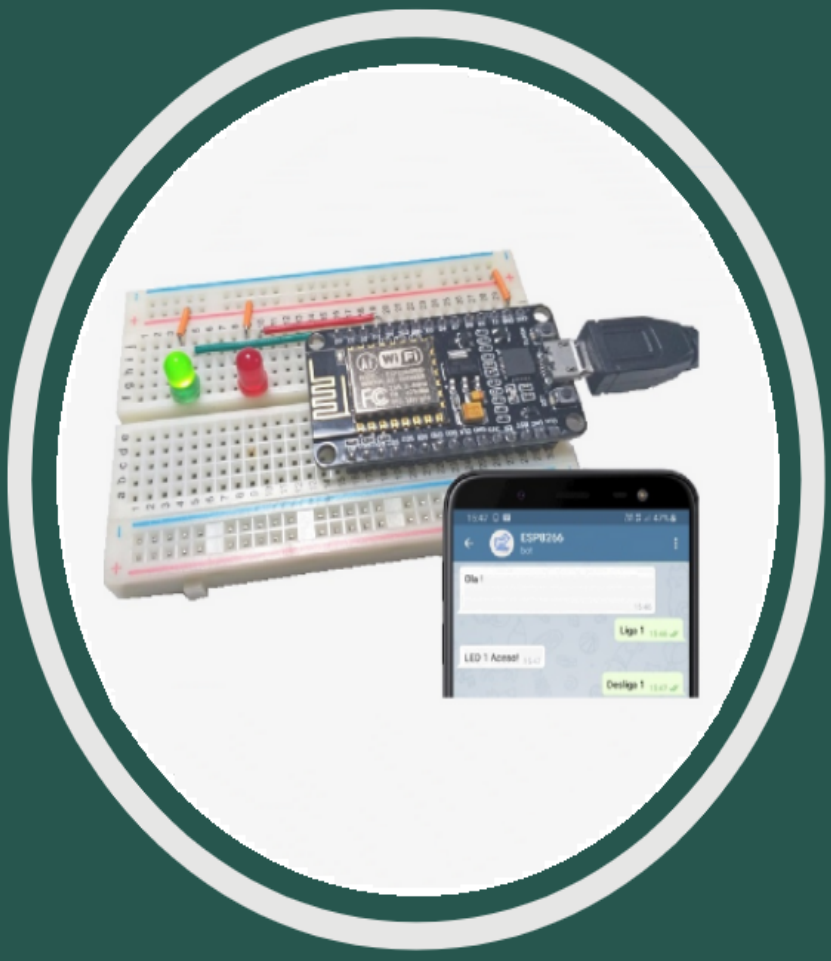
ESP8266 NODEMCU

DO PISCA LED À INTERNET DAS COISAS



João Alvarez Peixoto

JOÃO ALVAREZ PEIXOTO

ESP8266 NodeMCU: do pisca led à internet das coisas

1ª Edição

ISBN: 978-65-86105-23-0

Porto Alegre

UERGS

2021

Reitor
Leonardo Beroldt

Vice-Reitora
Sandra Monteiro Lemos

Pró-Reitora de Ensino
Rochelle da Silva Santaiana

Pró-Reitora de Extensão
Erli Schneider Costa

Pró-Reitor de Administração
Gabriel Borges da Cunha

Pró-Reitor de Pesquisa e Pós-Graduação
Rafael Haag

Universidade Estadual do Rio Grande do Sul
Reitoria: Rua 7 de Setembro, 1150, Centro, Porto Alegre – RS
CEP: 90.010-191 – Fone: (51) 32889000
www.uergs.edu.br

© 1. ed. 2021 – Universidade Estadual do Rio Grande do Sul (UERGS)
Qualquer parte desta publicação pode ser reproduzida, desde que citada a fonte.

Esta obra está licenciada com uma Licença Creative Commons Atribuição-Não Comercial-Sem Derivações 4.0 Internacional.

Esta licença permite que outros façam download dos seus trabalhos e os compartilhem desde que atribuam crédito a você, mas sem que possam alterá-los de nenhuma forma ou utilizá-los para fins comerciais

Texto da licença
https://creativecommons.org/licenses/by-nc-nd/4.0/

Catalogação da Publicação na Fonte (CIP)

P379e Peixoto, João Alvarez.
ESP8266 NodeMCU: do pisca led à internet das coisas/
João Alvarez Peixoto – 1.ed. - Porto Alegre: Uergs, 2021

210 p.; il.
ISBN: 978-65-86105-23-0

1. ESP8266 Node MCU. 2.Arduino. 3. Internet das coisas.
I. Peixoto, João Alvarez.
II. Universidade Estadual do Rio Grande do Sul.
III. Título.
CDU: 004.415 P379e

Ficha catalográfica por Simone Semensatto – CRB/10-1778

Revisão de normalização: Lucy Anne Rodrigues de Oliveira - CRB10/1545
Revisão de referências: Simone Semensatto – CRB/10-1778
Diagramação e Capa: João Alvarez Peixoto
Revisão textual: Taise Soares Peixoto Nascimento
Apresentação: João Alvarez Peixoto

ISBN: 978-65-86105-23-0

APRESENTAÇÃO

Em um mundo conectado, onde se tem informações sobre quase tudo, necessariamente há interação entre humanos e coisas. Na evolução digital, os computadores eram os meios de interação entre as pessoas, de forma que pacotes de informação de um indivíduo eram propagados a outro, através da internet. Mas, ainda assim, era necessário um recurso computacional para acessar a rede e propagar a informação, além da disponibilidade humana para proceder essa operação.

A ideia de coisas acessando a internet não demorou a surgir, pois se isto ocorre, as coisas podem interagir com humanos ou com outras coisas, autonomamente, bastando ter uma conexão com uma rede internet e protocolos adequados de interação.

E o que seria uma **coisa**? Tudo que não é humano! Na realidade, é tudo que possa ter acoplado a si um sistema computacional que permita o acesso à internet. E, com a evolução dos microcontroladores, já se pode ter recursos computacionais para esse propósito, cujo tamanho é pequeno o suficiente para ser acoplado à **coisa**, com a qual se deseja interação. Desta forma, fica fácil uma mala de bagagem ter em si mesma um microcontrolador e interagir com seu dono em hipótese de extravio, ou com o meio de transporte, a fim de certificar-se de que está sendo carregada para o destino certo. Neste caso, a mala de bagagem é uma **coisa** que interage no ambiente de **internet das coisas**.

O mercado evoluiu para uma grande oferta de microcontroladores e placas de desenvolvimento, que possibilitam a conexão através da internet, além de conexões lógicas de entrada e saída de sinais analógicos e digitais, permitindo o sensoriamento e atuação sobre a **coisa** a qual estará acoplado.

A placa de desenvolvimento ESP8266 NodeMCU é uma dessas ofertas, que possui funcionalidades de acesso à rede internet e portas de entrada e saída, onde o sensoriamento e atuação são realizados. É um componente de baixo custo, cuja programação pode ser realizada na linguagem C++ pela interface de desenvolvimento Arduino. Isto permite que pessoas com pouco conhecimento de programação e de circuitos eletrônicos possam realizar suas primeiras aplicações na **internet das coisas**.

A proposta deste livro é ser um guia, fornecendo as informações necessárias desde a configuração da placa de desenvolvimento até aplicações IoT (*Internet Of Things* – internet das coisas), desmistificando a complexidade de programação e servindo como um livro de anotações para o programador. Além de ser um motivador para os iniciantes adentrarem no mundo da programação.

Um bom começo para chegar a um grande fim.

Prof. Dr. João Alvarez Peixoto

ÍNDICE DE PROJETOS

PROJETO 1 – PISCA-LED COM 2 TEMPOS DISTINTOS.....44
PROJETO 2 – SISTEMA DE PARTIDA LIGA/DESLIGA EM UM LED.....47
PROJETO 3 – ACIONAMENTO DE UM LED COM EFEITO FADE-IN E FADE-OUT.....50
PROJETO 4 – POTENCIÔMETRO COMANDANDO LED COM EFEITO FADE-IN E FADE-OUT 52
PROJETO 5 – MONITOR SERIAL COMANDANDO LED.....55
PROJETO 6 – MONITOR DE TEMPERATURA AMBIENTE 63
PROJETO 7 – CONTADOR DE CAIXAS EM ESTEIRA COM *DISPLAY* MATRICIAL LCD-I2C 68
PROJETO 8 – MEDIDOR DE NÍVEL DE CAIXA D'ÁGUA COM *DISPLAY* MATRICIAL LCD-I2C.....73
PROJETO 9 – CONTROLADOR DE TEMPERATURA ON-OFF.....79
PROJETO 10 – COMANDO DE CARRO COM MOTORES CC ATRAVÉS DE BOTÕES.....83
PROJETO 11 – COMANDO DE CARRO COM MOTORES CC ATRAVÉS DE POTENCIÔMETRO COMO DIREÇÃO 85
PROJETO 12 – RELÓGIO CRONOMETRADO PARA PARTIDAS DE XADREZ 91
PROJETO 13 – CONTROLE DE ACESSO POR CARTÃO DE RFID.....97
PROJETO 14 – MONITOR DE TEMPERATURA E UMIDADE COM SENSOR DHT11.....101
PROJETO 15 – MONITOR DE REDES *WIFI* COM ESP8266 NODEMCU 107
PROJETO 16 – COMUNICAÇÃO ENTRE DOIS MÓDULOS ESP8266 NODEMCU ATRAVÉS DO ROTEADOR *WIFI*.....109
PROJETO 17 – COMANDO DE LUZES POR CHAVE DE TEMPO.....119
PROJETO 18 –PREVISÃO DO TEMPO.....121
PROJETO 19 – MEDIDOR DE UMIDADE E TEMPERATURA POR *WEB SERVER* 132
PROJETO 20 – COMANDOS PARA MÓDULO ESP8266 NODEMCU POR *WEB SERVER*....135
PROJETO 21 – COMANDO DE LUMINÁRIA E MONITORAMENTO DE TEMPERATURA E UMIDADE EM MQTT COM MÓDULO ESP8266 NODEMCU.....154
PROJETO 22 – TROCA DE MENSAGENS ENTRE ESP8266 NODEMCU E TELEGRAM.....163
PROJETO 23 – INTERAÇÃO ENTRE ESP8266 NODEMCU E O *BLYNK*.....175
PROJETO 24 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E *BLYNK* 178
PROJETO 25 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E ADAFRUIT 193

PROJETO 26 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E MQTT DASH.....203

ÍNDICE DE ILUSTRAÇÕES

Figura 1 – Placa de desenvolvimento ESP8266 NodeMCU 15
Figura 2 – Pinagem da placa ESP8266 NodeMCU 17
Figura 3 – Módulo ESP8266 NodeMCU conectado ao computador 19
Figura 4 – Arquivos compactados do *driver* de conversão Serial-USB CP210x 20
Figura 5 – Passos para instalação do *driver* conversor Serial-USB CP210x 20
Figura 6 – Driver CP210x instalado na COM3 do Gerenciador de Dispositivos do Windows 20
Figura 7 – Arquivos compactados do *driver* de conversão Serial-USB CH340G 21
Figura 8 – Passos para instalação do *driver* conversor Serial-USB CH340G 21
Figura 9 – Driver CH340 instalado na COM4 do Gerenciador de Dispositivos do Windows 21
Figura 10 – Download do arquivo de instalação do IDE do Arduino 22
Figura 11 – Passos para instalação do IDE do Arduino 22
Figura 12 – Configuração do caminho para o gerenciador de placas 23
Figura 13 – Inserção da placa ESP8266 NodeMCU no Gerenciador de Placas no IDE do Arduino 23
Figura 14 – Seleção da placa ESP8266 NodeMCU e seleção de porta de comunicação na IDE do Arduino 24
Figura 15 – Programa teste de funcionamento da instalação 24
Figura 16 – Carregamento do programa teste no módulo ESP8266 NodeMCU 25
Figura 17 – Funcionamento do programa de teste no monitor serial da IDE do Arduino 25
Figura 18 – Fluxo de um programa para o módulo ESP8266 NodeMCU na IDE Arduino 26
Figura 19 – Interpretação de um vetor de posições de memória 28
Figura 20 – Comportamento de uma função dentro de um programa 31
Figura 21 – Comportamento da estrutura *if-else* em um programa 32
Figura 22 – Comportamento da estrutura switch-case, execução de blocos condicionados, em um programa 33
Figura 23 – Comportamento da estrutura for, laço de repetição, em um programa 34
Figura 24 – Comportamento da estrutura while, execução condicional, em um programa 34
Figura 25 – Inserção de bibliotecas em um programa 35
Figura 26 – Protocolo UART e sua conexão com módulo ESP8266 NodeMCU 37
Figura 27 – Protocolo de comunicação serial I2C 38
Figura 28 – Protocolo SPI e sua pinagem 39
Figura 29 – Implementação de pisca-led no LED *Builtin*, utilizando a função delay() 41
Figura 30 – Implementação de pisca-led na saída digital D3, utilizando a função delay() 42
Figura 31 – Diagrama do projeto pisca-led com 2 tempos distintos 44

Figura 32 – Entrada digital com resistores de pull-down, pull-up, efeito bouncing e código para evitar ruídos 46
Figura 33 – Diagrama do projeto de sistema de partida liga/desliga em um LED 47
Figura 34 – Formas do sinal PWM na saída do microcontrolador 48
Figura 35 – Diagrama do projeto acionamento de um LED com efeito fade-in e fade-out 50
Figura 36 – Diagrama do projeto potenciômetro comandando o brilho do LED 52
Figura 37 – Diagrama e telas do projeto monitor serial comandando LED 55
Figura 38 – *Display* matricial com conexão I2C 58
Figura 39 – Módulo de acionamento de *display* matricial LCD com protocolo I2C 59
Figura 40 – Protocolo I2C de comunicação serial entre dispositivos 59
Figura 41- Instalação da biblioteca LiquidCrystal_I2C.h 60
Figura 42 – Diagrama exemplo de uma aplicação com *display* matricial LCD em protocolo I2C 60
Figura 43 – Diagrama para aplicação de reconhecimento de endereço I2C de dispositivos escravos 62
Figura 44 - Diagrama elétrico para o projeto monitor de temperatura ambiente 63
Figura 45 - Sensor indutivo e seu diagrama de ligação com módulo ESP8266 NodeMCU 66
Figura 46 - Sensor capacitivo e seu diagrama de ligação com o módulo 67
Figura 47 - Sensor ótico e seu diagrama de ligação com o módulo 67
Figura 48 – Diagrama do projeto contador de caixas em esteira com *display* matricial LCD-I2C 68
Figura 49 – Sensor ultrassônico HC-SR04, com diagrama de conexão e princípio de funcionamento 70
Figura 50 – Conversor lógico CNL35 e seu princípio de funcionamento 71
Figura 51 - Diagrama do projeto medidor de nivel de caixa d'água com *display* matricial LCD-I2C 73
Figura 52 – Driver transistorizado e relé eletromagnético como dispositivos periféricos 76
Figura 53 – Exemplo de acionamento de motor de corrente contínua com driver transistorizado e relé. 78
Figura 54 - Diagrama do projeto controlador de temperatura ON-OFF 79
Figura 55 – Ponte H L298N com sua pinagem e princípio de funcionamento 81
Figura 56 - Diagrama do projeto comando de carro com motores CC através de botões 83
Figura 57 - Diagrama do projeto comando de carro com motores CC através de potenciômetro como direção 85
Figura 58 – Princípio de funcionamento e diagrama de ligação de servo motor com módulo ESP8266 NodeMCU 87
Figura 59 – Instalação da biblioteca para acionamento de servo motor no módulo ESP8266 NodeMCU 88

Figura 60 – Exemplo de programa para acionamento de servo motor com módulo ESP8266 NodeMCU 89
Figura 61 - Diagrama do projeto relógio cronometrado para partidas de xadrez 91
Figura 62 – Módulo leitor de cartões RFID RC522 com pinagem e conexão com ESP8266 NodeMCU 93
Figura 63 – Biblioteca para leitura em cartão RFID MFRC522 e sua inserção na IDE do Arduino. 94
Figura 64 – Diagrama do programa leitor do ID de um cartão de RFID com módulo leitor RC522 96
Figura 65 - Diagrama e telas do projeto controle de acesso por cartão de RFID 97
Figura 66 – Pinagem do sensor DHT11 e inserção de suas bibliotecas 100
Figura 67 - Diagrama do projeto monitor de temperatura e umidade com sensor DHT11 101
Figura 68 – Ilustração de rede internet das coisas 102
Figura 69 - Estrutura de rede internet das coisas 103
Figura 70 – Módulo ESP8266 NodeMCU e sua conexão ao *modem* via *wifi* 104
Figura 71 – Exemplo de código para conexão do módulo ESP8266 NodeMCU em uma rede LAN *wifi* 105
Figura 72 – Conexão do módulo e tela de resultado esperado do projeto monitor de redes *wifi* 107
Figura 73 – Princípio da comunicação P2P com módulo ESP8266 NodeMCU através de roteador *wifi* 112
Figura 74 – Diagrama de aplicação exemplo como cliente e servidor 112
Figura 75 - Acesso do módulo ESP8266 NodeMCU a *sites* especializados via API 115
Figura 76 – Inserção das bibliotecas ESP8266WiFi e NTPClient 116
Figura 77 – Exemplo de acesso a horário sincronizado global com módulo ESP8266 NodeMCU 117
Figura 78 – Aplicação de comando de luzes por chave de tempo com ESP8266 NodeMCU 119
Figura 79 – Estrutura do projeto monitor de previsão do tempo 121
Figura 80 – Processo de assinatura no site *OpenWheather* 122
Figura 81 – Inserção das bibliotecas necessárias ao projeto previsão do tempo na IDE do Arduino. 123
Figura 82 - Estrutura de uma comunicação cliente-servidor do módulo ESP8266 NodeMCU na internet 128
Figura 83- Exemplo de programa para Conexão *web server* no módulo ESP8266 NodeMCU 128
Figura 84 – Projeto monitor de umidade e temperatura via *web service* 132
Figura 85 – Estrutura do projeto Comandos para modulo ESP8266 NodeMCU por *Web Server* .. 135
Figura 86 – Estrutura de conexão dos componentes em MQTT 138
Figura 87 – Estrutura de tópicos para comunicação MQTT 140
Figura 88 -Comunicação com qualidade de serviço QoS 141
Figura 89 - Diagrama de comunicação por retenção de mensagem em MQTT 142

Figura 90 - Processo de instalação da ferramenta MQTTBox 144
Figura 91 – Configuração do cliente publicador e assinante na ferramenta MQTTBox 144
Figura 92 - Instalação do pacote de biblioteca pubsubclient-master 146
Figura 93 - Aplicação exemplo de comunicação entre módulos ESP8266 NodeMCU por MQTT 147
Figura 94 – Diagrama da aplicação do projeto de comando e monitoramento de variáveis de uma residência 154
Figura 95 – Princípio de funcionamento da troca de mensagens entre ESP8266 NodeMCU e *Telegram* 160
Figura 96 – Resultado esperado do projeto troca de mensagens entre ESP8266 NodeMCU e *Telegram* 160
Figura 97 - Processo de criação do subusuário dedicado à comunicação. 162
Figura 98 – Processo de obtenção do numero de ID de usuário específico no *Telegram* 162
Figura 99 - Inserção do pacote de bibliotecas *Universal Telegram Bot* na IDE Arduino. 163
Figura 100 – Estrutura de interação entre ESP8266 NodeMCU e *Blynk* 167
Figura 101 – Proceso de criação de uma conta no *Blynk App* 168
Figura 102 – Processo de criação de um novo projeto no *Blynk App* 168
Figura 103 - Processo de inserção de objetos *widget* na área de trabalho do *Blynk* 169
Figura 104- Processo de instalação do pacote de bibliotecas *Blynk* na IDE Arduino 170
Figura 105- Conexão exemplo do *Blynk App* com o módulo ESP8266 NodeMCU 171
Figura 106 - Exemplo de interação *Blynk* com ESP8266 NodeMCU com valor puxado pelo *Blynk App* 172
Figura 107 - Exemplo de interação *Blynk* com ESP8266 NodeMCU com valor empurrado para o *Blynk App* 173
Figura 108 - Edição do aplicativo do projeto de interação no *Blynk* 175
Figura 109 - Projeto de interação entre ESP8266 NodeMCU e *Blynk* 176
Figura 110 – Configuração dos componentes do aplicativo criado no *Blynk App* para o projeto de monitoramento residencial 179
Figura 111 – Diagrama e estrutura do projeto de monitoramento residencial 179
Figura 112 - Estrutura do portal Adafruit 181
Figura 113 - Processo para criar uma conta no portal Adafruit 182
Figura 114 - Área de trabalho do portal Adafruit 183
Figura 115 - Processo para criar os *Feeds*, armazenadores de dados em repositório 184
Figura 116 - Processo para criar o painel visual *dashboard* 184
Figura 117 - Processo de inserção de componentes *widgets* no painel de visualização 185
Figura 118 - Processo para organizar os componentes no painel de trabalho 186
Figura 119 - Processo para gerar a chave de seguranca da aplicação 187

Figura 120 - Exemplo de interação entre Adafruit e ESP8266 NodeMCU 188
Figura 121 - Inserção do pacote de biblioteca Adafruit MQTT na IDE Arduino 188
Figura 122 - Projeto de monitoramento residencial com Adafruit e ESP8266 NodeMCU 193
Figura 123 - Exemplo de interação entre módulo ESP8266 NodeMCU e MQTT Dash 197
Figura 124 - Processo de Instalação e criação de um painel no MQTT Dash 198
Figura 125 - Processo de inserção dos componentes *widgets* no painel 199
Figura 126 - Edição de um objeto *widget* ou mudança de leiaute no MQTT Dash 200
Figura 127 - Inserção do pacote de biblioteca “PubSubClient by Nick OLeary” 200
Figura 128 – Estrutura e diagrama do projeto monitoramento de residência com ESP8266 NodeMCU e MQTT Dash 203

ÍNDICE DE QUADROS

Quadro 1 – Características placa ESP8266 NodeMCU 16
Quadro 2 – Pinagem completa do módulo ESP8266 NodeMCU 17
Quadro 3 – Portas de entrada e saída da placa ESP8266 NodeMCU 18
Quadro 4 – Constantes do sistema na IDE Arduino 27
Quadro 5 – Tipos de variáveis na IDE Arduino 28
Quadro 6 – Principais operadores aritméticos 29
Quadro 7 – Principais operações *bitwise* 29
Quadro 8 – Principais operadores lógicos 30
Quadro 9 – Principais operadores relacionais 30
Quadro 10 – Principais funções nativas do IDE Arduino 31
Quadro 11 – Relação de pinos de saídas digitais que o ESP8266 NodeMCU disponibiliza 40
Quadro 12 – Relação de pinos de entradas digitais que o ESP8266 NodeMCU disponibiliza 45
Quadro 13 – Relação de pinos de saídas PWM que o ESP8266 NodeMCU disponibiliza 48
Quadro 14 – Pino de entrada analógica que o ESP8266 NodeMCU disponibiliza 51
Quadro 15 – Principais funções da biblioteca nativa serial 54
Quadro 16 – Principais funções da biblioteca LiquidCrystal_I2C 61
Quadro 17 – Acionamento dos motores em função dos sinais nos pinos de entrada da Ponte H 81
Quadro 18 – Especificações de servo motores SG90 e SG5010 88
Quadro 19 – Tipos de mensagens no protocolo MQTT 139
Quadro 20 – Relação de agentes de recebimento e entrega disponíveis para teste de forma gratuíta 143

SUMÁRIO

1 INTRODUÇÃO AO ESP8266 NODEMCU 15

1.1 CARACTERÍSTICAS DO MÓDULO ESP8266 NODEMCU 15

1.2 CONEXÃO DO ESP8266 NODEMCU COM IDE ARDUINO 19

1.2.1 Adaptador Serial-USB CP210x 19

1.2.2 Adaptador Serial-USB CH340G 20

1.3 INSTALAÇÃO E CONFIGURAÇÃO DA IDE DO ARDUINO 21

1.4 ESTRUTURA DE PROGRAMAÇÃO C++ ARDUINO 25

1.4.1 Estrutura de programa 26

1.4.2 Diretivas 27

1.4.3 Constantes e Variáveis 27

1.4.4 Operações lógicas e aritméticas 29

1.4.5 Funções 31

1.4.6 Estruturas lógicas 32

1.4.7 Bibliotecas 35

1.5 PROTOCOLOS DE COMUNICAÇÃO 36

1.5.1 Protocolo UART 36

1.5.2 Protocolo I2C 38

1.5.3 Protocolo SPI 39

2 ENTRADAS E SAÍDAS 40

2.1 SAÍDAS DIGITAIS 40

2.2 ENTRADAS DIGITAIS 45

2.3 SAÍDAS ANALÓGICAS - PWM 48

2.4 ENTRADA ANALÓGICA 51

3 MONITORAMENTO 53

3.1 MONITOR SERIAL 53

3.2 DISPLAY MATRICIAL LCD COM COMUNICAÇÃO I2C 57

4 PERIFÉRICOS 65

4.1 SENSORES 65
4.1.1 Sensor indutivo 65
4.1.2 Sensor capacitivo 66
4.1.3 Sensor óptico 67
4.1.4 Sensor por ultrassom 69
4.2 RELÉ ELETROMAGNÉTICO 75
4.3 MOTOR CC COM PONTE H 80
4.4 SERVO MOTOR 87
4.6 SENSOR RFID 93
4.7 SENSOR DE UMIDADE e temperatura 99
5 INTERNET DAS COISAS 102
5.1 CONEXÃO *WIFI* do Módulo esp8266 nodemcu 104
5.2 CONEXÃO do módulo esp8266 nodemcu COM INTERNET 115
5.3 APLICAÇÃO CLIENTE- SERVIDOR (*WEB SERVER* HTTP) 127
5.4 APLICAÇÃO *PUBLISH–SUBSCRIBE* (MQTT) 138
5.4.1 Emulador MQTTBox 143
5.4.2 Aplicação com módulo ESP8266 NodeMCU com protocolo MQTT 145
6 INTERAÇÃO COM APLICATIVOS 159
6.1 INTERAÇÃO COM APLICATIVO TELEGRAM 159
6.2 Interação com APLICATIVO BLYNK 167
6.3 Interação com Adafruit IO via MQTT 181
6.4 Interação com APLICATIVO MQTT DASH 196
REFERÊNCIAS 207
SOBRE O AUTOR 210

1 INTRODUÇÃO AO ESP8266 NODEMCU

Também conhecido como NodeMCU Devkit 1.0, a placa de desenvolvimento ESP8266 NodeMCU[1] foi criada para servir como um dispositivo programável, com entradas e saídas, que permite a programação na linguagem LUA[2] (LUA, 2021), com módulo de *wifi* incorporado, permitindo acesso a rede internet, elevando o dispositivo para Internet das Coisas – IoT.

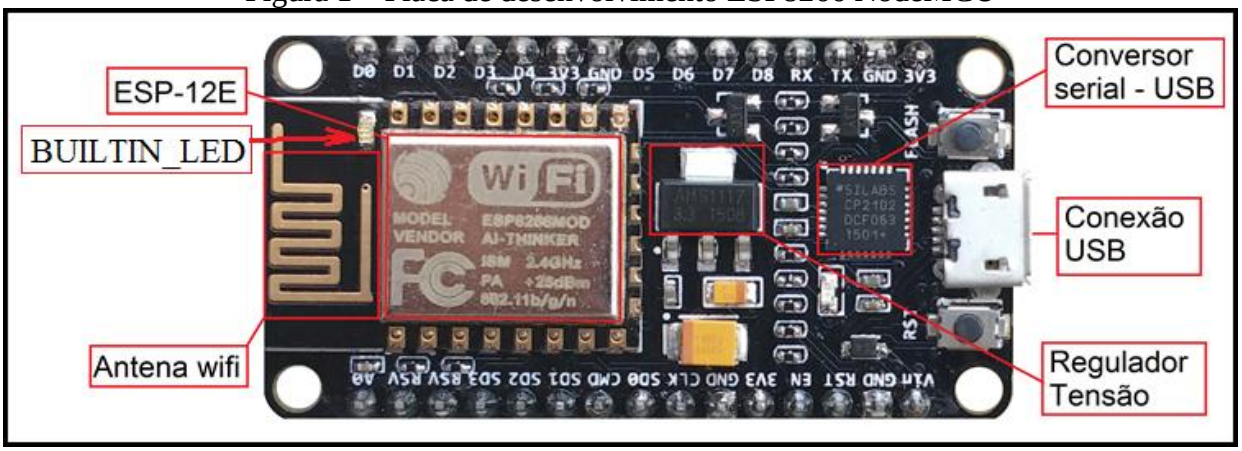
Apesar da intenção inicial de utilização da linguagem de programação LUA, o módulo permite que seja utilizada a interface de desenvolvimento do Arduino (Arduino IDE), o que facilita muito a programadores de Arduino que estão iniciando no módulo ESP8266 NodeMCU.

1.1 CARACTERÍSTICAS DO MÓDULO ESP8266 NODEMCU

O módulo ESP8266 NodeMCU consiste de um módulo na forma de placa de circuito de impresso, onde são embarcados um conversor serial-USB e o módulo ESP12E, com um *firmware* específico para prover o acesso aos periféricos disponibilizados. Assim, as características do módulo ESP8266 NodeMCU passam a ser as mesmas do módulo ESP12E, nele embarcado.

A Figura 1 apresenta a placa de desenvolvimento, com seus componentes periféricos.

Figura 1 – Placa de desenvolvimento ESP8266 NodeMCU



Fonte: Adaptado de Arduining(2021)

[1] Datasheet em https://cdn-shop.adafruit.com/datasheets/ESP8266_Specifications_English.pdf
[2] Projeto brasileiro de 1993, desenvolvido na PUC-RJ, a linguagem LUA foi criada para ser uma linguagem fácil de se embutir em outros ambientes. Fonte: http://www.lua.org/

A alimentação do módulo é realizada pela própria conexão USB, com uma tensão de 5V ou então através do pino Vin, para uma conexão externa. Internamente a tensão de trabalho do módulo é de 3,3V. Isto é provido pelo regulador de tensão AMS1117 (SYSTEMS, 2021). O regulador de tensão fornece 1A, considerando que o consumo do módulo é de aproximadamente 200mA, sobrando 800mA para alimentar periféricos, sendo que se recomenda que nunca ultrapasse os 500mA de consumo por periféricos conectados ao módulo.

O conversor serial-USB é o responsável pela conexão serial com o computador, a fim de que se realize a programação do módulo ou troca de mensagens/comandos através do monitor serial no computador. Esse conversor necessita de um *driver* específico[3], a ser instalado no computador, para reconhecer a conexão USB como uma porta serial do tipo COM.

Já o bloco ESP12E[4], inserido no módulo, é o responsável pela comunicação *wifi*, além de executar uma lógica programável, uma vez que permite embarcar um *firmware* executor de comandos programados pelo usuário. Disponibiliza, através de seus pinos GPIOs (*General Purpose Input/Output*), entradas e saídas digitais e analógicas, além de pinos para acesso a periféricos especiais, como é o caso de barramentos SPI (S*erial Peripheral Interface*) que, através dos pinos CS, SCLK, MOSI e MISO, promovem a comunicação serial entre o módulo e seus periféricos, e também ao barramento I2C, através dos pinos SDA e SCL para a comunicação mestre-escravo.

O Quadro 1 apresenta as características do módulo e seus parâmetros.

Quadro 1 – Características placa ESP8266 NodeMCU

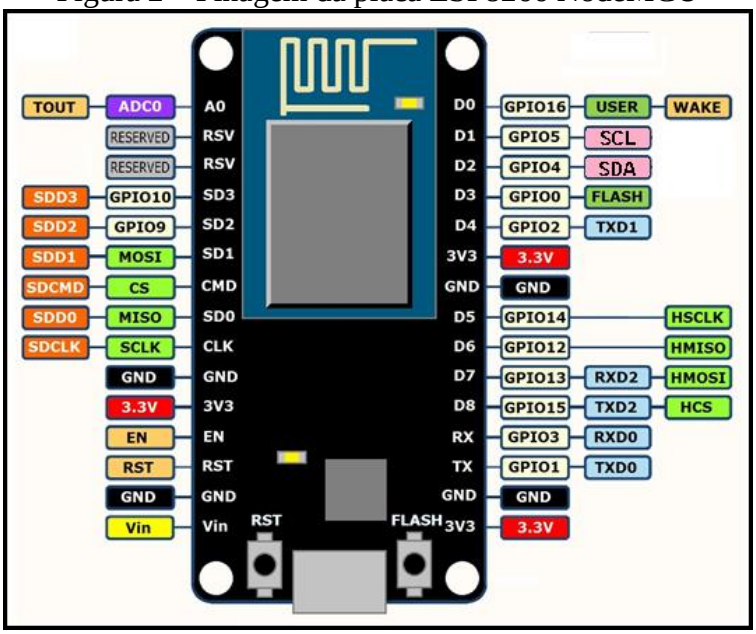
Parâmetros	**ESP8266 NodeMCU**
Processador	32 bits RISC Tensilica Xtensa L106
Frequência	80Mhz a 160Mhz
Memória RAM	64kB para instruções, 96kB para dados
Memória Flash	4MB
Wifi	IEEE 802.11 b/g/n wifi
Canal Analógico	1 porta de entrada, 10 bits
Portas digitais	9 portas (tensão 3,3V, corrente máxima 15mA)

Fonte: Autor (2021)

Os pinos de acesso ao módulo ESP8266 NodeMCU são apresentados na Figura 2.

[3] No exemplo da figura o conversor é um CP2102, cujo *drive* pode ser obtido em https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers . Para cada conversor deve ser buscado o *driver* específico.
[4] *Data sheet* do ESP12E pode ser visto em https://www.alldatasheet.com/datasheet-pdf/pdf/1179099/ETC2/ESP12E.html

Figura 2 – Pinagem da placa ESP8266 NodeMCU



Fonte: Adaptado de Arduining (2021)

O Quadro 2 apresenta a pinagem completa do ESP8266 NodeMCU, com sua descrição e função, na ordem física de seus pinos.

Quadro 2 – Pinagem completa do módulo ESP8266 NodeMCU

Pino	Descrição	Pino	Descrição
A0	Entrada analógica de 10 bits (0 a 1.023).	D0	Entrada ou saída digital (0V ou 3,3V) – GPIO-16[5]
RSV	Reservado internamente, não use.	D1	Entrada ou saída digital (0V ou 3,3V) – GPIO-5[5] Comunicação I2C, sinal SCL (*Serial Clock*)
RSV	Reservado internamente, não use.	D2	Entrada digital (0V ou 3,3V), saída digital (0V ou 3,3V) ou saída PWM – GPIO-4[5] Comunicação I2C, sinal SDA (*Serial Data*)
SD3	Comunicação com SD Card (SDD3) – GPIO-10	D3	Saída digital (0V ou 3,3V) – GPIO-0[5]
SD2	Comunicação com SD Card (SDD2) – GPIO-9	D4	Saída digital (0V ou 3,3V), ligado ao LED Builtin montado no módulo, com lógica invertida – GPIO-2[5]
SD1	Interface SPI (MOSI) – GPIO-8	3V3	Tensão de 3,3V, para uso em periférico
CMD	Interface SPI (CS) – GPIO-11	GND	Tensão de referência de 0V, para uso em periférico
SD0	Interface SPI (MISO) – GPIO-7	D5	Entrada digital, saída digital ou saída PWM – GPIO-14[5]
CLK	Interface SPI (SCLK) – GPIO-6	D6	Entrada digital, saída digital ou saída PWM – GPIO-12[5]
GND	Tensão de referência de 0V, para uso em periférico	D7	Entrada ou saída digital (0V ou 3,3V) – GPIO-13[5]
3V3	Tensão de 3,3V, para uso em periférico	D8	Saída digital ou saída PWM – GPIO-15[5]
EN	Habilita o módulo a operar	RX	Interface Serial RxD, usado para carga do programa e disponível para uso no programa – GPIO-3

Continua...

[5] Função de entrada ou saída definida pela função *pinMode*(pino, função), a ser executada dentro do *setup*().

RST	Reset no módulo, faz placa reinicializar (boot)	TX	Interface Serial TxD, usado para carga do programa e disponível para uso no programa – GPIO-1
GND	Tensão de referência de 0V, para uso em periférico	GND	Tensão de referência de 0V, para uso em periférico
Vin	Tensão de entrada no módulo, 5V a 9V.	3V3	Tensão de 3,3V, para uso em periférico

Fonte: Murta (2018a)

Os pinos de entradas e saídas do módulo possuem uma nomenclatura da placa ESP8266 NodeMCU, mas que no IDE do Arduino deve ser utilizada a numeração dos GPIO do ESP12E, embutido no módulo.

O Quadro 3 apresenta a relação das portas de entrada e saída com os GPIOs e suas características.

Quadro 3 – Portas de entrada e saída da placa ESP8266 NodeMCU

Pino	**GPIO**	**Saída digital**	**Entrada digital**	**Entrada analógica**	**Saída PWM**	**Função Especial**
A0	-	-	-	Sim	-	ADC10bits
D0	16	Sim	Sim	-	-	-
D1	5	Sim	Sim	-	-	-
D2	4	Sim	Sim	-	Sim	-
D3	0	Sim	-	-	-	-
D4	2	Sim	-	-	-	BUILTIN_LED
D5	14	Sim	Sim	-	Sim	-
D6	12	Sim	Sim	-	Sim	-
D7	13	Sim	Sim	-	-	-
D8	15	Sim	-	-	Sim	-

Fonte: Autor (2021)

Nesta tabela, se destaca a porta de entrada analógica, pino A0, que possui um conversor analógico digital interno de 10 bits, sendo sua tensão de entrada de 0V a 3,3V. Desta forma, há uma palavra digital convertida que varia de 0 a 1023.

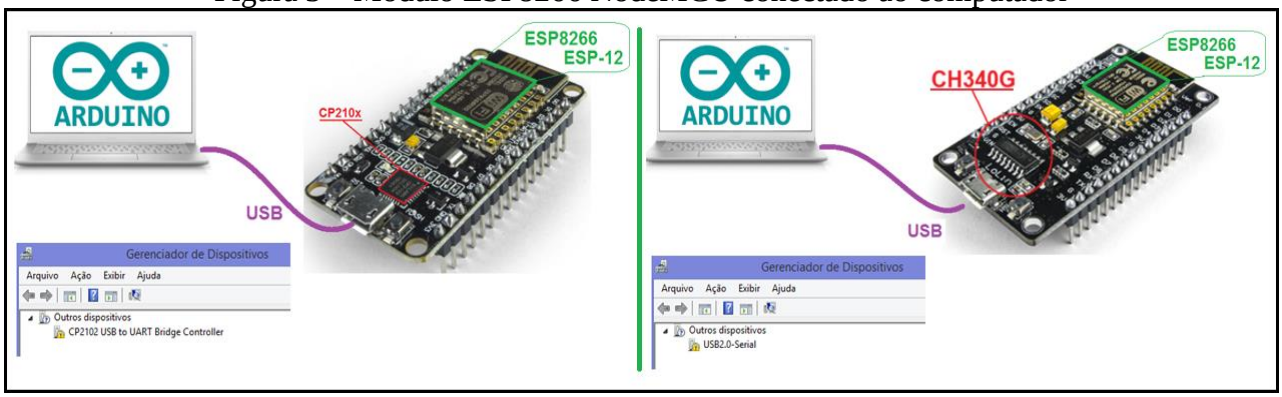
Outro destaque são as 4 portas de saída PWM (*Pulse Width Modulation*), modulação por largura de pulso, sendo eles D2, D5, D6 e D8. Essas saídas, apresentam um sinal pulsado de 0V e 3,3V, com um ciclo de trabalho (razão entre o tempo em nível alto pelo tempo de ciclo) determinado por um valor entre 0 e 1.023.

Também se ressalta a porta D4 (GPIO-2) que, além da saída digital, possui um LED (*Light Emitting Diode*) inserido no próprio módulo, chamado de BUILTIN_LED, porém com lógica inversa. Um sinal alto (HIGH) provoca o desligamento do BUILTIN_LED, já um sinal baixo (LOW) provoca a ligação do BUILTIN_LED.

1.2 CONEXÃO DO ESP8266 NODEMCU COM IDE ARDUINO

Para proceder à programação do módulo, faz-se necessária a conexão a um computador via cabo USB e a utilização da IDE do Arduino. Ao conectar o cabo USB no módulo e na entrada USB do computador, o NodeMCU já será reconhecido pelo computador como outro dispositivo. Isto pode ser percebido ao monitorar o **Gerenciador de Dispositivos** do computador. A conexão é reconhecida, mas não há o *driver* para identificar quem é o dispositivo que quer fazer a conexão. A Figura 3 apresenta o módulo sendo conectado ao computador, sendo sua conexão reconhecida no gerenciador de dispositivos.

Figura 3 – Módulo ESP8266 NodeMCU conectado ao computador



Fonte: Autor (2021)

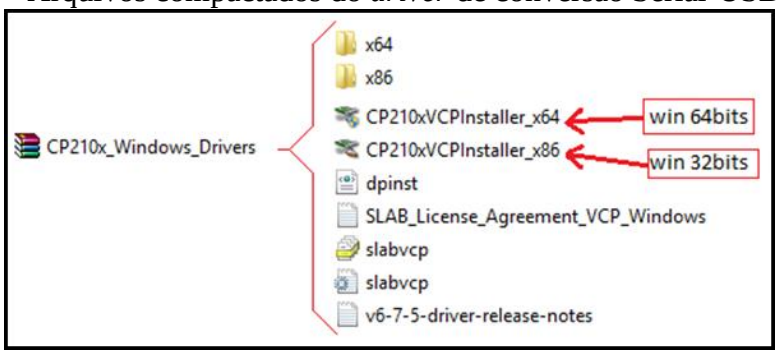
Para instalação do devido *drive*r de comunicação é necessário identificar qual conversor está sendo utilizado no módulo ESP8266 NodeMCU. O mais comum é ser um conversor CP210x da empresa Silabs ou um CH340G da empresa WCH Jiangsu Haoheng.

1.2.1 Adaptador Serial-USB CP210x

Neste caso o primeiro passo é proceder ao *download* do *drive* no endereço[6]: https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip. O arquivo baixado é uma pasta compactada. Descompacte a pasta e execute o instalador, segundo a opção de sistema operacional *Windows* utilizado, de 32 ou de 64 bits. A Figura 4 apresenta esses arquivos.

[6] *Driver* disponibilizado no portal https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip.

Figura 4 – Arquivos compactados do *driver* de conversão Serial-USB CP210x



Fonte: Autor (2021)

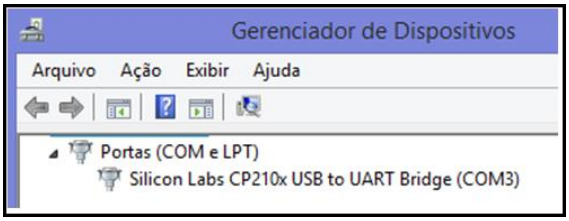
Uma vez executado o arquivo de instalação do *driver*, deve-se seguir os passos de instalação, conforme apresentados na Figura 5.

Figura 5 – Passos para instalação do *driver* conversor Serial-USB CP210x

Fonte: Autor (2021)

Após estes passos, o *driver* é instalado no computador, surgindo como uma porta dentro do Gerenciador de Dispositivos do Windows, apontando o número da porta na qual foi instalado. A Figura 6 apresenta o *driver* instalado na porta COM3, como exemplo.

Figura 6 – Driver CP210x instalado na COM3 do Gerenciador de Dispositivos do Windows



Fonte: Autor (2021)

1.2.2 Adaptador Serial-USB CH340G

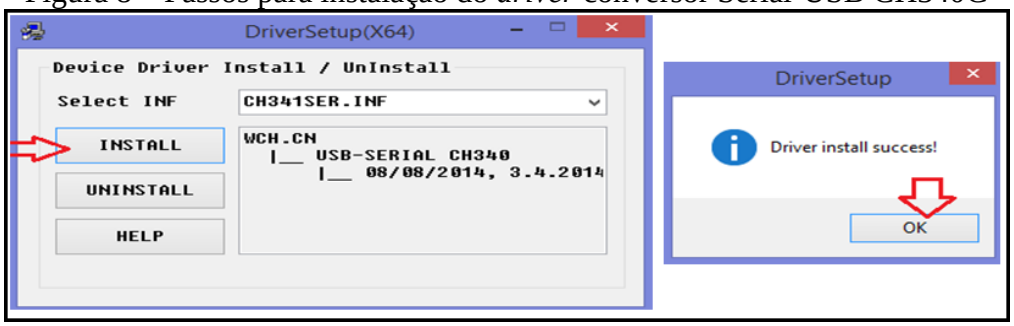
Neste caso o primeiro passo é proceder ao download do drive no endereço[7]: http://www.wch-ic.com/downloads/CH341SER_ZIP.html. O arquivo baixado é uma pasta compactada. Descompacte a pasta e execute o instalador. A Figura 7 apresenta esse arquivo.

Figura 7 – Arquivos compactados do *driver* de conversão Serial-USB CH340G

Fonte: Autor(2021)

Uma vez executado o arquivo de instalação do *driver*, deve-se seguir os passos de instalação, conforme apresentados na Figura 8.

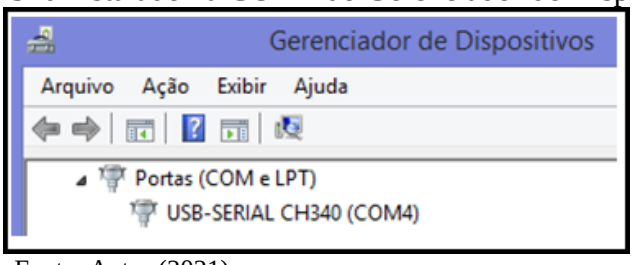
Figura 8 – Passos para instalação do *driver* conversor Serial-USB CH340G



Fonte: Autor (2021)

Procedido estes passos, o *driver* é instalado no computador, surgindo como uma porta dentro do Gerenciador de Dispositivos do Windows, apontando o número da porta na qual foi instalado. A Figura 9 apresenta o *driver* instalado na porta COM4, como exemplo.

Figura 9 – *Driver* CH340 instalado na COM4 do Gerenciador de Dispositivos do Windows



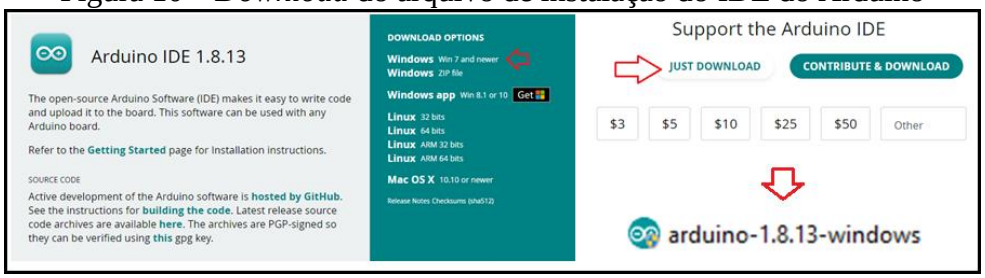
Fonte: Autor (2021)

1.3 INSTALAÇÃO E CONFIGURAÇÃO DA IDE DO ARDUINO

[7] *Driver* disponibilizado no portal http://www.wch-ic.com/downloads/CH341SER_ZIP.html.

A interface de desenvolvimento do Arduino (IDE – *Integrated Development Environment*) está disponível no endereço https://www.arduino.cc/en/software. É necessário baixar o arquivo instalador, seguindo os passos apontados na Figura 10.

Figura 10 – *Download* do arquivo de instalação do IDE do Arduino

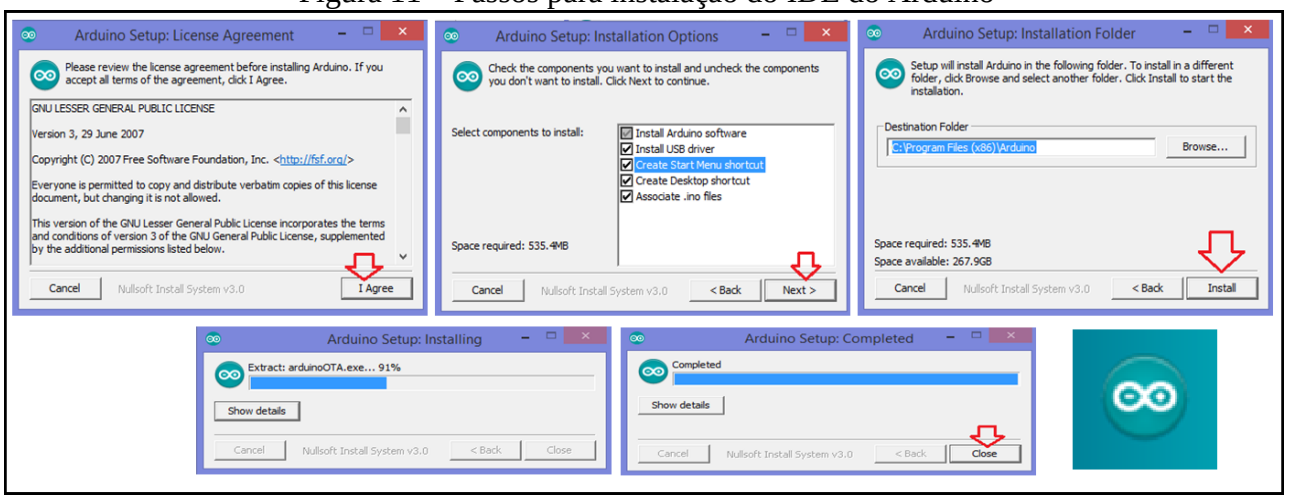


Fonte: Autor (2021)

Após, se executa o arquivo instalador do IDE do Arduino, concordando com a licença de uso (*License Agreement*), selecionando as opções de instalação (*Installation Options*) e definindo a pasta onde será instalado o IDE (*Installation Folder*). Inicialmente os arquivos serão transferidos para pasta indicada e após isso a instalação será concretizada. Então, o IDE do Arduino já pode ser executado.

Esses passos de instalação encontram-se na Figura 11.

Figura 11 – Passos para instalação do IDE do Arduino



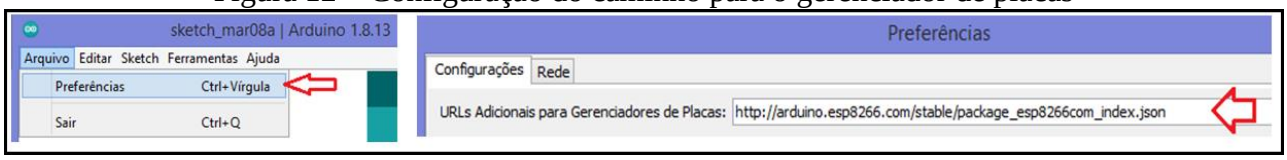
Fonte: Autor (2021)

O IDE do Arduino é instalado para programar placas Arduino. Cabe então configurar o IDE para reconhecer o módulo ESP8266 NodeMCU, procedendo assim a devida interpretação dos comandos, bibliotecas e programação do módulo.

Em executando o IDE do Arduino e na aba “Arquivo” e “Preferências” surgirá uma janela com a aba “configurações”. Nela haverá uma caixa de texto solicitando URLs Adicionais para Gerenciador de Placas:”. Deve-se inserir nesta caixa de texto o endereço

http://arduino.esp8266.com/stable/package_esp8266com_index.json. A Figura 12 aponta essa configuração para definir o caminho ao gerenciador de placas.

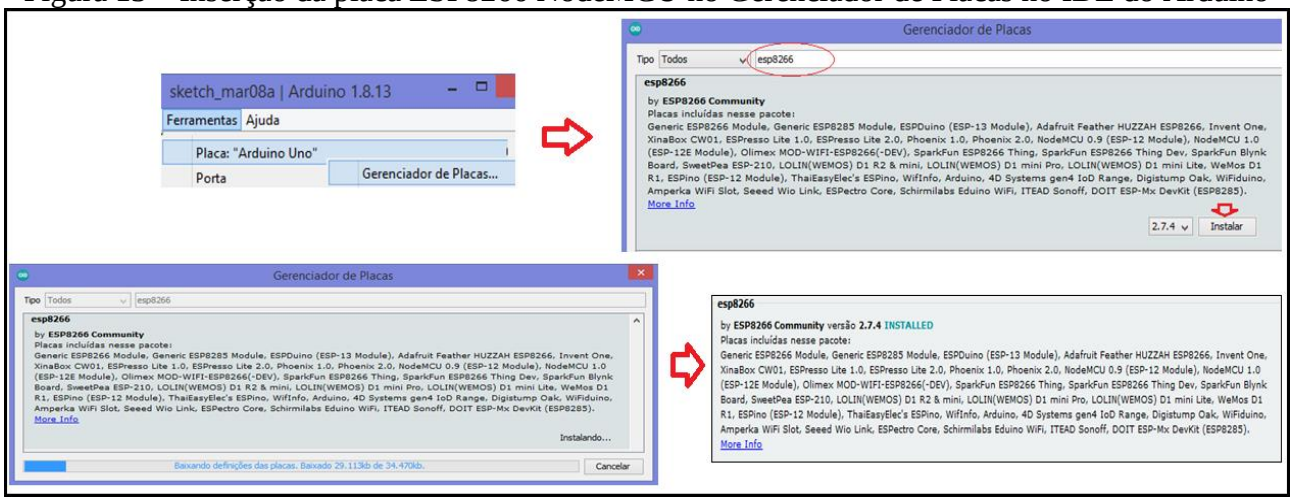
Figura 12 – Configuração do caminho para o gerenciador de placas



Fonte: Autor (2021)

O próximo passo é adicionar o módulo ESP8266 NodeMCU ao gerenciador de placas. Para isto, basta clicar em "Ferramentas", "Placa:", "Gerenciador de Placas...". Na caixa de pesquisa digita-se ESP8266. Será apresentado o pacote "ESP8266 *Community*". Deve-se selecionar ele e clicar em instalar. Os arquivos serão transferidos e o pacote instalado, conforme apontam os passos da Figura 13.

Figura 13 – Inserção da placa ESP8266 NodeMCU no Gerenciador de Placas no IDE do Arduino

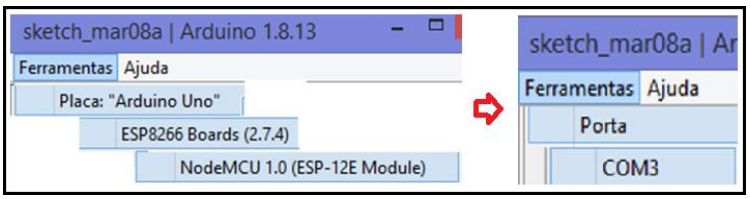


Fonte: Autor (2021)

Para poder desenvolver algum programa no IDE do Arduino, é necessário carregar a placa ESP8266 NodeMCU e definir a porta de comunicação.

Na barra de ferramentas, deve-se clicar em "Ferramentas", "Placa:", "ESP8266 Boards (2.7.4)" e "NodeMCU 1.0 (ESP-12E Module)". Na sequência, é necessário selecionar a porta onde o módulo está conectado, procedendo um clique na barra de ferramentas em "Ferramentas", "Porta" e "COM3". Pode ser outra porta, COM3 é só um exemplo. Isto dependerá de quais outras portas já estejam instaladas no microcomputador em uso. A Figura 14 apresenta o processo de seleção da placa e de seleção de porta.

Figura 14 – Seleção da placa ESP8266 NodeMCU e seleção de porta de comunicação na IDE do Arduino

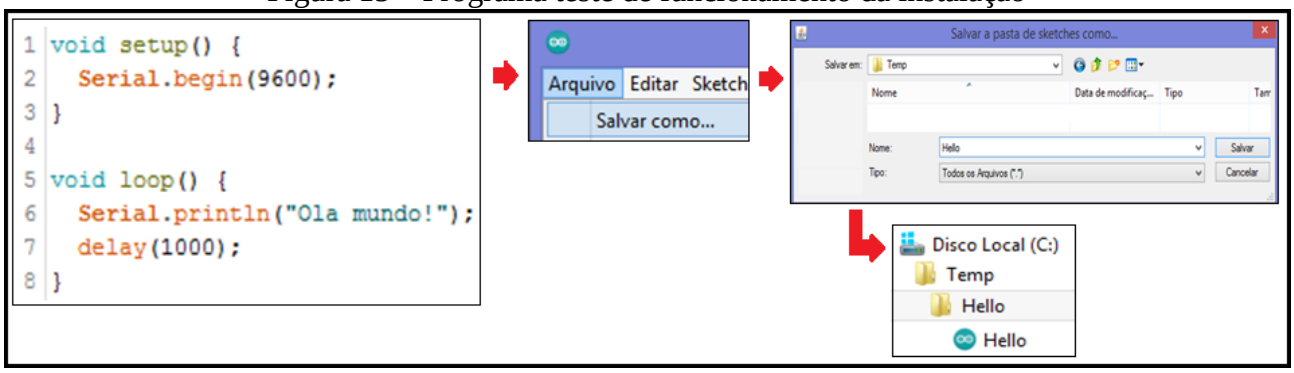


Fonte: Autor (2021)

Para testar seu funcionamento, é proposto um programa que comunica o módulo ESP8266 NodeMCU com o monitor serial do computador, emitindo uma mensagem a cada segundo, definida como “Ola Mundo!”.

Para isto, digite o trecho de código da Figura 15, salvando o arquivo, como indicado.

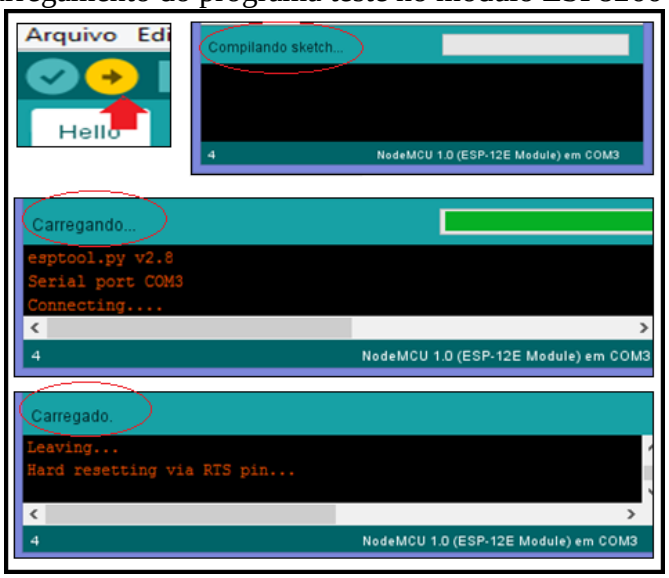
Figura 15 – Programa teste de funcionamento da instalação



Fonte: Autor (2021)

Então, basta enviar o programa para o módulo ESP8266 NodeMCU, clicando no botão “enviar” apresentado na Figura 16. O programa será compilado (onde são verificados erros de sintaxe e é realizada a conversão para o código de máquina) e transferido para o módulo. Ao final do carregamento do programa, ele já será executado pelo módulo ESP8266 NodeMCU.

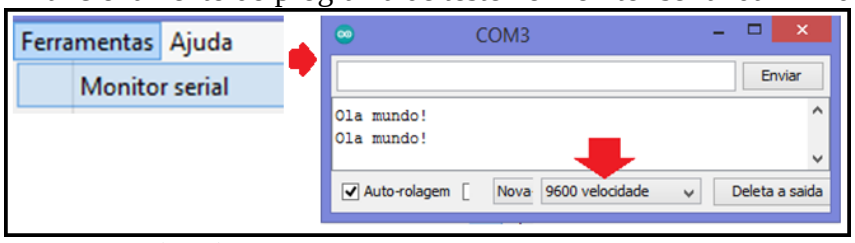
Figura 16 – Carregamento do programa teste no módulo ESP8266 NodeMCU



Fonte: Autor (2021)

Para testar o programa, é preciso abrir o monitor serial no IDE do Arduino. Na barra de ferramentas, deve-se clicar em “Ferramentas” e “Monitor Serial”, ajustando a velocidade para 9.600 bit/s (bits por segundo). A cada segundo, será apresentada a mensagem “Ola mundo!” no monitor serial. A Figura 17 apresenta o funcionamento do programa no monitor serial da IDE do Arduino.

Figura 17 – Funcionamento do programa de teste no monitor serial da IDE do Arduino



Fonte: Autor (2021)

Com esse teste, se conclui que o módulo está executando o programa e a comunicação está devidamente configurada, bem como os *drivers* para conversão Serial-USB.

1.4 ESTRUTURA DE PROGRAMAÇÃO C++ ARDUINO

O módulo ESP8266 NodeMCU foi projetado para ser programado pela linguagem de programação LUA. Porém, há bibliotecas que permitem que esse módulo seja programado em linguagem C++ pelo IDE do Arduino, e com isto se vale da estrutura do programa, diretivas, constantes, variáveis, funções e estruturas lógicas já utilizadas nesta interface (ARDUINO, 2021).

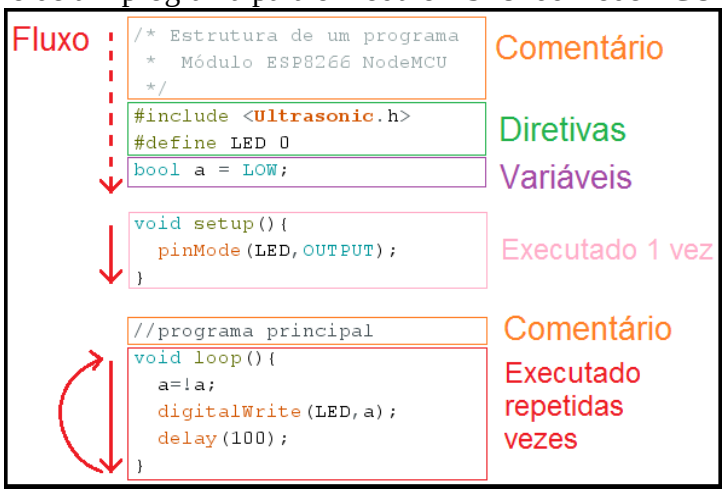
1.4.1 Estrutura de programa

O fluxo de execução de um programa no modelo do IDE do Arduino segue um comportamento, segundo a posição que se encontram as instruções e o tipo de instrução que é executado.

Ao energizar o módulo ESP8266 NodeMCU, a primeira função a ser executada é a função *setup()*. Essa função é nativa do IDE e indica que, dentro dela estarão as ações e configurações desejadas para ocorrer no início da execução de um programa. Após a execução desta função, o fluxo de programa passa a executar a função *loop()*, onde deverão estar as funções e lógicas desejadas para ocorrer a cada ciclo de programa, que se repetirá continuamente até que o módulo seja desenergizado.

Precedendo as funções *setup()* e *loop()*, há os elementos que auxiliam na configuração do programa, como é o caso das diretivas e variáveis. Elas representam definições iniciais, que hajem como recurso ao programa. A Figura 18 apresenta o fluxo de execução do programa.

Figura 18 – Fluxo de um programa para o módulo ESP8266 NodeMCU na IDE Arduino



Fonte: Autor (2021)

Cabe salientar alguns detalhes de sintaxe na estrutura de um programa:

a) Comentários: Textos inseridos na forma de comentários, que não devem ser interpretados como um programa, apenas como um auxílio ao leitor do programa. O comentário deve ser inserido precedidos de barras duplas (exemplo: //texto na mesma linha) ou com barra mais asterisco no início da frase e asterisco mais barra no final da frase (exemplo: /* texto mais de uma linha */);

b) Ponto e vírgula: Ao final de cada instrução deve haver a inserção de ponto e vírgula (;), indicando seu final.

1.4.2 Diretivas

São comandos que indicam ao programa, ao ser compilado[8], definições e outras estruturas que devem ser adicionadas a ele. Um exemplo deste último caso são as bibliotecas, arquivos com funções e parâmetros a serem utilizados pelo programa. Isto permite a utilização de outros códigos, com outros propósitos, em programas distintos. Duas diretivas se destacam aqui:

a) #include: inclui uma biblioteca ou outros arquivos que estão inseridos na IDE do Arduino, para fazer parte deste programa (sintaxe: #include <nome_biblioteca.h>) - **#include <Ultrasonic.h>**;

b) #define: permite dar um nome para um parâmetro, de forma que no programa não seja mais visto como um número, mas como um nome sugestivo (sintaxe: #define Nome Parâmetro) - **#define LED 0**.

1.4.3 Constantes e Variáveis

As constantes do sistema são palavras-chave já definidas no IDE Arduino, com uma correspondência numérica, lógica ou de interpretação. Estas constantes facilitam na padronização de programas, além de trazer nomes mais sugestivos para parâmetros ou valores numéricos.

O Quadro 4 apresenta as constantes de sistema mais usuais na programação Arduino.

Quadro 4 – Constantes do sistema na IDE Arduino

Constantes de sistemas	Valor interpretado
HIGH	Nível lógico alto (entre 2V e 3,3V)
LOW	Nível lógico baixo (entre 0V e 1V)
INPUT	Porta do módulo configurado como entrada
OUTPUT	Porta do módulo configurado como saída
LED_BUILTIN	2 - (GPIO-2 do led embarcado no módulo)
true	Algo diferente de zero (exemplo: 1, 2, -12, -1, ...)
false	Algo igual a zero.

Fonte: Autor (2021)

As variáveis são posições de memória nomeadas, em que se pode definir o tipo de valor ou informação nelas armazenadas. Segundo o tipo de variável, as posições de memória podem ocupar mais ou menos espaço na memória do microcontrolador. O Quadro 5 apresenta os principais tipos de variáveis e seu uso.

[8] Traduzido em uma linguagem para ser executada no microcontrolador ou em um interpretador do microcontrolador.

Quadro 5 – Tipos de variáveis na IDE Arduino

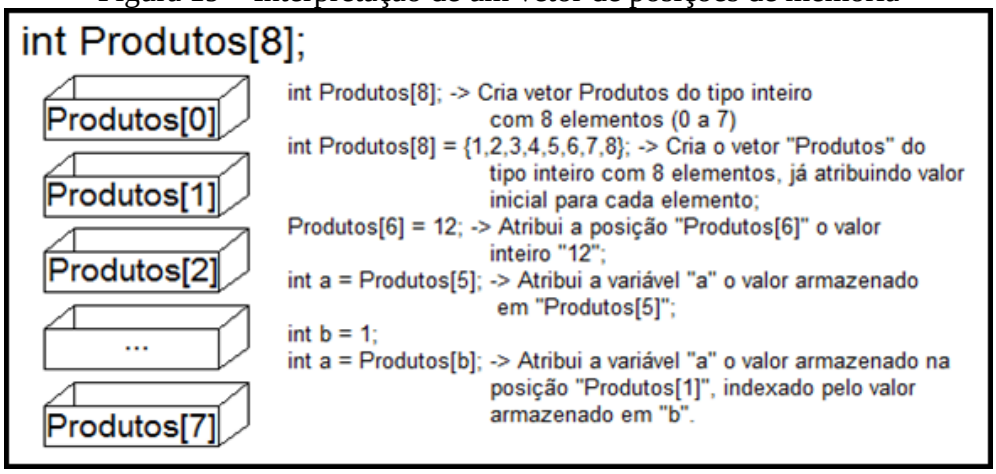
Tipos de variáveis	Valor	Definição	Exemplo
bool	false ou true	Valor binário de 1 bit	bool a = false;
byte	0 a 255	Valor numérico de 8 bits	byte a =124;
char	Caractere ou 0 a 255	Valor ASCII [9] do caractere	char a = 'B' ou char a = 66
string	frase	Conjunto de caracteres	string Nome = "Joao";
int	-32.768 a 32.767	Valor inteiro de 16 bits	int a=5623;
float	3,4028235 x 10^{38} a -3,4028235 x 10^{38}	Valor racional de 32 bits	float a = 3.14;
long	2147483648 a -2147483647	Valor inteiro de 32 bits	long a = 70000;
double	9223372036854775807 a -9223372036854775808	Valor inteiro de 64 bits	double a=65000;

Fonte: Autor (2021)

As variáveis são posições de memória para armazenamento de dados. Mas há aplicações em que se faz necessário armazenar um conjunto de dados em posições de memória (variável), sendo seu acesso indexado numericamente. Para estas situações existem os vetores.

