)

- **payload** – Esta é a propriedade mais importante, é através dela que os dados são transferidos de um *node* para o outro. Este atributo será sempre lido pelo *node* que recebe o pacote de informação e poderá atuar sobre estes dados. O atributo *Payload* pode vir a ser de um dos seguintes tipos: *Object*, *Date*, *Boolean*, *String* ou *Number*.
- **topic** – Essa propriedade é do tipo *String* e é usada para dar uma descrição amigável do que se está fazendo com o objeto *msg*. Pode ser útil quando para depurar uma mensagem.
- **\_msgid** – É uma propriedade que contém um identificador único que é dado à mensagem. Este é muito útil no processo de depuração.

Pode-se ver o simples código de um objeto *msg* (Código 2) com suas propriedades mais importantes.

```

1  {
2    _msgid: '1db99c6d.e24664',
3    topic: '',
4    payload: {}
5  };

```

Código 2: Código do objeto msg  
Fonte: (O autor, 2018)

Usando o mesmo esquema da Figura 14 onde é possível ver um *led* conectado a um pino de GPIO do Raspberry PI e admitindo ter instalado o Node-RED no mesmo, será mostrado a seguir um exemplo de um *flow* no Node-RED que torna possível acender e apagar o *led* através do *node inject* e também fazer, através de um *twitt*, com que o *led* acenda por 2s e logo após apague. Esse exemplo faz uso dos *nodos inject* (*input-node*), *rpi-gpio out* (*output-node*), *trigger* (*function-node*) e do *twitter-in* (*social-node*). A Figura 16 mostra o arranjo e as conexões dos *nodos*.

Esse pequeno arranjo de *nodos* permite que através de uma conta do Twitter, devidamente configurada no *node twitter-in* (Figura 17), ao se twittar a palavra **ledcasa** o *led* acenda e após 2s apague. O funcionamento básico do arranjo é o seguinte, ao ser twittada a palavra-chave **ledcasa** o *node twitter-in* identifica a palavra e gera um evento de saída que alimenta a entrada do *node* do tipo *trigger*, esse *node*, ao receber o evento, envia para sua saída o valor **1** que é enviado para o *rpi-gpio out* que então envia o sinal de ligar o *led* para a GPIO do Raspberry PI. O trigger está configurado para que após 2s, sem receber nenhuma entrada, envie um valor **0** para o *rpi-gpio out* que então apaga o *led*.

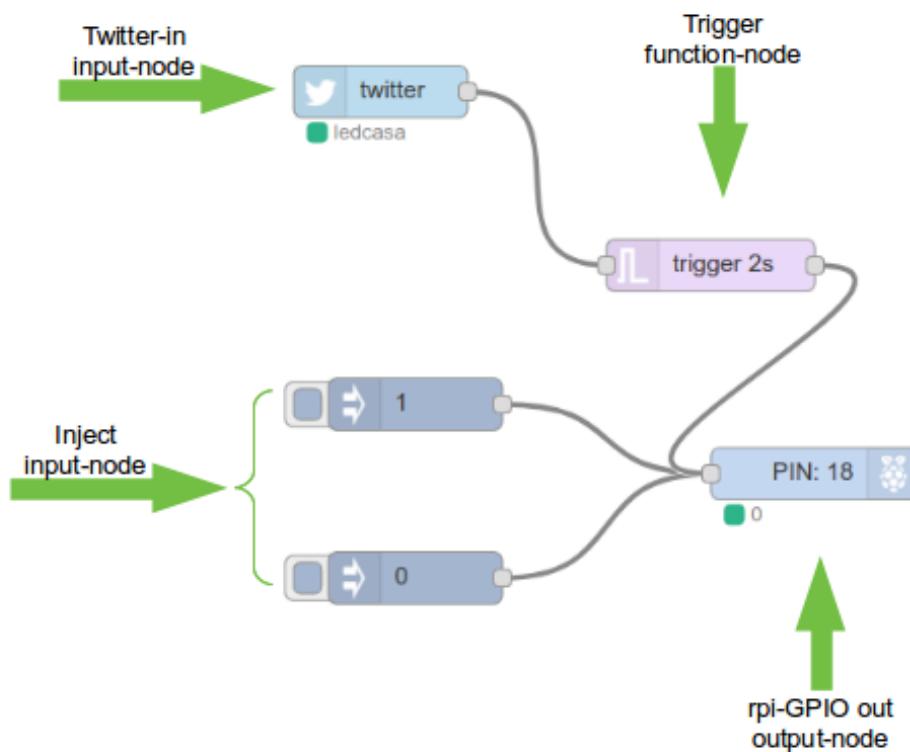


Figura 16: Flow - Controlando o *led* através do Twitter

Fonte: (O autor, 2018)

A outra ideia que esse arranjo aborda é a possibilidade de acender e apagar o *led* usando um *node* que funciona em tempo de projeto. Esse caso serve para exemplificar o uso do *input-node inject*, que permite ao ser clicado, enviar um sinal por sua conexão de saída.

Nesse caso existem dois *nodes* deste tipo, o primeiro está configurado para enviar o valor numérico **1** ao ser pressionado e o segundo o valor numérico **0**. Com isso é possível mudar o estado do pino de GPIO para **1** (ligado – primeiro *inject*) ou para **0** (desligado – segundo *inject*).

É necessário também que o *node rpi-gpio out* seja configurado para trabalhar com pino de GPIO correto, como mostra a Figura 18. Foi selecionado o pino 18 que equivale à GPIO 24, a mesma utilizada no código em python que aparece na Código 1.

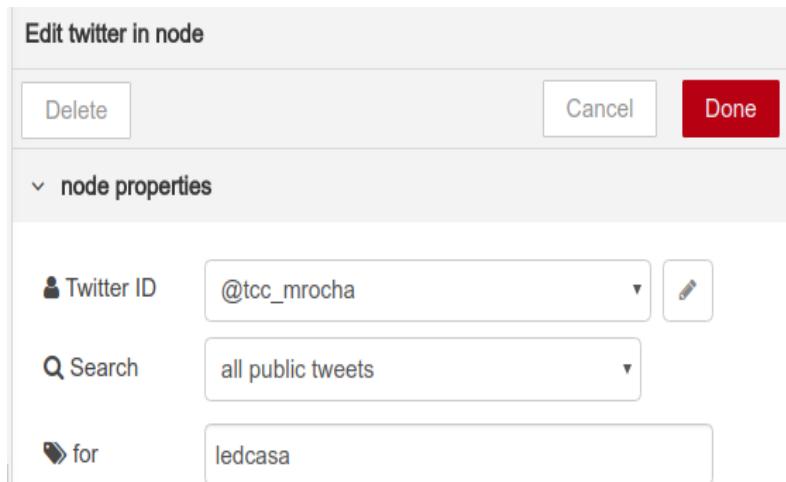


Figura 17: Node twitter-in - Configuração  
Fonte: (O autor, 2018)

Pin	3.3V Power - 1	2 - 5V Power
SDA1 - GPIO02 - 3	4 - 5V Power	
SCL1 - GPIO03 - 5	6 - Ground	
GPIO04 - 7	8 - GPIO14 - TxD	
Ground - 9	10 - GPIO15 - RxD	
GPIO17 - 11	12 - GPIO18	
GPIO27 - 13	14 - Ground	
GPIO22 - 15	16 - GPIO23	
3.3V Power - 17	18 - GPIO24	
MOSI - GPIO10 - 19	20 - Ground	
MISO - GPIO09 - 21	22 - GPIO25	
SCLK - GPIO11 - 23	24 - GPIO8 - CE0	
Ground - 25	26 - GPIO7 - CE1	
SD - 27	28 - SC	
GPIO05 - 29	30 - Ground	
GPIO06 - 31	32 - GPIO12	
GPIO13 - 33	34 - Ground	
GPIO19 - 35	36 - GPIO16	
GPIO26 - 37	38 - GPIO20	
Ground - 39	40 - GPIO21	

Type: Digital output

Figura 18: Configuração - rpi-gpio out  
Fonte: (O autor, 2018)

É importante ressaltar que há uma infinidade de *nodes* disponíveis e o processo de instalação é bem simples.

## 3.2 ARQUITETURAS

Neste capítulo e no próximo, serão apresentadas as arquiteturas para IoT propostas neste trabalho. Cada qual com seus componentes de hardware e software, fazendo uso de tudo que foi apresentado e descrito nos capítulos anteriores.

### 3.2.1 Arduino com shield ethernet

A primeira arquitetura tem como componentes de hardware o Arduino, o shield de ethernet W5100 (Figura 19) e o sensor de temperatura LM35. Os componentes de software utilizados são a IDE do Arduino usando a linguagem C++ e em termos de interface web será utilizada a plataforma ThingSpeak.

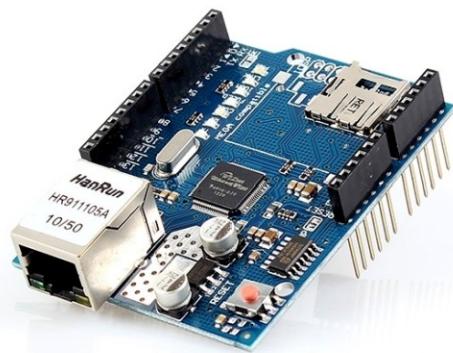


Figura 19: Ethernet Shield W5100  
Fonte: (THOMSEN, 2014)

Um esquema é montado com o objetivo de ler a temperatura através do sensor LM35, pegar essa informação e enviá-la para Internet usando o W5100. O *shield* será conectado a um roteador através de um cabo de rede padrão RJ-45. O ThingSpeak receberá essa informação através de um canal devidamente configurado e plotará um gráfico da temperatura em função do tempo. A Figura 21 mostra o esquema eletrônico.

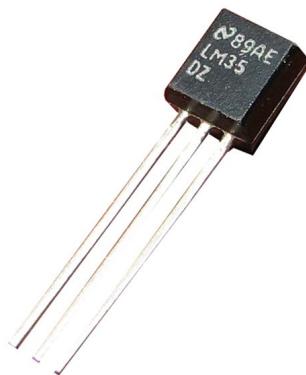


Figura 20: Sensor LM35  
Fonte: (MOTA, 2017)

O sensor LM35 (Figura 20) é um sensor de precisão, funciona de maneira linear e proporcional à temperatura. A sua saída apresenta 10mV para cada grau Celsius.

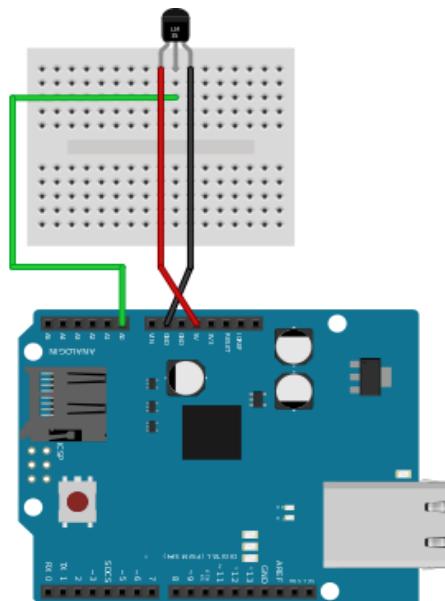


Figura 21: Esquema eletrônico - Arduino/Ethernet/LM35  
Fonte: (O autor, 2018)

O LM35 pode funcionar de dois modos. O primeiro modo, que será usado nesta arquitetura, é indicado na Figura 22 e o segundo modo na Figura 23.

**Basic Centigrade Temperature Sensor  
(2°C to 150°C)**

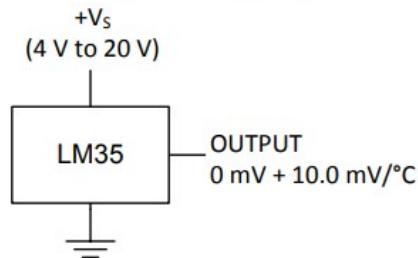
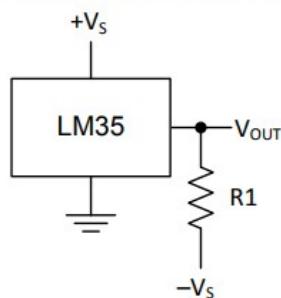


Figura 22: LM35 - Modo 1

Fonte: (MOTA, 2017)

**Full-Range Centigrade Temperature Sensor**



Choose  $R_1 = -V_S / 50 \mu\text{A}$

$V_{OUT} = 1500 \text{ mV}$  at  $150^\circ\text{C}$

$V_{OUT} = 250 \text{ mV}$  at  $25^\circ\text{C}$

$V_{OUT} = -550 \text{ mV}$  at  $-55^\circ\text{C}$

Figura 23: LM35 - Modo 2

Fonte: (MOTA, 2017)

O ThingSpeak é uma aplicação para IoT e uma API para enviar e receber dados de “coisas” usando HTTP sobre a Internet ou via rede local. É possível usar esta plataforma de duas maneiras: a primeira é pelo site [www.thingspeak.com](http://www.thingspeak.com), que tem alguma limitação para contas gratuitas<sup>44</sup> (é preciso criar uma conta) mas é o suficiente para esta arquitetura, a segunda maneira é baixar o código fonte<sup>45</sup> e instalar num servidor local. Instalando a partir do fonte, não há limitação de taxa de atualização dos dados.

<sup>44</sup> Um usuário gratuito pode atualizar um canal ThingSpeak a cada 15 segundos. Uma atualização com maior frequência resulta em erro. Para alterar as configurações a fim de atender a requisitos de frequências mais altas é preciso fazer o download do código fonte do GitHub e instalar o ThingSpeak.

<sup>45</sup> <https://github.com/iobridge/thingspeak>

The screenshot shows the 'My Channels' page on ThingSpeak. At the top left is a green 'New Channel' button. To its right is a search bar with the placeholder 'Search by tag' and a magnifying glass icon. Below the search bar is a table with one row. The table has three columns: 'Name', 'Created', and 'Updated'. The 'Name' column contains 'Arduino - LM35' with a lock icon. The 'Created' column shows '2017-05-07' and the 'Updated' column shows '2018-05-13 00:33'. Below the table is a horizontal menu with five items: 'Private' (selected), 'Public', 'Settings', 'Sharing', and 'API Keys / Data Import / Export'.

Name	Created	Updated
🔒 Arduino - LM35	2017-05-07	2018-05-13 00:33

Figura 24: Criando um canal no ThingSpeak  
Fonte: (MATHWORKS, 2018)

O elemento principal da atividade no ThingSpeak é o canal. Este canal contém campos de dados, de localização e um campo de status. É preciso criar um canal (Figura 24) no ThingSpeak e logo após já será possível gravar dados nele, processar estes dados, visualizá-los com código MATLAB e reagir a eles com tweets e outros alertas.

O nível de acesso a um canal pode ser de três tipos: privado (onde só você pode visualizar os dados), público (onde qualquer um pode ter acesso) e compartilhado com uma lista de usuários determinados. É preciso configurar o canal ThingSpeak (Figura 25), definir o nome do campo de dados *field1*, que neste caso se chamará “temperatura” e anotar o *channel ID* que será usado dentro do código do Arduino.

Uma outra informação é necessária para que seja possível enviar dados para o canal, a *Write API key* (Figura 26). Essa chave permite que dados sejam enviados para o canal especificado através do *channel ID*. Há também a *Read API key* que permite ter acesso às informações de um canal.

A maneira de enviar um dado para o canal é através de uma requisição HTTP. Ela deve seguir o seguinte padrão:

```
GET https://api.thingspeak.com/update?api_key=RBAYI4C7T5VXHQTT&field1=0
```

Private View    Public View    Channel Settings    Sharing    API Keys

## Channel Settings

Percentage complete    50%

Channel ID    269034

Name    Arduino - LM35

Description    Sensor de temperatura LM35 ligado ao Arduino.

Field 1    Temperatura

Figura 25: Configuração do canal ThingSpeak  
Fonte: (MATHWORKS, 2018)

Channel ID: 269034 | Sensor de temperatura LM35

Author: tccmrocha

Access: Public

Private View    Public View    Channel Settings    Sharing    API Keys

## Write API Key

Key    RBAYI4C7T5VXHQT

Figura 26: Channel ID e Write API Key  
Fonte: (MATHWORKS, 2018)

Para usar o ThingSpeak de maneira mais fácil no Arduino é preciso fazer a instalação da biblioteca ThingSpeak dentro da IDE do Arduino. Para isso, deve-se ir até o menu Sketch > Incluir Biblioteca > Gerenciar Bibliotecas, selecionar a biblioteca ThingSpeak e clicar em instalar (Figura 27).



Figura 27: Instalando a biblioteca ThingSpeak na IDE do Arduino  
Fonte: (O autor, 2018)

O Código 3 do Arduino, nas linhas 11, 12 e 13, importa as bibliotecas necessárias para o uso do ThingSpeak e do *shield* de ethernet.

```

1  /*
2   Publicando no ThingSpeak usando Arduino, Ethernet Shield
3   usando o sensor de temperatura LM35
4   Autor: Marcelo Marques da Rocha
5   CEDERJ/Universidade Federal Fluminense
6   TCC - Desenvolvimento Open-Source para a Internet das Coisas
7   (Arquiteturas para Interfaces Web e Móvel)
8   Orientador: Professor Dr. Luciano Bertini
9  */
10
11 #include "ThingSpeak.h"
12 #include <SPI.h>
13 #include <Ethernet.h>
```

Código 3: Importação das bibliotecas para o ThingSpeak

```

15 // define um endereço MAC para o shield ethernet
16 byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
17 EthernetClient cliente;
18 unsigned long numero_canal = 269034; // número do canal do ThingSpeak
19 const char *minha_write_api_key = "RBAYI4C7T5VXHQT"; // código da api de escrita
20 const int entrada_analogica = 0; // porta analógica para o sensor LM35
```

Código 4: Configuração do canal ThingSpeak

As próximas linhas (Código 4) definem algumas variáveis de escopo global. Na linha 16 um vetor de bytes é criado, este vetor é a definição de um endereço MAC para o *shield* de ethernet. A linha 17 cria um objeto do tipo EthernetClient. Na linha 18 é criada a variável que é inicializada com o *channel ID*, neste caso com o valor 269034. Na linha seguinte é criado um ponteiro para uma string que contém a *Write API key*. E finalmente na linha 20 uma constante é definida, essa constante é inicializada com o número da porta

analógica do Arduino, que está ligada à saída do sensor de temperatura. Como foi dito no capítulo 3.1.1 existem duas funções que devem ser obrigatoriamente definidas quando programando para o Arduino, são elas: `setup()` e `loop()`.

O bloco da função `setup()` (Código 5) vai da linha 22 à linha 26. Esse espaço é utilizado para inicializar e configurar alguns objetos e componentes do hardware do Arduino. Na linha 23 é definida a velocidade de transmissão da porta serial do Arduino, essa porta é muitas vezes usada como uma forma de depurar os programas na IDE do Arduino, tudo que é enviado para essa porta pode ser visualizado através do monitor serial da IDE. Na linha 24 ocorre a inicialização do *shield* de ethernet passando como parâmetro o número do endereço MAC. Na linha 25 o objeto ThingSpeak é inicializado tendo como parâmetro o objeto cliente.

```
22 void setup() {
23     Serial.begin(9600); // define a velocidade da comunicação serial
24     Ethernet.begin(mac);
25     ThingSpeak.begin(cliente);
26 }
```

Código 5: ThingSpeak - Inicialização

A função `loop()` (Código 6), como o nome já sugere, fica em loop infinito. Nesse ponto é que o código faz a leitura do sensor e envia para o canal. Na linha 29 a variável recebe o valor da leitura da porta analógica, este valor pode variar de 0 até 1023, já que a resolução do conversor AD do Arduino é de 10 bits. A tensão de referência para a conversão AD é de 5V, sendo assim cada unidade de leitura na porta analógica equivalerá a 4,8mV. Na linha 31 é feito o cálculo da quantidade de milivolts obtida no sensor. A linha 33 chega no valor da temperatura em graus Celsius, sabendo que cada 10mV equivalem a 1 °C. A linha 34 imprime no monitor serial o valor da temperatura (apenas para verificação local). Na linha 35 o método `writeField` do objeto ThingSpeak envia o valor da temperatura para o canal, ele recebe como parâmetros o número do canal (*channel ID*), o número do campo (neste caso o *field1*), o valor da temperatura, e a *Write API key*. Na linha 36 o comando `delay` faz com que o Arduino pause por um determinado tempo, o parâmetro que a função `delay` recebe é um valor em milissegundos. Neste caso, 1800000ms equivalem a 30 minutos. Este tempo é o intervalo entre os envios da informação para o canal ThingSpeak.

```

28 void loop() {
29     int valor_analog_lido = analogRead(entrada_analogica);
30     // 5000mv / 1024, aproximadamente 4,88mV por unidade
31     float milivolts = valor_analog_lido * 4.88;
32     // sensor sai 10mV por grau Celsius
33     float temperatura_celsius = milivolts / 10;
34     Serial.println(temperatura_celsius);
35     ThingSpeak.writeField(numero_canal, 1, temperatura_celsius, minha_write_api_key);
36     delay(1800000); // Envia as leituras com intervalo de 30min.
37 }

```

Código 6: Loop principal para leitura do sensor e publicação no ThingSpeak

O Gráfico 3.3 mostra o gráfico da temperatura plotado no site do ThingSpeak. Foram 12 horas de coleta de dados com intervalos de 30 min.

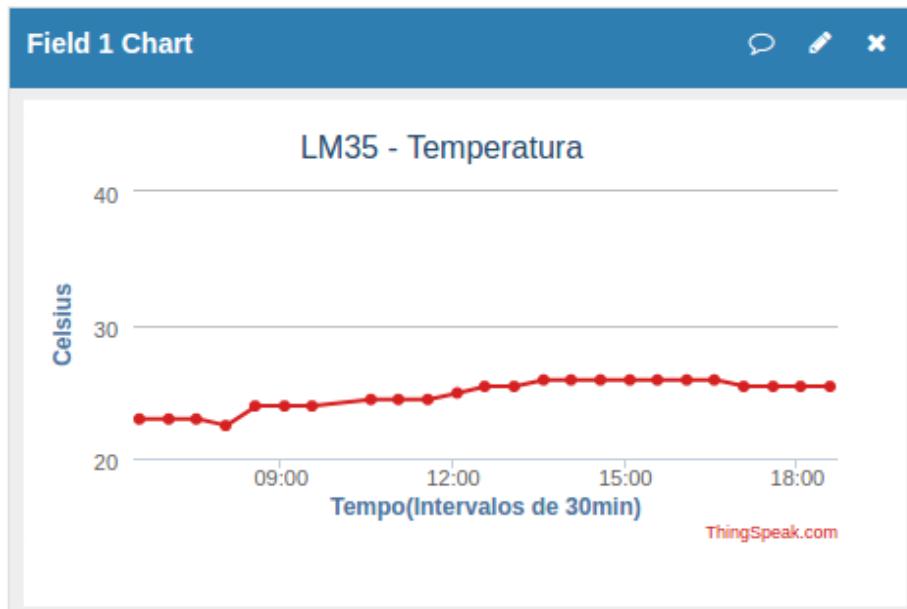


Gráfico 3.3: Gráfico da temperatura - LM35 - ThingSpeak  
Fonte: (MATHWORKS, 2018)

### 3.2.2 Arduino com WiFi via Esp8266 (comandos AT)

Nesta arquitetura a conexão com fio é abandonada e o módulo Esp8266 traz para o Arduino a capacidade de se comunicar com a rede via WiFi. Com isso, existe agora a facilidade na instalação de diversos dispositivos no ambiente, o que seria muito mais complicado e caro na conexão com cabo. O módulo Esp8266 passa a funcionar como um *shield*.

*eld* sem fio para o Arduino. Através de comandos AT o Arduino será capaz de conectar a um ponto de acesso e assim enviar os dados coletados para o canal ThingSpeak.

Os componentes de hardware para esta arquitetura são: Arduino Uno, Esp8266 (com *firmware* AT), sensor de temperatura e umidade DHT11 e mais três resistores. Os componentes de software são: IDE Arduino usando C++, comandos AT enviados para o Esp8266 e a interface web do ThingSpeak. Neste caso, será criado um novo canal no ThingSpeak com dois campos de dados (*field1* e *field2*), temperatura e umidade respectivamente. Esse procedimento já foi detalhado na seção 3.2.1.

O sensor usado nesta arquitetura é o DHT11. Um sensor digital de temperatura e umidade com 16 bits de resolução. A faixa de temperatura é de 0°C a 50°C e a faixa de umidade é de 20% a 90%. Uma característica importante deste sensor é que ele só trabalha com valores inteiros, uma limitação importante para alguns projetos de maior precisão. A Figura 28 mostra duas imagens do DHT11. Suas conexões são: Vcc, Gnd e Out.

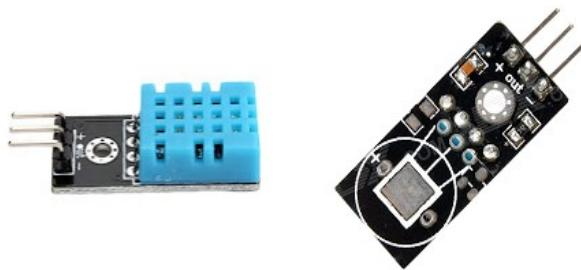


Figura 28: Sensor DHT11 e seus pinos  
Fonte: (ARDUINO&CIA, 2013)

O esquema eletrônico da arquitetura pode ser visto na Figura 29. Nela estão os componentes de hardware.

O Esp8266 é alimentado pela saída de 5 V do Arduino, que pode ser conectada na entrada Vin do Esp8266. Há um problema de compatibilidade entre as tensões de trabalho do Arduino Uno e do Esp8266. Portanto, é preciso ter cuidado ao conectar a saída serial do Arduino (TX) para a entrada serial do Esp8266 (RX). A TX do Arduino Uno trabalha com tensão de 5V enquanto o Esp8266 (TX e RX) trabalha com 3.3V. Por isso, a saída TX do Arduino precisa ser ligada a um divisor de tensão para que haja a compatibilidade. Este divisor pode ser montado usando três resistores de mesma resistência. É possível

ver na Figura 29 que a entrada RX do Esp8266 vem do divisor de tensão, que neste ponto, tem a tensão no valor de 2/3 de 5 V que é aproximadamente 3,3 V.