Um vetor é um conjunto de posições de memória definido por uma variável, sendo diferenciados pelo seu número de indexação. A primeira posição de memória é a 0 (zero) e a última corresponde ao número de posições subtraída de 1. A Figura 19 mostra a intepretação de um vetor de posições de memória.

Figura 19 – Interpretação de um vetor de posições de memória



Fonte: Autor (2021)

[9] Tabela ASCII (*American Standard Code for Information Interchange* - Código Padrão Americano para o Intercâmbio de Informação) é uma relação de valores numéricos de 0 a 255 com caracteres formatados. Vide: https://pt.wikipedia.org/wiki/ASCII.

1.4.4 Operações lógicas e aritméticas

As operações aritméticas são funcionalidades que permitem proceder operações algébricas dentro do programa. Respeitam as regras matemáticas, mas deve-se ter um cuidado especial com a sintaxe, a forma com que são escritas, para estarem corretas e devidamente interpretadas. O Quadro 6 apresenta os principais operadores aritméticos.

Quadro 6 – Principais operadores aritméticos

Operador Aritmético	Descrição	Exemplo
+	Retorna o valor da soma entre dois valores	int a = 2 + 5; // a=7
-	Retorna o valor da subtração entre dois valores	float a = 2.3 – 5.0; // a = -2.7
*	Retorna o valor da multiplicação entre dois valores	int a = 2 * 5; // a=10

/	Se os valores são ponto flutuante, retorna o valor da divisão entre dois valores; Se os valores são inteiros, retorna a parte inteira da divisão entre dois valores, sendo o resto perdido.	float a = 7.0 / 2.0; // a=3.5
		int a = 7 / 2; // a=3

%	Retorna o valor do resto da divisão de dois valores inteiros	int a = 7 / 2; // a=1
++	Retorna à própria variável o resultado do incremento do valor em 1 unidade.	int a = 12; a++; int b = a; // valor de b=13
--	Retorna à própria variável o resultado do decremento do valor em 1 unidade.	int a = 12; a--; int b = a; // valor de b=11

Fonte: Autor (2021)

Deve se observar o tipo de variável, pois as operações requerem que seja entre valores de mesmo tipo. Por exemplo, analise o seguinte trecho de código.

```
float a = 5.5;
float b = 6.6;
int c = 0;
c = a * b;  // a variável 'c' armazena o valor 36 em vez do resultado esperado 36.3
```

Já os operadores *bitwise* (bit-a-bit) lógicos retornam o resultado de uma operação realizada bit a bit entre os valores fornecidos. Importante observar se o tipo de variável permite essa operação. O Quadro 7 apresenta as principais operações *bitwise*.

Quadro 7 – Principais operações *bitwise*

Operador Lógico	Descrição	Exemplo

&	Retorna o resultado da operação lógica “E” (AND) entre dois valores.	byte a = 12; // 00001100b
		byte b = 54; // 00110110b
		byte c = a & b; // 00000100b (4)

|	Retorna o resultado da operação lógica “OU” (OR) entre dois valores.	byte a = 12; // 00001100b
		byte b = 54; // 00110110b
		byte c = a | b; // 00111110b (62)

Continua...

<<	Retorna o resultado do deslocamento indicado dos bits de um valor binário para esquerda.	byte a = 12; // 00001100b byte b = a << 3; // 01100000b (96)
>>	Retorna o resultado do deslocamento indicado dos bits de um valor binário para direira.	byte a = 12; // 00001100b byte b = a >> 3; // 00000001b (1)

^	Retorna o resultado da operação lógica "Ou-exclusivo" (XOR) entre dois valores.	byte a = 12; // 00001100b
		byte b = 54; // 00110110b
		byte c = a & b; // 00111010b (58)

~	Retorna o resultado da operação lógica de "Inversão" (NOT) em um valor, invertendo o estado de cada bit.	byte a = 103; // 01100111b byte b = ~a; // 10011000b (-104)

Fonte: Autor (2021)

Já os operadores lógicos retornam um valor booleano (verdadeiro ou falso) face a uma operação lógica realizada entre dois valores. Seu dado de retorno será sempre um *bool*, valendo *false* ou *true*. O Quadro 8 apresenta os principais operadores lógicos.

Quadro 8 – Principais operadores lógicos

Operador Lógico	Descrição	Sintaxe
!	Realiza a inversão do resultado de um operador booleano	int a = 12; int b=10; if (!(a == b)) verdadeiro
&&	Realiza a operação "E" entre o resultado de dois operadores booleanos.	int a = 12; if ((a > 0) && (a < 10)) verdadeiro
||	Realiza a operação "OU" entre o resultado de dois operadores booleanos.	int a = 12; if ((a > 20) || (a < -10)) falso

Fonte: Autor (2021)

Por fim, os operadores relacionais comparam diferentes valores, retornando um valor booleano conforme a comparação, conforme apresentado no Quadro 9.

Quadro 9 – Principais operadores relacionais

Operador Relacional	Descrição	Sintaxe
!=	Retorna verdadeiro se os valores forem diferentes.	int a = 12; int b=10; if (a !=b) verdadeiro
<	Retorna verdadeiro se o primeiro valor for menor que o segundo valor.	int a = 12; int b=10; if (a < b) falso
<=	Retorna verdadeiro se o primeiro valor for menor ou igual ao o segundo valor.	int a = 12; int b=10; if (a <= b) falso
>	Retorna verdadeiro se o primeiro valor for maior que o segundo valor.	int a = 12; int b=10; if (a > b) verdadeiro
>=	Retorna verdadeiro se o primeiro valor for maior ou igual ao o segundo valor.	int a = 12; int b=10; if (a >= b) verdadeiro
==	Retorna verdadeiro se o primeiro valor for igual ao segundo valor.	int a = 12; int b=10; if (a == b) falso

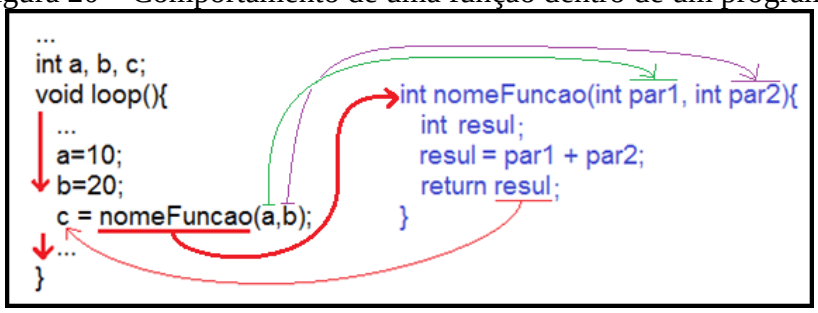
Fonte: Autor (2021)

Um cuidado especial se dá para as precedências, onde há mais de uma operação aritmética. A boa prática de programação recomenta utilizar parênteses para delimitar as precedências desejadas. Por exemplo, int a = 2 * (4 + 2), que corresponderá o ao resultado em "a" igual a 12.

1.4.5 Funções

As funções são trechos de código que realizam tarefas específicas, através da passagem de parâmetro e retorno de resultados. O uso de funções torna o programa melhor descrito, facilitando o entendimento e o reaproveitamento de trechos de código. A Figura 20 mostra o comportamento de uma função dentro de um programa.

Figura 20 – Comportamento de uma função dentro de um programa



Fonte: Autor (2021)

Além das funções que o programador pode criar para o programa específico, há as funções nativas do IDE Arduino (que não necessitam de inserção explícita de bibliotecas específicas) e as funções oriundas de bibliotecas que podem ser importadas para o programa em edição.

O Quadro 10 apresenta as principais funções nativas no IDE Arduino.

Quadro 10 – Principais funções nativas do IDE Arduino

Função	Descrição	Exemplo
abs(numero)	Retorna o valor do módulo do número: $(\sqrt{x^2})$.	int b = 2; int a = abs(b); // (a=2) int b = -2; int a = abs(b); //(a=2)
sq(numero)	Retorna o valor do quadrado de um número: x^2.	int b = 2; int a = sq(b); // (a=4)
sqrt(numero)	Retorna o valor da raiz quadrada um número: $\sqrt{x}$.	int b = 4; int a = sqrt(b); // (a=2)
random(min, max)	Retorna um valor tipo long aleatório entre um valor mínimo e máximo-1. random(min, max) ou random(max), neste último o min=0.	long a = random(100); int b = random(10, 90);

Continua...

map()	Retorna um valor de uma escala inicial, com um valor proporcional em uma escala final map(valor, minInicial, maxInicial, minFinal, maxFinal).	int a = 750;//range 0 a 1023
		// para range 0 a 100
		int b = map(a,0,1023,0,100);
		// b = 73

delay(tempo)	Faz o programa parar por um tempo em milissegundos, indicado na função.	delay(500); // para o programa por 500ms ou 0,5s
millis()	Retorna o número de milissegundos passados desde que a placa Arduino começou a executar o programa atual. Formato long.	long a = millis();
pinMode(GPIO, Tipo)	Configura o GPIO no Tipo de trabalho, no modelo pinMode(GPIO, Tipo). Não é necessário para os pinos de entrada analógica.	pinMode(2, OUTPUT); //GPIO-2 , pino D4, configurado para saída
digitalRead(GPIO)	Retorna o nível lógico do GPIO indicado.	digitalRead(2);// GPIO2, pino D4
digitalWrite(GPIO, Nível)	Atribui ao GPIO o nível lógico definido.	digitalWrite(2, HIGH);
analogRead(Pino)	Retorna o valor, de 0 a 1023, do pino de entrada analógica.	int a = analogRead(A0);
analogWrite(GPIO, valor)	Atribui ao GPIO o parâmetro *duty-cycle*, para sua ação PWM, com escala de 0 a 255.	analogWrite(15, 120); // GPIO-15, pino D8, com saída PWM em 47%.

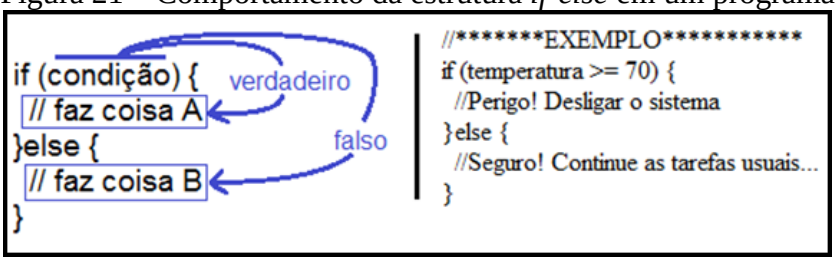
Fonte: Autor (2021)

1.4.6 Estruturas lógicas

As estruturas lógicas são as ações condicionais e tomadas de decisões que definem o fluxo de um programa. Elas implementam os algoritmos desejados, transformando a intenção em um fluxo lógico de execução de instruções. Dentre estas estruturas se destacam: estruturas condicionais e estruturas de repetição.

A execução condicional pode ser realizada pela instrução "*if*" seguida de uma condição, que deve ser verdadeira ou falsa. Caso a condição seja verdadeira, o trecho de código definido na sequência, entre chaves "{ }", será executado. Caso a condição seja falsa, o trecho de código definido após a instrução "*else*", entre chaves "{ }", será executado. A Figura 21 mostra o comportamento da estrutura "*if-else*", com um exemplo de aplicação.

Figura 21 – Comportamento da estrutura *if-else* em um programa



Fonte: Autor (2021)

Vale lembrar que pode haver mais de uma condição relacionada por operadores lógicos, como por exemplo:

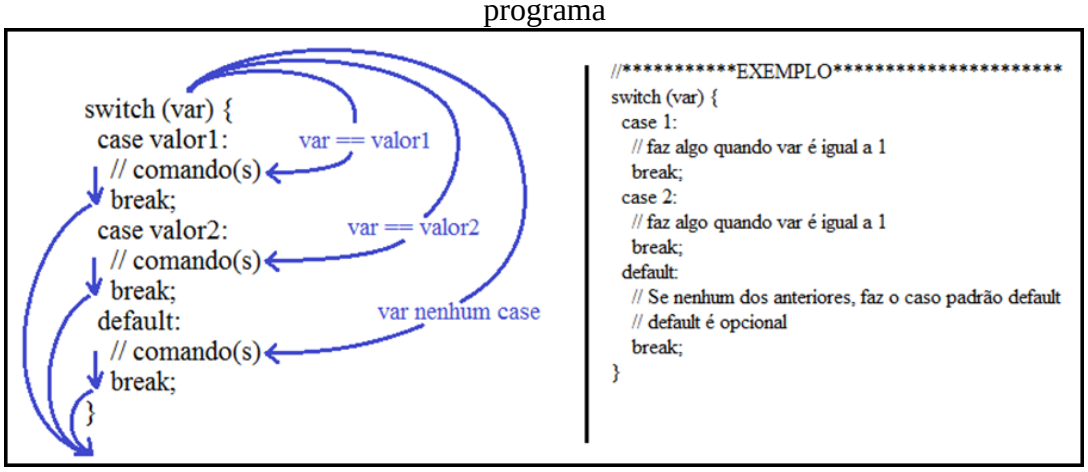
a) if((a>10) && (a<20)): se o valor de “a” for maior que 10 e também menor que 20, então a condição do teste condicional é verdadeira;

b) if((b == 2) || (a != 0)): se o valor de “b” for igual a 2 ou o valor de “a” for diferente de 0, então a condição do teste condicional é verdadeira.

Outra estrutura condicional baseada na comparação de uma variável com múltiplos valores é implementada pelas instruções “*switch*” e “*case*”. No início da estrutura a instrução “*switch*” carrega o valor da variável de condição. A partir de então, cada instrução “*case*” testa se o valor da variável de condição é igual ao valor definido. Caso verdadeiro, o trecho de código seguinte será executado até encontrar a instrução “*break*”, indo o fluxo para o final da estrutura. Já a instrução “*default*”, se chegar a sua realização, executa o trecho de código na condição de que nenhuma das situações definidas nos “*case*” anteriores foram verdadeiras. Por esse motivo, a instrução “*default*” fica sempre no final da estrutura, mas pode também não ser utilizada, excluindo-a.

A Figura 22 apresenta o comportamento de uma estrutura de execução condicional de blocos.

Figura 22 – Comportamento da estrutura *switch-case*, execução de blocos condicionados, em um programa



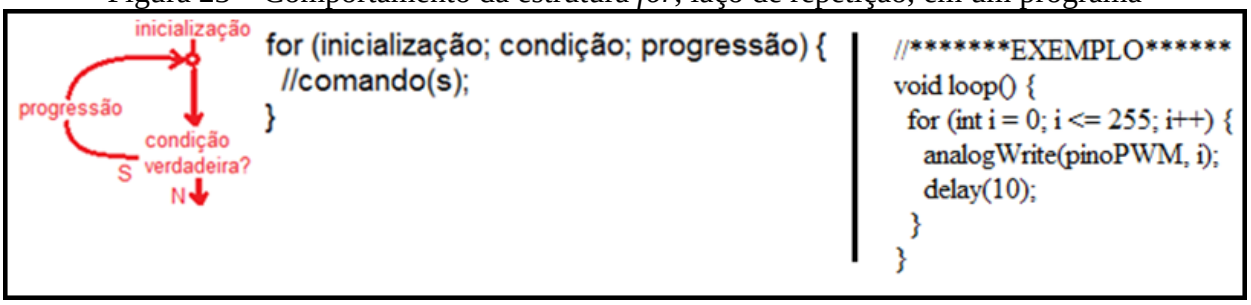
Fonte: Autor (2021)

Essa estrutura lógica só aceita a realização de teste condicional com variáveis do tipo *int* (inteiro) ou *char* (caractere).

O segundo tipo de estrutura são as estruturas de repetição. Uma delas é o laço de repetição implementado pela instrução “*for*”. Através dele é possível repetir a realização de um trecho de código, sendo o número de repetições e a forma com que progredirá a contagem definida por parâmetros. Um laço de repetição tem uma variável de contagem, que será inicializada no início de sua execução. A cada vez que uma passagem pelo laço de repetição chega ao final, a variável de contagem tem uma progressão indicada. Neste momento, se a condição de encerramento não foi atingida, o fluxo de programa retorna ao início do “*for*”, repetindo o trecho de código delimitado pelas chaves “{ }”.

A Figura 23 apresenta o comportamento de um laço de repetição, com exemplo de aplicação.

Figura 23 – Comportamento da estrutura *for*, laço de repetição, em um programa



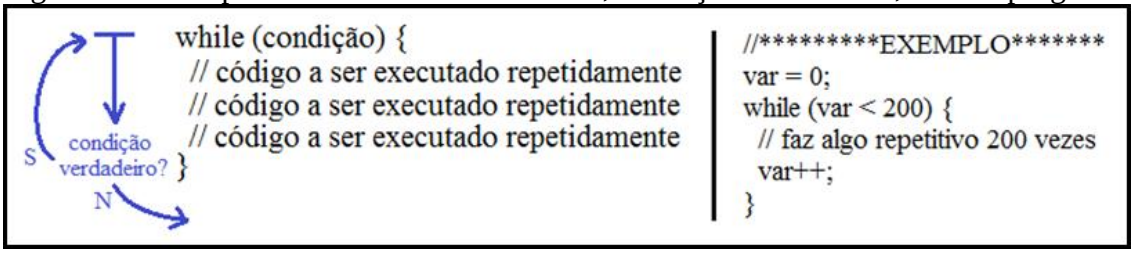
Fonte: Autor (2021)

No exemplo da figura, o laço de repetição "for" se inicia com a variável "i" valendo 0. O trecho de código definido entre as chaves é executado. Ao chegar ao seu final, a variável "i" é incrementada pela instrução "i++", procedendo o teste da condição "i <= 255". Sendo ela verdadeira, significando que "i" ainda não chegou ao valor 255, o trecho de código é novamente executado. Isto se repete até que a variável "i" assuma o valor 255 ou mais, encerrando o laço de repetição.

Já a estrutura de repetição implementada pela instrução "*while*" permite que um trecho de código seja executado enquanto uma condição seja verdadeira. É uma estrutura à qual se deve prestar atenção na utilização, pois pode provocar um *loop* infinito caso a condição nunca seja satisfeita. Assim como se deve ter cuidado que se o programa se encontra dentro desta estrutura, ele não realizará outras coisas.

Uma vez que a condição seja verdadeira, o trecho de código definido entre chaves "{ }" será executado até que a condição deixe de ser verdadeira. A Figura 24 apresenta o comportamento da estrutura "*while*", com exemplo de execução.

Figura 24 – Comportamento da estrutura *while*, execução condicional, em um programa



Fonte: Autor (2021)

Neste exemplo, o trecho de código definido entre as chaves será executado enquanto o valor da variável "*var*" for menor do que 200.

Por fim, há a estrutura “do {} while” (condição), que executará o trecho entre as chaves pelo menos uma vez antes de verificar a condição.

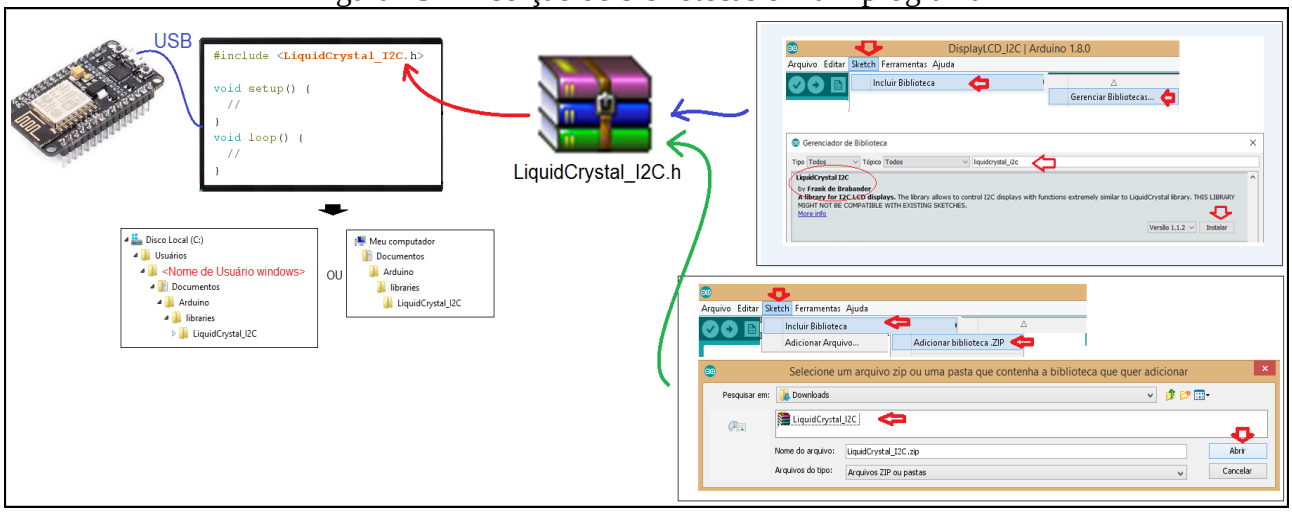
1.4.7 Bibliotecas

Uma biblioteca é um conjunto de códigos que podem ser adicionados a um programa. Representam um arquivo que contém funções, que são disponibilizadas ao programador, bastando sua inserção no programa que se está editando (BAUERMEISTER, 2018).

A inserção da biblioteca se dá pela diretiva #include <nome_biblioteca.h>, que inclui um cabeçalho (*header*) com as definições das funções.

As bibliotecas auxiliam muito desenvolvimento de programas, pois agregam funções que não aparecem no seu código, mas que estão lá inseridas, podendo ser utilizadas. Vale-se do princípio de que não é necessário conhecer o código de uma função, mas sim a funcionalidade que ela apresenta. A Figura 25 apresenta a estrutura de inserção de uma biblioteca em um programa para o módulo ESP8266 NodeMCU.

Figura 25 – Inserção de bibliotecas em um programa



Fonte: Autor (2021)

A inserção de uma biblioteca pode ser realizada por dois caminhos. O primeiro deles, e o mais indicado, é procurar bibliotecas já validadas pelo próprio IDE do Arduino. Isto é realizado clicando na aba “Sketch”, “Incluir Biblioteca” e “Gerenciar Bibliotecas”. Uma vez aberta a caixa de seleção de bibliotecas, indicar o nome da biblioteca que se está procurando e a mesma surgirá nas bibliotecas selecionadas. Para proceder sua instalação, basta clicar no botão “Instalar”.

Caso não se localize a biblioteca desejada entre as já validadas e disponíveis no IDE Arduino, é possível inserir uma biblioteca já desenvolvida e disponibilizada por outra pessoa, bastando ter o arquivo da biblioteca em uma pasta compactada. Isto é realizado clicando na aba “Sketch”, “Incluir Biblioteca”, “Adicionar biblioteca.zip”. Ao abrir a caixa de seleção, indicar o caminho onde se encontra a biblioteca compactada e selecionar “Abrir”.

Realizados estes passos, dentro do programa, na IDE Arduino, basta executar a diretiva #include <nome_biblioteca.h>, valendo-se das funções que a biblioteca disponibiliza.

Em ambos os casos, a biblioteca será armazenada na pasta “Documentos”, “Arduino”, “libraries”. Em alguns casos pode ser necessária a remoção da biblioteca, seja por não apresentar o resultado esperado ou pela necessidade de instalar uma outra biblioteca com mesmo nome. Neste caso, deve-se excluir a pasta da biblioteca alocada neste caminho: “Documentos”, “Arduino” e “libraries”.

1.5 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação permitem a troca de sinais entre o módulo ESP8266 NodeMCU com outros dispositivos, sejam microcontroladores, computadores ou periféricos, de forma consistente e utilizando poucos pinos. São compostos de blocos próprios embarcados no módulo, disponibilizando os pinos de acesso ao usuário em pinos GPIO específicos.

No módulo ESP8266 NodeMCU há 3 protocolos de comunicação disponibilizados: UART, I2C e SPI.

1.5.1 Protocolo UART

O protocolo UART (protocolo UART (*Universal Asynchronous Receiver/Transmitter*) é um método de transmissão e recepção de sinais seriais entre 2 dispositivos. Possui um canal de envio de mensagens (Tx) e um canal para o recebimento de mensagens (Rx). As mensagens em palavras de 8 bits são transmitidas com um conjunto de bits para controle (OLIVEIRA, 2017), sendo eles:

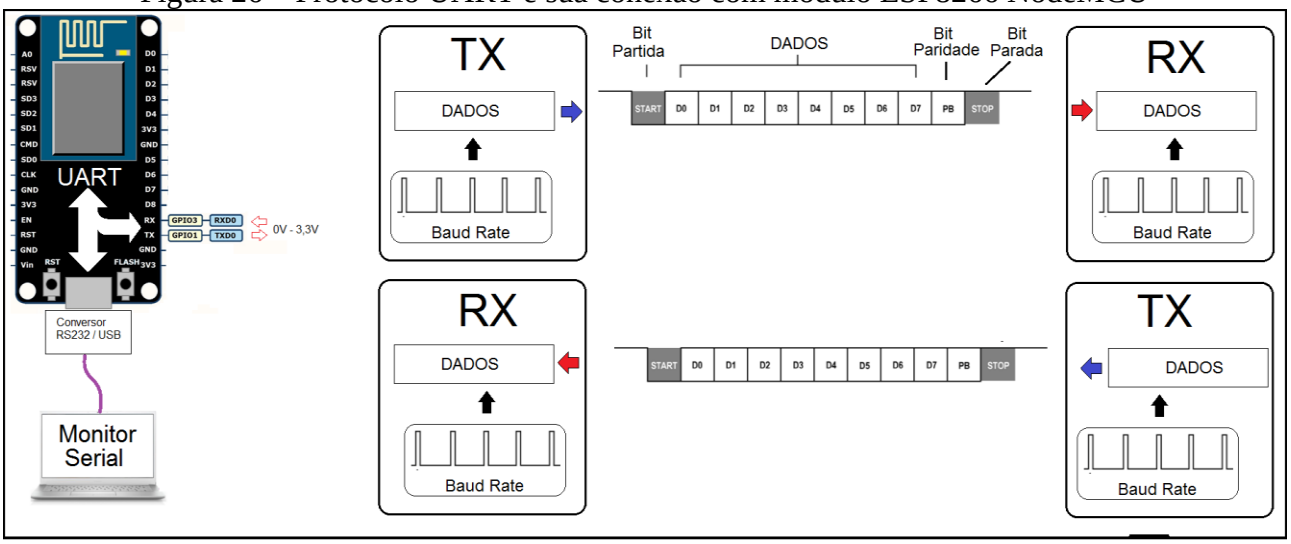
a) Bit de partida (SB - *Start Bit*), que indica que uma transmissão vai começar;
b) Bits de dados (DB – *Data Bits*), que correspondem a uma sequência de 8 bits correspondente à palavra digital que se quer transmitir;
c) Bit de paridade (PB - *Parity Bit*), que corresponde a uma informação definida pelo número de bits em nível alto (valor igual a 1) na mensagem, e que é utilizada para a detecção de erros de transmissão. Há dois esquemas possíveis: paridade par e paridade ímpar. Na paridade par, o bit

de paridade tem valor 0 se o número de bits de nível 1 nos dados for par, e tem valor 1 se o número de bits de nível 1 nos dados for ímpar (totalizando um número par com o bit de paridade). Já na paridade ímpar, o número total de bits de nível 1 no dado mais o do bit de paridade deve ser ímpar. A presença ou ausência do bit de paridade normalmente é configurada por *software*.

d) Bit de parada (SB – *Stop Bit*), que indica que a transmissão chegou ao fim.

A Figura 26 apresenta o protocolo UART e sua pinagem no módulo ESP8266 NodeMCU.

Figura 26 – Protocolo UART e sua conexão com módulo ESP8266 NodeMCU



Fonte: Autor (2021)

O sincronismo entre o envio e o recebimento de mensagens se dá por um circuito oscilador em cada um dos dispositivos de comunicação, oscilando na mesma frequência, parametrizado por um *Baud Rate* (taxa de transmissão). Ambos os elementos da comunicação (transmissor e receptor) devem estar configurados com mesmo *Baud Rate.* Da mesma forma, devem estar com a mesma configuração de bit de paridade.

Os sinais disponibilizados pelo módulo são os pinos Rx e Tx, com valores de tensão em 0V e 3,3V. Mas, internamente, há um conversor USB que padroniza estes níveis de tensão para uso neste meio físico, permitindo a conexão com um computador. É desta forma que a IDE Arduino se comunica com o módulo ESP8266 NodeMCU para carregamento dos programas criados. E também acessá-los como uma interface serial (Monitor Serial), enviando e recebendo mensagens de texto, onde cada caractere é transmitido ou recebido individualmente, em uma palavra de 8 bits, no protocolo UART.

Este protocolo corresponde aos seguintes parâmetros:

a) Taxa de comunicação: até 115.200 bit/s;

b) Método de transmissão: assíncrona;

c) Número de fios: 3 (Rx, Tx e Gnd);

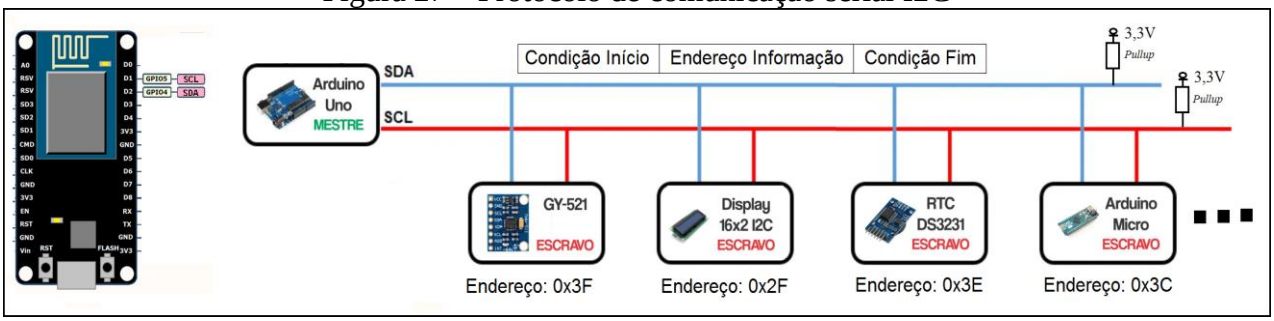
d) Máximo de dispositivos comunicando: 2.

1.5.2 Protocolo I2C

O protocolo de comunicação serial I2C permite que um dispositivo mestre se comunique com até 127 dispositivos escravos, através de dois sinais. Um deles é o pino *Serial Clock* (SCL), que emite um sinal de sincronismo para os dados seriais se propagarem na ordem e no tempo correto. O outro é o *Serial Data* (SDA), onde os dados são transmitidos do mestre para o escravo e do escravo para o mestre, através de uma sequência de dados seriais que começa com uma condição de início (SCL em alto e SDA indo de alto para baixo), uma palavra digital que identifica o endereço do escravo (8bits), uma palavra digital que identifica a informação a ser transferida e uma condição de fim (SCL em alto e SDA indo de baixo para alto) (MADEIRA, 2021).

A Figura 27 apresenta o protocolo de comunicação I2C com sua pinagem.

Figura 27 – Protocolo de comunicação serial I2C



Fonte: Adaptado de Madeira (2021)

No módulo ESP8266 NodeMCU este protocolo utiliza os pinos SCL(D1, GPIO-5) e SDA (D2, GPIO-4).

Para utilização das funções do protocolo I2C é necessário a inserção da biblioteca *Wire.h*, nativa na IDE Arduino. Não há necessidade de instala-la, basta inseri-la pela diretiva *#include <Wire.h>*.

Este protocolo corresponde aos seguintes parâmetros:

e) Taxa de comunicação: *Standard*(Slow, 100kbit/s), *Fast*(400kbit/s) e *FastPlus*(1Mbit/s);

f) Método de transmissão: síncrona;

g) Número de fios: 3 (SCL, SCA, Vdd);

h) Máximo de dispositivos comunicando: até 127.

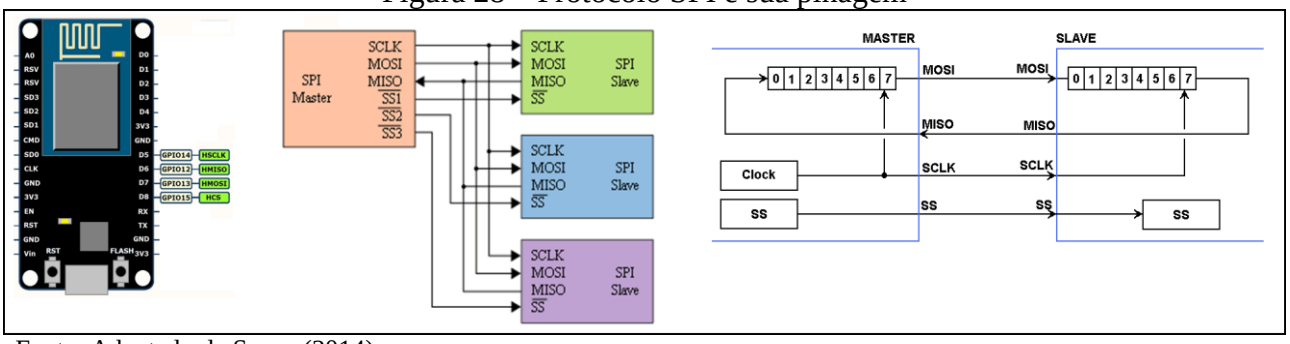
1.5.3 Protocolo SPI

O protocolo é a troca rápida de dados entre um componente mestre e o escravo selecionado, utilizando os pinos SCLK (gera o sinal de sincronismo), MOSI (que transmite o dado do mestre para o escravo) e o MISO (que transmite o dado do escravo para o mestre). Há também o sinal de CS (*Chip Select*) ou SS (*Select Slave*) que habilita o escravo que irá se comunicar (SACCO, 2014).

O sinal de CS ou SS pode ser qualquer saída digital do módulo ESP8266 NodeMCU, inclusive uma expansão destas, de forma que o número de escravos que esse protocolo limita é o número de sinais de seleção de escravos que o módulo disponibilizar.

A Figura 28 apresenta o protocolo SPI com sua pinagem.

Figura 28 – Protocolo SPI e sua pinagem



Fonte: Adaptado de Sacco (2014)

Este protocolo corresponde aos seguintes parâmetros:

i) Taxa de comunicação: até 10Mbit/s (depende do dispositivo destino);
j) Método de transmissão: síncrona;
k) Número de fios: 3 fios, mais um fio para cara escravo e um fio para Gnd;
l) Máximo de dispositivos comunicando: não há limitação, depende do quanto se pode selecionar os escravos.

2 ENTRADAS E SAÍDAS

Os componentes programáveis que se destinam a acionamentos e leitura de dispositivos físicos são caracterizados por terem portas lógicas, como interface ao meio físico. No módulo ESP8266 NodeMCU essas portas são chamadas GPIO (*General Purpose Input/Output*). Essas portas podem ser digitais ou analógicas, dependendo do que o módulo disponibiliza, além de poderem ser portas de entrada ou saída, sendo essa definição configurável pelo programa, a ser executado.

2.1 SAÍDAS DIGITAIS

As saídas digitais no módulo ESP8266 NodeMCU são portas que fornecem um sinal digital. Ou a saída está em nível lógico 0 (*LOW*) ou a saída está em nível lógico 1 (*HIGH*). O nível lógico 1 fornece uma tensão de 3,3V, com uma capacidade máxima de corrente elétrica de 15mA.

Assim sendo, para que o módulo acione um LED na sua saída digital é necessário garantir que a máxima corrente não será ultrapassada, dimensionando corretamente o resistor de limitação. A Equação 1 demonstra esse cálculo para um LED vermelho com tensão entre anodo e catodo de 1,7V e corrente direta de 15mA.

$$R = \frac{3{,}3V - V_{LED}}{15mA} = \frac{3{,}3-1{,}7}{0{,}015} = 106{,}6\Omega\ mínimo \qquad (1)$$

O Quadro 11 apresenta a relação de pinos de saída digital que o módulo ESP8266 NodeMCU disponibiliza para uso.

Quadro 11 – Relação de pinos de saídas digitais que o ESP8266 NodeMCU disponibiliza

Pino	GPIO	Saída Digital
D0	GPIO-16	Sim
D1	GPIO-5	Sim
D2	GPIO-4	Sim
D3	GPIO-0	Sim
D4 [10]	GPIO-2	Sim
D5	GPIO-14	Sim
D6	GPIO-12	Sim
D7	GPIO-13	Sim
D8	GPIO-15	Sim

Fonte: Autor (2021)

[10] Conexão com o LED *Builtin* da placa, com lógica invertida

O pino de saída deve ser configurado como tal durante a execução da função *setup(),* através da execução da função *pinMode(número_GPIO, estado_GPIO).*

Como exemplo: *pinMode(0, OUTPUT);* configura o GPIO-0, pino D3, como saída.

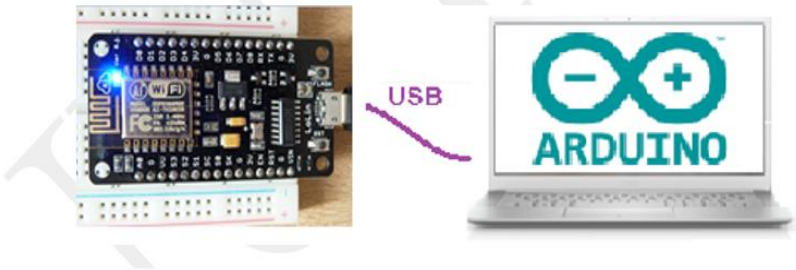
Já o acionamento dessa saída digital se dá pela execução da função *digitalWrite(GPIO, Nível_Lógico).* Neste caso, o nível lógico será *HIGH* (3,3V) ou *LOW* (0V).

Como exemplo, o comando *digitalWrite(0, HIGH);* coloca o pino de saída D3 em estado lógico 1 (3,3V). Já o comando *digitalWrite(0, LOW);* coloca o pino de saída D3 em estado lógico 0 (0V).

A Figura 29 apresenta a implementação de um pisca-led no LED disponibilizado no próprio módulo, chamado *LED_BUILTIN*. Esse LED está conectado internamente no GPIO 2, pino D4, sendo sua lógica invertida: um nível lógico alto (*HIGH*) faz o LED apagar; um nível lógico baixo (*LOW*) faz o LED ligar.

Figura 29 – Implementação de pisca-led no LED *Builtin*, utilizando a função *delay*()

```
void setup(){
    pinMode(LED_BUILTIN, OUTPUT);
}
void loop(){
    digitalWrite(LED_BUILTIN, LOW);
    delay(500);
    digitalWrite(LED_BUILTIN, HIGH);
    delay (500);
}
```



Fonte: Autor (2021)

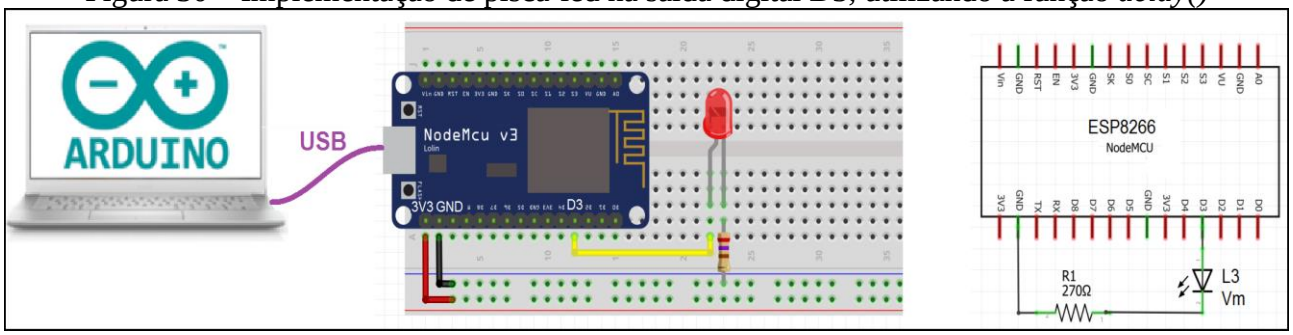
A função *setup*(), que é executada uma única vez ao ligar o módulo, configura o GPIO-2, pino D4 como uma saída digital, através da execução de *pinMode(LED_BUILTIN, OUTPUT)*. A expressão *LED_BUILTIN* já é uma constante interna, definida como sendo igual a 2, ou seja, o comando *#define LED_BUILTIN 2* já é subentendido. Porém, para algumas constantes, isto já está definido no IDE Arduino, não necessitando sua definição prévia.

Já na função *loop*(), que é executada repetidamente após a execução da função *setup*(), há o comando *digitalWrite(LED_BUILTIN, LOW)*, que liga o *LED_BUILTIN*, seguido do comando *delay*(500), que para a execução do programa por 500ms (milissegundos) ou 0,5s (segundos), quando então executa o comando *digitalWrite(LED_BUILTIN, HIGH)*, que desliga o *LED_BUILTIN*, seguindo novamente pelo comando *delay*(500). Dentro da função *loop()*, essa sequência de 4

comandos irá se repetir até que o módulo seja desligado, tendo como resultado o fato do *LED_BUILTIN* piscar a cada 0,5 segundos.

A mesma lógica se aplica para utilização dos componentes externos. Neste caso, se propõe a colocação de um LED conectado ao pino D3, GPIO-0, com objetivo de piscar a cada 0,5 segundos. A Figura 30 apresenta essa implementação.

Figura 30 – Implementação de pisca-led na saída digital D3, utilizando a função *delay()*



Fonte: Autor (2021)

O código para esse circuito é o seguinte:

```
//Led 3 ligando no pino D3, GPIO-0
#define L3 0
void setup(){
    pinMode(L3, OUTPUT);
}
void loop(){
    digitalWrite(L3, HIGH);
    delay(500);
    digitalWrite(L3, LOW);
    delay(500);
}
```

A diferença aqui é que se deve definir o GPIO em que está ligado o LED L3 no *hardware*. E isso é definido pela diretiva *#define L3 0*.

Como o LED L3 está ligado ao pino D3 (GPIO-0) no seu terminal anodo, tendo seu terminal catodo ligado à referência (GND) através de um resistor de 270Ω, então para que L3 ligue é necessário um nível lógico alto, proporcionado ao executar o comando *digitalWrite*(L3, HIGH), assim como o desligar de L3 ocorre com o comando *digitalWrite*(L3, LOW).

Em ambos os programas se utilizou a função *delay*() para a execução do programa pelo tempo indicado, em milissegundos. Exemplo: *delay*(10000), para o programa por 10 segundos. Isso significa que enquanto estiver executando a função *delay*() nada mais estará ocorrendo na execução do programa. E isso pode ser um inconveniente em uma aplicação.

Na situação desejada em que há 2 LEDs, um deve piscar a cada 1 segundo e outro deve piscar a cada 0,5 segundos. Nesse caso, ao executar o *delay*() na rotina de acionamento de um LED impactaria na rotina do outro.

Para esse tipo de situação, mais frequente do que se imagina, sugere-se a utilização da função *millis*(), que retorna um valor numérico correspondente ao tempo decorrido desde a energização do módulo ESP8266 NodeMCU.

Dessa forma, o programa em *loop*() fica monitorando a cada ciclo se o tempo necessário já foi atingido, podendo então realizar outras coisas até alcançar o tempo desejado. Basta observar o trecho de código seguinte:

```
long Tempo1, Tempo2;
void loop(){
   if(millis() > Tempo1 + 1000){ // trecho de programa executado a cada 1s
        Tempo1 = millis();
   }
   if(millis() > Tempo2 + 5000){ // trecho de programa executado a cada 5s
        Tempo2 = millis();
   }
   ...
}
```

O teste condicional *if(millis() > Tempo1 + 1000)* irá comparar se o tempo atual (em milissegundos deste a energização do módulo) é maior do que a última vez que foi monitorado, acrescido de 1000, ou seja, a cada 1 segundo. No momento em que a condição for verdadeira, o fluxo de programa entra na rotina e executa o trecho de programa desejado, atribuindo ao final disso a Tempo1 o tempo atual, em Tempo1 = *millis()*.

Assim, a função *loop*() ficará monitorando cada teste condicional no seu tempo de execução, podendo criar janelas de tempo diferenciadas para cada trecho de programa desejado.

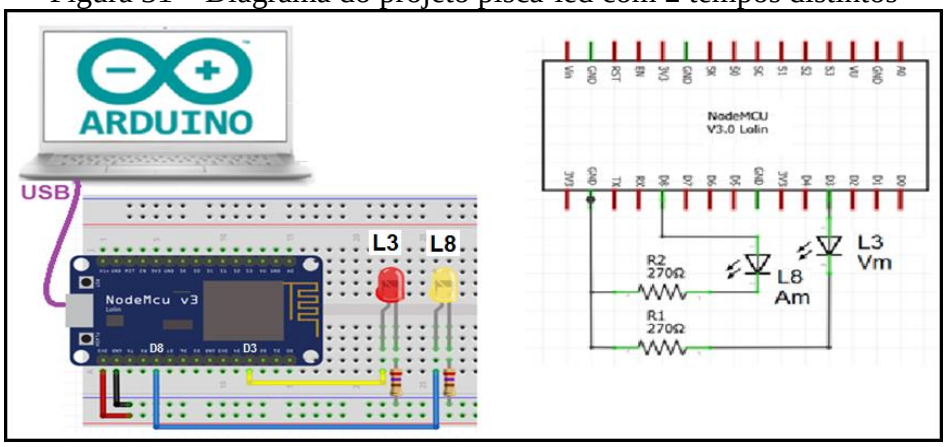
É bom lembrar que uma saída digital que fornece níveis lógicos binários, cujas tensões são 0V e 3,3V, pode acionar outros dispositivos periféricos, dependendo do *driver* que se construa. Como exemplo: relés elétricos, motores elétricos, lâmpadas, solenoides, entre outros.

PROJETO 1 – PISCA-LED COM 2 TEMPOS DISTINTOS

"A proposta deste projeto é fazer 2 LEDs distintos piscarem em frequências diferentes, sendo um a cada meio segundo e outro a cada segundo."

Este projeto apresenta a implementação de um pisca-led com dois LEDs, onde cada um pisca em uma frequência diferente, nas saídas D3 e D8, utilizando a função *millis*(), como representado na Figura 31.

Figura 31 – Diagrama do projeto pisca-led com 2 tempos distintos



Fonte: Autor (2021)

O código para execução dessa aplicação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define L3 0 //led L3 ligando no pino GPIO-0
#define L8 15 //led L8 ligando no pino GPIO-15
long Tempo1, Tempo2;
int referenciaTempo1 = 1000; // 1000 milisegundos
int referenciaTempo2 = 500; // 500 milisegundos

void setup() {
  pinMode(L3,OUTPUT);
  pinMode(L8,OUTPUT);
  Tempo1= millis();
  Tempo2= millis();
}

void loop(){
  if (millis() > Tempo1 + referenciaTempo1){
      Tempo1= millis();
      digitalWrite(L3,HIGH);
      delay(50);
      digitalWrite(L3,LOW);
  }
  if (millis() > Tempo2 + referenciaTempo2){
      Tempo2= millis();
      digitalWrite(L8,HIGH);
      delay(50);
      digitalWrite(L8,LOW);
  }
}
```

2.2 ENTRADAS DIGITAIS

As entradas digitais do módulo ESP8266 NodeMCU são portas configuradas para esse fim, de modo a monitorar seu estado lógico. Se a entrada tem aplicada uma tensão de 0V a 1V é considerado um nível lógico 0 (*LOW*), se a entrada tem aplicada uma tensão de 2V a 3,3V é considerado um nível lógico 1 (*HIGH*). O Quadro 12 apresenta a relação de pinos de entradas digitais que o módulo ESP8266 NodeMCU disponibiliza para uso.

Quadro 12 – Relação de pinos de entradas digitais que o ESP8266 NodeMCU disponibiliza

Pino	**GPIO**	**Entrada Digital**
D0	GPIO-16	Sim
D1	GPIO-5	Sim
D2	GPIO-4	Sim
D5	GPIO-14	Sim
D6	GPIO-12	Sim
D7	GPIO-13	Sim

Fonte: Autor (2021)

Para utilização de uma porta como entrada é necessário que a configure como tal. Isso é realizado pela função *pinMode(GPIO, estado),* sendo o estado uma constate que neste caso deve ser INPUT. Como trata-se de configuração, esta função deve estar dentro da função *setup*(), para ser realizada uma única vez, ao energizar o módulo.

Já a leitura de estado lógico da entrada é realizada pela função *digitalRead(GPIO),* que retorna um valor booleano (bool) indicando as constantes *LOW* (nível baixo) ou *HIGH* (nível alto).

O trecho de código seguinte é um bom exemplo. Se a entrada D0 (GPIO-16) estiver em nível lógico alto, um trecho de código será realizado. Do contrário nada acontece.

```
#define CH 16
bool a;
void setup(){
    pinMode(CH, INPUT);
}
void loop(){
    a = digitalRead(CH);
    if(a == HIGH){
        // realiza o programa para entrada acionada
    }
}
```

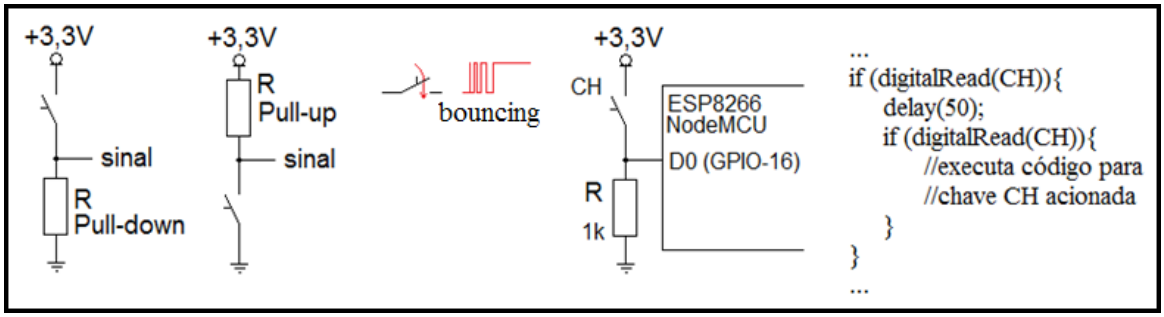
No caso específico de se utilizar chaves ou botões como elementos de entrada de sinal, deve-se levar em consideração o efeito *bouncing* (BERNARDO, 2016). Trata-se do fato de que tanto a chave quanto o botão possuem contatos mecânico que, quando acionados, acabam por "ricochetear" sua conexão física, emitindo vários sinais até que se estabilize mecanicamente. Como a velocidade

de processamento do microcontrolador é muito mais rápida do que esse ruído mecânico, ele acaba por ser percebido.

Outro fator relevante é que botões e chaves permitem ou não permitem a circulação de corrente elétrica. Nesse caso, um botão quando ligado entre um pino de entrada e a alimentação 3,3V, por si só envia um sinal de 3,3V quando pressionado, mas não envia nada quando está solto. O mesmo ocorre se o botão estiver conectado entre o pino de entrada e a referência (0V), pois ele envia 0V quando estiver pressionado, mas não envia nada quando estiver solto. Nesses dois casos é necessária a colocação de um resistor para garantir o outro sinal de tensão que o botão não consegue dar conta.

Em o botão conectado entre o pino e a alimentação 3,3V, se utiliza um resistor de *pull-up*. Já se o botão estiver conectado entre o pino e a referência, se utiliza um resistor de *pull-down*. O valor ômico deste resistor pode ser entre 1k e 4k7. A Figura 32 mostra a configuração desses resistores, o efeito *boucing* e o código para evitar o ruído do botão.

Figura 32 – Entrada digital com resistores de *pull-down*, *pull-up*, efeito *bouncing* e código para evitar ruídos



Fonte: Autor (2021)

Uma forma bem eficaz de evitar o efeito *bouncing*, como demonstrado na figura, é monitorar duplamente o sinal de entrada. A primeira monitoração serve para saber se o sinal de entrada foi acionado, a segunda para saber se continua acionado, sendo definido um tempo estre os dois monitoramentos, o suficiente para que o efeito *bouncing* já tenha ocorrido e o sinal estabilizado. Essa técnica é conhecida como *Debouncing*.

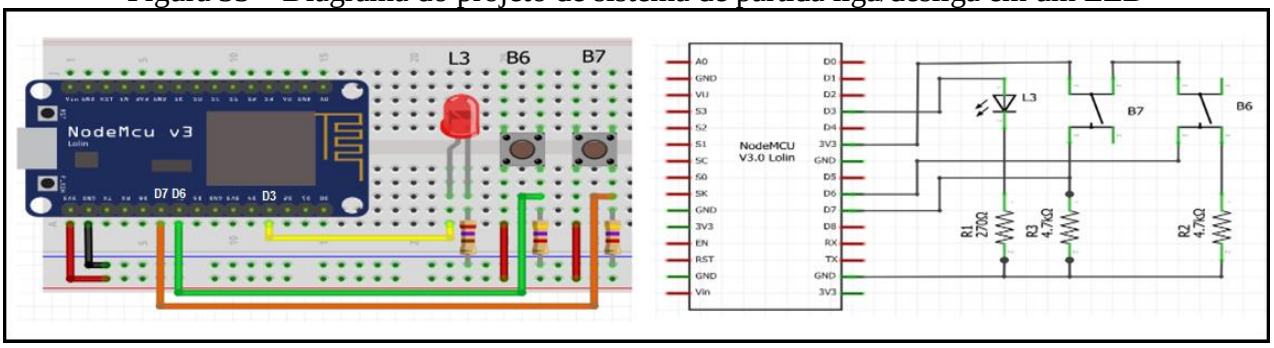
PROJETO 2 – SISTEMA DE PARTIDA LIGA/DESLIGA EM UM LED

"O projeto é composto de 2 botões de entrada e 1 LED de saída. Ao pulsar o botão B6, o LED L3 liga, ao pulsar o botão B7, o LED L3 desliga.

Projeto onde, ao pulsar o botão B6, o LED L3 liga e ao pulsar o botão B7, o LED L3 desliga.

A Figura 33 apresenta o diagrama desse projeto.

Figura 33 – Diagrama do projeto de sistema de partida liga/desliga em um LED



Fonte: Autor (2021)

O código para essa aplicação é o seguinte.

```
//Projeto 2 - Entrada digital liga / desliga
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define L3 0  //pino D3
#define B6 12 //pino D6
#define B7 13 //pino D7

void setup(){
  pinMode(L3,OUTPUT);
  pinMode(B6,INPUT);
  pinMode(B7,INPUT);
}
void loop(){
  if(digitalRead(B6)){
    delay(50);
    if(digitalRead(B6)){
      digitalWrite(L3,HIGH);
    }
  }
  if(digitalRead(B7)){
    delay(50);
    if(digitalRead(B7)){
      digitalWrite(L3,LOW);
    }
  }
}
```

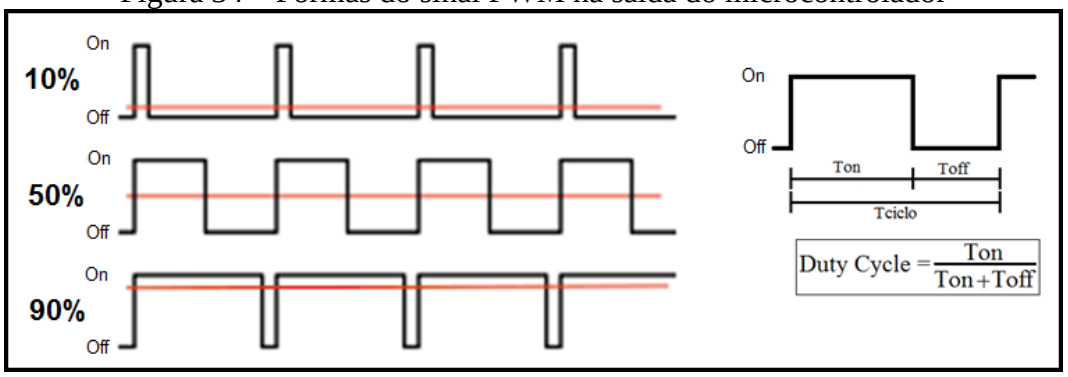
2.3 SAÍDAS ANALÓGICAS - PWM

As saídas analógicas disponibilizadas pelo módulo ESP8266 NodeMCU fornecem um sinal que possui 1.024 valores distintos (escala). Porém, não são níveis de tensão direta, mas sinal de modulação por largura de pulso, chamado PWM (*Pulse Width Modulation*) (MADEIRA, 2019).

O sinal PWM consiste de um valor digital, ou 0V ou 3,3V, que é apresentado na saída do módulo na forma pulsada, sendo que se pode ajustar o tempo que o sinal fica em nível lógico alto (3,3V) e o tempo que o sinal fica em nível lógico baixo (0V), dentro de um ciclo que internamente é ajustado entre 500Hz a 1kHz, dependendo do microcontrolador. Com isso, o sinal médio de tensão acaba por ter um comportamento analógico, respondendo a média da tensão resultante do sinal.

A configuração que se dá no sinal PWM é alterar o seu *duty cycle* (ciclo de trabalho), razão entre o tempo ligado e o tempo total do ciclo. Com isso, se diminuir o tempo ligado, aumentará o tempo desligado, e vice-versa, o que implica que a saída terá sempre a mesma frequência fundamental do sinal. A Figura 34 apresenta as formas de sinais PWM como exemplo e a formação do *duty cycle*.

Figura 34 – Formas do sinal PWM na saída do microcontrolador



Fonte: Autor (2021)

Não são todos pinos de saída do módulo ESP8266 NodeMCU que tem a função de saída analógica PWM. O Quadro 13 apresenta a relação de pinos de saída digital que o módulo disponibiliza para uso.

Quadro 13 – Relação de pinos de saídas PWM que o ESP8266 NodeMCU disponibiliza

Pino	GPIO	Saída PWM	Escala
D2	GPIO-4	Sim	0 a 1023
D5	GPIO-14	Sim	0 a 1023
D6	GPIO-12	Sim	0 a 1023
D8	GPIO-15	Sim	0 a 1023

Fonte: Autor (2021)

Para utilização de uma porta como saída é necessário que a configure como tal. Isso é realizado pela função *pinMode(GPIO, estado)*, sendo o estado uma constante que, neste caso deve, ser OUTPUT. Como se trata de configuração, essa função deve estar dentro da função *setup*(), para ser realizada uma única vez, ao energizar o módulo.

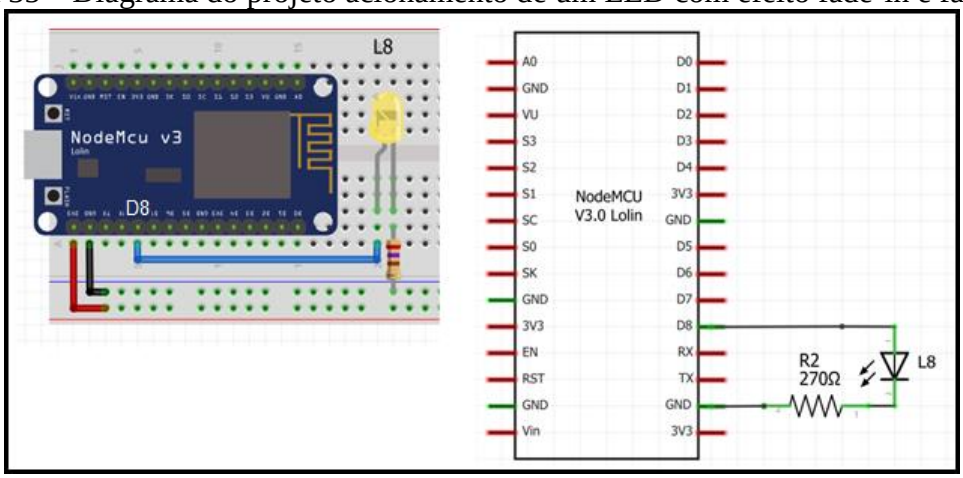
Já a escrita do parâmetro PWM na saída é realizado pela função *analogWrite(GPIO, valor)*, sendo o "valor" um número de 0 a 1.023, correspondendo proporcionalmente o *duty-cycle* do sinal PWM. Por exemplo, um valor de 500 na escala de 0 a 1.023 corresponde a um *duty-cycle* de 48%, o que significa que o sinal de saída ficará 48% do tempo em nível alto (3,3V) e o restante do tempo em nível baixo (0V), dentro do tempo de ciclo.

PROJETO 3 – ACIONAMENTO DE UM LED COM EFEITO FADE-IN E FADE-OUT

"O projeto faz com que o LED L8 comece a aumentar seu brilho até atingir o nível máximo, quando então começa a diminuir o brilho até atingir o nível mínimo, repetindo esse ciclo."

O projeto apresenta o código e diagramas do projeto em que o LED L8 liga e desliga em um efeito *fade-in* e *fade-out*. A Figura 35 apresenta o diagrama desse projeto.

Figura 35 – Diagrama do projeto acionamento de um LED com efeito fade-in e fade-out



Fonte: Autor (2021)

O código para essa aplicação é o seguinte:

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define L8 15 //led ligando no pino GPIO15
void setup() {
  pinMode(L8,OUTPUT);
}

void loop(){
  for(int a=0;a<1023;a++){
    analogWrite(L8,a);
    delay(10);
  }
  for(int a=1023;a>0;a--){
    analogWrite(L8,a);
    delay(10);
  }
}
```

2.4 ENTRADA ANALÓGICA

O módulo ESP8266 NodeMCU possui uma única entrada de sinal analógico, no pino A0. Essa entrada está ligada internamente ao canal analógico do chip ESP8266, cuja resolução é de 10 bit, a uma taxa de amostragem típica de 5kHz. Isso significa que pode converter o valor de tensão de entrada em uma faixa de valores de 0 a 1.023 (10 bits, 2^{10}=1.024). Internamente, no chip ESP8266 há um limite de tensão de máxima no canal analógico de 1V, mas o módulo ESP8266 NodeMCU já possui um circuito divisor de tensão que permite, na entrada A0 do módulo, uma tensão de 0V a 3,3V (CADERNO DE LABORATÓRIO, 2021).

O Quadro 14 apresenta o pino de entrada analógica que o módulo ESP8266 NodeMCU disponibiliza para uso.

Quadro 14 – Pino de entrada analógica que o ESP8266 NodeMCU disponibiliza

Pino	**GPIO**	**Entrada Analógica**
A0	GPIO-0	0V a 3,3V

Fonte: Autor (2021)

Para leitura desse sinal analógico de entrada, após o conversor analógico digital, é necessário executar a função *analogRead*(GPIO) dentro do programa, essa função irá retornar um valor inteiro (int) de 0 a 1.023, proporcional a escala analógica de 0V a 3,3V. Antes disto, por vezes será necessário configurar esse GPIO como entrada, através da função *pinMode*(GPIO, INPUT), executada dentro da função *setup*(), mas isso dependerá do IDE Arduino em uso. O trecho de código seguinte fornece uma noção das funções e diretiva necessária para uma leitura do canal analógico:

```
#define A0 0
int valor;
void setup(){
   pinMode(A0, INPUT);
}
void loop(){
   valor = analogRead(A0);
   //restante do programa
}
```

PROJETO 4 – POTENCIÔMETRO COMANDANDO LED COM EFEITO FADE-IN E FADE-OUT

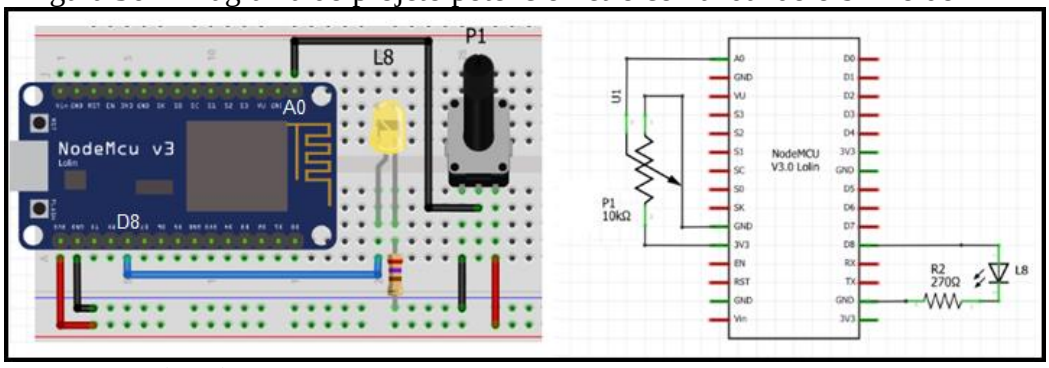
"O projeto faz com que o LED mude sua intensidade de brilho segundo o ajuste realizado no potenciômetro, ligado ao canal analógico."

O diagramas do projeto, em que o LED L8 tenha a intensidade de brilho segundo o ajuste no potenciômetro ligado ao canal analógico.

Neste projeto, o potenciômetro P1 tem suas extremidades ligadas a tensão 3,3V e 0V, o que significa que em seu terminal central, ligado ao pino A0, haverá uma variação de 0V a 3,3V, segundo ajuste do usuário. Essa faixa de valor será convertida pelo conversor analógico digital interno ao microcontrolador em uma faixa numérica de 0 a 1.023, valor esse que será lido pela função *analogRead*(). Esse valor lido é então escrito na saída analógica do pino D8, que emitirá um sinal PWM com *duty-cicle* proporcional a esta faixa de variação numérica. Logo, o ajuste do potenciômetro entre um valor mínimo e máximo representará uma iluminação no LED L8 de apagado a totalmente aceso.

A Figura 36 apresenta o diagrama desse projeto.

Figura 36 – Diagrama do projeto potenciômetro comandando o brilho do LED



Fonte: Autor (2021)

O código para essa aplicação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define L8 15 //led ligando no pino GPIO15
#define P1 A0 //potenciômetro ligado em A0
void setup() {
  pinMode(L8,OUTPUT);
  pinMode(P1,INPUT);
}
void loop(){
  analogWrite(L8,analogRead(P1));
  delay(10);
}
```

3 MONITORAMENTO

O monitoramento da execução de um programa em um microcontrolador é importante para conhecer o que está ocorrendo, qual processo se está executando, qual sinal se está esperando, qual atuador está sendo acionado, ou para servir como uma interface ao usuário, emitindo mensagens para que ele responda através de sinais de entrada do microcontrolador.

Há várias formas de monitorar um programa em execução ou implementar uma interface humano-máquina[11]. Aqui são mostrados dois principais canais para isso: o monitor serial e o uso de *display* matricial.

3.1 MONITOR SERIAL

O módulo ESP8266 NodeMCU possui um canal de comunicação serial, através de dois pinos: RxD e TxD. Estes sinais obedecem ao protocolo de comunicação serial, fornecendo sinais digitais pulsados, com uma sequência formatada dos bits, a uma taxa de transmissão definida (*baud rate*). Isso é realizado pelos pinos RXD (GPIO-3) e TXD (GPIO-1). Porém, o módulo NodeMCU já possui um conversor serial-USB montado na placa, conectado a estes pinos e disposto em um conector USB, que deve ser conectado ao computador para proceder o carregamento do programa. Mas também pode ser utilizado para transmitir e receber informações seriais, fazendo uso do Monitor Serial incorporado no IDE Arduino, ou outro monitor serial que preferir.

Assim, pelo próprio cabo de conexão USB do computador com a placa já é possível enviar e receber caracteres ou *strings* (conjuntos de caracteres).

Para implementar a comunicação serial é necessário executar a biblioteca para porta serial, nativa do IDE Arduino. Logo, não é necessária sua inclusão.

O Quadro 15 apresenta as principais funções da biblioteca para porta serial e suas descrições.

[11] Interface Humano-Máquina (IHM) é um equipamento que propicia a interação entre o humano e a máquina, emitindo mensagens e recebendo comando/informações. Vide https://www.mundodaeletrica.com.br/ihm-o-que-e-para-que-serve/

Quadro 15 – Principais funções da biblioteca nativa serial

Funções	Descrição	Exemplo
Serial.begin()	Configura a taxa de transferência em bits por segundo (*baud rate*) para transmissão serial.	Serial.begin(9600);

Serial.available()	Retorna o número de bytes (caracteres) disponíveis para leitura da porta serial.	if(Serial.available() > 0){
		// tem caracter na entrada serial
		}

Serial.read()	Lê o caractere recebido pelo canal serial.	char a = Serial.read();
Serial.write()	Envia um caractere pelo canal serial	Serial.write('L');
Serial.print()	Envia uma *string* (conjunto de caracteres), sendo impressa na posição atual do cursor que receber.	Serial.print("Hello");
Serial.println()	Envia uma *string* (conjunto de caracteres), agregado da informação de passar o cursor para próxima linha, ao final da impressão.	Serial.println("Hello");

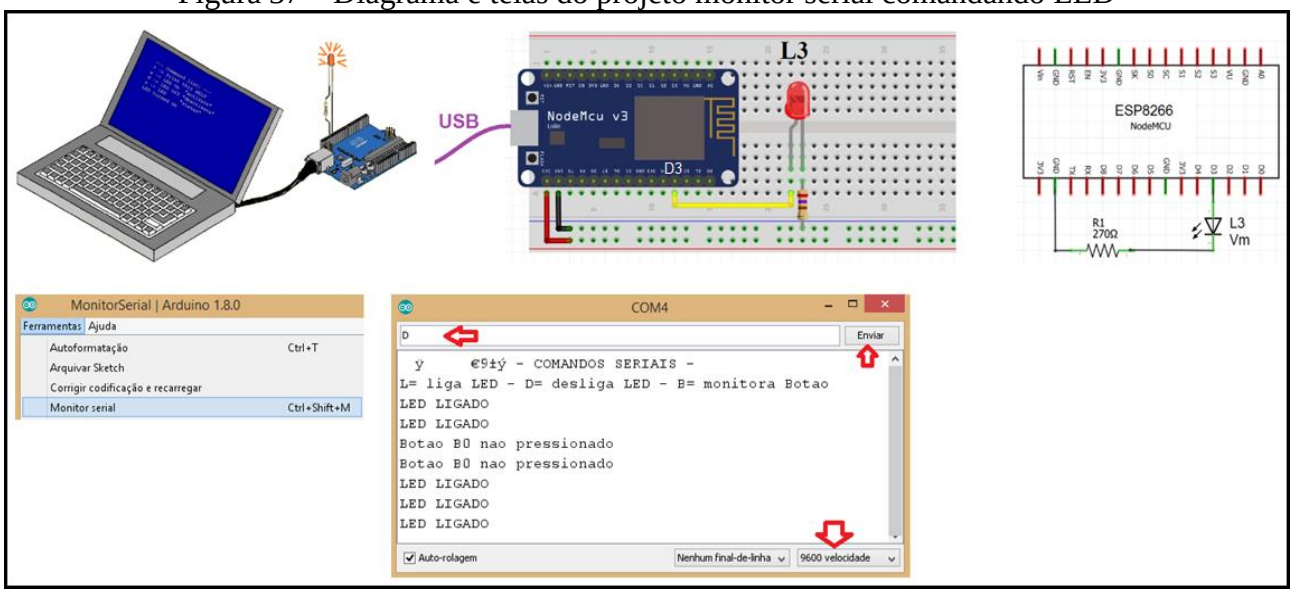
Fonte: Arduino (2021)

PROJETO 5 – MONITOR SERIAL COMANDANDO LED

"O projeto consiste em um monitor serial conectado ao módulo ESP8266 NodeMCU, que emite comandos para ligar ("L") e desligar ("D") o LED L3, recebendo mensagens a respeito do atendimento ao comando solicitado."

O projeto apresenta o diagrama e seu código a ser executado para efetuar comandos e receber mensagens pelo canal serial. A Figura 37 mostra o diagrama e telas do projeto monitor serial comandando um LED.

Figura 37 – Diagrama e telas do projeto monitor serial comandando LED



Fonte: Autor (2021)

O código para essa aplicação é o seguinte.

```
// Placa NodeMCU 1.0 (ESP12E-Module)
#define L3 0 //Pino D3 - GPIO-0
#define B0 16 //Pino D0 - GPIO-16
char rxChar=0; //caracter entrada
void setup(){
  Serial.begin(9600);
  pinMode(L3, OUTPUT);
  pinMode(B0, INPUT);
  Serial.flush(); // zera o buffer de entrada.
  Serial.println(" - COMANDOS SERIAIS - ");
  Serial.println("L= liga LED - D= desliga LED - B= monitora Botao");
}
void loop(){
  if (Serial.available() >0){ // testa se chegou caracter.
    rxChar = Serial.read();   // lê caracter que chegou.
    Serial.flush();           // zera o buffer de entrada.
    switch (rxChar){
      case 'L':  // recebeu 'L':
        digitalWrite(L3,HIGH); // Liga o LED
        Serial.println("LED LIGADO");
```

```
      break;
    case 'D':  // recebeu 'D':
      digitalWrite(L3,LOW); // Desliga o LED
      Serial.println("LED LIGADO");
      break;
    case 'B':  // recebeu 'B':
      if(digitalRead(B0) == HIGH){
        Serial.println("Botao B0 pressionado");
      }else{
        Serial.println("Botao B0 nao pressionado");
      }
      break;
    default:
      Serial.print((char)rxChar);
      Serial.println(" comando invalido!");
   }
  }
}
```

3.2 DISPLAY MATRICIAL LCD COM COMUNICAÇÃO I2C

O *display* matricial é um componente eletrônico cuja função é apresentar mensagens para visualização do usuário. Ele é muito importante para provocar a interação do projeto implementado com o seu usuário, solicitando comandos e informações e enviando mensagens com sua situação atual.

A geração dos caracteres no *display* de cristal líquido (LCD – *Liquid Cristal Display*) ocorre pela polarização de pequenos cristais, provocando um giro que irá permitir a passagem de uma luz posterior. Desta forma, se girar o cristal haverá passagem de luz, se não girar, então a luz não passará. Isto ocorre para cada pixel[12] do *display*, que é gerenciado por um microcontrolador integrado ao LCD, que fornece então os pinos de dados, comandos e controles formatados de forma matricial. O fato de os *displays* matriciais utilizarem cristais líquidos como princípio de funcionamento garante um baixo consumo (MURTA, 2018b).

O *display* matricial possui os pinos de alimentação (VDD e VSS), que devem ser alimentados com 5V e 0V, respectivamente. O pino Vo corresponde ao ajuste de foco, onde deve ser aplicada uma tensão de 0V a 5V, como sugere no diagrama da figura, sendo ajustado por um *trimpot*[13] de 1kΩ. Os pinos D0 à D7 são as vias pelas quais pode-se enviar comandos ou mensagens. Os comandos são informações de 8 bits (número de 0 a 255) sobre como o *display* deve agir com as mensagens que serão enviadas, tais como: posicionar o cursor no início da linha, fazer o cursor piscar, deslocar o cursor a cada entrada de novo caractere, entre outras. Já as mensagens são informações de 8 bits (números de 0 a 255) que correspondem a caracteres previamente formatados, respeitando a Tabela ASCII (WIKIPÉDIA, 2021). Como exemplo, o número 74 corresponde ao caractere “J”, já o número 111 corresponde ao caractere “o” e o número 97 corresponde ao caractere “a”. Logo, ao enviar a sequência de números 74, 111, 97 e 111 como mensagens, o *display* apresentará na sua tela “Joao”.

Através do pino RS é que se define se os dados colocados de D0 a D7 serão interpretados como uma mensagem ou como um comando. Um nível lógico 0 (0V) neste pino indica um comando, um nível lógico 1 (5V) representa uma mensagem.

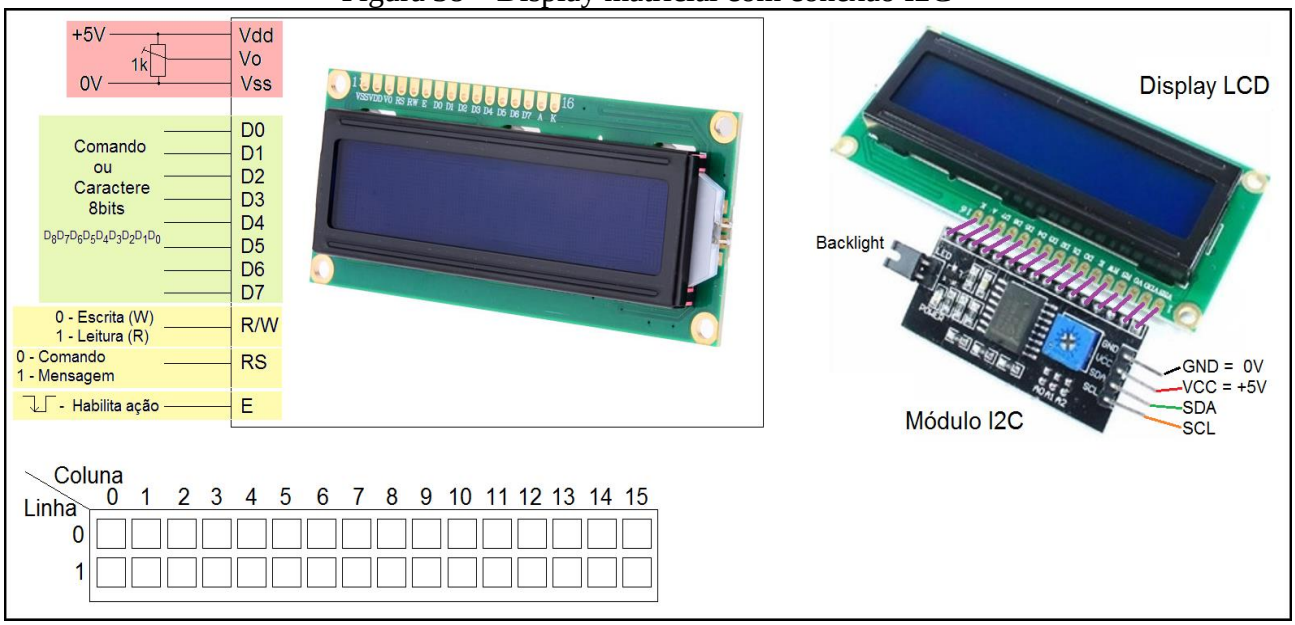
Já o pino R/W indica se a ação executada é uma escrita ou leitura no display. Um pouco estranho, já que um *display* é um elemento de exibição de mensagens, logo, o mais sensato é fazer sempre escritas. A questão é que, por ser um dispositivo microcontrolado, é possível criar caracteres especiais e armazená-los na memória do *display*, para que possam ser lidos por outro dispositivo depois.

[12] *Pixel* é o menor elemento em um dispositivo de exibição. Disponível em: https://pt.wikipedia.org/wiki/Pixel.
[13] *Trimpot* é um resistor variável, onde seu ajuste se dá por um acesso com ferramenta apropriada, como chave de fenda, garantindo características de calibração.

Para sincronizar os sinais de *display* matricial, de forma que sejam lidos/escritos após todos eles estarem devidamente ajustados, há o sinal de habilitação (*Enable*), pino E, que, quando colocado um sinal transitando de nível 1 para nível 0 (5V para 0V), indica ao *display* que ele já pode reconhecer os sinais de controle e efetuar a ação ou informação indicada nos pinos de dados (D0 a D7).

A Figura 38 apresenta a pinagem de um display matricial, bem como sua conexão com módulo I2C e sua matriz de colunas e linhas.

Figura 38 – Display matricial com conexão I2C

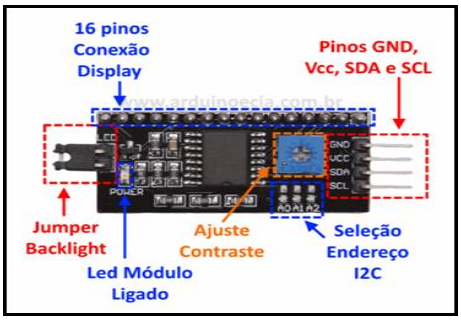


Fonte: Autor (2021)

O display matricial possui dois modos de envio dos 8 bits de comando/mensagem: ou ele envia diretamente os 8 bits, ocupando 8 pinos do microcontrolador que o aciona ou os envia em 2 pacotes de 4 bits, utilizando somente os pinos D5 a D7, ocupando 4 pinos do microcontrolador que o aciona. De qualquer forma, são pinos que devem ficar alocados para esse fim.

Uma forma de evitar a necessidade de alocar portas específicas é utilizar um módulo de acionamento de *display* matricial com protocolo I2C. Esse módulo se comunica com o microcontrolador através de protocolo I2C e provoca o acionamento do *display* matricial LCD, fazendo com que somente 2 pinos do microcontrolador sejam utilizados para esse propósito, com a possibilidade de agregar outros dispositivos escravos. A Figura 39 apresenta o módulo de acionamento do display matricial LCD I2C, com sua pinagem e seus principais componentes.

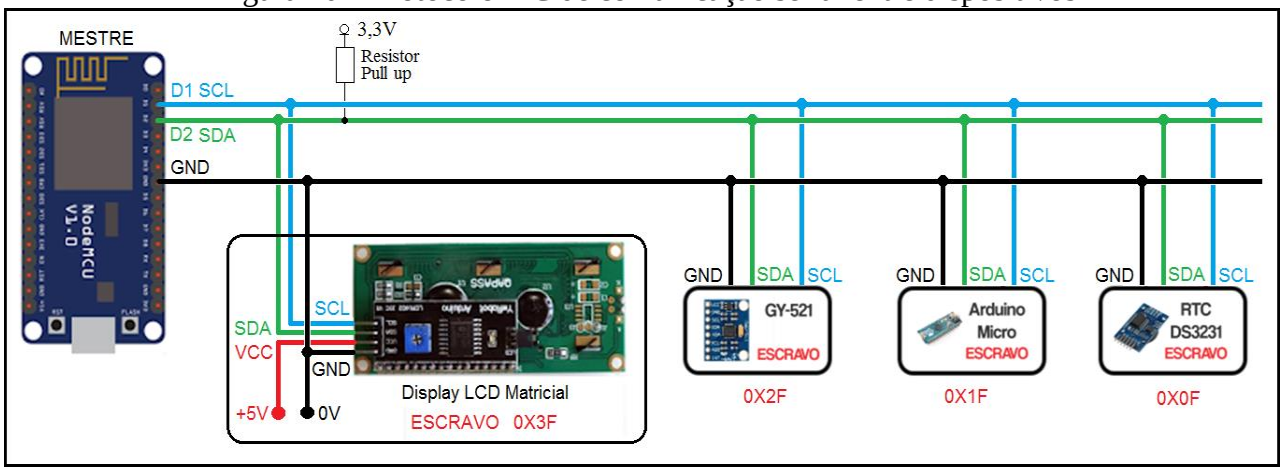
Figura 39 – Módulo de acionamento de *display* matricial LCD com protocolo I2C



Fonte: Arduino e Cia (2014)

O protocolo I2C é baseado na interação entre os elementos seguindo uma hierarquia mestre/escravo. Possui vários dispositivos se comunicando, todos conectados no mesmo barramento de dados, sendo um deles o mestre que solicita e envia informações aos escravos, que se limitam a receber informações e responde-las, caso solicitados. Cada dispositivo deve ter um endereço físico diferente, para não haver conflito de informações. Durante a comunicação, os pinos SDA (*Serial Data*) de todos dispositivos são ligados em conjunto, sendo o dispositivo mestre o responsável por gerenciar as informações. Todas as informações devem passar pelo mestre. Um escravo não interage diretamente com outro escravo. O pino SCL (*Serial Clock*) também é interligado de forma comum entre todos os dispositivos, sendo responsável por orquestrar a sequência de dados no barramento serial (MADEIRA, 2021). A Figura 40 apresenta o diagrama de um arranjo mestre/escravo do protocolo I2C.

Figura 40 – Protocolo I2C de comunicação serial entre dispositivos



Fonte: Autor (2021)

No módulo ESP8266 NodeMCU, os pinos de SDA e SCL são respectivamente os pinos D2 e D1, que se interligam a todos os dispositivos escravos, que possuem endereços individuais.

Em termos de *software*, será necessário o uso de duas bibliotecas para o acesso ao *display* matricial em I2C:

a) *#include <Wire.h>*, nativa no IDE do Arduino, bastando invoca-la, responsável pelo protocolo I2C;
b) *#include <LiquidCristal_I2C.h>*, que deve ser instalada, responsável pelo acesso ao display LCD matricial através de protocolo I2C.

Para instalar a biblioteca LiquidCristal_I2C.h é necessário realizar o seguinte procedimento (ARDUINO E CIA, 2014):

1. No IDE Arduino, abra o gerenciador de bibliotecas: “Sketch”, “Incluir Biblioteca” e “Gerenciar Bibliotecas”;
2. Na janela do Gerenciador de Bibliotecas, faça uma busca por “liquidcrystal_I2c” e instale a biblioteca “LiquidCrystal I2C by Frank de Brabander”;
3. Clicar sobre o botão INSTALAR.

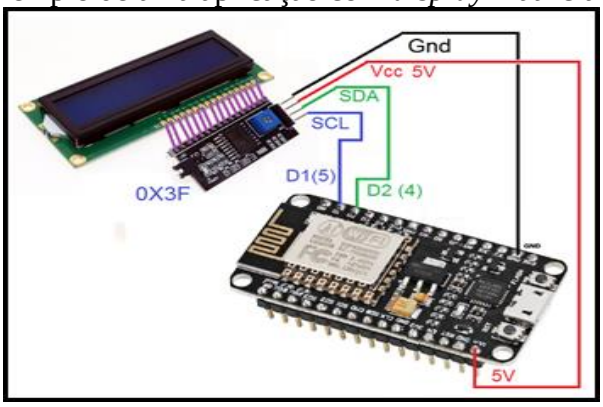
A Figura 41 apresenta o processo de instalação desta biblioteca.

Figura 41- Instalação da biblioteca LiquidCrystal_I2C.h

Fonte: Autor (2021)

O seguinte diagrama, na Figura 42, irá exibir duas mensagens no *display*, sendo que a luz de *backlight* irá piscar a cada segundo.

Figura 42 – Diagrama exemplo de uma aplicação com *display* matricial LCD em protocolo I2C



Fonte: Autor (2021)

Código e circuito para exibição de mensagens no display matricial LCD com protocolo I2C.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Ligação da I2C no ESP8266  SCL -> 5(D1) - SDA -> 4(D2)
LiquidCrystal_I2C lcd(0x3F,16,2);//(usar codigo scan para descobrir)
void setup(){
  lcd.init();
}
void loop(){
  lcd.setBacklight(HIGH); //acende o backlight do LCD
  lcd.setCursor(0,0);//posiciona o cursor para escrita (coluna, linha)
  lcd.print("ESP8266 NodeMCU ");
  lcd.setCursor(0,1);
  lcd.print("by Joao Peixoto ");
  delay(500);
  lcd.setBacklight(LOW);//desliga o backlight do LCD
  delay(500);
}
```

Neste código se destacam as seguintes linhas que chamam as principais funções da biblioteca LiquidCrystal_I2C, exposta no Quadro 16.

Quadro 16 – Principais funções da biblioteca LiquidCrystal_I2C

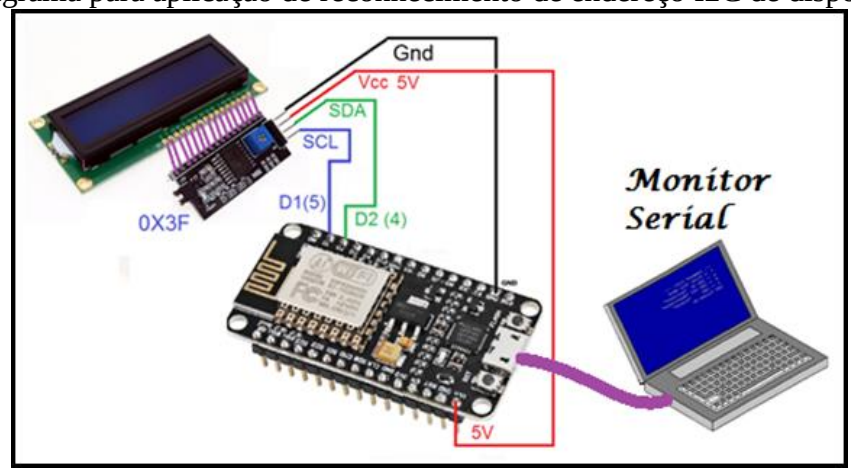
Código	**Detalhamento**
LiquidCrystal_I2C lcd(<endereço>, <n° colunas>,<n° linhas>);	Cria um objeto **lcd** do tipo LiquidCrystal_I2C, passando os parâmetros: <endereço>: endereço do escravo em hexadecimal (ex; 0X3F); <n° colunas>: número de colunas que o *display* matricial LCD possui; <n° linhas>: número de linhas que o *display* matricial LCD possui.
lcd.init();	Inicializa o display.
lcd.setBacklight(estado);	Liga e desliga a luz de fundo; **estado** = HIGH : Liga a luz de fundo; **estado** = LOW : Desliga a luz de fundo.
lcd.setCursor(coluna,linha);	Posiciona o cursor do display matricial LCD na <coluna> e <linha> indicada.
lcd.print(mensagem);	Exibe a **mensagem** na posição atual do cursor, deslocando-o para direita a cada caractere; Pode ser uma mensagem direta, estando esta entre aspas (exemplo: lcd.print(“by Joao Peixoto”); Pode ser um valor de uma variável (exemplo: int a = 12; lcd.print(a).

Fonte: Autor (2021)

Caso não se consiga descobrir fisicamente o endereço do dispositivo I2C, é possível descobrir através da execução de um programa que testa todos os endereços possíveis e apresenta no monitor serial o endereço identificado do dispositivo conectado. Importante que, se for identificar mais de um

componente, deve-se manter somente um escravo conectado por vez, para ter certeza da relação do endereço com o escravo físico. Um programa para fazer esse reconhecimento é apresentado na Figura 43.

Figura 43 – Diagrama para aplicação de reconhecimento de endereço I2C de dispositivos escravos



Fonte: Autor (2021)

O programa para reconhecimento de endereço I2C de dispositivos escravos é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)" - SCL 5 (D1)- SDA 4 (D2)
#include<Wire.h>  //biblioteca para I2C

void setup(){
  Serial.begin(9600);//inicializa canal serial em 9600bps
  while(!Serial){//aguarda a conexão serial ocorrer
  }
}
void loop(){
  Serial.println("Pesquisando Dispositivos.....");
  Wire.begin();//Inicializa protocolo I2C
  for(byte i=0; i<120; i++){//testando os endereços de 0 a 199
    Wire.beginTransmission(i);//tentativa de trasmitir para endereço "i"
    if(Wire.endTransmission() == 0){//se conseguiu, retorna 0
      Serial.print("Encontrou I2C em 0x");
      Serial.println(i,HEX);//imprime o valor de "i" em hexadecimal
      delay(5);
    }
  }
  delay(1000);
}
```

Como aplicação, se propõe o projeto 6, que monitora a temperatura ambiente, apresentando os resultados em um *display* matricial LCD, com protocolo I2C.

PROJETO 6 – MONITOR DE TEMPERATURA AMBIENTE

"O projeto consiste em display matricial LCD conectado ao módulo ESP8266 NodeMCU por protocolo I2C, que monitora um sensor de temperatura LM35 conectado à entrada analógica A0, exibindo no display o valor da temperatura ambiente em Graus Celsius."

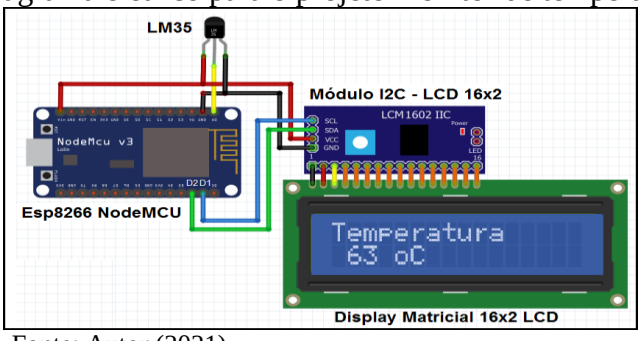
Como sensor de medição de temperatura, esse projeto utiliza o circuito integrado LM35[14], que fornece em sua saída um valor de tensão de 10mV/°C. Isso significa que uma temperatura de 0°C provoca um sinal de 0V, já uma temperatura de 100°C provoca um sinal de 1V, de forma linear. Como a entrada analógica do Módulo ESP8266 NodeMCU recebe valores de 0 a 3,3V, em uma conversão de 10 bits (0 a 1.023), então uma variação de 0V a 1V na entrada analógica A0 representa uma palavra digital de 0 a 310.

A função *map(analogRead(A0),0,310,0,100)* traduz o sinal lido na entrada analógica A0 de uma variação de 0 a 310, para uma variação de 0 a 100, correspondendo a 0°C e a 100°C, respectivamente.

Como os valores de temperatura poderão ser unidades, dezenas ou centenas, isto implica na impressão de 1, 2 ou 3 caracteres em sequência. Mas o display matricial LCD mantém o último caractere impresso sendo exibido na tela. Assim, a função *ajustaPosicao(int Temp)* monitora o número de dígitos que o valor da temperatura possui, imprimindo espaços em branco onde não há impressão, apagando o caractere anterior. Assim, sempre haverá impressão de 3 caracteres. Se o valor for em unidades (0 a 9), ela imprime 2 espaços brancos e depois a unidade. Se o valor incluir as dezenas (10 a 99), imprime 1 espaço branco e depois a dezena. Se o valor incluir a centena, então imprime a centena.

A Figura 44 apresenta o diagrama elétrico para o projeto monitor de temperatura ambiente.

Figura 44 - Diagrama elétrico para o projeto monitor de temperatura ambiente



Fonte: Autor (2021)

[14] Sensor de Temperatura integrado, que uma vez alimentado com uma tensão de 4V a 30V, fornece em sua saída uma tensão de 10mV por grau Célsius de temperatura. Disponível em: https://www.ti.com/product/LM35.

O código para essa aplicação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define A0 0
int Temperatura = 0;
// Ligação da I2C no ESP8266  SCL -> 5(D1) - SDA -> 4(D2)
LiquidCrystal_I2C lcd(0x3F,16,2);//(usar codigo scan para descobrir)
void setup(){
  lcd.init();
  lcd.setBacklight(HIGH); //acende o backlight do LCD
  lcd.setCursor(0,0);//posiciona o cursor para escrita (coluna, linha)
  lcd.print("Temperatura:    ");
}
void ajustaPosicao(int Temp){
  if(Temp < 10){
    lcd.print("  ");
  }else{
    if(Temp<100){
      lcd.print(" ");
    }
  }
}
void loop(){
  Temperatura = map(analogRead(A0),0,310,0,100);
  lcd.setCursor(0,1);//coluna 0 - linha 1
  ajustaPosicao(Temperatura);
  lcd.print(Temperatura);
  lcd.setCursor(4,1);//coluna 4 - linha 1
  lcd.print("oC");
  delay(100);
}
```

4 PERIFÉRICOS

Os dispositivos periféricos são componentes que se juntam a um componente central para lhe conceder mais funcionalidades. Neste capítulo, serão estudados alguns dos dispositivos periféricos utilizados no módulo ESP8266 NodeMCU, dando-lhe mais funcionalidades do que as que já possui.

4.1 SENSORES

Sensores são componentes periféricos que detectam grandezas físicas do ambiente e as traduzem em um sinal elétrico, perceptível por um controlador (CITISYSTEMS, 2021). Segundo a natureza da grandeza a ser medida, um tipo de princípio físico-químico é utilizado. Da mesma forma, o sinal de saída dos sensores, tende a ser compatível com o controlador que o monitora. Dependendo da aplicação, os sensores podem ser mais ou menos complexos, pois cada foco de estudo tem sua gama de princípios e sensores distintos.

Considerando um foco de estudo na detecção de presença no ambiente, seja de produtos ou obstáculos, se destacam para esse fim os sensores indutivos, capacitivos, ópticos e ultrassônicos, a serem detalhados nos subcapítulos seguintes.

4.1.1 Sensor indutivo

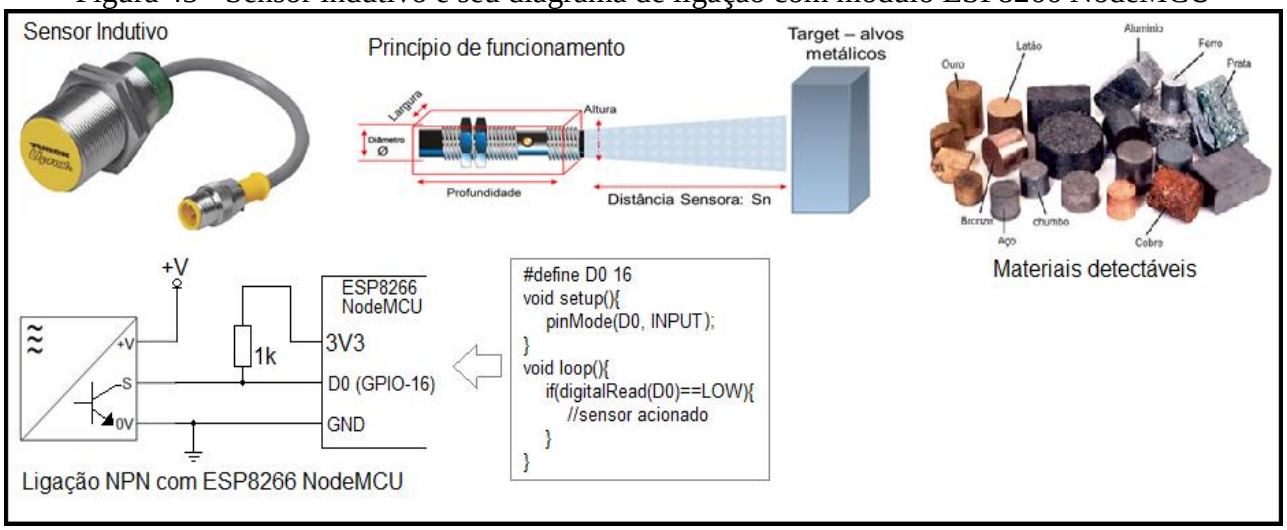
Seu princípio de funcionamento baseia-se na geração de um campo eletromagnético de alta frequência, que é desenvolvido por uma bobina instalada na face sensora do sensor. A bobina faz parte de um circuito oscilador que, em condição normal (sem a presença de objetos), gera um sinal. Quando um metal se aproxima do campo, absorve parde de sua energia, diminuindo a amplitude do sinal gerado no oscilador. A variação de amplitude deste sinal é convertida em um sinal de saída através do circuito interno do sensor. Esse circuito chaveia um transístor bipolar NPN[15] na sua saída, de forma que ao se colocar um resistor de *pull-up*, possa se ter um sinal binário, com nível lógico compatível com o controlador que o está monitorando, independentemente do nível de tensão que alimenta o sensor (+V). Deve-se se ter cuidado, pois esse recurso torna a lógica invertida. O sensor

[15] Transístor bipolar NPN é um dispositivo semicondutor de 3 terminais: coletor, emissor e base. Por ser na configuração NPN, significa que coletor e emissor são semicondutores do tipo N e a base são semicondutores do tipo P. Este dispositivo se comporta como uma chave eletrônica onde, ao circular uma corrente elétrica do terminal de base para o terminal de emissor, possibilita uma circulação de corrente elétrica entre os terminais de coletor e emissor, em uma razão proporcional de ganho, peculiar a cada modelo de transístor. Fonte: https://www.embarcados.com.br/transistor-pnp-canal-eletronica-facil/.

inativo significa um nível lógico alto (HIGH) na entrada do controlador, enquanto o sensor detectando algo metálico significa um nível lógico baixo (LOW) na entrada do controlador.

A leitura do sensor se dá pela leitura do estado lógico da entrada do controlador. No caso do módulo ESP8266 NodeMCU é utilizada a função *digitalRead(GPIO).* A Figura 45 apresenta os aspectos relativos a um sensor indutivo e seu diagrama de ligação com o módulo.

Figura 45 - Sensor indutivo e seu diagrama de ligação com módulo ESP8266 NodeMCU



Fonte: Adaptado de Citisystems (2021)

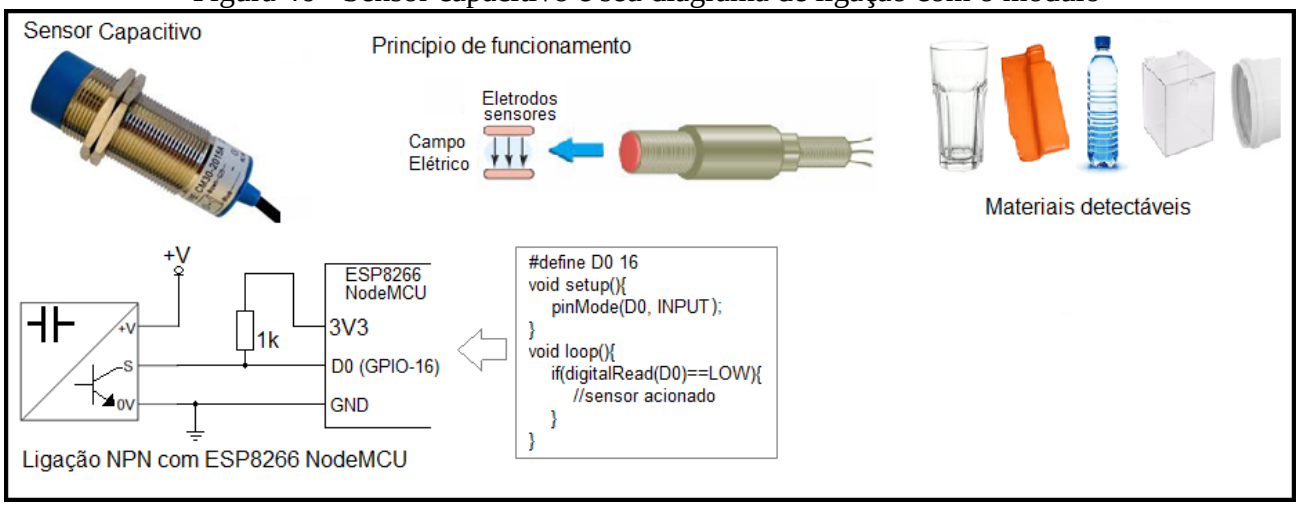
4.1.2 Sensor capacitivo

Os sensores capacitivos são equipamentos eletrônicos capazes de detectar a proximidade de alvos sólidos, líquidos ou pastosos, independentemente de serem de metal ou não. A detecção ocorre pela interferência que o material a ser sensoreado provoca no campo elétrico de dois eletrodos, situados na parte frontal do sensor. A detecção acontece sem que ocorra contato físico entre o alvo e o sensor, aumentando assim a vida útil do sensor, por não haver desgaste mecânico. Por detectar uma variação no campo elétrico gerado (isto depende de cada tipo de material), é comum haver um parafuso de ajuste (*trimpot*) junto ao seu corpo, para calibração de sensibilidade do sensor.

Assim como no sensor indutivo, o sensor capacitivo também possui saída com transístor de chaveamento, de forma que ao se colocar um resistor de *pull-up*, possa se ter um sinal binário, com nível lógico compatível com o controlador que o está monitorando, independentemente do nível de tensão que alimenta o sensor (+V).

A leitura do sensor se dá pela leitura do estado lógico da entrada do controlador. No caso do módulo ESP8266 NodeMCU é utilizada a função *digitalRead(GPIO).* A Figura 46 apresenta os aspectos relativos a um sensor capacitivo e seu diagrama de ligação com o módulo.

Figura 46 - Sensor capacitivo e seu diagrama de ligação com o módulo



Fonte: Adaptado de Citisystems (2021)

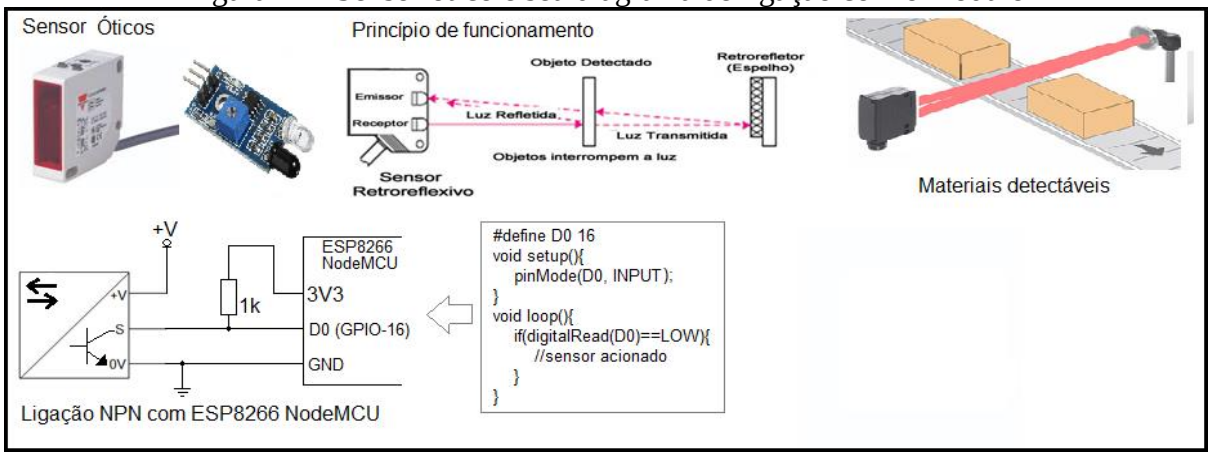
4.1.3 Sensor óptico

Os sensores ópticos usam a luz para detectar objetos ou para medir a distância em que um objeto se encontra. A luz pode ser visível (LED vermelho ou laser vermelho) ou invisível (luz infravermelha) ao olho humano. Como os sensores ópticos baseiam-se na transmissão e recepção de luz, essa luz pode ser refletida ou interrompida por um objeto a ser detectado. Assim, eles podem detectar qualquer tipo de material, seja metálico, condutivo ou poroso.

Os sensores ópticos são compostos por dois circuitos: um responsável pela emissão do feixe de luz, denominado transmissor e outro responsável pela recepção do feixe de luz, denominado receptor. Dependendo do tipo de sensor, o transmissor e o receptor podem se localizar no mesmo corpo ou em corpos separados.

A leitura do sensor se dá pela leitura do estado lógico da entrada do controlador. No caso do módulo ESP8266 NodeMCU é utilizado a função *digitalRead(GPIO).* A Figura 47 apresenta os aspectos relativos a um sensor óptico e seu diagrama de ligação com o módulo.

Figura 47 - Sensor ótico e seu diagrama de ligação com o módulo



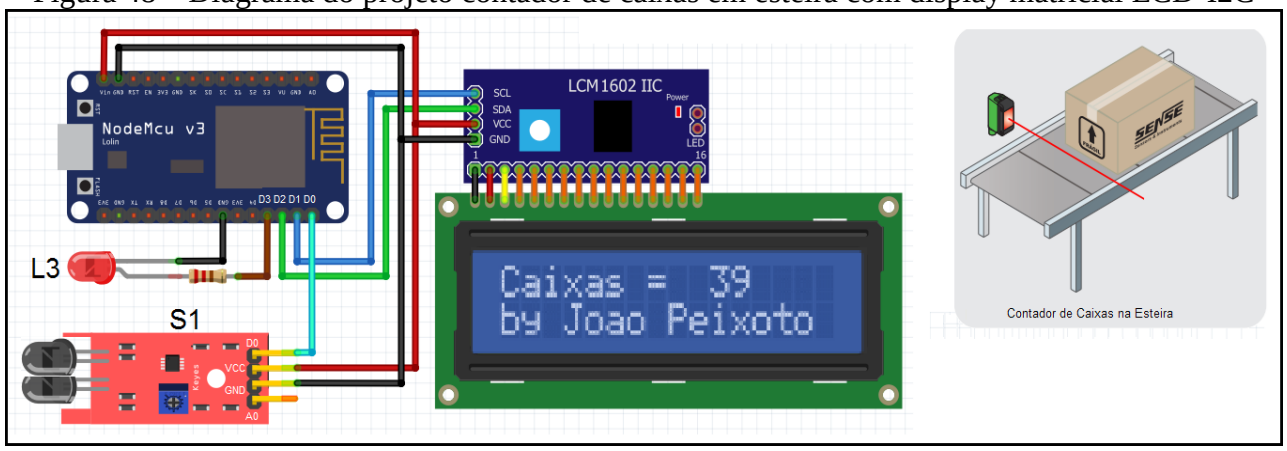
Fonte: Autor (2021)

PROJETO 7 – CONTADOR DE CAIXAS EM ESTEIRA COM DISPLAY MATRICIAL LCD-I2C

"O projeto consiste em sensor óptico que envia sinal para o módulo ESP8266 NodeMCU, indicando a presença de peça de uma esteira transportadora, procedendo a contagem das peças e apresentando o resultado em um display matricial LCD com protocolo I2C.

O diagrama para o projeto do contador de caixas em esteira é o apresentado na Figura 48. Nela consta um sensor óptico S1 que irá identificar as peças que passarem a sua frente na esteira.

Figura 48 – Diagrama do projeto contador de caixas em esteira com display matricial LCD-I2C



Fonte: Autor (2021)

O código para essa aplicação é o seguinte.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define CH 16 //D0 GPIO-16
#define LED 0 //D3 GPIO-0
long TempoAnt;
int Caixas = 0;
bool fLED = LOW, fCHant = LOW;
LiquidCrystal_I2C lcd(0x3F,16,2);//(usar codigo scan para descobrir)
void setup(){
  lcd.init();
  lcd.setBacklight(HIGH);
  lcd.print("Caixas =        ");
  lcd.setCursor(0,1);
  lcd.print("by Joao Peixoto ");
  pinMode(CH, INPUT);
  pinMode(LED, OUTPUT);
}
void ajustaPosicao(int num){
  if(num<10){
    lcd.print("   ");
  }else{
    if(num<100){
      lcd.print("  ");
    }else{
      if(num<10){
        lcd.print(" ");
      }
    }
  }
}
void loop(){
```

```
 if(digitalRead(CH)==HIGH){
  delay(50);
  if(digitalRead(CH)==HIGH){
   if(fCHant==LOW){
    fCHant= HIGH;
    Caixas++;
    lcd.setCursor(8,0);
    ajustaPosicao(Caixas);
    lcd.print(Caixas);
   }
  }
 }
 if(digitalRead(CH)==LOW){
  fCHant= LOW;
 }
 if(millis()>TempoAnt+500){
  if(fLED){
   digitalWrite(LED, LOW);
   fLED = LOW;
  }else{
   digitalWrite(LED, HIGH);
   fLED = HIGH;
  }
  TempoAnt=millis();
 }
}
```

4.1.4 Sensor por ultrassom

Sensores ultrassônicos são sensores que usam ondas sonoras para detectar um objeto. Seu princípio de funcionamento consiste em emitir ondas sonoras que são refletidas quando atingem um objeto, voltando ao sensor. O tempo que a onda sonora demora para refletir no objeto e retornar ao sensor é calculado e utilizado para determinar a distância do objeto. Dessa forma, os sensores ultrassônicos podem detectar a distância em que um objeto se encontra e não apenas sua presença (THOMSEN, 2011).

O som é classificado em grupos de categorias diferentes, dependendo de frequências:

a) Infrassom, menor que 16Hz;

b) Som audível, entre 16Hz e 20kHz;

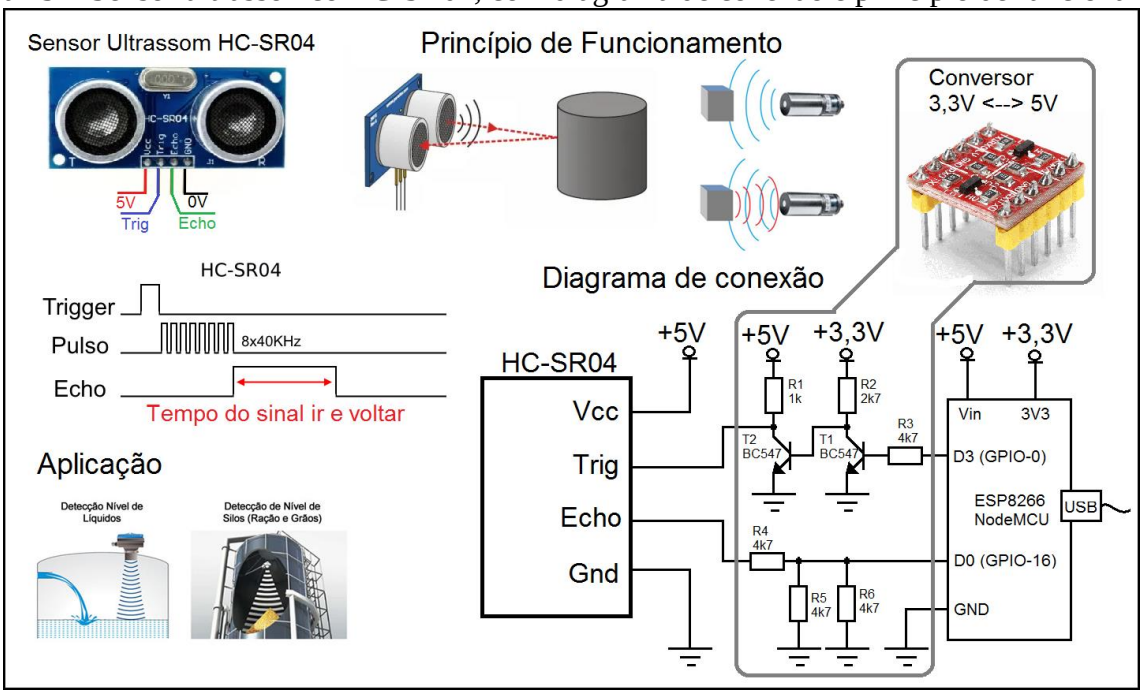
c) Ultrassom, entre 20 kHz e 1GHz;

d) Hipersom, maior que 1 GHz.

Os sensores ultrassônicos operam na faixa entre 125 kHz a 400 kHz. Isso explica o porquê do som não ser audível.

Um sensor ultrassônico muito utilizado em aplicações com microntroladores é HC-SR04[16], apresentado na Figura 49 junto com seu princípio de funcionamento, aplicação, diagrama de conexão e código para sua leitura.

Figura 49 – Sensor ultrassônico HC-SR04, com diagrama de conexão e princípio de funcionamento



Fonte: adaptado de Citisystems (2021)

O código para essa aplicação de medição de distância é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define Echo 16 //D0 GPIO-16
#define Trig 0 //D3 GPIO-0
long Distancia;

void setup(){
  Serial.begin(9600);
  pinMode(Echo, INPUT);
  pinMode(Trig, OUTPUT);
}
long medirDistancia(){
  long dist;
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);//aguarda 2us
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);//aguarda 10us
  digitalWrite(Trig, LOW);
  dist = 0.01723 * pulseIn(Echo, HIGH);
  return dist;
}
void loop(){
  Distancia = medirDistancia();
  Serial.println(" ");
  Serial.print("Distancia: ");
```

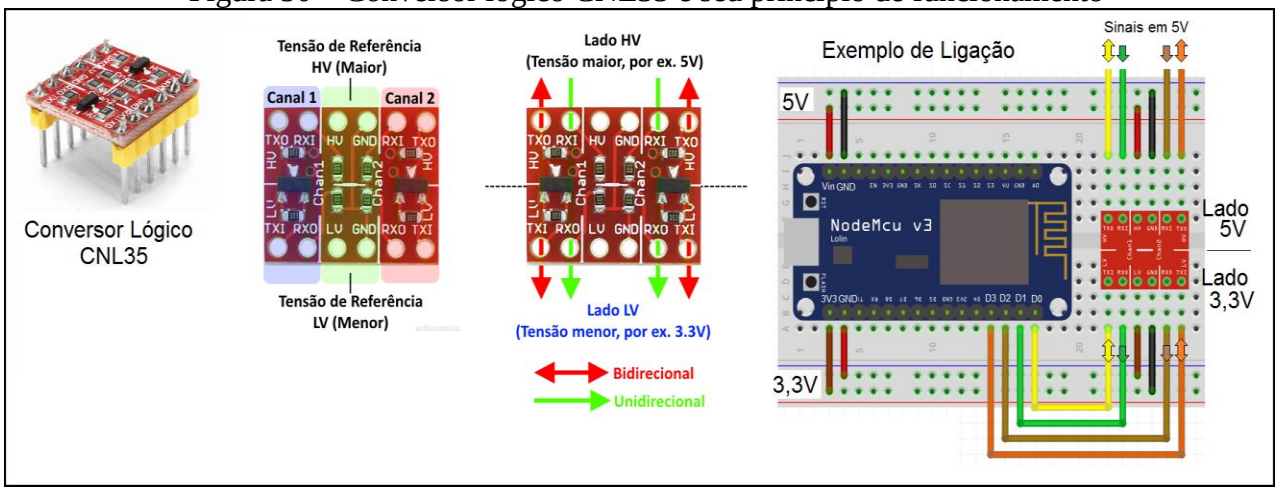
[16] Sensor Ultrassônico HC-SR04, alimentação de 5V, mede distâncias entre 2cm e 4m, com precisão de 3mm. *Datasheet* disponível em: https://datasheetspdf.com/pdf-file/1380136/ETC/HC-SR04/1.

```
  Serial.print(Distancia);
  Serial.println(" cm");
  delay(1000);
}
```

O funcionamento do sensor ultrassom HC-SR04 parte da emissão de um sinal em nível alto de “Trig”, com duração de 10µs. A partir de então, o sensor emite um sinal de ultrassom no seu emissor, repetido 8 vezes em 40kHz, monitorando o tempo para que esse sinal retorne ao receptor do sensor. Ao definir o tempo decorrido entre o envio do sinal ultrassônico e sua recepção de retorno, o sensor aciona um nível alto em seu pino de “Echo”, com duração igual ao tempo que levou para o sinal ultrassônico sair e retornar ao sensor. A função do módulo ESP8266 NodeMCU é provocar o sinal de “Trig” e monitorar o tempo de duração do sinal de “Echo”. Como esse tempo terá uma proporção direta com a distância do objeto que está refletindo o sinal ultrassônico e a velocidade deste sinal (velocidade do som), um tratamento matemático define o valor da distância entre o sensor e o objeto que o está refletindo.

Fato é que o módulo ESP8266 NodeMCU opera com tensão de 3,3V, ao passo que o sensor ultrassom HC-SR04 opera com 5V. Para interligar esses dois sinais é necessária a implementação de um *driver* para esse propósito, como o diagrama da figura. Ou então, utilizar um conversor bidirecional 3V3-5V comercial, já projetado para esse propósito, demonstrado na Figura 50 (ARDUINO E CIA, 2015).

Figura 50 – Conversor lógico CNL35 e seu princípio de funcionamento



Fonte: Adaptado de Arduino e Cia (2015)

O conversor lógico CNL35 possui dois canais de conversão unidirecional (RX), onde o fluxo de sinal só pode ocorrer do lado maior de tensão (RXI) para o lado menor de tensão (RXO). Isso porque o circuito que o compõe é um divisor de tensão resistivo. Possui também dois canais de conversão bidirecional (TX), onde o fluxo de sinal pode ir da tensão maior para a menor ou vice-

versa, pois o circuito que o compõe é um *driver* transistorizado tipo MOSFET[17], que permite fluxo nos dois sentidos.

Para efetuar a medição do sensor ultrassônico há várias bibliotecas disponíveis em canais especializados na internet. Mas neste exemplo, é mostrada uma função implementada para provocar o sinal de "Trig" e monitorar o tempo de permanência em nível alto do sinal de "Echo". Isso é realizado pela função *pulseIn(pino, estado_lógico)*, onde uma vez passados os parâmetros do número do **pino** GPIO e o **estado lógico** a ser monitorado (LOW ou HIGH), retorna um valor do tipo *long* com o tempo em microssegundos que o pino ficou no estado_lógico (ARDUINO, 2021).

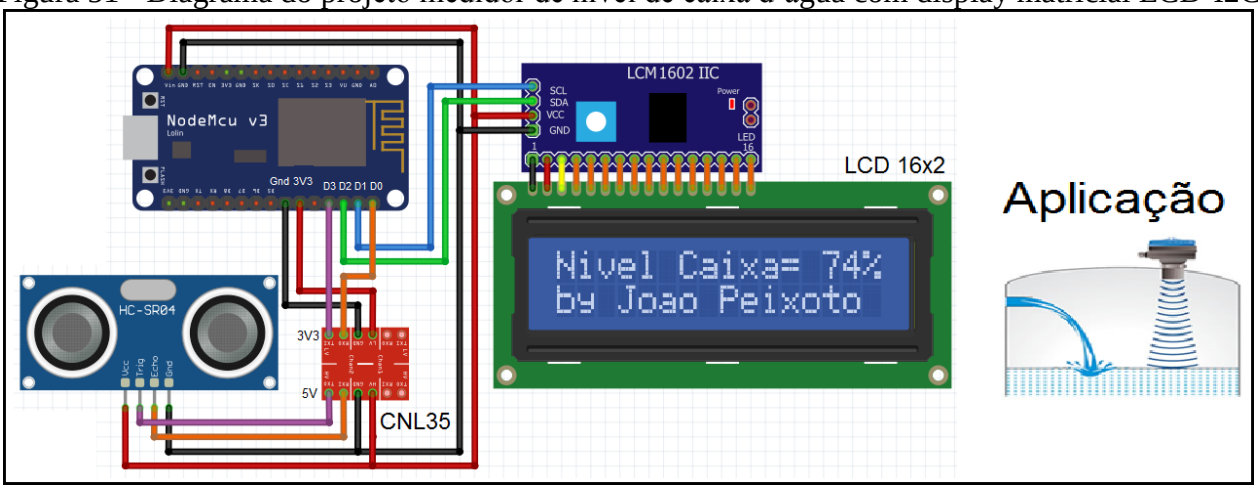
[17] MOSFET, transistor de efeito de campo, que permite a circulação de corrente elétrica em um canal interno, cujo fluxo é controlado por uma tensão de gate, que atua sobre esse canal. Fonte: https://www.newtoncbraga.com.br/index.php/como-funciona/6417-art977.

PROJETO 8 – MEDIDOR DE NÍVEL DE CAIXA D'ÁGUA COM DISPLAY MATRICIAL LCD-I2C

"O projeto consiste em um sensor ultrassônico medindo a distância da sua posição em relação ao nível de água de uma caixa d'água, através do módulo ESP8266 NodeMCU, indicando o nível em percentual e apresentando o resultado em um display matricial LCD com protocolo I2C.

A Figura 51 apresenta o diagrama para o projeto medidor de nivel de caixa d'água com display matricial LCD, no protocolo I2C. Nele se observa o sensor de ultrassom, que deve ser instalado na parte superior do reservatório, sendo que sua medição ocorre pela medição de distância entre o sensor e a lâmina superficial da água. Por esta razão, é necessário calibrar no local de instalação os valores de distância medidos quando o reservatório está cheio e quando está vazio, pois isto muda para cada aplicação.

Figura 51 - Diagrama do projeto medidor de nivel de caixa d'água com display matricial LCD-I2C



Fonte: Autor(2021)

O código para essa aplicação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define Echo 16 //D0 GPIO-16
#define Trig 0 //D3 GPIO-0
long Distancia;
int dPercentual;
int NivelMinimo = 1700;//calibrar no local em mm
int NivelMaximo = 200;//calibrar no local em mm
// Ligação da I2C no ESP8266  SCL -> 5(D1) - SDA -> 4(D2)
LiquidCrystal_I2C lcd(0x3F,16,2);//(usar codigo scan para descobrir)
void setup(){
  lcd.init();
  lcd.setBacklight(HIGH);
  lcd.print("Nivel Caixa=   %");
  lcd.setCursor(0,1);
  lcd.print("by Joao Peixoto ");
  pinMode(Echo, INPUT);
  pinMode(Trig, OUTPUT);
}
```

```
long medirDistancia(){
  long dist;
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);//aguarda 2us
  digitalWrite(Trig, HIGH);
  delayMicroseconds(10);//aguarda 10us
  digitalWrite(Trig, LOW);
  dist = 10*0.01723 * pulseIn(Echo, HIGH);
  return dist;
}
void ajustaPosicao(int num){
  if(num<10){
    lcd.print("  ");//2 espaços
  }else{
    if(num<100){
      lcd.print(" ");//1 espaço
    }
  }
}
void loop(){
  Distancia = medirDistancia();
  dPercentual= map(Distancia,NivelMaximo,NivelMinimo,0,100);
  lcd.setCursor(9,0);
  ajustaPosicao(dPercentual);
  lcd.print(dPercentual);
  delay(500);
}
```

4.2 RELÉ ELETROMAGNÉTICO

O relé eletromagnético é um componente elétrico que permite o acionamento de cargas em um patamar de tensão elétrica e corrente elétrica diferentes do utilizado para seu comando. Como exemplo, esse componente pode acionar uma carga de 220V/5A em corrente alternada, a partir de um sinal de comando de 12V em corrente contínua. Isso provoca um grande ganho em termos de capacidade de comandar cargas de alta potência, mas traz o inconveniente de que o tempo de comutação de contato é lento em relação ao acionamento da saída do microcontrolador que o comanda, razão pela qual deve ser utilizado somente em casos onde a carga não necessite de comandos rápidos.

O princípio de funcionamento de um relé eletromagnético consiste em uma bobina eletromagnética que recebe um sinal de comando. Essa bobina gera um campo magnético a partir da corrente elétrica que nela circula. Esse campo magnético irá atrair um jogo de contatos elétricos, fazendo-o comutar seus contatos, o que era fechado passa a ficar aberto, o que era aberto passa a ficar fechado. E, a partir destes contatos, ocorre a circulação de corrente elétrica para a carga, de forma isolada do sinal que comanda a bobina do relé (BRAGA, 2012).

Um relé eletromagnético é composto de uma bobina com terminais A1 e A2, conjunto de contatos normalmente abertos (NA) e contatos normalmente fechados (NF). A numeração dos contatos se dá por um número na forma de dezena, onde o algarismo da unidade representa o tipo de contato (1-2 para fechado, 3-4 para aberto, 1-2-4 para comutadores) e o algarismo da dezena representa o número do contato. Um relé pode ter 1 ou mais conjuntos de contatos.

Em relação aos parâmetros de um relé eletromagnético, é importante observar:

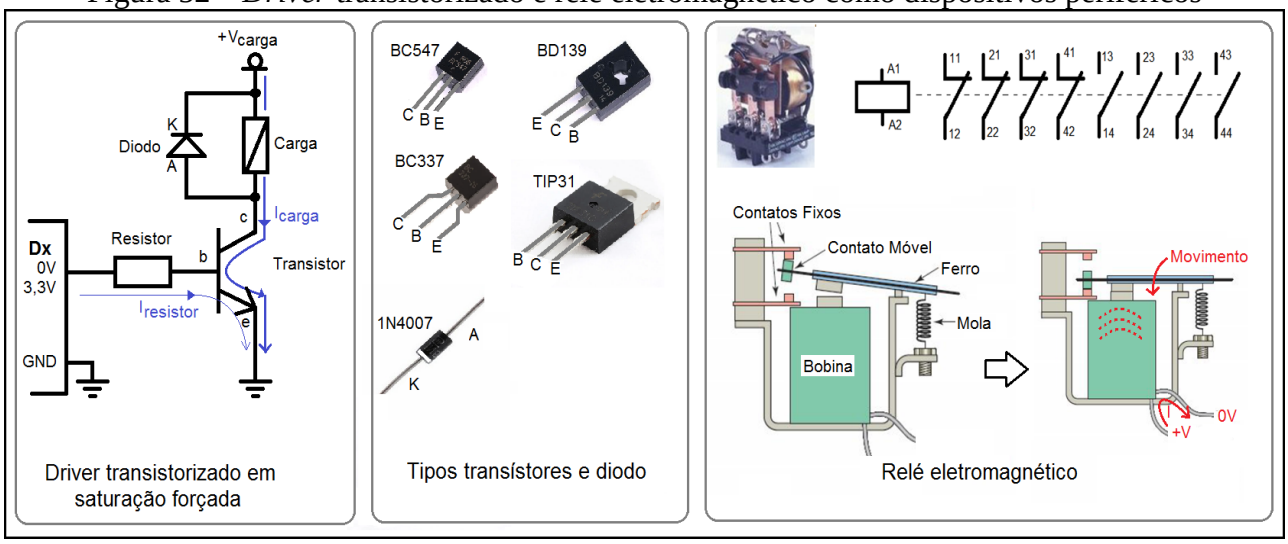
a) Tipo e magnitude da tensão da bobina: exemplo, 12Vcc (cc = em corrente contínua);
b) Tipo dos contatos: exemplo, normalmente abertos, normalmente fechados e/ou comutadores;
c) Máxima corrente que os contatos podem suportar: exemplo, 5A;
d) Tensão de isolação entre os contatos: exemplo, 250Vca (ca = em corrente alternada).

Esse componente resolve o problema de ter que comandar cargas com tipos e potências elevadas a partir de sinais de comando menores. Em alguns casos, há relés eletromagnéticos que possuem bobinas com sinal de 5Vcc, com correntes de cerca de 20mA, o que permite liga-los diretamente na porta de saída de controladores existentes. Porém, é difícil encontrar um relé eletromagnético para ser acionado por uma tensão de 3,3V, que é a tensão de saída do módulo ESP8266 NodeMCU. Nesses casos é importante a utilização de um circuito de *driver* transistorizado.

Um *driver* é um circuito que converte um patamar de tensão/corrente em outro. É muito utilizado em saídas de circuitos microprocessados em que o objetivo da porta de saída é fornecer um estado lógico, não um comando em potência.

Há vários modelos de *driver*, concebidos segundo a aplicação necessária. Para o comando de um relé eletromagnético, será utilizado um *driver* transistorizado, com objetivo de converter a tensão de 3,3V da porta de saída do módulo ESP8266 NodeMCU em um sinal de 12V para comandar a bobina do relé eletromagnético. A Figura 52 apresenta o *driver* transistorizado, os modelos de transístores bipolares e o relé eletromagnético com sua simbologia e princípio de funcionamento.

Figura 52 – *Driver* transistorizado e relé eletromagnético como dispositivos periféricos



Fonte: Autor (2021)

O *driver* transistorizado utiliza um transístor bipolar na configuração emissor comum. Nessa aplicação se está utilizando um transístor bipolar tipo NPN, como aponta o símbolo.

Um transístor bipolar possui 3 terminais, sendo eles: base (b), coletor (c) e emissor (e). Na configuração de emissor comum, o transístor bipolar, ao receber uma corrente circulante entre base e emissor, provocará uma corrente circulante entre coletor e emissor, sendo esta multiplicada por um fator de ganho (h_{fe}) típico do transístor, respeitando a seguinte Equação 2.

$$I_c = I_b . h_{fe} \quad (2)$$

A faixa de ganho de um transístor é um parâmetro que varia de modelo para modelo e, a partir destes parâmetros, é que se define a polarização do transístor. Neste *driver* será utilizado a técnica de saturação forçada, para polarizar o transístor. Esta técnica consiste em considerar o ganho do transístor tão somente igual a 10. Isso implica que o transístor ou estará cortado (sem corrente no coletor) ou estará saturado (com máximo de corrente permitida no coletor).

Para esse *driver* são necessárias 2 escolhas: o tipo (modelo) do transístor e o valor ôhmico do resistor.

O tipo de transístor se dará por dois parâmetros: Corrente máxima permissível entre coletor e emissor e tensão máxima suportada entre coletor e emissor.

Como o transístor bipolar irá comandar a bobina do relé eletromagnético, deve-se conhecer a corrente e a tensão que a bobina precisa, definido esses dois parâmetros.

No mercado, existe uma quantidade grande de transístores bipolares disponíveis para aquisição. Aqui se destacam os 4 modelos, que já devem atender a maioria dos casos simples de projetos com microcontroladores: BC547[18], BC337[19], BD139[20] e TIP31[21].

Para dimensionar o valor do resistor conectado entre o microcontrolador e a base do transístor, são necessárias algumas considerações: a corrente de carga é a mesma corrente de coletor; a corrente de base é a mesma corrente do resistor; a tensão entre base e emissor, quando acionado, é uma constante no valor de 0,7V; a soma da tensão no resistor e a tensão entre base e emissor resulta na tensão aplicada pelo microcontrolador (3,3V); a corrente da bobina do relé será de 15mA.

Definidas essas considerações, então o valor ôhmico do resistor será calculado pela Equação 3:

$$R = \frac{V_{aplicada} - V_{be}}{\left({}^{I_{carga}}/_{10}\right)} = \frac{3{,}3 - 0{,}7}{\left({}^{0{,}015}/_{10}\right)} = 1733\Omega \quad \therefore 1k8\Omega \qquad (3)$$

Como não existe um valor comercial de resistor em 1.733Ω, atribui-se o valor imediatamente superior.

No caso de utilização de cargas indutivas, que possuem bobinas geradoras de campos eletromagnéticos, pelo fato de gerarem picos de tensão invertida no momento do seu desligamento, faz-se necessário instalar um diodo retificador em anti-paralelo à carga. Essa configuração, também conhecida como "roda-livre", garante a extinção do pico reverso quando do desligamento da bobina, por permitir a recirculação da corrente.

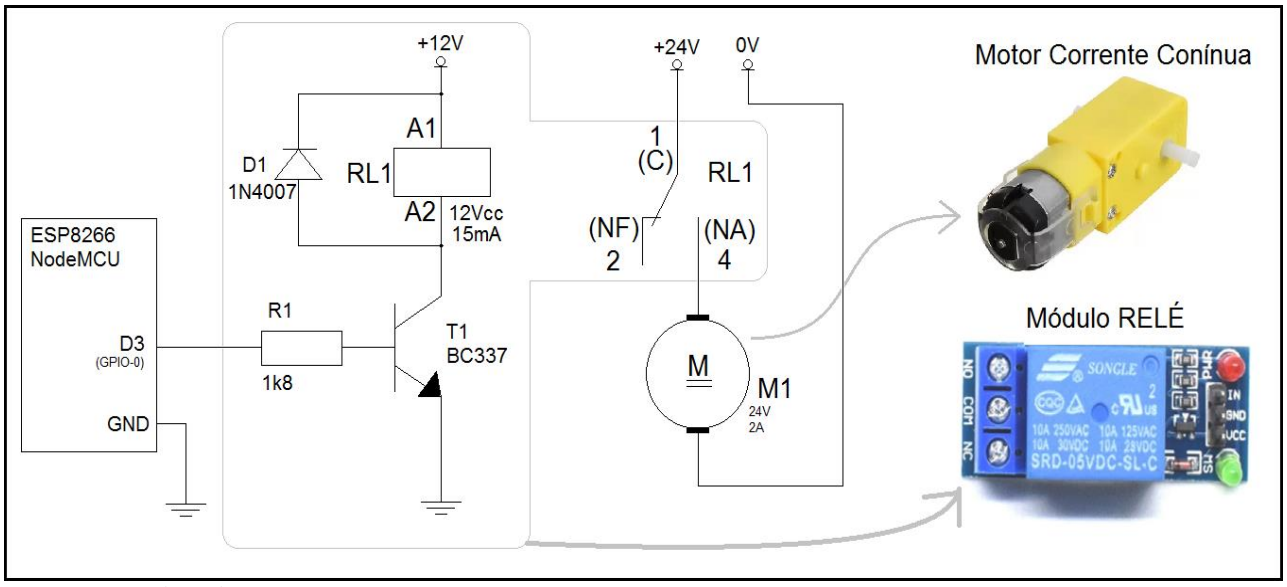
Um diodo retificador é um componente eletrônico com dois terminais: um anodo (A) e um Catodo (K). Sua função é permitir que a corrente circule do anodo para o catodo, caso a tensão elétrica no anodo seja maior do que a tensão elétrica do catodo. Caso contrário, se a tensão elétrica do catodo

[18] BC547 é um transístor bipolar NPN, com Vce=65V, Ic=100mA e h_{fe}=110. *Datasheet* disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/548887/FAIRCHILD/BC547.html.
[19] BC337 é um transístor bipolar NPN, com Vce=45V, Ic=800mA e h_{fe}=100. *Datasheet* disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/171972/ONSEMI/BC337.html.
[20] BD139 é um transístor bipolar NPN, com Vce=80V, Ic=3A e h_{fe}=25. *Datasheet* disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/351702/ONSEMI/BD139.html.
[21] TIP31 é um transístor bipolar NPN, com Vce=40V, Ic=1,5A e h_{fe}=25. *Datasheet* disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/2776/MOSPEC/TIP31.html.

for maior do que a tensão elétrica do anodo, então ocorre o corte, impedindo a passagem de corrente elétrica. Os parâmetros para a escolha de um diodo retificador são: máxima corrente direta permissível entre anodo e catodo e a máxima tensão reversa suportada entre anodo e catodo. Como sugestão de projeto, indica-se o diodo retificador 1N4007[22], que possui parâmetros para suportar a corrente reversa que a bobina do relé irá produzir.

Como aplicação, segue a Figura 45, que mostra um motor de corrente contínua sendo ligado e desligado por um relé eletromagnético, acionado por um *driver* transistorizado, sendo comandado pelo módulo ESP8266 NodeMCU. A figura salienta que por vezes o *driver* é fornecido como um dispositivo periférico comercial, agregando o transístor, seu circuito e o relé eletromagnético.

Figura 53 – Exemplo de acionamento de motor de corrente contínua com *driver* transistorizado e relé.



Fonte: Autor (2021)

O projeto 9 traz um sistema de aquecimento controlado, do tipo *on-off* (liga-desliga), com objetivo de manter a temperatura em 40°C.

[22] 1N4007 é um diodo retificador, com $V_{reversa}$ = 700V e I_{direta} = 1A. *Datasheet* disponível em: https://pdf1.alldatasheet.com/datasheet-pdf/view/341139/SSC/1N4007.html.

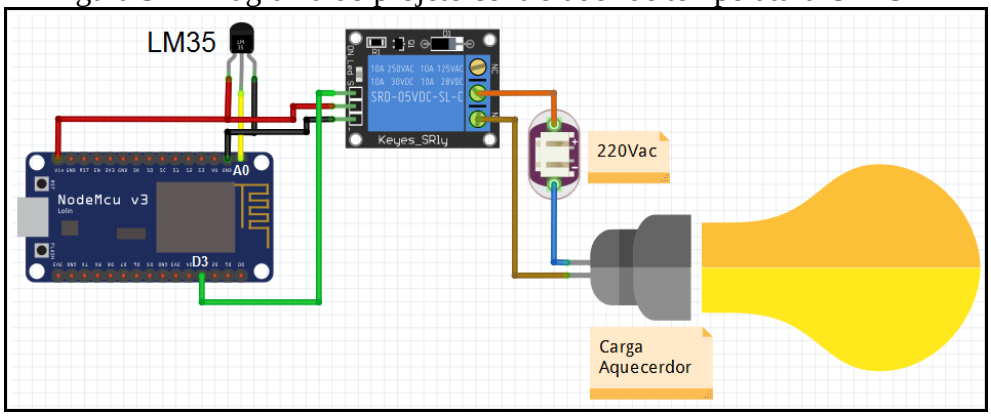
PROJETO 9 – CONTROLADOR DE TEMPERATURA ON-OFF

"O projeto consiste em um sensor de temperatura LM35 conectado à entrada analógica A0, que irá acionar um dispositivo de aquecimento quando a temperatura for inferior a 40°C"

Neste projeto se utilizou um módulo relé comercial, que já possui o *driver* transistorizado incorporado. A lâmpada, indicada no diagrama, representa qualquer tipo de carga que provoque aquecimento, tendo cuidado para que a corrente consumida pela carga não seja maior do que a capacidade de condução de corrente elétrica pelo contato do relé. Como sensor de medição de temperatura, o projeto utiliza o circuito integrado LM35[23] que fornece, em sua saída, um valor de tensão de 10mV/°C. E, para monitorar a temperatura medida, faz-se o uso do monitor serial, que exibirá o valor medido pelo sensor de temperatura, já convertido em Graus Célsius.

A Figura 54 apresenta o diagrama desse projeto.

Figura 54 - Diagrama do projeto controlador de temperatura ON-OFF



Fonte: Autor(2021)

O código de programa para essa aplicação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define A0 0
#define D3 0
int Temperatura = 0;
void setup(){
  Serial.begin(9600);
  pinMode(A0,INPUT);
  pinMode(D3,OUTPUT);
}
void loop(){
  Temperatura = map(analogRead(A0),0,310,0,100);
  Serial.println(Temperatura);
  if(Temperatura < 40){
    digitalWrite(D3, HIGH);
  }else{
    digitalWrite(D3, LOW);
```

[23] Sennsor de Temperatura integrado, que uma vez alimentado com uma tensão de 4V a 30V, fornece em sua saída uma tensão de 10mV por grau Célcius de temperatura. Disponível em: https://www.ti.com/product/LM35.

```
  }
  delay(100);
}
```

4.3 MOTOR CC COM PONTE H

Os acionamentos de motores de corrente contínua possuem uma característica peculiar quando é solicitada uma variação de velocidade. Ter uma velocidade mínima ou máxima não é problema, pois são duas condições favoráveis, uma vez que, se o motor estiver com a mínima velocidade, significa uma baixa corrente elétrica circulando através do dispositivo de acionamento. Assim como em uma máxima velocidade, toda tensão se concentrará sobre o motor e pouca restará sobre o dispositivo de acionamento. Em ambos os casos, ora o dispositivo de acionamento está com máxima tensão elétrica e mínima corrente elétrica circulando sobre si, ora está com mínima tensão elétrica e máxima corrente elétrica circulando sobre si. Nas duas formas a potência elétrica (produto entre tensão e corrente elétrica) será próximo de zero.