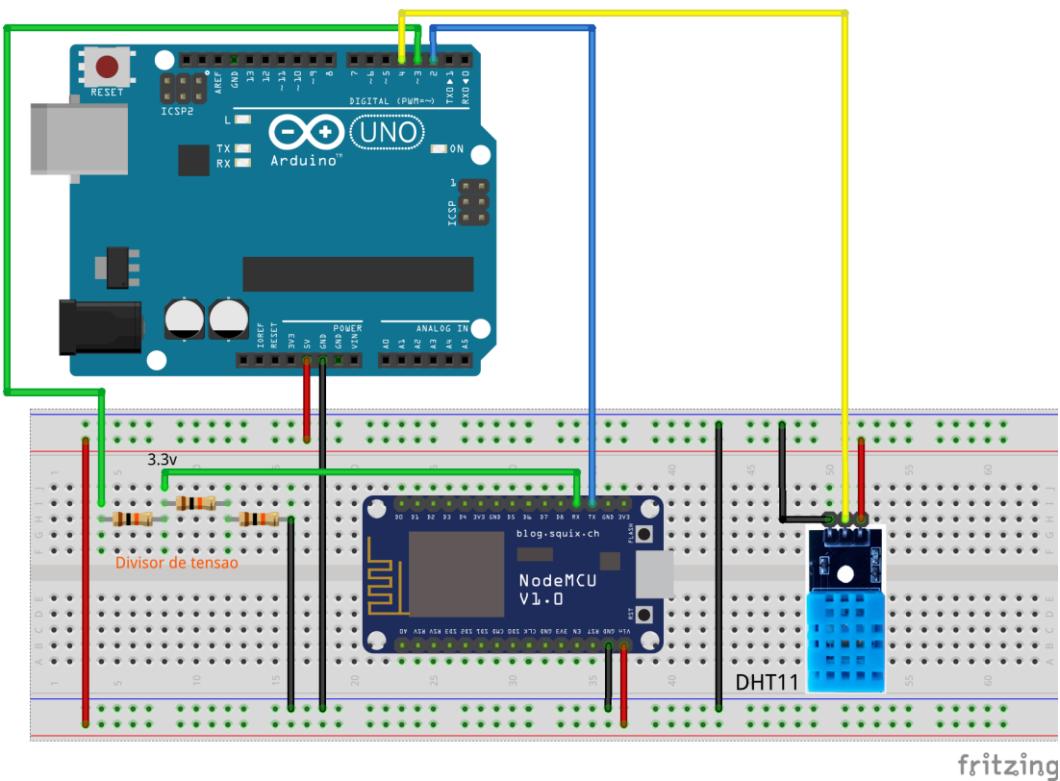


Figura 29: Esquema eletrônico - Arduino / DHT11 / Esp8266 (AT)  
Fonte: (O autor, 2018)

Existem alguns pontos importantes para que esta arquitetura venha a funcionar perfeitamente.

- O Esp8266 deve estar com o *firmware* AT instalado
- A velocidade de comunicação do módulo Esp8266 deve ser a mesma da interface software serial<sup>46</sup> do Arduino, e esta pode não funcionar muito bem com velocidades muito altas
- É preciso instalar a biblioteca DHT na IDE do Arduino da mesma maneira como a biblioteca do ThingSpeak foi instalada (Figura 27)

Normalmente o Esp8266 já vem com o *firmware* AT. No Apêndice A, existe uma explicação detalhada de como fazer o *upload* do *firmware* para o Esp8266. Geralmente, o *firmware* vem de fábrica configurado para a velocidade de 115200 *bauds*<sup>47</sup>. No Arduino a

<sup>46</sup> A biblioteca SoftwareSerial torna possível a comunicação serial através de outros pinos digitais do Arduino, e usa software para emular uma porta serial por hardware.

<sup>47</sup> Unidade de velocidade de transmissão igual ao número de vezes que um sinal muda de estado por segundo. Um baud é equivalente a um bit por segundo.

interface software serial pode não funcionar bem nessa velocidade, sendo assim, seria ideal baixar a velocidade do Esp8266. Esta arquitetura usa a velocidade de 9600 *bauds* para a comunicação serial por software e por hardware.

O código do Arduino (Código 7) só pode ser compilado após a instalação da biblioteca DHT. O código começa importando as bibliotecas e definindo o pino 4 como sendo a entrada digital que será ligada ao sensor, define também o tipo do DHT, já que a biblioteca pode trabalhar com os modelos DHT11 e DHT22. São definidos também os pinos digitais que serão usados pela interface software serial para emular uma porta serial por software, pino 2 RX (recepção) e pino 3 TX (transmissão). Por último é configurado o intervalo de tempo (1h) em que as informações serão enviadas para o ThingSpeak.

```

10 #include <SoftwareSerial.h>
11 #include <DHT.h> // importa a biblioteca DHT ver. 1.0
12
13 #define PINO_DHT 4 // define o pino que esta o dht11
14 #define TIPO_DHT DHT11 // define o tipo do dht, nesse caso o dht11
15
16 DHT dht(PINO_DHT, TIPO_DHT); // configura o sensor
17
18 #define RX 2 // pino rx SoftwareSerial
19 #define TX 3 // pino tx SoftwareSerial
20 // intervalo entre os envios das leituras (1h) em milisegundos
21 #define INTERVALO_DE_ENVIO 3600000

```

Código 7: Definição dos pinos do DHT11

Das linhas 23 à 29 (Código 8) são criadas variáveis do tipo string que serão usadas como parâmetros em outras funções, na linha 31 é criado um objeto do tipo “Software Serial” que recebe os pinos RX e TX como parâmetros.

Nas linhas 34 e 36 (Código 9), respectivamente, é feita a configuração da porta serial (por hardware), que será usada para mostrar (no monitor serial) os comandos AT que estão sendo enviados para o Esp8266 e suas devidas respostas, também é feita configuração da outra porta serial (por software), que efetivamente se comunicará com o módulo WiFi.

```

23 String ssid    = "August Rush"; // ssid do ponto de acesso
24 String senha   = "7036c0014f"; // senha
25 String api     = "OVKABCP4PXA85AEG"; // api write key do canal ThingSpeak
26 String host    = "api.thingspeak.com";
27 String porta   = "80";
28 String campo1 = "field1"; // campo da temperatura definido no canal thingspeak
29 String campo2 = "field2"; // campo da umidade definido no canal thingspeak
30
31 SoftwareSerial esp8266(RX,TX); // cria o objeto SoftwareSerial

```

Código 8: Configuração do acesso à rede, host e campos do canal ThingSpeak

Nas linhas 38, 39 e 41 a função `envia_Commando` é usada para enviar uma sequência de comandos AT para o Esp8266, o primeiro comando AT apenas manda o comando “AT” e espera receber um OK como resposta, logo após, o comando “AT+CWMODE=1” é enviado, esse comando configura o Esp8266 para o modo *station* permitindo assim ao Esp8266 se conectar a um ponto de acesso. O comando seguinte, da linha 41, envia o comando “AT+CWJAP=” concatenado com as variáveis `ssid` e `senha`, esperando um OK como resposta. O OK vai indicar que a conexão ao ponto de acesso foi um sucesso. Após a concatenação, a saída do comando fica assim: AT+CWJAP="August Rush","7036c0014f".

```

33 void setup(){
34     Serial.begin(9600); // velocidade da HardwareSerial
35     // a velocidade do esp8266 deve estar setada para 9600 bauds
36     esp8266.begin(9600); // velocidade da SoftwareSerial
37     // configura o wifi do modulo esp8266
38     envia_Commando("AT", 5, "OK"); // envia comando "AT" e espera OK como resposta
39     envia_Commando("AT+CWMODE=1", 5, "OK"); // seta o wifi para o modo station
40     // conecta ao ponto de acesso
41     envia_Commando("AT+CWJAP=\""+ ssid + "\",\"", senha +"\"", 20, "OK");
42 }

```

Código 9: Configuração da velocidade das portas seriais

Dentro da função `loop()` (Código 10) são definidas as variáveis `umidade` e `temperatura` e são do tipo `float`. A linha 47 faz a leitura do sensor usando a função `le_Sensor` que coloca o resultado da leitura nas variáveis `umidade` e `temperatura`. Na linha 48 é criada a string que é a requisição HTTP, citada na primeira arquitetura, que é usada para enviar uma informação para o canal ThingSpeak. Após a concatenação, a variável `string_get` contém a seguinte string:

```
GET /update?api_key=OVKABCP4PXA85AEG&field1=27.00&field2=64.00
```

Esta requisição HTTP será enviada ao canal ThingSpeak contendo a *Write API key*, os dois campos (*field1* e *field2*) e seus respectivos valores.

Da linha 50 à 54 são executados quatro comandos AT da camada TCP/IP (Tabela 3). O comando “AT+CIPMUX=1” habilita múltiplas conexões, o comando “AT+CIPSTART” estabelece uma conexão TCP, seus parâmetros são: tipo, endereço do host, porta. Estas informações são concatenadas ao comando e são envidas ao Esp8266.

O comando seguinte é o “AT+CIPSEND”, que define o tamanho da informação (em bytes) que será enviada para o host. Este comando recebe apenas um parâmetro, a quantidades de bytes.

No caso da linha 52 a quantidade de bytes é igual ao número de caracteres de `string_get + 4`, estes quatro bytes a mais são por conta dos caracteres `\r\n` que são automaticamente acrescentados após o conteúdo de `string_get` no comando seguinte, o `esp8266.println(string_get)`. Para encerrar o envio o comando “AT+CIPCLOSE=0” fecha a conexão. O último comando é o delay que é responsável pelo intervalo nos envios da informação para o ThingSpeak.

```
44 void loop(){
45     float umidade; // umidade
46     float temperatura; // temperatura
47     le_Sensor(&umidade, &temperatura); // faz uma leitura
48     String string_get = "GET /update?api_key=" + api + "&" + campo1 +
49     "=" + String(temperatura) + "&" + campo2 + "=" + String(umidade);
50     envia_Commando("AT+CIPMUX=1", 8, "OK");
51     envia_Commando("AT+CIPSTART=0,\\"TCP\"," + host + "\",," + porta, 15, "OK");
52     envia_Commando("AT+CIPSEND=0," + String(string_get.length() + 4), 4, ">");
53     esp8266.println(string_get);
54     envia_Commando("AT+CIPCLOSE=0", 8, "OK");
55     delay(INTERVALO_DE_ENVIO);
56 }
```

Código 10: Envio dos dados lidos do sensor DHT11 para o ThingSpeak

A função `le_Sensor` (Código 11) é bem simples, ela recebe como parâmetros dois ponteiros para variáveis do tipo `float` (passagem por referência). Dentro da função, estes ponteiros recebem os valores da umidade e da temperatura do sensor DHT11, que resultam das funções `readHumidity()` e `readTemperature()` da biblioteca DHT.

```

58 void le_Sensor(float *umidade, float *temperatura){
59     *umidade = dht.readHumidity(); // le o valor da umidade
60     *temperatura = dht.readTemperature(); // le a temperatura em celsius
61 }
```

Código 11: Função que faz a leitura do sensor DHT11

```

63 void envia_Commando(String comando_at, int tempo_limite, char resposta_esperada[]){
64     //variavel que conta o numero de loops da funcao
65     // serve como medida de tempo
66     int tempo_do_comando = 0;
67     boolean encontrou = false;
68     Serial.print("Comando AT => ");
69     Serial.print(comando_at);
70     Serial.print(" ");
71     while(tempo_do_comando < tempo_limite){
72         esp8266.println(comando_at);
73         if(esp8266.find(resposta_esperada)){
74             encontrou = true;
75             break;
76         }
77         tempo_do_comando++;
78     }
79     if(encontrou == true){
80         Serial.println(" <OK>");
81     }
82     if(encontrou == false){
83         Serial.println(" <FALHA>");
84     }
85 }
```

Código 12: Função que envia os comandos AT para o Esp8266

A função `envia_Commando` (Código 12) tem três parâmetros, o primeiro é `comando_at` que é a string com o comando a ser enviado para o Esp8266. O segundo é a variável inteira `tempo_limite` que é o número de tentativas de envio do comando, ela serve como um “tempo limite” para a tentativa de enviar o comando e é baseada nas repetições do loop de envio. O terceiro parâmetro é um ponteiro para um vetor de caracteres que passa a string da resposta esperada. Na maioria dos casos essa string é um OK. Essa função tem como destaque as linhas 68, 69 e 70 que imprimem os comandos no monitor serial a fim de visualizar as respostas dos comandos. A linha 73 também merece um destaque. O método `esp8266.find(resposta_esperada)` efetua uma busca no buffer de entrada da porta serial associada ao Esp8266. Ele busca a string `resposta_esperada`, caso en-

contre, retorna o valor booleano true. Todo comando AT enviado para o Esp8266 tem uma resposta de retorno que pode ser avaliada.

A Figura 30, mostra um exemplo de conexão bem-sucedida e também do sucesso do envio da informação para o ThingSpeak no monitor serial da IDE do Arduino.



```

/dev/ttyACM0 (Arduino/Genuino Uno)
Enviar

Comando AT => AT <OK>
Comando AT => AT+CWMODE=1 <OK>
Comando AT => AT+CWJAP="August Rush", "7036c0014f" <OK>
Comando AT => AT+CIPMUX=1 <OK>
Comando AT => AT+CIPSTART=0, "TCP", "api.thingspeak.com", 80 <OK>
Comando AT => AT+CIPSEND=0, 66 <OK>
Comando AT => AT+CIPCLOSE=0 <OK>

```

Auto-rolagem Ambos, NL e CR 9600 velocidade Deleta a saída

Figura 30: Monitor serial - Comandos AT

Fonte: (O autor, 2018)

No Gráfico 3.4, é possível ver os gráficos da temperatura e umidade, mostrados no site ThingSpeak.

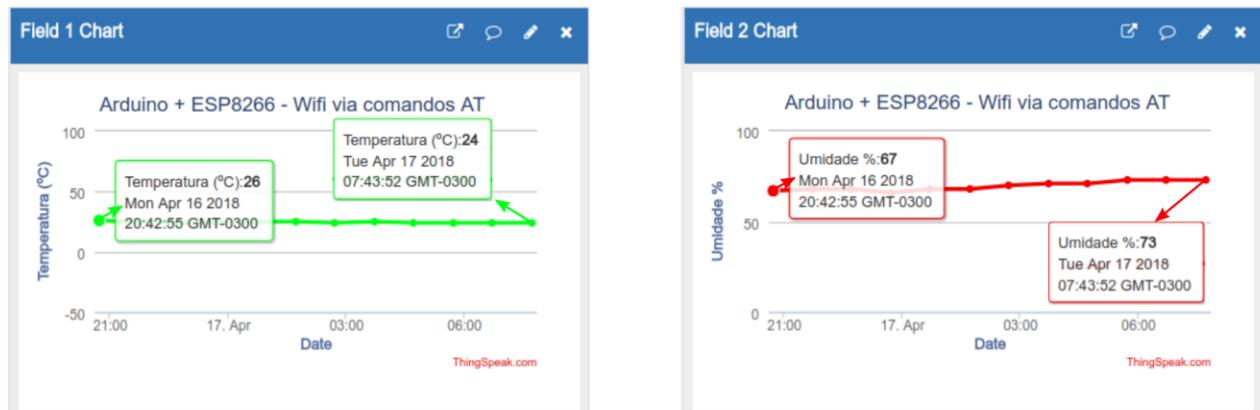


Gráfico 3.4: Gráficos de temperatura e umidade - ThingSpeak (Comandos AT)

Fonte: (MATHWORKS, 2018)

### 3.2.3 Esp8266 (programando o *firmware* com a IDE do Arduino)

Nesta arquitetura o Arduino passa a não ser mais necessário. A partir de agora será possível ver que o Esp8266 pode vir a ser muito mais do que um *shield WiFi* para o Arduino. As características dessa placa já foram mostradas na seção 3.1.2. Pode-se desenvolver um sistema bem completo apenas com Esp8266. Ele é capaz de se comunicar via porta serial, fazer leituras de sensores analógicos, se conectar a um ponto de acesso via WiFi (modo *station*) e até mesmo funcionar como um ponto de acesso (modo *access point*).

A IDE do Arduino foi criada para servir à família de placas Arduino, mas com o passar do tempo, ela foi se adaptando e hoje é capaz de funcionar com outras placas e outras famílias de microcontroladores. Com isso, hoje é possível usar a IDE do Arduino para programar o Esp8266, mas antes, é necessário configurar a IDE.

O primeiro passo é ir na IDE do Arduino, no menu Arquivo->Preferências e entrar com a seguinte url no campo URLs adicionais (Figura 31) para gerenciadores de placas:

[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

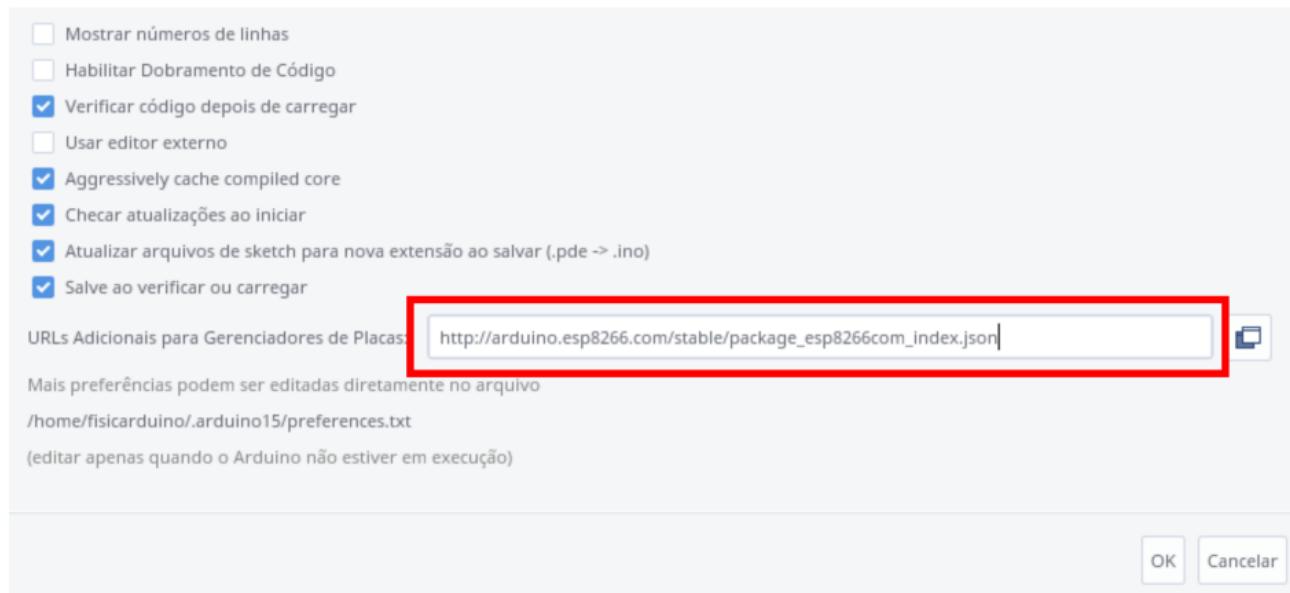


Figura 31: Configurando o Esp8266 na IDE do Arduino  
Fonte: (O autor, 2018)

Após entrar com a url é só clicar em ok. Este procedimento fará com que a IDE tenha a opção de trabalhar com o Esp8266.

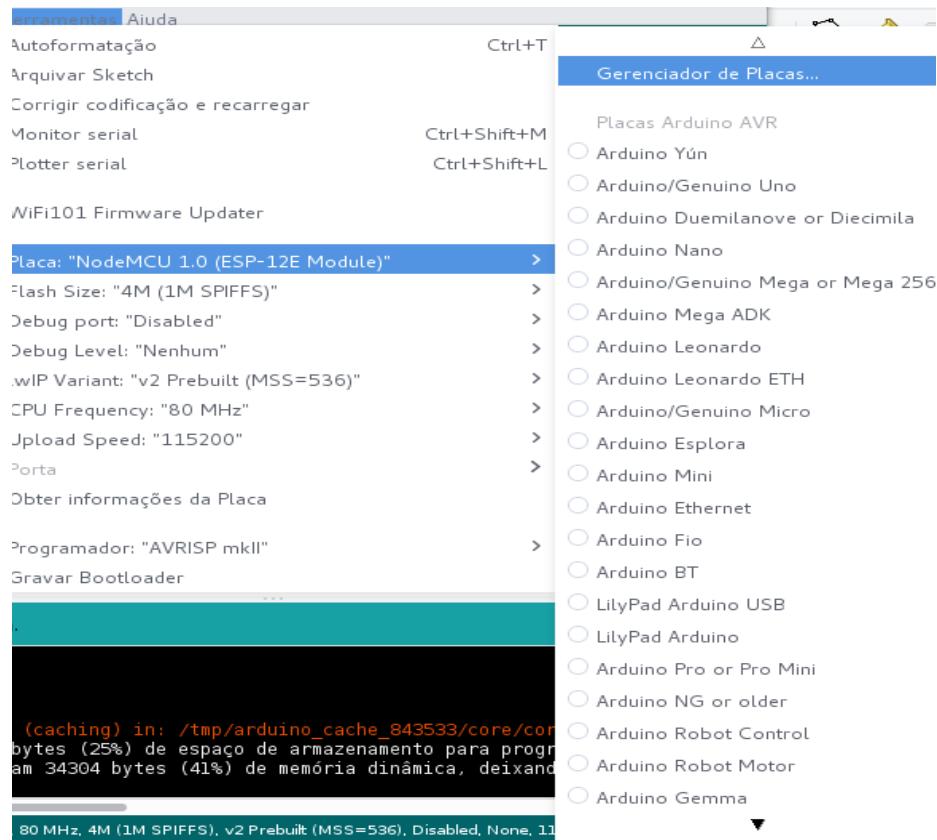


Figura 32: Instalando a placa Esp8266 na IDE do Arduino  
Fonte: (O autor, 2018)

O segundo passo é ir no menu Ferramentas->Placa->Gerenciador de Placas (Figura 32), selecionar a placa Esp8266 (Figura 33) e clicar em instalar.



Figura 33: Selecionando a placa Esp8266  
Fonte: (O autor, 2018)

A partir de agora, a placa Esp8266 fará parte da lista de opções de placas. O terceiro passo é selecionar o modelo correto do Esp8266. A opção usada nesta arquitetura é a NodeMcu 1.0 (ESP-12E Module) e pode ser vista na Figura 34. Os parâmetros de configuração da placa como Flash Size, CPU Frequency e os restantes permanecerão com os valores padrão exceto o parâmetro Porta.

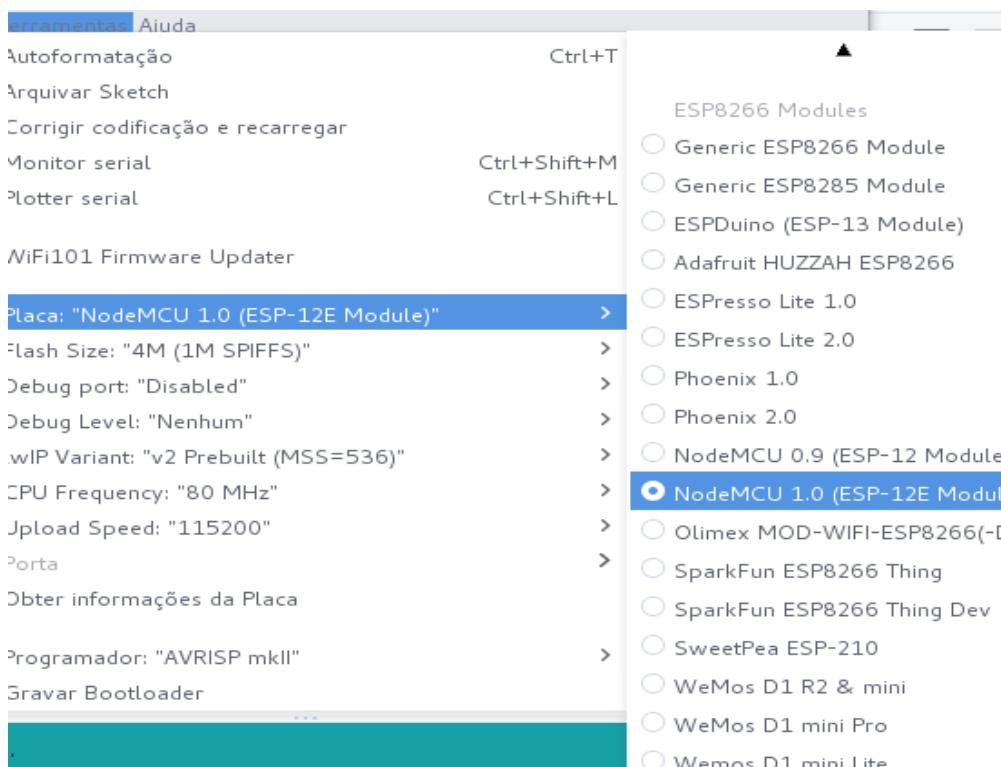


Figura 34: Selecionado o modelo do Esp8266 correto

Fonte: (O autor, 2018)

O quarto e último passo (Figura 35) é selecionar a porta em que Esp8266 aparece no sistema. É importante lembrar que todas as arquiteturas estão sendo implementadas usando GNU/Linux.

A Figura 36 mostra o esquema eletrônico da arquitetura. Ela é alimentada por uma fonte de corrente contínua de 5V. O Esp8266 trabalha com tensão de 3.3V nos seus pinos, essa tensão pode ser insuficiente para alimentar o sensor DHT11, no datasheet do sensor diz que sua tensão de alimentação deve ser de 3.5V à 5.5V, por isso o circuito será alimentado por uma fonte externa de 5V.

O Esp8266 possui uma entrada chamada Vin que permite que ele seja alimentado por uma tensão de 5V sem que ele sofra qualquer dano. Ele possui um regulador interno de 3.3V. A saída do sinal do DHT11 opera em 5V e não pode ser conectada na entrada digital do Esp8266, logo, será usado o mesmo divisor de tensão especificado na Seção

3.2.2. A entrada para o pino D0 do Esp8266 vem do divisor de tensão, e já vem com a tensão compatível com o Esp8266.

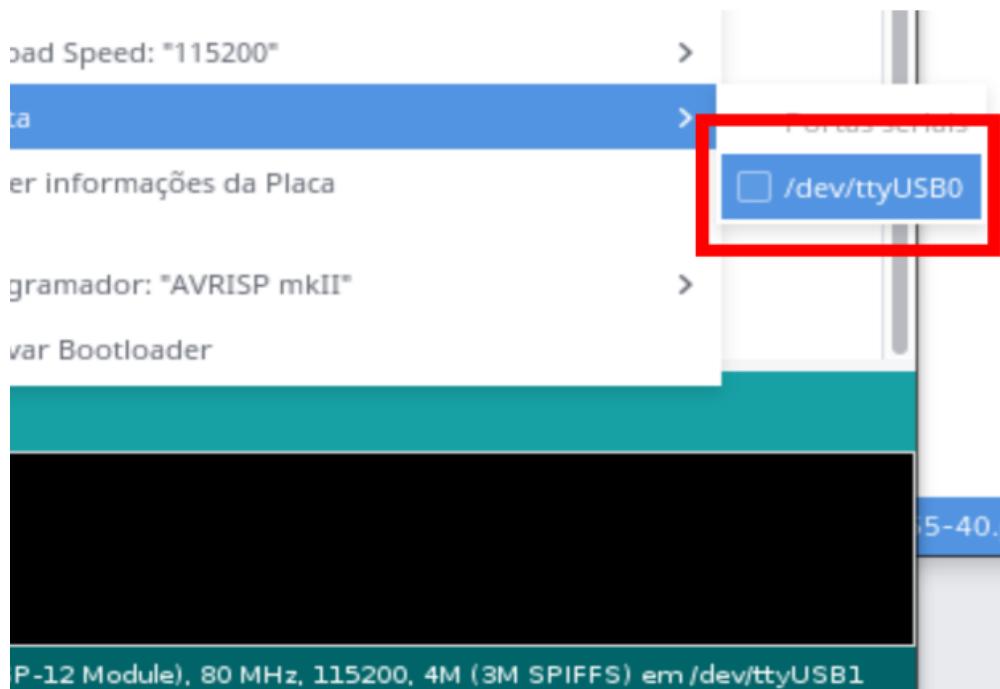


Figura 35: Selecionado a porta do Esp8266  
Fonte: (O autor, 2018)

O trecho inicial do Código 13 começa com o processo de importar as bibliotecas necessárias para a aplicação, as bibliotecas das linhas 11 e 12 já foram vistas anteriormente. Na linha 13 será usada pela primeira vez a biblioteca ESP8266WiFi, essa biblioteca traz para aplicação os recursos de conexão WiFi do Esp8266.