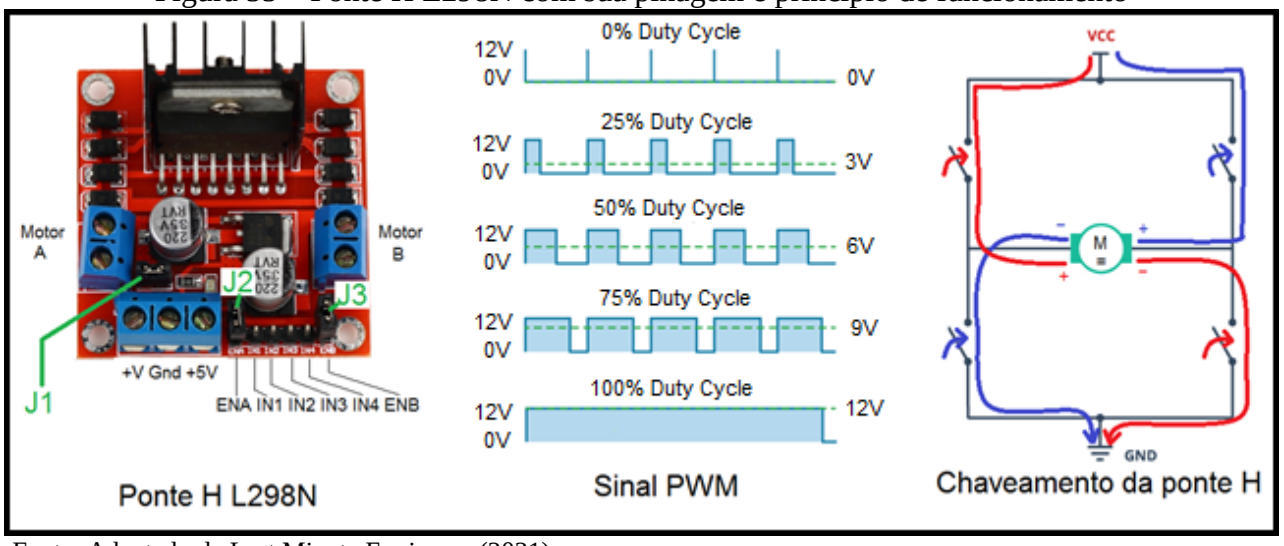
Porém, quando solicitada uma velocidade intermediária, o motor terá uma tensão elétrica intermediária, assim como uma corrente intermediária. O mesmo vale para o dispositivo de acionamento, que acabará tendo que dissipar uma potência elétrica resultante dessa corrente e tensão intermediárias. O que significa aquecimento.

Uma boa forma de evitar isto é a utilização de uma ponte H, que permite comandar o sentido da corrente elétrica a partir do conjunto de acionadores que são chaveados e controlar o a tensão resultante no motor, por trabalhar com sinais chaveados, por princípio PWM (*Pulse Width Modulation*).

A ponte H L298N é um componente bem comum e largamente empregado em comando de motores de corrente contínua, de pequeno porte. Ela possui uma entrada de alimentação entre os terminais +V e GND, que recebe uma tensão de 7V a 35V, que irá alimentar os motores A e B com uma corrente máxima de 2A. O *jumper* J1 permite que uma tensão de 5V interna seja distribuída aos *jumpers* J2, J3 e ao terminal de saída +5V. Os terminais ENA e ENB comandam os respectivos motores A e B, através de um sinal fixo de 5V com a ativação dos *jumpers* J2 e J3, promovendo velocidade máxima nos motores, ou com a inserção de sinal PWM (*jumpers* J2 e J3 desligados), obtendo assim uma velocidade relativa ao percentual *duty cycle* do sinal PWM.

A Figura 55 mostra uma ponte H L298N, com sua pinagem, seu sistema de chaveamento e seu sinal PWM para comando variável da velocidade do motor.

Figura 55 – Ponte H L298N com sua pinagem e princípio de funcionamento



Fonte: Adaptado de Last Minute Engineers (2021)

O Quadro 17 apresenta o estado do motor em relação aos sinais de comando da ponte H L298N.

Quadro 17 – Acionamento dos motores em função dos sinais nos pinos de entrada da Ponte H

MOTOR A				**MOTOR B**			
ENA	**IN1**	**IN2**	**Motor**	**ENB**	**IN3**	**IN4**	**Motor**
J2(5V)	0V	0V	Desligado	J3(5V)	0V	0V	Desligado
J2(5V)	5V	0V	Giro Máximo Horário	J3(5V)	5V	0V	Giro Máximo Horário
J2(5V)	0V	5V	Giro Máximo Anti-horário	J3(5V)	0V	5V	Giro Máximo Anti-horário
J2(5V)	5V	5V	Travado	J3(5V)	5V	5V	Travado
PWM (J2-aberto)	0V	0V	Desligado	PWM (J3-aberto)	0V	0V	Desligado
PWM (J2-aberto)	5V	0V	Giro Controlado Horário (0% a 100% velocidade)	PWM (J3-aberto)	5V	0V	Giro Controlado Horário (0% a 100% velocidade)
PWM (J2-aberto)	0V	5V	Giro Controlado Anti-horário (0% a 100% velocidade)	PWM (J3-aberto)	0V	5V	Giro Controlado Anti-horário (0% a 100% velocidade)

Fonte: Autor (2021)

Por se tratar de um periférico que opera com tensão elétrica em 5V, é necessária a utilização de conversor de nível lógico, a fim de converter os sinais de saída do módulo ESP8266 NodeMCU que são de 3,3V em um sinal de 5V. Uma forma de fazê-lo é utilizar um conversor lógico comercial, implementado para esse fim e vendido em um único invólucro. Outra forma é implementar o circuito conversor transistorizado, que apresenta boa resposta e baixo custo de componentes.

Ao editar o programa, as instruções que serão utilizadas são as instruções de escrita digital na saída digital para comando nos sinais IN1, IN2, IN3 e IN4, e a instrução de escrita analógica na saída

PWM comandar os sinais ENA e ENB. Lembrando que o sinal PWM do módulo ESP8266 NodeMCU possui um conversor de 10 bits em cada saída PWM, o que significa que a escala de *duty cycle* de 0% a 100% estará relacionada com uma palavra digital de 0 a 1.023.

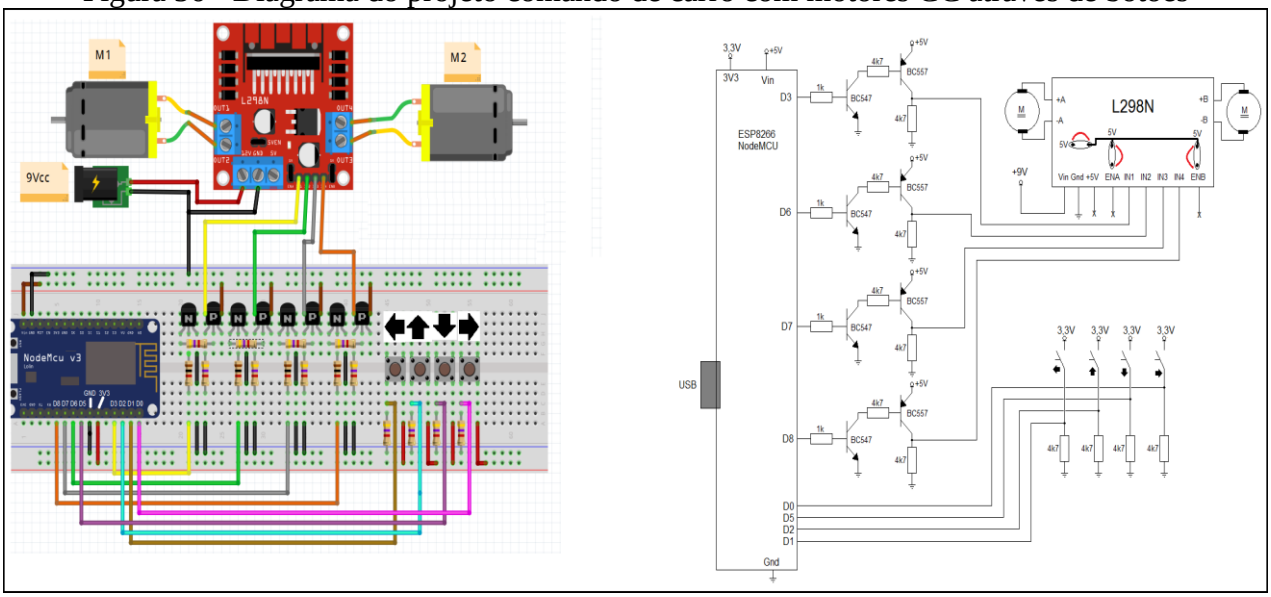
digitalWrite(Pino, Estado); // sinais IN1, IN2, IN3 e IN4 – Estado igual a HIGH ou LOW
analogWrite(Pino, Valor); // sinais ENA e ENB – Valor de 0 a 1.023

PROJETO 10 – COMANDO DE CARRO COM MOTORES CC ATRAVÉS DE BOTÕES

"O projeto consiste em um conjunto de 4 botões que, quando pressionados, provocam o movimento de um carro para frente, para trás, para direita ou para esquerda"

Este projeto se compõe de um chassi com duas rodas tracionadas por dois motores, individualmente, tendo uma terceira roda como dispositivo mecânico de roda livre. A tração em cada roda é independente, sendo que, ao tracionar ambas as rodas para frente ou para trás, provoca-se um movimento linear, ao passo que, ao tracionar as rodas em sentidos opostos, provoca-se um movimento rotacional no carro. O acionamento do motor se dá por uma ponte H L298N, interfaceada com o módulo ESP8266 NodeMCU através de um conjunto de 4 *drivers* implementado por transístores. A Figura 56 apresenta o diagrama do projeto.

Figura 56 - Diagrama do projeto comando de carro com motores CC através de botões



Fonte: Autor (2021)

O código de programa para esse projeto é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define M1h 0    //Motor 1 Horário D3 GPIO0
#define M1ah 12  //Motor 1 Anti-Horário D6 GPIO12
#define M2h 13   //Motor 2 Horário GPIO D7 GPIO13
#define M2ah 15  //Motor 2 Anti-horário D8 GPIO15
#define Direita 15 // D0 GPIO16
#define Esquerda 5 // D1 GPIO5
#define Frente 4   // D2 GPIO4
#define Tras 14    // D5 GPIO14
void setup() {
  pinMode(M1h,OUTPUT);
  pinMode(M1ah,OUTPUT);
  pinMode(M2h,OUTPUT);
  pinMode(M2ah,OUTPUT);
  pinMode(Direita,INPUT);
```

```
  pinMode(Esquerda,INPUT);
  pinMode(Frente,INPUT);
  pinMode(Tras,INPUT);
 }
void loop(){
  if(digitalRead(Direita)){
    digitalWrite(M1h,HIGH);
    digitalWrite(M1ah,LOW);
    digitalWrite(M2h,LOW);
    digitalWrite(M2ah,HIGH);
  }
    if(digitalRead(Esquerda)){
    digitalWrite(M1h,LOW);
    digitalWrite(M1ah,HIGH);
    digitalWrite(M2h,HIGH);
    digitalWrite(M2ah,LOW);
  }
    if(digitalRead(Frente)){
    digitalWrite(M1h,HIGH);
    digitalWrite(M1ah,LOW);
    digitalWrite(M2h,HIGH);
    digitalWrite(M2ah,LOW);
  }
    if(digitalRead(Tras)){
    digitalWrite(M1h,LOW);
    digitalWrite(M1ah,HIGH);
    digitalWrite(M2h,LOW);
    digitalWrite(M2ah,HIGH);
  }
 }
```

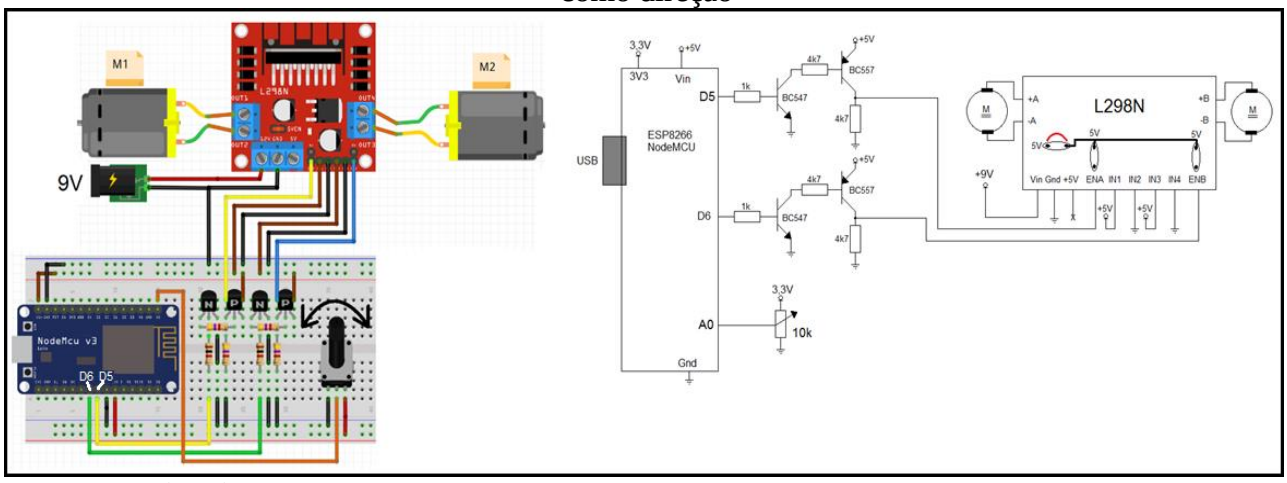
PROJETO 11 – COMANDO DE CARRO COM MOTORES CC ATRAVÉS DE POTENCIÔMETRO COMO DIREÇÃO

"O projeto consiste em um potenciômetro que, em sua posição central faz com que os dois motores girem em máxima velocidade à frente. Ao excursionar o potenciômetro para direita ou esquerda, o respectivo motor diminui a velocidade, provocando um giro, servindo o potenciômetro como uma direção ao carro"

Este projeto se compõe de um chassi com duas rodas tracionadas por dois motores, individualmente, tendo uma terceira roda como dispositivo mecânico de roda livre. A tração em cada roda é independente, sendo que, ao tracionar ambas as rodas para frente, provoca-se um movimento linear, ao passo que ao reduzir a velocidade de uma das rodas, provoca-se um movimento rotacional no carro. O acionamento do motor se dá por uma ponte H L298N, interfaceada com o módulo ESP8266 NodeMCU através de um conjunto de 2 *drivers* implementado por transístores.A Figura 57 apresenta o diagrama do projeto.

A Figura 57 apresenta o diagrama do projeto comando de carro com motores CC através de potenciômetro como direção.

Figura 57 - Diagrama do projeto comando de carro com motores CC através de potenciômetro como direção



Fonte: Autor (2021)

O código de programa para implementar esse projeto é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#define D5 14 //Motor 1 GPIO14
#define D6 12 //Motor 2 GPIO 12
#define A0 0 //Entrada do Potenciômetro
int valor, valor_M1, valor_M2;

void setup() {
  pinMode(D5,OUTPUT);
  pinMode(D6,OUTPUT);
```

```
  pinMode(A0,INPUT);
}

void loop(){
  valor = analogRead(A0)-512; // 0 a 1023 -> -512  a +511
  if(valor <0){
    valor_M1= (1023 + (2*valor));
    analogWrite(D5, valor_M1);//PWM 1023 a 0
    analogWrite(D6, 1023);//PWM =1023
  }else{
    valor_M2= (1023 - (2*valor));
    analogWrite(D6, valor_M2);//PWM 1023 a 0
    analogWrite(D5, 1023);//PWM =1023
  }
}
```

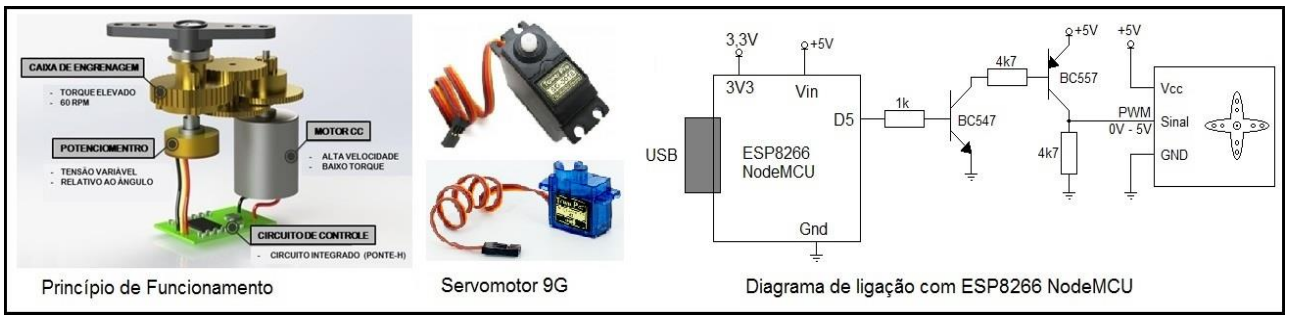
4.4 SERVO MOTOR

Os servos são atuadores projetados para aplicações onde é necessário fazer o controle de movimento com posicionamento de alta precisão, reversão rápida e de alto desempenho. Eles são amplamente usados em robótica, sistemas automatizados, máquinas CNC e em outras diversas aplicações (MATTEDE, 2021).

Os servos motores apresentam diferenças em relação aos demais tipos de motores, pois eles possuem um sensor e um controlador de posição incorporados. Com isso, o sinal de comando que se emite ao servo motor é uma referência desejada de posição, tratando ele de se posicionar nessa referência e lá se manter, independente do que ocorra com a carga.

A Figura 58 apresenta o princípio de funcionamento do servo motor e seu diagrama de ligação com o módulo ESP8266 NodeMCU.

Figura 58 – Princípio de funcionamento e diagrama de ligação de servo motor com módulo ESP8266 NodeMCU



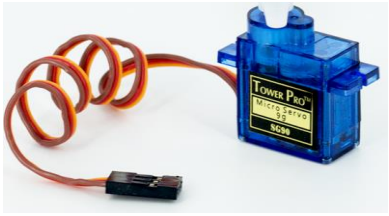
Fonte: Adaptado de Mattede (2021)

O sinal de comando emitido ao servo motor é um sinal PWM, onde o *duty cycle* (razão percentual entre o tempo do sinal em alto pelo tempo do ciclo do sinal) é interpretado como o *range* em graus de deslocamento desejado do servo motor. Como exemplo, o sinal PWM tem *duty cycle* de 0 a 100%, o que corresponde a um ângulo de 0° a 180°.

Como o sinal de entrada do servo motor opera em 5V e a saída do módulo ESP8266 NodeMCU opera em 3,3V, se faz necessário um conversor de nível lógico, que pode ser um dispositivo comercial ou implementado a partir de um circuito transistorizado, como mostra o diagrama da figura.

Em relação às especificações, os servos motores mudam de um modelo ao outro, sua escolha segue a necessidade do projeto. O Quadro 18 apresenta a comparação entre as especificações dos servos motores SG90 e SG5010.

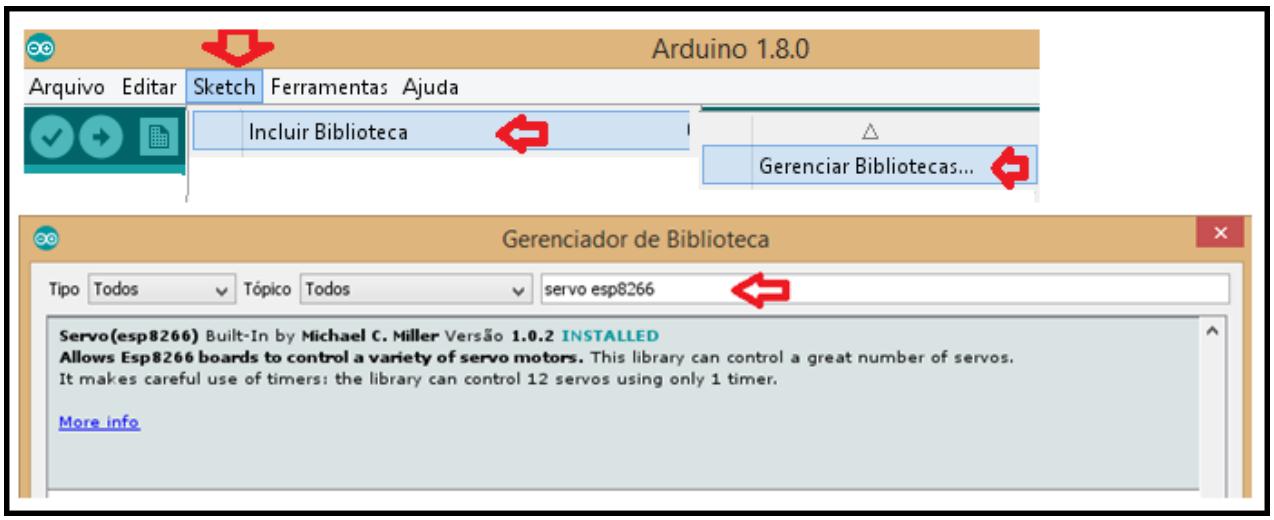
Quadro 18 – Especificações de servo motores SG90 e SG5010

Especificações	**Servo motor 9G - SG90**[24] 	**Servo motor 9G - SG-5010**[25] 
Voltagem de operação	4,8V a 6V	4,8V a 6,6V
Rotação	0° a 180°	0° a 180°
Velocidade	0,12 seg/60°	0,2 seg/60°
Torque médio	2,5 kg.cm	6 kg.cm
Dimensões	32x30x12 mm	38x40x20 mm

Fonte: Autor (2021)

Para implementar o programa de acionamento do servo motor no módulo ESP8266 NodeMCU, é necessária a inclusão de uma biblioteca específica, para agilizar a conexão entre módulo e servo motor. A biblioteca aqui sugerida é a "Servo(esp8266) by Michael C. Miller", que deve ser instalada segundo os passos da Figura 59.

Figura 59 – Instalação da biblioteca para acionamento de servo motor no módulo ESP8266 NodeMCU



Fonte: Autor (2021)

Instalada a biblioteca, então o programa deve começar com a diretiva de inclusão da mesma:

#include <Servo.h>

Se realiza a criação de um objeto tipo "Servo", a fim de poder executar as funções das quais a biblioteca dispõe. Como exemplo, está sendo criado um objeto "Servo" chamado meuServo:

[24] Data sheet do servo motor SG90: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf
[25] Data sheet do servo motor SG5010: https://datasheetspdf.com/pdf-file/633270/TowerPro/SG-5010/1

Servo meuServo;

Dentro da função *setup()*, deve-se anexar ao objeto criado o número da porta que está sendo utilizada para comunicação do sinal ao servo motor. Lembrando que deve ser um GPIO de saída PWM. Como exemplo, está se utilizando o GPIO-14, pino D5, que é uma saída que comporta sinal PWM:

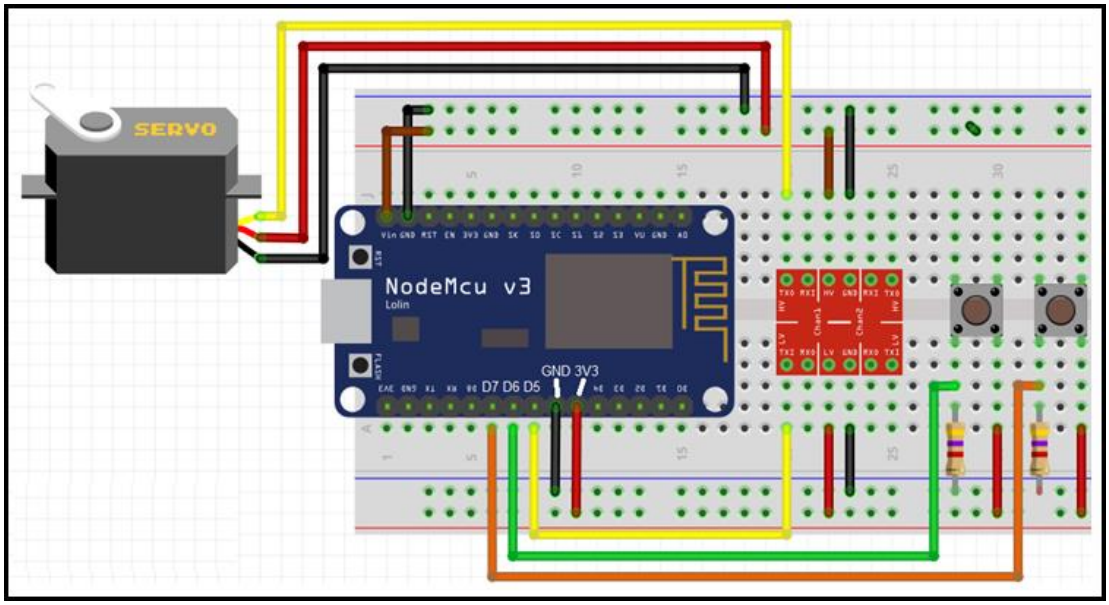
meuServo.attach(14); //porta D5 no GPIO-14

Agora, dentro do programa que será executado, *loop()* ou em uma função, é executada a função *write*(ângulo), para emitir ao servo motor o ângulo desejado para seu posicionamento. Como exemplo, o objeto meuServo indica para o servo motor se posicionar no ângulo de 90°.

meuServo.write(90);

A Figura 60 traz um código de programa para posicionar um servo motor no ângulo 0° e 90°, segundo o botão que for pressionado. Nessa aplicação está sendo utilizado um conversor lógico CNL35.

Figura 60 – Exemplo de programa para acionamento de servo motor com módulo ESP8266 NodeMCU



Fonte: Autor(2021)

O código de programa para esse exemplo de acionamento de servomotor é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#include <Servo.h>
#define B6 12 //porta D6 no GPIO12
#define B7 13 //porta D7 no GPIO13
#define pinServo 14 //porta D5 no GPIO14
Servo meuServo; //cria o objeto Servo

void setup() {
  meuServo.attach(pinServo);
  pinMode(B6, INPUT);
  pinMode(B7, INPUT);
```

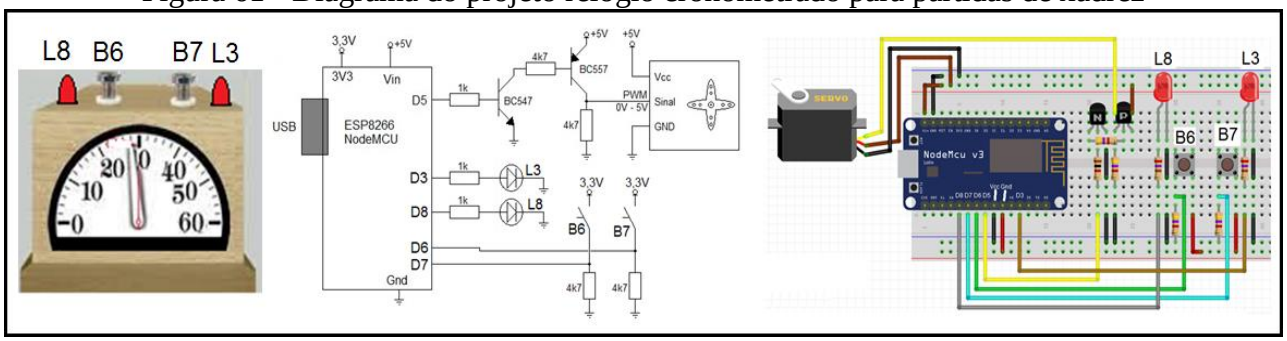
```
  pinMode(pinServo, OUTPUT);
}
void loop() {
  if(digitalRead(B6)){meuServo.write(0);}//angulo de 0º
  if(digitalRead(B7)){meuServo.write(180);}//angulo de 180º
  delay(500);
}
```

O Projeto 12 traz uma aplicação para sistema com servo motor, que se propõe a implementar um relógio cronometrado para partidas de xadrez.

PROJETO 12 – RELÓGIO CRONOMETRADO PARA PARTIDAS DE XADREZ
"O projeto consiste em um conjunto de 2 botões, um para cada jogador, 2 LEDs indicativos e um servo motor que movimenta um ponteiro de relógio, marcador de segundos. A cada vez que um botão é pressionado, o relógio zera e começa a contar os segundos para o outro jogador executar sua jogada, sendo indicado por um LED específico. Caso o jogador não faça a jogada, o tempo de 60 segundos extrapola e o jogo acaba."

A Figura 61 apresenta o diagrama do projeto relógio cronometrado para partidas de xadrez, além de uma sugestão de modelo físico para essa implementação.

Figura 61 - Diagrama do projeto relógio cronometrado para partidas de xadrez



Fonte: Autor (2021)

O código de programa para essa implementação é o seguinte.

```
//Placa: "NodeMCU1.0(ESP-12EModule)"
#include <Servo.h>
#define L3 0  //porta D3 no GPIO-0
#define L8 15 //porta D8 no GPIO-15
#define B6 12 //porta D6 no GPIO-12
#define B7 13 //porta D7 no GPIO-13
#define pinServo 14 //porta D5 no GPIO-14
Servo meuServo; //cria o objeto Servo
bool jogador=LOW;
long TempoAnterior=0;
int passo=0;
void setup() {
  pinMode(B6, INPUT);
  pinMode(B7, INPUT);
  pinMode(pinServo, OUTPUT);
  pinMode(L3, OUTPUT);
  pinMode(L8, OUTPUT);
  meuServo.attach(pinServo);
  meuServo.write(0);//angulo de 0º
}
void loop() {
  if((millis()-TempoAnterior)>1000){
    TempoAnterior=millis();
    passo++;
    meuServo.write(passo);
    if(passo==60){//ocorreu 60 segundos?
      passo--;
      if(jogador==HIGH){
```

```
      digitalWrite(L3,HIGH);
      delay(50);
      digitalWrite(L3,LOW);
     }else{
      digitalWrite(L8,HIGH);
      delay(50);
      digitalWrite(L8,LOW);
     }
    }
  }
  if(digitalRead(B6)){
    delay(100);
    if(digitalRead(B6)){
      digitalWrite(L3,HIGH);
      digitalWrite(L8,LOW);
      jogador=HIGH;
      passo=0;
    }
  }
  if(digitalRead(B7)){
    delay(100);
    if(digitalRead(B7)){
      digitalWrite(L8,HIGH);
      digitalWrite(L3,LOW);
      jogador=LOW;
      passo=0;
    }
  }
}
```

4.6 SENSOR RFID

O leitor MFRC522 Mifare é um leitor de TAGs RFId (*Radio-Frequency Identification*) de 13,56MHz, com interface SPI para troca de dados com microcontroladores. É uma tecnologia de identificação por radiofrequência, geralmente de curto alcance, muito utilizada em controle de acesso, sistemas de bilhetes de transporte público, identificação de produtos, entre outros (THOMSEN, 2014).

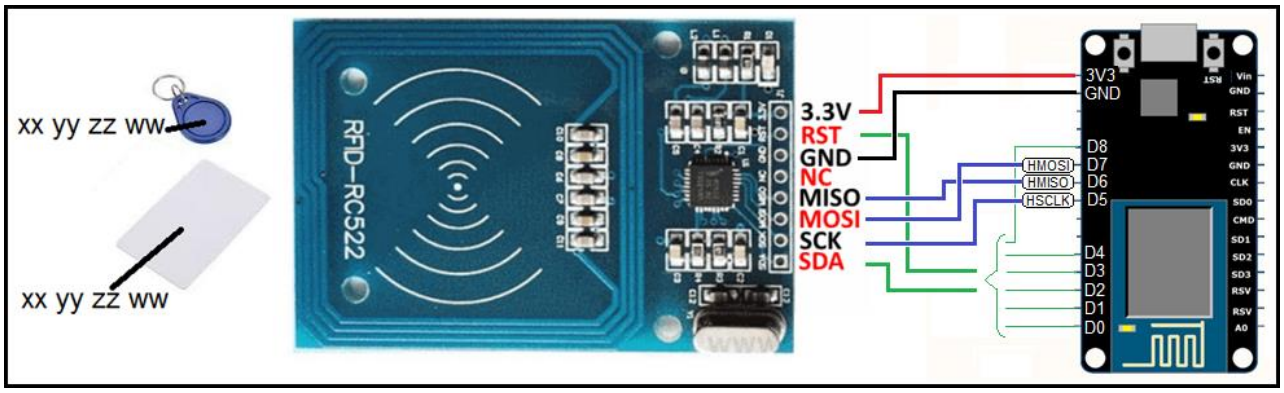
Cada cartão RFID possui um código identificador (UID), de 4 posições no formato "XX YY ZZ WW", sendo que cada posição corresponde a um número de 0 a 255. Isso significa que existem 4.294.967.296 combinações diferentes de cartões, garantindo assim sua individualidade, de forma que não há dois cartões com o mesmo código.

O módulo RC522 é composto dos pinos de alimentação 3V3 e GND, dos quais recebe uma alimentação de 3,3V. Já os pinos SCK, MISO e MOSI obrigatoriamente devem ser conectados aos pinos HSCLK (D5), HMISO (D6) e HMOSI (D7), respectivamente no módulo ESP8266 NodeMCU. Isso porque são os pinos que acessam o módulo interno de comunicação SPI no microcontrolador.

O pino RST faz a inicialização do módulo leitor, devendo ser conectado a uma saída digital de livre escolha. Assim como o pino SDA, também chamado de SS (*Slave Select*) ou CS (*Chip Select*), responsável pela seleção do escravo com o qual está se comunicando.

A Figura 62 apresenta o módulo leitor RC522 com sua pinagem e conexão com ESP8266 NodeMCU.

Figura 62 – Módulo leitor de cartões RFID RC522 com pinagem e conexão com ESP8266 NodeMCU



Fonte: Autor (2021)

O protocolo SPI (*Serial Peripheral Interface*) se destina a uma comunicação bidirecional entre mestre e escravo, através dos pinos MOSI (do mestre para o escravo) e MISO (do escravo para o mestre). Conta com o sinal de sincronismo na transmissão (*Serial Clock*) e o sinal que identifica o

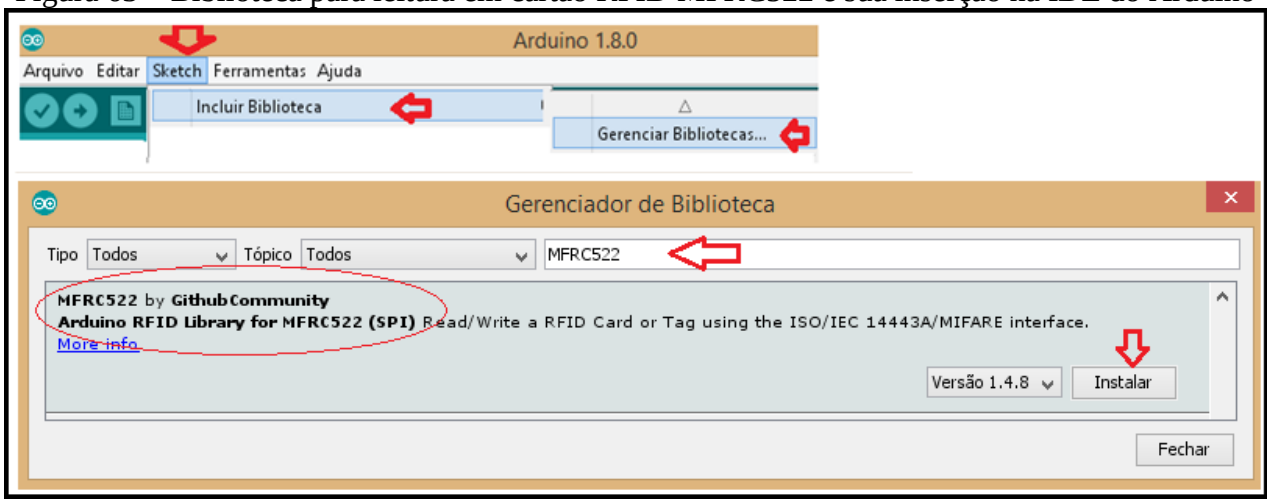
escravo com o qual está se comunicando SS (*Slave Select*). Desta forma, com 4 sinais se permite a comunicação com os devidos periféricos (SACCO, 2014).

O leitor MFRC522 possui as seguintes características (ALBUQUERQUE, 2021):

a) Tipos de cartões suportados: Mifare1 S50, S70 Mifare1, Mifare UltraLight, Mifare Pro, Mifare Desfire;
b) Frequência de operação: 13,56MHz;
c) Corrente de trabalho: 13 a 26mA / em 3,3Vcc;
d) Taxa de transferência: 10Mbps;
e) Interface de comunicação: SPI.

Para leitura dos dados do módulo RC522 é necessária a inserção de biblioteca específica para esse fim. A biblioteca "*MFRC522 by Github Community*" é a indicada para esse estudo, por já ser validada pela IDE Arduino, devendo ser inserida através do Gerenciador de Bibliotecas, como apresentado na Figura 63.

Figura 63 – Biblioteca para leitura em cartão RFID MFRC522 e sua inserção na IDE do Arduino



Fonte: Autor (2021)

Uma vez instalado o pacote de bibliotecas, deve-se utilizar a diretiva #include para incluir no programa a biblioteca MFRC522.h, responsável pelo acesso ao módulo leitor, e a biblioteca SPI.h, responsável pela comunicação via protocolo SPI.

```
#include <SPI.h>
#include <MFRC522.h>
```

Após essas inserções, deve-se criar um objeto tipo MFRC522, a fim de utilizar os métodos disponibilizados pela biblioteca, passando os parâmetros relativos ao número dos pinos de SDA e RST correspondentes à ligação física no *hardware* escolhido.

```
#define SDArfid 0
#define RSTrfid 2
MFRC522 mfrc522(SDArfid, RSTrfid);
```

Para inicialização do módulo leitor de RFID e do protocolo SPI, faz-se necessário dentro da função *setup()* o chamado para as funções *SPI.begin()* e *mfrc522.PCD_Init().*

```
SPI.begin();
mfrc522.PCD_Init();
```

Para fazer a leitura do cartão em RFID, dentro da função *loop(),* é necessário verificar se o cartão está presente e se há condições de reconhecer os dados do cartão. O método *PICC_IsNewCardPresent(),* quando executado, monitora a presença do cartão no leitor, retornando *TRUE* se há um cartão ou *FALSE* se não houver cartão para leitura. Já o método *PICC_ReadCardSerial(),* quando executado, monitora a leitura do cartão RFID, retornando *TRUE* se o cartão pode ser lido ou *FALSE* se não houve condições de leitura do cartão. Esses dois métodos estão presentes no objeto "mfrc522" criado.

```
if (mfrc522.PICC_IsNewCardPresent()){//há cartão presente }
if (mfrc522.PICC_ReadCardSerial()){//se conseguiu ler o cartão}
```

A leitura dos dados do cartão RFID se dá *byte* a *byte*. Por esse motivo é realizado um laço de repetição, a fim de monitorar todos os *bytes* armazenados. O parâmetro **mfrc522.uid.size** retorna o valor do número de *bytes* armazenados no vetor **mfrc522.uid.uidByte[ ]**.

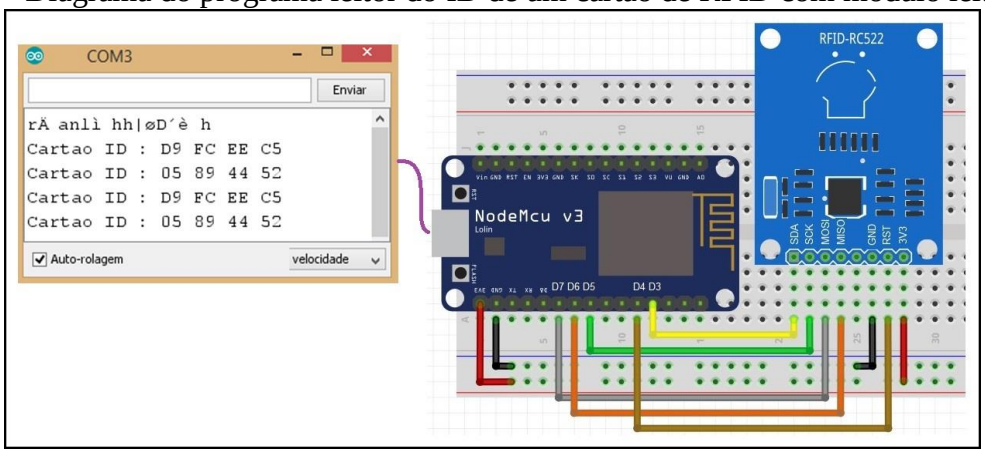
```
for (byte i = 0; i < mfrc522.uid.size; i++){
    conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    conteudo.concat(String(mfrc522.uid.uidByte[i], HEX));
}
```

Ao monitorar a *String* **conteúdo.substring(1)** haverá o valor do ID do cartão, composto por 4 conjunto de *bytes*, separado por espaços. Por exemplo: "D9 FC EE C5" (em decimal: 217 252 238 197).

```
conteudo.toUpperCase();
Serial.println(conteudo substring(1));
```

A Figura 64 apresenta um programa que implementa uma leitura de cartão RFID, apresentando no monitor serial o valor do seu ID na forma hexadecimal.

Figura 64 – Diagrama do programa leitor do ID de um cartão de RFID com módulo leitor RC522



Fonte: Autor (2021)

O código de programa para implementação desse exemplo de leitura de cartão RFId é o seguinte.

```
#include <SPI.h>
#include <MFRC522.h>
//***** Pinagem do leitor RFID*****
//SCKrfid -> D5(GPIO-14) HSCLK
//MISOrfid -> D6(GPIO-12) HMISO
//MOSIrfid -> D7(GPIO-13) HMOSI
#define SDArfid 0 //Qualquer saida digital - D3(GPIO-0)
#define RSTrfid 2 //Qualquer saida digital - D4(GPIO-2)
String anteriorID = "00 00 00 00";
MFRC522 mfrc522(SDArfid, RSTrfid);//cria objeto

void setup() {
  Serial.begin(9600);//inicializa canal serial
  Serial.println();//dar nova linha
  SPI.begin();//inicializa comunicação SPI
  mfrc522.PCD_Init();//inicializa módulo RC522
}

void loop() {
  if (mfrc522.PICC_IsNewCardPresent()){//se há cartão presente
     if (mfrc522.PICC_ReadCardSerial()){//se conseguiu ler o cartão
       String conteudo = "";// cria uma string
       for (byte i = 0; i < mfrc522.uid.size; i++){// faz uma verificacao dos bits da memoria do cartao
          conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));//para valores menor que 0x10, acrescenta
um zero
          conteudo.concat(String(mfrc522.uid.uidByte[i], HEX));
diagrama do projeto relógio cronometrado para partidas de xadrez.       }
       conteudo.toUpperCase();// deixa as letras da string todas maiusculas
       if(conteudo != anteriorID){
         Serial.print("Cartao ID :");
         Serial.println(conteudo);
         anteriorID = conteudo;
       }
     }
  }
}
```

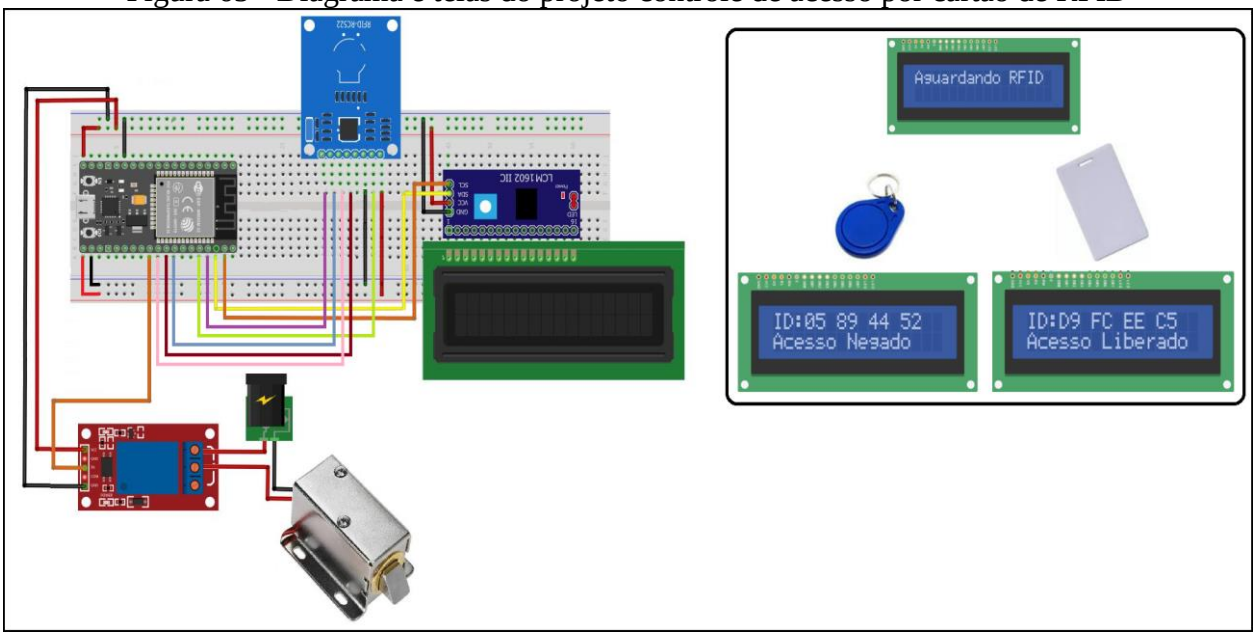
O projeto 13 apresenta um sistema de controle de acesso por leitura de cartão RFID, onde o cartão correto permite a abertura de uma tranca, garantindo o acesso de pessoas autorizadas.

PROJETO 13 – CONTROLE DE ACESSO POR CARTÃO DE RFID

"O projeto consiste em um leitor de RFID RC522 conectado a um módulo ESP8266 NodeMCU, com display LCD I2C e uma tranca magnética acionada por um relé. Ao passar o cartão RFID correto, a tranca abre por 5 segundos, indicando essa situação no display como "Acesso Liberado" no display. Se o cartão for o errado, então uma mensagem no display indica o "Acesso Negado" no display. Em ambos os casos, o ID do cartão é apresentado no display para conferência."

A Figura 65 apresenta o diagrama e as telas do projeto de um controle de acesso por cartão de RFID, onde os códigos de ID de cada cartão são previamente armazenados no programa, de forma a poder comparar se o cartão tem acesso ao ambiente proposto.

Figura 65 - Diagrama e telas do projeto controle de acesso por cartão de RFID



Fonte: Autor (2021)

O código de programa para execução do projeto é o seguinte.

```
#include <SPI.h>
#include <MFRC522.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
//***** Pinagem do leitor RFID*****
//SCKrfid -> D5(GPIO-14) HSCLK
//MISOrfid -> D6(GPIO-12) HMISO
//MOSIrfid -> D7(GPIO-13) HMOSI
#define SDArfid 0 //Qualquer saida digital - D3(GPIO-0)
#define RSTrfid 2 //Qualquer saida digital - D4(GPIO-2)
//***** Ligação da I2C no ESP8266  SCL -> 5(D1) - SDA -> 4(D2)
LiquidCrystal_I2C lcd(0x27,16,2);//(usar codico scan para descobrir)
MFRC522 mfrc522(SDArfid, RSTrfid);//cria objeto
#define Trava 15 // Trava conectada em D8 GPIO-15
#define ID "D9 FC EE C5" //valor de ID para destravar
```

```
void setup() {
  lcd.init();//inicializa o LCD
  lcd.setBacklight(HIGH); //acende o backlight do LCD
  SPI.begin();//inicializa comunicação SPI
  mfrc522.PCD_Init();//inicializa módulo RC522
  pinMode(Trava, OUTPUT);
}
void loop(){
    lcd.clear();
    lcd.setCursor(0,0);// bota o cursor do lcd na posicao inicial
    lcd.print("Aguardando RFID ");   // imprime na primeira linha a string "Aguardando"
    if (!mfrc522.PICC_IsNewCardPresent()){
      return;//se não há cartão presente, retorna ao loop
    }
    if (!mfrc522.PICC_ReadCardSerial()){
      return;//se conseguiu ler o cartão, retorna ao loop
    }
    String conteudo = "";// cria uma string
    for (byte i = 0; i < mfrc522.uid.size; i++){// faz uma verificacao dos bits da memoria do cartao
       conteudo.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));//para valores menor que 0x10, acrescenta um
zero
       conteudo.concat(String(mfrc522.uid.uidByte[i], HEX));
    }
    conteudo.toUpperCase();// deixa as letras da string todas maiusculas
    lcd.setCursor(0,0);// seta o cursor para a segunda linha
    lcd.print("ID:");
    lcd.print(conteudo.substring(1)); // mostra na tela a string "Leitura RFID"
    if(conteudo.substring(1) == ID){
      lcd.setCursor(0,1);// seta o cursor para a segunda linha
      lcd.print("Acesso Liberado ");
      digitalWrite(Trava, HIGH);
      delay(5000);
      digitalWrite(Trava, LOW);
    }else{
      lcd.setCursor(0,1);// seta o cursor para a segunda linha
      lcd.print("Acesso Negado   ");
      delay(5000);
    }
}
```

4.7 SENSOR DE UMIDADE E TEMPERATURA

O Sensor de umidade e temperatura DHT11[26] faz leituras de temperaturas entre 0°C e 50°C e de umidade entre 20% e 90%, com grande aplicação em projetos domóticos[27]. O elemento sensor de temperatura é um termistor do tipo NTC e o sensor de umidade é do tipo HR202. Há um circuito interno que faz a leitura dos sensores e se comunica com um microcontrolador através de um sinal serial de uma via (VIANA, 2020).

As especificações desse sensor são as seguintes:

a) Faixa de medição de umidade: 20% a 90% UR;
b) Faixa de medição de temperatura: 0ºC a 50ºC;
c) Alimentação: 3Vcc a 5Vcc;
d) Corrente: 200uA a 500uA, em *stand by* de 100uA a 150 uA;
e) Precisão de medição de umidade: ± 5,0% UR;
f) Precisão de medição de temperatura: ± 2,0 ºC;
g) Tempo de resposta: 2s;
h) Dimensões: 23mm x 12mm x 5mm, (incluindo terminais).

Uma vez alimentado o sensor em seus pinos Vcc e GND, o pino de sinal Data deve ser conectado ao pino SDA do módulo ESP8266 NodeMCU (D2, GPIO-4), por onde será lido o dado serial enviado pelo sensor.

Para edição do programa para leitura do sensor é necessária a instalação de 2 bibliotecas: a biblioteca **DHT-sensor-library** e a biblioteca **Adafruit_Sensor-master**. Essas duas bibliotecas estão disponibilizadas nos seguintes repositórios:

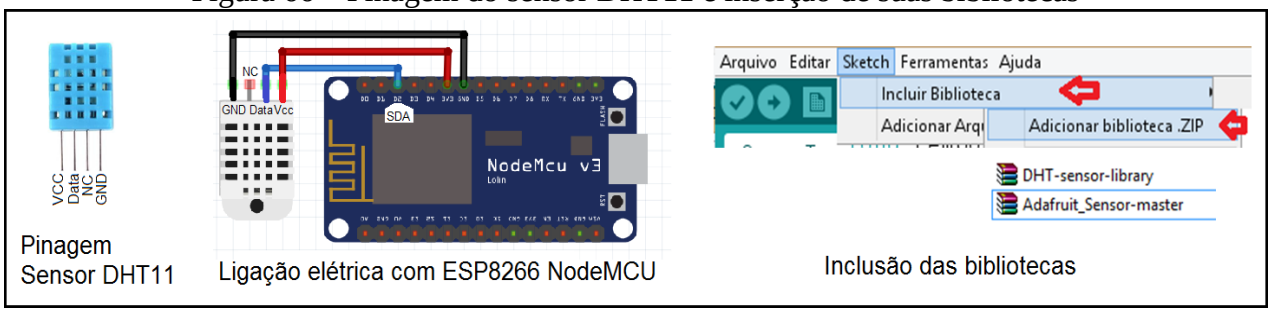
a) Sensor DHT: https://drive.google.com/file/d/16brDLNmOVzmQs6sFh-_CAMivQu-cn6kg/view;
b) Sensor Adafruit: https://drive.google.com/file/d/19ypgAVNwn5f-kb4KDxFr0eOvMQz_yxSU/view.

Depois de procedido o *download* destas duas bibliotecas, as inclusões se dão ao clicar em: “Sketch”, “Incluir Biblioteca”, “Adicionar Biblioteca.zip”, indicando o caminho onde foram salvas.

A Figura 66 apresenta a pinagem do sensor DHT11 e a forma de inserir suas bibliotecas.

[26] Sensor DHT11: *Datasheet* em https://img.filipeflop.com/files/download/Datasheet_DHT11.pdf
[27] Domótica: Controle e automação de sistemas residenciais. Fonte: https://pt.wikipedia.org/wiki/Domótica

Figura 66 – Pinagem do sensor DHT11 e inserção de suas bibliotecas



Fonte: Autor (2021)

Instaladas as bibliotecas, elas serão incluídas no programa através da diretiva:

#include <DHT.h>

Realizada a inclusão da biblioteca, o próximo passo é a criação de um objeto do tipo DHT para poder passar os parâmetros da ligação do sensor e utilizar suas funções. Isso é realizado pela criação de um objeto com a sintaxe DHT nome_objeto(pino_dados, Tipo_sensor). O exemplo mostra a criação de um objeto chamado “dht”, com seu pino de dados ligado ao GPIO-4 (SDA) e com sensor tipo DHT11:

DHT dht(4, DHT11);

O próximo passo é inicializar o sensor, através da função *begin()* vinculada ao objeto “dht” (dht.begin()), chamada dentro da função *setup()* da seguinte forma:

dht.begin();

Para leitura da temperatura e umidade dentro da função *loop()*, basta executar as funções *readTemperature()* e *readHumidity()*, que retornarão respectivamente a temperatura e a umidade medidas, como exemplo:

float temperatura = dht.readTemperature();
float umidade = dht.readHumidity();

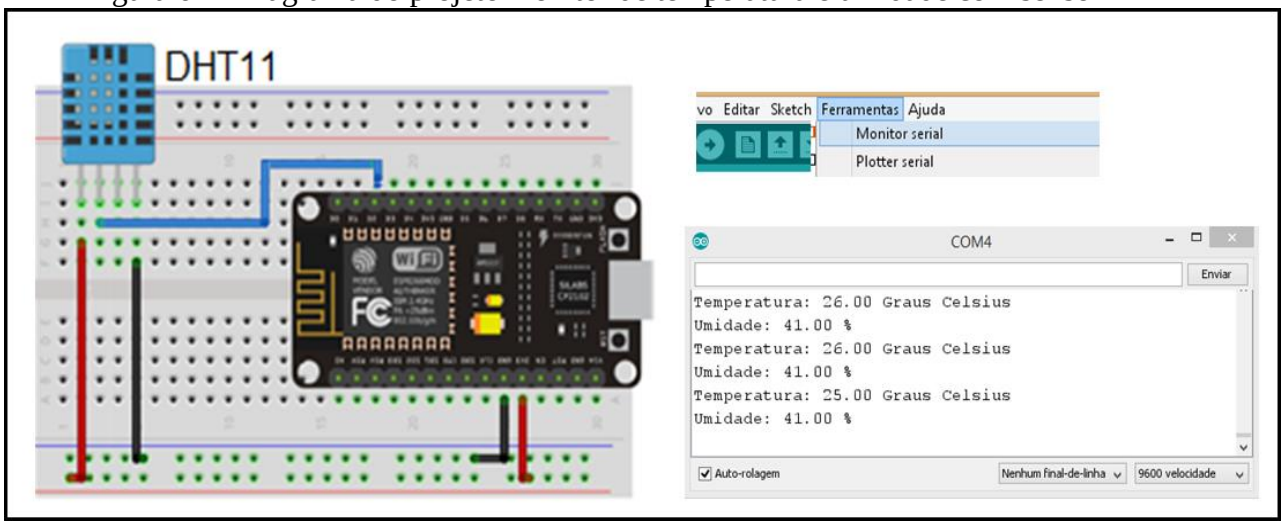
O Projeto 14 apresenta uma aplicação de monitor de temperatura e umidade com sensor DHT11, que mostra os valores lidos no monitor serial do computador, conectado pela serial USB.

PROJETO 14 – MONITOR DE TEMPERATURA E UMIDADE COM SENSOR DHT11

"O projeto consiste em um sensor de temperatura e umidade DHT11, conectado a um módulo ESP8266 NodeMCU, que faz as leituras e apresenta os resultados através do monitor serial."

A Figura 67 apresenta o diagrama do projeto, onde constam as ligações do sensor DHT11 com o módulo e sua forma de exibição pelo monitor serial da IDE Arduino.

Figura 67 - Diagrama do projeto monitor de temperatura e umidade com sensor DHT11



Fonte: Autor (2021)

O código de programa para esse projeto é o seguinte.

```
#include <DHT.h>
#define DHTPIN 4 // pino do sensor D2 - GPIO4 - SDA (não pode ser outro)
#define DHTTYPE DHT11 //tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
unsigned long tempoAnterior = 0;
float temperatura; //variável para armazenar a temperatura
float umidade; //Variável para armazenar a umidade

void setup(){
  Serial.begin(9600);
  dht.begin();
}
void loop(){
  if(millis() - tempoAnterior > 5000){//leitura a cada 5s
    tempoAnterior = millis();
    temperatura = dht.readTemperature(); //Le a temperatura
    umidade = dht.readHumidity(); //Le a Humidade
    //Mostra a temperatura e Humidade no Serial Monitor
    Serial.print("Temperatura: ");
    Serial.print(temperatura); //Imprime temperatura lida
    Serial.println(" Graus Celsius");
    Serial.print("Umidade: ");
    Serial.print(umidade); //Imprime umidade lida
    Serial.println(" %");
  }
}
```

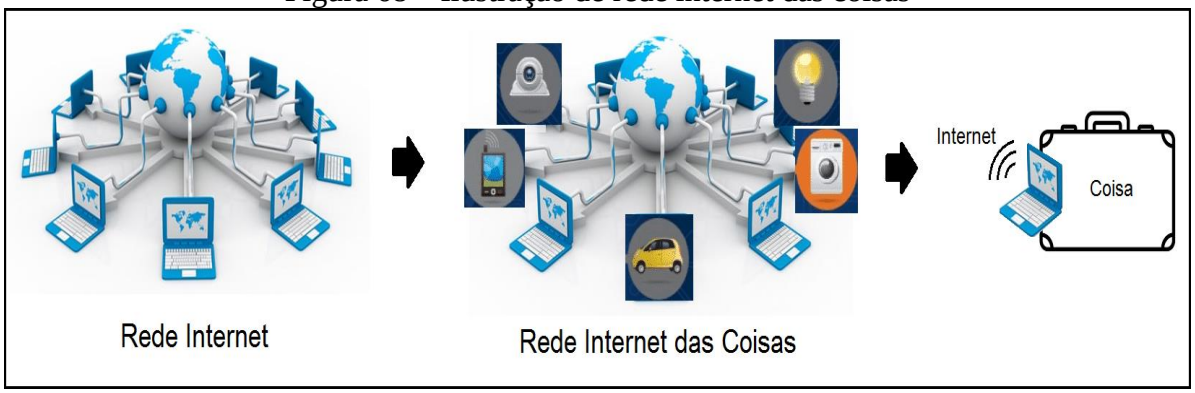
5 INTERNET DAS COISAS

A internet nasceu no final dos anos 1960, por iniciativas militares, como forma de compartilhar dados em lugares distintos. Desde sua criação, o propósito era que uma informação de uma pessoa chegasse à outra, ou então que uma informação em um servidor de dados pudesse ser acessada por uma pessoa (SILVA, 2017a). Isto acabou formando a rede mundial de computadores conectados na internet, onde pessoas podem trocar informações (dados) com outras pessoas ou acessar informações em *site*s que hospedam dados de interesse.

Mas, com a chegada da 4ª revolução industrial, a indústria 4.0, a necessidade de interação entre máquinas, equipamentos, produtos e pessoas aumentou. Evoluiu para que as próprias "coisas"[28] (máquinas, equipamentos e produtos) necessitassem interagir entre si, trocando dados, comandos e compartilhando informações. Foi além da esfera fabril, foco da indústria 4.0, atingindo a vida no dia a dia (SILVA, 2017b).

Agora há a possibilidade de as coisas acessarem a internet, buscarem interação e participarem de uma rede de coisas e humanos interligada. A Figura 68 apresenta a proposta de internet das coisas (*Internet of Things* – IoT).

Figura 68 – Ilustração de rede internet das coisas



Fonte: Adaptado de Silva (2017a)

O que de fato ocorreu foi que já havia capacidade computacional para que computadores acessassem a rede internet, porém, eram muito grandes e demandavam muita energia para sua operação. Com a evolução dos microcontroladores, se chegou a pequenos dispositivos que possuem capacidade computacional suficiente para promover o acesso à rede internet, com baixo consumo de

[28] "Coisas", tudo que não é humano. Mais especificamente relacionado a elementos que possuem recursos computacional que permita o acesso a rede *internet*, mas que não são um computador propriamente dido.

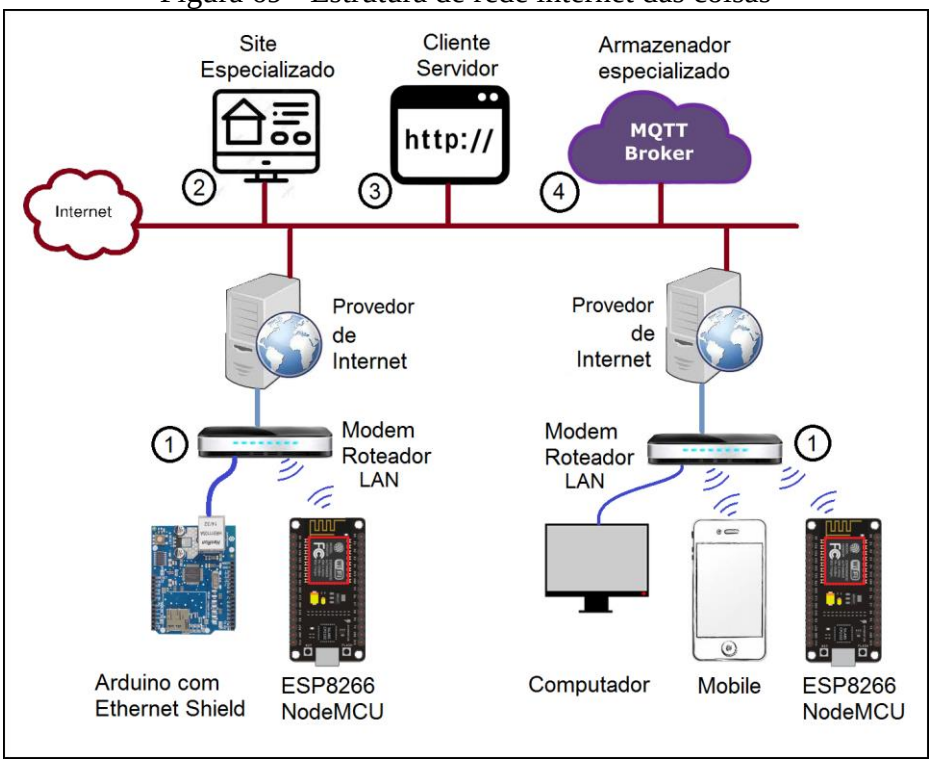
energia. Esses novos microcontroladores passaram a se incorporar às máquinas, aos equipamentos, aos produtos, enfim, às coisas.

Agora, um sensor de temperatura e umidade pode enviar suas leituras via internet para um computador, um dispositivo *mobile*, ou até mesmo para um microcontrolador que irá tomar alguma ação frente a esses dados. Vale-se de uma das principais características: a mobilidade.

Para que ocorra esta conexão do microcontrolador com a internet, algumas coisas devem ocorrer, tanto no meio físico quanto do protocolo para troca de mensagens (LAVRATTI, 2016). Não é algo trivial, mas nada que assuste muito.

Primeiro deve se ter em mente que para um microcontrolador acessar a internet, ele deve ter acesso a uma rede de dados local – LAN[29] (*Local Area Network*), disponibilizada por um *modem*[30]. Este é o ponto de acesso que se tem para entrar em um sistema em rede que permitirá chegar até a internet. O *modem*, por sua vez irá permitir a conexão com o provedor de internet contratado, sendo este o responsável por prover o acesso a rede internet. A Figura 69 apresenta a estrutura de internet das coisas.

Figura 69 - Estrutura de rede internet das coisas



Fonte: Autor (2021)

[29] LAN é *Local Área Network*. Esse termo geralmente se refere a redes de computadores restritas a um local físico definido como uma casa, escritório ou empresa em um mesmo prédio. Uma rede sem fio de uma empresa também faz parte da LAN. Fonte: https://helpdigitalti.com.br/o-que-e-uma-rede-lan-e-uma-rede-wan/

[30] *Modem* é o dispositivo que conecta a rede doméstica ao provedor de internet, através de um meio físico próprio (linha telefônica a fio, linha telefônica 4G, fibra ótica, sinal de rádio, entre outras). Fonte: https://conceito.de/modem

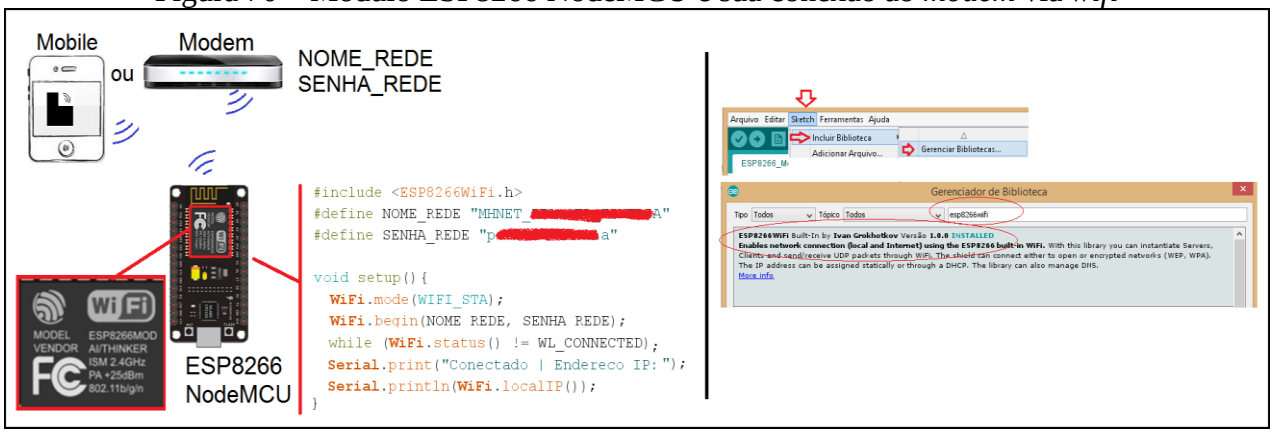
Dessa forma, para que uma “coisa” se conecte à internet, basta que ela tenha uma conexão com uma rede LAN ou um ponto de acesso ①. E, estando conectada, a “coisa” poderá acessar sites especializados ②, atuar como cliente/servidor em uma ação *webservice* ③ ou acessar armazenadores de dados especializados ④.

5.1 CONEXÃO *WIFI* DO MÓDULO ESP8266 NODEMCU

O módulo ESP8266 NodeMCU é um conjunto de componentes montados em uma mesma placa, permitindo assim sua função como kit de desenvolvimento. Um destes componentes é o *chip* ESP8266, principal ator do módulo, que propicia a conexão *wifi* com um *modem*, permitindo a conexão à internet. Esse *chip* utiliza o padrão IEEE 802.11b/g/n *wifi* para sua comunicação, além de outras funcionalidades que facilitam sua conexão e uso.

A Figura 70 destaca o chip ESP8266 no módulo NodeMCU, apresenta a inserção da biblioteca com com métodos de acesso à conexão e os comandos para sua execução.

Figura 70 – Módulo ESP8266 NodeMCU e sua conexão ao *modem* via *wifi*



Fonte: Autor (2021)

Para programar o módulo para conectar-se a uma rede LAN, é necessária a inserção da biblioteca “ESP8266WiFi.h”, através da diretiva #include <ESP8266WiFi.h>, mas, para isso, é preciso instalar o pacote de bibliotecas que traz essa biblioteca específica. Isso pode ser realizado através do gerenciador de bibliotecas da IDE Arduino.

Basta clicar em “Sketch”, “Incluir Biblioteca” e “Gerenciador Bibliotecas”, solicitando pesquisa sobre “esp8266wifi”, que lhe será apresentado o pacote “ESP8266WiFi by Ivan Grokhotkov”. Se deve instalar este pacote e no programa que estiver construindo, inserindo a diretiva:

#include <ESP8266WiFi.h>

Dentro da função *setup()* deve-se indicar o modo de trabalho como uma estação *wifi*, através da função WiFi.mode(WIFI_STA). Após, é necessário inicializar o *chip wifi* com a inserção do nome

da rede LAN (parâmetro NOME_REDE) e sua senha de acesso (parâmetro SENHA_REDE), através da função WiFi.begin(NOME_REDE, SENHA_REDE).

```
setup(){
   WiFi.mode(WIFI_STA);
   WiFi.begin(NOME_REDE, SENHA_REDE);
```

Após, deve-se esperar a conexão ocorrer, o que pode demorar, sendo necessário um monitoramento da conexão. Isso é realizado através da função WiFi.status() que retorna o texto WL_CONNECTED caso a conexão esteja realizada. Desta forma, é utilizado o comando *while* para identificar essa condição, no formato while(WiFi.status() != WL_CONNECTED).

```
setup(){
   while(WiFi.status() != WL_CONNECTED);
```

Como se trata de uma conexão a um *modem* roteador, na sua aplicação padrão atribui um endereço de IP aos seus elementos que solicitam conexão a cada vez que isso ocorre, evitando assim conflito de endereços na rede. Para monitorar o IP que foi atribuído, basta utilizar a função WiFi.localIP(), que retorna o endereço de IP que está atribuído após a conexão.

```
Serial.println(WiFi.localIP());
```

A Figura 71 apresenta um exemplo para conexão do módulo ESP8266 NodeMCU em uma rede LAN ou a um ponto de acesso de um *mobile* roteador *wifi*.

Figura 71 – Exemplo de código para conexão do módulo ESP8266 NodeMCU em uma rede LAN *wifi*

Fonte: Autor (2021)

O código de programa para implementação desse exemplo é o seguinte.

```
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <ESP8266WiFi.h>
#define NOME_REDE "CASA" //Nome da Rede LAN
#define SENHA_REDE "MINHA_CASA" //Senha de acesso na Rede LAN

void setup()
{
  Serial.begin(115200); // Inicia Serial 115200Bps
  Serial.println();
  WiFi.mode(WIFI_STA); //Habilita o modo estação
  WiFi.begin(NOME_REDE, SENHA_REDE); //Inicia WiFi na rede e senha
```

```
  delay(1000); // aguarda 1 segundo
  Serial.print("Conectando");
  while (WiFi.status() != WL_CONNECTED){ // Tentando conectar na Rede WiFi
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Conectado | Endereco IP: "); // Imprime na Serial
  Serial.println(WiFi.localIP()); //Imprime o Endereço IP Local do ESP8266
  delay(1000); // aguarda 1 segundo
}

void loop(){
  // programa a ser executado
}
```

Fonte: Autor (2021).

Como aplicação das conexões *wifi* com roteador, são apresentados 2 projetos ilustrativos das funcionalidades de conexão, sendo que o Projeto 15 apresenta um monitor de redes *wifi* disponíveis ao módulo e o Projeto 16 traz a comunicação entre dois módulos através da conexão *wifi* pelo roteador.

PROJETO 15 – MONITOR DE REDES WIFI COM ESP8266 NODEMCU

"O projeto consiste em um módulo ESP8266 NodeMCU, que monitora as redes wifi que estão dentro de sua área de alcance, exibindo seus nomes, nível de sinal em decibéis (dB) e condição de conexão no monitor serial"

Para realização desse projeto, algumas funções específicas são utilizadas na biblioteca "ESP8266WiFi.h".

De começo, o módulo deve ser configurado como uma estação *wifi* e seu canal de comunicação *wifi* desabilitado, para que possa monitorar todas as redes e não se ater a uma rede específica para comunicação. Isso deve ser realizado dentro da função *setup().*

```
WiFi.mode(WIFI_STA); // configura rede no modo estacao
WiFi.disconnect();   // desconecta rede WIFI
```

Dentro da função *loop*() é então executado o método WiFi.scanNetworksAsync(), que monitora as redes disponíveis, retornando o número de redes encontradas, que deve ser endereçada para uma variável do tipo int de uma função (int&). Por exemplo: WiFi.scanNetworksAsync(pesquisarRedes) → void pesquisarRedes(int nRedes). A variável nRedes conterá o número de redes encontradas.

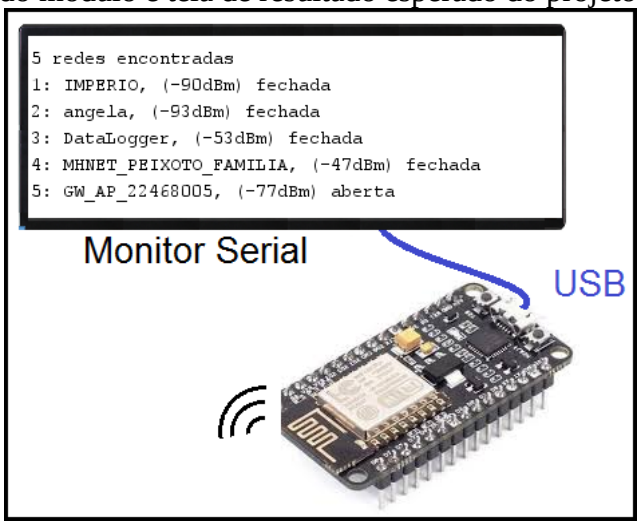
```
WiFi.scanNetworksAsync(int&);
```

Uma vez encontradas as redes, as informações delas ficam armazenadas no *chip* ESP8266, podendo ser lidas por funções específicas.

```
WiFi.SSID(i).c_str(); //retorna uma string com o nome da rede
WiFi.RSSI(i); // retorna um int com o nível do sinal em decibéis
WiFi.encryptionType(i) == ENC_TYPE_NONE ? "aberta" : "fechada");
```

A Figura 72 apresenta a tela de resultado esperado do projeto.

Figura 72 – Conexão do módulo e tela de resultado esperado do projeto monitor de redes wifi



Fonte: Autor (2021)

O código de programa para implementação do projeto monitor de redes *wifi* com ESP8266 NodeMCU é o seguinte.

```
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <ESP8266WiFi.h>
void setup(){
  Serial.begin(115200);// configura monitor serial 115200 Bps
  Serial.println();    // leva cursor para nova linha
  WiFi.mode(WIFI_STA); // configura rede no modo estacao
  WiFi.disconnect();   // desconecta rede WIFI
  delay(100);          // atraso de 100 milisegundos
}
void imprimeResultado(int NumeroRedes){
  Serial.printf("\n");                              // cursor para nova linha
  Serial.printf("%d redes encontradas\n", NumeroRedes); // imprime numero de redes encontradas
  for (int i = 0; i < NumeroRedes; i++){ // contagem das redes encontradas
    Serial.printf("%d: %s, (%ddBm) %s\n", i + 1, WiFi.SSID(i).c_str(), WiFi.RSSI(i), WiFi.encryptionType(i) ==
ENC_TYPE_NONE ? "aberta" : "fechada");
  }
}
void loop(){
  WiFi.scanNetworksAsync(imprimeResultado);// Faz busca pelas redes
  delay(500);                          // atraso de 0,5 segundos
}
```

PROJETO 16 – COMUNICAÇÃO ENTRE DOIS MÓDULOS ESP8266 NODEMCU ATRAVÉS DO ROTEADOR WIFI

"O projeto consiste em dois módulos ESP8266 NodeMCU, conectados a um roteador wifi, que trocam informações entre si, onde um botão conectado a um módulo aciona um LED no outro módulo e vice-versa"

Esse projeto prevê uma comunicação de mensagens entre dois módulos ESP8266 NodeMCU, através do roteador *wifi*, inspirado no projeto de Marques (MARQUES, 2019). A essa comunicação dá-se o nome de ponto a ponto (*Point to point* – P2P). Isso significa que um módulo (cliente) transmite uma mensagem a outro módulo (servidor) com endereço conhecido na rede e recebe resposta desse módulo. As mensagens são no formato de *String*, que podem ser interpretadas como informação ou comando no programa.

Esse tipo de abordagem resolve problemas de segurança na comunicação, pois trata-se de uma comunicação local, fisicamente definida, sem capacidade de intervenção externa. Além de resolver necessidades imediatas, onde se necessita da mobilidade como fator importante no projeto.

Para implantação, é necessária a inclusão da biblioteca ESP8266WiFi.h, que pode ser inserida no Gerenciador de bibliotecas, procurando por "ESP8266WIFI by Ivan Grokhotkov". E também é necessária a inclusão da biblioteca SPI.h, nativa no IDE Arduino, que auxilia na comunicação.

```
#include <SPI.h>
#include <ESP8266WiFi.h>
```

No programa para o módulo cliente (transmissor) é necessário indicar o endereço IP do servidor (receptor), com o qual a comunicação se dará. Isso é realizado pela criação de um objeto IPAddress com nome de "server", passando os parâmetros do endereço IP do servidor. É também criado um objeto WiFiClient com nome "client", para utilizar as funções necessárias à comunicação.

```
IPAddress server(192,168,0,240);
WiFiClient client
```

Dentro da função *setup*(), será inicializado o módulo cliente na rede, com a função *WiFi.begin*, indicando por parâmetro o nome da rede e a senha da rede. Depois de inicializada, é necessário esperar a conexão ocorrer, através da monitoração da função *WiFi.status*(), que retorna a constante WL_CONNECTED quando a conexão ocorrer e se estabilizar.

```
void setup() {
   WiFi.begin(NOME_REDE, SENHA_REDE); //Conexão com roteador wifi
   while (WiFi.status() != WL_CONNECTED);
```

Para que as transmissões ocorram e as respostas sejam recebidas, dentro da função *loop*() deve-se iniciar uma conexão com o servidor, através da função *client.connect*(*server*,80), indicando por parâmetro o servidor e a porta que será utilizada, nesse caso a porta 80.

```
void loop(){
   client.connect(server, 80);
```

Para enviar uma mensagem para o servidor (receptor) deve-se utilizar a função *client.print*(“FRASE\r”), indicando por parâmetro a mensagem na forma de *String*, terminando com o caractere ‘\r’. Após proceder todas as transmissões, deve-se utilizar a função *client.println*(“*Transmmiter*”) para instigar uma resposta.

```
client.print("LIGAR\r");
client.println("Transmitter");
```

Uma vez transmitida a mensagem, faz-se necessário receber a resposta do servidor (receptor). Como a resposta ocorrerá na forma de frase no formato *String*, é utilizada a função *client.readStringUntil*(‘\r’), que irá ler a *String* recebida como resposta até atingir o caractere ‘\r’. Na sequência, é necessária a função *client.flush*(), para garantir a chegada de toda a mensagem no *buffer* de entrada do módulo. Essa mensagem de resposta pode ser interpretada como uma informação ou comando pelo programa do módulo cliente.

```
String resposta = client.readStringUntil('\r');
client.flush();
```

O programa para o módulo servidor (receptor), além de incluir as bibliotecas SPI.h e ESP8266WiFi.h, é necessário criar o objeto *WiFiServer* com nome de “*server*(80)”, indicando por parâmetro a porta 80.

```
#include <SPI.h>
#include <ESP8266WiFi.h>
WiFiServer server(80);
```

Como se trata de um servidor que deve ser reconhecido pelos clientes que solicitarão resposta, lhe deve ser atribuído um IP fixo, através da função *IPAddress*, criando os objetos *ip*, *gateway* e *subnet*, com os dados da rede. É importante ressaltar que o IP fixo não pode ser um endereço já utilizado por algum componente da rede. Também deve ser preservada a máscara de rede, pois elas definem o endereço da rede em que o servidor está operando (192.168.0.0 neste caso).

```
IPAddress ip(192, 168, 0, 240);
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
```

Dentro da função *setup*() do programa servidor (receptor) é necessário configurar o módulo servidor com um IP fixo, através da função *WiFi.config*(*ip*, *gateway*, *subnet*), passando por parâmetro os dados fixos que se quer para o módulo.

```
void setup(){
   WiFi.config(ip, gateway, subnet);
```

Após isso, ainda dentro da função *setup*(), deve-se inicializar o módulo servidor na rede, através da função *WiFi.begin*(NOME_REDE, SENHA_REDE), passando por parâmetro o nome da rede e sua senha. Após, deve-se inicializar o módulo como um servidor, através da função

server.begin(), aguardando a conexão ocorrer e se estabilizar através do monitoramento da função *WiFi.status*(), que retorna a constante WL_CONNECTED se a conexão ocorrer.

```
void setup() {
    WiFi.begin(NOME_REDE, SENHA_REDE);
    server.begin();
    while (WiFi.status() != WL_CONNECTED);
```

Para receber as solicitações do cliente (transmissor), dentro da função *loop*() o servidor deve verificar se há um pedido do cliente, o que é realizado pela função *server.available*(). Caso não haja solicitação *if*(!*client*), a função *loop*() retorna a seu início.

```
void loop() {
    WiFiClient client = server.available();
    if (!client) return;
```

Uma vez que haja uma solicitação do cliente, então o servidor recebe a mensagem do solicitante pela função *client.readStringUntil*('\r'), que irá ler a *String* recebida até atingir o caractere '\r'. Na sequência, é necessária a função *client.flush*(), para garantir a chegada de toda a mensagem no *buffer* de entrada do módulo. Essa mensagem de solicitação pode ser interpretada como uma informação ou comando pelo programa do módulo cliente.

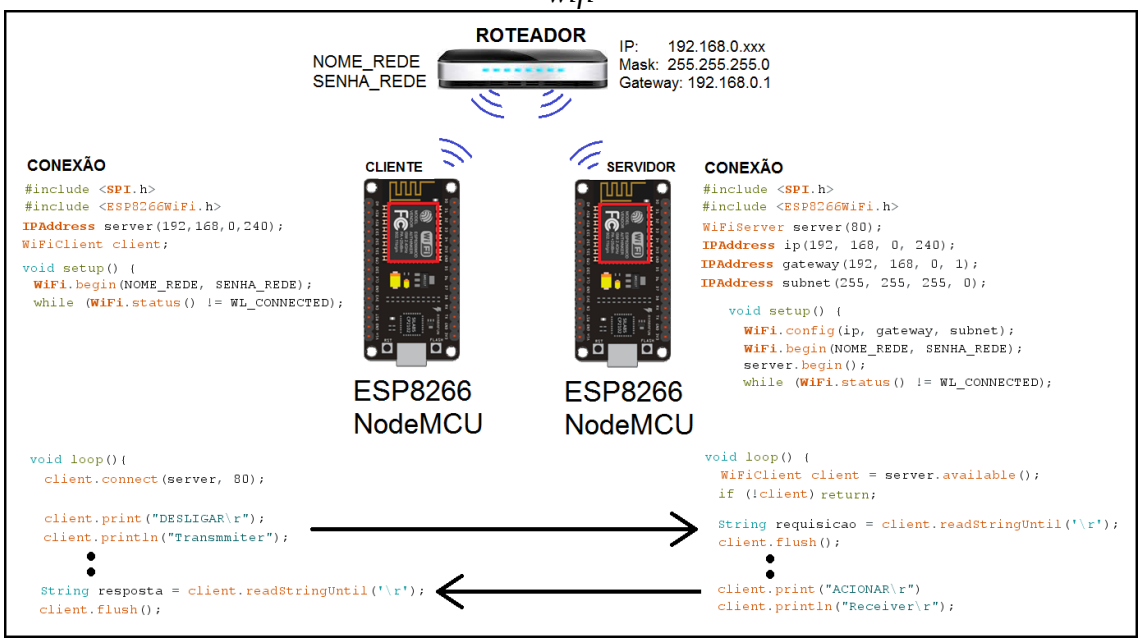
```
String requisicao = client.readStringUntil('\r');
client.flush();
```

Para responder ao solicitante, o servidor o fará através de mensagem de resposta na forma de *String*, através da função *client.print*("FRASE\r"), que passará, por parâmetro, a frase de resposta, terminada pelo caractere '\r'. Após o envio da resposta, deve-se executar a função *client.println*("Receiver\r"), para instigar uma nova transmissão.

```
client.print("DESACIONAR\r");
client.println("Receiver\r");
```

A Figura 73 mostra o princípio da comunicação P2P (ponto a ponto) com módulo ESP8266 NodeMCU através de roteador *wifi*.

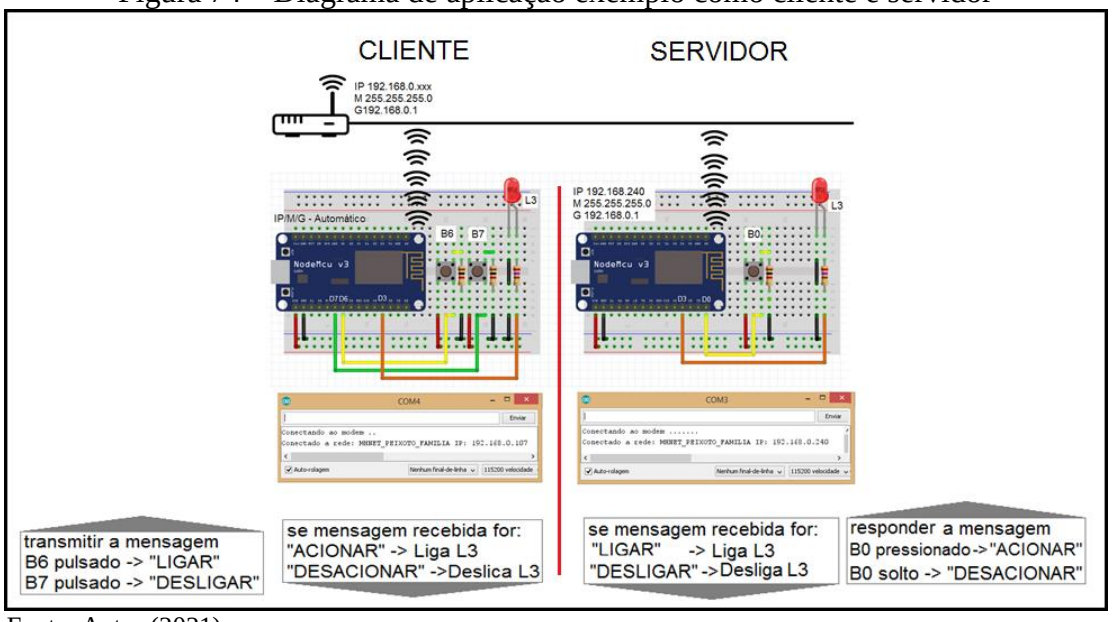
Figura 73 – Princípio da comunicação P2P com módulo ESP8266 NodeMCU através de roteador *wifi*



Fonte: Autor (2021)

A Figura 74 traz uma aplicação para o transmissor (Cliente) e receptor (Servidor), com seu diagrama e mensagens de acesso.

Figura 74 – Diagrama de aplicação exemplo como cliente e servidor



Fonte: Autor (2021)

O trecho de código para essa aplicação como transmissor (Cliente) é o seguinte.

```
//**TRANSMISSOR DE MENSAGENS PONTO A PONTO****
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
#include <SPI.h> //nativa IDE Arduino
//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
```

```
#include <ESP8266WiFi.h>
#define LedBoard 2 // Led Build_in D4 GPIO-2
#define L3 0  // L3 vermelho D3 GPIO-0
#define B6 12 //chave em D6 GPIO-12
#define B7 13 //chave em D7 GPIO-13
char NOME_REDE[] = "*********";//Nome da rede wifi
char SENHA_REDE[] = "*********";//Senha da rede wifi
IPAddress server(192,168,0,240);//Endereço do receptor destino
WiFiClient client;//cria o objeto "client"

void setup() {
  pinMode(LedBoard, OUTPUT);
  digitalWrite(LedBoard, LOW);//liga o LEDBOARD
  WiFi.begin(NOME_REDE, SENHA_REDE); //Conexão com roteador wifi
  //*** OPCIONAL SO PARA O MONITORAMENTO INICIAL ****
  Serial.begin(115200);
  Serial.println();
  Serial.print("Conectando ao modem ");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Conectado a rede: ");
  Serial.print(WiFi.SSID());
  Serial.print(" IP: ");
  Serial.println(WiFi.localIP());
  //***********************************************
  pinMode(L3, OUTPUT);
  pinMode(B6, INPUT);
  pinMode(B7, INPUT);
  digitalWrite(L3, LOW);//Desliga o LED L3
  digitalWrite(LedBoard, HIGH);//desliga o LEDBOARD
}

void loop(){
  client.connect(server, 80); //conexão com o receptor na porta 80
  if(digitalRead(B6) == HIGH){
    client.print("LIGAR\r");//envia para o receptor
    delay(200);
  }
  if(digitalRead(B7) == HIGH){
    client.print("DESLIGAR\r");//envia para o receptor
    delay(200);
  }
  client.println("Transmitter"); //instiga uma resposta
  String resposta = client.readStringUntil('\r');//recebe resposta até o caractere \r
  client.flush();//carrega toda mensagem que chegou
  if(resposta == "ACIONAR"){//compara mensagem que chegou
    digitalWrite(L3, HIGH);//liga o led L3
  }
  if(resposta == "DESACIONAR"){ //compara mensagem que chegou
    digitalWrite(L3, LOW); //desliga o led L3
    delay(200);
  }
}
```

O trecho de código para essa aplicação como receptor (Servidor) é o seguinte.

```
//**RECEBEDOR DE MENSAGENS PONTO A PONTO****
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
#include <SPI.h> //nativa IDE Arduino
//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <ESP8266WiFi.h>
```

```
#define LedBoard 2 // Led Build_in D4 GPIO-2
#define L3 0  // L3 vermelho D3 GPIO-0
#define B0 16 // Botao B0 em D0 GPIO-16
char NOME_REDE[] = "**********";//Nome da rede wifi
char SENHA_REDE[] = "**********";//Senha da rede wifi
WiFiServer server(80);//Cria o objeto "server" na porta 80
IPAddress ip(192, 168, 0, 240);     //IP fixo do receptor
IPAddress gateway(192, 168, 0, 1);  // gateway da rede
IPAddress subnet(255, 255, 255, 0); // máscara da rede

void setup() {
  pinMode(LedBoard, OUTPUT);
  digitalWrite(LedBoard, LOW);//liga o LEDBOARD
  WiFi.config(ip, gateway, subnet);//força um IP fixo
  WiFi.begin(NOME_REDE, SENHA_REDE); //Conexão com roteador wifi
  server.begin();//inicializa o receptor como server
  //*** OPCIONAL SO PARA O MONITORAMENTO INICIAL ****
  Serial.begin(115200);
  Serial.println();
  Serial.print("Conectando ao modem ");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println();
  Serial.print("Conectado a rede: ");
  Serial.print(WiFi.SSID());
  Serial.print(" IP: ");
  Serial.println(WiFi.localIP());
  //***********************************************
  pinMode(L3, OUTPUT);
  pinMode(B0, INPUT);
  digitalWrite(L3, LOW);//Desliga o LED L3
  digitalWrite(LedBoard, HIGH);//desliga o LEDBOARD
}
void loop() {
  WiFiClient client = server.available();
  if (!client) {//se não houver conexão, reinicia o loop
    return;
  }
  String requisicao = client.readStringUntil('\r');//recebe resposta até o caractere \r
  client.flush();//carrega toda mensagem que chegou
  if (requisicao == "LIGAR") {//compara mensagem que chegou
    digitalWrite(L3, HIGH);//liga o led L3
    delay(200);
  }
  if (requisicao == "DESLIGAR") {//compara mensagem que chegou
    digitalWrite(L3, LOW);//liga o led L3
    delay(200);
  }
  if(digitalRead(B0)) {//monitora o botão B0
    client.print("ACIONAR\r");//responde ao transmissor
    delay(200);
  }else{
    client.print("DESACIONAR\r");//responde ao transmissor
    delay(200);
  }
  client.println("Receiver\r"); //instiga uma nova transimissão
  delay(100);
}
```

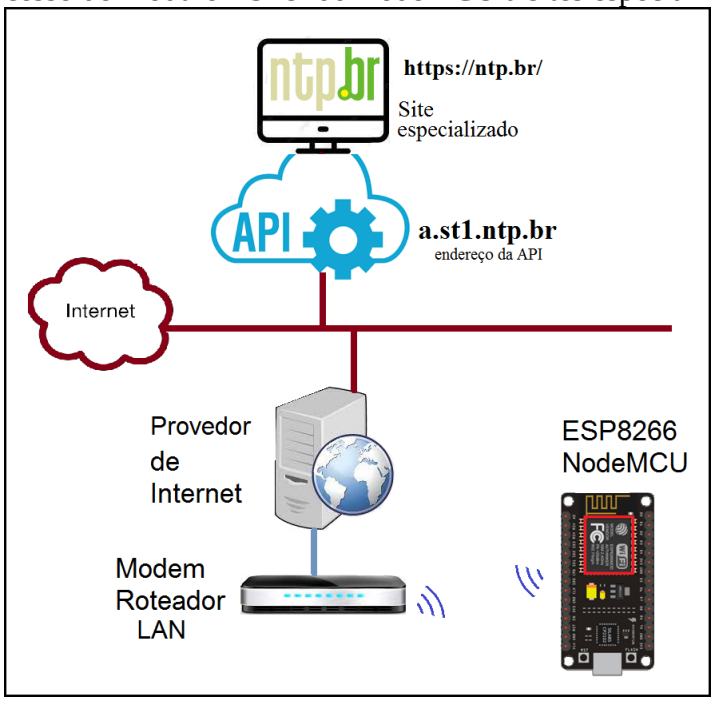
5.2 CONEXÃO DO MÓDULO ESP8266 NODEMCU COM INTERNET

A conexão do módulo ESP8266 NodeMCU à internet tem propósitos específicos, dentro do universo da internet das coisas. O acesso a sites especializados é um desses propósitos.

Os sites especializados fornecem informações e serviços que podem ser acessados por recursos computacionais, como o módulo ESP8266. Eles fornecem localização geográfica, previsão do tempo, atualização de hora global, entre outros serviços.

A conexão do módulo ao *site* especializado se dá por uma API[31] (*Application Programming Interface* – Interface de programação de aplicações que quando solicitada responde com um pacote de informações formatadas, a ser interpretada pelo módulo e utilizada como informações ou comandos. A Figura 75 apresenta a estrutura de conexão do módulo com um *site* especializado via API.

Figura 75 - Acesso do módulo ESP8266 NodeMCU a sites especializados via API



Fonte: Autor (2021)

Como um exemplo de aplicação, será mostrado o acesso de um módulo ESP8266 NodeMCU ao *site* http://ntp.br, que fornece um serviço de horário global sincronizado, através do protocolo NTP[32] (*Network Time Protocol*). As informações são acessadas em uma API, que retornará um pacote

[31] API: *Application Programming Interface*, é um conjunto de normas que possibilita a comunicação entre plataformas através de uma série de padrões e protocolos. Por meio de APIs, desenvolvedores podem criar novos softwares e aplicativos capazes de se comunicar com outras plataformas. Fonte: https://www.techtudo.com.br/listas/2020/06/o-que-e-api-e-para-que-serve-cinco-perguntas-e-respostas.ghtml

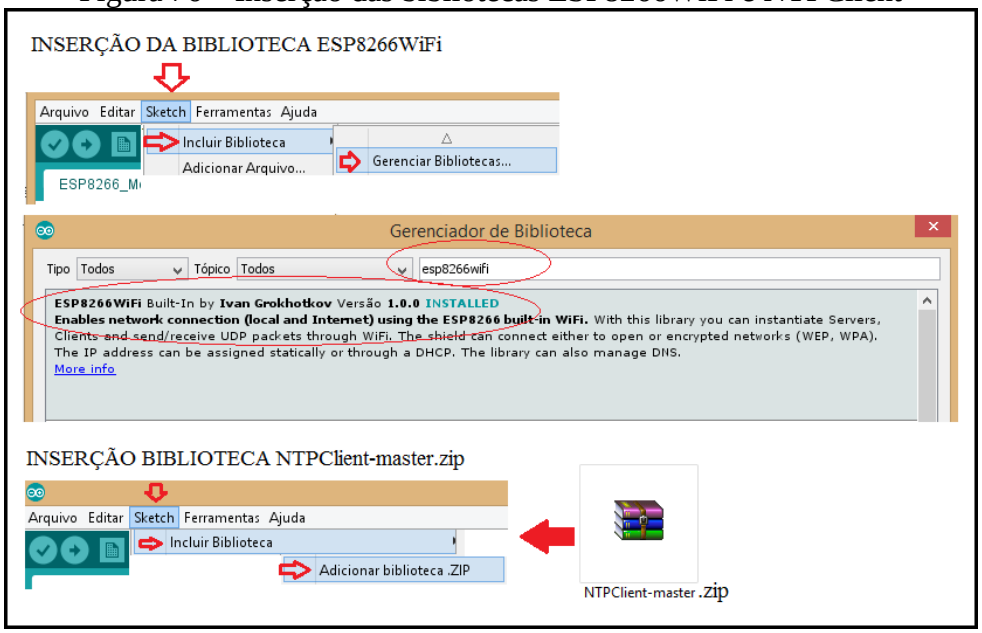
[32] NTP significa *Network Time Protocol* ou Protocolo de Tempo para Redes. É o protocolo que permite a sincronização dos relógios dos dispositivos de uma rede como servidores, estações de trabalho, roteadores e outros equipamentos a partir de referências de tempo confiáveis. Fonte: https://ntp.br/

de informações relativo ao horário mundial. Esse serviço é muito utilizado por servidores, a fim de sincronizar seus relógios.

A proposta exemplo é que o módulo busque na API do portal NTP.BR o pacote de informação relativa ao horário atual global e informe no monitor serial como um relógio, sincronizado com horário global.

Para isto, na IDE Arduino, é necessário inserir o pacote de bibliotecas "ESP8266WiFi by Ivan Grokhotkov", através das abas "Sketch", "Incluir Biblioteca" e "Gerenciar Bibliotecas...", solicitando sua instalação. Também é necessário instalar o pacote de bibliotecas "NTPClient", que deve ser baixado do *site* https://github.com/arduino-libraries/NTPClient, sendo na IDE Arduino acionada as abas "Sketch", "Incluir Biblioteca" e "Adicionar biblioteca .ZIP", indicando a pasta onde foi baixado o arquivo NTPClient-master.zip. A Figura 76 demonstra esse processo de instalação das bibliotecas.

Figura 76 – Inserção das bibliotecas ESP8266WiFi e NTPClient



Fonte: Autor (2021)

Inicialmente ser faz necessária a inclusão das bibliotecas "NTPClient.h", "ESP8266WiFi.h" e "WiFiUdp.h", através da diretiva #include.

```
#include <NTPClient.h>
#include <ESP8266WiFi.h>
#include <WiFiUdp.h> //dentro pacote ESP8266WiFi
```

Na sequência é criado o objeto do tipo WiFiUDP com o nome de "udp" para ser utilizado na instância do protocolo NTP. Em seguida, cria-se uma instância de objeto NTPClient, com nome "ntp", passando por parâmetro o tipo de conexão (udp), o endereço da API ("a.st1.ntp.br"), o fuso horário desejado em segundos (-3*3600s = -3h) e o padrão de milissegundos que compõe 1 minuto no padrão brasileiro (60000ms).

```
WiFiUDP udp;
NTPClient ntp(udp, "a.st1.ntp.br", -3 * 3600, 60000);
```

Dentro da função *setup*(), deve ser inicializada a conexão *wifi* do módulo com o modem roteador *wifi*, através da função *WiFi.begin*(), passando, por parâmetro, o nome da rede *wifi* e a senha da mesma. Na sequência se faz a verificação de conexão, monitorando a função *WiFi.status*(), que retorna a *String* WL_CONNECTED quando a conexão confirmar. Como será emitido o valor horário no monitor serial, esse deve ser inicializado com a função Serial.begin(), indicando por parâmetro a taxa de transmissão (*baud rate*) em *bits* por segundos (bit/s ou bps).

```
void setup(){
    WiFi.begin("NOME_REDE", "SENHA_REDE");
    while (WiFi.status() != WL_CONNECTED)delay(500);//aguarda conexão
    Serial.begin(115200);
```

Ainda dentro da função *setup*(), é necessário iniciar o objeto *ntp* através da função *ntp.begin*(). E, na sequência, se realiza uma atualização da informação solicitada na API, através da função *ntp.forceUpdate*().

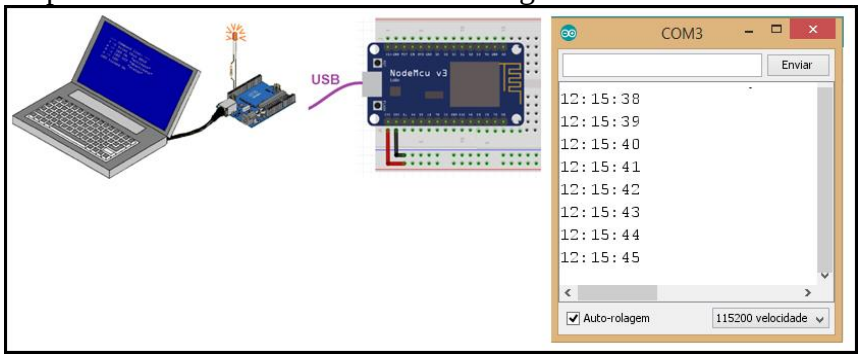
```
void setup(){
    ntp.begin();
    ntp.forceUpdate();
```

Dentro da função *loop*(), se utiliza a função *ntp.getFormattedTime()*, que retorna uma *String* com a hora global no formato hh:mm:ss, fornecido pela API. Esse valor armazenado em uma variável é impresso no monitor serial pela função *Serial.println*(), passando, por parâmetro, a variável onde foi armazenada a busca na API.

```
void loop(){
    String hora = ntp.getFormattedTime();
    Serial.println(hora);
```

A Figura 77 apresenta esse exemplo de acesso a um *site* especializado via API.

Figura 77 – Exemplo de acesso a horário sincronizado global com módulo ESP8266 NodeMCU



Fonte: Autor (2021)

O código de programa para essa aplicação é a seguinte.

```
//***RELÓGIO SINCRONIZADO COM HORÁRIO DE BRASÍLIA*******
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
//incluir biblioteca .zip de https://github.com/arduino-libraries/NTPClient
#include <NTPClient.h>
//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <ESP8266WiFi.h>
#include <WiFiUdp.h> //dentro pacote ESP8266WiFi
#define LedBoard 2 // Led Build_in D4 GPIO-2
//-------- Configurações de Wi-fi-----------
const char* ssid = "**********"; // Substitua pelo nome da rede
const char* password = "**********";    // Substitua pela senha
//-------- Configurações de relógio on-line-----------
WiFiUDP udp;
//Cria um objeto "NTP" com as configurações utilizadas no Brasil
//-3*3600 ajuste do fuso horário para Brasília(em segundos)
NTPClient ntp(udp, "a.st1.ntp.br", -3 * 3600, 60000);
String hora; // Variável que armazena o instante atual

void setup(){
  pinMode(LedBoard, OUTPUT);
  digitalWrite(LedBoard, LOW);//liga o LEDBOARD
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)delay(500);//aguarda conexão
  digitalWrite(LedBoard, HIGH);//desliga o LEDBOARD
  ntp.begin();// Inicia o protocolo
  ntp.forceUpdate();// Atualização
}
void loop(){
  //Armazena na variável hora, o horário atual.
  hora = ntp.getFormattedTime();
  Serial.println(hora);// Escreve a hora no monitor serial.
  delay(1000);         // Espera 1 segundo.
}
```

O Projeto 17 apresenta uma aplicação de um “Comando de Luzes por Chave de Tempo”, em que é lido o tempo global pela API do portal NTP.BR e comanda o acionamento de uma lâmpada, segundo o horário programado para ligar e desligar. Cabe salientar que esse projeto comanda um relé de saída que, através de seus contatos, comanda cargas. E essas cargas podem ser, além de luminárias, alimentadores para animais, condicionador de ar, sinalizador sonoro, entre outras. Enfim, qualquer aplicação em que se deseja que algo ligue em um horário e desligue em outro.

Já o Projeto 18 apresenta um “Monitorador de Previsão do Tempo para os Próximos 3 Dias”, acessando a API do portal OpenWheather, que dispõe essas informações em seu *site* na internet.

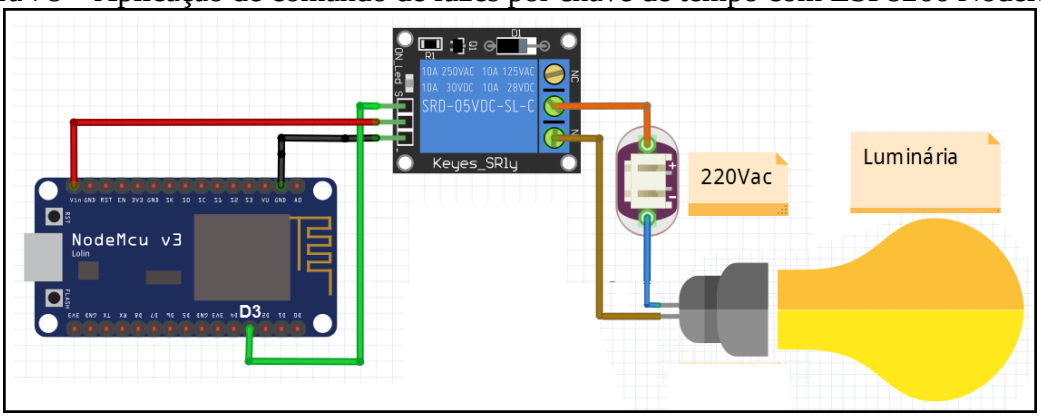
Por fim, o Projeto 19 apresenta um aplicativo de “Troca de Mensagens entre ESP8266 NodeMCU e Telegram”, onde o módulo acessa a API do aplicativo de mensagems Telegram, emitindo e recebendo mensagens, pela funcionalidade BotFather. Uma boa forma de humanos interagirem com a internet das coisas. As mensagens enviadas ao módulo podem ser interpretadas como comandos ou como informações.

PROJETO 17 – COMANDO DE LUZES POR CHAVE DE TEMPO

"O projeto consiste em um módulo ESP8266 NodeMCU, conectado a um modem/roteador wifi, que acessa o site especializado na internet NTP.BR, que através de sua API fornece o horário atual. Com essa informação, o módulo liga a lâmpada no horário programado para esse evento ocorrer e desliga a lâmpada no horário programado para esse evento ocorrer.

A Figura 78 mostra a aplicação de comando de luzes por chave de tempo, com acesso ao *site* especializado para monitoramento de tempo.

Figura 78 – Aplicação de comando de luzes por chave de tempo com ESP8266 NodeMCU



Fonte: Autor (2021)

O código de programa para realização desse projeto é o seguinte.

```
//******COMANDO DE LUZES POR UMA CHAVE DE TEMPO******
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
#include <NTPClient.h> //https://github.com/arduino-libraries/NTPClient
#include <ESP8266WiFi.h>//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <WiFiUdp.h> //dentro pacote ESP8266WiFi
#define HORA_LIGAR "01:26:00"
#define HORA_DESLIGAR "01:27:00"
#define L3 0  // L3 vermelho D3 GPIO-0
#define LedBoard 2 // Led Build_in D4 GPIO-2
const char* ssid = "**********"; // Substitua pelo nome da rede
const char* password = "**********";    // Substitua pela senha
WiFiUDP udp;
NTPClient ntp(udp, "a.st1.ntp.br", -3 * 3600, 60000);//Cria um objeto "NTP" com as configurações.utilizada no Brasil
String hora; // Variável que armazena

void setup(){
  pinMode(LedBoard, OUTPUT);
  digitalWrite(LedBoard, LOW);//liga o LEDBOARD
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)delay(500);//aguarda conexão
  digitalWrite(LedBoard, HIGH);//desliga o LEDBOARD
  ntp.begin();// Inicia o protocolo
  ntp.forceUpdate();// Atualização .
```

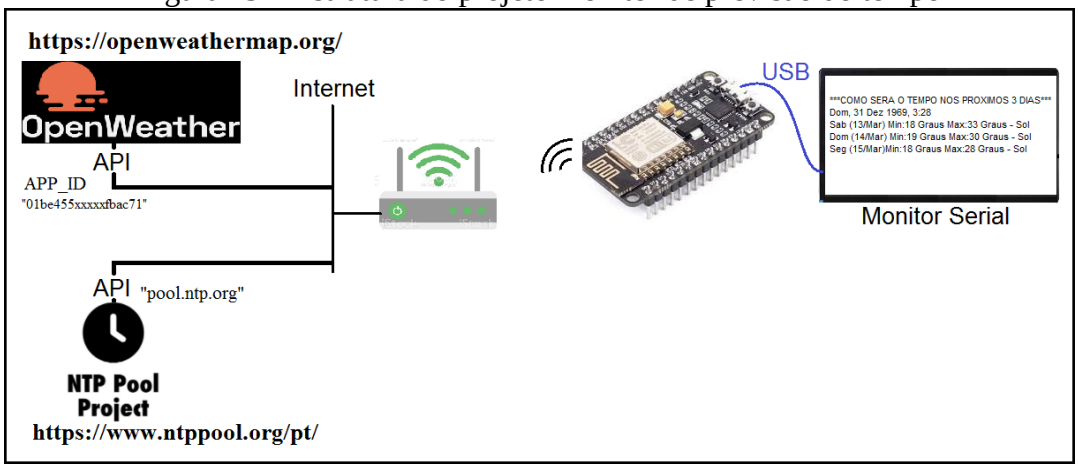
```
  pinMode(L3,OUTPUT);
}
void loop(){
  hora = ntp.getFormattedTime();
  Serial.println(hora);// Escreve a hora no monitor serial.
  if(hora == HORA_LIGAR)digitalWrite(L3,HIGH);
  if(hora == HORA_DESLIGAR)digitalWrite(L3,LOW);
  delay(1000); // Espera 1 segundo.
}
```

PROJETO 18 –PREVISÃO DO TEMPO

"O projeto consiste em um módulo ESP8266 NodeMCU, conectados a um roteador wifi, que acessa os sites OpenWeather o NTP Pool Project para obter a informação de horário global e a previsão do tempo, através de suas APIs, interpretando as informações e exibindo no canal serial a previsão do tempo para os próximos 3 dias."

Esse projeto se propõe apresentar no monitor serial a previsão de temperatura e condições climáticas para 3 dias sequentes, através do uso do *site* de internet **Open Weather**, tendo sido adaptado do artigo de Rosana Guse (GUSE, 2020). O projeto também apresenta a hora sincronizada global da previsão, obtida no *site* de internet **NTP Pool Project**. Ambos os acessos se dão através de APIs específicas, que fornecem as informações em pacotes de dados a serem interpretados. A Figura 79 apresenta a estrutura do projeto monitor de previsão do tempo, com sua conexão à internet e aos *sites* especializados.

Figura 79 – Estrutura do projeto monitor de previsão do tempo



Fonte: Autor (2021)

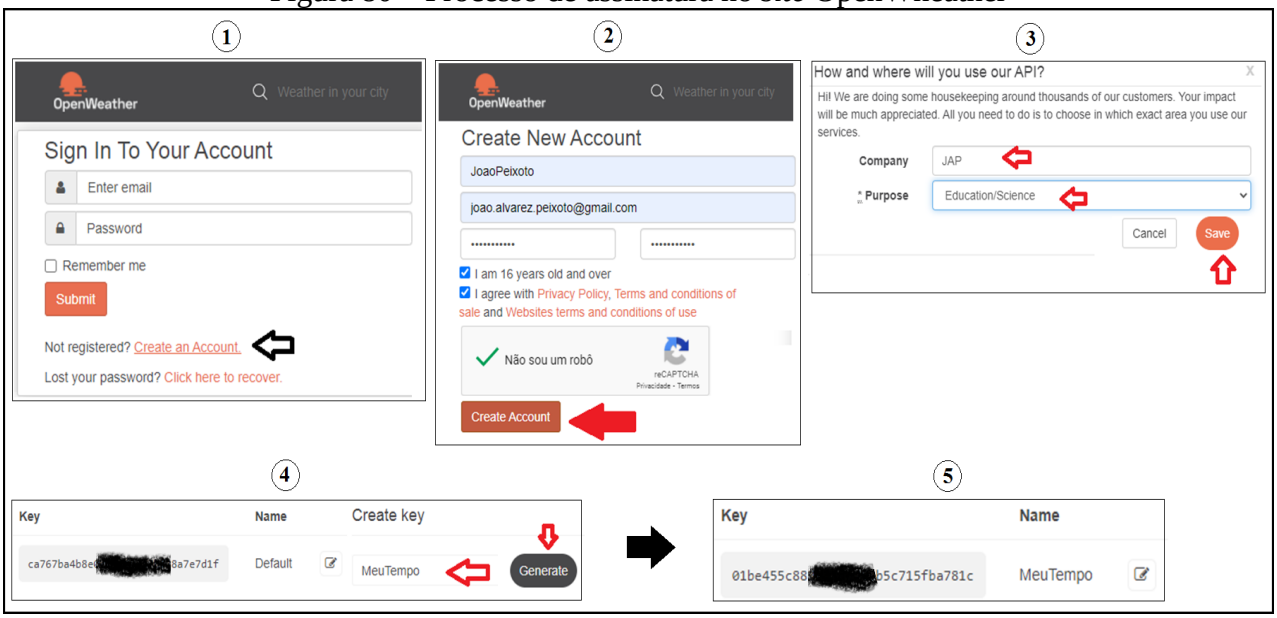
Para implementação desse projeto será necessário um cadastro de conta no portal OpenWeather, o qual disponibilizará uma chave que permitirá o uso de sua API, e que será inserida no ESP8266 NodeMCU. Diferente de outras APIs, esse *site* envia um APP_ID[33] para cada usuário cadastrado, não havendo um URL único de acesso.

Para obtenção de um APP_ID no *site* Open Weather, é necessário realizar um cadastro neste portal. Deve-se começar acessando o *site* https://home.openweathermap.org/users/sign_in, criando uma conta, seguindo os passos 1, 2 e 3 da Figura 80. Uma vez criada uma conta, faz-se necessário gerar uma chave de API, clicando na aba superior no seu nome de *login* e em 'My API Keys'. Insira um nome para sua chave e clique em 'Generate'. Sua API será gerada e um código será

[33] APP_ID é um código indicador de acesso a um aplicativo, customizado para cada usuário previamente cadastrado.

disponibilizado para ser copiado e inserido no programa do ESP8266 NodeMCU, passos 4 e 5 da Figura 80.

Figura 80 – Processo de assinatura no *site* OpenWheather

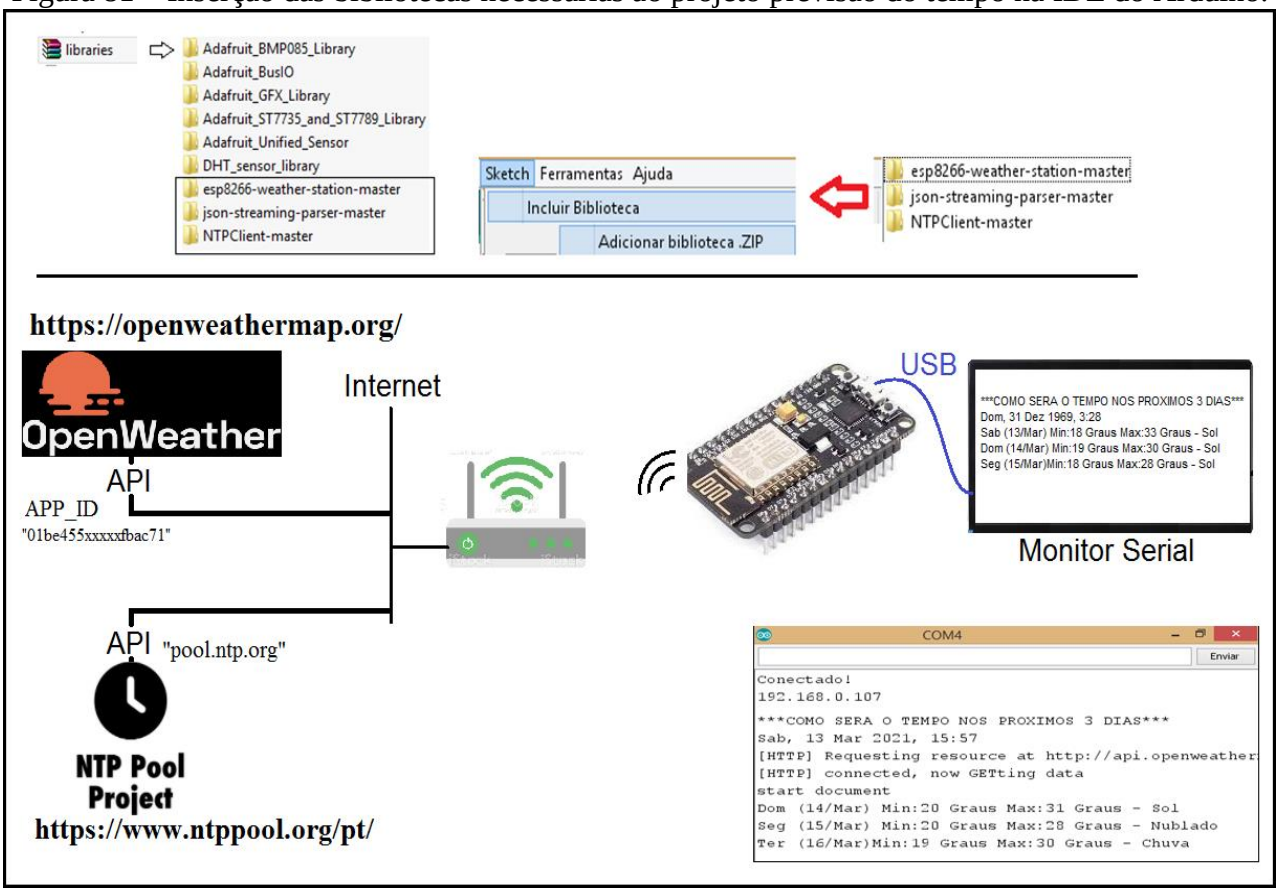


Fonte: Autor (2021)

Para acessar o *site* NTP Pool Project, não é necessário um código específico, basta acessar o endereço da sua API como "pool.ntp.org".

Como início da programação do módulo ESP8266 NodeMCU, é necessária a instalação de pacotes de bibliotecas específicas para acesso às APIs. Esses pacotes podem ser conseguidos no *link* http://img.filipeflop.com/files/download/BLU06/libraries.rar. Deve-se baixe esse arquivo e o descompactar em uma pasta conhecida, aproveitando somente os pacotes de bibliotecas: “esp8266-weather-station-master”, “json-streaming-parser-master” e “NTPClient-master”. No ambiente da IDE do Arduino, insira os pacotes de bibliotecas, clicando em “Sketch”, “Incluir Biblioteca” e “Adicionar bibliotecas.ZIP”, apontando para cada uma das pastas indicadas como bibliotecas a serem inseridas. A Figura 81 mostra esse processo.

Figura 81 – Inserção das bibliotecas necessárias ao projeto previsão do tempo na IDE do Arduino.



Fonte: Autor (2021)

Há cinco linhas de programa que deverão ser parametrizadas segundo o local de execução do projeto.

Você deve inserir o nome e a senha de acesso à rede *wifi* onde o módulo ESP8266 NodeMCU está conectado:

const char* WIFI_SSID = "NOME DA SUA REDE";
const char* WIFI_PASSWORD = "senha da sua rede";

É preciso inserir a chave de API do serviço Open Weather Map, definida na sua conta neste portal, a fim do módulo acessar esses dados na internet (https://home.openweathermap.org/api_keys):

String OPEN_WEATHER_MAP_APP_ID = "sua chave de API do serviço Open Weather Map";

Por fim, deve inserir as coordenadas de longitude e latitude da cidade da qual deseja identificar a previsão do tempo. Isso pode ser conseguido através do portal https://www.latlong.net/.

float OPEN_WEATHER_MAP_LOCATTION_LAT = -29.943760;
float OPEN_WEATHER_MAP_LOCATTION_LON = -50.993561;

Agora, é só proceder à compilação do programa e ao carregamento no módulo. Ao fim do carregamento, o módulo ESP8266 NodeMCU se conecta à internet, acessa o serviço Open Weather Map e transmite para o monitor serial (que pode ser da IDE do Arduino) os dados de previsão metereológica dos próximos três dias, atualizando a cada 10 segundos.

Por fim, se insere o trecho de códigos na IDE Arduino, conforme segue:

```
//******PREVISÃO DO TEMPO******
//Placa: "NodeMCU 1.0 (ESP-12E Module)"
#include <ESP8266WiFi.h>//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <WiFiUdp.h>//Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <NTPClient.h>//https://github.com/arduino-libraries/NTPClient
#include <time.h>
#include <Wire.h>
#include <OpenWeatherMapOneCall.h>

const char* NOME_REDE ="**********";
const char* SENHA_REDE ="**********";
WiFiClient wifiClient;
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", -10800);//Fuso -3h(-3*3600s)
//Definições Open Weather Map
//Acesse https://home.openweathermap.org/api_keys
String OPEN_WEATHER_MAP_APP_ID = "01be455c883b**********15fba781c";
//Acesse para obter os parâmetros https://www.latlong.net/
float OPEN_WEATHER_MAP_LOCATTION_LAT = -29.943760; //insira a latitude da sua cidade;
float OPEN_WEATHER_MAP_LOCATTION_LON = -50.993561; //insira a longitude da sua cidade;

String OPEN_WEATHER_MAP_LANGUAGE = "pt";
boolean IS_METRIC = true;

OpenWeatherMapOneCallData openWeatherMapOneCallData;

// Definições Dias da Semana e Mês
const String WDAY_NAMES[] = {"Dom", "Seg", "Ter", "Qua", "Qui", "Sex", "Sab"};
const String MONTH_NAMES[] = {"Jan", "Fev", "Mar", "Abr", "Mai", "Jun", "Jul", "Ago", "Set", "Out", "Nov", "Dez"};

void setup() {
  // Inicialização comunicação serial
  Serial.begin(115200);
  // Inicialização WiFi
  connectWifi();
  // Inicialização cliente NTP
  timeClient.begin();
  timeClient.update();
}

void loop() {
  Serial.println();
  Serial.println();
  Serial.print("***COMO SERA O TEMPO NOS PROXIMOS 3 DIAS***");
  data();
  horario();
  previsao();
  delay(10000);
}
void connectWifi() {
  WiFi.begin(NOME_REDE,SENHA_REDE);
  Serial.print("Conectando...");
  Serial.println(NOME_REDE);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("Conectado!");
  Serial.println(WiFi.localIP());
  Serial.println();
  delay(7000);
```

```
}
void data(){
  String weekDay = WDAY_NAMES[timeClient.getDay()];
  timeClient.update();
  Serial.println("");
  Serial.print(weekDay);//dia da semana
  Serial.print(", ");
  unsigned long epochTime = timeClient.getEpochTime();
  struct tm*ptm = gmtime ((time_t *)&epochTime);
  int monthDay = ptm->tm_mday;
  Serial.print(monthDay);// dia do mês
  Serial.print(" ");
  int currentMonth = ptm->tm_mon+1;
  String currentMonthName = MONTH_NAMES[currentMonth-1];
  int currentYear = ptm->tm_year+1900;
  Serial.print(currentMonthName);// nome do mês
  Serial.print(" ");
  Serial.print(currentYear);// nome do ano
}
void horario(){
  int currentHour = timeClient.getHours();
  int currentMinute = timeClient.getMinutes();
  Serial.print(", ");
  Serial.print(currentHour);//hora
  Serial.print(":");
  if(currentMinute<10){Serial.print("0");}
  Serial.print(currentMinute);//minuto
}
void previsao() {
  Serial.println("");//para colocar cursor na nova linha
  OpenWeatherMapOneCall *oneCallClient = new OpenWeatherMapOneCall();
  oneCallClient->setMetric(IS_METRIC);
  oneCallClient->setLanguage(OPEN_WEATHER_MAP_LANGUAGE);
  long executionStart = millis();
oneCallClient>update(&openWeatherMapOneCallData,OPEN_WEATHER_MAP_APP_ID,OPEN_WEATHER_MAP
_LOCATTION_LAT,OPEN_WEATHER_MAP_LOCATTION_LON);
  delete oneCallClient;
  oneCallClient = nullptr;
  time_t time;
  time_t observationTimestamp1 = openWeatherMapOneCallData.daily[1].dt;
  struct tm* timeInfo1;
  timeInfo1 = localtime(&observationTimestamp1);
  Serial.print(WDAY_NAMES[timeInfo1->tm_wday]);
  Serial.print(" (");
  Serial.print(timeInfo1->tm_mday);
  Serial.print("/");
  Serial.print(MONTH_NAMES[timeInfo1->tm_mon]);
  Serial.print(") ");
  Serial.print("Min:");
  int tmin1 = openWeatherMapOneCallData.daily[1].tempMin;
  Serial.print(tmin1);
  Serial.print(" Graus ");
  Serial.print("Max:");
  int tmax1 = openWeatherMapOneCallData.daily[1].tempMax;
  Serial.print(tmax1);
  Serial.print(" Graus - ");
  String cond1 = openWeatherMapOneCallData.daily[1].weatherMain;
  if (cond1 == "Thunderstorm"){Serial.print("Tempestade");}
  else if (cond1 == "Rain"){Serial.print("Chuva");}
  else if(cond1 == "Drizzle"){Serial.print("Pouca Chuva");}
  else if(cond1 == "Clouds"){Serial.print("Nublado");}
  else if(cond1 == "Clear"){Serial.print("Sol");}
  time_t observationTimestamp2 = openWeatherMapOneCallData.daily[2].dt;
  struct tm* timeInfo2;
  timeInfo2 = localtime(&observationTimestamp2);
```