Das linhas 15 à 19 há o procedimento padrão de definição e inicialização de variáveis como numero\_canal, ssid, senha, host e etc. Na linha 21 há um ponto importante, a definição dos pinos do Esp8266, quando programando na IDE do Arduino, usa uma nomenclatura diferente da que vem serigrafada na placa do Esp8266.

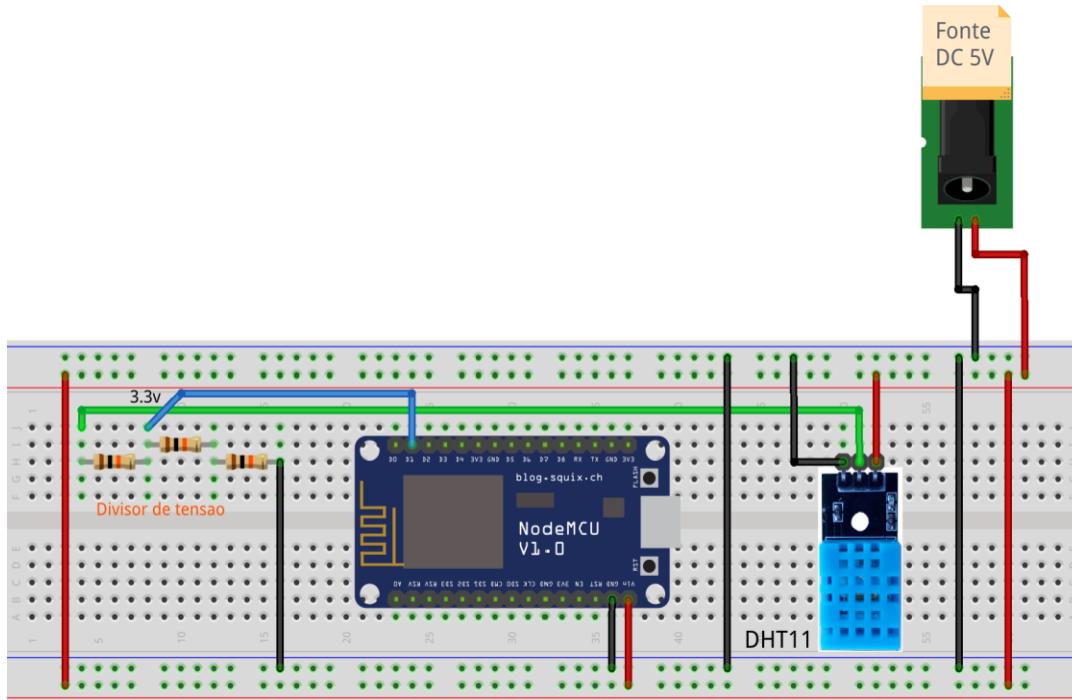


Figura 36: Esquema eletrônico - Esp8266/DHT11 - Programando em C++  
Fonte: (O autor, 2018)

```

11 #include "ThingSpeak.h"
12 #include <DHT.h>
13 #include <ESP8266WiFi.h>
14
15 unsigned long numero_canal = 495522;
16 const char* ssid = "August Rush";
17 const char* senha = "7036c0014f";
18 const char* host = "api.thingspeak.com";
19 const char* minha_write_api_key = "582V8VB0M2QZGS0N";
20
21 #define PINO_DHT 5 // pino D1 no nodemcu
22 #define TIPO_DHT DHT11
23 DHT dht(PINO_DHT, TIPO_DHT);
24
25 WiFiClient cliente; // cria um objeto cliente WiFi

```

Código 13: Importando a biblioteca WiFi para o Esp8266

A Tabela 4 mostra essa relação. Dentro do código (na IDE) a referência adotada é o número das GPIO (coluna ESP8266 pin) enquanto na placa a referência é a coluna IO index. Sendo assim, a definição do PINO\_DHT como 5 na linha 21 (Código 13) se refere à

GPIO5 que equivale ao pino 1 (D1) impresso na placa Esp8266. Na linha 25 é criado um objeto WiFiClient.

IO index	ESP8266 pin	IO index	ESP8266 pin
0[*]	GPIO16	7	GPIO13
1	GPIO5	8	GPIO15
2	GPIO4	9	GPIO3
3	GPIO0	10	GPIO1
4	GPIO2	11	GPIO9
5	GPIO14	12	GPIO10
6	GPIO12		

Tabela 4: Relação das GPIO com os pinos do Esp8266

Fonte: (NODEMCU, 2018)

O Código 14 contém a função setup() que como já é conhecida, ele faz a maior parte das inicializações do hardware interno da placa. A linha 28 define a velocidade da porta serial, a linha 30 inicializa o objeto dht, as linhas 29 e 31 usam um delay de 10ms apenas para dar um tempo para que as configurações sejam terminadas pelo hardware da placa, afinal toda operação tem um custo de tempo, mesmo que seja muito pequeno.

Na linha 32 o objeto ThingSpeak é inicializado e tem como parâmetro o objeto cliente (WiFiClient). A linha 33 tenta se conectar à rede WiFi, o bloco while (das linhas 36 à 39) fica verificando, num intervalo de 0,5 s, o valor de retorno da função WiFi.status(), se não conectou, um ponto é impresso no monitor serial e um novo teste será feito até que a conexão tenha sucesso.

O próximo fragmento de código, o da função loop() (Código 15), faz a leitura do sensor atribuindo os valores lidos às devidas variáveis. O bloco if que vai das linhas 47 à 50 testa a integridade dos valores retornados na leitura do sensor, a função isnan (*is not a number*) retorna 1 se o valor contido na variável testada não for um número, caso não seja um número, o loop reinicia e é feita outra leitura. Sendo um sucesso a leitura do sensor é hora de enviar estes dados para o site. Aqui aparecem duas novas funções (métodos) do objeto ThingSpeak.

```

27 void setup() {
28     Serial.begin(9600);
29     delay(10);
30     dht.begin();
31     delay(10);
32     ThingSpeak.begin(cliente); // inicializa a biblioteca ThingSpeak
33     WiFi.begin(ssid, senha); // configura a conexao WiFi
34     Serial.print("Conectando a ");
35     Serial.println(ssid);
36     while (WiFi.status() != WL_CONNECTED) {
37         delay(500);
38         Serial.print(".");
39     }
40     Serial.println("");
41     Serial.println("Conectado ao WiFi");
42 }
```

Código 14: Configurando e conectando ao WiFi

Na Seção 3.2.1 foi usada a função ThingSpeak.writeField() que enviava apenas um campo para o canal ThingSpeak, esta arquitetura precisa que dois campos sejam enviados. A maneira correta de proceder neste caso é usando a função ThingSpeak.setField(), que tem como parâmetros o número do campo e o valor a ser enviado. Ela faz uma associação entre campo e valor.

A outra função é a ThingSpeak.writeFields(), com os parâmetros numéricos do canal e a api de escrita. Ela é capaz de enviar de uma só vez, os campos definidos anteriormente.

Após a execução destas funções os valores de umidade e temperatura são mostrados no monitor serial para averiguação, na linha 62 é encerrada a conexão do cliente com o servidor e define-se um intervalo de 60 s até a próxima leitura e envio de dados.

```

44 void loop(){
45     float umidade = dht.readHumidity();
46     float temperatura_celsius = dht.readTemperature();
47     if (isnan(umidade) || isnan(temperatura_celsius)){
48         Serial.println("Falha ao ler do sensor DHT11");
49         return;
50     }
51     // seta os campos para enviar para o ThingSpeak
52     ThingSpeak.setField(1, temperatura_celsius);
53     ThingSpeak.setField(2, umidade);
54     // envia os dois campos de uma vez
55     ThingSpeak.writeFields(numero_canal,minha_write_api_key);
56     // imprime dados no monitor serial
57     Serial.print("Temperatura em Celsius: ");
58     Serial.println(temperatura_celsius);
59     Serial.print("Umidade : ");
60     Serial.println(umidade);
61     Serial.println("Pausa... ");
62     cliente.stop(); // encerra a conexao com o servidor
63     delay(60000); // pausa de 1 minuto
64 }
```

Código 15: Função loop() - Leitura do sensor e envio dos dados para o T. Speak

Pode-se ver o gráfico da temperatura e umidade (Gráfico 3.5) gerados pelo ThingSpeak. Os dados foram coletados com intervalos de 60 s apenas para demonstração da arquitetura.



Gráfico 3.5: Leitura do DHT - Esp8266 programado em C++  
Fonte: (MATHWORKS, 2018)

### 3.2.4 NodeMcu + ThingSpeak (usando Lua)

A partir de agora, as duas últimas arquiteturas passam a usar o kit de desenvolvimento NodeMcu, que vem a ser um Esp8266 rodando o *firmware* Lua. Para seguir nesta configuração é necessário gravar o *firmware* Lua no Esp8266, esse processo é descrito com detalhes no Apêndice A. Toda programação será feita usando a linguagem Lua com o auxílio da IDE Esplorer, o Apêndice B descreve as funcionalidades da IDE. O esquema eletrônico é o mesmo da Figura 36.

O código desta arquitetura será composto por cinco arquivos:

- init.lua
- credenciais.lua
- tabelas.lua
- conecta\_wifi.lua
- thingspeak\_http.lua

Estes arquivos podem ser chamados de módulos. A aplicação poderia ser escrita em apenas um arquivo, mas, a modularização será importante para mostrar alguns detalhes da programação Lua. O NodeMc ao iniciar, procura por um arquivo que se chama init.lua. Se ele for encontrado será então executado, se ele não existir, o NodeMcu exibirá o prompt de linha de comando e ficará aguardando que algo seja digitado. É através do init.lua que a aplicação pode ser executada automaticamente. O objetivo do init.lua é dar o *start* na aplicação principal.

O Código 16 mostra a pequena quantidade de linhas do init.lua. Ele imprime no terminal a mensagem dizendo que a aplicação será iniciada em 3 segundos, na linha 5, a função delay do objeto tmr, que recebe como parâmetro o tempo em microsegundos, pausa o NodeMcu por 3 segundos. A função tmr.delay() é muito semelhante à função delay da biblioteca Arduino, só que seu uso no ambiente Lua é desencorajado, pois essa função pára completamente o funcionamento do NodeMcu e como já foi visto, o *firmware* Lua é assíncrono e baseado em eventos, portanto, a função tmr.delay() bloqueia o NodeMcu prejudicando a sua resposta assíncrona aos eventos. Essa função só é usada no init.lua por dois motivos: o primeiro é que a aplicação principal ainda não está em execução, então, não há problema nesse pequeno bloqueio, o segundo motivo é que essa pausa de 3

segundos permite que o arquivo init.lua seja removido através do comando no prompt file.remove("init.lua"), caso haja a necessidade de reprogramar o NodeMcu. Na prática isso significa que, se não houver essa pequena pausa, o init.lua iniciará e rapidamente chamará a aplicação principal. Assim, não será mais possível conectar ao NodeMcu a fim de programá-lo, somente será possível acessá-lo se o *firmware* for regravado e com a opção “yes, wipes all data” selecionada (ver ilustração na Figura 71), o que apagará completamente a memória flash e com isso o init.lua.

```

1 print("Iniciando o NodeMcu em 3s....")
2
3 -- essa pausa permite a exclusao do arquivo init.lua, caso seja necessario
4 -- para remover usar: file.remove("init.lua")
5 tmr.delay(3000000) -- 3s em microsegundos
6
7 dofile("conecta_wifi.lua") -- carrega o arquivo que conecta ao wifi

```

Código 16: Executando automaticamente com init.lua

A linha 7 executa o comando dofile("conecta\_wifi") que executa o módulo conecta\_wifi. A função dofile() também funciona como um comando de importação, semelhante à diretiva include da linguagem C++. Se o módulo tiver apenas declarações de variáveis, de funções, de estruturas de dados, o dofile() chama o módulo e todas essas variáveis e funções passam a estar disponíveis. Agora, se o módulo for uma sequência de comandos, então esses comandos serão executados por dofile().

O próximo módulo (Código 17) se chama credenciais.lua. Ele tem uma tabela (linha 3) e nas linhas 4 e 5 são criados dois índices para essa tabela e seus respectivos valores associados. O índice ssid e o pwd representam o ssid da rede e sua senha. Tabelas em Lua são arrays associativos implementados eficientemente como uma combinação de array e hash (tabela de dispersão) e podem ter qualquer valor como índice menos *nil* (sem valor em Lua), podem haver inclusive índices negativos. Esse módulo será importado mais em frente e o uso da tabela ficará mais claro.

O módulo tabelas.lua (Código 18) é semelhante ao módulo anterior, sua função é apenas criar tabelas que serão usadas em outros módulos. São criadas três tabelas, uma com os modos do WiFi, outra com os estados da conexão WiFi e a terceira com os motivos de erros com o MQTT *broker*. O código mostra que, em Lua, há mais de uma maneira de criar uma tabela e declarar seus índices e valores associados. É importante destacar a presença de índices negativos (linhas 21 à 25), uma característica da linguagem Lua.

```

1 -- arquivo de configuracao
2 -- ssid e senha
3 config = {}
4 config.ssid = "August Rush"
5 config.pwd = "7036c0014f"

```

Código 17: Módulo credenciais.lua

O Código 19, referente ao módulo de nome conecta\_wifi.lua é chamado pelo módulo init.lua. O seu início importa os já conhecidos módulos tabela.lua e credenciais.lua. A linha 7 coloca o WiFi no modo *station*, o que significa que o NodeMcu será capaz de se

```

1 -- tabela com os modos do wifi
2 modo_wifi = {}
3 modo_wifi[1] = 'wifi.STATION'
4 modo_wifi[2] = 'wifi.SOFTAP'
5 modo_wifi[3] = 'wifi.STATIONAP'
6 modo_wifi[4] = 'wifi.NULLMODE'

7
8 -- tabela com os estados da conexao wifi
9 estado_wifi = {}
10 estado_wifi[0] = 'wifi.STA_IDLE'
11 estado_wifi[1] = 'wifi.STA_CONNECTING'
12 estado_wifi[2] = 'wifi.STA_WRONGPWD'
13 estado_wifi[3] = 'wifi.STA_APNOTFOUND'
14 estado_wifi[4] = 'wifi.STA_FAIL'
15 estado_wifi[5] = 'wifi.STA_GOTIP'

16
17
18 -- tabela com as razoes dos erros de conexao com o mqtt broker
19 -- 0 esta OK - nunca levanta uma excecao de numero 0
20 err_conn = {}
21 err_conn[-5] = 'mqtt.CONN_FAIL_SERVER_NOT_FOUND'
22 err_conn[-4] = 'mqtt.CONN_FAIL_NOT_A_CONNACK_MSG'
23 err_conn[-3] = 'mqtt.CONN_FAIL_DNS'
24 err_conn[-2] = 'mqtt.CONN_FAIL_TIMEOUT RECEIVING'
25 err_conn[-1] = 'mqtt.CONN_FAIL_TIMEOUT_SENDING'
26 err_conn[0] = 'mqtt.CONNACK_ACCEPTED'
27 err_conn[1] = 'mqtt.CONNACK_REFUSED_PROTOCOL_VER'
28 err_conn[2] = 'mqtt.CONNACK_REFUSED_ID_REJECTED'
29 err_conn[3] = 'mqtt.CONNACK_REFUSED_SERVER_UNAVAILABLE'
30 err_conn[4] = 'mqtt.CONNACK_REFUSED_BAD_USER_OR_PASS'
31 err_conn[5] = 'mqtt.CONNACK_REFUSED_NOT_AUTHORIZED'

```

Código 18: Módulo tabela.lua

conectar a um ponto de acesso. A linha 8 configura o ssid e senha passando como parâmetro a tabela config que foi criada dentro do módulo credenciais.lua. A linha 12 usa o objeto tmr (timer), o mesmo que foi usado no módulo init.lua, só que agora ele é usado de uma maneira bem interessante.

O NoceMcu possui 7 timers numerados de 0 à 6. Eles podem funcionar de três modos:

- **tmr.ALARM\_SINGLE** – Que dispara apenas uma vez e não necessita da função tmr.unregister().
- **tmr.ALARM\_SEMI** – Que dispara uma vez mas pode ser reiniciada pela função tmr.start().
- **tmr.ALARM\_AUTO** – Que repete automaticamente.

Para programar os timers usa-se a função tmr.alarm() e ela recebe os parâmetros id (de 0 à 6), intervalo em milisegundos, modo, func(). O id é o número do timer utilizado, o intervalo de tempo é o tempo em que o timer vai esperar para disparar, o modo é um dos três citados anteriormente, e a função é uma função que será executada pelo timer no intervalo determinado. Para o timer, pode ser passada uma função (com nome) definida em alguma parte do módulo ou pode ser usada uma função anônima que é definida dentro do parâmetro função de tmr.alarm().

Na linha 12 (Código 19) pode-se ver que o timer utilizado é o 2 e está configurado assim: intervalo de tempo de 1 segundo, modo automático (tmr.ALARM\_AUTO) e que recebe uma função anônima para ser executada periodicamente.

O bloco dessa função vai da linha 13 à 26. Na linha 13 verifica-se se a placa tem um IP, se esse valor for *nil* então será impresso no monitor serial o estado do WiFi (baseado na tabela estado\_wifi) e a função será reiniciada no segundo seguinte. Mas, se o NodeMcu tiver um IP válido a função então imprimirá o estado da rede (conectado), o endereço de IP e o MAC address do NodeMcu. Na linha 23 o timer é desativado e o seu id fica liberado para a programação de outro timer. Na linha 24 o módulo principal é então carregado e executado.

As linhas de 9 à 12 (Código 20) declaram variáveis já conhecidas de arquiteturas anteriores e cujo os nomes são autoexplicativos. A linha 15 faz uma primeira leitura do sensor que retorna um tupla e as variáveis recebem os valores na devida ordem.

Na linha 41 (Código 21) o timer 1 é registrado para funcionar em intervalos de 15 segundos no modo de repetição automática e como corpo de sua função anônima ele

executa a linha 43, que chama a função postThingSpeak(), que será explicada mais em frente, e na linha 44 chama a função le\_DHT11() que faz mais uma leitura do sensor. O timer será disparado novamente nos próximos 15 segundos até que a aplicação seja interrompida.

```

1  -- conecta ao wifi
2  -- conectando ao roteador
3  dofile("tabelas.lua") -- importa tabelas
4  -- importa tabela config com ssid e senha def. no arq. credenciais.lua
5  dofile("credenciais.lua")
6
7  wifi.setmode(wifi.STATION) -- define o modo da conexao wifi
8  wifi.sta.config(config) -- configura a conexao
9
10 print("Conectando ao wifi...")
11 -- tenta conectar a cada 1s
12 tmr.alarm(2, 1000, tmr.ALARM_AUTO, function ()
13     if wifi.sta.getip() == nil
14         then
15             print("Estado WIFI... " .. estado_wifi[wifi.sta.status()])
16             print("IP indisponivel, aguardando...") -- falha ao conectar
17         else
18             meu_ip = wifi.sta.getip()
19             meu_mac = wifi.sta.getmac()
20             print("Estado WIFI... " .. estado_wifi[wifi.sta.status()])
21             print("Conectado, o IP é " .. meu_ip)
22             print("O End. MAC é " .. meu_mac)
23             tmr.unregister(2) -- libera o timer
24             dofile("thingspeak_http.lua") -- carrega o firmware principal
25     end
26 end)

```

Código 19: Módulo conecta\_wifi.lua

A linha 17 do Código 22 declara a função. Na linha 18 um objeto cliente HTTP é criado, na linha 19 é definida uma função *callback* para o evento receive do cliente. Assim que o cliente receber alguma informação o corpo da função *callback* será executado. A função recebe dois parâmetros, o primeiro é o próprio objeto cliente e o segundo é uma string que guardará a resposta enviada pelo servidor. Dentro do corpo da *callback* a string com a resposta é varrida (função string.find) em busca de uma indicação de sucesso da parte do servidor, a string buscada é “Status: 200 OK” que indica o sucesso na resposta do servidor. Em caso de sucesso uma mensagem de OK é impressa no monitor serial, caso contrário nada ocorre.

Na linha 26 uma nova função *callback* é definida, desta vez para o evento *connection* que é disparado quando o cliente se conecta ao servidor. Essa função tem o mesmo formato da anterior, ou seja, recebe primeiro o objeto cliente e depois uma string que armazena a resposta do servidor. Essa string é opcional, como não há interesse na string de resposta, ela não será utilizada.

```

1  --Publicando no ThingSpeak usando NodeMcu e Lua
2  --Autor: Marcelo Marques da Rocha
3  --CEDERJ/Universidade Federal Fluminense
4  --TCC - Desenvolvimento Open-Source para a Internet das Coisas
5  --(Arquiteturas para Interfaces Web e Móvel)
6  --Orientador: Professor Dr. Luciano Bertini
7
8  -- variaveis locais
9  local pin_DHT11 = 1 -- porta do sensor
10 local write_api_key = "0XL2LID040NNT7EJ"
11 local timer = 1 -- define o timer
12 local intervalo = 15000 -- intervalo de atualizacao do thingspeak
13
14 -- faz a primeira leitura
15 status,temp_celsius,umidade,temp_decimal,umi_decimal = dht.read11(pin_DHT11)
```

Código 20: Módulo thingspeak\_http.lua - Declarando variáveis

Na linha 29, o método *send* do objeto cliente envia a string com a requisição HTTP contendo: a api de escrita, temperatura e umidade no formato adequado. Isso já foi mostrado anteriormente. A concatenação de strings em Lua é feita usando .. (dois pontos seguidos).

Por último, ainda dentro da função *postThingSpeak()*, o método *connect* do objeto cliente é chamado tendo como parâmetros a porta padrão para serviços HTTP (porta 80) e a string *api.thingspeak.com*. A partir desse comando um evento *connection* ocorre, o cliente então responde ao evento enviando a requisição HTTP.

O servidor responde e um outro evento ocorre, só que desta vez é um evento recebido. Então, o cliente responde a este evento executando o código que verifica o conteúdo da resposta. E assim, o código vai executando indefinidamente em intervalos de 15 segundos.

```

41 tmr.alarm(timer, intervalo, tmr.ALARM_AUTO,
42     function()
43         postThingSpeak() -- envia os campos para o thingspeak
44         le_DHT11() -- faz outra leitura do sensor
45     end)
46
47 -- funcao que le o sensor dht11
48 function le_DHT11()
49     status,temp_celsius,umidade,temp_decimal,umi_decimal = dht.read11(pin_DHT11)
50     print ("Temperatura (C): "..temp_celsius.." , Humidade (%): "..umidade)
51 end

```

Código 21: Configurando o timer para o envio de dados

```

17 function postThingSpeak()
18     cliente = net.createConnection(net.TCP, 0)
19     cliente:on("receive",
20         function(cliente, string_resposta)
21             if (string.find(string_resposta, "Status: 200 OK") ~= nil) then
22                 print("Posted OK");
23             end
24         end)
25
26     cliente:on("connection",
27         function(cliente)
28             print ("Posting...");
29             cliente:send("GET /update?api_key"..write_api_key
30                 .. "&field1"..temp_celsius.."&field2"..umidade
31                 .. " HTTP/1.1\r\n"
32                 .. "Host: api.thingspeak.com\r\n"
33                 .. "Connection: close\r\n"
34                 .. "Accept: */*\r\n"
35                 .. "User-Agent: Mozilla/4.0 (compatible; esp8266 Lua; Windows NT 5.1)\r\n"
36                 .. "\r\n")
37         end)
38         cliente:connect(80, 'api.thingspeak.com')
39     end

```

Código 22: Postando no ThingSpeak - Cliente HTTP

## 4 IMPLEMENTANDO UMA CASA INTELIGENTE

O projeto da casa inteligente é a implementação da quinta arquitetura. Ela faz uso de praticamente tudo que foi visto nas arquiteturas anteriores. Apesar de usar muitas das tecnologias já apresentadas até o momento, esta arquitetura acrescenta novos componentes de hardware e de software.

Componentes de hardware adicionados:

- *LED*
- PIR<sup>48</sup> – Sensor de movimento
- DHT22 – Sensor de temperatura e umidade
- Sensor ultrassônico

Componentes de software adicionais:

- Protocolo MQTT
- Servidor HTTP
- Servidor Telnet

Nas arquiteturas anteriores, durante a interação com o ThingSpeak, a comunicação era basicamente unidirecional, ou seja, a informação era enviada do Arduino ou Esp8266 para o site. Em nenhum momento o ThingSpeak controlou os dispositivos, não que isso não pudesse ser feito, mas esse não era o foco deste trabalho. A aplicação da casa inteligente implementa uma interação bidirecional. O hardware continua coletando dados do ambiente e enviando para o servidor, mas, a aplicação que está no servidor passa a ter o controle sobre dispositivos.

A proposta desta arquitetura é implementar um software de controle que é capaz de ser acessado remotamente a partir de um computador, celular ou tablet. Essa aplicação vai interagir com uma rede de nós NodeMcu instalados pelos cômodos de uma casa.

---

<sup>48</sup> Sensor infravermelho passivo.

Cada NodeMcu terá suas funções bem definidas de acordo com o ambiente em que ele se encontrar.