```
  Serial.println("");
  Serial.print(WDAY_NAMES[timeInfo2->tm_wday]);
  Serial.print(" (");
  Serial.print(timeInfo2->tm_mday);
  Serial.print("/");
  Serial.print(MONTH_NAMES[timeInfo2->tm_mon]);
  Serial.print(") ");
  Serial.print("Min:");
  int tmin2 = openWeatherMapOneCallData.daily[2].tempMin;
  Serial.print(tmin2);
  Serial.print(" Graus ");
  Serial.print("Max:");
  int tmax2 = openWeatherMapOneCallData.daily[2].tempMax;
  Serial.print(tmax2);
  Serial.print(" Graus - ");
  String cond2 = openWeatherMapOneCallData.daily[2].weatherMain;
  if (cond2 == "Thunderstorm"){Serial.print("Tempestade");}
  else if (cond2 == "Rain"){Serial.print("Chuva");}
  else if(cond2 == "Drizzle"){Serial.print("Pouca Chuva");}
  else if(cond2 == "Clouds"){Serial.print("Nublado");}
  else if(cond2 == "Clear"){Serial.print("Sol");}
  time_t observationTimestamp3 = openWeatherMapOneCallData.daily[3].dt;
  struct tm* timeInfo3;
  timeInfo3 = localtime(&observationTimestamp3);
  Serial.println("");
  Serial.print(WDAY_NAMES[timeInfo3->tm_wday]);
  Serial.print(" (");
  Serial.print(timeInfo3->tm_mday);
  Serial.print("/");
  Serial.print(MONTH_NAMES[timeInfo3->tm_mon]);
  Serial.print(")");
  Serial.print("Min:");
  int tmin3 = openWeatherMapOneCallData.daily[3].tempMin;
  Serial.print(tmin3);
  Serial.print(" Graus ");
  Serial.print("Max:");
  int tmax3 = openWeatherMapOneCallData.daily[3].tempMax;
  Serial.print(tmax3);
   Serial.print(" Graus - ");
    String cond3 = openWeatherMapOneCallData.daily[3].weatherMain;
  if (cond3 == "Thunderstorm"){Serial.print("Tempestade");}
  else if (cond3 == "Rain"){Serial.print("Chuva");}
  else if(cond3 == "Drizzle"){Serial.print("Pouca Chuva");}
  else if(cond3 == "Clouds"){Serial.print("Nublado");}
  else if(cond3 == "Clear"){Serial.print("Sol");}
}
```

5.3 APLICAÇÃO CLIENTE- SERVIDOR (*WEB SERVER* HTTP)

Uma aplicação *Web Server*[34] consiste de um método de comunicação cliente-servidor[35], que utiliza a internet como meio de propagação de seus sinais de comunicação. Mas não necessariamente a aplicação cliente-servidor precisa da internet, há possibilidade de a comunicação ocorrer dentro de uma rede LAN[36], de forma local.

O princípio de uma comunicação cliente-servidor parte de uma estação cliente que fará uma requisição a uma estação servidora que a responderá, dentro de uma rede local ou global, através do protocolo HTTP (*Hipertext Transfer Protocol*), que possibilita acesso a outras estações através de sua URL[37] (*Uniform Resource Locator*), permitindo a localização dos recursos pelo seu endereço na internet.

Quando o cliente realiza uma requisição http através de um *browser* (navegador de internet), ele envia um comando *handle* para um servidor, informando seu endereço TCP/IP ou sua URL. O servidor recebe esse comando e responde com sua oferta para o cliente, em um pacote de dados codificados na linguagem HTML, onde o cliente ao receber essa resposta, traduz os elementos HTML[38] em uma tela formatada do navegador.

A Figura 82 apresenta a estrutura de comunicação cliente-servidor, com a proposta de inserção do módulo ESP8266 NodeMCU como servidor. Nela se observa que o cliente realizou uma requisição para o servidor, indicando seu IP como 192.168.0.112 no navegador de internet. Essa informação foi reconhecida pelo servidor e respondida com um arquivo tipo html, que foi interpretado pelo navegador do cliente e exibido em sua área de trabalho. Assim, se percebe claramente que, quando se acessa uma página de internet por uma estação cliente, essa página não está nessa estação, apesar de estar sendo exibida por ela. A página está alocada no servidor, que forneceu seus códigos e seus serviços para serem exibidos ao cliente na forma de comandos *handle*, para que sejam solicitados. É como se o servidor fosse um "garçom" de um restaurante e o cliente pedisse a ele o cardápio disponível para o dia.

---

[34] Web Server é uma estrutura de cliente-servidor através do meio internet. Fonte: https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/What_is_a_web_server.

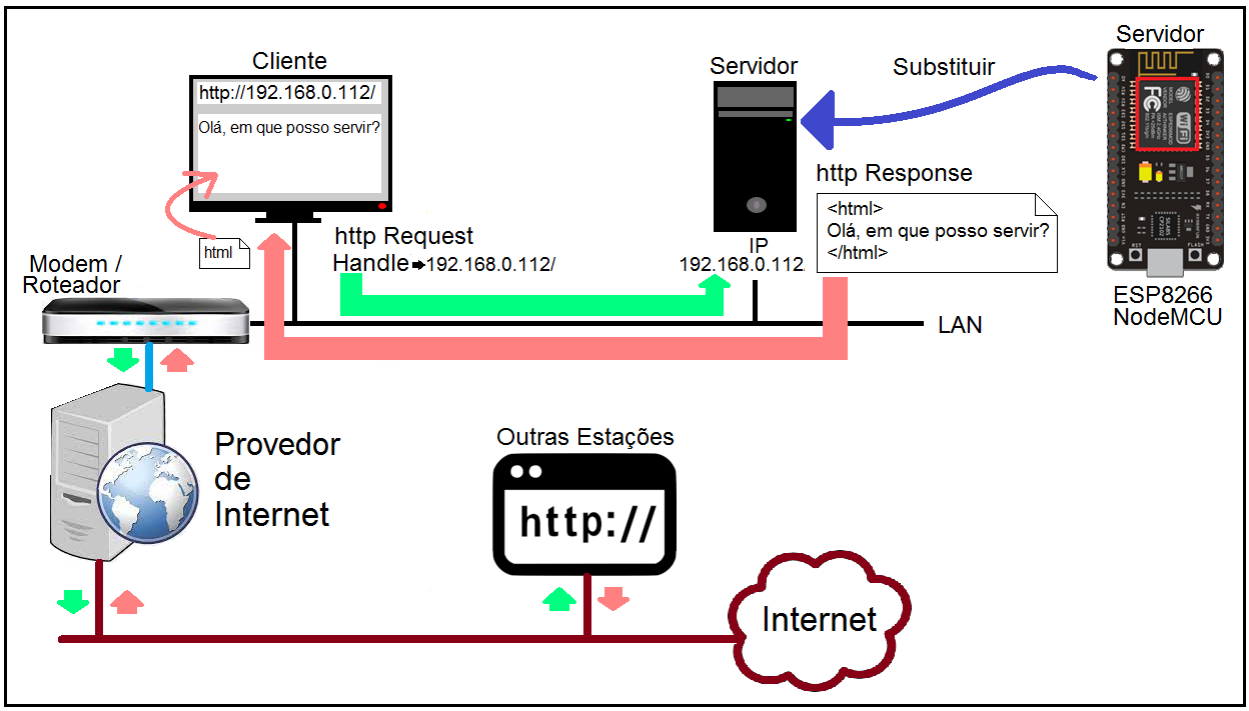
[35] Cliente-servidor é um método de comunicação entre estações em uma rede LAN ou internet, onde o cliente solicita ao servidor os serviços ou informações que este pode lhe oferecer. Fonte: https://pt.wikipedia.org/wiki/Modelo_cliente%E2%80%93servidor.

[36] LAN é uma rede de dados para estações computacionais ligadas a um roteador, responsável pela distribuição de informações. Fonte: https://helpdigitalti.com.br/o-que-e-uma-rede-lan-e-uma-rede-wan/.

[37] URL é o mesmo que endereço *web*, o texto que você digita na barra de endereços de seu navegador para acessar determinada página ou serviço. Fonte: https://tecnoblog.net/312185/o-que-e-url/.

[38] Elementos HTML são pseudocódigos que trazem um significado para o aplicativo que o irá traduzir. Por exemplo: o elemento indica que todos os caracteres a partir dele serão exibidos em fonte negrita, até que se encontre o elemento . Fonte: https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element.

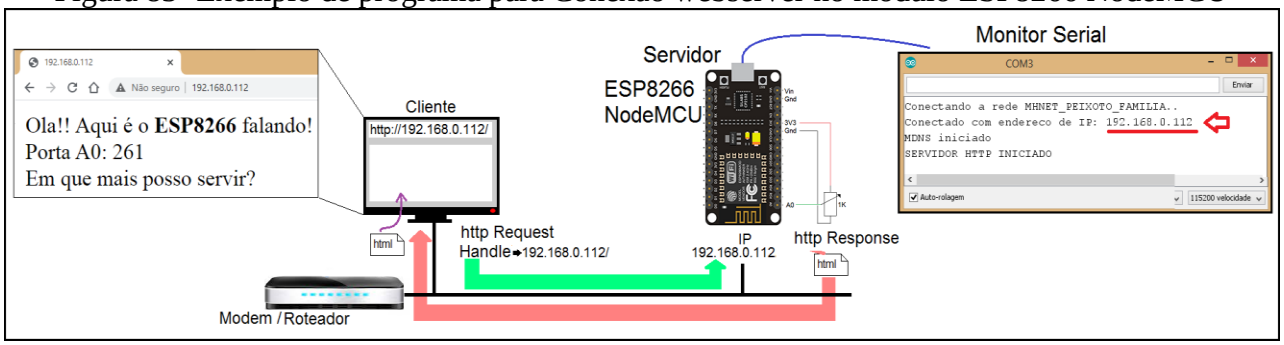
Figura 82 - Estrutura de uma comunicação cliente-servidor do módulo ESP8266 NodeMCU na internet



Fonte: Autor (2021)

Como exemplo dessa aplicação, se propõe aqui uma aplicação cliente-servidor, onde um módulo ESP8266 NodeMCU atua como um servidor, que fornece o valor do canal analógico A0 (0 a 1023), onde está conectado um potenciômetro. Uma vez conectado na rede LAN, pelo monitor serial é possível verificar o endereço IP que foi definido pelo roteador ao módulo. Em um navegador de internet, a partir de um computador conectado a esta rede, se faz a requisição http para o endereço IP do módulo, recebendo como resposta um texto de mensagem e o valor do canal analógico lido. Isso se repete a cada solicitação do cliente, apertando na tela F5 (atualizar navegação). A Figura 83 apresenta essa estrutura de conexão cliente-servidor.

Figura 83- Exemplo de programa para Conexão webserver no módulo ESP8266 NodeMCU



Fonte: Adaptado de Guimarães (2018)

O código para execução desse exemplo se inicia na IDE Arduino com a inserção das bibliotecas “ESP8266WiFi.h” (que pode ser encontrada no gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"), “WiFiClient.h”, “ESP8266WebServer.h” e “ESP8266mDNS.h”, pelo uso da diretiva #include.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
```

Para conexão ao roteador *wifi* é necessário dentro da função *setup*() iniciar o módulo *wifi* como uma estação, através da função e parâmetro *WiFi.mode(WIFI_STA).* Após, se deve inicializar o módulo *wifi*, com a função *WiFi.begin(),* passando os parâmetros de nome e senha da rede LAN a ser conectada. Então, se faz a verificação de conexão, monitorando a função *WiFi.status*(), que retorna a *String* WL_CONNECTED quando a conexão confirmar. E como será necessário conhecer o IP do módulo servidor, que é definido pelo roteador quando se dá a conexão, se configura aqui o módulo serial com a função Serial.begin() em 115.200bps e apresenta no monitor serial o IP que foi conectado, com a função *WiFi.localIP*(), que retorna uma *String* com o endereço de IP atribuído.

```
void setup(void){
   WiFi.mode(WIFI_STA);
   WiFi.begin(NOME_REDE, SENHA_REDE);
   while (WiFi.status() != WL_CONNECTED) delay(500);
   Serial.begin(115200);
   Serial.println(WiFi.localIP());
```

Para configuração do módulo como um servidor, inicialmente, deve-se criar um objeto do tipo ESP8266WebServer, com nome de “server” e com parâmetro 80, relativo à porta de conexão padrão para *web server*. Dentro da função *setup*(), é inicializado o módulo MDNS, com a função *MDNS.begin*(), que retorna um valor booleano verdadeiro (*TRUE*) se a inicialização ocorreu. Após isso, são ativados no servidor os métodos de tratamento de eventos *handleRoot* e *handleNotFond*, que são funções para reconhecer e dar respostas às requisições do cliente. Por fim, é ativado o módulo servidor com a função *server.begin*().

```
ESP8266WebServer server(80);
setup(void){
   if (MDNS.begin("esp8266"))Serial.println("MDNS iniciado");
   server.on("/", handleRoot);
   server.onNotFound(handleNotFound);
   server.begin();
```

Na função *loop*() será chamada a função *server.handleClient(),* para ficar monitorando se chegou alguma mensagem do cliente.

```
void loop(void){
```

```
server.handleClient();
```

Caso uma requisição chegue do cliente, uma das sub-rotinas será chamada: *handleRoot*() ou *handleNotFound*().

Uma requisição tratada por *handleRoot*() é uma requisição com o URL ou IP na sua pasta raiz, ou seja, sem nenhum complemento. Como exemplo: http://192.168.0.112/ é uma pasta raiz do host, pois não há complementos. Como é esta requisição que se está esperando, se monta uma *String* com os caracteres e elementos html e se envia resposta com a função *server.send*(), sendo os parâmetros 200 para indicar que a solicitação foi atendida com sucesso, o comando "text/html" e a *String* de resposta.

```
void handleRoot(){
  String textoHTML;//texto que será enviado
  textoHTML = "Ola!! Aqui &eacute; o <b>ESP8266</b> falando! <br>";//
&eacute= "é"
  textoHTML += "Porta A0: ";
  textoHTML += analogRead(A0);
  textoHTML += "<br> Em que mais posso servir?";
  server.send(200, "text/html", textoHTML);
```

Uma requisição tratada por *handleNotFound*() significa uma requisição com o URL ou IP com complementos que não estão previstos para esse programa. Afinal, há possibilidade de solicitar os complementos, para terem solicitações diversificadas no serviço prestado. Como exemplo: http://192.168.0.112/ligar/ é uma pasta raiz do *host* com um complemento "ligar/", que não está previsto no programa. Nesse caso é necessário responder para o cliente, com o código 404, caso contrário, a conexão se desfaz. Isso é realizado pela função *server.send(),* passando os parâmetros 404, indicando página não encontrada, o comando "text/plain" e a mensagem com a *String* de resposta.

```
void handleNotFound(){
    String message = "Erro 404, endereco desconhecido\n";
    message += "URI: ";
    message += server.uri();
    server.send(404, "text/plain", message);
```

O código completo para o exemplo é o seguinte:

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#define LED_BUILD 2 //D4 GPIO-2 Led Onboard com lógica invertida
const char* ssid = "**************";   //nome da rede wifi
const char* password = "**************";//senha da rede wifi
ESP8266WebServer server(80);   //porta de atendimento dos chamados web, padrão

//Este método será executado quando ouver uma requisição do cliente
void handleRoot(){
```

```
  digitalWrite(LED_BUILD,LOW);   //liga o led onboard
  String textoHTML;   //texto que será enviado
  textoHTML = "Ola!! Aqui &eacute; o <b>ESP8266</b> falando! <br>";   // &eacute = "é"
  textoHTML += "Porta A0: ";
  textoHTML += analogRead(A0);
  textoHTML += "<br> Em que mais posso servir?";
  server.send(200, "text/html", textoHTML);
  digitalWrite(LED_BUILD, HIGH);  //apaga o led onboard
}

//este método será executado se endereço for desconhecido
void handleNotFound(){
  digitalWrite(LED_BUILD,LOW);  //liga o led onboard
  String message = "Erro 404, endereco desconhecido\n";
  message += "URI: ";
  message += server.uri();
  server.send(404, "text/plain", message);
  digitalWrite(LED_BUILD, HIGH);  //apaga o led onboard
}

void setup(void){
  pinMode(LED_BUILD, OUTPUT);
  digitalWrite(LED_BUILD, LOW);  //liga o led onboard
  Serial.begin(115200);          //configura o canal seria com baude rate
  WiFi.mode(WIFI_STA);      //configura o modo de conexão wifi
  WiFi.begin(ssid, password); //configura rede e senha
  Serial.println("");
  Serial.print("Conectando a rede ");
  Serial.print(ssid);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado com endereco de IP: ");
  Serial.println(WiFi.localIP());
  if (MDNS.begin("esp8266"))Serial.println("MDNS iniciado");
  server.on("/", handleRoot); //ativa a resposta para página na raiz
  server.onNotFound(handleNotFound); //ativa a resposta para página não encontrada
  server.begin(); //ativa o servidor
  Serial.println("SERVIDOR HTTP INICIADO");
  digitalWrite(LED_BUILD, HIGH);//apaga o led onboard
}

void loop(void){
  server.handleClient(); //fica sempre ouvindo uma conexão externa.
}
```

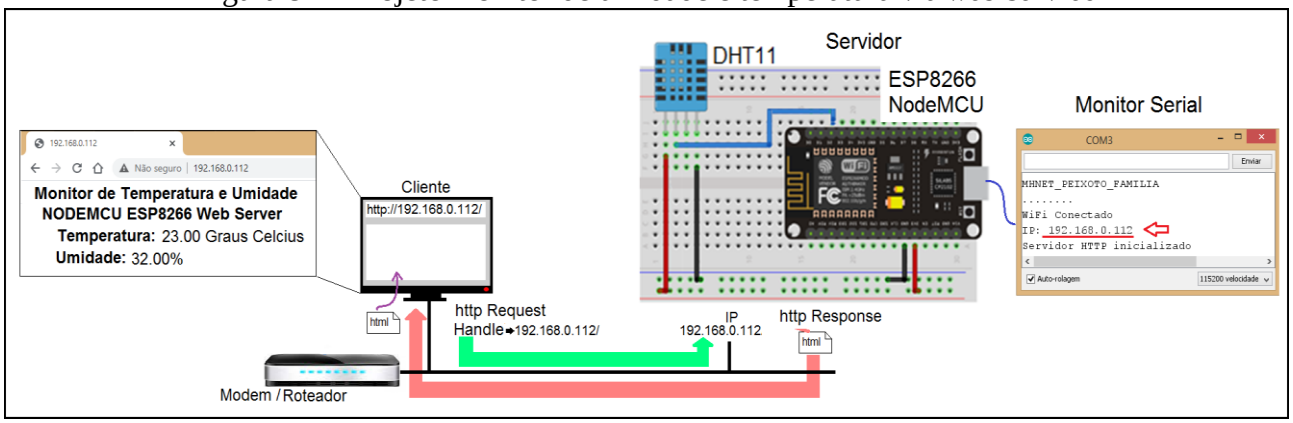
O Projeto 19 apresenta um medidor de temperatura e umidade ambiente, com um sensor DHT11 monitorado por um módulo ESP8266 NodeMCU, sendo as informações disponibilizadas como um serviço em uma estrutura *web server*.

PROJETO 19 – MEDIDOR DE UMIDADE E TEMPERATURA POR WEBSERVER

"O projeto consiste em um módulo ESP8266 NodeMCU, conectados a um modem wifi, que acessa a rede internet, disponibilizando o serviço de monitoramento de umidade e temperatura, como servidor. O cliente pode solicitar uma requisição, que receberá uma página web com os dados desejados"

O projeto medidor de umidade e temperatura por *web server* fornece como serviço os dados na forma de uma página html, ao ser requisitado por um cliente (VIANA, 2020). Utiliza o sensor DHT11 que provê a medição das duas grandezas. A Figura 84 apresenta a estrutura e telas desse projeto.

Figura 84 – Projeto monitor de umidade e temperatura via *web service*



Fonte: Adaptado de Viana (2020)

O código para execução desse exemplo se inicia na IDE Arduino com a inserção das bibliotecas "ESP8266WiFi.h" (que pode ser encontrada no gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"), "ESP8266WebServer.h" e "DHT.h", pelo uso da diretiva #include. O pacote de biblioteca DHT-sensor-library está disponibilizado no repositório https://drive.google.com/file/d/16brDLNmOVzmQs6sFh-_CAMivQu-cn6kg/view. Uma vez procedido o *download* deste pacote de biblioteca, a sua inclusão na IDE Arduino se dá ao clicar em: "Sketch", "Incluir Biblioteca", "Adicionar Biblioteca.zip", indicando o caminho onde foi salva.

O código completo do projeto monitor de temperatura e umidade por webservice é o seguinte:

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include<ESP8266WiFi.h> //Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include<ESP8266WebServer.h> //Biblioteca para o ESP funcionar como servidor
#include <DHT.h> //Biblioteca para funcionamento do sensor de temperatura e umidade DHT11
const char* ssid = "MHNET_PEIXOTO_FAMILIA";  // Rede WiFi
const char* password = "peixotofamilia";  //Senha da Rede WiFi
ESP8266WebServer server(80); //server na porta 80
#define DHTPIN 4  //Pino D2(GPIO-4) conectado ao DHT11
#define DHTTYPE DHT11 //Tipo do sensor DHT11
```

```
DHT dht(DHTPIN, DHTTYPE); //Inicializando o objeto dht do tipo DHT passando como parâmetro o pino (DHTPIN) e o tipo do sensor (DHTTYPE)
float temperatura; //variável para armazenar a temperatura
float umidade; //Variável para armazenar a umidade

void setup() {
  Serial.begin(115200); //Inicializa a comunicação serial
  delay(50); // Intervalo para aguardar a estabilização do sistema
  dht.begin(); //Inicializa o sensor DHT11
  Serial.println("Conectando a Rede: "); //Imprime na serial a mensagem
  Serial.println(ssid); //Imprime na serial o nome da Rede Wi-Fi
  WiFi.begin(ssid, password); //Inicialização da comunicação Wi-Fi
  while (WiFi.status() != WL_CONNECTED) { //Enquanto estiver aguardando status da conexão
    delay(500);
    Serial.print("."); //Imprime pontos
  }
  Serial.println("");
  Serial.println("WiFi Conectado");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP()); //Função para exibir o IP da ESP
  server.on("/", handle_OnConnect); //Servidor recebe uma solicitação HTTP - chama a função handle_OnConnect
  server.onNotFound(handle_NotFound); //Servidor recebe uma solicitação HTTP não especificada - chama a função handle_NotFound
  server.begin(); //Inicializa o servidor
  Serial.println("Servidor HTTP inicializado");
}

void loop(){
  server.handleClient(); //Fica monitorando chegada de requisição do cliente
}

void handle_OnConnect(){
  temperatura = dht.readTemperature(); //Realiza a leitura da temperatura
  umidade = dht.readHumidity(); //Realiza a leitura da umidade
  server.send(200, "text/html", EnvioHTML(temperatura, umidade)); //Envia as informações usando o código 200, especifica o conteúdo como "text/html" e chama a função EnvioHTML
}

void handle_NotFound(){//Função para lidar com o erro 404
  server.send(404, "text/plain", "Pagina desconhecida");
}

String EnvioHTML(float Temperaturastat, float Umidadestat) { //Exibindo a página da web em HTML
  String ptr = "<!DOCTYPE html> <html>\n"; //Indica o envio do código HTML
  ptr += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
  ptr += "<meta http-equiv='refresh' content='2'>";//Atualizar browser a cada 2 segundos
  ptr += "<link href=\"https://fonts.googleapis.com/css?family=Open+Sans:300,400,600\" rel=\"stylesheet\">\n";
  ptr += "<title>Monitor de Temperatura e Umidade</title>\n"; //Define o título da página
  //Configurações de fonte do título e do corpo do texto da página web
  ptr += "<style>html { font-family: 'Open Sans', sans-serif; display: block; margin: 0px auto; text-align: center;color: #000000;}\n";
  ptr += "body{margin-top: 50px;}\n";
  ptr += "h1 {margin: 50px auto 30px;}\n";
  ptr += "h2 {margin: 40px auto 20px;}\n";
  ptr += "p {font-size: 24px;color: #000000;margin-bottom: 10px;}\n";
  ptr += "</style>\n";
  ptr += "</head>\n";
  ptr += "<body>\n";
  ptr += "<div id=\"webpage\">\n";
  ptr += "<h1>Monitor de Temperatura e Umidade</h1>\n";
  ptr += "<h2>NODEMCU ESP8266 Web Server</h2>\n";
  //Exibe as informações de temperatura e umidade na página web
  ptr += "<p><b>Temperatura: </b>";
  ptr += (float)Temperaturastat;
  ptr += " Graus Celsius</p>";
```

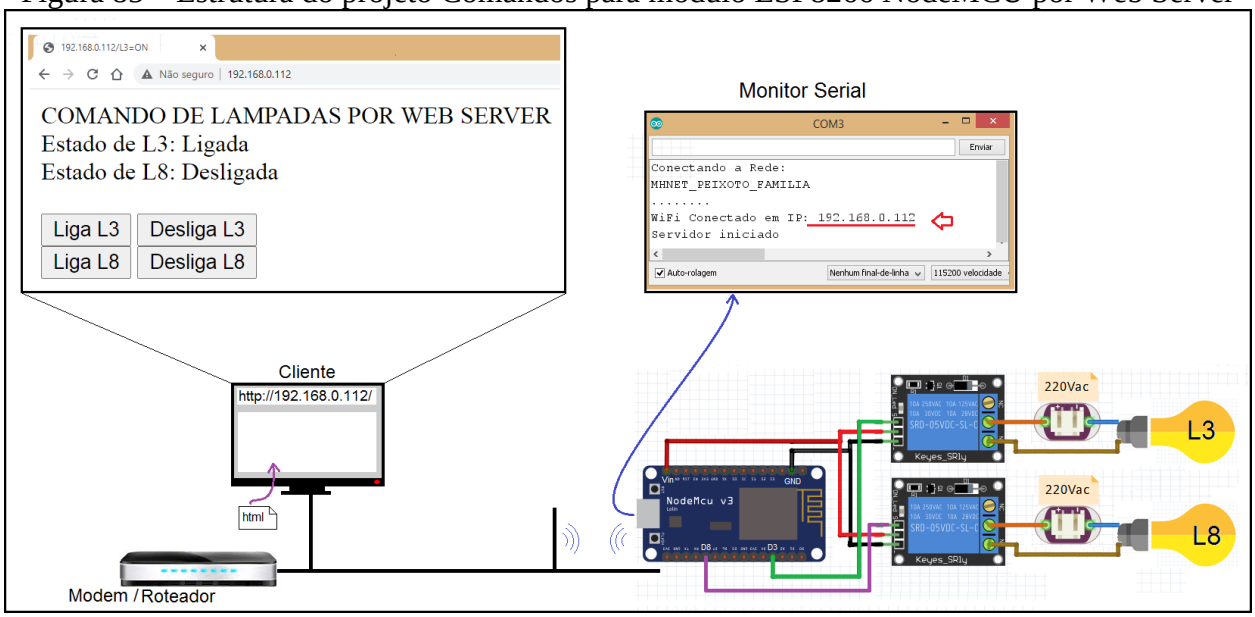
```
  ptr += "<p><b>Umidade: </b>";
  ptr += (float)Umidadestat;
  ptr += " %</p>";
  ptr += "</div>\n";
  ptr += "</body>\n";
  ptr += "</html>\n";
  return ptr;
}
```

PROJETO 20 – COMANDOS PARA MÓDULO ESP8266 NODEMCU POR WEB SERVER

"O projeto consiste em um módulo ESP8266 NodeMCU, conectado a um modem wifi, que se oferta na rede internet como um servidor, podendo ser acessado por um cliente, que receberá uma página web com a infomação do estado de duas lâmpadas e a possibilidade de ligá-las ou desligá-las, via web service."

Este projeto inicialmente se propõe ao comando de lâmpadas, mas pode comandar o acionamento de alarme residencial, atuadores para irrigação de jardim, alimentadores de animais domésticos, entre outros (ROCHA, 2017). A Figura 85 apresenta a estrutura e telas do projeto.

Figura 85 – Estrutura do projeto Comandos para modulo ESP8266 NodeMCU por Web Server



Fonte: Autor (2021)

O que se pode destacar do código são as seguintes linhas:

a) if(request.indexOf("/L3=ON") != -1){//ação}: identifica se a requisição do cliente possui o complemento"L3=ON", "http://192.168.0.112/**L3=ON**";

b) client.print("Ligada"): responde ao cliente a *string* "Ligada", a ser exibida na página na posição atual do cursor;

c) client.println("<button>Liga L3 </button>"): cria um botão na página web de resposta, com o texto "Liga L3" e quando pressionado, envia uma requisição do cliente com o endereço IP do servidor e o complemento "/L3=ON", "http://192.168.0.112/**L3=ON**".

O código completo do projeto é apresentado a seguir.

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#define L3 0  //Pino D3 (GPIO-0)
#define L8 15 //Pino D8 (GPIO-15)
const char* ssid = "**********"; //NOME da Rede LAN
const char* password = "**********";    //SENHA da rede LAN
WiFiServer server(80); //Porta padrão 80
WiFiClient client;
bool estadoL3 = LOW;
bool estadoL8 = LOW;

void setup() {
Serial.begin(115200);
pinMode(L3, OUTPUT);
pinMode(L8, OUTPUT);
// Comunicação com a rede WiFi
Serial.println("");
Serial.println("Conectando a Rede: "); //Imprime na serial a mensagem
Serial.println(ssid); //Imprime na serial o nome da Rede Wi-Fi
WiFi.begin(ssid, password); //Inicialização da comunicação Wi-Fi
while (WiFi.status() != WL_CONNECTED) { //Enquanto estiver aguardando status da conexão
  delay(500);
  Serial.print("."); //Imprime pontos
}
Serial.println("");
Serial.print("WiFi Conectado em IP: ");
Serial.println(WiFi.localIP()); //Função para exibir o IP
server.begin(); // Comunicação com o servidor
Serial.println("Servidor iniciado");
}

void loop() {// Verificação se o cliente está conectado
  client = server.available();
  if (!client)return;// Verifica se o cliente está conectado ao servidor
  while(!client.available())delay(1);// Espera até o cliente enviar dados
  String request = client.readStringUntil('\r');// Ler a primeira linha do pedido
  client.flush();
//Identifica o comando e atua
  if(request.indexOf("/L3=ON") != -1){digitalWrite(L3, HIGH);estadoL3 = HIGH;}
  if(request.indexOf("/L3=OFF") != -1){digitalWrite(L3, LOW);estadoL3 = LOW;}
  if(request.indexOf("/L8=ON") != -1){digitalWrite(L8, HIGH);estadoL8 = HIGH;}
  if(request.indexOf("/L8=OFF") != -1){digitalWrite(L8, LOW);estadoL8 = LOW;}
  paginaWEB();// Envia a página HTML
}
void paginaWEB(){//Resposta página HTML para cliente
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println("");
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");
  client.println("COMANDO DE LAMPADAS POR WEB SERVER");
  client.println("<br>");
  client.print("Estado de L3: ");
  if(estadoL3 == HIGH) {
    client.print("Ligada");
  }else{
    client.print("Desligada");
  }
  client.println("<br>");
  client.print("Estado de L8: ");
  if(estadoL8 == HIGH){
    client.print("Ligada");
  }else{
    client.print("Desligada");
  }
```

```
  client.println("<br><br>");
  client.println("<a href=\"/L3=ON\"\"><button>Liga    L3 </button></a>"); //Cria botão ligar L3
  client.println("<a href=\"/L3=OFF\"\"><button>Desliga L3 </button></a><br />"); //Cria botão desligar L3
  client.println("<a href=\"/L8=ON\"\"><button>Liga    L8 </button></a>"); //Cria botão ligar L8
  client.println("<a href=\"/L8=OFF\"\"><button>Desliga L8 </button></a><br />"); //Cria botão desligar L8
  client.println("</html>");
  delay(1);
}
```

5.4 APLICAÇÃO *PUBLISH–SUBSCRIBE* (MQTT)

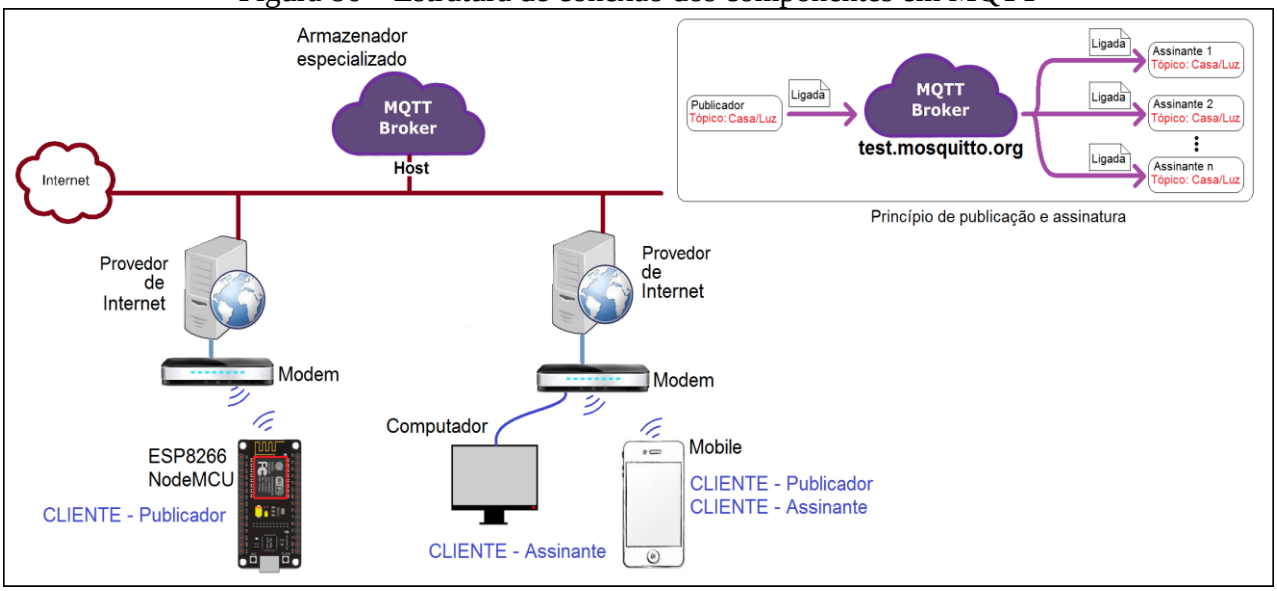
O protocolo de comunicação *publish-subscribe* (publicação/assinatura) permite aos dispositivos atuarem na internet das coisas (IoT), trocando mensagens em ambiente de internet e valendo-se da mobilidade que essa rede disponibiliza. Nesse protocolo há três componentes que atuam em conjunto. O cliente publicador (*publisher*), o cliente assinante (*subscriber*) e o agente de recebimento e entrega de mensagens (*broker*) (LOCATELLI, 2020a).

O cliente publicador é o componente que se propõe a fornecer uma informação de valor ou mensagem para todos que quiserem essa informação. Para isso, ele a dispõe em um agente de recebimento e entrega, com uma estrutura de tópico (identificador). Ao agente, cabe a missão de receber a informação com seu tópico e identificar, dentre os clientes assinantes, quais solicitaram assinatura no tópico em questão, reenviando a todos eles a informação publicada. Aos clientes assinantes, uma vez identificados pelo agente como assinantes do tópico específico, receberão simultaneamente a informação enviada pelo cliente publicador.

Toda essa comunicação ocorre sobre o protocolo de conexão TCP/IP, em rede de internet, sendo definido como protocolo MQTT (*Message Queueing Telemetry Transport* - Transporte de Telemetria por Enfileiramento de Mensagem).

A Figura 86 mostra a estrutura de conexão dos componentes no conceito MQTT.

Figura 86 – Estrutura de conexão dos componentes em MQTT



Fonte: Autor (2021)

Em relação ao protocolo HTTP (*Client-Server* – cliente e servidor), que tem seu princípio na comunicação entre dois componentes, o protocolo MQTT permite que um componente possa enviar a mesma informação para um número maior de receptores. Além de apresentar outros recursos, tais

como: mensagem, tipo de mensagem, tópico, a qualidade de serviço e retenção de mensagem, formando um cabeçalho para a mensagem em si, que será identificado pelo Broker (BARROS, 2015). Como exemplo: “Ligada” <Mensagem>, tipo <Publish>, tópico <Joao123/Casa/Sala>, qualidade <QoS1>, retenção <Não>.

Quanto ao conteúdo da **mensagem** (*Payload*), trata-se de um valor numérico ou uma frase (*string*), onde sua representatividade só é de interesse dos clientes, não cabendo ao *broker* fazer quaisquer filtros.

Quanto ao **tipo de mensagem** (*Command*), são performativas[39] que indicam ao *Broker* o que se está solicitando ou informando. O Quadro 19 apresenta os principais tipos de menagens no protocolo MQTT e sua descrição.

Quadro 19 – Tipos de mensagens no protocolo MQTT

Nº	Tipo de Mensagem	Descrição
1	CONNECT	Pedido do cliente ao *Broker* para se conectar
2	CONNACK	Resposta do *Broker* ao cliente informando que a conexão foi reconhecida
3	PUBLISH	Pedido do cliente ao *Broker* ou do *Broker* ao cliente para publicar uma mensagem em um tópico indicado
4	PUBACK	Resposta do *Broker* ou cliente informando que a solicitação de publicação foi recebida
5	PUBREC	Resposta do *Broker* ao cliente informando que a solicitação de publicação foi realizada (em QoS1)
6	PUBREL	Resposta do cliente ao *Broker* informando que entendeu que sua solicitação foi atendida (em QoS2)
7	PUBCOMP	Resposta do *Broker* ao cliente informando que o processo de solicitação foi concluído (em QoS2)
8	SUBSCRIBE	Pedido do cliente ao *Broker* para assinar um tópico
9	SUBACK	Resposta do *Broker* ao cliente informando que a assinatura foi realizada
10	UNSUBSCRIBE	Pedido do cliente ao *Broker* para deixar de assinar um tópico
11	UNSUBACK	Resposta do *Broker* ao cliente informando que a assinatura foi cancelada
14	DISCONNECT	Informação do cliente ao Broker informando que irá se desconectar

Fonte: Adaptado de (NERI, RENAN; LOMBA, MATHEUS; BULHÕES, 2019).

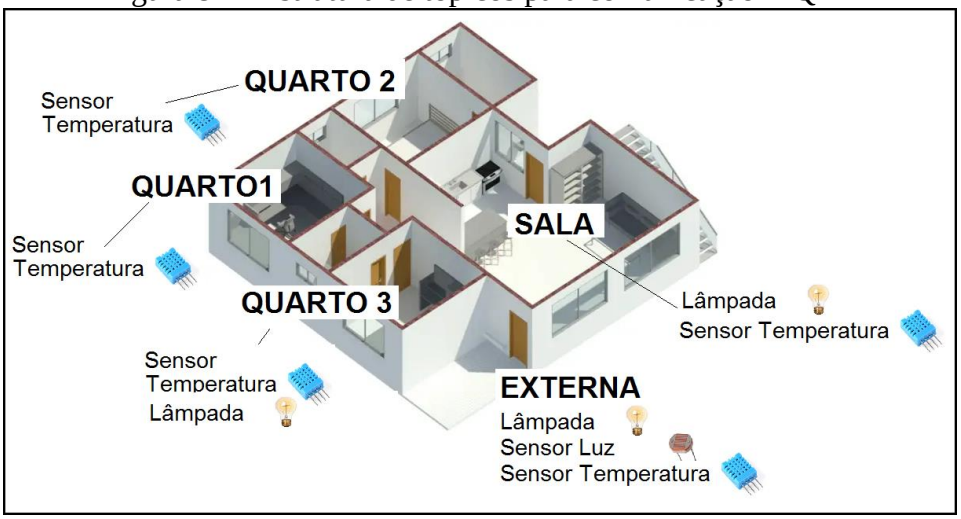
Quanto ao **tópico** (*Topic*), trata-se de um endereço para qual uma mensagem será encaminhada. A criação do endereço na forma de tópico por parte do cliente publicador se dá no momento que ocorre a publicação no *Broker* [PUBLISH]. Se houver um ou mais clientes assinantes, o Broker irá encaminhar a mensagem postada nesse endereço para os assinantes. A proposta de um tópico é que ele seja dividido em subtópicos, de forma sugestiva (como as “pastas” no *windows explorer*, por exemplo). Isto se dá pela divisão dos tópicos com o caractere “/”. Essas subdivisões

[39] Palavra que indica rapidamente do que se trata uma frase (mensagem), como exemplo: <Pergunta> “que dia é hoje”. Antes de ler a mensagem já se sabe que é uma pergunta e que requer uma resposta.

permitem acessar informações específicas ou grupo de informações, organizando-as melhor (NERI, RENAN; LOMBA, MATHEUS; BULHÕES, 2019).

A Figura 87 apresenta uma sugestão de tópicos para publicação e assinatura de informações de uma residência, pensando em uma automação residencial (domótica).

Figura 87 – Estrutura de tópicos para comunicação MQTT



Fonte: Adaptado de Archanjo (2018)

Neste exemplo há cinco ambientes que são monitorados ou acionados. Há a sala, onde é monitorada a temperatura e se pode comandar a lâmpada. Há a parte externa, onde se pode comandar a lâmpada e monitorar a luminosidade e temperatura. Há o quarto 1 e 2 onde se pode sensoriar a temperatura. E o quarto 3 onde se pode sensoriar a temperatura e comandar a lâmpada.

A sugestão de tópicos para cada elemento dessa aplicação domótica é a seguinte:

a) Casa/Sala/0/Lampada;

b) Casa/Sala/0/Temperatura;

c) Casa/Externa/0/Lampada;

d) Casa/Externa/0/Luz;

e) Casa/Externa/0/Temperatura;

f) Casa/Quarto/1/Temperatura;

g) Casa/Quarto/2/Temperatura;

h) Casa/Quarto/3/Temperatura;

i) Casa/Quarto/3/Lampada.

Com essa estrutura de tópicos, separados em subtópicos, é possível utilizar os caracteres coringas, ou seja, caracteres que permitem assinar vários tópicos de forma simultânea. São os caracteres “+” e “#”.

O caractere “+” no lugar de um subtópico significa que esse subtópico específico pode ser qualquer coisa. Como exemplo, o tópico Casa/Quarto/+/Temperatura irá assinar todos os tópicos que indicam a temperatura dos quartos, sendo eles: “Casa/Quarto/1/Temperatura”, “Casa/Quarto/2/Temperatura” e “Casa/Quarto/3/Temperatura”.

Já o caractere “#” no final de um tópico indica que serão assinados todos os tópicos indicados pelo subtópicos que antecedem o caractere coringa. Como exemplo, o tópico Casa/Quarto/# irá assinar todos os tópicos que trazem alguma informação relativa aos 3 quartos, sendo eles: “Casa/Quarto/1/Temperatura”, “Casa/Quarto/2/Temperatura”, “Casa/Quarto/3/Temperatura” e “Casa/Quarto/3/Lampada”.

Quanto à **qualidade de serviço** (*Quality of Service* – QoS), as mensagens que são trocadas podem ter configurações distintas de segurança de que o ato comunicativo ocorreu, classificados segundo sua importância e necessidade de recebimento (HIVEMQ TEAM, 2015). As mensagens podem ser divididas em 3 níveis distintos de segurança:

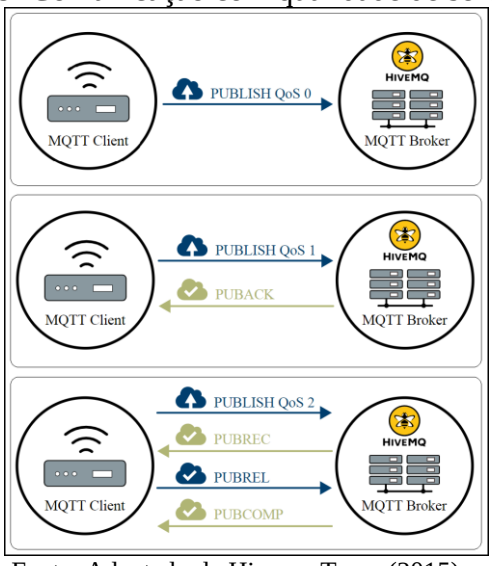
a) QoS 0 (*at most once* – no máximo 1 vez): o publicador apenas envia a mensagem [PUBLISH], sem se certificar se a mensagem chegou ao destino;

b) QoS 1 (*at least once* – ao menos 1 vez): o publicador envia a mensagem [PUBLISH] e fica reenviando até que o destinatário envie uma confirmação de que a mensagem foi recebida [PUBACK];

c) QoS 2 (*exactly once* – exatamente 1 vez): o publicador envia a mensagem [PUBLISH] e fica reenviando até que o destinatário envie uma confirmação de que a mensagem foi recebida [PUBREC], quando então o publicador envia uma mensagem reconhecendo que entendeu que a mensagem foi recebida [PUBREL] e aguarda o destinatário encerrar o atendimento ao pedido [PUBCOMP].

A Figura 88 apresenta o gráfico de comunicação nos 3 níveis de segurança.

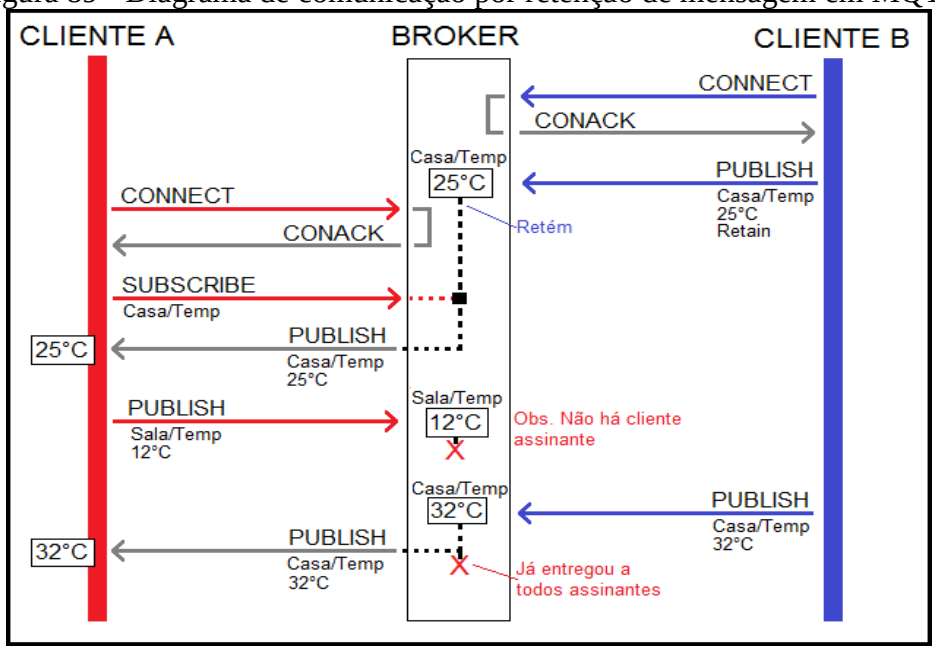
Figura 88 -Comunicação com qualidade de serviço QoS



Fonte: Adaptado de Hivemq Team (2015)

Quanto à **retenção de mensagens** (*Retain*), trata-se de um recurso do *Broker*, que permite armazenar os tópicos para futuros assinantes. Quando uma mensagem é publicada em um tópico, o *Broker* busca entre seus assinantes os que possuem assinatura nesse tópico. Caso localize, encaminha a mensagem e encerra o processo. Caso não localize, apaga a publicação, pois não há para quem entrega-la. Realiza aqui apenas um serviço de entrega do que chega, de forma instantânea. Mas se a mensagem é publicada com a solicitação de retenção do tópico (*Retain* ativo), então este fica armazenado no *Broker* para que um cliente futuro que venha assinar esse tópico possa receber essa mensagem. O *Broker* passa então a ter a função de repositório temporário. A Figura 89 apresenta o diagrama de comunicação por retenção de mensagens no protocolo MQTT.

Figura 89 - Diagrama de comunicação por retenção de mensagem em MQTT



Fonte: Autor (2021)

Em comparação ao protocolo HTTP, a comunicação MQTT requer a presença de um terceiro componente, que é o agente de recebimento e entrega de mensagens (*Broker*), recurso computacional alocado em um servidor de internet, que disponibiliza o serviço de encaminhamento de mensagens, segundo tópicos.

Os servidores de serviço de *broker* oferecem um canal de *host* (hospedeiro) para que os clientes o acessem, além de outras funcionalidades, dependendo da conta que é adquirida pelo usuário. Alguns destes servidores de *broker* disponibilizam canais de forma gratuita, como meio de teste, que podem ser utilizados por usuários. Mas, com a ressalva que trata-se de um *host* público, onde outras pessoas também podem acessar esse mesmo endereço e, por uma questão de coincidência, acessar o mesmo tópico. Por essa razão sugere-se, durante os testes, que se utilizem tópicos com ao menos um nível com nome diferenciado (exemplo: joao123), para evitar essa coincidência.

O Quadro 20 apresenta uma relação de agentes de recebimento e entrega (*Broker*) disponíveis para teste de forma gratuita, com sua indicação de *host* e porta de acesso.

Quadro 20 – Relação de agentes de recebimento e entrega disponíveis para teste de forma gratuíta

BROKER	***Site***	***Host***	**Porta**
HIVEMQ	https://www.hivemq.com/public-mqtt-broker/	broker.hivemq.com	TCP Port: 1883 Web socket: 8000
	http://www.mqtt-dashboard.com/	broker.mqttdashboard.com	TCP Port: 1883
mosquitto	https://test.mosquitto.org/	test.mosquitto.org	TCP Port: 1883 TLS Port: 8883 / 8884 Web socket: 80
EMQ	https://www.emqx.io/mqtt/public-mqtt5-broker	broker.emqx.io	TCP Port: 1883 TLS Port: 8883 Web socket: 8084

Fonte: Autor (2021)

5.4.1 Emulador MQTTBox

Para teste da comunicação no protocolo MQTT é importante o uso de ferramentas que facilitem a publicação e assinatura de tópicos em um *Broker* especificado, pois isso permite certificar-se de que cada componente da comunicação está funcionando corretamente de forma individual. Como exemplo, se você produz um código para executar um cliente publicador de tópico em um *Broker*, através de uma ferramenta de emulação MQTT se pode assinar esse tópico e verificar se o encaminhamento da mensagem ocorreu.

O MQTTBox é um aplicativo que simula a ação de um cliente publicador e assinante, de forma a que se possa publicar mensagens em um *Broker* e assinar tópicos (LOCATELLI, 2020a).

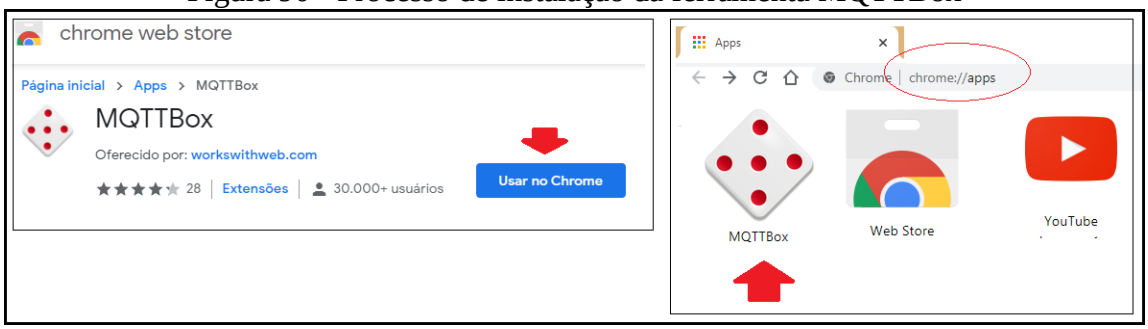
Esse aplicativo pode ser inserido no computador através de uma **extensão**[40] do Google Chrome. Para instalar essa extensão do MQTTBox deve-se acessar a página Chrome web store[41] ou buscar no navegador Google pelo nome “MQTTBox Online”. Na página de inserção da extensão clicar em “Usar no Chrome”. Uma vez instalada a extensão, ela pode ser localizada no seu navegador Google Chrome pelo endereço “chrome://apps”, que mostrará todas extensões instaladas no seu navegador.

A Figura 90 mostra esse processo de instalação da ferramenta MQTTBox.

[40] Extensões são aplicativos que se executam a partir de um navegador de internet, aumentando suas funcionalidades. Fonte: https://www.i-tecnico.pt/extensoes-de-navegadores-o-que-sao/

[41] Ferramenta MQTTBox disponível como extensão do Google Chrome em https://chrome.google.com/webstore/detail/mqttbox/kaajoficamnjijhkeomgfljpicifbkaf

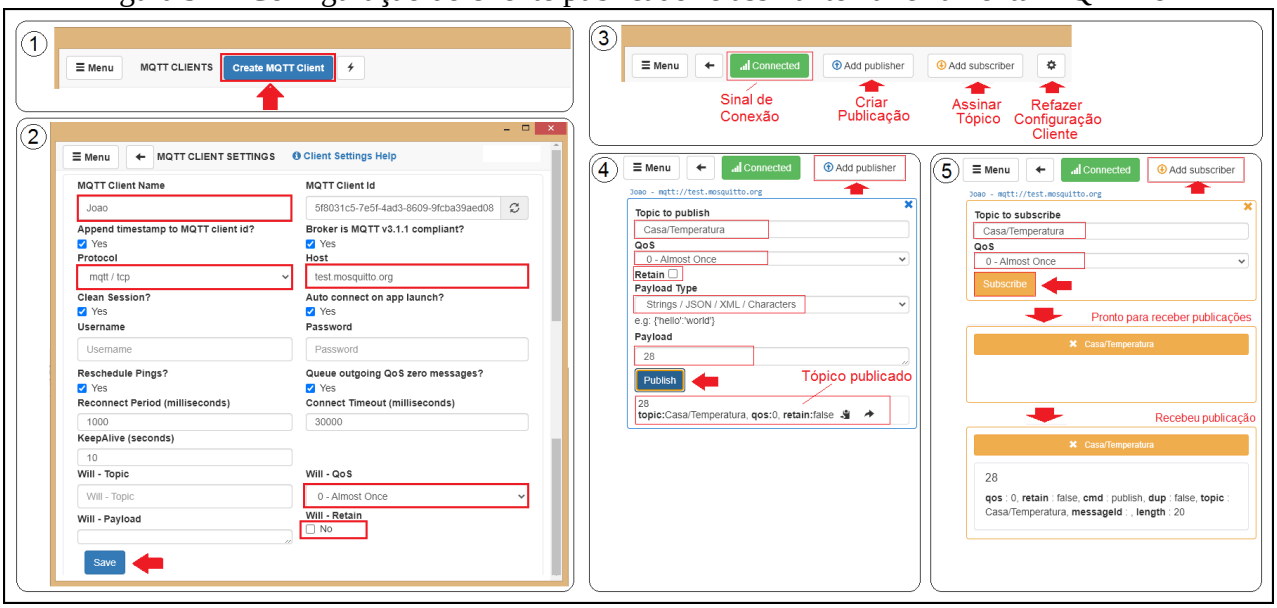
Figura 90 - Processo de instalação da ferramenta MQTTBox



Fonte: Autor (2021)

Uma vez executada a ferramenta MQTTBox, deve-se configurar o cliente para acessar um *Broker* desejado. A Figura 91 mostra o processo de configuração do cliente para publicar e assinar tópicos.

Figura 91 – Configuração do cliente publicador e assinante na ferramenta MQTTBox



Fonte: Autor (2021)

O primeiro passo, item ① da figura, é criar um cliente, clicando sobre o botão “Create MQTT Client”. No passo ② da figura, será aberta uma tela para configurar o cliente, sendo importante atribuir um nome para o cliente na caixa “MQTT Client Name”, definir o protocolo como “mqtt/tcp” em “Protocol”, indicar o endereço de “Host” do *Broker* que se quer acessar, indicar a qualidade de serviço geral do cliente em “Will - QoS” e definir se o padrão de mensagem terá retenção, em “Will - Retain”.

Será então indicada no passo ③ da figura a conexão com o *Broker*, ficando o botão em cor verde e indicando “Connected”, mostrando 3 botões: “Add Publisher” para criar uma publicação;

“Add subscriber” para criar um assinante e “Configurações” para alterar as configurações gerais do cliente.

Clicando no botão “Add publisher” no passo ④ da figura, será aberta a tela de configuração e envio da postagem. Aqui, deve-se indicar: o tópico no campo “Topic to publish”; a qualidade de serviço dessa mensagem no campo “QoS”; se essa mensagem terá retenção no campo “Retain”; o tipo de mensagem no campo “Payload Type”; em seguida editar a mensagem que se quer enviar no campo “Payload” e clicar no botão “Publish” para proceder a publicação.

Clicando no botão “Add subscriber” no passo ⑤ da figura, será aberta a tela de configuração da assinatura no *Broker*. Deve se definir o tópico que se deseja assinar no campo “Topic to subscribe”, definir a qualidade de serviço para recebimento destas mensagens no campo “QoS” e apertar no botão “Subscribe”. Será aberta a janela para receber assinaturas, tão logo uma publicação nesse tópico ocorra.

A ferramenta MQTTBox permite criar mais de uma publicação e mais de uma assinatura, podendo utilizar os caracteres coringas “+” e “#” da edição do tópico.

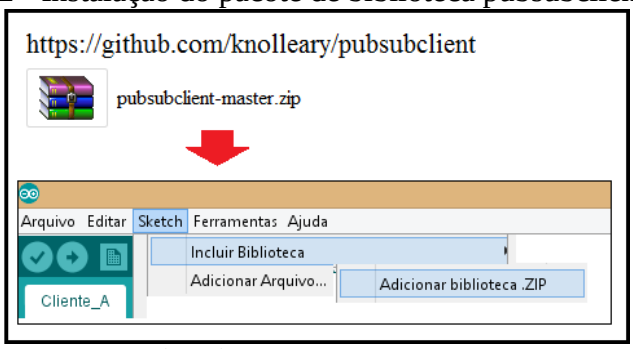
5.4.2 Aplicação com módulo ESP8266 NodeMCU com protocolo MQTT

Utilizar o módulo ESP8266 NodeMCU para publicar e assinar tópicos em um protocolo MQTT é um bom artifício para compartilhar informações e comandos, valendo-se de funcionalidades de mobilidade entre os componentes e a possibilidade de que uma publicação possa ser assinada por vários outros componentes distintos, em um processo de supervisão e controle (BERTOLETI, 2016).

Para implementação de códigos de programas a fim de que o módulo se comunique em MQTT é necessária a instalação de um pacote de biblioteca com esse propósito. O que se sugere aqui é o pacote “pubsubclient-master.zip”, disponível no endereço https://github.com/knolleary/pubsubclient.

Uma vez executado o *donwload* desse pacote compactado de biblioteca, na IDE Arduino se deve clicar na aba “Sketch”, “Incluir Biblioteca” e “Adicionar biblioteca .ZIP”, apontando o caminho onde o pacote foi salvo no momento do *download*. A Figura 92 apresenta esse processo de instalação do pacote de biblioteca “pubsubclient-master”.

Figura 92 - Instalação do pacote de biblioteca pubsubclient-master



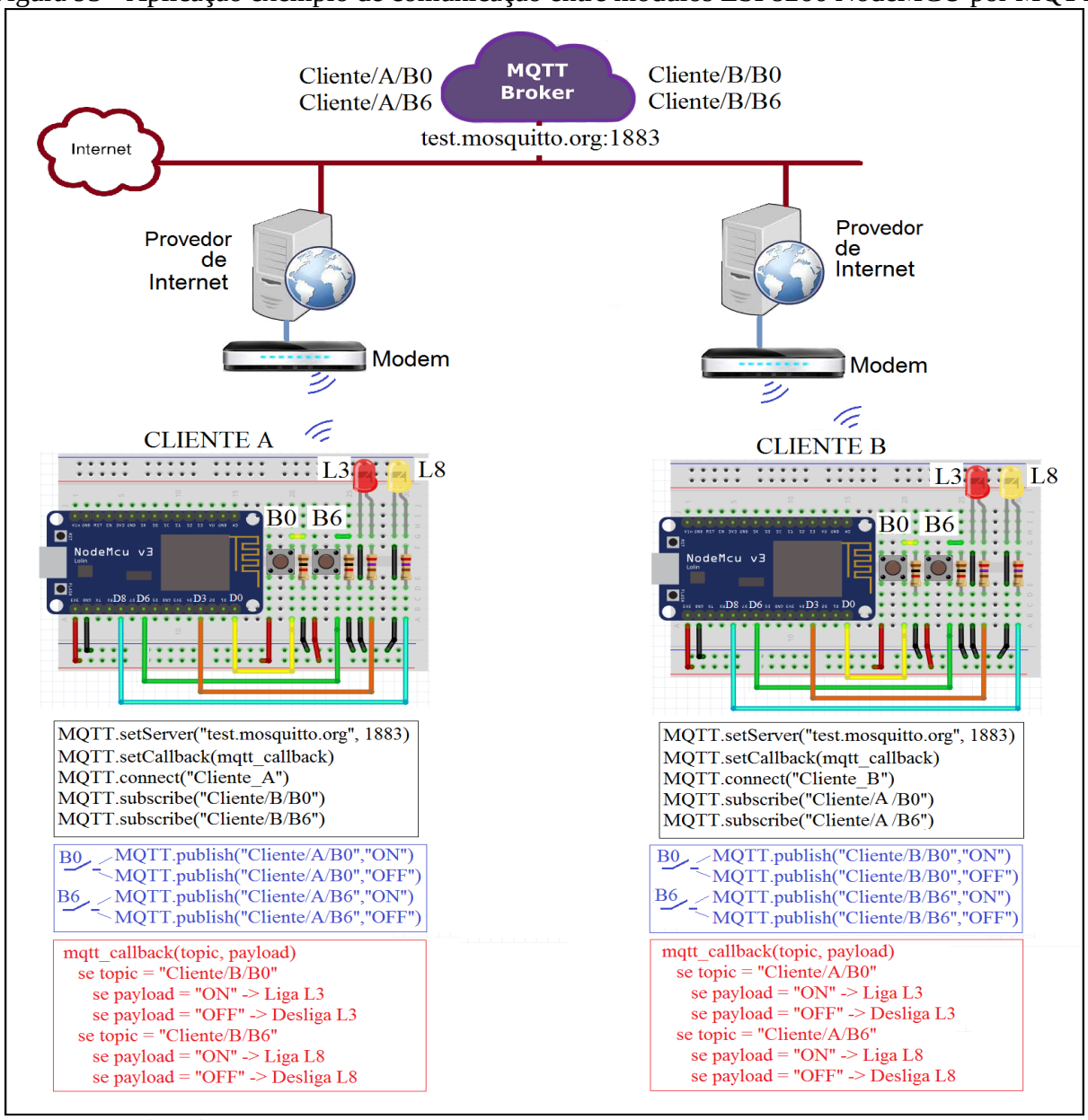
Fonte: Autor (2021)

Como aplicação de exemplo, se propõe um circuito em que há 2 módulos ESP8266 NodeMCU (Cliente A e Cliente B), onde cada módulo contém 2 botões (B0 e B6) e 2 LED (L3 e L8), sendo que, ao comutar o botão B0 do Cliente A, o LED L3 do Cliente B dever atuar, o mesmo valendo para o botão B6 do Cliente A para o LED L8 do Cliente B. E o mesmo ocorre para o botão B0 do Cliente B, que deve provocar o acionamento do LED L3 do Cliente A, o mesmo ocorrendo com botão B6 do Cliente B em relação ao LED L8 do Cliente A (BERTOLETI, 2016).

Para que isso ocorra, cada cliente irá publicar em tópicos os estados de seus botões em um *Broker*, assim como assinar o tópico relativo ao estado dos botões do outro cliente. A partir desses dados assinados, cada cliente irá atuar sobre seus respectivos LEDs.

A Figura 93 mostra a aplicação exemplo de comunicação entre os módulos em MQTT.

Figura 93 - Aplicação exemplo de comunicação entre módulos ESP8266 NodeMCU por MQTT



Fonte: Autor (2021)

Para implementar este código de programa, deve-se iniciar com a inclusão da biblioteca <ESP8266WiFi.h>, disponível no gerenciador de bibliotecas como "ESP8266WIFI by Ivan Grokhotkov". Assim como incluir a biblioteca <PubSubClient.h>, disponível para *download* como pacote de biblioteca compactado no *site* https://github.com/knolleary/pubsubclient. Essas duas bibliotecas são incluídas através da diretiva #include.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
```

Em seguida, deve-se criar o objeto “WiFiClient”, com o nome de “wifiClient”. Assim como criar o objeto “PubSubClient”, com o nome de “MQTT”, passando como parâmetro o objeto “wifiClient”.

```
WiFiClient wifiClient;
PubSubClient MQTT(wifiClient);
```

Dentro da função *setup*() deve-se inicializar a conexão com a rede *wifi*, através da função *WiFi.begin*(), passando como parâmetros o nome e senha da rede LAN em uso. E, na sequência, deve-se aguardar a conexão com a rede LAN, através do moninoramento da função *WiFi.status*(), que retorna um valor verdadeiro (TRUE) quando a conexão ocorrer.

```
void setup(){
   WiFi.begin(NOME_REDE, SENHA_REDE);
   while (WiFi.status() != WL_CONNECTED) delay(100);
```

Ainda na função *setup*() deve-se proceder a indicação do endereço do *Broker* e sua porta de acesso, através da função *MQTT.setServer*(), passando como parâmetro o endereço e a porta de acesso do *Broker*. E, como será assinado tópicos, deve-se ativar essa funcionalidade através da função *MQTT.setCallback*(), passando como parâmetro no nome da função que será executada quando um dos tópicos assinados responderem com publicação. Após, deve-se realizar a conexão, através da função *MQTT.connect*(), passando como parâmetro o nome do cliente, como um identificador dentro do *Broker*, que deve ser único. E por fim, assinar os tópicos desejados, através da função *MQTT.subscribe*(), passando por parâmetro o nome do tópico que se deseja assinar. Se houver mais tópicos, deve-se repetir essa função para cada tópico desejado.

```
void setup(){
   MQTT.setServer("ENDERECO_BROKER", PORTA_ACESSO);
   MQTT.setCallback(FUNCAO_CALLBACK);
   MQTT.connect("NOME_CLIENTE");
   MQTT.subscribe("NOME_TOPICO_ASSINAR);
```

Dentro da função *loop*() deve ocorrer a função *MQTT.loop*(), para que fique monitorando se ocorreu algum pedido de publicação por parte do *Broker* a algum tópico assinado. Quanto às publicações, estas podem ocorrer no momento em que for adequado dentro do programa, utilizando a função *MQTT.publish*(), passando por parâmetro o nome do tópico que se deseja publicar e a mensagem que se deseja transmitir.

```
void loop(){
   MQTT.loop();//keep alive do protocolo
   MQTT.publish("NOME_TOPICO","MENSAGEM");
```

Já o recebimento de tópicos assinados ocorre pela execução da função de *callback*, previamente ativada na função *setup*(). Cada vez que ocorrer um envio de mensagem de um tópico assinado por parte do *Broker*, o processamento do módulo será interrompido para que essa função seja executada. A função de *callback* irá receber por parâmetro o nome do tópico (*topic*) e sua mensagem (*payload*), cabendo à lógica implementada na função identificar o tópico que chegou e o conteúdo de sua mensagem, para atuar sobre a sua aplicação.

```
void FUNCAO_CALLBACK (char* topic, byte* payload, unsigned int length){
   String topico = topic;
   String mensagem;
```

```
for(int i = 0; i < length; i++){ char c = (char)payload[i]; mensagem += c;}
if(topico.equals("NOME_TOPICO")){
   if(mensagem.equals("MENSAGEM_DESEJADA"))
      {//toma ação dependendo da string recebida no tópico desejado}
```

Para implementação desse exemplo serão necessários 2 códigos, um para cada cliente. Nesses códigos foram utilizadas duas funções específicas para conexão à rede *wifi* e ao *Broker* (*conectaWiFi*() e *conectaMQTT*()). Isso serve para que ocorram esses pedidos de conexão durante a função *setup*() e também durante a função *loop*(), na hipótese de ter ocorrido uma desconexão da rede *wifi* ou com o *Broker*. Assim, a reconexão se torna automática.

O código para implementar o cliente A é o seguinte.

```
//************ CLIENTE A ********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> // no gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //Biblioteca.zip disponível em https://github.com/knolleary/pubsubclient
//configuração WiFi
const char* SSID = "*********"; //NOME da Rede LAN
const char* PASSWORD = "*********";    //SENHA da rede LAN
WiFiClient wifiClient;
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "Cliente_A" //Nome do Cliente, que deve ser único no Broker.
#define TOPIC_PUB_B0 "Cliente/A/B0" //Topico para publicação do estado B0 de A(ON ou OFF)
#define TOPIC_PUB_B6 "Cliente/A/B6" //Topico para publicação do estado B6 de A(ON ou OFF)
#define TOPIC_SUB_B0 "Cliente/B/B0" //Topico para Assinatura do estado B0 de B(ON ou OFF)
#define TOPIC_SUB_B6 "Cliente/B/B6" //Topico para Assinatura do estado B6 de B(ON ou OFF)
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient

#define B0 16  //porta D0 (GPIO-16)
#define B6 12  //porta D6 (GPIO-12)
#define L3 0    //Pino D3 (GPIO-0)
#define L8 15 //Pino D8 (GPIO-15)
bool estadoB0 = LOW;
bool estadoB6 = LOW;

void setup(){
  Serial.begin(115200);
  pinMode(B0,INPUT); //botão NA ligado em D0
  pinMode(B6,INPUT); //botão NA ligado em D6
  pinMode(L3,OUTPUT); //LED ligado em D3
  pinMode(L8,OUTPUT); //LED ligado em D8
  conectaWiFi();//conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT); //configura o cliente
  MQTT.setCallback(mqtt_callback); //ativa a função para receber postagens de tópicos assinados
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
```

```
    while(!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if(MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
            MQTT.subscribe(TOPIC_SUB_B0); //assina o tópico "Cliente/B/B0"
            MQTT.subscribe(TOPIC_SUB_B6); //assina o tópico "Cliente/B/B6"
        }
        else {
            Serial.println("Falha ao conectar, nova tentativa em 10s");
            delay(10000);
        }
    }
}
void publicaDados(){
  if(digitalRead(B0)==HIGH){
    delay(100);
    if(digitalRead(B0)==HIGH){
      if(estadoB0==LOW){
        estadoB0=HIGH;
        Serial.println("Botao B0 pressionado");
        MQTT.publish(TOPIC_PUB_B0,"ON"); //publica no tópico "Cliente/A/B0"
      }
    }
  }
   if(digitalRead(B0)==LOW){
    delay(100);
    if(digitalRead(B0)==LOW){
      if(estadoB0==HIGH){
        estadoB0=LOW;
        Serial.println("Botao B0 solto");
        MQTT.publish(TOPIC_PUB_B0,"OFF"); //publica no tópico "Cliente/A/B0"
      }
    }
  }
  if(digitalRead(B6)==HIGH){
    delay(100);
    if(digitalRead(B6)==HIGH){
      if(estadoB6==LOW){
        estadoB6=HIGH;
        Serial.println("Botao B6 pressionado");
        MQTT.publish(TOPIC_PUB_B6,"ON"); //publica no tópico "Cliente/A/B6"
      }
    }
  }
  if(digitalRead(B6)==LOW){
    delay(100);
    if(digitalRead(B6)==LOW){
      if(estadoB6==HIGH){
        estadoB6=LOW;
        Serial.println("Botao B6 solto");
        MQTT.publish(TOPIC_PUB_B6,"OFF"); //publica no tópico "Cliente/A/B6"
      }
    }
  }
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
     char c = (char)payload[i];
     mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_B0)){//lê o topico "Cliente/B/B0"
```

```
   //toma ação dependendo da string recebida:
   if(mensagem.equals("ON"))digitalWrite(L3,HIGH);
   if(mensagem.equals("OFF"))digitalWrite(L3,LOW);
  }
  if(topico.equals(TOPIC_SUB_B6)){//lê o topico "Cliente/B/B6"
   //toma ação dependendo da string recebida:
   if(mensagem.equals("ON"))digitalWrite(L8,HIGH);
   if(mensagem.equals("OFF"))digitalWrite(L8,LOW);
  }
}

void loop() {
  if(!MQTT.connected())conectaMQTT(); //caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  publicaDados();
  MQTT.loop(); //keep alive do protocolo
}
```

O códico para implementar o cliente B nessa aplicação é o seguinte.

```
//****************** CLIENTE B **********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //no gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //Biblioteca.zip disponível em https://github.com/knolleary/pubsubclient
//configuração WiFi
const char* SSID = "**********"; //NOME da Rede LAN
const char* PASSWORD = "**********";    //SENHA da rede LAN
WiFiClient wifiClient;
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "Cliente_B" //Nome do Cliente, que deve ser único no Broker.
#define TOPIC_PUB_B0 "Cliente/B/B0" //Topico para publicação do estado B0 de B(ON ou OFF)
#define TOPIC_PUB_B6 "Cliente/B/B6" //Topico para publicação do estado B6 de B(ON ou OFF)
#define TOPIC_SUB_B0 "Cliente/A/B0" //Topico para Assinatura do estado B0 de A(ON ou OFF)
#define TOPIC_SUB_B6 "Cliente/A/B6" //Topico para Assinatura do estado B6 de A(ON ou OFF)
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient

#define B0 16 //porta D0 (GPIO-16)
#define B6 12 //porta D6 (GPIO-12)
#define L3 0  //Pino D3 (GPIO-0)
#define L8 15 //Pino D8 (GPIO-15)
bool estadoB0 = LOW;
bool estadoB6 = LOW;

void setup(){
  Serial.begin(115200);
  pinMode(B0,INPUT); //botão NA ligado em D0
  pinMode(B6,INPUT); //botão NA ligado em D6
  pinMode(L3,OUTPUT); //LED ligado em D3
  pinMode(L8,OUTPUT); //LED ligado em D8
  conectaWiFi();//conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT);//configura o cliente
  MQTT.setCallback(mqtt_callback); //ativa a função para receber postagens de tópicos assinados
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
```

```
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
    while(!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if(MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
            MQTT.subscribe(TOPIC_SUB_B0); //assina o tópico "Cliente/A/B0"
            MQTT.subscribe(TOPIC_SUB_B6); //assina o tópico "Cliente/A/B6"
        }
        else {
            Serial.println("Falha ao conectar, nova tentativa em 10s");
            delay(10000);
        }
    }
}
void publicaDados(){
  if(digitalRead(B0)==HIGH){
    delay(100);
    if(digitalRead(B0)==HIGH){
      if(estadoB0==LOW){
        estadoB0=HIGH;
        Serial.println("Botao B0 pressionado");
        MQTT.publish(TOPIC_PUB_B0,"ON"); //publica no tópico "Cliente/B/B0"
      }
    }
  }
   if(digitalRead(B0)==LOW){
    delay(100);
    if(digitalRead(B0)==LOW){
      if(estadoB0==HIGH){
        estadoB0=LOW;
        Serial.println("Botao B0 solto");
        MQTT.publish(TOPIC_PUB_B0,"OFF"); //publica no tópico "Cliente/B/B0"
      }
    }
  }
  if(digitalRead(B6)==HIGH){
    delay(100);
    if(digitalRead(B6)==HIGH){
      if(estadoB6==LOW){
        estadoB6=HIGH;
        Serial.println("Botao B6 pressionado");
        MQTT.publish(TOPIC_PUB_B6,"ON"); //publica no tópico "Cliente/B/B6"
      }
    }
  }
  if(digitalRead(B6)==LOW){
    delay(100);
    if(digitalRead(B6)==LOW){
      if(estadoB6==HIGH){
        estadoB6=LOW;
        Serial.println("Botao B6 solto");
        MQTT.publish(TOPIC_PUB_B6,"OFF"); //publica no tópico "Cliente/B/B6"
      }
    }
  }
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
     char c = (char)payload[i];
```

```
    mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_B0)){//lê o topico "Cliente/A/B0"
    //toma ação dependendo da string recebida:
    if(mensagem.equals("ON"))digitalWrite(L3,HIGH);
    if(mensagem.equals("OFF"))digitalWrite(L3,LOW);
  }
  if(topico.equals(TOPIC_SUB_B6)){//lê o topico "Cliente/A/B6"
    //toma ação dependendo da string recebida:
    if(mensagem.equals("ON"))digitalWrite(L8,HIGH);
    if(mensagem.equals("OFF"))digitalWrite(L8,LOW);
  }
}

void loop() {
  if(!MQTT.connected())conectaMQTT(); //caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  publicaDados();
  MQTT.loop(); //keep alive do protocolo
}
```

O Projeto 21 apresenta uma aplicação de monitoramento de variáveis de uma residência, que disponibiliza seus dados de umidade e temperatura em tópicos MQTT, recebendo um comando para o acionamento de uma luminária. Podendo esses dados serem acessados por outros clientes que assinarem os tópicos no *Broker* “test.mosquitto.org” utilizado.

PROJETO 21 – COMANDO DE LUMINÁRIA E MONITORAMENTO DE TEMPERATURA E UMIDADE EM MQTT COM MÓDULO ESP8266 NODEMCU

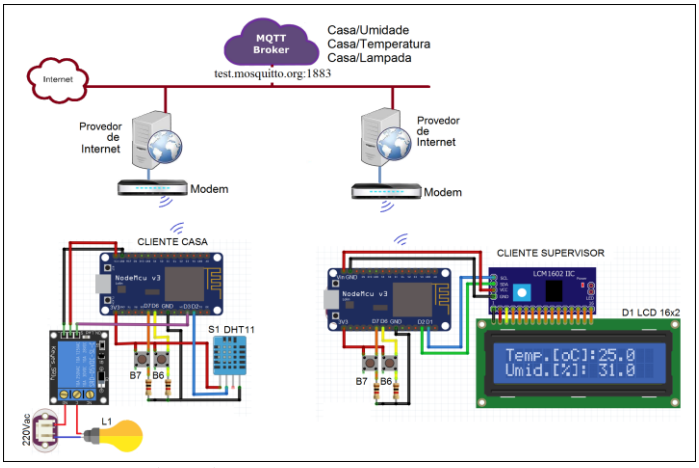
"O projeto consiste em um módulo ESP8266 NodeMCU, conectados a um modem wifi, que gerencia o monitoramento de umidade e temperatura de uma residência, assim como o comando de uma luminária, disponibilizando essas informações em tópicos MQTT em um Broker. Um outro módulo ESP8266 NodeMCU, conectados a um modem wifi, assina esses tópicos e exibe em um display matricial LCD as informações de umidade e temperatura da casa monitorada e, através de um botão, comanda o acionamento de uma luminária na casa, de forma remota, através do tópico MQTT.

Nesse projeto há o módulo de cliente "Casa", que monitora a temperatura e umidade pelo sensor S1 (DHT11) e aciona uma lâmpada L1 através de um módulo relé, sendo que ao acionar o botão B6, o relé liga a lâmpada L1, ao acionar o botão B7, o relé desliga a lâmpada, de forma local. Esse cliente publica o valor da temperatura no tópico "Casa/Temperatura" e o valor da umidade no tópico "Casa/Umidade". Ele também assina o tópico "Casa/Lampada", onde recebe publicação do cliente "Supervisor" que pode ligar e desligar a lâmpada de forma remota, emitindo as mensagens "Ligar" e Desligar", respectivamente.

O projeto prevê ainda um cliente "Supervisor" que assina os tópicos "Casa/Temperatura" e "Casa/Umidade", exibindo esses valores em um *display* matricial LCD, em protocolo I2C. Esse cliente também publica no tópico "Casa/Lampada", para emitir comando remoto para ligar e desligar a lâmpada.

A Figura 94 apresenta o diagrama de aplicação do projeto.

Figura 94 – Diagrama da aplicação do projeto de comando e monitoramento de variáveis de uma residência



Fonte: Autor (2021)

O código para implementação do cliente “Casa” é o seguinte.

```
//************ CLIENTE CASA ********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <DHT.h>
#define DHTPIN 4 // pino do sensor D2 - GPIO4 - SDA (não pode ser outro)
#define DHTTYPE DHT11 //tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
#include <ESP8266WiFi.h> // no gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //Biblioteca.zip disponível em https://github.com/knolleary/pubsubclient
WiFiClient wifiClient;
//configuração WiFi
const char* SSID = "*********"; //NOME da Rede LAN
const char* PASSWORD = "*********";    //SENHA da rede LAN
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "CASA" //Nome do Cliente, que deve ser único no Broker.
#define TOPIC_PUB_TEMP "Casa/Temperatura" //Topico para publicar Temperatura
#define TOPIC_PUB_UMID "Casa/Umidade" //Topico para publicar Umidade
#define TOPIC_SUB_LAMP "Casa/Lampada" //Topico para Assinatura o comando da Lâmpada
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient
#define B6 12 //porta D6 (GPIO-12)
#define B7 13 //porta D7 (GPIO-13)
#define L1 0  //Pino D3 (GPIO-0)
unsigned long tempoAnterior = 0;

void setup(){
  Serial.begin(115200);
  dht.begin();
  pinMode(B6,INPUT); //botão NA ligado em D6
  pinMode(B7,INPUT); //botão NA ligado em D7
  pinMode(L1,OUTPUT); //Relé ligado em D3
  conectaWiFi(); //conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT); //configura o cliente
  MQTT.setCallback(mqtt_callback); //ativa a função para receber postagens de tópicos assinados
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
    while(!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if(MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
            MQTT.subscribe(TOPIC_SUB_LAMP);//assina o tópico "Casa/Lampada"
        }
        else {
            Serial.println("Falha ao conectar, nova tentativa em 10s");
            delay(10000);
        }
    }
}
void comandaLampada(){
  if(digitalRead(B6)==HIGH){
     delay(100);
```

```
    if(digitalRead(B6)==HIGH){
       digitalWrite(L1, HIGH);
    }
  }
  if(digitalRead(B7)==HIGH){
    delay(100);
    if(digitalRead(B7)==HIGH){
       digitalWrite(L1, LOW);
    }
  }
}
void publicaDados(){
  if(millis() - tempoAnterior > 5000){//leitura a cada 5s
    tempoAnterior = millis();
    char MsgUmidadeMQTT[10];
    char MsgTemperaturaMQTT[10];
    float temperatura = dht.readTemperature(); //Le a temperatura
    float umidade = dht.readHumidity(); //Le a Humidade
    sprintf(MsgTemperaturaMQTT,"%.1f",temperatura);
    MQTT.publish(TOPIC_PUB_TEMP,MsgTemperaturaMQTT); //publica no tópico "Casa/Temperatura"
    sprintf(MsgUmidadeMQTT,"%.1f",umidade);
    MQTT.publish(TOPIC_PUB_UMID,MsgUmidadeMQTT); //publica no tópico "Casa/Umidade"
  }
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
    char c = (char)payload[i];
    mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_LAMP)){//lê o topico "Casa/Lampada"
    //toma ação dependendo da string recebida:
    if(mensagem.equals("Ligar"))digitalWrite(L1,HIGH);
    if(mensagem.equals("Desligar"))digitalWrite(L1,LOW);
  }
}

void loop() {
  if(!MQTT.connected())conectaMQTT();//caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  comandaLampada();
  publicaDados();
  MQTT.loop();//keep alive do protocolo
}
```

O código para implementação do cliente “Supervisor” é o seguinte.

```
//****************** CLIENTE SUPERVISOR ***********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
// Ligação da I2C no ESP8266  SCL -> 5(D1) - SDA -> 4(D2)
LiquidCrystal_I2C lcd(0x3F,16,2);//(usar codico scan para descobrir 0x3F)
#include <ESP8266WiFi.h> // no gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //Biblioteca.zip disponível em https://github.com/knolleary/pubsubclient
//configuração WiFi
const char* SSID = "**********"; //NOME da Rede LAN
const char* PASSWORD = "**********";    //SENHA da rede LAN
WiFiClient wifiClient;
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "SUPERVISOR" //Nome do Cliente, que deve ser único no Broker.
```

```
#define TOPIC_PUB_LAMP "Casa/Lampada" //Topico para publicação do comando para lampada
#define TOPIC_SUB_TEMP "Casa/Temperatura" //Topico para assinatura da temperatura
#define TOPIC_SUB_UMID "Casa/Umidade" //Topico para Assinatura da umidade
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient
#define B6 12 //porta D6 (GPIO-12)
#define B7 13 //porta D7 (GPIO-13)

void setup(){
  Serial.begin(115200);
  pinMode(B6,INPUT); //botão NA ligado em D6
  pinMode(B7,INPUT); //botão NA ligado em D7
  conectaWiFi();//conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT);//configura o cliente
  MQTT.setCallback(mqtt_callback);//ativa a função para receber postagens de tópicos assinados
  lcd.init();
  lcd.setBacklight(HIGH);
  lcd.print("Temp.[oC]:      ");
  lcd.setCursor(0,1);
  lcd.print("Umid.[%]:       ");
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
    while(!MQTT.connected()) {
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if(MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
            MQTT.subscribe(TOPIC_SUB_TEMP);  //assina o tópico "Cliente/A/B0"
            MQTT.subscribe(TOPIC_SUB_UMID);  //assina o tópico "Cliente/A/B6"
        }
        else {
            Serial.println("Falha ao conectar, nova tentativa em 10s");
            delay(10000);
        }
    }
}
void publicaDados(){
  if(digitalRead(B6)==HIGH){
    delay(100);
    if(digitalRead(B6)==HIGH){
      MQTT.publish(TOPIC_PUB_LAMP,"Ligar");  //publica no tópico "Casa/Lampada"
    }
  }
  if(digitalRead(B7)==HIGH){
    delay(100);
    if(digitalRead(B7)==HIGH){
      MQTT.publish(TOPIC_PUB_LAMP,"Desligar");  //publica no tópico "Casa/Lampada"
    }
  }
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
```

```
    char c = (char)payload[i];
    mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_TEMP)){//lê o topico "Casa/Temperatura"
    Serial.print("Temperatura: ");
    Serial.println(mensagem);
    lcd.setCursor(10,0);
    lcd.print(mensagem);
  }
  if(topico.equals(TOPIC_SUB_UMID)){//lê o topico "Casa/Umidade"
    Serial.print("Umidade: ");
    Serial.println(mensagem);
    lcd.setCursor(10,1);
    lcd.print(mensagem);
  }
}

void loop() {
  if(!MQTT.connected())conectaMQTT();//caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  publicaDados();
  MQTT.loop();//keep alive do protocolo
}
```

6 INTERAÇÃO COM APLICATIVOS

A interação do módulo ESP8266 NodeMCU com aplicativos, sejam eles *software* de computadore ou aplicativos *mobile*, é a grande proposta desse módulo se prestar como uma coisa, no universo da internet das coisas. Não se trata somente de acessar a rede internet para trocar mensagem com outro módulo, mas de realmente interagir com outros aplicativos, enviando informações e recebendo dados.

Neste capítulo será apresentada a interação do módulo com algumas aplicações.

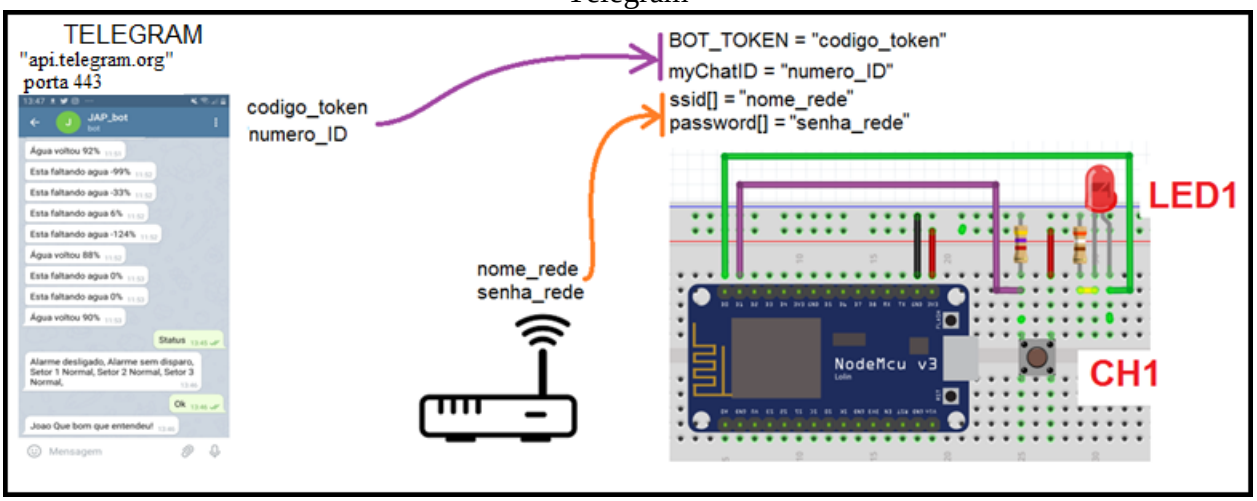
6.1 INTERAÇÃO COM APLICATIVO TELEGRAM

A interação do módulo ESP8266 NodeMCU com o aplicativo Telegram parte de uma assinatura do usuário ao aplicativo de mensagens Telegram, que dispõe de acesso a API através de um código identificador do usuário, na forma de um *token* e um número de identificação de um subusuário criado no aplicativo para se comunicar com o módulo ESP8266 NodeMCU (TELEGRAM, 2021).

Uma vez tendo esse acesso à API ("api.telegram.org", porta 443) com código de *token* e número ID, o módulo ESP8266 NodeMCU pode enviar mensagens de texto e recebe-las, cabendo a ele interpretar essas mensagens para gerar comandos sobre atuadores do projeto ou gerar respostas para o usuário dedicado no Telegram (CODE LEARN, 2021).

A Figura 95 apresenta o princípio de funcionamento desse acesso ao *site* especializado através de API, sendo um aplicativo de troca de mensagens.

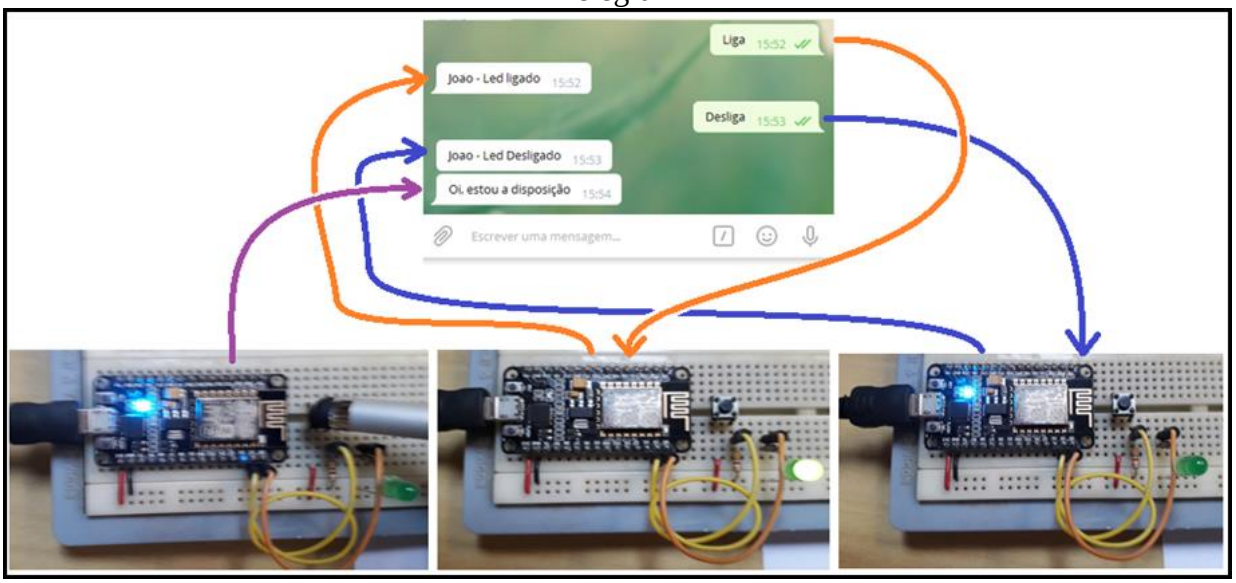
Figura 95 – Princípio de funcionamento da troca de mensagens entre ESP8266 NodeMCU e Telegram



Fonte: Autor (2021)

O resultado do projeto é um circuito com um LED (LED1) que será ligado se a mensagem de chegada ao módulo ESP8266 NodeMCU for a *String* “Liga”. Caso a mensagem de chegada seja “Desliga”, então o LED será desligado. E como resposta, o módulo irá enviar a mensagem “Oi, estou a disposição” à Telegram, a cada vez que o botão CH1 for pulsionado. A Figura 96 apresenta o resultado esperado desse projeto.

Figura 96 – Resultado esperado do projeto troca de mensagens entre ESP8266 NodeMCU e Telegram



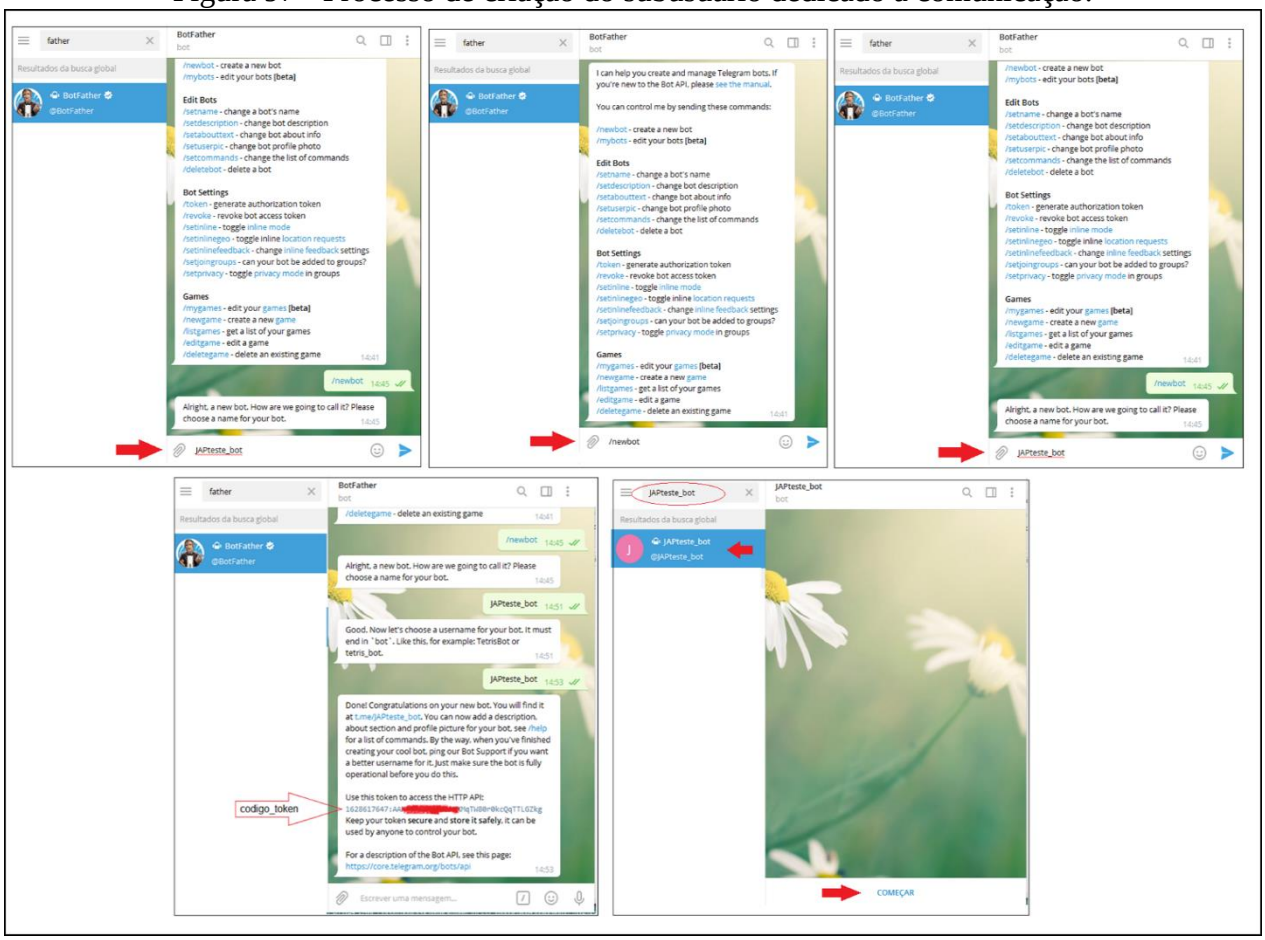
Fonte: Autor (2021)

Para criação do subusuário no Telegram, é necessário que se procure nos contatos do aplicativo o recurso “BotFather”, selecionando e clicando em “COMEÇAR”. Surgirão então várias opções de comandos. Nesse instante, se deve indicar o comando “/newbot”, a fim de criar um novo

usuário do tipo Bot. É solicitado então que se dê um nome para o usuário_bot. Pode ser qualquer nome, desde que termine com os caracteres "_bot". Neste exemplo, foi colocado o nome como JAPteste_bot. Será solicitado então que se entre novamente com o nome do usuário criado "JAPteste_bot". Após isso, será apresentado o código do *token*, que deverá ser copiado e inserido no programa do módulo. Com o usuário criado, é só procurar pelo nome do usuário "JAPteste_bot" nos contatos, selecionar o usuário localizado e clicar em "COMEÇAR" ou digitar o comando "/start".

A Figura 97 apresenta as telas com o processo de criação desse usuário.

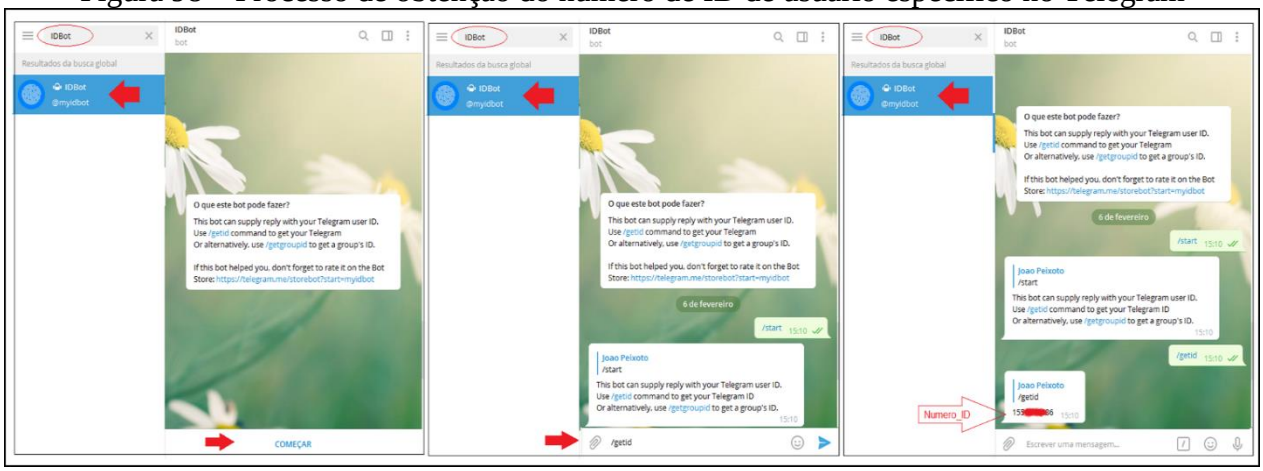
Figura 97 - Processo de criação do subusuário dedicado à comunicação.



Fonte: Autor (2021)

Para identificar o parâmetro número de ID, localize nos contatos do Telegram o usuário IDBot, selecione-o e pressione em COMEÇAR. Digite então o comando "/getid" para buscar o identificador da conta Telegram, na qual está sendo gerado o usuário específico para comunicação (JAPteste_bot). O código de número de ID é apresentado e deverá ser inserido no código da programação no módulo ESP8266 NodeMCU, segundo Figura 98.

Figura 98 – Processo de obtenção do numero de ID de usuário específico no Telegram



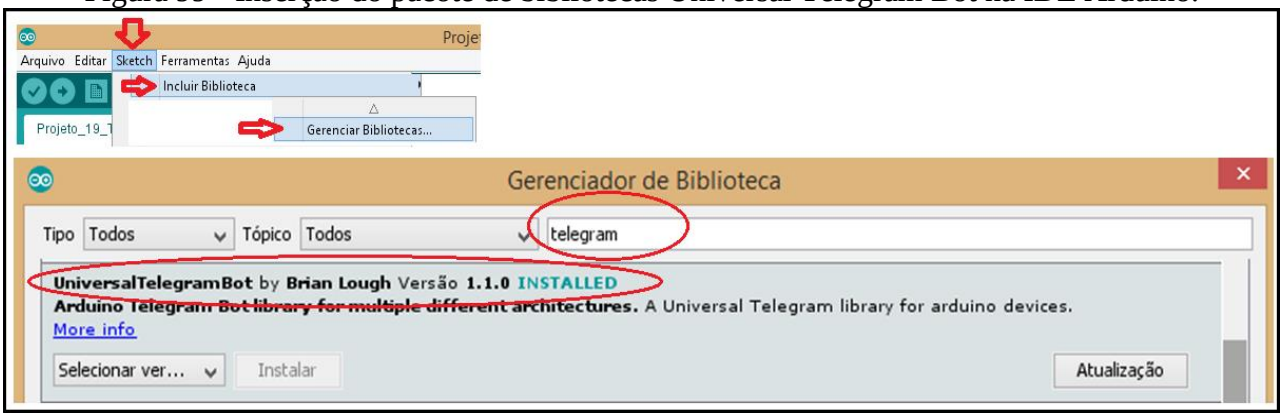
Fonte: Autor (2021)

PROJETO 22 – TROCA DE MENSAGENS ENTRE ESP8266 NODEMCU E TELEGRAM

"O projeto consiste em um módulo ESP8266 NodeMCU, conectados a um modem wifi, que acessa um aplicativo de mensagens Telegram, por uma API em código Token, procedendo a troca de mensagens de texto entre si, podendo essas mensagens serem interpretadas como informações ou comando."

Para edição do programa no módulo ESP8266 NodeMCU na IDE Arduino, é necessário instalar o pacote de bibliotecas "UniversalTelegramBot by Brian Lough", que pode ser encontrado no gerenciador de bibliotecas com a palavra-chave "telegram". Para chegar ao gerenciador de bibliotecas deve-se clicar na aba "Sketch", "Incluir Biblioteca" e "Gerenciar Biblioteca", seguindo os passos da Figura 99.

Figura 99 - Inserção do pacote de bibliotecas Universal Telegram Bot na IDE Arduino.



Fonte: Autor (2021)

Uma vez instalado esse pacote de biblioteca, se edita o código na IDE do Arduino. Algumas linhas do código devem ser destacadas para uma explicação mais detalhada.

A inserção das bibliotecas "ESP8266WiFi.h" se dá após instalar o pacote de bibliotecas "ESP8266WiFi by Ivan Grokhotkov", através das abas "Sketch", "Incluir Biblioteca" e "Gerenciar Bibliotecas...", solicitando sua instalação, o que remete à biblioteca identificada pelo arquivo de cabeçalho "WiFiClientSecure.h", que estará no pacote instalado. A biblioteca "UniversalTelegramBot.h" pode ser localizada e instalada no Gerenciador de bibliotecas, como "UniversalTelegramBot by Brian Lough". Todas essas bibliotecas são inseridas pela diretiva #include.

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
```

É necessária a criação de um objeto do tipo *WiFiClientSecure*, com nome de "client", que será utilizado como parâmetro na criação do objeto UniversalTelegramBot, com nome de bot, que recebe também o número do *token*, extraído do usuário dedicado no Telegram.

```
#define BOT_TOKEN "1628617647:AAHX6f**********TW80r0kcQqTTLGZkg"
 WiFiClientSecure client;
UniversalTelegramBot bot(BOT_TOKEN, client);
```

Dentro a função *setup(),* deve-se configurar o módulo como uma estação *wifi*, através da função e parâmetro *WiFi.mode(WIDI_AP_STA).* Na sequência, desconecta-se o *wifi* para poder iniciar uma conexão própria, através da função *WiFi.disconnect().* Então, executa-se a função *WiFi.begin(),* passando por parâmetro no nome da rede *wifi* e sua senha. Por fim, se faz a verificação de conexão, monitorando a função *WiFi.status*(), que retorna a constante WL_CONNECTED quando a conexão confirmar.

```
void setup(){
   WiFi.mode(WIFI_AP_STA);
   WiFi.disconnect();
   WiFi.begin(NOME_REDE, SENHA_REDE);
   while (WiFi.status() != WL_CONNECTED) delay(200);
```

Para segurança na troca de mensagens com o Telegram, é necessária a inserção de um código para teste de segurança, obtido no *site* Telegram, onde um dos códigos é "F2:AD:29:9C:34:48:DD:8D:F4:CF:52:32:F6:57:33:68:2E:81:C1:90". A inserção desse código no módulo se dá pela função *client.setFingerprint().* Em seguida, se faz a conexão com a API Telegram, através da função *client.connect(),* passando por parâmetro o endereço da API "api.telegram.org" e a indicação da porta 443. Essa função retorna um valor boleano verdadeiro (TRUE) se a conexão ocorrer, logo, deve-se ficar monitorando até que a conexão ocorra.

```
void setup(){
client.setFingerprint("F2:AD:29:9C:34:48:DD:8D:F4:CF:52:32:F6:57:33:68:2E:81:C1:
90");
while (!client.connect("api.telegram.org", 443)) delay(500);
```

Para enviar mensagens do módulo ESP8266 NodeMCU para o usuário dedicado Telegram, dentro da função *loop()* se utiliza a função bot.sendMessage(), passando por parâmetro o número do ID do usuário ao qual se deseja enviar a mensagem.

```
String myChatID = "15******86"; // ver Figura 98 para obtenção do chatID
void loop(){
   bot.sendMessage(myChatID,"Oi, estou a disposição", "");
```

Já para receber mensagens de texto é um pouco mais complexo pois, ao solicitar o recebimento de mensagens pela função bot.getUpdates(), poderá vir mais que uma mensagem. Assim, é necessário identificar o número de mensagens que chegaram, armazenadas em um vetor de String.

```
void loop(){
int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    while(numNewMessages){
     for (int i=0; i<numNewMessages; i++){
        String chat_id = String(bot.messages[i].chat_id);//ID de quem enviou
        String text = bot.messages[i].text;//texto enviado
        String from_name = bot.messages[i].from_name; //Nome do usuário que enviou
        if (text == "Liga" ){  //comandos a serem executados no projeto
          bot.sendMessage(chat_id, from_name + " - Led ligado", "");
        }
        if (text == "Desliga"){ //comandos a serem executados no projeto
          bot.sendMessage(chat_id, from_name + " - Led Desligado", "");
        }
     }
     numNewMessages = bot.getUpdates(bot.last_message_received + 1);
```

O código completo do programa para troca de mensagens entre o módulo ESP8266 NodeMCU e o aplicativo de mensagens Telegram é o seguinte:

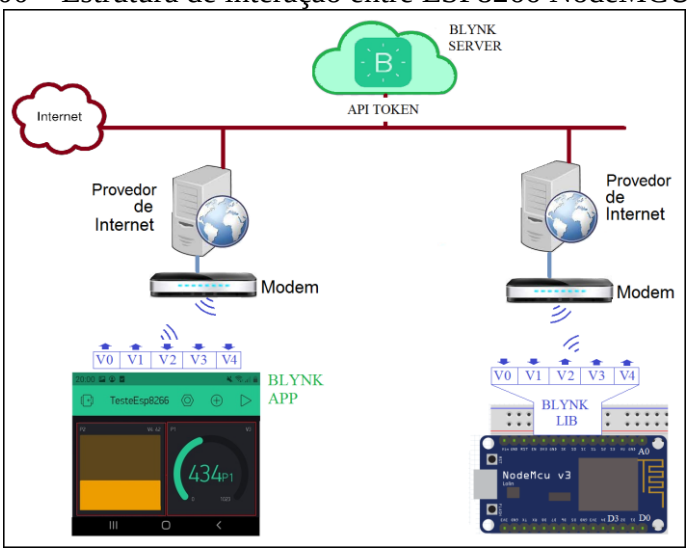
```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <UniversalTelegramBot.h> //Gerenciador de bibliotecas "UniversalTelegramBot by Brian Lough"
#include <ESP8266WiFi.h> //Gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <WiFiClientSecure.h> //dentro pacote ESP8266WiFi
char ssid[] = "**********";// nome da sua rede WIFI (SSID)
char password[] = "**********";// senha da sua rede WIFI
#define BOT_TOKEN "1628617647:AAHX6**********qTTLGZkg"  // sua chave Token Bot Telegram
WiFiClientSecure client;
UniversalTelegramBot bot(BOT_TOKEN, client);

#define LED1 16    // GPIO_16 ->D0
#define CH1 5  //GPIO_5  ->D1
int Bot_mtbs = 1000; //tempo entre as mensagens (ms)
long Bot_lasttime;   //tempo do último scanner das mensagens
String myChatID = "15********86";// Para obter o Chat ID, acesse Telegram => usuario IDBot => comando /getid

void setup()
{
  pinMode(LED1, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(CH1, INPUT);
 // Inicializa conexão Wifi
  WiFi.mode(WIFI_AP_STA);   // Configura o WIFI para modo estação e Acess point
  WiFi.disconnect();        // desconecta o WIFI
  delay(100);               // atraso de 100 milisegundos
  WiFi.begin(ssid, password);//insere o nome da rede e senha
  while (WiFi.status() != WL_CONNECTED)    // aguardando a conexão WEB
  {digitalWrite(LED_BUILTIN, LOW); delay(50); digitalWrite(LED_BUILTIN,HIGH); delay(50);}
 // Conecta cliente com SSL
  client.setFingerprint("F2:AD:29:9C:34:48:DD:8D:F4:CF:52:32:F6:57:33:68:2E:81:C1:90");
  while (!client.connect("api.telegram.org", 443))
  {digitalWrite(LED_BUILTIN, LOW); delay(200); digitalWrite(LED_BUILTIN,HIGH); delay(200);}
}

void loop(){
  if (millis() > Bot_lasttime + Bot_mtbs){
      {digitalWrite(LED_BUILTIN, LOW); delay(50); digitalWrite(LED_BUILTIN,HIGH); delay(50);}
      if(digitalRead(CH1)){
        bot.sendMessage(myChatID,"Oi, estou a disposição", "");
      }
      int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
      while(numNewMessages){
        for (int i=0; i<numNewMessages; i++){
```

```
        String chat_id = String(bot.messages[i].chat_id);
        String text = bot.messages[i].text;
        String from_name = bot.messages[i].from_name;
        if (text == "Liga" ){
          digitalWrite(LED1,HIGH);
          bot.sendMessage(chat_id, from_name + " - Led ligado", "");
        }
        if (text == "Desliga"){
          digitalWrite(LED1,LOW);
          bot.sendMessage(chat_id, from_name + " - Led Desligado", "");
        }
      }
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
  Bot_lasttime = millis();
  }
}
```

6.2 INTERAÇÃO COM APLICATIVO BLYNK

O portal Blynk é uma solução de comunicação entre coisas pela internet. Trata-se de um servidor (*Blynk Server*) na internet que promove a conexão e troca de informações entre o aplicativo *mobile* (*Blynk App*) e dispositivos microcontrolados que suportem a sua biblioteca (*Blynk Library*) (BLYNK, 2021). A Figura 100 apresenta a estrutura de comunicação do Blynk.

Figura 100 – Estrutura de interação entre ESP8266 NodeMCU e Blynk



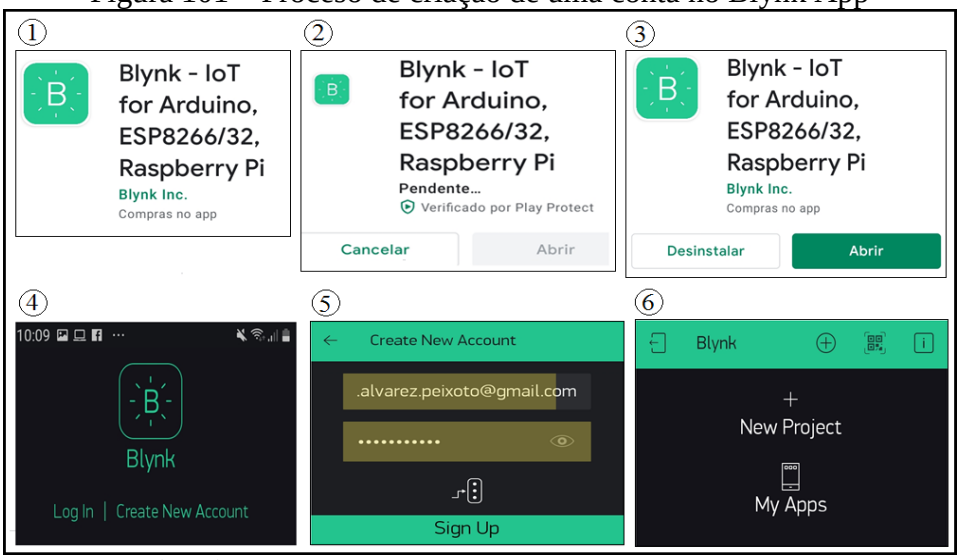
Fonte: Autor (2021)

A conexão do microcontrolador com o servidor Blynk se dá por API *Token*, onde cada aplicação que se desenvolve no aplicativo Blynk recebe um código de *token* esperífico, para ser acessado pelo dispositivo ligado à internet. Uma vez estabelecida a conexão, o aplicativo Blynk irá compartilhar variáveis com o dispositivo microcontrolador, através de uma biblioteca Blynk Lib, lendo ou escrevendo dados nessas variáveis.

Da parte do aplicativo Blynk (*Blynk App*), são disponibilizados objetos (*widgets*), em que cada um possui um preço, definido por uma moeda intitulada “Energia”. Inicialmente, todo usuário recebe de bônus 2.000 energias para adquirir objetos *widgets*. Caso necessite de mais, deverá comprar moedas de energia, no próprio aplicativo *mobile*. Portanto, para pequenas implementações não há custo, mas a medida que a complexidade da aplicação aumenta, faz-se necessário a aquisição de mais componentes (SERRANO, 2018). Cada objeto *widget* é vinculado a uma variável, que será compartilhada pelo servidor Blink com o dispositivo conectado a ele pelo Blynk Token. Essas variáveis serão inseridas no dispositivo através da biblioteca Blynk, podendo ser acessadas pelo programa nele implementado.

O processo se inicia com a instalação do aplicativo Blynk no dispositivo *mobile* e criação de uma conta no servidor Blink. Neste trabalho será mostrada a instalação do aplicativo em um *mobile* com sistema operacional Android, por ser o mais usual. A Figura 101 apresenta esse processo.

Figura 101 – Proceso de criação de uma conta no Blynk App

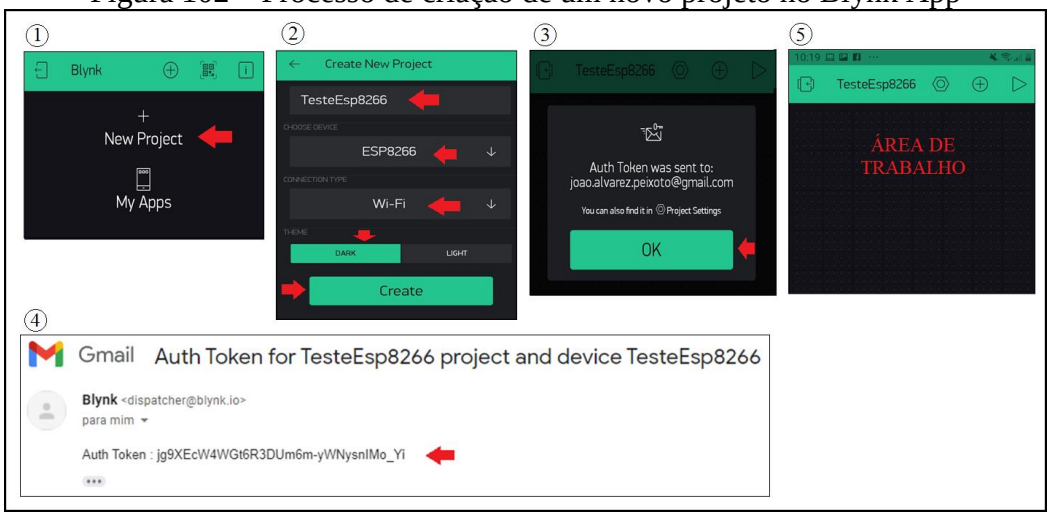


Fonte: Autor (2021)

O primeiro passo é procurar na Play Store do aplicativo Android do *mobile* pelo aplicativo "Blynk – IoT for Arduino, ESP8266/32, Raspberry Pi", item ① da figura, executando sua instalação ②. Procede-se a abertura do aplicativo Blynk App, item ③, e na sequência se solicita a criação de uma conta, clicando no botão "Create New Account", item ④. Deve-se indicar um endereço de e-mail válido e uma senha de acesso, importante, pois é para esse e-mail que o código de Blynk Token será enviado, item ⑤. Na tela do item ⑥ o aplicativo Blynk já está pronto para gerar um projeto.

O processo para criação de um novo projeto é apresentado na Figura 102.

Figura 102 – Processo de criação de um novo projeto no Blynk App

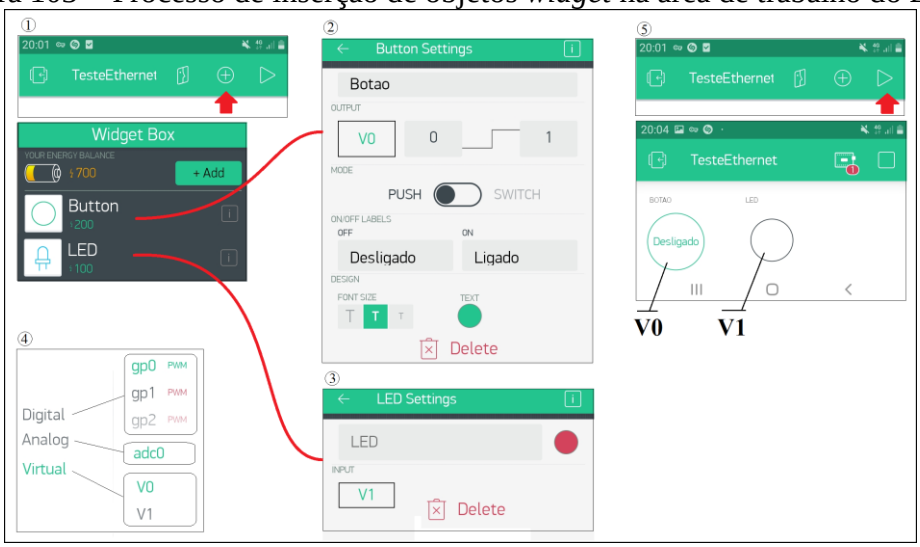


Fonte: Autor (2021)

Com o aplicativo Blynk aberto, na sua tela inicial, se deve clicar no botão "*New Project*", item ①. Abrirá uma tela para entrada dos dados relativos à aplicação, item ②. Deve-se inserir um nome para o projeto, como exemplo "TesteEsp8266", selecionar o tipo de componente em "*Choose Device*", o tipo de conexão em "*Connection Type*" e selecionar o tema (cor de fundo) em "*Theme*", por fim,basta clicar no botão "*Create*". Uma vez realizado esse passo, uma tela de aviso indicará que o código de *Token* foi enviado para o e-mail cadastrado pelo usuário, item ③, o que pode ser percebido ao abrir o e-mail recebido, item ④. Esse código deverá ser copiado para posterior inserção no código do módulo ESP8266 NodeMCU. Por fim, item ⑤, é então apresentada a área de trabalho para edição do projeto que se deseja interagir com o módulo.

Para inserção de objetos, deve-se seguir os passos da Figura 103, que mostra a inserção de um botão e um LED.

Figura 103 - Processo de inserção de objetos *widget* na área de trabalho do *Blynk*



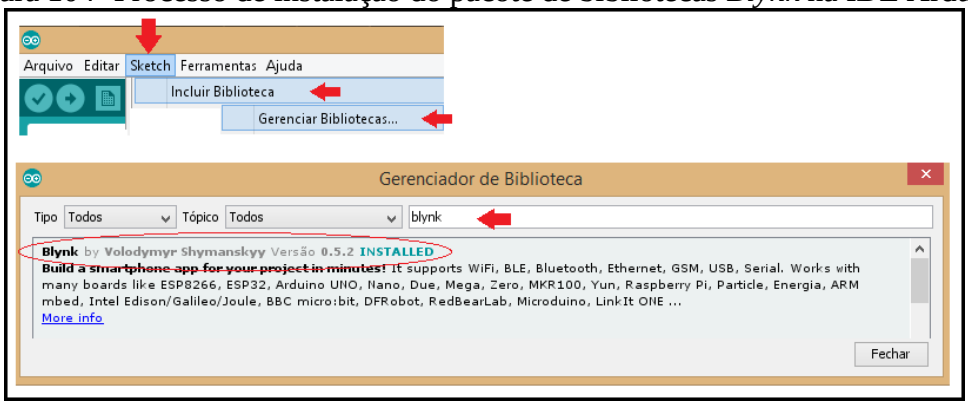
Fonte: Autor (2021)

No item ①, deve-se clicar no botão "+" para abrir a caixa "Widget Box", onde são apresentados o saldo de energia e os componentes que podem ser inseridos. Neste exemplo, será inserido um objeto botão (*Button*) e um objeto LED. Ao clicar sobre o objeto botão inserido na área de trabalho, é aberta uma caixa "*Button Settings*", item ②, que permite parametrizar esse botão, onde é definido o pino de saída de dados e valores, quando este for acionado, no campo "*output*". No campo "*mode*", é definido se ele terá o comportamento de um botão ou chave. No campo "*On/off labels*", é definido o texto que deverá ser exibido no botão segundo sua situação de acionamento. No campo "*design*", é definido o tamanho da fonte dos textos e a cor do botão quando acionado. Ao clicar sobre o objeto LED inserido na área de trabalho é aberta a caixa "*LED Settings*" para parametrização do LED, item ③, onde se define o pino de recebimento de sinal, no campo "*input*", além da cor do LED. O comportamento do LED é analógico, ou seja, ele muda de intensidade em função do valor numérico que lhe é inserido, sendo valor 0 para LED apagado e 255 para LED com

máxima intensidade. Importante destacar os tipos de pinos que um objeto pode assumir, item ④, podendo ser um pino "*Digital*", que se conecta diretamente com o pino do microcontrolador, indicando seu GPIO (aqui indicado somente como gp). Pode também ser um pino "*Analog*", que se conecta diretamente ao pino do canal serial no microcontrolador. Ou pode ser um pino "Virtual", que são posições de memórias, endereçadas como V0, V1, V2, ... , V255, que armazenam valores inteiros de 0 a 255. Se aconselha fortemente que os pinos sejam sempre virtuais ("*Virtual*"), pois isso garante maiores recursos durante a programação do módulo. Por fim, item ⑤, ao clicar no botão "play" o projeto é executado, neste momento ele indica que ainda não possui conexão com o módulo, apenas está disponibilizando seus pinos ao servidor Blynk.

Para inserção do código no módulo ESP8266 NodeMCU, é necessário incluir na IDE Arduino a biblioteca "Blynk by Volodymyr Shymanskyy", que pode ser inserida clicando na aba "Sketch", "Incluir Biblioteca" e "Gerenciar Bibliotecas", colocando na caixa de texto para busca a palavra "Blynk", selecionando a biblioteca indicada para sua instalação na IDE (PAVEL, 2021). A Figura 104 mostra como deve ocorrer esse procedimento.

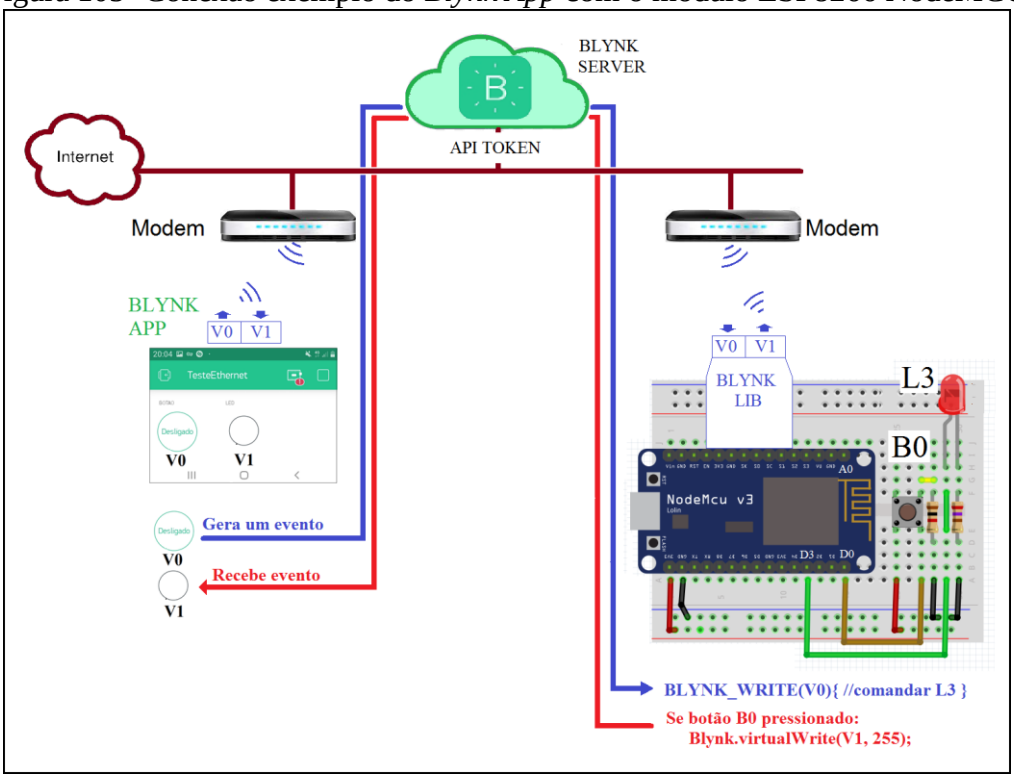
Figura 104- Processo de instalação do pacote de bibliotecas *Blynk* na IDE Arduino



Fonte: Autor (2021)

No que diz respeito ao código, a ser editado na IDE do Arduino, partindo do proposto na Figura 103, onde um botão no Blynk App irá colocar um valor de 0 ou 1 no pino virtual V0, segundo seu estado de acionamento, e onde um LED no Blynk App irá assumir a luminosidade posta no pino virtual V1, com valores de 0 a 255, o código exemplo de interação entre Blynk e módulo se propõe a monitorar o valor do pino virtual V0 e acionar o LED L3, segundo os valores 0 e 1 armazenados em V0. Assim como o botão B0 do módulo irá provocar o carregamento de valores de 0 e 255 no pino virtual V1, segundo sua condição de acionamento (CURTO CIRCUITO, 2018). A Figura 105 mostra esta proposta de exemplo que está sendo construída.

Figura 105- Conexão exemplo do *Blynk App* com o módulo ESP8266 NodeMCU



Fonte: Autor (2021)

A alimentação dos pinos virtuais ocorre por eventos. O pressionar de um botão no *Blynk App* provoca um evento que armazena um valor em um pino virtual e anuncia ao módulo ESP8266 NodeMCU que o evento ocorreu, indicando um desvio do fluxo de programa para uma função chamada BLYNK_WRITE(Pin), fornecendo como parâmetro o nome do pino virtual que foi atualizado.

O mesmo ocorre quando o módulo ESP8266 NodeMCU altera o valor de um pino virtual, pela função Blynk.virtual(Pin,valor), indicando como parâmetro o nome do pino e o valor a carregar. Esta função indica para o Blynk App que ele deva atualizar seu objeto vinculado ao pino na sua área de trabalho.

O programa seguinte implementa o exemplo de interação do módulo ESP8266 NodeMCU com o *Blynk App* com 1 botão e 1 LED.

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <BlynkSimpleEsp8266.h> //No gerenciador de bibliotecas: "Blink by Volodymyr Shymanskyy"
#define BLYNK_PRINT Serial
char ssid[] = "**********"; // nome da sua rede WIFI
char password[] = "**********"; // senha da sua rede WIFI
char auth[] = "**********"; //Token da aplicação enviado por email
#define L3 0  //LED pino D3 GPIO-0
#define B0 16 //Botao NA pino D0 (GPIO-16)

void setup(){
  Blynk.begin(auth, ssid, password);
  pinMode(L3, OUTPUT);
```

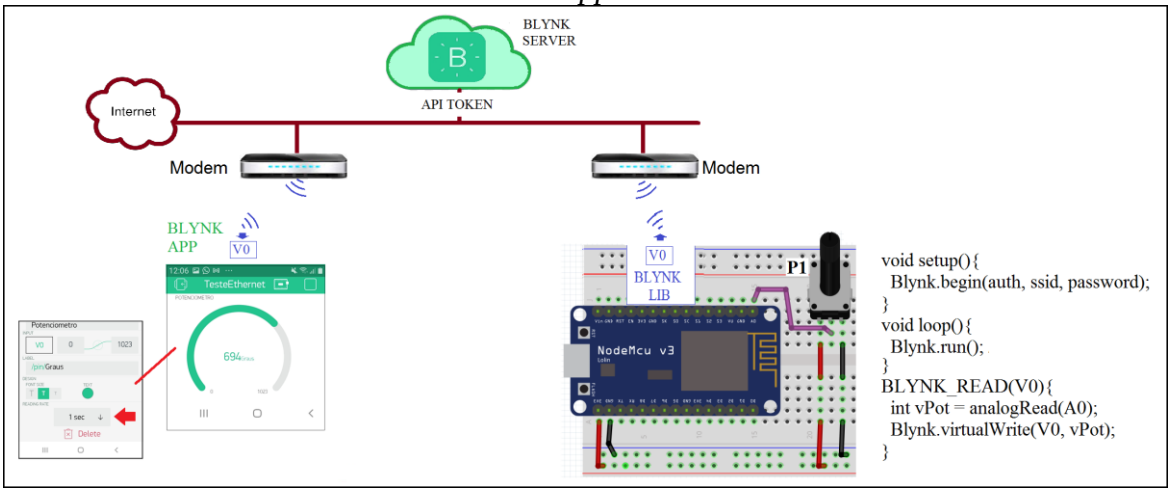
```
  pinMode(B0, INPUT);
}
void loop(){
  Blynk.run(); //Executa o Blynk
  enviarDadosBlynk();
}
//***Estas função são chamadas por evento no Blynk
BLYNK_WRITE(V0){ // Blynk app escreveu em V0
  int pinValue = param.asInt();
  if(pinValue == 0) digitalWrite(L3, LOW);
  if(pinValue == 1) digitalWrite(L3, HIGH);
}
void enviarDadosBlynk(){ //para empurrar para o Blynk a informação
  if(digitalRead(B0)== HIGH){
    Blynk.virtualWrite(V1,255); //255 para ligar na máx intensidade
  }else{
    Blynk.virtualWrite(V1,0); //0 para desligar
  }
}
```

Em se tratando de objetos e componentes ativos, como botões e chaves, os eventos são gerados de forma natural. O aplicativo ou o módulo só farão alguma coisa se o evento ocorrer.

Porém, nos objetos e componentes reativos, como os mostradores *displays*, não há uma ação natural para que ocorra o evento que irá disparar o carregamento do pino virtual e indicar para o Blynk App que deve atualizar seu objeto ou ao módulo ESP8266 NodeMCU.

Nesse caso, a primeira alternativa é utilizar um recurso do próprio objeto do Blynk App, que permite selecionar no campo “Reading Rate” um intervalo de tempo para que o próprio aplicativo gere um evento, o módulo é então informado e executa a função BLYNK_READ(pin), recebendo como parâmetro o nome do pino, a fim de que o programa do módulo atualize o valor desse pino virtual. A Figura 106 apresenta um exemplo de interação do Blynk com o módulo, sendo o valor puxado pelo Blynk App.

Figura 106 - Exemplo de interação *Blynk* com ESP8266 NodeMCU com valor puxado pelo *Blynk App*



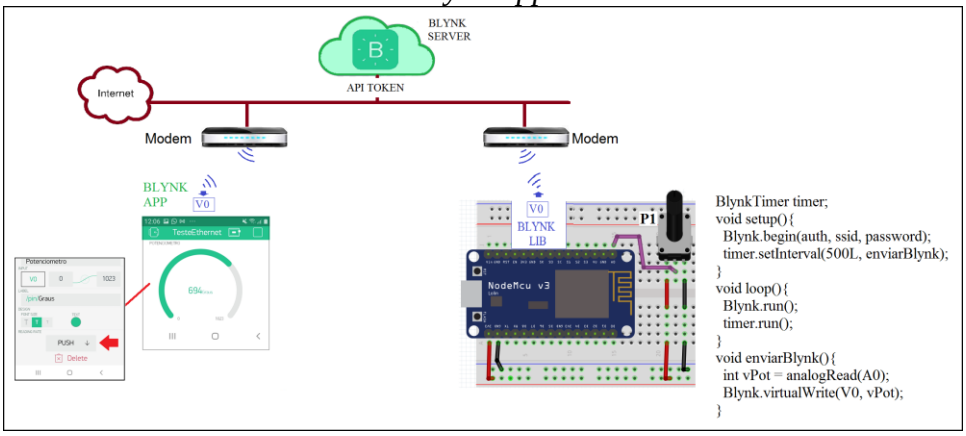
Fonte: Autor (2021)

O programa para interação de um valor analógico sendo puxado pelo Blynk App é descrito no código de programação seguinte:

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <BlynkSimpleEsp8266.h> //No gerenciador de bibliotecas: "Blink by Volodymyr Shymanskyy"
#define BLYNK_PRINT Serial
char ssid[] = "**********"; // nome da sua rede WIFI
char password[] = "**********"; // senha da sua rede WIFI
char auth[] = "**********"; //Token da aplicação enviado por email

void setup(){
  Blynk.begin(auth, ssid, password);
}
void loop(){
  Blynk.run(); //Executa o Blynk
}
BLYNK_READ(V0){//Para caso do Blynk puxar a informação
  int vPot = analogRead(A0);
  Blynk.virtualWrite(V0, vPot);
}
```

A outra alternativa é ativar um temporizador no módulo ESP8266 NodeMCU, já disponível na biblioteca Blynk, criando um objeto do tipo *BlynkTimer* com o nome de "*timer*", sendo ajustados seus parâmetros com a função timer.setInterval(TEMPO, FUNÇÃO), passando por parâmetro o valor do tempo em milisegundos e o nome da função que deverá ser executada ao ser contado o tempo do parâmetro. Dentro da função *loop*(), esse timer é executado, pela função *timer.run*() e, na função definida, se coloca o comando *Blynk.virtualWrite*(pin, valor) para enviar o valor para o pino virtual a cada tempo definido para função se repetir, sendo um aviso para o Blynk App atualizar seus objetos com o novo valor do pino virtual. O objeto que receberá o valor do pino virtual no Blynk App não deverá ter seu campo "*Reading Rate*" definido, devendo somente ter a indicação de "*push*". A Figura 107 apresenta esse procedimento de interação do *Blynk* com ESP8266 NodeMCU com um valor empurrado do módulo para o *Blynk App*.

Figura 107 - Exemplo de interação *Blynk* com ESP8266 NodeMCU com valor empurrado para o *Blynk App*



Fonte: Autor (2021)

O programa para interação de um valor sendo empurrado pelo Módulo ESP8266 NodeMCU ao Blynk App é o seguinte:

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <BlynkSimpleEsp8266.h> //No gerenciador de bibliotecas: "Blink by Volodymyr Shymanskyy"
#define BLYNK_PRINT Serial
char ssid[] = "**********";           // nome da sua rede WIFI
char password[] = "**********"; // senha da sua rede WIFI
char auth[] = "**********";         //Token da aplicação enviado por email
BlynkTimer timer; //Objeto timer do Blynk
void setup(){
  Blynk.begin(auth, ssid, password);
  timer.setInterval(500L, enviarBlynk); //a cada 500ms
}
void loop(){
  Blynk.run(); //Executa o Blynk
  timer.run();   //Executa o timer a cada tempo definido
}
void enviarBlynk(){  //para empurrar para o Blynk a informação
  int vPot = analogRead(A0);
  Blynk.virtualWrite(V0, vPot);
}
```

O Projeto 23 apresenta um exemplo completo de interação entre o módulo ESP8266 NodeMCU e o Blink App.

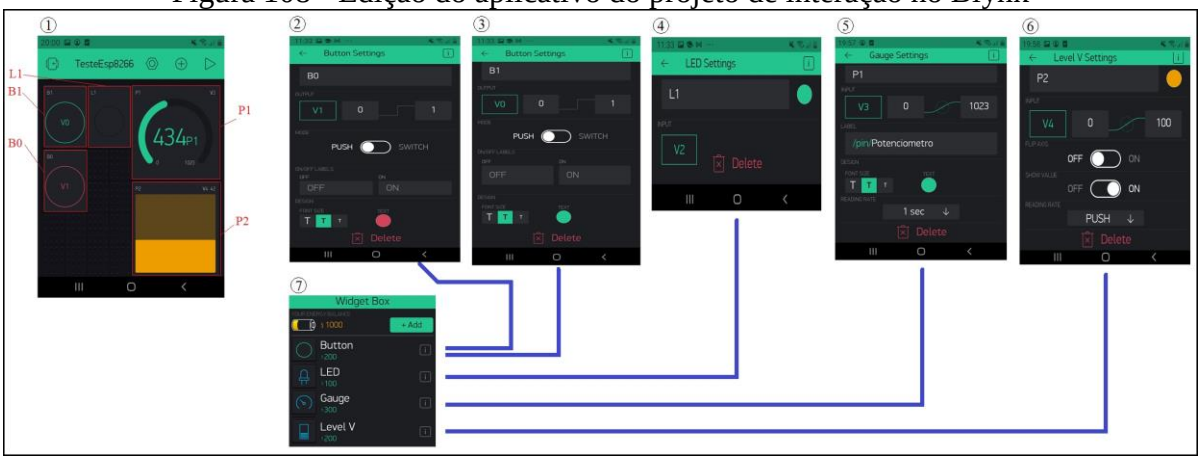
Já o Projeto 24 apresenta um sistema que monitora dados de uma residência através do Blink App, interagindo com módulo ESP8266 NodeMCU, sendo eles: temperatura, umidade, nível do reservatório de água e estado da central de alarme.