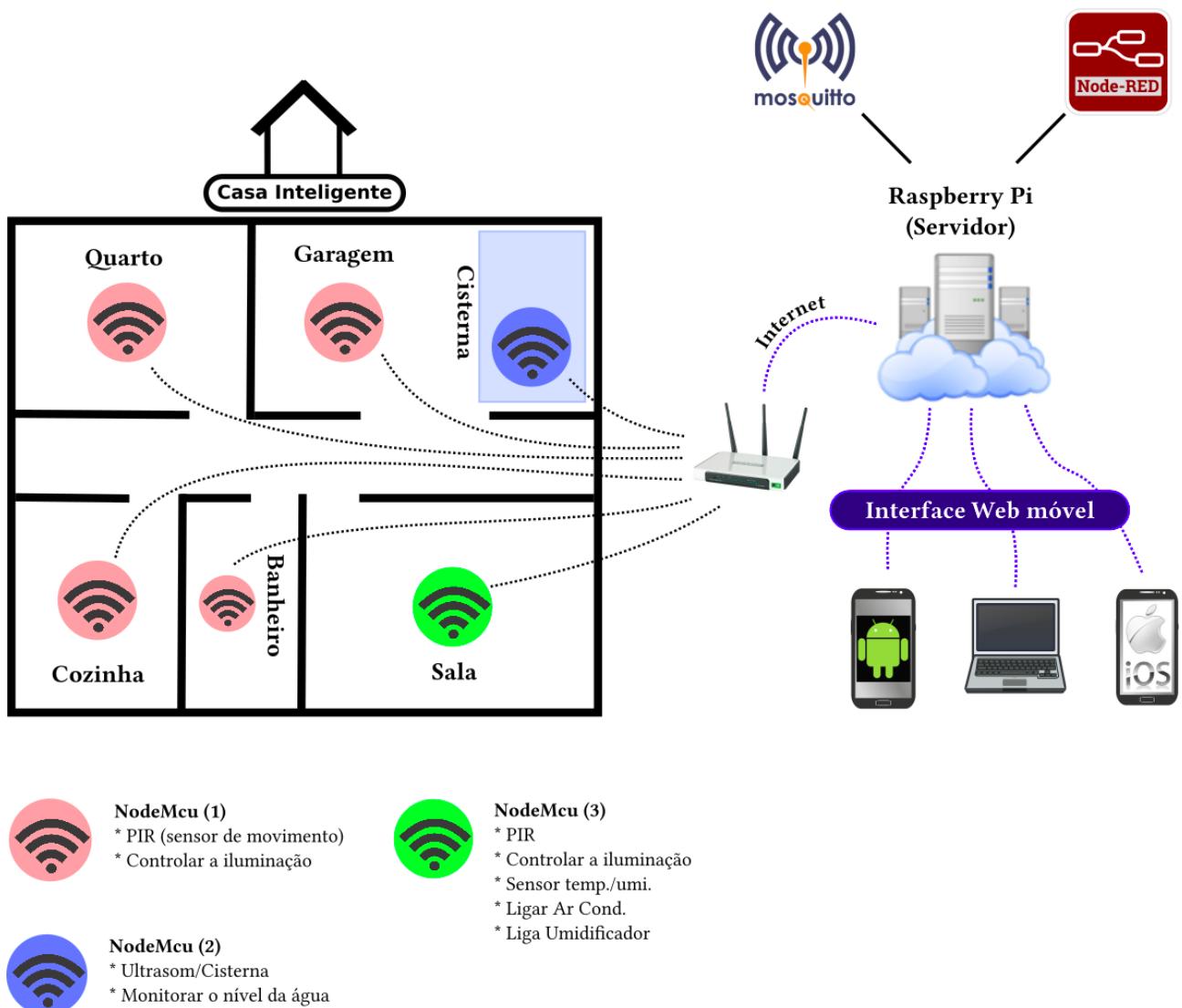


Figura 37: Casa Inteligente - Cenário  
Fonte: (O autor, 2018)

Na Figura 37 é possível ver o cenário da casa inteligente. Através da imagem pode-se ter uma ideia do esquema de posicionamento dos nós NodeMcu. São ao todo seis, um para cada parte da casa, cada qual com seus sensores e funções. Essa rede (WiFi) de nós NodeMcu se conecta ao roteador. Um Raspberry Pi que está conectado à rede local através do mesmo roteador é usado como servidor. Os serviços fornecidos por ele podem ser acessados através das portas 1880 (serviço do Node-Red) e 1883 (Mosquitto/MQTT broker).

## 4.1 PREPARANDO A INFRAESTRUTURA DA REDE

Para que a aplicação seja acessível de fora da rede local, via web, é necessário algumas configurações no modem, no roteador e também usar de um serviço de DNS<sup>49</sup> dinâmico. Neste cenário são usados o Modem SAGEMCOM F@st 5310 (Tim Live), o roteador sem fio TP-LINK modelo TL-WR1043ND e como serviço de DNS dinâmico, o site www.noip.com, que é gratuito. É preciso então, configurar o modem para o modo *bridge*<sup>50</sup>. Sem estar nesse modo, o modem da Tim Live bloqueia o acesso externo a todas as portas, por mais que elas estejam abertas localmente. A forma mais fácil de fazer a configuração para o modo *bridge*, é ir no site da Tim Live, baixar o arquivo do *firmware* já configurado e fazer o upload para o modem. Após essa mudança, deve-se conectar o roteador TP-LINK ao modem. A partir desse ponto, é preciso fazer alguns ajustes importantes no roteador.

Address Reservation					
ID	MAC Address	Reserved IP Address	Status	Modify	
1	5C-C9-D3-62-DB-1E	192.168.0.100	Enabled	<a href="#">Modify</a>	<a href="#">Delete</a>
2	B8-27-EB-52-09-11	192.168.0.50	Enabled	<a href="#">Modify</a>	<a href="#">Delete</a>

Figura 38: Servidor Raspberry Pi - IP estático

Fonte: (O autor, 2018)

Primeiro, deve-se definir um endereço de ip estático (Figura 38) para o servidor Raspberry PI. Segundo, é preciso expor esse endereço de ip à Internet. No TP-LINK, isso é feito ativando a função DMZ (Figura 39) e entrando com o ip do servidor no campo *DMZ Host IP Adress*. O endereço ip do servidor está configurado como 192.168.0.50. Após o término da configuração, o roteador encaminhará todos os pacotes, de todos os serviços, para o DMZ host.

Para configurar o serviço de DNS, é necessário criar uma conta na modalidade gratuita no site www.noip.com, escolher o nome do *host* desejado e fazer uma pequena configuração no roteador. Para isso, é preciso ir no roteador na seção *Dynamic DNS* (Figura

<sup>49</sup> DNS é a sigla em inglês para Domain Name System (Sistema de Nomes e Domínios).

<sup>50</sup> Um equipamento em modo bridge apenas recebe a informação de um lado e entrega do outro, fazendo apenas as conversões de padrão necessárias.

40), entrar com o endereço do serviço de DNS, nome de usuário e senha (registrados no site) e o nome do domínio escolhido.

The screenshot shows a configuration interface for the DMZ (Demilitarized Zone). At the top, there is a green header bar with the text "DMZ". Below it, the main area has a light gray background. It contains two radio buttons for "Current DMZ Status": one labeled "Enable" (which is selected) and another labeled "Disable". Below these buttons is a text input field labeled "DMZ Host IP Address" containing the value "192.168.0.50".

Figura 39: Configurando o DMZ  
Fonte: (O autor, 2018)

Com essa configuração, o roteador periodicamente envia o seu ip (que pode mudar a qualquer momento) para o site de DNS. Assim, mesmo que o ip venha a mudar, o nome do *host* vai sempre apontar para o ip do roteador. O nome do *host* usado para acessar a aplicação no servidor Raspberry Pi é <http://tccmrocha.webhop.me:1880/ui/#/0>. O noip funciona muito bem, apesar de ser gratuito. Ele exige apenas que, a cada 30 dias, o nome do *host* seja confirmado no site deles.

The screenshot shows a configuration interface for Dynamic Domain Name System (DDNS). At the top, there is a green header bar with the text "DDNS". Below it, the main area has a light gray background. It contains four input fields: "Service Provider" (set to "No-IP ( www.no-ip.com )"), "User Name" (set to "tccmrocha"), "Password" (the value is masked by dots), and "Domain Name" (set to "tccmrocha.webhop.me"). Below these fields is a checkbox labeled "Enable DDNS" which is checked. There is also a link "Go to register..." next to the service provider dropdown.

Figura 40: Configurando o DNS dinâmico no roteador  
Fonte: (O autor, 2018)

Após esse processo inicial de configuração, tem-se o servidor Raspberry Pi configurado com o ip fixo, seus serviços expostos à Internet através da opção *DMZ host*, e o nome de *host* apontando para o ip do roteador.

## 4.2 OS ESQUEMAS ELETRÔNICOS DOS NÓS NODEMCU

Há seis nós NodeMcu na casa, e de acordo com suas funções e recursos eles podem ser separados em três tipos (Figura 41). Os nós NodeMcu do tipo 1 (quatro ao todo), estão instalados no quarto, na cozinha, na garagem e no banheiro. Eles podem detectar movimento e controlar a iluminação. O do tipo 2, fica instalado na cisterna e usa um sensor ultrassônico. Sua função é monitorar o nível de água da cisterna. O do tipo 3, fica instalado na sala. Ele pode detectar movimento, medir a temperatura e a umidade ambiente, pode ligar e desligar o ar-condicionado, ligar e desligar o umidificador de ar e pode controlar a iluminação.

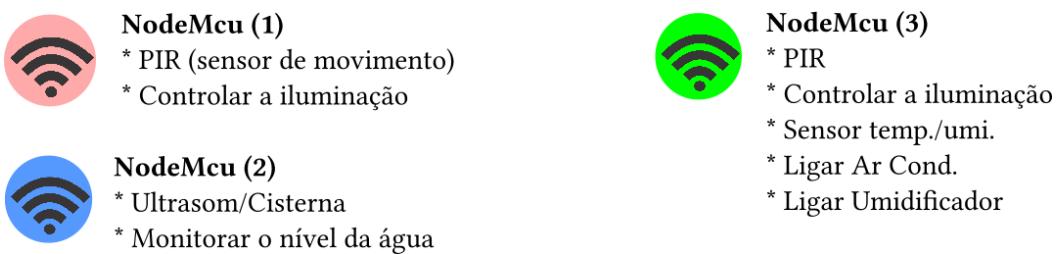


Figura 41: Tipos de nós (NodeMcu) da rede  
Fonte: (O autor, 2018)

A Figura 42 mostra o esquema eletrônico dos nós NodeMcu do tipo 1. Como é mostrado, um sensor PIR e um *led* vermelho são conectados à placa. O *led* vermelho representará o controle da iluminação do ambiente. Há também um *led* azul, que é interno ao NodeMcu, e que ao acender, indicará que um movimento foi detectado.

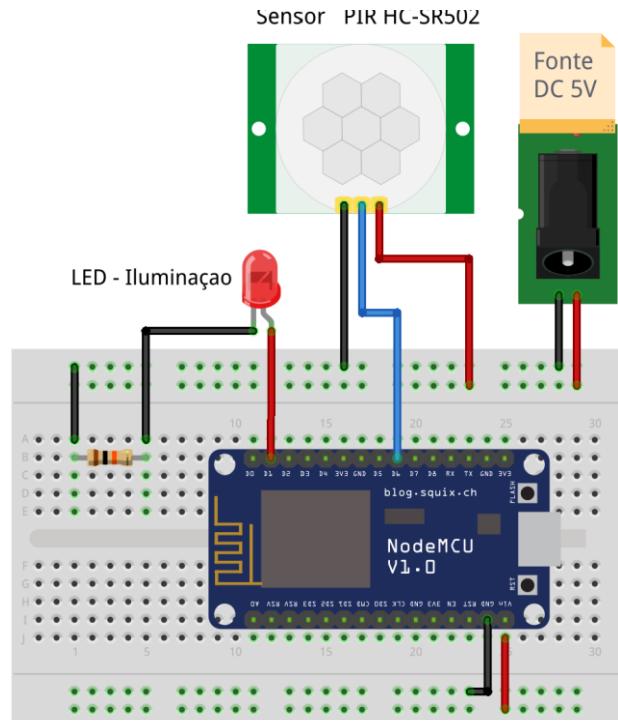


Figura 42: Esquema eletrônico do NodeMcu - Tipo 1  
Fonte: (O autor, 2018)

O esquema eletrônico da Figura 43 é o esquema do NodeMcu do tipo 2. O único componente ligado a ele é o sensor ultrassônico. Este NodeMcu fica instalado dentro da cisterna e sua função é detectar o nível de água.

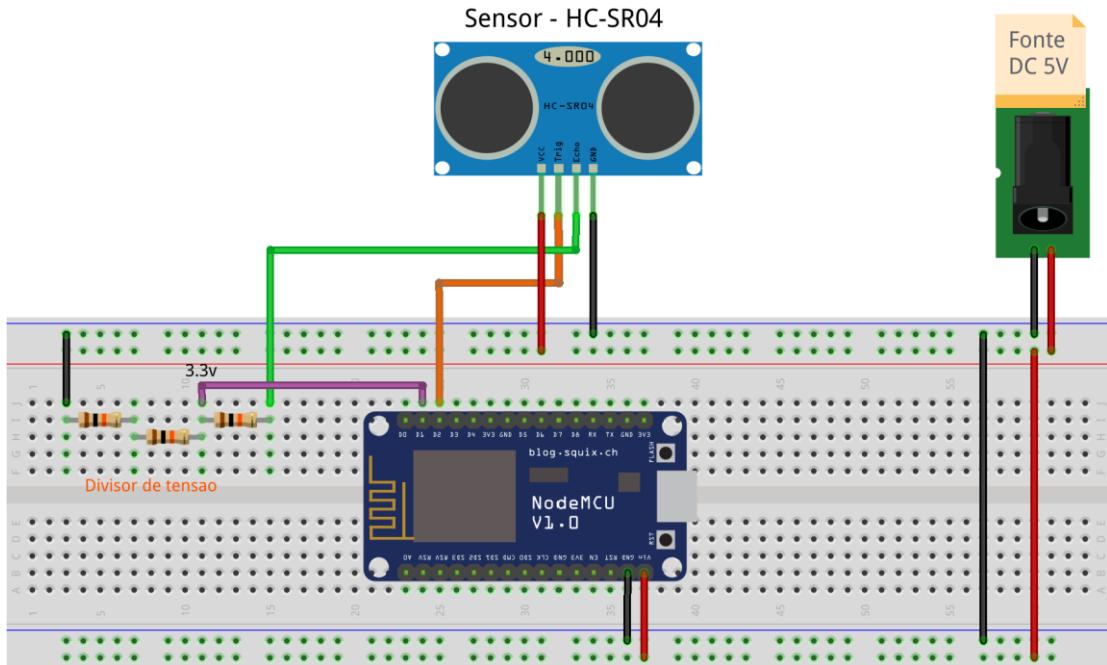


Figura 43: Esquema eletrônico do NodeMcu - Tipo 2  
Fonte: (O autor, 2018)

A Figura 44 mostra o esquema eletrônico do NodeMcu do tipo 3. Este é o mais importante da arquitetura, e se encontra instalado na sala da casa inteligente. Ligado a ele, estão, o sensor PIR, o sensor de temperatura e umidade DHT22, três *leds* (vermelho, azul e amarelo), que representarão, respectivamente, os controles da iluminação, do ar-condicionado e do umidificador, e também o *led* interno da placa, que indicará, que um movimento foi detectado na sala da casa. Todas as arquiteturas anteriores, fizeram uso do sensor DHT11, que não trabalhava com valores decimais. Nesta quinta arquitetura, será usado o DHT22, que é capaz de trabalhar com valores em ponto flutuante.

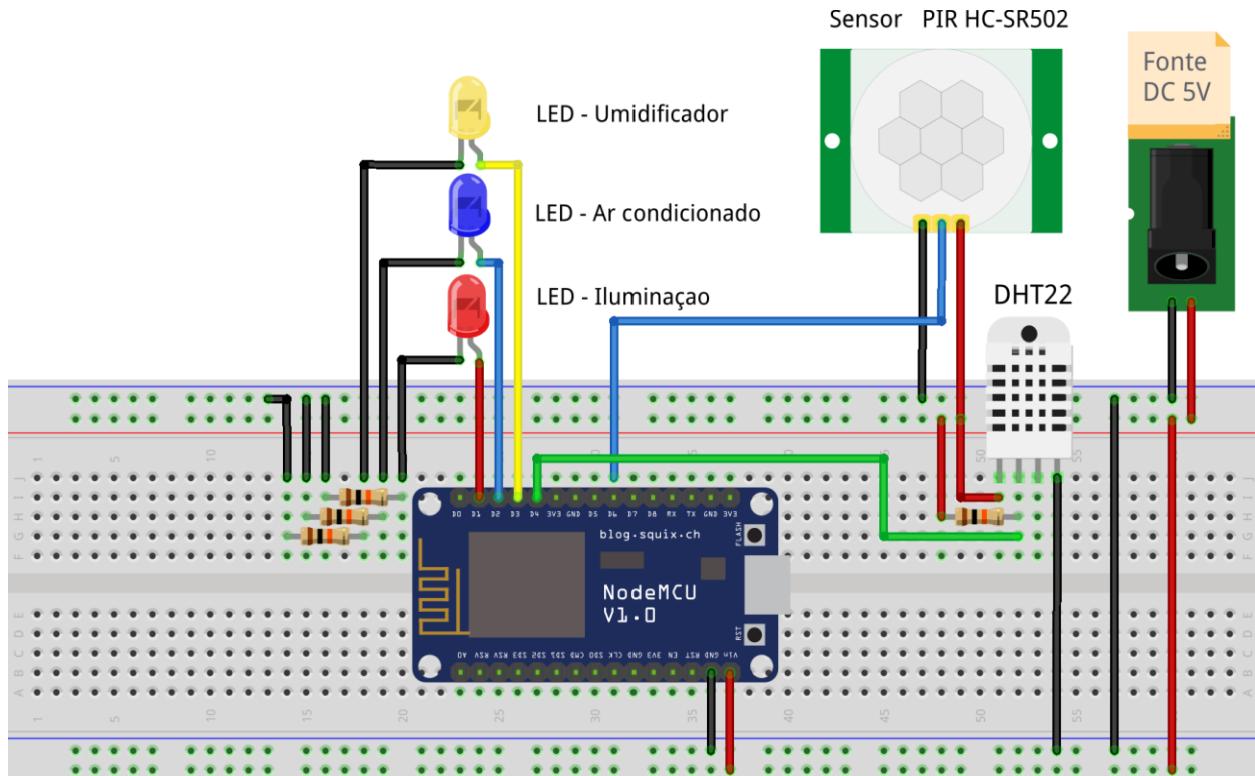


Figura 44: Esquema eletrônico do NodeMcu - Tipo 3  
Fonte: (O autor, 2018)

#### 4.3 ANALISANDO O FUNCIONAMENTO DO CÓDIGO-FONTE

A análise do código, será iniciada por um NodeMcu do tipo 2, usado no quarto, banheiro, garagem e cozinha. O *firmware* do NodeMcu que fica instalado no quarto é composto de sete arquivos:

- init.lua

- conecta\_wifi\_quarto.lua
- credenciais.lua
- tabelas.lua
- firmware\_node\_quarto.lc
- node\_http\_server.lc
- node\_telnet\_server.lc

Esta ideia de modularizar o *firmware* usando vários arquivos já foi vista na Seção 3.2.4. É possível notar que três arquivos têm uma extensão diferente, a extensão .lc. Essa extensão indica que o arquivo já foi compilado para o formato de bytecodes. Há dois arquivos extras nesta arquitetura. Ela traz embarcada um servidor HTTP que pode ser acessado através do ip do NodeMcu (na rede local) e que exibe informações como: o ID do chip, endereço MAC, total de memória, nome do ssid, nome do NodeMcu e etc. Há também, rodando no NodeMcu, um servidor Telnet que pode ser acessado via terminal, tornando possível acompanhar o funcionamento do NodeMcu.

```

1 --Casa Inteligente - Firmware node-quarto
2 --Autor: Marcelo Marques da Rocha
3 --CEDERJ/Universidade Federal Fluminense
4 --TCC - Desenvolvimento Open-Source para a Internet das Coisas
5 --(Arquiteturas para Interfaces Web e Móvel)
6 --Orientador: Professor Dr. Luciano Bertini
7
8 node_nome = "node-quarto" -- nome do dispositivo
9 host = "192.168.0.50" -- endereço do servidor mosquitto rodando no raspberry pi
10 porta_mqtt = 1883 -- porta onde roda o mosquitto
11 top_base = "tccmrocha/quarto" -- tópico base do dispositivo

```

Código 23: Parâmetros de conexão - MQTT broker

Esta arquitetura usa o protocolo MQTT, sendo assim, cada NodeMcu usa uma sequência de variáveis que contêm os campos para usar como parâmetros na função que conecta ao servidor Mosquitto. O Código 23, mostra a declaração desses campos. O primeiro, *node\_nome* é uma string definida arbitrariamente para representar o nome do NodeMcu, essa string nada tem a ver como o id da placa, que vem gravado de fábrica dentro do NodeMcu. As variáveis *host* e *porta\_mqtt* representam o ip do servidor MQTT na rede local e a porta do serviço, respectivamente. Por padrão, o Mosquitto opera na porta 1883. A variável mais importante desse trecho de código é a *top\_base*, que representa o tópico

base em que o node-quarto vai publicar e assinar. Todo nodeMcu da rede tem o seu tópico base.

O próximo trecho (Código 24) configura alguns pinos do NodeMcu. O pino *led\_presença*, está associado ao estado do sensor PIR. Esse *led* é interno ao NodeMcu e funciona com lógica inversa, isto é, quando está em nível alto, o *led* está apagado, quando está em nível baixo, o *led* está aceso. Ao acender, o *led* indica que um movimento foi detectado. Uma outra configuração de pino importante precisa ser citada aqui. Na linha 29, o pino referenciado como *pino\_luz*, é definido como uma saída do tipo PWM<sup>51</sup>(*Pulse-width modulation*). Este pino está conectado a um *led* vermelho, ele representa o controle da iluminação. A configuração para o modo PWM permite que a intensidade do brilho do *led* seja controlada, simbolizando assim, um controle total da luz do ambiente. O PWM do NodeMcu é de 10 bits, logo, permite valores de 0 à 1023. O valor 0 apaga completamente o *led*, o valor 1023 o acende completamente. Os pinos das linhas 27 à 31 funcionam como pinos de saída que enviam um sinal para o meio externo. Na linha 33, pode-se ver um exemplo de um pino configurado como entrada, este pino lê o sinal do sensor de movimento conectado a ele. Nas linhas 36 e 37, são criadas duas tabelas que implementarão duas estruturas de dados do tipo fila (FIFO<sup>52</sup>). Essas duas tabelas (filas) têm um papel importantíssimo no processo de publicação dos tópicos.

```

26 -- configurando os pinos
27 gpio.mode(led_presença, gpio.OUTPUT) -- este pino funciona com logica invertida
28 gpio.write(led_presença, gpio.HIGH)
29
30 pwm.setup(pino_luz, 512, 0) -- config. pino, freq. e duty
31 pwm.start(pino_luz)
32
33 gpio.mode(pino_pir, gpio.INPUT)
34
35 -- tabelas para os topicos e valores
36 topico = {} -- fila de topicos
37 valor = {} -- fila de valores

```

Código 24: Configurando os pinos do node-quarto

O processo de boot dos nós NodeMcu foi programado para seguir a seguinte sequência: O init.lua executa, espera por três segundos e chama o arquivo conecta\_wifi.lua. Que por sua vez, tenta conectar ao WiFi. Após obter seu ip na rede, o conecta\_wifi.lua

<sup>51</sup> Modulação por largura de pulso – É uma técnica para conseguir resultados analógicos usando saídas digitais.

<sup>52</sup> First in is first out – O primeiro a entrar é o primeiro a sair.

chama o arquivo principal, que nesse caso é o `firmware_node_quarto.lc`. Esse último arquivo conecta ao servidor MQTT e tenta assinar os tópicos definidos no seu código. Após o sucesso na assinatura dos tópicos, uma função faz o *led* interno piscar três vezes. O objetivo dessa simples função é dar uma indicação visual ao usuário de que o dispositivo iniciou corretamente. Sendo assim, se o *led* piscou três vezes durante a inicialização, significa que o NodeMcu se conectou ao WiFi, se conectou a servidor MQTT e assinou os tópicos com sucesso.

É possível compilar um código Lua usando o comando `node.compile("arquivo.lua")`. A IDE Esplorer permite fazer o mesmo através da interface gráfica (Figura 45). O código compilado tem um tamanho menor. Na memória flash do dispositivo, além do *firmware* principal e dos arquivos auxiliares, estão os servidores HTTP e Telnet, exigindo o máximo do espaço disponível. Portanto, qualquer redução no tamanho do código é bem-vinda.

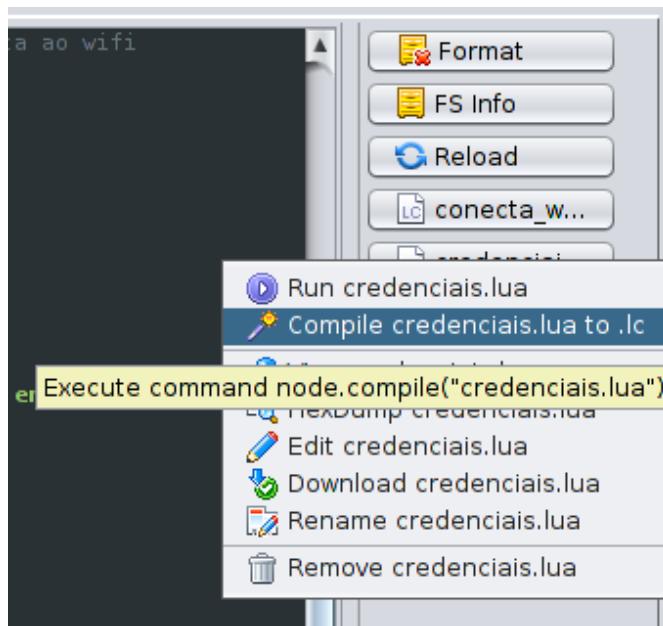


Figura 45: Compilando código Lua com o auxílio da IDE Esplorer

Fonte: (O autor, 2018)

A próxima função, no Código 25, faz a leitura do pino em que está conectado o sensor de movimento. Se um movimento foi detectado, essa informação precisa ser publicada em um tópico. Esse tópico é a concatenação do tópico base (do node-quarto) com a string “/presença”. Então, o nome do tópico em que será publicada a detecção do movimento será “tccmrocha/quarto/presença”. A informação a ser publicada será 1 (para movimento detectado) ou 0 (para ausência de movimento). Um ponto importante nesse código, é que ele analisa o sinal do sensor, define o nome do tópico e o valor a ser publicado,

mas não faz a publicação diretamente. A maneira mais estável de se publicar quando há vários tópicos, é inserir tópico e valor em duas tabelas (filas), como já foi citado. Toda publicação feita neste código segue este mesmo princípio, de adicionar a publicação a uma fila de tópicos e valores. Mais em frente será analisada a função que se encarrega de processar essa fila e publicar tópicos e valores periodicamente. Este procedimento é usado por todas as funções que têm algum tipo de informação a publicar. E isso vale para todos os nós NodeMcu nesta aplicação.

```

79  -- função que faz a leitura do sensor de movimento PIR
80  function lePIR()
81      print("Lendo PIR...")
82      presenca = gpio.read(pino_pir) -- le o pino referente ao sensor PIR
83
84      if presenca == 0 then -- presenca = 0 indica ausencia de movimento
85          gpio.write(led_presenca, 1) -- apaga o led de presenca - logica inversa
86      else
87          gpio.write(led_presenca, 0) -- acende o led de presenca - logica inversa
88      end
89
90      if pir_anterior ~= presenca then -- so' publica se houver mudanca de estado
91          if presenca == 1 then
92              print("Movimento detectado...")
93          end
94          table.insert(topico, top_base .. "/presenca")
95          table.insert(valor, presenca)
96          pir_anterior = presenca -- inverte o estado
97      end

```

Código 25: Lendo o estado do sensor de movimento

O processo de se conectar ao servidor MQTT pode ser visto no Código 26. Ele tem o seu início com a criação do objeto cliente MQTT (linha 101) onde são passados alguns parâmetros para a função construtora do objeto. O primeiro parâmetro é o clientid, que foi passado como a variável nodeName. O segundo, determina o tempo em que a função *callback*, associada ao evento *offline*, vai esperar para iniciar depois que o NoceMcu se encontrar sem conexão. A linha 103, registra a função *callback* para o evento *offline* do NodeMcu, ou seja, se a rede cair por algum motivo, o corpo da função anônima, que é passada como parâmetro, será executado. O código da função imprime uma notificação no monitor serial, desabilita os timers de id 0, 1 e 2 (que são usados por outras funções) e

reprograma o timer 0 para que ele chame a função conecta\_servidor em 2s. Essa função tenta retomar a conexão com o roteador.

```

100  -- cria o cliente mqtt com as opcoes especificadas
101  mqtt_cliente = mqtt.Client(nodeNome, 120, "marcelo", "mrbf8051")
102  -- registra a callback que responde ao estado de offline
103  mqtt_cliente:on("offline",
104    function(client)
105      print ("offline.....")
106      print("reconectando..")
107      tmr.unregister(0)
108      tmr.unregister(1)
109      tmr.unregister(2)
110      tmr.alarm(
111        0,
112        2000,
113        tmr.ALARM_SINGLE,
114        conecta_servidor
115      )
116    end
117  )

```

Código 26: Criando o objeto cliente MQTT

Após a criação do objeto cliente, é o momento de se conectar ao servidor MQTT (Mosquitto) e assinar alguns tópicos. O Código 27 mostra esse processo. É importante observar, através do layout do código, um nível de indentação bastante grande. Ele serve pra mostrar a importância da organização visual do código, e que sem essa indentação organizada, o entendimento do programa se torna quase impossível. Muitas funções têm como parâmetros funções anônimas, que também usam funções anônimas. E isso pode bagunçar as coisas.

```

158 -- conecta ao mqtt broker
159 ~ function conecta_servidor()
160   -- registra a callback de conexao com o servidor mqtt
161   mqtt_cliente:connect(host, porta_mqtt,
162     function(client) -- se a conexao foi um sucesso entao assina os topics
163       print("Conectado ao mqtt broker...")
164       print("Topics assinados")
165       print("-----")
166       -- cada topico e' assinado a partir do sucesso do anterior
167       client:subscribe(top_base .. "/luz", 0,
168         function(client)
169           print("Topico: " .. top_base .. "/luz OK")
170           client:subscribe(top_base .. "/luz/intensidade", 0,
171             function(client)
172               print("Topico: " .. top_base .. "/luz/intensidade OK")
173             client:subscribe(top_base .. "/reset", 0,
174               function(client)
175                 print("Topico: " .. top_base .. "/reset OK")
176                 client:subscribe("tccmrocha/all/pergunta", 0,
177                   function(client)
178                     print("Topico: tccmrocha/all/pergunta OK")
179
180                     -- pisca o led interno 3x indicando que inicializou corretamente
181                     pisca_led_3x()
182
183                     -- reseta a interface grafica
184                     table.insert(topico, top_base .. "/reset")
185                     table.insert(valor, "interface_grafica")

```

Código 27: Conectando ao servidor MQTT parte 1

A função `connect` (linha 161) faz uso de quatro parâmetros, o host, a porta MQTT, uma `callback` associada ao sucesso na conexão e outra associada ao fracasso. Há um detalhe importante, só é possível assinar um tópico quando existe uma conexão com o MQTT *broker*, por isso, é no corpo da `callback` associada ao sucesso da conexão que os tópicos são assinados um a um, sendo que, cada tópico só é assinado após o sucesso na assinatura do seu antecessor. Esse é o outro motivo do nível de indentação do código, há um encadeamento de funções. A linha 181 chama a função que pisca o *led* três vezes e as linhas 184 e 185 mandam para as tabelas tópico e valor, um comando que faz com que a interface gráfica da aplicação seja reiniciada. Esse procedimento faz parte da inicialização da interface gráfica da aplicação web. O Código 28 continua dentro da função que conecta ao servidor MQTT. A função registra dois timers (linhas 188 e 197). O timer de id 1 fica encarregado de ler o sensor de movimento periodicamente em intervalos de 300ms. O timer de id 2 é o responsável por fazer a publicações dos tópicos, que serão feitas em

intervalos periódicos de 200ms. Da linha 204 à 212, tem-se o fechamento dos blocos de funções declaradas dentro da função associada ao sucesso da conexão.

A parte 3 do código de conexão ao servidor (Código 29) define o corpo da função *callback* chamada no caso de insucesso na conexão. Ela exibe algumas mensagens e mostra qual foi o motivo do insucesso. Exibe o tipo do erro, fazendo uso dos índices da tabela *err\_conn* declarada no arquivo *tabelas.lua*, desabilita os timers usados no programa e por fim, registra um timer que inicia uma nova tentativa de conexão.

```

188          tmr.alarm( -- programa as leituras do PIR para cada 300ms
189              1,
190              300,
191              tmr.ALARM_AUTO,
192              lePIR
193          )
194      ---]]
195
196      ---[[[
197      tmr.alarm( -- programa as publicacoes para cada 200ms
198          2,
199          200,
200          tmr.ALARM_AUTO,
201          publicaTopico
202      )
203      ---]]
204      end
205  )
206  end
207  )
208  end
209  )
210  end
211  )
212  end,
```

Código 28: Conectando ao servidor MQTT parte 2

O Código 30 trata do recebimento de mensagens do servidor MQTT. São mensagens publicadas nos tópicos assinados anteriormente, durante o processo de conexão ao servidor. A chegada de uma mensagem é sinalizada através do evento *message*, e como quase todas as funções declaradas até agora, ela tem uma função *callback* que será chamada quando o evento ocorrer. Essa função possui no seu cabeçalho, parâmetros que retornam informação. Os parâmetros *topic* e *data*, trazem consigo, o tópico e a informação contidas na mensagem. A partir daí, o que se faz é verificar tópico e informação a fim de

reagir de acordo com o programado. É possível ver os tópicos: “top\_base/luz”, “top\_base/luz/intensidade”. Os tópicos com a palavra “luz” estão relacionados ao controle dos *leds*.

O tópico “tccmrocha/all/pergunta” é uma mensagem enviada pela aplicação Node-RED com o objetivo de saber se os nós NodeMcu estão online. Caso esteja, o nó responde essa mensagem publicando no tópico “tccmrocha/all/resposta/quarto” a string contendo o seu número ip concatenado à string “Online”. A resposta fica dessa forma: 192.168.0.3 – Online.

```

213     function(client, reason)
214         print("Falha ao conectar.....")
215         print("Razao da falha: " .. err_conn[reason]) -- indice na tabela de erro
216         print("Reconectando..")
217         -- desliga todos os timers
218         tmr.unregister(0)
219         tmr.unregister(1)
220         tmr.unregister(2)
221         tmr.alarm(
222             0,
223             2000,
224             tmr.ALARM_SINGLE,
225             conecta_servidor
226         )
227     end
228 end
229 end

```

Código 29: Conectando ao servidor MQTT parte 3

A última função do programa pode ser vista no Código 32. Essa função é responsável por pegar os itens da fila de tópicos e valores e publicá-los. A função começa verificando se o primeiro item, tanto dos tópicos quanto dos valores, tem o valor nil.

```

121 -- registra a callback de recebimento de mensagem
122 mqtt_cliente:on(
123   "message",
124   function(client, topic, data)
125     print(topic .. " = " .. data)
126     if topic == top_base .. "/luz" and data == "liga" then
127       pwm.setduty(pino_luz, 1023) -- liga a luz
128     end
129     if topic == top_base .. "/luz" and data == "desliga" then
130       pwm.setduty(pino_luz, 0) -- desliga a luz
131     end
132     if topic == top_base .. "/luz/intensidade" then
133       pwm.setduty(pino_luz, data) -- seta o pwm da intensidade da luz

```

Código 30: Recebendo uma mensagem MQTT parte 1

Um erro de leitura em algum sensor pode ter retornado um valor nil e ele ter sido enviado para a fila de publicação. Há duas situações que certamente provocarão um *panic* no NodeMcu fazendo com que ele reinicie, são elas: tentar publicar um valor nil e tentar concatenar uma string a um valor do tipo nil. Essa situação ocorre geralmente quando se tenta concatenar uma variável que contém uma string a uma variável a qual não foi atribuída valor algum. Em Lua, toda variável sem atribuição de valor, possuirá inicialmente, o valor nil. Lua é dinamicamente tipada, ou seja, as variáveis não possuem tipos, os valores possuem tipo.

```

149   if topic == "tccmrocha/all/pergunta" and data == "ping" then
150     -- publica o ip do dispositivo indicando que esta online
151     table.insert(topico, "tccmrocha/all/resposta/quarto")
152     table.insert(valor, wifi.sta.getip().." - Online")
153     print(node_nome .. "-OnLine")
154   end
155 end
156 )

```

Código 31: Recebendo uma mensagem MQTT parte 2

Após a verificação da ausência de valor nil na tabela tópico e na tabela valor, o programa segue chamando o método (publish) do objeto mqtt\_cliente que publica a informação no tópico. Os parâmetros da função publish são: tópico, valor (*payload*), QoS, *retain*

(valor\_retido), *callback*. Os significados dos parâmetros tópico e valor, dispensam comentários. A explicação detalhada sobre os parâmetros QoS e *retain* pode ser vista nas Seções 2.2.3 e 2.2.4, respectivamente. O último parâmetro é a função *callback* que é chamada caso a publicação seja bem-sucedida. Após o sucesso no envio da informação, o programa imprime no monitor serial a mensagem dizendo que está publicando, em seguida, remove da fila de tópicos e da fila de valores, as informações já enviadas. A função publish, após ser chamada, retorna um valor booleano *true*, em caso de sucesso na publicação, ou *false*, em caso de fracasso. Esse valor é atribuído à variável pub\_status, que é testada na linha 71. Em caso de pub\_status possuir o valor *false*, será impresso no monitor serial, a mensagem indicando o erro na publicação.

O trecho final do programa é mostrado no Código 33. Ele simplesmente inicia o servidor HTTP, inicia o servidor Telnet e agenda o início da chamada à função conecta\_servidor, que dá início a todo ciclo de código explicado até aqui.

```

57 -- funcao que publica em um topico mqtt
58 -- ela le a tabela de topics e valores e os publica um por vez
59 -- as tabelas funcionam como uma fila (FIFO)
60 function publicaTopico ()
61   if (topico[1] ~= nil and valor[1] ~= nil) then -- primeiro topico da fila
62     -- pub_status recebe true ou false indicando o sucesso da publicacao do topico
63     pub_status = mqtt_cliente:publish(topico[1], valor[1], 0, 0,
64                                         function()
65               print('Publicando... ')
66               -- remove topico e valor
67               table.remove(topico, 1)
68               table.remove(valor, 1)
69             end
70           )
71   if pub_status == false then
72     print("Erro ao publicar.....")
73   end
74 else
75   -- nao havia topics na fila
76 end
77 end

```

Código 32: Publicando mensagens MQTT

Os nós NodeMcu do tipo 2 e tipo 3, node-cistena e node-sala, respectivamente, fazem uso da mesma filosofia usada neste tipo 1. Os timers são usados para, periodicamente, executar funções de leitura de sensores e também ler a tabela de tópicos e publicá-los. Esta é uma idéia padrão seguida em todos os firmwares desta aplicação.

```

232
233 dofile("node_http_server_lc") -- roda o servidor http
234 dofile("node_telnet_server_lc") -- roda o servidor telnet
235
236 -- chama a função que conecta ao mqtt broker
237 tmr.alarm(
238     0,
239     1000,
240     tmr.ALARM_SINGLE,
241     conecta_servidor
242 )
243

```

Código 33: Iniciando os serviços HTTP e Telnet

Para acessar o servidor HTTP (Figura 46) que roda no node-quarto ou em qualquer outro nó NodeMcu da rede (exceto o node-cisterna) é só entrar o ip do NodeMcu no navegador. O servidor fornece uma página com informações sobre o nó NodeMcu. É possível ter acesso ao código do servidor HTTP no Apêndice E.

Informações do dispositivo - node-sala	
Hardware	Wifi Settings
Major version	2
Minor version	1
Dev. version	0
Chip ID	14194525
Flash ID	1458400
Flash size	4096 Kbytes
Flash mode	2
Flash speed	40 MHz
Protótipo desenvolvido por <b>Marcelo Marques Da Rocha</b> Aluno do curso: <b>Tecnologia em Sistemas de Computação</b> Instituição: <b>CEDERJ / UFF</b> Ano: <b>2018</b>	

Figura 46: Acessando o servidor HTTP do nó NodeMcu  
Fonte: (O autor, 2018)

O servidor Telnet espelha as informações enviadas para o monitor serial. Pode-se ter acesso ao código do servidor Telnet no Apêndice F. Para acessar o servidor (Figura 47) é preciso abrir um terminal e digitar:

telnet <no. do ip do NodeMcu> 2323

```

marcelo@ubu-note: ~
Lendo PIR...
Lendo PIR...
Lendo PIR...
Lendo PIR...
Movimento detectado...
Publicando...
Lendo PIR...
Lendo DHT22...
Temperatura = 24.6
Umidade = 46.7
Lendo PIR...
Publicando...
tccmrocha/sala/arcondicionado = desliga
Publicando...
  
```

Flash ID|1458488  
Flash size|4096 Kbytes  
Flash mode|2  
Flash speed|40 MHz  
  
Protótipo desenvolvido por Aluno do curso: Introdução à Internet das Coisas - IPI  
Hora 59 - Acessando o NodeMCU via Telnet

Figura 47: Saída do servidor Telnet  
Fonte: (O autor, 2018)

Há uma observação importante sobre o NodeMcu que monitora a cisterna. O cálculo da distância usando o sensor ultrassônico implica em medir intervalos de tempos muito pequenos, sendo assim, neste NodeMcu não estão disponíveis os servidores HTTP e Telnet pois eles comprometiam o resultado dos valores aferidos.

#### 4.4 CONSTRUINDO A APLICAÇÃO MÓVEL COM O NODE-RED

Esta seção trata da aplicação construída no Node-RED. Uma ferramenta poderosa que permite a criação de interfaces gráficas, a comunicação utilizando diversos protocolos (HTTP, MQTT), permite a plotagem de gráficos, criação de botões e sliders, comunicação serial com Arduino, comunicação direta com o Raspberry Pi (quando instalado nele), permite interagir com contas de Twitter, enviar e-mails, tocar áudio, permite a comunicação com o sistema de arquivos local e permite obter informações sobre clima, usando o node openweathermap. Há, na verdade, uma infinidade de nodes para aplicações diversas.

Com o Node-RED, é muito fácil assinar e publicar em tópicos no MQTT *broker*. Há um componente para cada ação. Uma outra grande vantagem em desenvolver com o Node-RED, é que a interface gráfica gerada por ele pode ser acessada por um navegador de Internet, seja na rede local ou via web, e isso tudo, em qualquer dispositivo, seja ele um Pc, tablet ou celular.

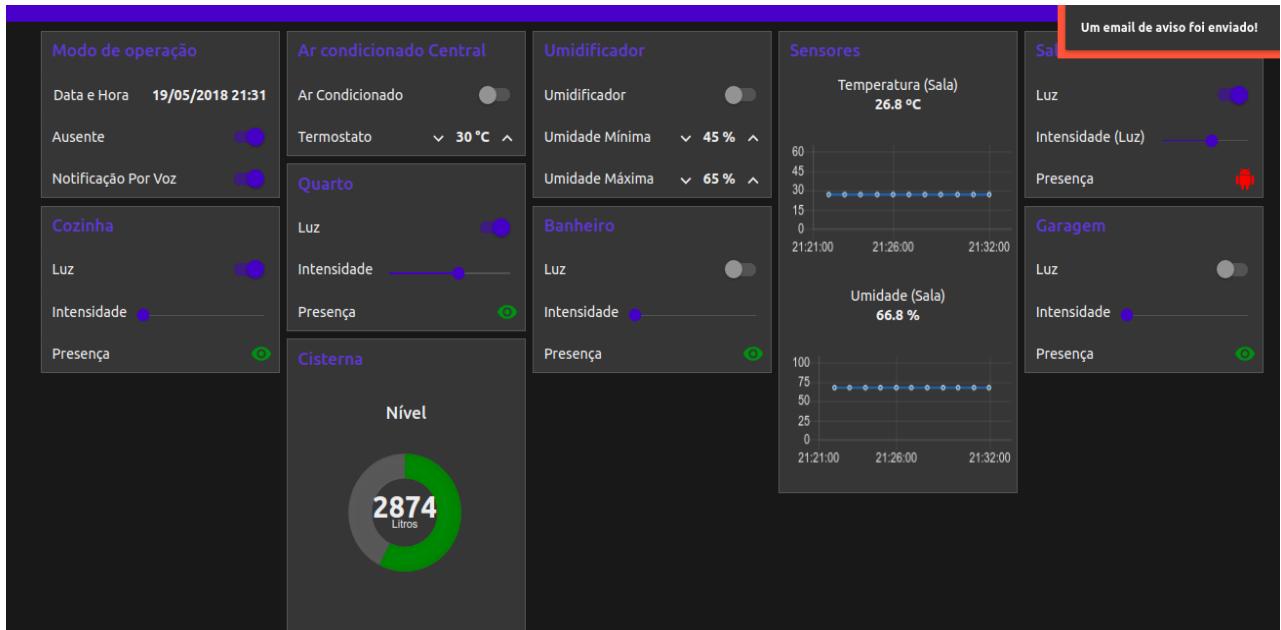


Figura 48: Interface gráfica da aplicação (Node-RED) - Visão dos controles  
Fonte: (O autor, 2018)

A aplicação pode ser acessada localmente através do endereço <http://192.169.0.50:1880/ui/#/0> ou via Internet com a url <http://tccmrocha.webhop.me:1880/ui/#/0>.

Antes de falar sobre a utilização dos componentes do Node-RED, é interessante mostrar uma imagem da aplicação e falar sobre suas características e recursos. A aplicação é dividida em duas telas. A Figura 48 mostra a primeira tela da aplicação.

Estas telas podem ser acessadas através do menu no canto superior esquerdo da aplicação (Figura 49).

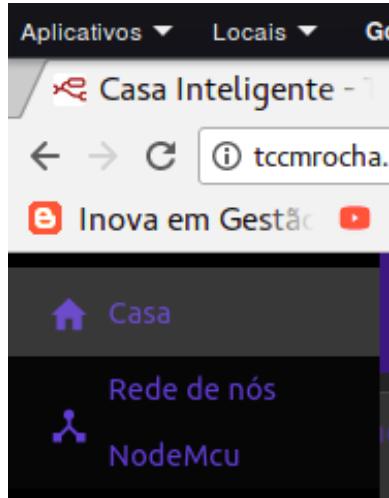


Figura 49: Menu da aplicação no NodeRED

Fonte: (O autor, 2018)

Na imagem que descreve o cenário da aplicação, que é uma casa inteligente (veja a Figura 37), pode-se ver que seis nós NodeMcu, que são dispositivos WiFi, estão conectados a sensores (de movimento, temperatura, umidade, ultrassom) e a *leds*. Cada *led*, representa o envio de um sinal para o meio externo. Dentro da rede, os nós NodeMcu são dispositivos inteligentes. Aqui o termo inteligente acompanha a definição dada por BOLZANI:

O termo “inteligente” é utilizado neste trabalho como diferenciador entre o estado mais simples do sistema em questão e o seu estado de valores agregados ou evoluído. No momento que o dispositivo é construído com a possibilidade de gerir ele próprio ou outros equipamentos acredita-se que o termo “inteligente” possa ser empregado sem restrições. Vale ressaltar que o termo tem sido muito utilizado comercialmente e com fins publicitários para enaltecer novas funcionalidades de equipamentos, dispositivos, sistemas ou serviços. Porém, na verdade, muitas vezes não desempenham efetivamente uma capacidade de aprender, compreender, interpretar ou desenvolver alguma forma de intelecto ou perspicácia como descrito nos dicionários. (2004, p. IX)

A aplicação se comunica com a rede de nós NodeMcu, recebendo os dados vindos dos sensores processando esses dados e enviando comandos de controle para os dispositivos. Há integração, análise e processamento de eventos, que como foi dito na Seção 2.1, são os três pilares da IoT.

Primeiro serão mostrados os elementos pertencentes à tela 1 (menu – casa). Para começar a falar sobre os comandos da interface da tela 1, foi escolhido como objeto de análise, a interação da aplicação como o node-sala. O node-sala está conectado a um sensor de temperatura e umidade, a um sensor de movimento e é capaz de controlar *leds*. Esse controle engloba os sinais de ligar e desligar, além do o controle de intensidade do brilho do *led*. Como já tem sido falado anteriormente neste capítulo, esses *leds* simboli-

zam um sinal digital de controle, a fim de simular o controle de um dispositivo externo, como um ar-condicionado, um umidificador de ar e lâmpadas.

O node-sala foi escolhido para a análise por ser o dispositivo que mais ocupa a interface gráfica, e também pela variedade de componentes que essa interação requer. A Figura 50, mostra a seção da interface que controla o *led* que simboliza a iluminação da sala, o *led* vermelho (veja a figura Figura 44).



Figura 50: Seção de controle da iluminação da sala

Fonte: (O autor, 2018)

Na Figura 50, O item (1) é o botão que liga e desliga o *led* vermelho, o item (2) é o controle de brilho do *led* e o item (3) é o ícone que indica a presença de movimento na sala, quando o ícone aparece como um olho, significa que o sensor está atento, tomando conta e não há movimento, quando o ícone se transforma em um robozinho (Figura 51), é a indicação de movimento detectado no local.



Figura 51: Ícone do Android

Fonte: (O autor, 2018)



Figura 52: Seção de controle do ar-condicionado central

Fonte: (O autor, 2018)

O node-sala, usa três seções de controle na interface. A primeira seção já foi vista, a segunda seção, é a que controla a parte do ar-condicionado central (Figura 52). O item (1) representa a capacidade de ligar manualmente o ar-condicionado. O *led* que simula este controle é o azul. O item (2) é o controle do termostato, que regula a temperatura em que o ar-condicionado deve ligar ou desligar. Esse controle usa uma variável, a nível da aplicação Node-RED, que guarda o valor selecionado permitindo que ele possa ser comparado com a informação vinda do NodeMcu e assim, desligar ou ligar o ar-condicionado automaticamente.

A terceira seção de controle do node-sala, é a parte que controla a umidade do ar. O item (1) da Figura 53, é o botão de controle manual do umidificador. O item (2) é a configuração do mínimo e do máximo da umidade relativa do ar, se ela estiver abaixo do mínimo, o umidificador é acionado automaticamente, se estiver acima do máximo, o aparelho é desligado. O *led* amarelo, simboliza o umidificador de ar no circuito.



Figura 53: Seção de controle do Umidificador de ar

Fonte: (O autor, 2018)

A última seção ocupada pelo node-sala, é a seção de plotagem de gráficos tipo linha e pode ser vista na Figura 54. Esta seção é responsável por traçar dois gráficos, um da temperatura e outro da umidade, todos em função do tempo. Os dados são enviados pelo NodeMcu em intervalos de 1 minuto. A segunda seção gráfica, ainda da mesma figura, é a que desenha um gauge exibindo o nível de água da cisterna, em litros. Esta seção está associada ao node-cisterna. A capacidade máxima da cisterna é de 5000 litros.

Os nós NodeMcu restantes, node-garagem, node-quarto, node-cozinha e node-banheiro usam uma seção com os mesmos controles da Figura 50.

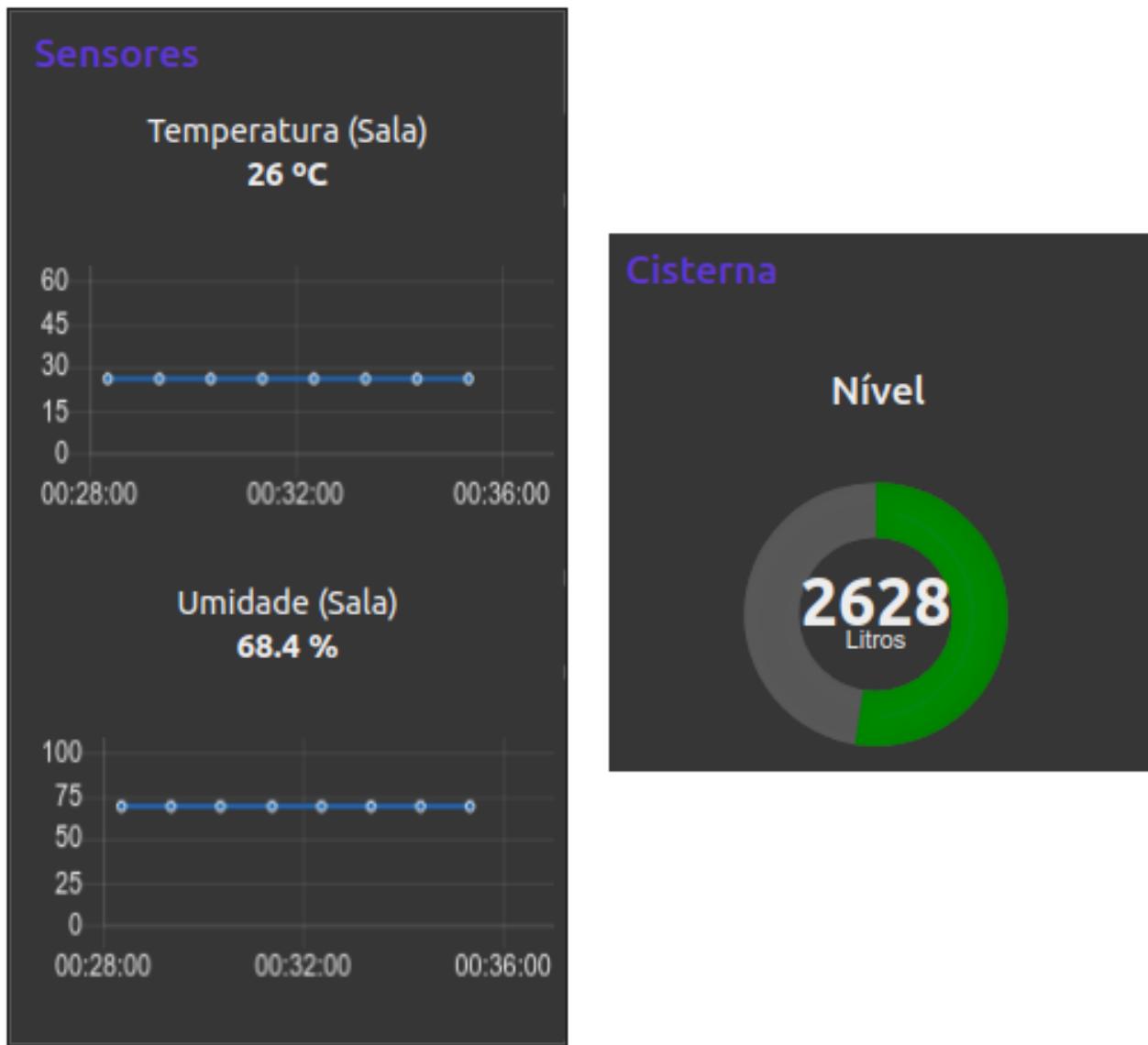


Figura 54: Seção de gráficos - Temperatura, umidade e nível da cisterna  
Fonte: (O autor, 2018)

A última seção da tela 1 (Figura 48), é uma seção que define o modo de operação da aplicação e pode ser vista na Figura 55. Nela é mostrada a data e a hora do sistema e mais dois botões que acionam dois modos de operação, que podem funcionar simultaneamente. Um dos modos é o modo ausente, que quando ativado, envia um e-mail para o usuário caso seja detectado um movimento em um determinado cômodo da casa. O outro modo de operação é o de notificação por voz. Neste modo, a aplicação passa a usar uma voz sintetizada para emitir algumas notificações, por exemplo, quando um movimento é detectado e também quando o nível da cisterna está em nível crítico. Uma observação importante, quando a notificação por voz é ativada, a string “sim” é atribuída a uma variável

global chamada “notivoz”. Essa variável será comentada logo adiante. Com relação ao nível da cisterna, a aplicação sempre envia um twitter quando a cisterna se encontra em nível crítico, abaixo de 2000 litros.

A Figura 56, mostra os elementos da tela 2 (menu – Rede de nós NodeMcu). Esta parte da interface mostra informações sobre a rede de nós NodeMcu. Na seção *Status dos dispositivos*, é possível ver os endereços ip dos nós NodeMcu e também ver quais estão online. Essa listagem do status é atualizada automaticamente a cada 1 minuto, mas pode ser atualizada manualmente pressionando o botão “atualizar”. O botão limpar, limpa a lista.

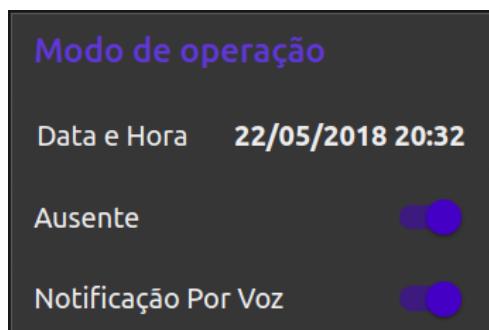


Figura 55: Interface gráfica - Modo de operação  
Fonte: (O autor, 2018)

Na seção *Reiniciar dispositivos*, existem sete botões que permitem que os nós sejam reiniciados, isto é claro, se eles estiverem online. O primeiro botão reinicia todos os nós e os outros seis reiniciam cada nó individualmente.



Figura 56: Interface gráfica da aplicação Node-RED - Rede de nós NodeMcu  
Fonte: (O autor, 2018)

Daqui em diante, será apresentado como estabelecer a comunicação com o MQTT *broker*. Será visto como assinar e como publicar nos tópicos através dos componentes do Node-RED.

O que acontece nos “bastidores” do sistema, e entenda-se por sistema (os *firmwares* nos nós NodeMcu, o código Node-RED e a interface gráfica) é a troca de mensagens no protocolo MQTT. Todas as informações vindas dos nós Node-Mcu, todos os comandos enviados pela aplicação web para os nós NodeMcu, são mensagens MQTT. O Node-RED permite que essa comunicação bidirecional seja implementada de uma maneira muito prática e rápida.

A fim de demonstrar uma parte importante do desenvolvimento da aplicação com o Node-RED, dois exemplos serão detalhados.

O primeiro exemplo é o do recebimento da informação do nível da cisterna, vinda do node-cisterna. A Figura 57, mostra o fluxo (*flow*) da conexão dos componentes do Node-RED para a se comunicar com o NodeMcu da cisterna. Para criar uma aplicação entrando com os componentes no Node-RED, é preciso ter acesso à tela de edição de fluxos, ela pode ser acessada através do navegador usando o endereço do servidor Node-RED junto com a porta do serviço, desta maneira <http://192.168.0.50:1880>.

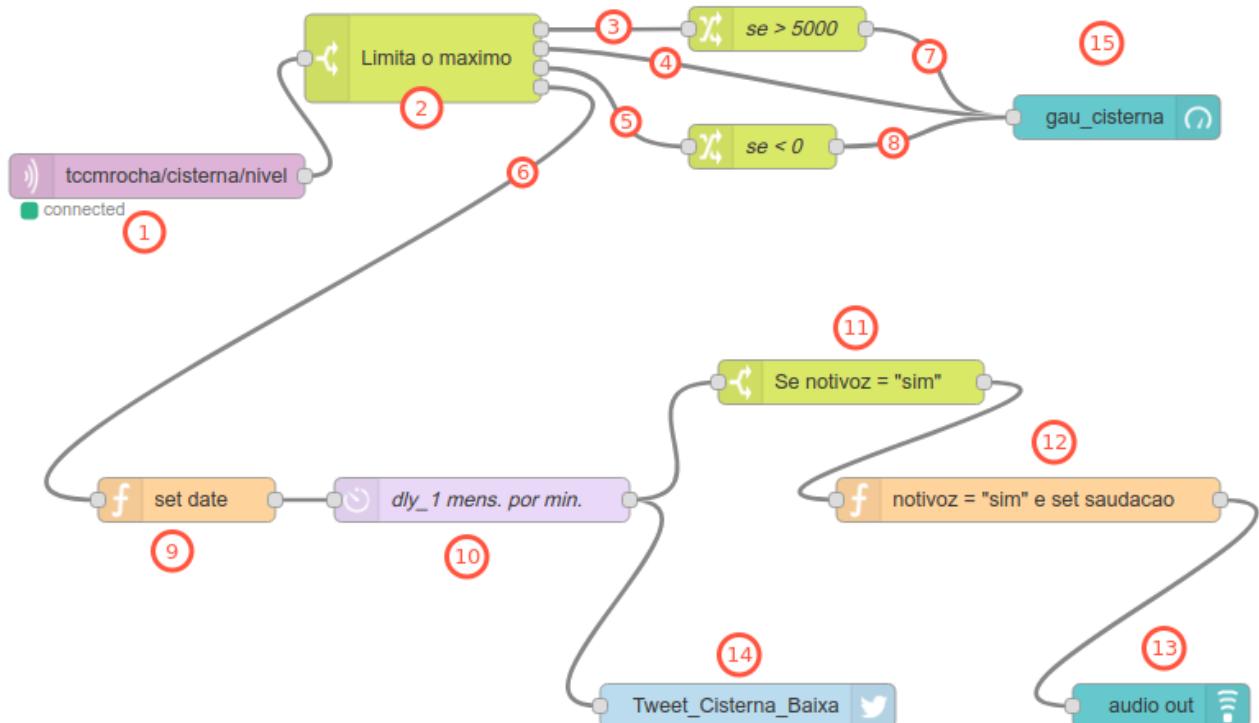


Figura 57: Esquema do Node-RED - Conexão com a cisterna  
Fonte: (O autor, 2018)

O componente (1) é um input-node MQTT. Seus parâmetros de configuração são o endereço ip e a porta do MQTT *broker* mais o tópico a ser assinado. Quando o NodeMcu da cisterna publica o nível da água no tópico “tccmrocha/cisterna/nível” essa informação é recebida pelo input-node MQTT. Ao receber a informação, ela é passada para o componente (2), o function-node switch. Ele é capaz de rotear várias saídas baseadas em critérios que são definidos dentro da sua configuração. A conexão (3) é usada para valores maiores que 5000. A conexão (4) é usada para valores no intervalo [0, 5000]. A conexão (5) para valores menores do que 0 e por último a conexão (6) que é usada quando o valor se encontra no intervalo [0, 2000]. Essas faixas de valores são definidas na tela de configuração do componente (Figura 58). As saídas (3) e (5) carregam valores maiores que 5000 e menores que 0, respectivamente. Elas são encaminhadas para outros dois function-nodes chamados change que limitam o valor máximo para 5000 e o mínimo para 0. As saídas desses changes, conexões (7) e (8), alimentam o componente gráfico (15) que desenha um gauge com o valor recebido em sua entrada.

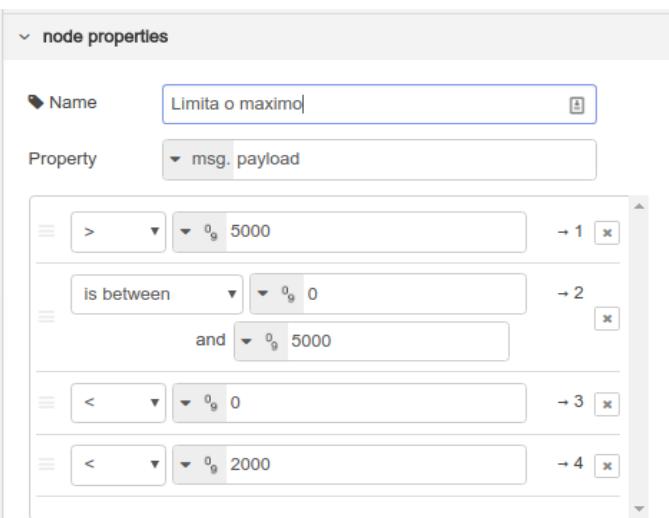


Figura 58: Configuração do function-node Switch  
Fonte: (O autor, 2018)

Seguindo a rota da conexão (6) que carrega um valor no intervalo [0, 2000]. A informação é direcionada para o function-node function (9), que recebe o valor do nível, gera uma string com a data e a hora do sistema e cria uma string única com uma mensagem de aviso. Essa mensagem passa para o function-node delay (10) que tem a função de limitar a taxa na qual a informação passará adiante, está taxa está configurada para 1 minuto. A partir daqui, a informação é duplicada e direcionada para o componente (14) que vai tweetar a mensagem na conta do usuário (Figura 59) e para o componente (11) que

verifica se a variável global “notivoz” é igual a “sim”, caso a condição seja verdadeira, a mensagem segue para o function-node function(12) que troca a mensagem recebida por um texto que será falado pela aplicação, esse texto é “Cisterna em nível crítico, fique atento!”. Essa string é passada para o output-node audio-out que falará o texto contido na string.

## Tweets Tweets e respostas



Figura 59: Tweet de aviso de cisterna em nível crítico  
Fonte: (O autor, 2018)

O segundo exemplo será a publicação de mensagens em um tópico MQTT. Para isso será usado o fluxo que gera a seção *Reinicia dispositivos* (Figura 56) da interface gráfica. A Figura 60, mostra o fluxo responsável pela criação dos botões de reset individual dos nós NodeMcu na interface e também o componente responsável pela publicação das mensagens. Os componentes (1) são dashboard-nodes do tipo botão, eles são elementos da interface com o usuário e são exibidos na tela da aplicação.

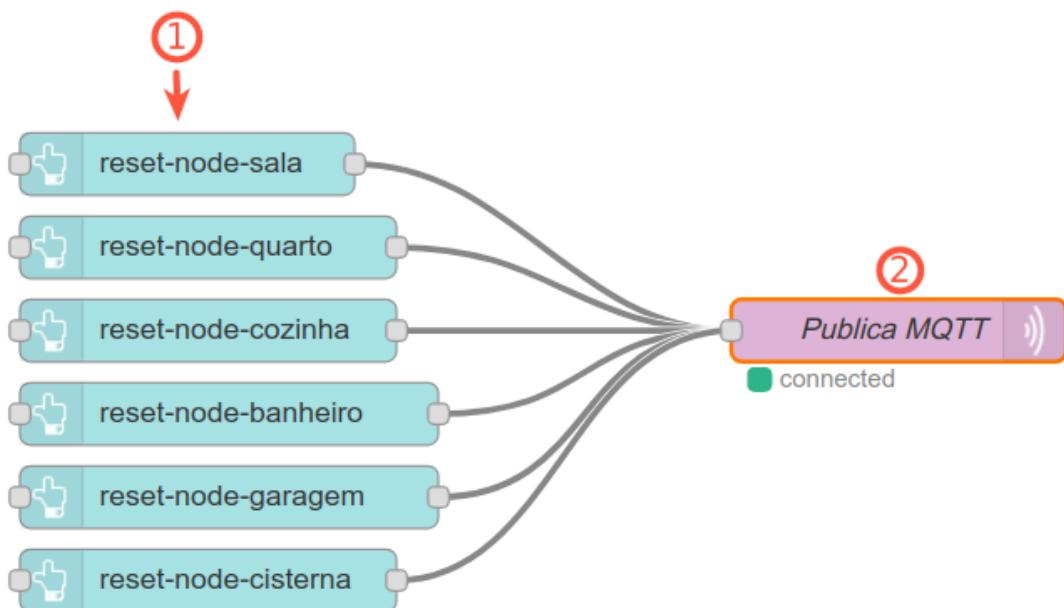


Figura 60: Esquema do Node-RED - Publicando em tópicos  
Fonte: (O autor, 2018)

Esses botões respondem ao evento de clique do mouse. Após clicados enviam para o componente (2) (output-node MQTT) a informação como o tópico e valor a serem publicados. Cada botão, envia o seu tópico e valor específico e o componente MQTT se encarrega da publicação. A configuração do componente (2) usa somente um parâmetro, o endereço e a porta do MQTT *broker*.

A Figura 61 mostra o esquema (*flow*) que interage com o node-sala, o nó NodeMcu que tem mais funções e que publica e assina um maior número de tópicos. É possível ver que há uma complexidade muito maior neste *flow* do que no do node-cistena (Figura 57). Os princípios de comunicação com o MQTT *broker* (publicando/assinando) são os mesmos já explicados.

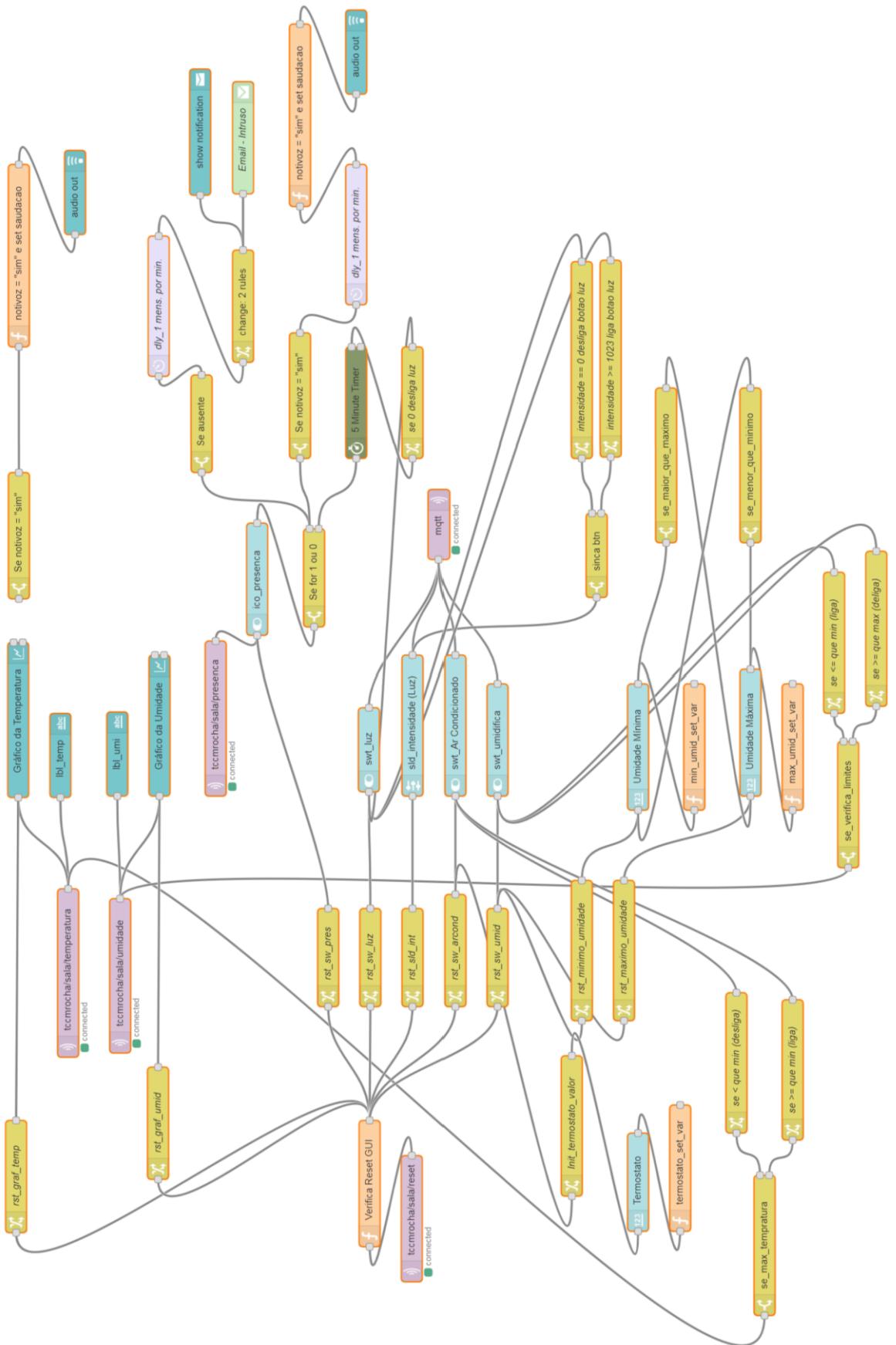


Figura 61: Node-RED (Flow) - node-sala  
Fonte: (O autor, 2018)

#### 4.5 PROTÓTIPO CONSTRUÍDO

Esta seção apresenta o protótipo que foi construído com os elementos reais, porém, não instalados em uma casa real. As montagens foram realizadas em bancada por questões práticas. A Figura 62 mostra o Raspberry Pi, onde roda o MQTT *broker* (servidor Mosquito) e o serviço Node-RED.

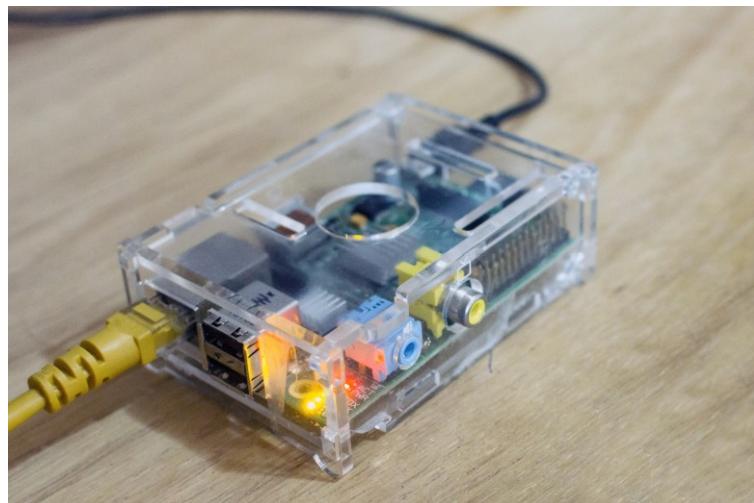


Figura 62: Raspberry Pi - Servidor MQTT  
Fonte: (O autor, 2018)

A Figura 63 mostra a montagem em *protoboard* de três nós NodeMcu do tipo 1, daqueles que ficam no quarto, na cozinha, no banheiro e na garagem.

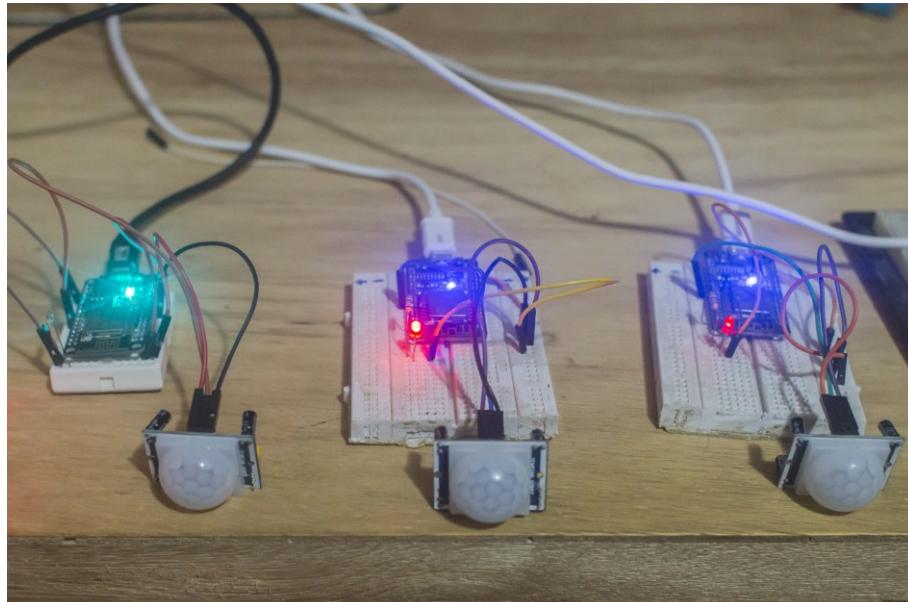


Figura 63: Protótipo dos nós NodeMcu do tipo 1  
Fonte: (O autor, 2018)

A Figura 64 mostra o protótipo do nó NodeMcu do tipo 2 usado na cisterna. Este nó usa somente o sensor ultrassônico.

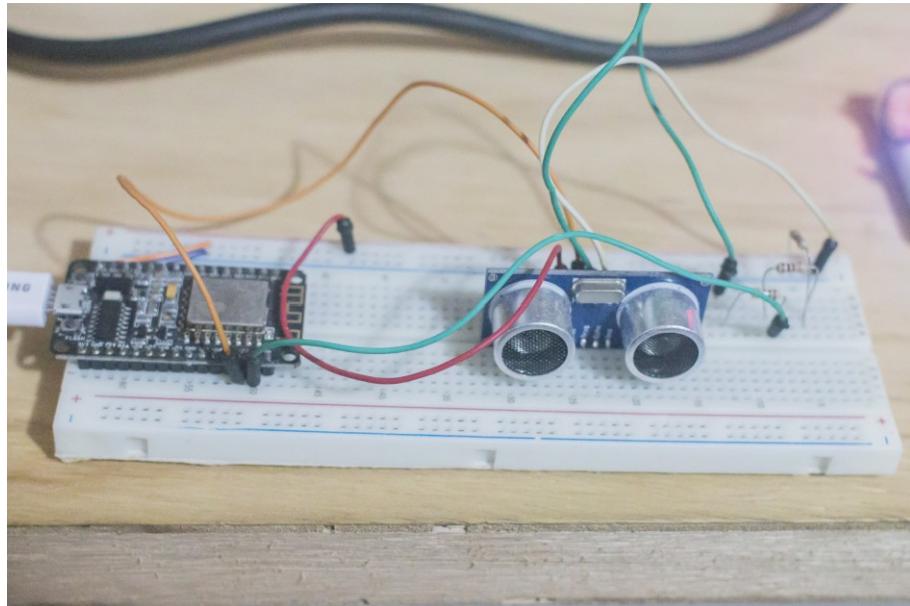


Figura 64: Protótipo do nó NodeMcu do tipo 2  
Fonte: (O autor, 2018)

O protótipo do nó NodeMcu do tipo 3, pode ser visto na Figura 65. Este nó é o que interage com mais sensores e também atua em mais *leds*.

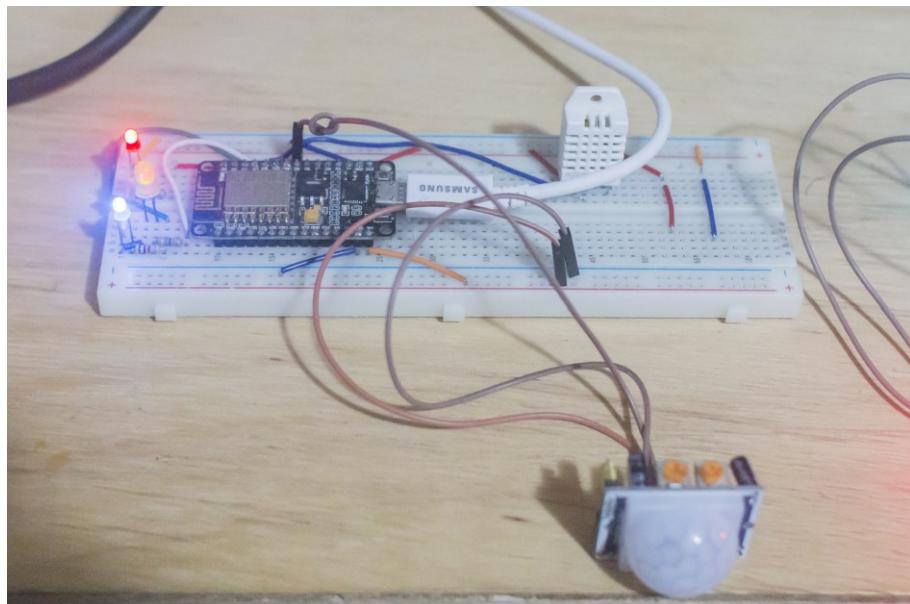


Figura 65: Protótipo do nó NodeMcu do tipo 3  
Fonte: (O autor, 2018)

A Figura 66 mostra o protótipo completo, ou seja, a rede de seis nós NodeMcu da casa inteligente.

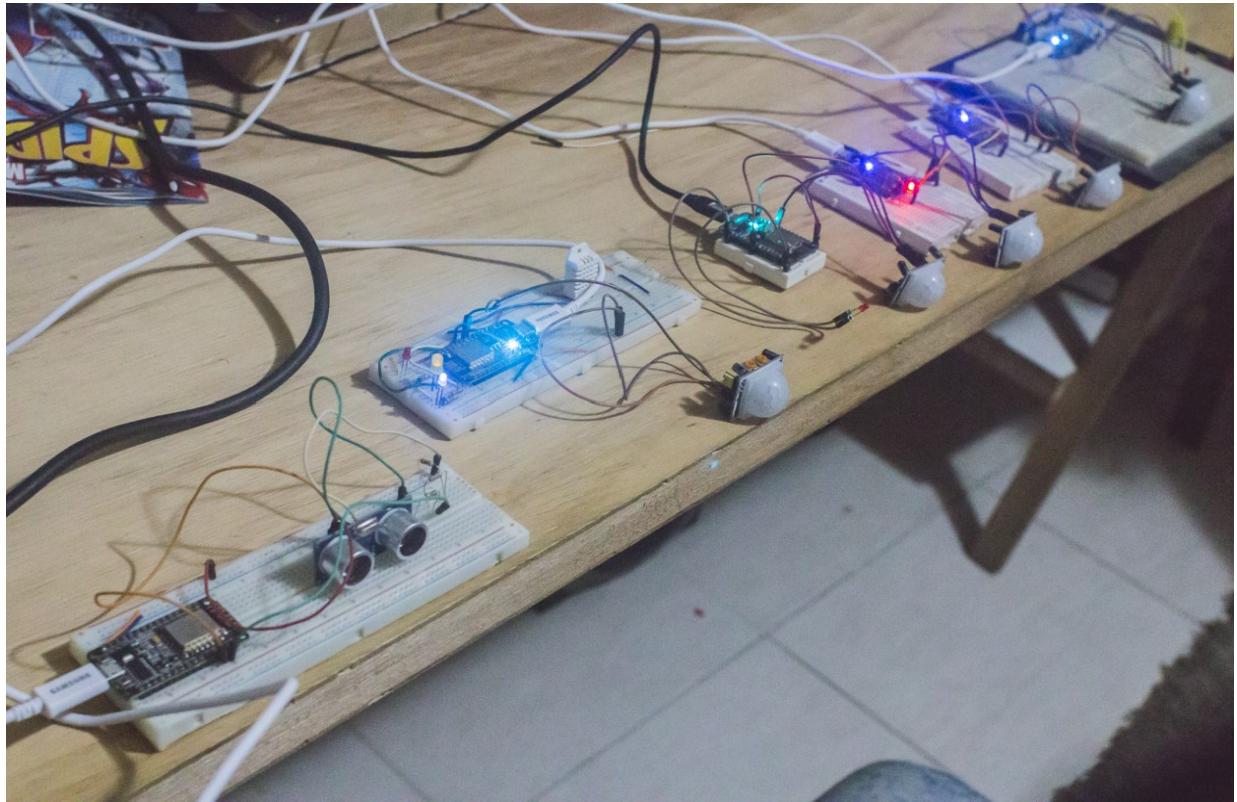


Figura 66: Protótipo completo da rede de nós NodeMcu  
Fonte: (O autor, 2018)

A Figura 67 mostra uma imagem da aplicação rodando em um Tablet da Samsung com sistema Android.

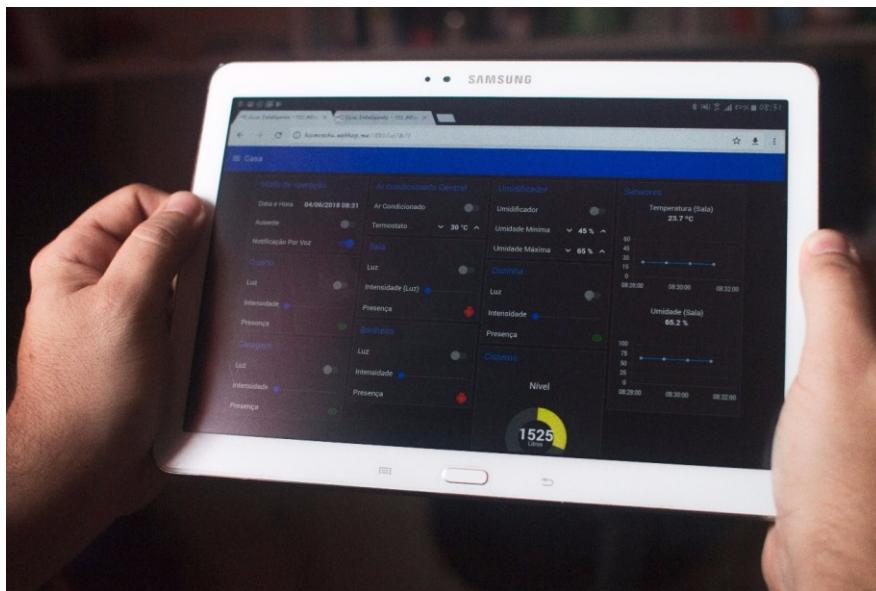


Figura 67: Foto da aplicação rodando no Tablet Samsung  
Fonte: (O autor, 2018)

A Figura 68 mostra uma imagem da aplicação rodando também em um celular LG X cam.

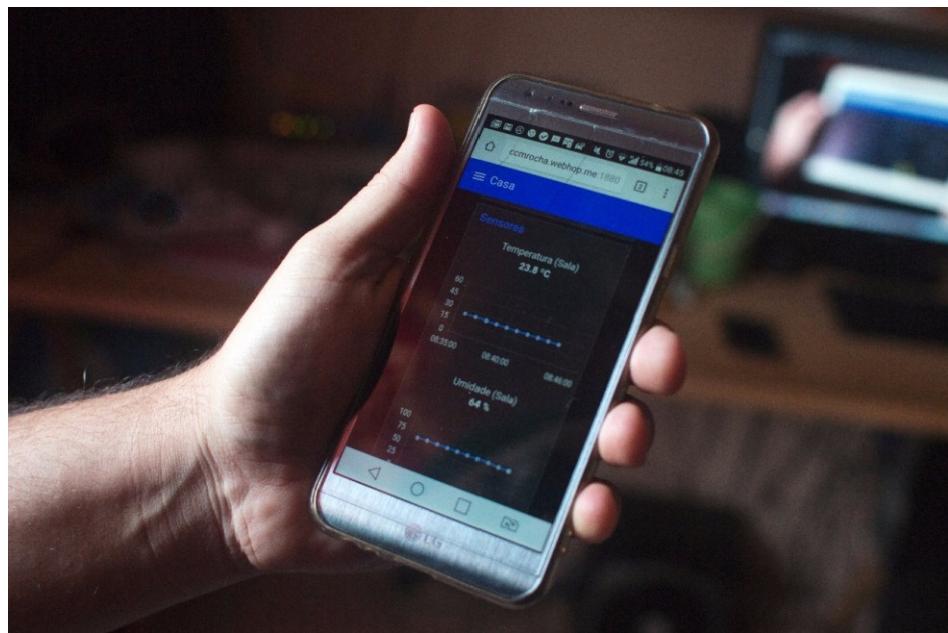


Figura 68: Foto da aplicação rodando em um celular modelo LG X cam  
Fonte: (O autor, 2018)

## CONCLUSÕES E TRABALHOS FUTUROS

Após a definição do tema deste trabalho, eu comecei a pesquisar sobre o assunto Internet das Coisas. Eu tinha alguma experiência no desenvolvimento de alguns projetos de computação física com o Arduino e já tinha ouvido falar do termo, porém não tinha idéia da dimensão e do poder de transformação da IoT.

Pude perceber, com profundidade, a ideia de ubiquidade e constatar que a IoT é uma tendência sem limites que vai mudar a nossa maneira de viver, afetando vários setores da sociedade. A vida das pessoas está sendo transformada através da automação residencial com as casas inteligentes. Cidades estão reduzindo custos e economizando energia usando a IoT. Fábricas “inteligentes” estão aprimorando o seu processo de produção usando o conceito da “Indústria 4.0” que tem como sinônimos os termos manufatura avançada e indústria inteligente. O setor automotivo já possui um pneu inteligente que é capaz de monitorar as condições de pressão e desgaste e se comunicar com o computador de bordo do carro, agendando futuras revisões. Dispositivos vestíveis monitoram atividades. Na medicina há um projeto open-source de um pâncreas artificial chamado *OpenAPS*. Na agricultura, o cultivo inteligente baseado em IoT é altamente eficiente quando comparado com a abordagem convencional.

Foi uma experiência muito rica usar somente tecnologias open-source para a construção do protótipo da Casa Inteligente. Sou fã do GNU/Linux e da filosofia open-source já há algum tempo e vejo como é possível aprender muito mais fazendo uso dessas ferramentas.

Quando se trata de sistemas open-source, nem tudo é tão simples. Muitas das ferramentas de software e de hardware não possuem uma boa documentação, muitas vezes ela é escassa e existe muita informação confusa e divergente. Eu tive bastante dificuldade na hora de publicar em vários tópicos MQTT usando o NodeMcu. Alguns exemplos na documentação do NodeMcu são fora de contexto e muitas vezes são pequenos fragmentos de código mal comentados. Foi preciso testar várias ideias e gastar muitas horas trabalhando em cima dos *firmwares* em linguagem Lua até encontrar uma solução. Foi

muito difícil implementar a parte do sensor de ultrassom com o NodeMcu usando Lua. Na verdade o funcionamento dessa função ainda demonstra instabilidade e falta de precisão.

No geral as tecnologias usadas no trabalho corresponderam às expectativas e o sistema se comportou como o esperado.

Todo este trabalho foi feito somente com ferramentas open-source, desde o editor de textos até as ferramentas de desenho vetorial e editores de imagem. Eu gostaria de citá-los:

- Inkscape (Desenho vetorial)
- Gimp (Editor de imagem)
- Fritzing (Criação de esquemas eletrônicos em *protoboard*)
- LibreOffice (Editor de textos)
- Geany e Atom (Editores de código fonte)
- Deepin Scrot e Shutter (Print screen e edição de gráficos)
- Debian OS
- Ubuntu OS
- Raspbian OS

Como trabalhos futuros, há algumas áreas no desenvolvimento da IoT que podem ser exploradas. A questão da segurança e da privacidade em IoT é muito importante. Com carros, cidades inteiras e indústrias conectadas à Internet, o acesso indevido a essas redes pode ser catastrófico. Imagine um carro ou um reator nuclear sendo hackeado. Outro desafio que pode servir como ponto de partida para um projeto futuro é a questão da conectividade. Os sistemas de IoT usam muitas tecnologias de transmissão sem fio como Wi-Fi, Bluetooth, LoRa WAN, SigFox, Zigbee e etc. Cada um desses sistemas possui suas próprias vantagens e especificações. A implementação de plataformas de tecnologia múltipla é um desafio para os desenvolvedores, uma vez que a informação deve ser compartilhada entre dispositivos e aplicativos.

O meu objetivo é seguir meus estudos e ingressar no mestrado. A minha área de interesse é a Inteligência artificial e as redes neurais artificiais. Pretendo também intensificar os experimentos sobre as arquiteturas exploradas neste trabalho, talvez definir novas arquiteturas e cenários de IoT, e publicar um livro com o mesmo título desta monografia.

## REFERÊNCIAS BIBLIOGRÁFICAS

ARDUINO&CIA. **Sensor de umidade e temperatura DHT11.** Disponível em: <<https://www.arduinoecia.com.br/2013/05/sensor-de-umidade-e-temperatura-dht11.html>>. Acesso em: 10/05/2018.

BOLZANI, Caio Augustus M. **Residências inteligentes: Um curso de domótica.** São Paulo: Editora Livraria da Física, 2004.

CHAUHAN, Akshay. **Building a security alarm using Raspberry Pi.** Disponível em: <<https://hackernoon.com/building-a-security-alarm-using-raspberry-pi-725a18f42e84>>. Acesso em: 25/05/2018.

CURVELLO, André. **Arduino Yún.** Disponível em: <<https://www.embarcados.com.br/arduino-yun/>>. Acesso em: 13/05/2018.

DARNELL, Lucas. **The Internet of Things: A Look at Real-World Use Cases and Concerns.** : Lucas Darnell, 2015.

DELOURA, Mark. **The Engine Survey: General Results.** Disponível em: <<http://www.satori.org/2009/03/the-engine-survey-general-results/>>. Acesso em: 26/05/2018.

HAWKINS, Matt. **Simple Guide to the Raspberry Pi gpio Header and Pins.** Disponível em: <<https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/>>. Acesso em: 25/05/2018.

HiveMQ. **Interprise MQTT broker.** Disponível em: <<https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe>>. Acesso em: 22/05/2018.

IERUSALIMSCHY, Roberto. **Programando em Lua**: Tradução de Ana Lúcia DeMoura. 3. ed. Rio de Janeiro: LTC, 2015.

LEA, Rodger. **Node-RED: Lecture 1: A brief introduction to Node-RED**. Disponível em: <<http://noderedguide.com/nr-lecture-1/>>. Acesso em: 10/05/2018.

LIGHT, Roger A. **Mosquitto: server and client implementation of the MQTT protocol**, The Journal of Open Source Software, vol. 2, no. 13, May 2017, DOI: 10.21105/joss.00265.

MOTA, Allan. **Sensor de temperatura LM35 – Medindo temperatura com Arduino**. Disponível em: <<https://portal.vidadesilicio.com.br/lm35-medindo-temperatura-com-arduino/>>. Acesso em: 26/05/2018).

NodeMCU. **NodeMCU custom builds**. Disponível em: <<https://nodemcu-build.com/>>. Acesso em: 20/10/2017.

NodeMCU. **NodeMCU Documentation: GPIO Module**. Disponível em: <<https://nodemcu.readthedocs.io/en/latest/en/modules/gpio/>>. Acesso em: 20/10/2017.

PENNINKHOF, Jan. **Getting started with the ESP8266**. Disponível em: <<https://www.-penninkhof.com/2015/01/getting-started-with-the-esp8266/>>. Acesso em: 26/05/2018.

SILVA, Everton; et al. **Computação Ubíqua: Definição e Exemplos**. 2015. Disponível em: <<https://seer.imed.edu.br/index.php/revistasi/article/download/926/739>>. Acesso em: 27/05/2018.

SINGH, Pradeep. **Esp8266**. Disponível em: <<https://iotbytes.wordpress.com/esp8266-2/>>. Acesso em: 01/03/2018.

STANFORD-CLARK, Andy; TRUONG, Hong Linh. **MQTT For Sensor Networks (MQTT-SN): Protocol Specification**. Disponível em: <[http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN\\_spec\\_v1.2.pdf](http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf)>. Acesso em: 07/05/2018.

THE MATHWORKS, Inc. **Analyze Your Data.** Disponível em: <[https://www.mathworks.com/help/thingspeak/analyze-your-data.html#visualize\\_dew\\_point\\_measurement](https://www.mathworks.com/help/thingspeak/analyze-your-data.html#visualize_dew_point_measurement)>. Acesso em: 20/10/2017.

THOMSEN, Adilson. **Como comunicar com o Arduino Ethernet Shield W5100.** Disponível em: <<https://www.filipeflop.com/blog/tutorial-ethernet-shield-w5100/>>. Acesso em: 05/05/2018.

UPTON, Eben; HALFACREE, Gareth. **Raspberry Pi User Guide 2nd Edition.** United Kingdom: Wiley, 2014.

VEIGA, Ricardo. **Wireless RFID door lock using NodeMcu.** Disponível em: <<http://www.instructables.com/id/Wireless-RFID-Door-Lock-Using-Nodemcu/>>. Acesso em: 25/05/2018.

## APÊNDICE A – Esp8266 (fazendo o upload do *firmware* Lua)

Para usar o Esp8266 com a linguagem Lua, é necessário fazer o *upload* do *firmware* que roda o interpretador Lua. Normalmente, o *firmware* que vem gravado por padrão é o *firmware* AT. Todos os protótipos implementados neste trabalho fizeram uso do módulo Esp12E NodeMcu. Devido à grande quantidade de bibliotecas (ou módulos) que estão surgindo para o NodeMcu, agora não é mais disponibilizado um *firmware* Lua padrão, pronto para download.

Um *firmware* customizado pode ser gerado no site do NodeMcu<sup>53</sup>. É importante não confundir o uso do termo módulo, usado para referenciar a placa Esp8266 (hardware) com o termo módulo usado para se referir às bibliotecas (software).

Neste site cada módulo pode ser excluído ou adicionado na customização. Assim pode-se economizar espaço selecionando somente as bibliotecas necessárias ao projeto. Por exemplo, se a aplicação não trabalha com sinal pwm, não será necessário ter este módulo no *firmware*, assim economizando memória. Para ter acesso ao *firmware* customizado é preciso fornecer um e-mail para receber o link para download. Duas versões do *firmware* serão disponibilizadas, uma que será capaz de trabalhar com ponto flutuante e outra que trabalhará somente com números inteiros. Elas também diferem em tamanho, a que trabalha só com números inteiros ocupa um pouco menos de memória. O tempo de espera, para a geração do *firmware* e a disponibilização do link no e-mail, fica em torno de 10 minutos.

Existem basicamente duas maneiras de gravar o *firmware* no Esp8266. Primeiramente será mostrada a que usa puramente a linha de comando:

- (1) Para fazer o *upload* no Linux você pode usar o programa em python chamado:

```
esptool.py
```

Basta baixá-lo usando:

```
git clone https://github.com/themadinventor/esptool.git
```

---

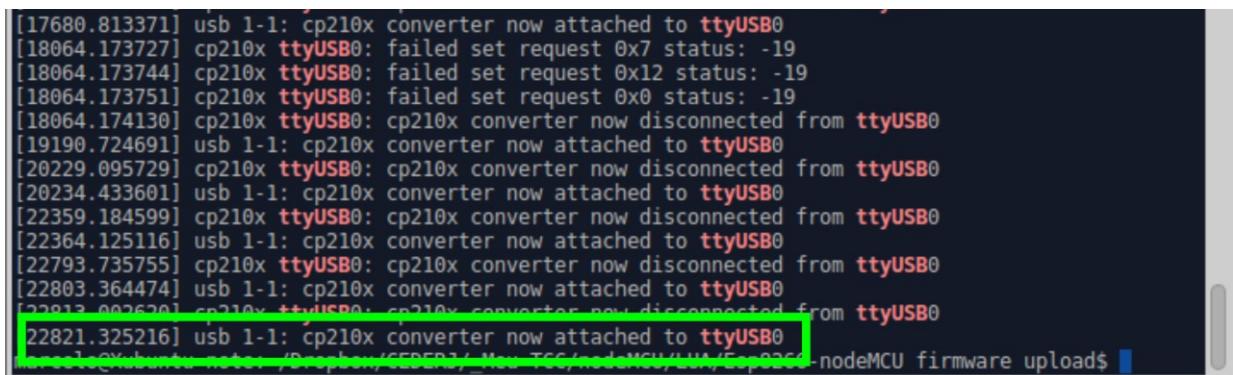
<sup>53</sup> <https://nodemcu-build.com/>

Após baixar é só entrar na pasta `esptool` e digitar o comando abaixo para instalar a ferramenta.

```
sudo python setup.py install
```

É preciso conectar o Esp8266 na porta usb e digitar o comando a seguir para ver em que porta ele está aparecendo no sistema. A Figura 69 mostra que ele está na porta `ttyUSB0`.

```
dmesg | grep tty
```



```
[17680.813371] usb 1-1: cp210x converter now attached to ttyUSB0
[18064.173727] cp210x ttyUSB0: failed set request 0x7 status: -19
[18064.173744] cp210x ttyUSB0: failed set request 0x12 status: -19
[18064.173751] cp210x ttyUSB0: failed set request 0x0 status: -19
[18064.174130] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[19190.724691] usb 1-1: cp210x converter now attached to ttyUSB0
[20229.095729] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[20234.433601] usb 1-1: cp210x converter now attached to ttyUSB0
[22359.184599] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[22364.125116] usb 1-1: cp210x converter now attached to ttyUSB0
[22793.735755] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[22803.364474] usb 1-1: cp210x converter now attached to ttyUSB0
[22813.002620] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[22821.325216] usb 1-1: cp210x converter now attached to ttyUSB0
root@raspberrypi:~/Downloads/CEP071/Mou_TCC/nodemcu-firmware_upload$
```

Figura 69: Verificando a porta associada ao Esp8266  
Fonte: (O autor, 2018)

Para fazer o upload do *firmware* é preciso copiar o arquivo do *firmware* para a pasta do `esptool` e executar a próxima linha de comando:

\*\*\* Lembrando que o programa `esptool.py` tem como dependência o módulo `python-serial` que pode ser instalado facilmente via `apt-get`.

```
sudo python esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash 0x00000 NodeMcu_float_0.9.6-dev_20150704.bin
```

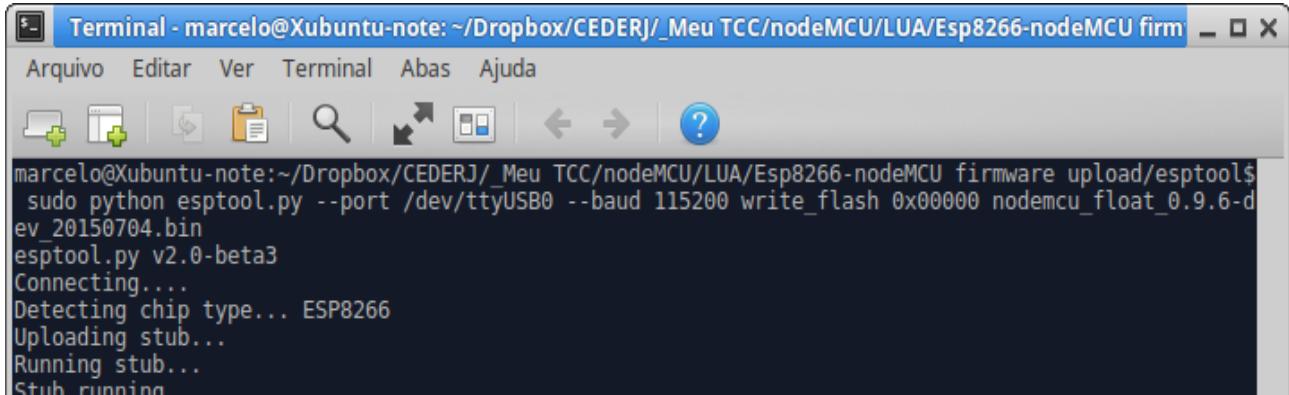
A Figura 70 mostra o progresso do upload do *firmware* para a placa.

Agora será mostrado como fazer o upload do *firmware* usando o PyFlasher, um *frontend* para o `esptool.py`.

Ele é escrito em Python, é baseado no `esptool.py` e tem como dependência as bibliotecas `pyserial` e `wxpython` que podem ser instaladas facilmente nas distribuições GNU/Linux.

O PyFlasher pode ser baixado através do link:

<https://github.com/marcelstoer/nodemcu-pyflasher.git>



```
Terminal - marcelo@Xubuntu-note: ~/Dropbox/CEDERJ/_Meu TCC/nodeMCU/LUA/Esp8266-nodeMCU firm | _ □ X
Arquivo Editar Ver Terminal Abas Ajuda
marcelo@Xubuntu-note:~/Dropbox/CEDERJ/_Meu TCC/nodeMCU/LUA/Esp8266-nodeMCU firmware upload/esptool$ sudo python esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash 0x00000 nodemcu_float_0.9.6-dev_20150704.bin
esptool.py v2.0-beta3
Connecting...
Detecting chip type... ESP8266
Uploading stub...
Running stub...
Stub running...
```

Figura 70: Progresso do upload do *firmware* para a placa

Fonte: (O autor, 2018)

É possível rodar o programa da seguinte forma:

```
python nodemcu-pyflasher.py
```

A Figura 71 mostra a interface do PyFlasher. A seguir, uma explicação básica dos elementos da interface do programa.

- (1) Seleciona a porta do NodeMcu.
- (2) Seleção do *firmware* a ser gravado.
- (3) Escolha da velocidade de gravação. Nada tem a ver com a velocidade de comunicação da placa como a IDE Explorer.
- (4) A opção “yes, wipes all data” deve ser usada no caso em que o *firmware* já gravado na placa, tenha sido corrompido ou em casos de pane no *firmware*.
- (5) Inicia o processo de gravação do *firmware*.
- (6) Terminal de saída que exibe o processo de gravação.

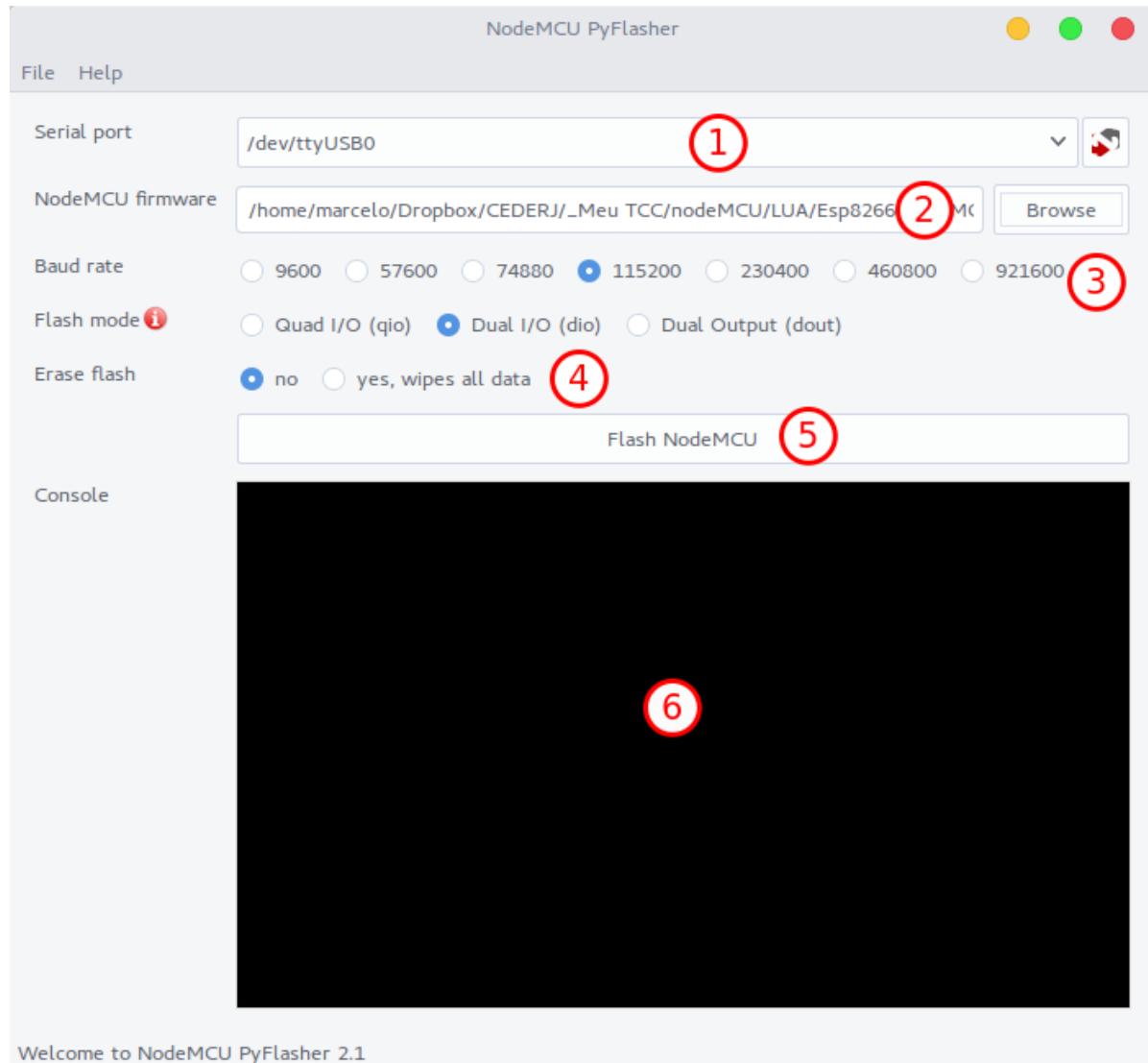


Figura 71: NodeMcu - PyFlasher  
Fonte: (O autor, 2018)

## APÊNDICE B – Usando a IDE Esplorer

Após o upload do *firmware* a ferramenta usada para escrever e fazer o upload dos códigos escritos em Lua é o Esplorer.

Deve-se baixar o Esplorer daqui:

<https://esp8266.ru/esplorer/>

Descompactar o arquivo e entrar na pasta.

Para executar o Esplorer digita-se na linha de comando `java -jar esplorer.jar` (é preciso ter o Java instalado). Para fazer o upload de código é necessário que o NodeMcu esteja ligado a uma porta usb.

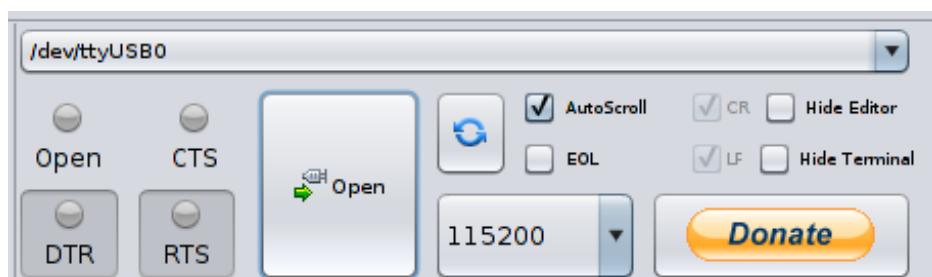


Figura 72: Conectando o NodeMcu à IDE Esplorer  
Fonte: (O autor, 2018)

```
NodeMCU custom build by frightanic.com
branch: master
commit: 443e8219527f5b2190324a969a4586f9d3d731bf
SSL: false
modules: adc,dht,file,gpio,http,mqtt,net,node,pwm,tmr,uart,wifi
build    built on: 2017-11-02 22:01
powered by Lua 5.1.4 on SDK 2.1.0(116b762)
lua: cannot open init.lua
>
```

Figura 73: Conexão da IDE com o NodeMcu  
Fonte: (O autor, 2018)

Após clicar em *open* é preciso apertar e soltar o botão de *reset* da placa (Figura 8). A Figura 73 exibe a tela com a conexão aberta entre a IDE e a placa. É possível ver os módulos que estão contidos no *firmware* e também a versão de Lua que é 5.1.4. No final

da imagem pode se ver uma mensagem dizendo que Lua não conseguiu abrir o arquivo init.lua. Toda vez que o NodeMcu é iniciado, seja no momento em que ele é energizado ou em um *reset*, o NodeMcu procura pelo arquivo init.lua e caso o encontre o executa. A função principal deste arquivo é chamar o módulo principal e fazer com a aplicação se inicie automaticamente.

A Figura 74 tem o objetivo de mostrar resumidamente algumas funções da IDE Esplorer. (1) Área de digitação do código fonte. (2) Monitor serial que serve para exibir as saídas dos comandos print e também usar o NodeMcu interativamente, como um Shell de comandos. (3) Gerenciador de arquivos que permite executar, compilar, renomear e remover um arquivo. (4) Botões para conectar e desconectar à placa, e selecionar a velocidade de conexão serial. (5) Área de Snippets.<sup>54</sup> (6) Caixa de entrada de comandos no modo interativo. (7) Menu de abertura, salvamento, criação e fechamento de arquivos no hd. (8) Faz o upload do código para a memória flash do NodeMcu.

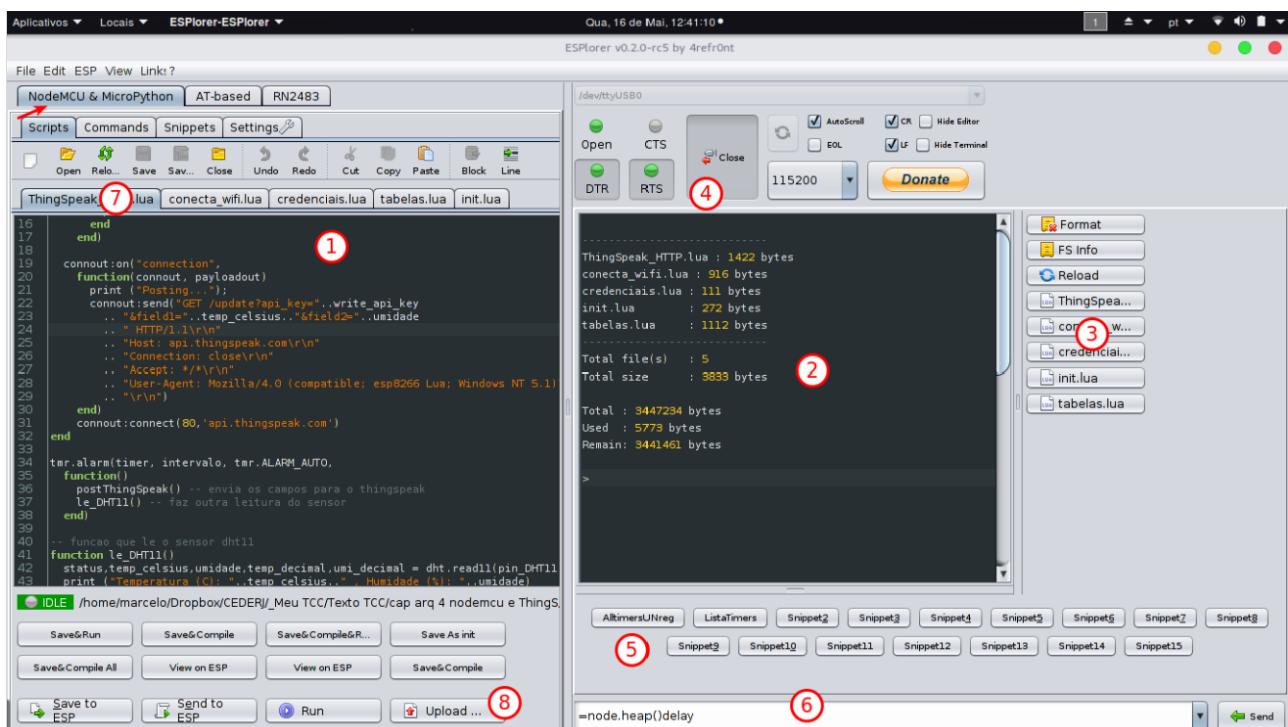


Figura 74: IDE Esplorer - Funções  
Fonte: (O autor, 2018)

A IDE, também oferece o recurso, de trabalhar com o NodeMcu usando o *firmware AT*. Para isso, é só selecionar a aba AT-based (Figura 75), na parte superior da interface, indicada pela seta vermelha. Este recurso é muito útil, principalmente para quem está co-

<sup>54</sup> São fragmentos de códigos que podem ser executados (disparados) através dos botões Snippets.

meçando a estudar comandos AT. Existem botões, que ao serem clicados, enviam código AT para o NodeMcu, permitindo assim interagir como a placa, de maneira fácil e amigável.

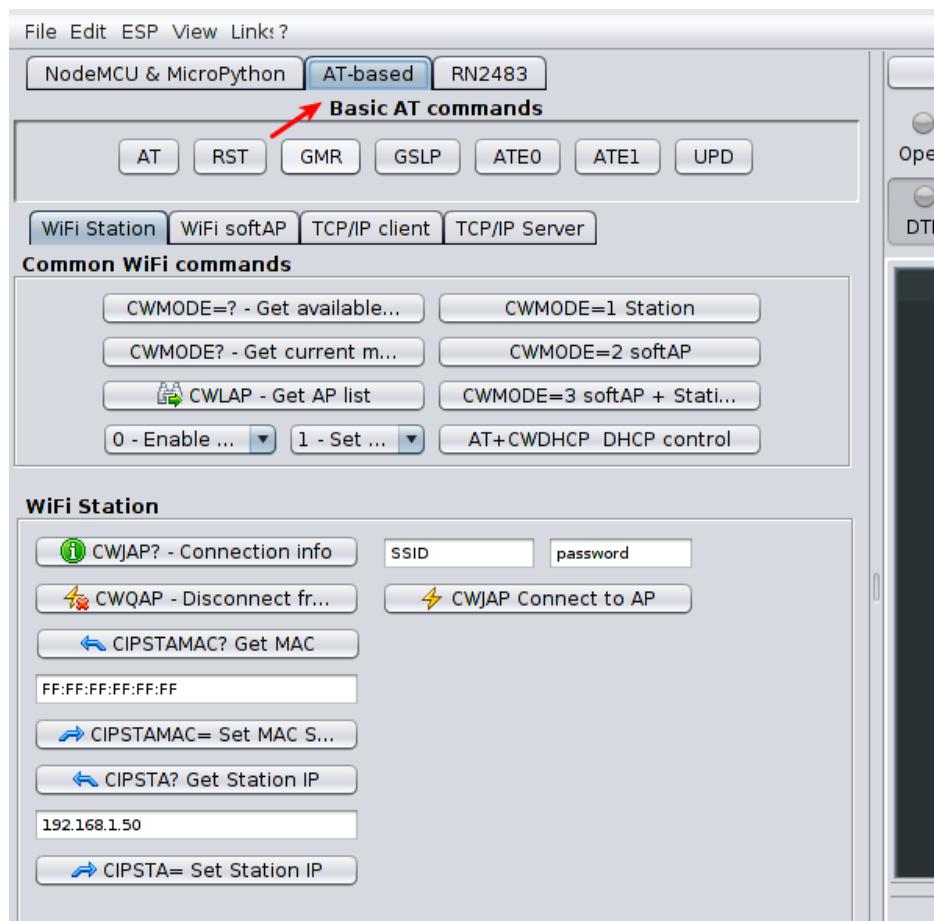


Figura 75: IDE Esplorer - Aba AT-based  
Fonte: (O autor, 2018)

## APÊNDICE C – Instalando e configurando o servidor Mosquitto

A distribuição GNU/Linux usada no Raspberry Pi, é a Raspbian, que é baseada no Debian 9 (*Stretch*). Para instalar o servidor Mosquitto no Raspbian, é preciso abrir um terminal e usar o comando *apt-get*. O Mosquitto já se encontra nos repositórios do Debian, sendo assim, é só digitar o comando no terminal:

```
sudo apt-get install mosquitto
```

Agora, é preciso instalar o cliente MQTT, usando o comando:

```
sudo apt-get install mosquitto-clients
```

A instalação do Mosquitto está feita e o seu serviço, por padrão, roda na porta 1883. Agora, será criado um usuário, e lhe será atribuída uma senha. Esses dados de login, foram utilizados na conexão como o Mosquitto, dentro do código Lua, no NodeMcu.

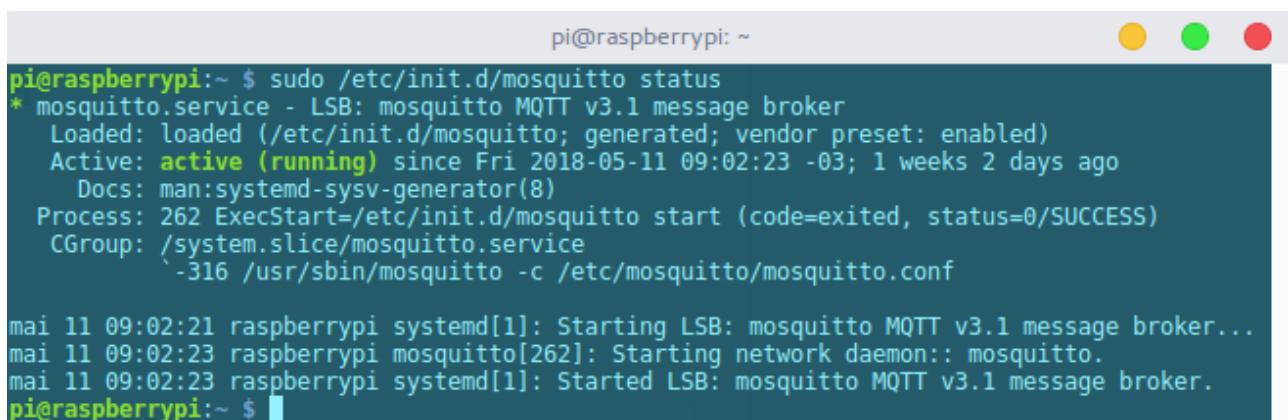
```
sudo mosquitto_passwd -c /etc/mosquitto/passwd marcelo
```

Será necessário entrar uma senha para o usuário *marcelo*.

Pode-se fazer um teste, a fim de verificar se o Mosquitto está rodando e conferir a versão instalada. Para isso, será usado o comando:

```
sudo /etc/init.d/mosquitto status
```

A Figura 76, mostra o status do serviço do Mosquitto na tela do terminal.



```
pi@raspberrypi:~ $ sudo /etc/init.d/mosquitto status
* mosquitto.service - LSB: mosquitto MQTT v3.1 message broker
  Loaded: loaded (/etc/init.d/mosquitto; generated; vendor preset: enabled)
  Active: active (running) since Fri 2018-05-11 09:02:23 -03; 1 weeks 2 days ago
    Docs: man:systemd-sysv-generator(8)
  Process: 262 ExecStart=/etc/init.d/mosquitto start (code=exited, status=0/SUCCESS)
  CGroup: /system.slice/mosquitto.service
          -316 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

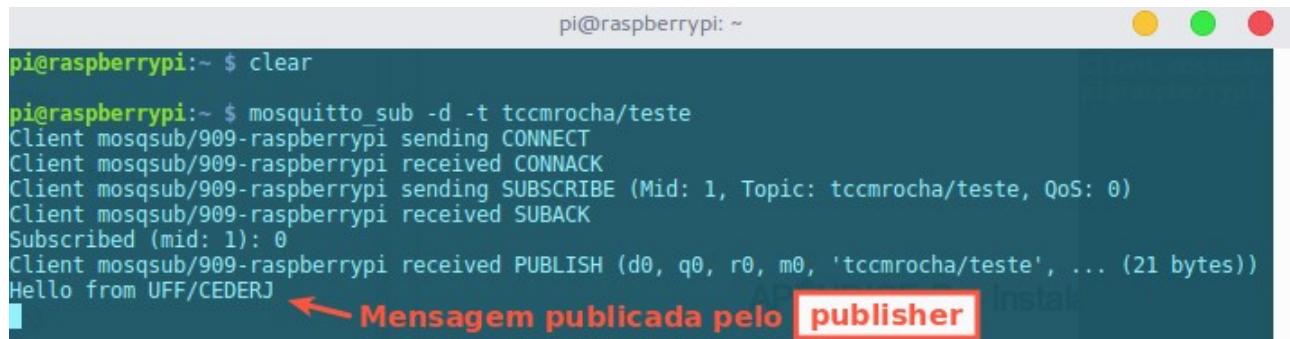
mai 11 09:02:21 raspberrypi systemd[1]: Starting LSB: mosquitto MQTT v3.1 message broker...
mai 11 09:02:23 raspberrypi mosquitto[262]: Starting network daemon:: mosquitto.
mai 11 09:02:23 raspberrypi systemd[1]: Started LSB: mosquitto MQTT v3.1 message broker.
pi@raspberrypi:~ $
```

Figura 76: Verificando o status do servidor Mosquitto

Fonte: (O autor, 2018)

É possível testar o servidor, criando um tópico, que receberá mensagens de um cliente, que publicará neste tópico. Para isso, será necessário usar duas telas de terminal. A Figura 77, mostra a tela do primeiro terminar. É preciso digitar o comando a seguir:

```
mosquitto_sub -d -t tccmrocha/teste
```



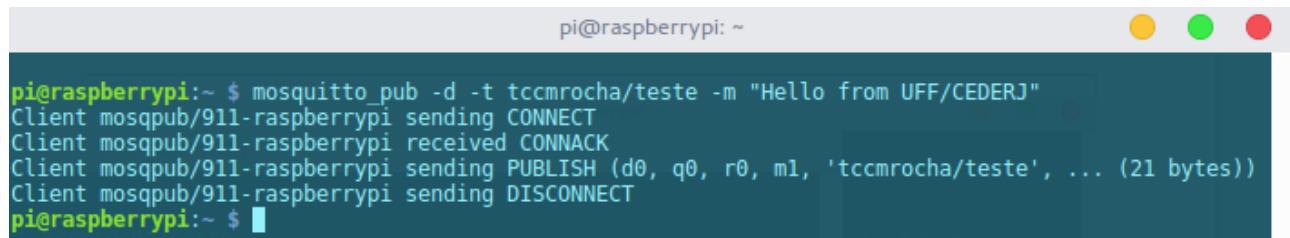
```
pi@raspberrypi:~ $ clear
pi@raspberrypi:~ $ mosquitto_sub -d -t tccmrocha/teste
Client mosqsub/909-raspberrypi sending CONNECT
Client mosqsub/909-raspberrypi received CONNACK
Client mosqsub/909-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: tccmrocha/teste, QoS: 0)
Client mosqsub/909-raspberrypi received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/909-raspberrypi received PUBLISH (d0, q0, r0, m0, 'tccmrocha/teste', ... (21 bytes))
Hello from UFF/CEDERJ
```

Figura 77: Criando um tópico de teste no Mosquitto

Fonte: (O autor, 2018)

Para publicar neste tópico, basta abrir outro terminal e entrar o comando:

```
mosquitto_pub -d -t tccmrocha/teste -m "Hello from UFF/CEDERJ"
```



```
pi@raspberrypi:~ $ mosquitto_pub -d -t tccmrocha/teste -m "Hello from UFF/CEDERJ"
Client mosqpub/911-raspberrypi sending CONNECT
Client mosqpub/911-raspberrypi received CONNACK
Client mosqpub/911-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'tccmrocha/teste', ... (21 bytes))
Client mosqpub/911-raspberrypi sending DISCONNECT
pi@raspberrypi:~ $
```

Figura 78: Publicação de teste no servidor Mosquitto

Fonte: (O autor, 2018)

A Figura 78, mostra o envio da informação e as respostas enviadas pelo servidor. E assim, se encerra, a instalação e configuração do Mosquitto, que são necessárias para o funcionamento da casa inteligente.

## APÊNDICE D – Instalando o Node-RED

O Node-RED tem como dependência o Node.js. Sendo assim, é preciso instalá-lo antes de tudo. O comando a seguir instalará o Node.js e o seu gerenciador de pacotes, o npm.

```
sudo apt-get install nodejs npm
```

Após a instalação, pode-se verificar a versão instalada do Node.js com o comando:

```
node --version
```

A versão do Node.js utilizada neste projeto é a 6.12.0. Para instalar o Node-RED, pode-se executar o script:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-deb-package/master/resources/update-nodejs-and-nodered)
```

Após o final da execução do script o Node-RED estará instalado. O comando a seguir levanta o servidor:

```
node-red-start
```

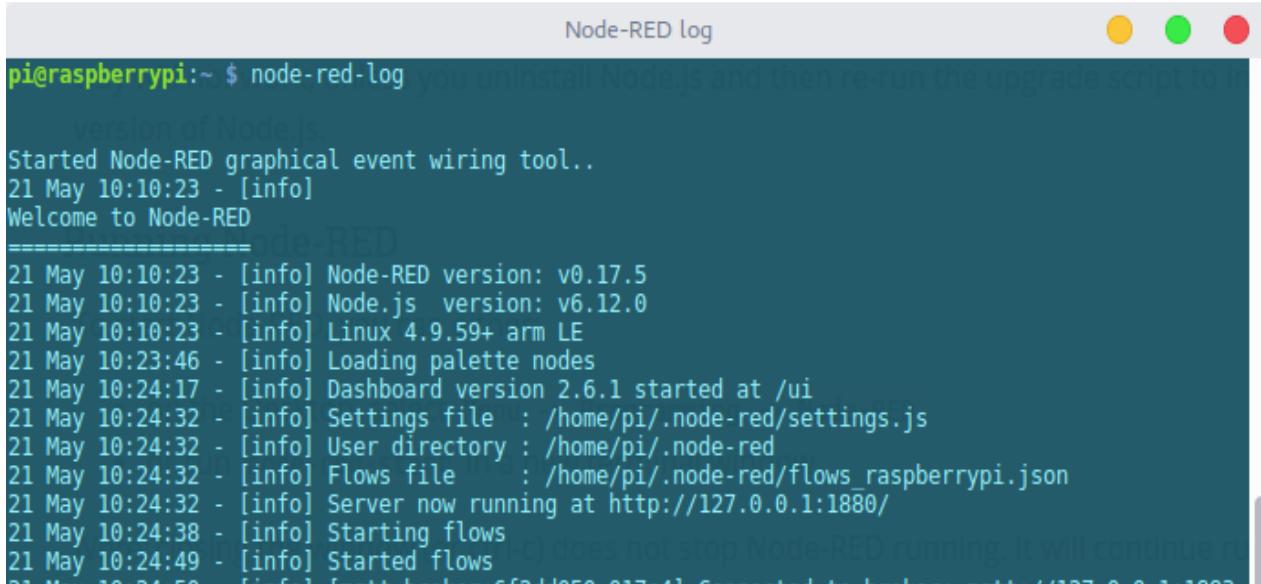
Para fazer com que o serviço rode automaticamente após a reiniciação do sistema, é só executar o comando:

```
sudo systemctl enable nodered.service
```

A Figura 79, mostra o resultado comando:

```
node-red-log
```

Após o comando, são exibidas as versões do Node-RED, do Node.js, do Kernel Linux, os diretórios de instalação, a porta (1880) em que o serviço Node-RED está rodando e também a porta (1883) do serviço do MQTT broker.



Node-RED log

pi@raspberrypi:~ \$ node-red-log

version of Node.js  
Started Node-RED graphical event wiring tool..  
21 May 10:10:23 - [info]  
Welcome to Node-RED  
=====

21 May 10:10:23 - [info] Node-RED version: v0.17.5  
21 May 10:10:23 - [info] Node.js version: v6.12.0  
21 May 10:10:23 - [info] Linux 4.9.59+ arm LE  
21 May 10:23:46 - [info] Loading palette nodes  
21 May 10:24:17 - [info] Dashboard version 2.6.1 started at /ui  
21 May 10:24:32 - [info] Settings file : /home/pi/.node-red/settings.js  
21 May 10:24:32 - [info] User directory : /home/pi/.node-red  
21 May 10:24:32 - [info] Flows file : /home/pi/.node-red/flows\_raspberrypi.json  
21 May 10:24:32 - [info] Server now running at http://127.0.0.1:1880/  
21 May 10:24:38 - [info] Starting flows  
21 May 10:24:49 - [info] Started flows  
21 May 10:24:50 - [info] [full logs] See https://nodered.org/docs/logging/full-logs for more information about how to view the full log output.

Figura 79: Tela de log do serviço Node-RED

Fonte: (O autor, 2018)

## APÊNDICE E – Código do servidor HTTP

Embutido nos nós NodeMcu do tipo (1) e tipo (3), um servidor HTTP retorna uma página com informações sobre o hardware da placa e também detalhes a respeito da rede. Pode ser acessado via navegador de Internet usando o endereço de ip do nó NodeMcu.

```

1  --Casa Inteligente - Servidor HTTP
2  --Autor: Marcelo Marques da Rocha
3  --CEDERJ/Universidade Federal Fluminense
4  --TCC - Desenvolvimento Open-Source para a Internet das Coisas
5  --(Arquiteturas para Interfaces Web e Móvel)
6  --Orientador: Professor Dr. Luciano Bertini
7
8  -- servidor http que mostra uma pagina com informacoes sobre o hardware do dispositivo
9  http_server = net.createServer(net.TCP)
10
11 -- servidor passa a ouvir na porta 80 (porta padrao http)
12 http_server:listen(80,
13     function(conn) -- funcao callback que responde apos uma conexao ser efetuada
14         conn:on("receive",
15             function(conn, payload) -- apos conexao, responde com a pagina com os dados
16                 print(payload)
17                 pagina = '<!doctype html>' ..
18                 '<html>' ..
19                 '<head>' ..
20                 '<meta charset="utf-8"><meta http-equiv="refresh" content="60">' ..
21                 '<title>informações do dispositivo' .. node_nome .. '</title>' ..
22                 '</head>' ..
23                 '<body>' ..
24                 '<div align="center"> <tb colspan="2"></tb><tb></tb><tb colspan="2"></tb><tb></tb>' ..
25                 '<table style="width: 60%; border="2">' ..
26                 '<tbody>' ..
27                 '<tr>' ..
28                 '<td rowspan="1" colspan="4" style="text-align: center;">' ..
29                 '<h3><b>informações do dispositivo - ' .. node_nome .. '</b></h3>' ..
30                 conn:send(pagina) -- envia fragmento da pagina
31                 pagina = '</td>' ..
32                 '</tr>' ..
33                 '<tr>' ..
34                 '<td style="text-align: center;" colspan="2">' ..
35                 '<h3>hardware</h3>' ..
36                 '</td>' ..
37                 '<td colspan="2" style="text-align: center;">' ..
38                 '<h3>wifi settings</h3>' ..

```

Código 34: Código fonte - Servidor HTTP parte 1

```

39      '</td>' ...
40      '</tr>' ...
41      '<tr>' ...
42      '<td style="text-align: right; width: 20%;"><b>major version</b></td>' ...
43      '<td style="width: 30%;"> ' ... majorver ... '<br>' ...
44      '</td>' ...
45      '<td style="text-align: right; width: 20%;"><b>endereço ip</b><br>' ...
46      '</td>' ...
47      conn:send(pagina)-- envia fragmento da pagina
48      pagina =  '<td>' ... meu_ip ... '<br>' ...
49      '</td>' ...
50      '</tr>' ...
51      '<tr>' ...
52      '<td style="text-align: right;"><b>minor version</b></td>' ...
53      '<td> ' ... minorver .. '<br>' ...
54      '</td>' ...
55      '<td style="text-align: right;"><b>endereço mac</b><br>' ...
56      '</td>' ...
57      '<td> ' ... meu_mac .. '<br>' ...
58      '</td>' ...
59      '</tr>' ...
60      '<tr>' ...
61      conn:send(pagina)-- envia fragmento da pagina
62      pagina =  '<td style="text-align: right;"><b>dev. version</b></td>' ...
63      '<td> ' ... devver .. '<br>' ...
64      '</td>' ...
65      '<td style="text-align: right;"><b>ssid</b><br>' ...
66      '</td>' ...
67      '<td> ' ... meu_ssid .. '<br>' ...
68      '</td>' ...
69      '</tr>' ...
70      '<tr>' ...
71      '<td style="text-align: right;"><b>chip id</b></td>' ...
72      '<td> ' ... chipid .. '<br>' ...
73      '</td>' ...
74      '<td style="text-align: right;"><b>wifi mode</b><br>' ...
75      '</td>' ...
76      '<td> ' ... modo_wifi[wifi.getmode()] .. '<br>' ...
77      conn:send(pagina)-- envia fragmento da pagina
78      pagina = '</td>' ...
79      '</tr>' ...
80      '<tr>' ...
81      '<td style="text-align: right;"><b>flash id</b></td>' ...
82      '<td colspan="1"> ' ... flashid .. '<br>' ...
83      '</td>' ...
84      '</tr>' ...
85      '<tr>' ...

```

Código 35: Código fonte - Servidor HTTP parte 2

```

87      '<td colspan="1"> ' .. flashsize .. ' kbytes<br>' ..
88      '</td>' ..
89      '</tr>' ..
90      '<tr>' ..
91      '<td style="text-align: right;"><b>flash mode</b></td>' ..
92      '<td colspan="1"> ' .. flashmode .. '<br>' ..
93      '</td>' ..
94      '</tr>' ..
95      '<tr>' ..
96      conn:send(pagina)-- envia fragmento da pagina
97      pagina = '<td style="text-align: right;"><b>flash speed</b></td>' ..
98      '<td colspan="1"> ' .. (flashspeed/1000/1000) .. ' mhz<br>' ..
99      '</td>' ..
100     '</tr>' ..
101     '</tbody>' ..
102     '</table>' ..
103     '</div>' ..
104     '<p style="text-align: center;"> protótipo desenvolvido por <b>' ..
105     'marcelo marques da rocha</b><br>' ..
106     'aluno do curso: <b>tecnologia em sistemas de computação</b><br>' ..
107     'instituição: <b>cederj / uff </b><br>' ..
108     'ano: <b>2018</b></p>' ..
109     '</body>' ..
110     '</html>' ..
111     conn:send(pagina)-- envia fragmento da pagina
112   end
113 )
114 -- registra a função que executa apos algo ser enviado
115 conn:on("sent",
116   function(conn)
117     conn:close() -- fecha a conexao
118   end
119 )
120 end
121 )

```

Código 36: Código fonte - Servidor HTTP parte 3

## APÊNDICE F – Código do servidor Telnet

Um servidor Telnet está disponível nos nós NodeMcu do tipo (1) e (3). Ele funciona na porta 2323 e pode ser acessado através de um terminal, basta usar o comando:

```
telnet <ip do NodeMcu> 2323
```

O que esse servidor faz é espelhar, na saída do servidor Telnet, toda a informação que é enviada para o monitor serial.

```

1 --Casa Inteligente - Servidor Telnet
2 --Autor: Marcelo Marques da Rocha
3 --CEDERJ/Universidade Federal Fluminense
4 --TCC - Desenvolvimento Open-Source para a Internet das Coisas
5 --(Arquiteturas para Interfaces Web e Móvel)
6 --Orientador: Professor Dr. Luciano Bertini
7
8 -- Cria o objeto servidor do TCP
9 telnet_serv = net.createServer(net.TCP)
10 -- Escuta na porta 2323
11 telnet_serv:listen(2323,
12     function(socket)
13         print("\nServidor telnet rodando...")
14         sck_con = socket
15         -- funcao que recebe a saida da serial (str) e imprime no terminal
16         function serial_output(str)
17             if(sck_con~=nil) then
18                 sck_con:send(str)
19             end
20         end
21         -- redireciona a saida da serial para a funcao serial_output
22         -- A opcao 1 mantem a saida da serial também ativa
23         node.output(serial_output, 1)
24         -- ao desconectar executa a callback
25         socket:on("disconnection",
26             function(socket)
27                 sck_con = nil
28                 -- desabilita o redirecionamento para da porta serial
29                 node.output(nil)
30             end
31         )
32     end
33 )
```

Código 37: Código fonte - Servidor Telnet