PROJETO 23 – INTERAÇÃO ENTRE ESP8266 NODEMCU E O BLYNK

"O projeto consiste em um módulo ESP8266 NodeMCU, conectado a um modem wifi, que acessa aplicativo Blynk, por uma API em código Token, onde os comandos do módulo refletem no Blink App com sinais visuais e gráficos, e os comandos no Blynk App comandam dispositivos conectados ao módulo"

O projeto é composto de um aplicativo *mobile Blink App*, onde foram implementados os botões B1(ligado ao pino virtual V0) e o B2 (ligado ao pino virtual V1). Ambos colocam o valor 1 nos respectivos pinos virtuais, quando pulsionados, e colocam o valor 0, caso desacionados. Há também o LED conectado ao pino virtual V2, que ajusta sua intensidade luminosa segundo o valor colocado no pino virtual V2, que pode ser um valor entre 0 (desligado) e 255 (totalmente ligado). Um mostrador tipo relógio (*Gauge*) apresenta de forma gráfica o valor do pino virtual V3, com valores de 0 a 1.023, escursionando em um modelo de relógio. Além de um mostrador tipo barra (*Bar Graph*) vinculado ao pino virtual V4, que recebe valores de 0 a 100. A Figura 108 apresenta a configuração dos objetos no aplicativo Blink App desenvolvido para o projeto.

Figura 108 - Edição do aplicativo do projeto de interação no Blynk



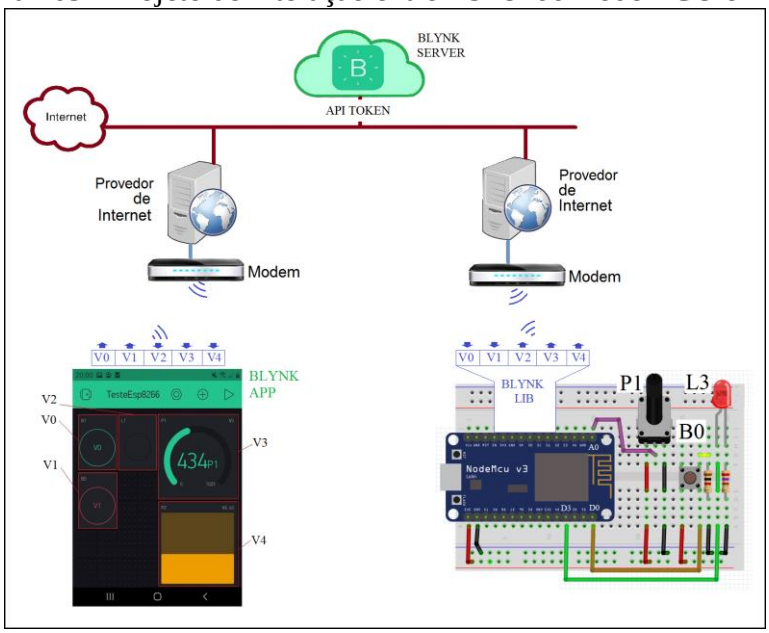
Fonte: Autor (2021)

Por parte do módulo ESP8266 NodeMCU, há um programa que faz a leitura do potenciômetro P1, ligado ao canal analógico A0 e, periodicamente, envia o valor convertido em uma escala de 0 a 100 para o pino virtual V3, sempre que o objeto do Blink APP o solicita. Envia também o valor do potenciômetro P1 na escala de 0 a 1.023, lida no canal analógico, para o pino virtual V4, numa periodicidade definida pelo objeto "*timer*", que também envia para o pino virtual V2 os valores 0 ou 255, segundo condição de acionamento do botão B0. Ainda no módulo, ocorre a leitura dos pinos

virtuais V0 e V1, sendo que se o pino V0 estiver com valor igual a 1, irá ligar o LED L3, caso o valor do pino V1 for igual a 1, então irá desligar o LED L3.

A Figura 109 apresenta a estrutura do projeto.

Figura 109 - Projeto de interação entre ESP8266 NodeMCU e Blynk



Fonte: Autor (2021)

O programa que deverá ser inserido no módulo ESP8266 NodeMCU é o seguinte.

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <BlynkSimpleEsp8266.h> //No gerenciador de bibliotecas: "Blink by Volodymyr Shymanskyy"
#define BLYNK_PRINT Serial
BlynkTimer timer; //Objeto timer do Blynk
char ssid[] = "**********"; // nome da sua rede WIFI
char password[] = "**********"; // senha da sua rede WIFI
char auth[] = "**********"; //Token da aplicação enviado por email
#define L3 0  //LED pino D3 GPIO-0
#define B0 16 //Botao NA pino D0 (GPIO-16)
void setup(){
  Serial.begin(115200);
  Blynk.begin(auth, ssid, password);
  timer.setInterval(500L, enviarDadosBlynk); //a cada 500ms
  pinMode(L3, OUTPUT);
  pinMode(B0, INPUT);
}
void loop(){
  Blynk.run(); //Executa o Blynk
  timer.run(); //Executa o timer a cada tempo definido
}
//***Estas função são chamadas por evento no Blynk
BLYNK_WRITE(V0){ // Blynk app escreveu em V0
  int pinValue = param.asInt();
  if(pinValue == 1) digitalWrite(L3, HIGH);
}
BLYNK_WRITE(V1){ // Blynk app escreveu em V1
  int pinValue = param.asInt();
  if(pinValue == 1) digitalWrite(L3, LOW);
```

```
}
BLYNK_READ(V3){ //Para caso do Blynk puxar a informação
  int valorPotenciometro = analogRead(A0);
  Blynk.virtualWrite(V3, valorPotenciometro);
}
void enviarDadosBlynk(){  //para empurrar para o Blynk a informação
  int valorPorcentual = map(analogRead(A0),0,1023,0,100);
  Blynk.virtualWrite(V4, valorPorcentual);
  if(digitalRead(B0)== HIGH){
    Blynk.virtualWrite(V2,255);  //valor 255 para ligar na máx intensidade no led
  }else{
    Blynk.virtualWrite(V2,0); //valor 0 para desligar o led
  }
}
```

PROJETO 24 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E BLYNK

"O projeto consiste em um módulo ESP8266 NodeMCU, conectado a um modem wifi, que acessa o aplicativo Blynk, por uma API em código Token, e envia os sinais de temperatura, umidade, nível do reservatório de água e condição do alarme de uma residência."

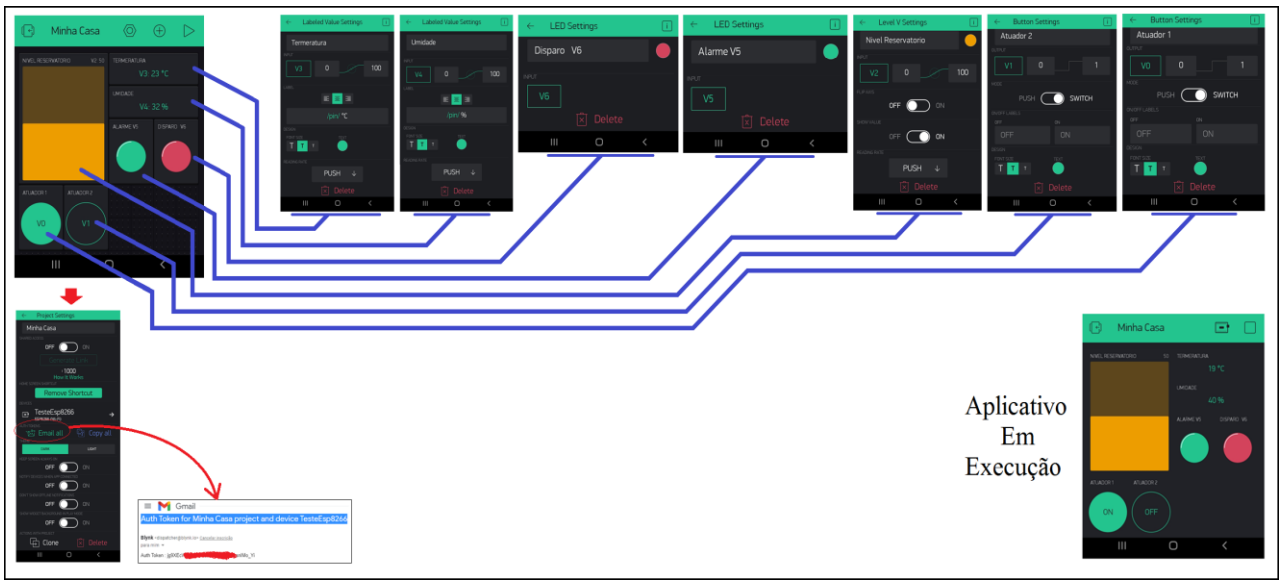
O projeto é composto de um aplicativo mobile Blink App, onde foi implementada uma área de trabalho, em que há os seguintes objetos:

a) Botões do tipo chave ligados aos pinos virtuais V0 e Vi, respectivamente, que, quando acionados, colocam o valor 1 no seu pino e, quando desacionados, colocam o valor 0, uma ação que deve ser reconhecida no módulo ESP8266 NodeMCU para o comando de relés que, através de seus contatos, podem comandar cargas, tais como: sistema de irrigação, luzes, alimentadores de animais domésticos, entre outros;
b) Barra de nível, vinculada ao pino virtual V2, que mostra graficamente o valor de 0 a 100, uma indicação alusiva ao nível de um reservatório de água, uma vez que o módulo ESP8266 NodeMCU envie esses valores a partir do sensoriamento físico do reservatório;
c) *Displays* numéricos vinculados aos pinos virtuais V3 e V4, que mostram numericamente valores de temperatura e umidade, em uma escala de 0 a 100, alimentados pelo módulo ESP8266 NodeMCU, que monitoram essas grandezas físicas por um sensor DHT11;
d) LEDs vinculados aos pinos virtuais V5 e V6, que mostram a situação de uma central de alarme residencial indicando, respectivamente, se o alarme está ligado ou desligado e se o alame está em situação normal ou diparado, uma vez que o módulo ESP8266 NodeMCU se interfacea à central de alarme por relés, capturando esses dois dados, a serem exibidos no aplicativo *Blynk*.

Uma situação a ser considerada é que o sensor de nível ligado ao módulo é oriundo de um potenciômetro conectado mecanicamente ao um dispositivo de bóia, dentro do reservatório, que ao mudar o nível faz com que a bóia mecanicamente suba ou abaixe, fazendo o potenciômetro rotacionar e provocar uma variação elétrica na entrada analógica do módulo.

A Figura 110 apresenta a área de trabalho do aplicativo criado no *Blynk App*, com a configuração de cada um dos componentes.

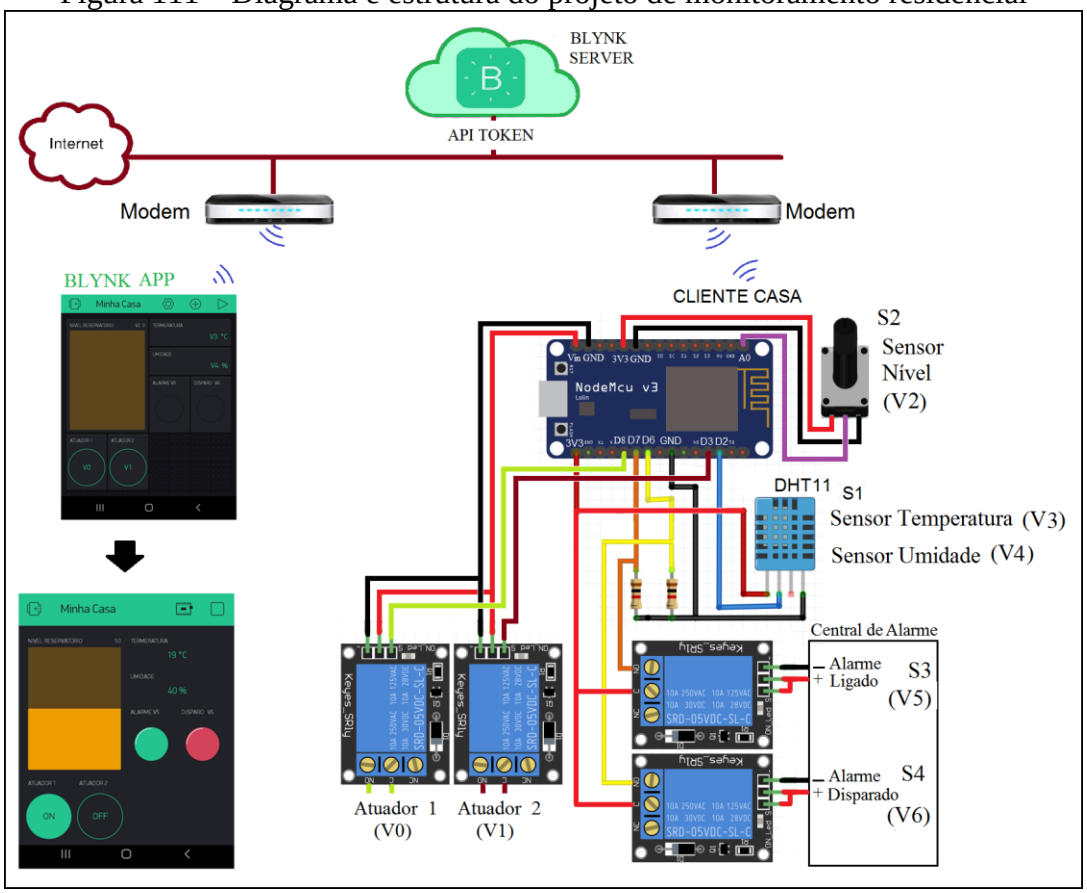
Figura 110 – Configuração dos componentes do aplicativo criado no Blynk App para o projeto de monitoramento residencial



Fonte: Autor (2021)

A Figura 111 apresenta o diagrama e estrutura do projeto.

Figura 111 – Diagrama e estrutura do projeto de monitoramento residencial



Fonte: Autor (2021)

O código de implementação o projeto monitoramento de residência com ESP8266 NodeMCU e Blynk é o seguinte.

```
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas: "ESP8266WIFI by Ivan Grokhotkov"
#include <BlynkSimpleEsp8266.h> //No gerenciador de bibliotecas: "Blink by Volodymyr Shymanskyy"
#define BLYNK_PRINT Serial
BlynkTimer timer; //Objeto timer do Blynk
char ssid[] = "**********";// nome da sua rede WIFI
char password[] = ""**********";// senha da sua rede WIFI
char auth[] = ""**********";//Token da aplicação enviado por email
#include <DHT.h> //disponível em https://drive.google.com/file/d/16brDLNmOVzmQs6sFh-_CAMivQu-cn6kg/view
#define DHTPIN 4 // pino do sensor em D2 - GPIO4 - SDA
#define DHTTYPE DHT11 //tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
#define Atuador1 15  //Relé pino D8 GPIO-15, pino virtural V0
#define Atuador2 0  //Relé pino D3 GPIO-0, pino virtural V1
#define pinNivel A0 //Potenciômetro ligado a AO (GPIO-0)
#define Alarme 12 // Contato relé indicando estado do alarme, D6 (GPIO-12), pino virtual V5
#define Disparo 13 // Contato relé indicando alarme disparado, D7 (GPIO-13), pino virtual V6

void setup(){
  Blynk.begin(auth, ssid, password);
  timer.setInterval(2000L, enviarBlynk); //a cada 2s
  pinMode(Atuador1, OUTPUT);
  pinMode(Atuador2, OUTPUT);
  pinMode(pinNivel, INPUT);
  pinMode(Alarme, INPUT);
  pinMode(Disparo, INPUT);
}

void loop(){
  Blynk.run(); //Executa o Blynk
  timer.run(); //Executa o timer a cada tempo definido
}
//***Estas função são chamadas por evento no Blynk
BLYNK_WRITE(V0){// Blynk app escreveu em V0
  int pinValue = param.asInt();
  if(pinValue == 0) digitalWrite(Atuador1, LOW);
  if(pinValue == 1) digitalWrite(Atuador1, HIGH);
}
BLYNK_WRITE(V1){// Blynk app escreveu em V1
  int pinValue = param.asInt();
  if(pinValue == 0) digitalWrite(Atuador2, LOW);
  if(pinValue == 1) digitalWrite(Atuador2, HIGH);
}
//**********************************
void enviarBlynk(){ //para empurrar para o Blynk a informação
  int Nivel = map(analogRead(pinNivel),0,1023,0,100);
  Blynk.virtualWrite(V2, Nivel); //valor de Nível, pino virtual V2
  int Temperatura = dht.readTemperature(); //Le a temperatura
  Blynk.virtualWrite(V3, Temperatura); //valor de temperatura, pino virtual V3
  int Umidade = dht.readHumidity(); //Le a Umidade
  Blynk.virtualWrite(V4, Umidade); //valor de Umidade, pino virtual V4
  if(digitalRead(Alarme) == HIGH){
    Blynk.virtualWrite(V5, 255); //Alarme Ligado
  }else{
    Blynk.virtualWrite(V5, 0); //Alarme Desligado
  }
  if(digitalRead(Disparo) == HIGH){
    Blynk.virtualWrite(V6, 255); //Alarme Disparado
  }else{
    Blynk.virtualWrite(V6, 0); //Alarme Normal
  }
}
```

6.3 INTERAÇÃO COM ADAFRUIT IO VIA MQTT

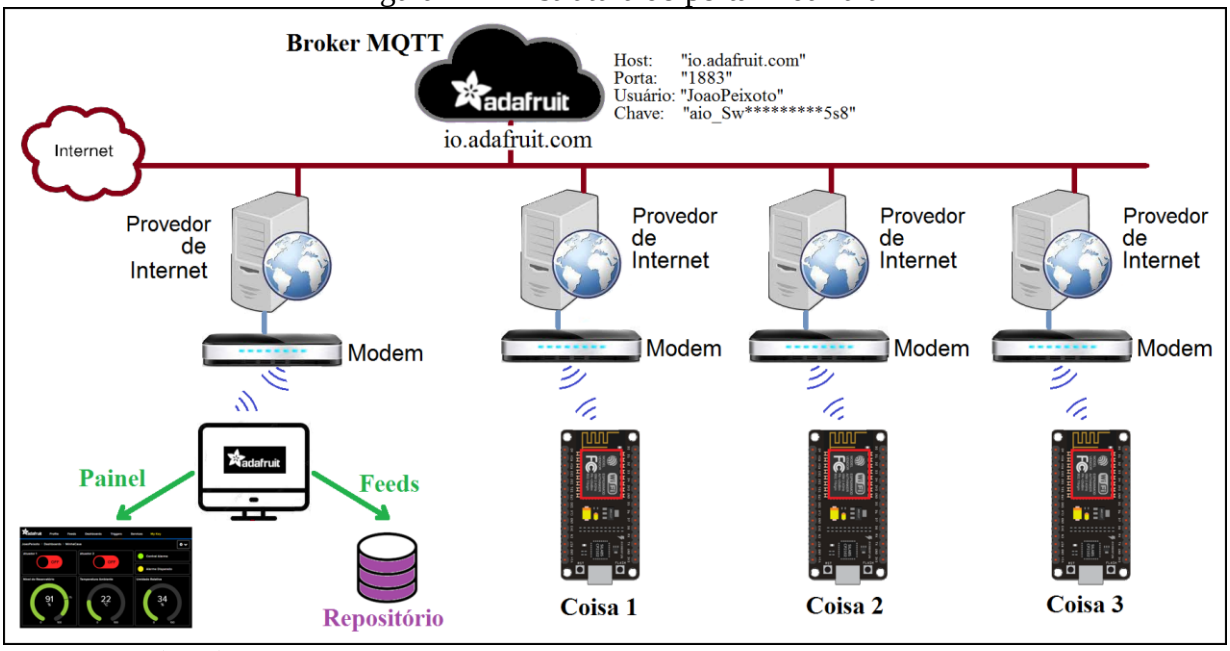
O portal Adafruit é uma aplicação de internet das coisas que envolve 3 funcionalidades a serem disponibilizadas: é um *broker* de protocolo MQTT (*host*:“io.adafruit.com”); um sistema de supervisão que permite criação de painéis de exibição de variáveis na forma gráfica e é também um sistema repositório em nuvem, onde os dados são recebidos e enviados pelo sistema de supervisão, via Broker MQTT, onde são armazenados com informações de valor, data e hora em que ocorreu a postagem, o que permite ter um histórico de dados das variáveis de um projeto (LOCATELLI, 2020b).

É uma ferramenta comercial, que permite a aquisição de pacotes de acesso aos dados distintos, adequando-se à complexidade do projeto a que se propõe. Mas é disponibilizada uma versão gratuita, que dá conta de aplicações básicas, com objetivo de aprendizado. Essa versão gratuita do portal Adafruit permite (ADAFRUIT, 2021):

a) 30 entradas de dados por minuto;
b) 30 dias de armazenamento de dados;
c) Leituras a cada 15 minutos;
d) Implementação de 10 *feeds* (armazenadores de tópicos MQTT);
e) Implementação de 5 *dashboard* (painéis de monitoramento).

A Figura 112 apresenta a estrutura do portal Adafruit.

Figura 112 - Estrutura do portal Adafruit



Fonte: Autor (2021)

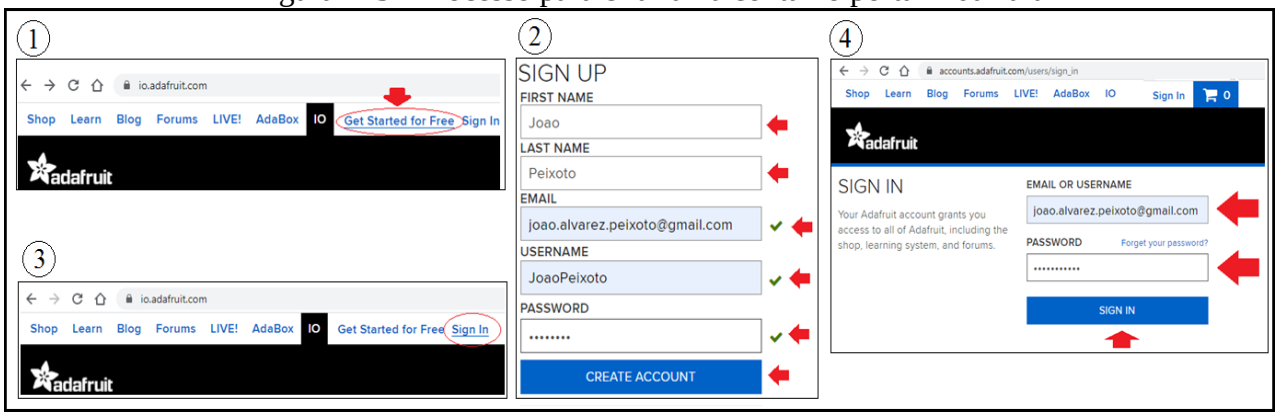
Nessa estrutura, as “coisas” acessam o Broker MQTT Adafruit, através do seu endereço de host, sua porta de acesso, seu código de usuário e chave de segurança, itens obtidos no momento da criação da conta no portal Adafruit. E, neste acesso, as coisas postam ou assinam tópicos MQTT, obtendo e informando dados.

No portal Adafruit, em uma conta cadastrada, há um painel (*dashboard*) onde os dados publicados no Broker podem ser lidos por objetos de exibição gráfica instantaneamente e publicar dados através de objetos ativos. Nesse mesmo portal, os dados assinados e publicados (*Feeds*) são armazenados em um repositório, catalogados com a data e horário que foram apresentados, mantendo um histórico da aplicação, acessível ao usuário do portal.

Dessa forma, o portal Adafruit se apresenta como uma boa solução para aplicações que, além de supervisionar e comandar sistemas de forma remota em IoT, necessitem manter um histórico das variáveis, a fim de verificação ou correção de um processo que se deseja automatizar.

Então, tudo se inicia com a criação de uma conta no portal Adafruit. A Figura 113 apresenta o processo de criação de uma conta ou acesso a uma conta já cadastrada.

Figura 113 - Processo para criar uma conta no portal Adafruit

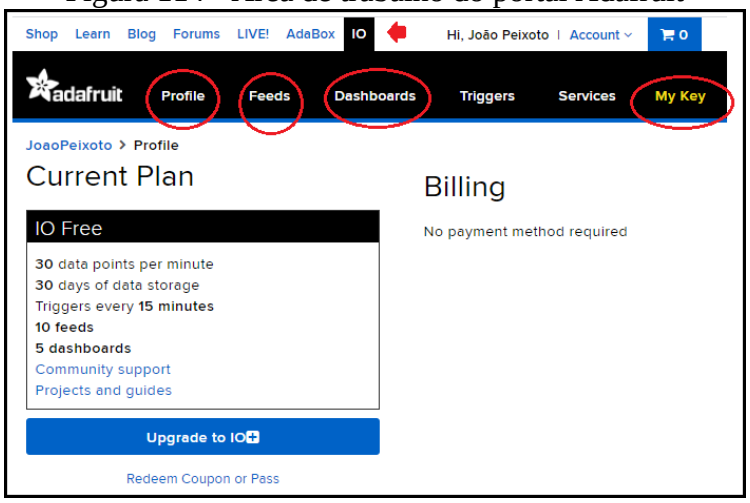


Fonte: Autor (2021)

Inicialmente, deve se acessar o *site* https://io.adafruit.com e clique em “*Get Started for Free*”, item ① da figura. Será aberta uma caixa de parâmetros, item ②, onde deverá ser inserida sua identificação (*FIRST NAME*, *LAST NAME*), endereço válido de email, nome de usuário (*USERNAME*) e deverá atribuir uma senha para seu acesso, clicando ao final no botão “*CREATE ACCOUNT*”. Caso já possua uma conta e necessite acessar novamente o portal, deve-se clicar em “*Sign In*”, item ③, que abrirá uma caixa de texto para inserção do e-mail e senha cadastrados, item ④.

A Figura 114 apresenta a área de trabalho do portal Adafruit, após a criação da conta ou acesso a ela. Nela há a aba "IO", que remete a uma tela inicial com algumas opções, sendo que as 4 mais importantes para esse primeiro momento são: *Profile*, *Feeds*, *Dashboards* e *My Key*.

Figura 114 - Área de trabalho do portal Adafruit



Fonte: Autor (2021)

Na aba "*Profile*" (perfil) é possível verificar as informações da conta, número de feeds utilizados, número de painéis, taxa de transmissão em uso, todos dados relativos à conta.

Na aba "*Feeds*" (armazenador) da área de trabalho é possível criar ou visualizar os tópicos do projeto. Um *feed* é uma posição de memória que recebe os dados oriundos de uma postagem ou de uma submissão. A característica principal de um *feed* é que seus dados ficam armazenados no repositório, segundo data e hora que seu valor foi alterado. Cada componente *widgets*[42] do painel de visualização necessita estar vinculado a um *feed*.

Uma aplicação Adafruit começa com a criação dos *feeds*. A Figura 115 apresenta o processo de criação dos *feeds*, com objetivo de armazenar os dados da aplicação no repositório.

[42] Um componente *widget*, numa interface gráfica, é um elemento de interação, tal como janelas, botões, menus, ícones, barras de rolagem, entre outros. O termo pode também se referir aos pequenos aplicativos que flutuam pela área de trabalho e fornecem funcionalidades específicas ao utilizador (previsão do tempo, cotação de moedas, relógio, entre outros). Fonte: https://pt.wikipedia.org/wiki/Widget.

Figura 115 - Processo para criar os *Feeds*, armazenadores de dados em repositório

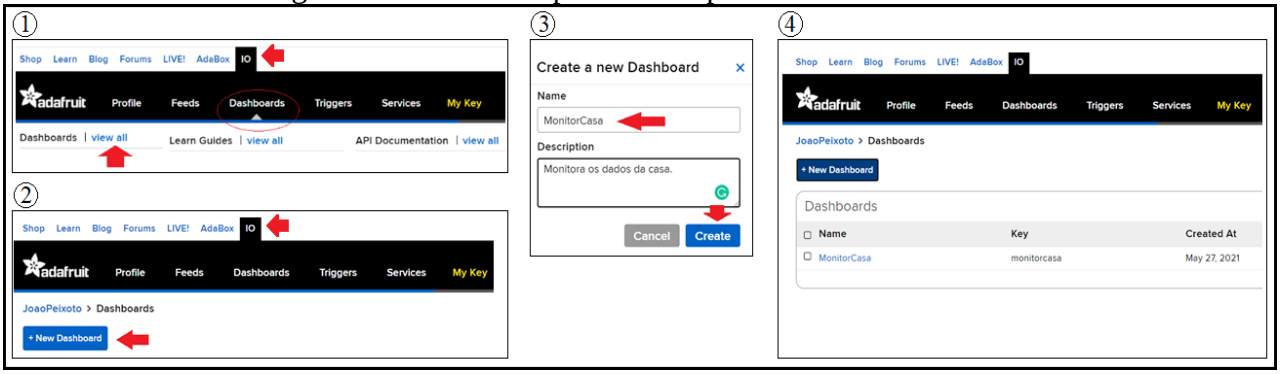
Fonte: Autor (2021)

Na tela "IO", deve-se clicar na aba "*Feeds*" que, para uma primeira aplicação deverá estar vazia. No item ① da figura clique em "*view all*" para entrar na tela de visualização dos *Feeds*. Dentro do grupo "*Default*", escolha criar um *feed*, clicando em "*+New Feed*", item ② da figura. Surgirá uma caixa de parametrização, item ③ e ④, onde se deve colocar um nome ("*Name*") para o feed e clicar no botão "*Create*". A proposta deste exemplo é criar o feed lâmpada e nível, sendo que seus respectivos tópicos MQTT serão "JoaoPeixoto/feeds/lampada" e "JoaoPeixoto/feeds/nivel". O padrão de formação de um tópico Adafruit MQTT é "<usuário>/feeds/<nome_do_feed>".

Na aba "*Dashboards*" (painel) da área de trabalho, é possível criar ou visualizar os painéis compostos de objetos *widgets*. Quando um *widget* é do tipo reativo, ele apresenta de forma gráfica o valor da variável *feed* a qual está vinculado. Quando um *widget* é do tipo ativo, ele altera o valor da variável *feed* a qual está vinculado, segundo ação do usuário no painel.

Um *dashboard* é um painel que fornece um conjunto de informações de forma visual, organizadas de forma a tornar claras e precisas as informações relativas a um processo. Os *dashboards* Adafruit irão mostrar os *widgets*, em um leiaute definido pelo programador do painel. Dessa forma, os feeds serão apresentados graficamente no painel, à medida que são alimentados pelas postagens dos clientes MQTT. A Figura 116 mostra o processo de criação de um *dashboard* no portal Adafruit.

Figura 116 - Processo para criar o painel visual *dashboard*

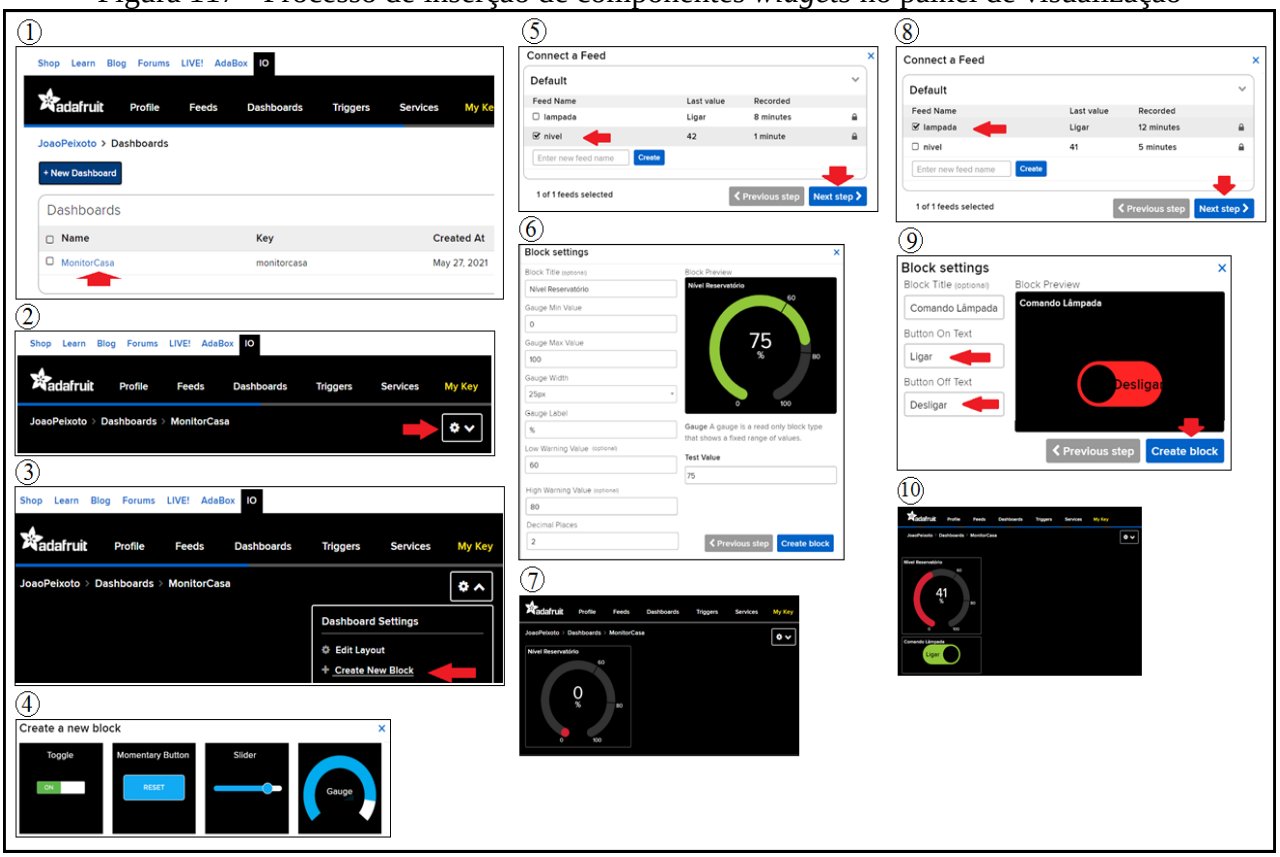


Fonte: Autor (2021)

Na tela “IO”, deve-se clicar na aba “*Dashboard*”, em que será mostrada a tela com os *dashboards* já criados, item ① da figura, clicando em “*view all*”. Na tela do item ②, deve-se clicar em “+*New Dashboard*”, onde será exibida a caixa de inserção de parâmetros, item ③, onde se deve inserir o nome para o painel (“*Name*”) e clicar em “*Create*”. Será exibida a tela com o painel criado, item ④.

Uma vez criado um *dashboard*, faz-se necessária a inserção dos *widgets*. A Figura 117 apresenta o processo de inserção de componentes *widgets* no painel de visualização.

Figura 117 - Processo de inserção de componentes *widgets* no painel de visualização



Fonte: Autor (2021)

Na tela “IO”, deve-se clicar sobre o painel já criado, neste exemplo o painel “MonitorCasa”, item ①. É apresentada a área de trabalho do painel, onde serão inseridos os objetos *widgets*, item ② , clicando no botão de seleção indicado. Nessa aba de seleção deve-se clicar em “*Create New Block*”, item ③. Serão exibidos os *widgets* disponíveis, item ④, onde, para esse exemplo de aplicação, deverãoser selecionados os objetos “*Gauge*” e “*Toggle*”.

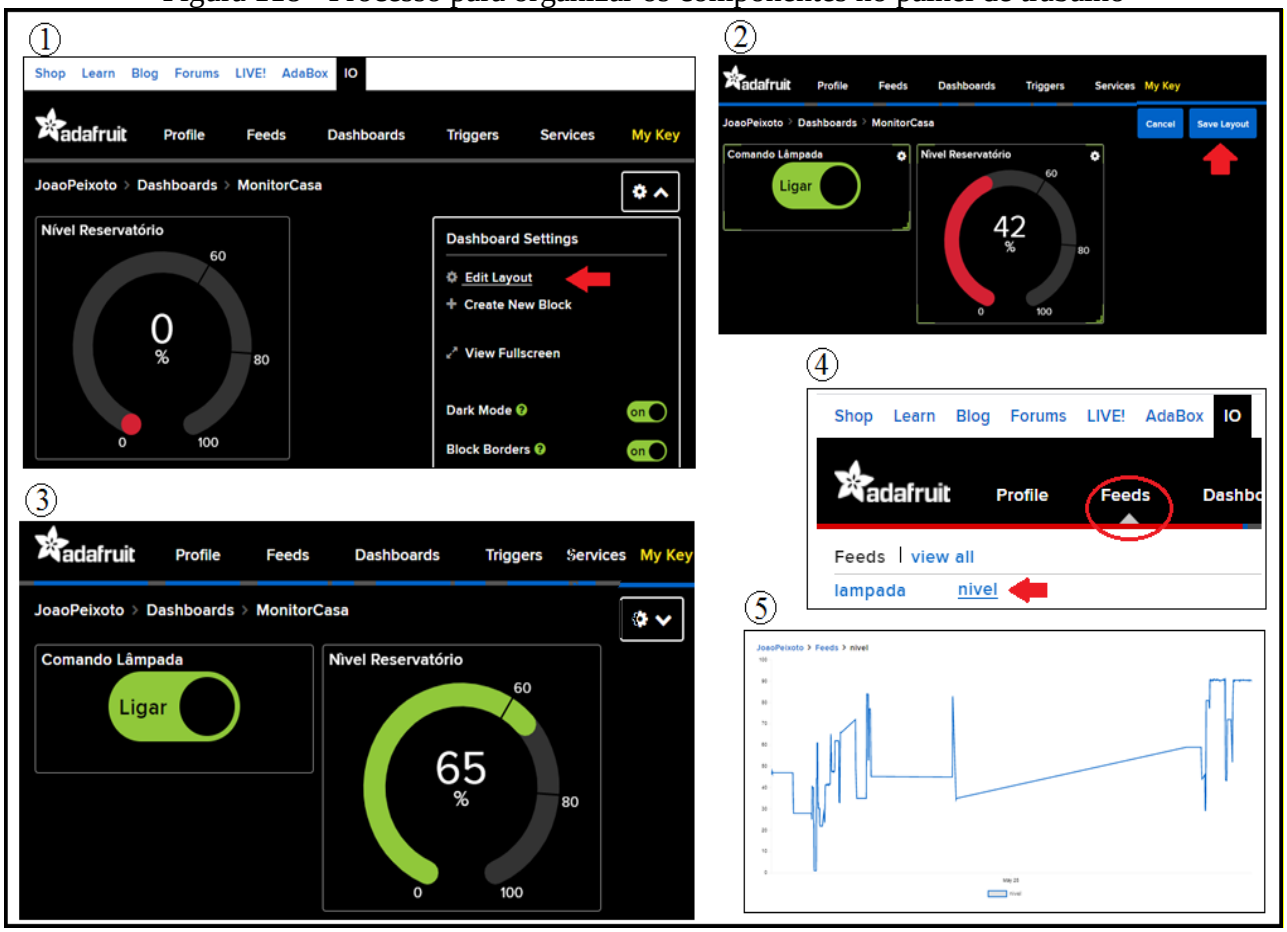
Uma vez selecionado o componente “*Gauge*”, será apresentada uma caixa para selecionar o *feed* que será vinculado a esse componente, item ⑤, selecionando “nível” e clicando em “*Next step*”. Será aberta uma caixa de parametrização, item ⑥, onde devem ser inseridos o nome do componente

(“*Block Title*”), o valor mínimo (“*Gauge Min Value*”), o valor máximo (“*Gauge Max Value*”) e a unidade de medida (“*Gauge Label*”). Nesse componente, é possível inserir limites inferiores e superiores para o valor apresentado, o que provoca a troca da cor do gráfico quando os limites são ultrapassados. Ao fim, deve-se clicar em “*Create Block*” e o *widget* é apresentado na tela, item ⑦.

Uma vez selecionado o componente “*Toggle*”, item ⑧, é aberta uma caixa diálogo para selecionar o *feed* a ser vinculado, devendo selecionar “lâmpada” para esse exemplo. É então aberta uma caixa de parametrização, item ⑨, onde se deve inserir o nome do componente (“*Block Title*”), o texto que aparecerá quando o botão for ligado e que será publicado no tópico MQTT (“*Botton On Text*”) e o texto que aparecerá quando o botão for desigado e que será publicado no tópico MQTT (“*Botton Off Text*”), por fim, deve-se clicar no botão “*Create block”*. O botão será apresentado na área de trabalho do painel “MonitorCasa”, conforme item ⑩.

Para mudar o leiaute dos componentes no painel, deixando o visual mais sugestivo, deve-se proceder como apresentado na Figura 118, onde há o processo para organizar os componentes no painel de trabalho.

Figura 118 - Processo para organizar os componentes no painel de trabalho

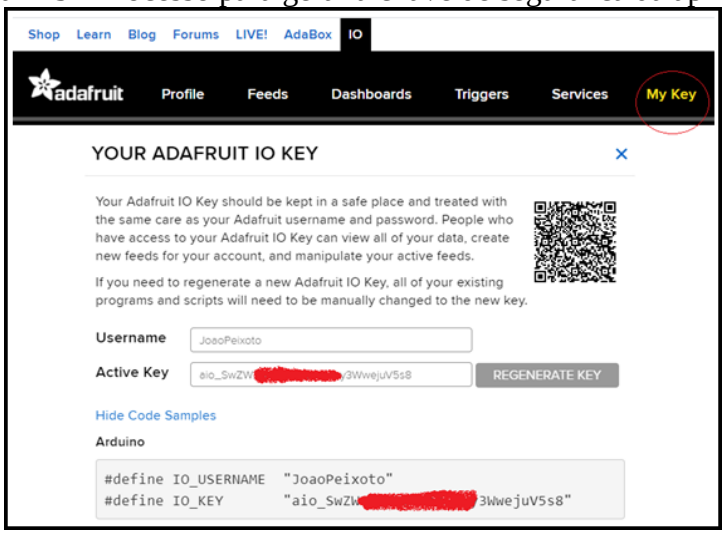


Fonte: Autor (2021)

Dentro do painel criado, neste exemplo o painel "MonitorCasa", deve-se abrir a aba de seleção de configurações, item ①, clicando sobre "*Edit Layout*". Agora, o painel permitirá que cada componente seja movido na tela, assim como ter seu tamanho alterado, item ②. Ao final do ajuste, deve-se clicar em "*Save Layout*" para finalizar, apresentando a tela final no item ③. Nesse momento, já se tem o monitoramento instantâneo das variáveis *feed* postas no painel. Caso seja necessário monitorar o histórico de dados da variável feed, na tela "IO", aba "*Feeds*", deve-se clicar sobre o nome do *feed* que se deseja verificar, item ④, será apresentada uma tela, item ⑤, com o histórico gráfico dos eventos e uma lista de data e hora que ocorreu cada alteração na variável, quando de sua publicação.

Na aba "*My Key*" (minha chave) da área de trabalho, é possível verificar a chave de segurança, que é criada para cada conta, sendo necessário conhecimento por parte dos clientes MQTT para poder ter acesso à aplicação. Para isso é necessário clicar na tela "IO" e aba "*My Key*". A Figura 119 apresenta esse processo.

Figura 119 - Processo para gerar a chave de seguranca da aplicação

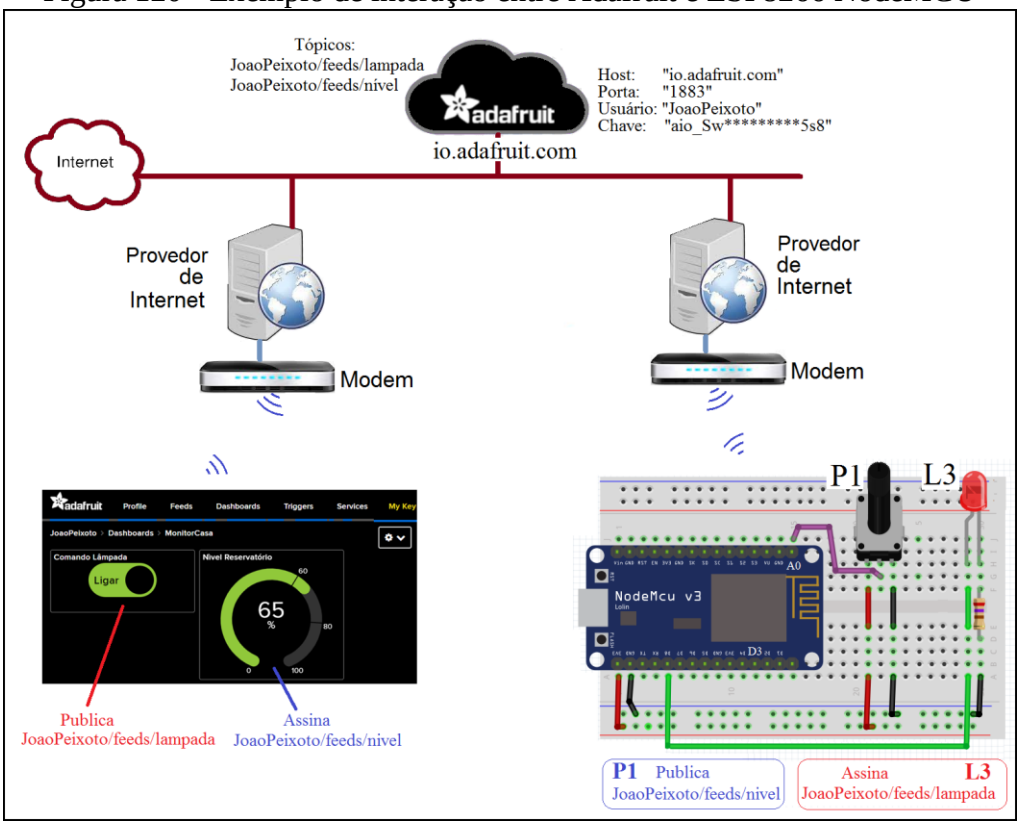


Fonte: Autor (2021

Esse código de chave de segurança (*Active Key*) e nome do usuário (*Username*), além do endereço de *host* ("io.adafruit.com") e porta de acesso (*Port*: 1883), são as informações que o módulo ESP8266 NodeMCU deve ter para poder publicar e assinar os tópicos MQTT "JoaoPeixoto/feeds/nível" e "JoaoPeixoto/feeds/lampada", respecivamente.

A Figura 120 traz o diagrama e estrutura desse exemplo de interação Adafruit e módulo ESP8266 NodeMCU.

Figura 120 - Exemplo de interação entre Adafruit e ESP8266 NodeMCU



Fonte: Autor (2021)

Para começar a edição do código de execução deste exemplo de interação Adafruit com módulo ESP8266 NodeMCU, é necessária a instalação de um pacote de bibliotecas específico para interação Adafruit em MQTT. Esse pacote deve ser procurado na IDE Arduino, clicando na aba "*Sketch*", "Incluir Biblioteca" e "Gerenciar Biblioteca", procurando na caixa de busca pelo nome de "adafruit mqtt". Deverá ser encontrada a biblioteca Adafruit MQTT Library by Adafruit, clicando então no botão "Instalar". Com isso, se poderá utilizar as bibliotecas Adafruit_MQTT.h e Adafruit_MQTT_Client.h. A Figura 121 apresenta o processo de inserção do pacote de biblioteca Adafruit MQTT na IDE Arduino.

Figura 121 - Inserção do pacote de biblioteca Adafruit MQTT na IDE Arduino

Fonte: Autor (2021)

Na implementação do programa para o módulo, é necessário conhecer algumas linhas de código importantes.

Para conexão com o *modem* via *wifi*, é necessária a inclusão da biblioteca **ESP8266WiFi.h**, onde na função **conectarWiFi()**, inicialmente é monitorado se há conexão. Caso não haja, uma conexão é estabelecida pela função WiFi.*begin*(), informando, por parâmetro, o nome da rede e sua senha. Na sequência, é monitorado se a conexão ocorreu. Esse processo de conexão se dá dentro da função *setup*(), com o chamamento da função conectarWiFi() e também dentro da função *loop*(). Dessa forma, caso a conexão por algum motivo se desfaça, será instigada uma nova conexão.

```
#include <ESP8266WiFi.h>
void setup(){
     conectarWiFi();
void loop(){
    conectarWiFi();
void conectarWiFi(){
     if (WiFi.status() == WL_CONNECTED) return;
     WiFi.begin("NOME_REDE","SENHA_REDE");
     while (WiFi.status() != WL_CONNECTED) delay(500);
```

Para configurar o acesso ao Broker MQTT Adafruit:

```
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client,”HOST”,<PORT>,“NOME_USUARIO”,
“CHAVE_SEGURANÇA”);
void conectarBroker(){
  int8_t ret;
  if(mqtt.connected()) return;
  while ((ret = mqtt.connect()) != 0)  delay(5000);
void loop(){
  conectarBroker();
```

Para publicação e assinatura de tópicos no Broker MQTT Adafruit, é necessário criar os objetos Adafruit_MQTT_Publish e Adafruit_MQTT_Subscribe, passando, por parâmetro, os respectivos tópicos para publicação e assinatura. Dentro da função *setup(),* é então realizada a assinatura do tópico “JoaoPeixoto/feeds/lâmpada” através da função *mqtt.subscribe*(&lâmpada). E, ainda dentro dessa função, é ativada a função *setCallback(),* que passa por parâmetro o nome da função que deverá ser executada quando uma publicação ocorrer no tópico assinado, indicado pelo objeto “lâmpada”.

```
Adafruit_MQTT_Publish nivel = Adafruit_MQTT_Publish(&mqtt, "JoaoPeixoto/feeds/nivel");
Adafruit_MQTT_Subscribe lampada =Adafruit_MQTT_Subscribe(&mqtt,"JoaoPeixoto/feeds/lampada");
void setup(){
```

```
mqtt.subscribe(&lampada);
lampada.setCallback(callbackLampada);
```

O recebimento de envios do Broker MQTT Adafruit, na assinatura do tópico, ocorrerá pela chamada do programa indicado, neste exemplo callbackLampada(). Essa função receberá por parâmetro o dado que foi publicado, que deverá ser comparado com uma referência, para proceder uma ação ou outra.

```
void callbackLampada(char *data, uint16_t len){
    String estado = data;
    if(estado.equals("Ligar")) {//executar ação}
    if(estado.equals("Desligar")) {//executar ação}
```

Para publicar tópicos no Broker MQTT Adafruit, é necessário o comando *publish()*, passando, por parâmetro, o valor a ser publicado no tópico, indicado por um objeto, como exemplo, nivel.*publish*(valor). Esse comando é executado dentro de uma função, que é periodicamente chamada pela função *loop*(). Mas, como fora executado na função *loop()* o comando mqtt.*processPackets*(), que passa o parâmetro de valor em milissegundos, que indica o tempo em que ocorrerá publicações de pacotes no *Broker*.

```
void publicarDados(){
     nivel.publish(valor);
void loop(){
    publicarDados();
    mqtt.processPackets(5000);
```

Por fim, a função *loop*() executa o comando mqtt.*ping*(), a fim de forçar uma resposta do *broker*, garantindo que a conexão ainda está ocorrendo. Caso não haja resposta satisfatória, executa então o comando mqtt.*disconect*(), que exigirá uma nova solicitação de conexão quando a função conectarBroker() for solicitada.

```
void loop(){
    conectarBroker();
    mqtt.processPackets(5000);
   publicarDados();
   if(! mqtt.ping()) mqtt.disconnect();
```

O código completo de programapara executar esse exemplo é o seguinte.

```
//************ CLIENTE CASA ********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <Adafruit_MQTT.h> //No gerenciador de bibliotecas "Adafruit MQTT Library by Adafruit"
#include <Adafruit_MQTT_Client.h> //incluso no pacote de biblioteca Adafruit

#define SSID "**********" //Nome da rede WiFi
#define PASS "**********" //Senha da rede Wifi
#define AIO_SERVER "io.adafruit.com" //Servidor MQTT
#define AIO_SERVERPORT 1883 //Porta do Servidor
#define AIO_USERNAME "JoaoPeixoto" //Nome do usuário na Adafruit.IO
#define AIO_KEY "**********" //Chave de segurança Adafruit IO
WiFiClient client; //classe do WiFiClient
```

```
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

//O padrão para subscrição e publicação é: <username>/feeds/<feedname>
Adafruit_MQTT_Publish nivel = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/nivel");
Adafruit_MQTT_Subscribe lampada = Adafruit_MQTT_Subscribe(&mqtt,AIO_USERNAME "/feeds/lampada");
#define L8 15 //LED L8 ligado ao pino D8(GPIO-15)
#define P1 0 //Potenciômetro P1 ligado ao pino A0(GPIO-0)
long tempoAnterior = 0;

void setup(){
  Serial.begin(115200);  //baudrate do canal serial
  pinMode(L8, OUTPUT);
  pinMode(P1, INPUT);
  conectarWiFi();
  mqtt.subscribe(&lampada);
  lampada.setCallback(callbackLampada);
}
void conectarWiFi(){
  if (WiFi.status() == WL_CONNECTED) return; //se já está conectado, retorna
  Serial.println();
  Serial.print("Conectando-se na rede ");
  Serial.println(SSID);
  WiFi.begin(SSID, PASS);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Conectado com sucesso: IP= ");
  Serial.println(WiFi.localIP());
}
void conectarBroker(){
  int8_t ret;
  if(mqtt.connected()) return;
  Serial.print("Conectando-se ao broker mqtt...");
  while ((ret = mqtt.connect()) != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    Serial.println("Falha ao se conectar. Tentando se reconectar em 5 segundos.");
    mqtt.disconnect();
    delay(5000);
  }
  Serial.println("Conectado ao broker com sucesso.");
  mqtt.subscribe(&lampada);
}
//função que recebe mensagens dos tópicos assinados
void callbackLampada(char *data, uint16_t len){
  String estado = data;
  if(estado.equals("Ligar"))digitalWrite(L8,HIGH);
  if(estado.equals("Desligar")) digitalWrite(L8,LOW);
}
//função para publicar em tópicos
void publicarDados(){
 int valNivel = map(analogRead(P1),0,1023,0,100);
 nivel.publish(valNivel); // mas só ocorrerá a cada 5s pois -> mqtt.processPackets(5000);
}

void loop(){
  conectarWiFi(); //caso não haja conexão, refaz
  conectarBroker();//caso não haja conexão, refaz
  mqtt.processPackets(5000);// Função responsável por ler e enviar dados a cada 5s
  publicarDados(); //envia o valor de P1
  if(! mqtt.ping()) mqtt.disconnect();//executa ping para manter a conexão ativa
}
```

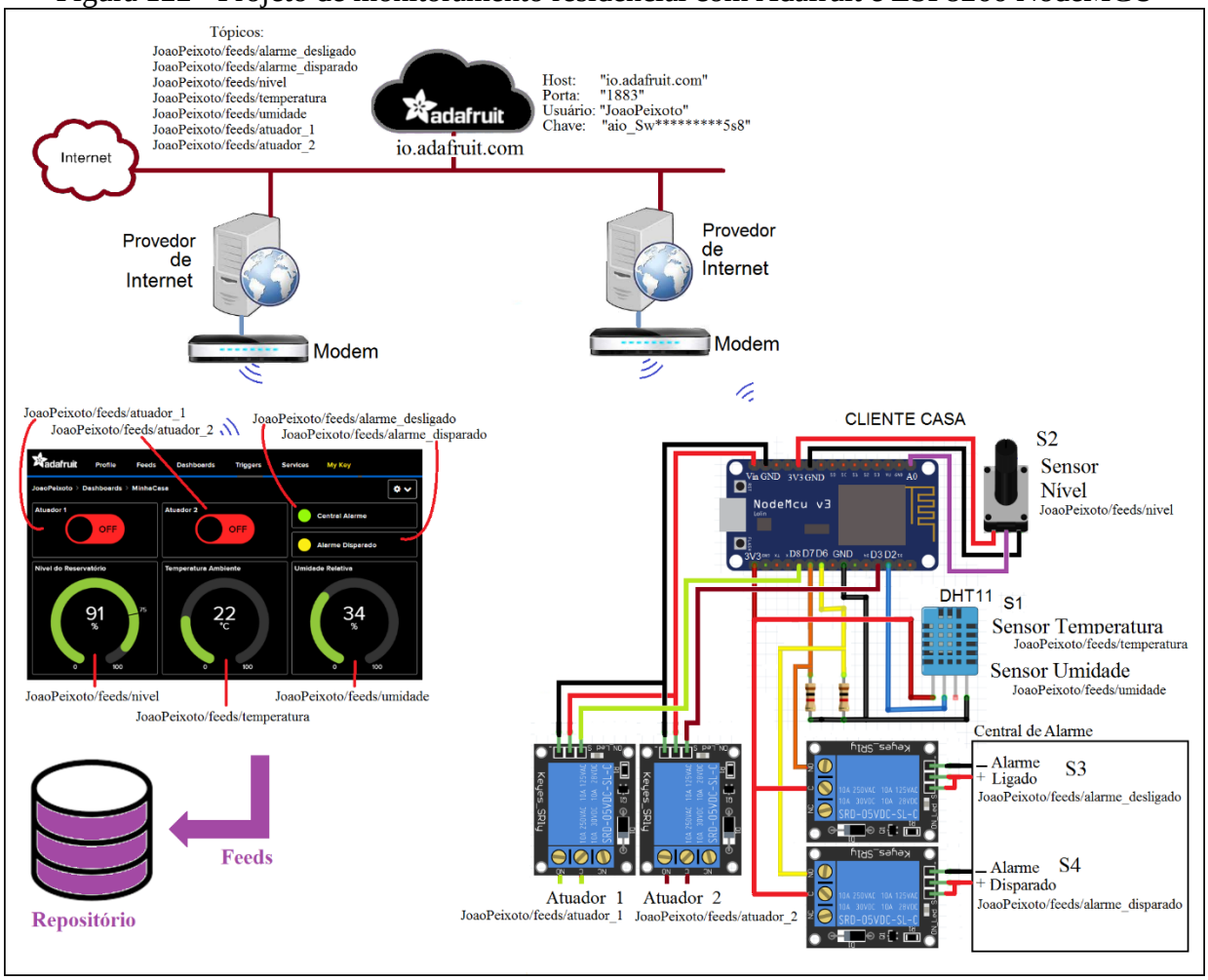
O Projeto 25 apresenta um sistema de monitoramento de dados de uma residência através do portal Adafruit, interagindo com módulo ESP8266 NodeMCU por protocolo MQTT, sendo esses dados: temperatura, umidade, nível do reservatório de água e estado da central de alarme.

PROJETO 25 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E ADAFRUIT

"O projeto consiste em um módulo ESP8266 NodeMCU, conectados a um modem wifi, que acessa um broker Adafruit, por protocolo MQTT, em uma conta criada por um usuário e com uma chave de segurança gerada pelo portal Adafruit. Neste módulo são implementados Feeds para armazenamento de informações de temperatura, umidade e nível de reservatório. É gerado um painel, onde essas variáveis são apresentadas de forma gráfica, somadas às variáveis de estado do alarme (ligado/desligado, normal/disparado) e juntamente com dois botões para acionamento de atuadores no módulo, como comando remoto de dispositivo na residência, tais como: irrigadores, iluminação, alimentadores de animais, entre outros."

A Figura 122 apresenta o diagrama e estrutura do projeto, onde os tópicos MQTT realizam a interação entre o módulo ESP8266 NodeMCU e o portal Adafruit. Além da interação, o portal Adafuit promove o armazenamento das informações para consulta, servindo como um repositório de dados em nuvem.

Figura 122 - Projeto de monitoramento residencial com Adafruit e ESP8266 NodeMCU



Fonte: Autor (2021)

O programa que deve ser inserido no módulo ESP8266 NodeMCU é o seguinte.

```
//************ CLIENTE CASA ********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <Adafruit_MQTT.h> //No gerenciador de bibliotecas "Adafruit MQTT Library by Adafruit"
#include <Adafruit_MQTT_Client.h> //incluso no pacote de biblioteca Adafruit
#define SSID "**********"  //Nome da rede WiFi
#define PASS "**********" //Senha da rede Wifi
#define AIO_SERVER "io.adafruit.com" //Servidor MQTT
#define AIO_SERVERPORT 1883 //Porta do Servidor
#define AIO_USERNAME "JoaoPeixoto" //Nome do usuário na Adafruit.IO
#define AIO_KEY "**********" //Chave de segurança do usuário na Adafruit IO
WiFiClient client; //classe do WiFiClient
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
#include <DHT.h> //Biblioteca .zip em https://drive.google.com/file/d/16brDLNmOVzmQs6sFh-_CAMivQu-cn6kg/view
#define DHTPIN 4 // pino do sensor em D2 - GPIO4 - SDA
#define DHTTYPE DHT11 //tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
//O padrão para subscrição e publicação é: <username>/feeds/<feedname>
Adafruit_MQTT_Publish nivel = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/nivel");
Adafruit_MQTT_Publish temperatura = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/temperatura");
Adafruit_MQTT_Publish umidade = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/umidade");
Adafruit_MQTT_Publish alarme = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/alarme_desligado");
Adafruit_MQTT_Publish disparo = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/alarme_disparado");
Adafruit_MQTT_Subscribe atuador_1 = Adafruit_MQTT_Subscribe(&mqtt,AIO_USERNAME "/feeds/atuador_1");
Adafruit_MQTT_Subscribe atuador_2 = Adafruit_MQTT_Subscribe(&mqtt,AIO_USERNAME "/feeds/atuador_2");
#define Atuador1 15 //LED L8 ligado ao pino D8(GPIO-15)
#define Atuador2 0 //LED L3 ligado ao pino D3(GPIO-0)
#define Alarme 12 //Central de alarme ao pino D6(GPIO-12)
#define Disparo 13 //Central de alarme ao pino D7(GPIO-13)
#define P1 0 //Potenciômetro P1 ligado ao pino A0(GPIO-0)
bool Alar = LOW;
bool Disp = LOW;
int NivelAnterior = 0;
int UmidadeAnterior = 0;
int TemperaturaAnterior = 0;
void setup(){
  Serial.begin(115200);  //baudrate do canal serial
  pinMode(Atuador1, OUTPUT);
  pinMode(Atuador2, OUTPUT);
  pinMode(Alarme, INPUT);
  pinMode(Disparo, INPUT);
  pinMode(P1, INPUT);
  conectarWiFi();
  mqtt.subscribe(&atuador_1);
  mqtt.subscribe(&atuador_2);
  atuador_1.setCallback(callbackAtuador_1);
  atuador_2.setCallback(callbackAtuador_2);
}
void conectarWiFi(){
  if (WiFi.status() == WL_CONNECTED) return; //se já está conectado, retorna
  Serial.println();
  Serial.print("Conectando-se na rede ");
  Serial.println(SSID);
  WiFi.begin(SSID, PASS);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Conectado com sucesso: IP= ");
  Serial.println(WiFi.localIP());
}
```

```
void conectarBroker(){
  int8_t ret;
  if(mqtt.connected()) return;
  Serial.print("Conectando-se ao broker mqtt...");
  while ((ret = mqtt.connect()) != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    Serial.println("Falha ao se conectar. Tentando se reconectar em 5 segundos.");
    mqtt.disconnect();
    delay(5000);
  }
  Serial.println("Conectado ao broker com sucesso.");
  mqtt.subscribe(&atuador_1);
  mqtt.subscribe(&atuador_2);
}

void callbackAtuador_1(char *data, uint16_t len){ //função que recebe mensagens dos tópicos assinados
  String estado = data;
  if(estado.equals("ON")){digitalWrite(Atuador1,HIGH);Serial.println("Ligando Atuador 1");}
  if(estado.equals("OFF")){digitalWrite(Atuador1,LOW);Serial.println("Desligando Atuador 1");}
}

void callbackAtuador_2(char *data, uint16_t len){
  String estado = data;
  if(estado.equals("ON")){digitalWrite(Atuador2,HIGH);Serial.println("Ligando Atuador 2");}
  if(estado.equals("OFF")){digitalWrite(Atuador2,LOW);Serial.println("Desligando Atuador 2");}
}

void publicarDados(){//função para publicar em tópicos
 int valNivel = map(analogRead(P1),0,1023,0,100);
 if(valNivel != NivelAnterior){
    NivelAnterior = valNivel;
    nivel.publish(valNivel);
    Serial.print("Nivel Reservatorio em: ");
    Serial.println(valNivel);
 }

 int valTemperatura = (int) dht.readTemperature(); //Le a temperatura
 if(valTemperatura != TemperaturaAnterior){
    TemperaturaAnterior = valTemperatura;
    temperatura.publish(valTemperatura);
    Serial.print("Temperatura em: ");
    Serial.println(valTemperatura);
 }

 int valUmidade = (int) dht.readHumidity(); //Le a Umidade
 if(valUmidade != UmidadeAnterior){
    UmidadeAnterior = valUmidade;
    umidade.publish(valUmidade);
    Serial.print("Umidade Relativa em: ");
    Serial.println(valUmidade);
 }
 if(digitalRead(Alarme)==HIGH){
   if(Alar==LOW){Alar=HIGH; alarme.publish(1); Serial.println("Alarme ligado"); }
 }
 if(digitalRead(Alarme)==LOW){
   if(Alar==HIGH){ Alar=LOW; alarme.publish(0); Serial.println("Alarme desligado");}
 }
 if(digitalRead(Disparo)==HIGH){
   if(Disp==LOW){ Disp=HIGH; disparo.publish(1); Serial.println("Alarme Disparado");}
 }
 if(digitalRead(Disparo)==LOW){
   if(Disp==HIGH){ Disp=LOW; disparo.publish(0); Serial.println("Alarme Normal");}
 }
}
```

```
void loop(){
  conectarWiFi(); //caso não haja conexão, refaz
  conectarBroker(); //caso não haja conexão, refaz
  mqtt.processPackets(5000); // Função responsável por ler e enviar dados a cada 5s
  publicarDados(); //envia o valor de P1
  if(! mqtt.ping()) mqtt.disconnect(); //executa ping para manter a conexão ativa
}
```

6.4 INTERAÇÃO COM APLICATIVO MQTT DASH

O MQTT Dash é um aplicativo *mobile*, produzido pela empresa de desenvolvimento *Routix Software*, que permite a utilização de objetos *widgets* vinculados a tópicos definidos em um *broker* MQTT. Por sua característica de desenvolvimento e aprendizado, gerado com licença *Creative Commons*, esse aplicativo para versão Android do sistema operacional do *mobile* não possui custo, desde que sua aplicação não tenha fins lucrativos e sirva para o estudo e aprendizado (MOSTOVOY, 2021).

O aplicativo dispõe de componentes *widgets*, tais como:

a) Texto (*Text*): permite exibir ou emitir textos, na forma de mensagens para o usuário;
b) Chaves ou botões (*Switch/button*): permite emitir valores na forma de texto, quanto a condição de acionamento de botão ou chave, podendo também receber valores de texto para que exibam condições de acionamento, uma vez que ambas as ações se valem de ícones para apresentação gráfica e podem atuar como publicadores ou assinantes;
c) Gráficos (*Range/progress*): exibe de forma gráfica um valor recebido em seu tópico de vínculo;
d) Abas de multiescolha (*Multi choice*): emite textos segundo escolha realizada em uma lista editável;
e) Imagens (*Image*): comanda a exibição de imagem, segundo uma URL parametrizada.

Cada componente está vinculado a um tópico, definido em um *broker* de escolha do usuário, sendo que os valores postados nesses tópicos indicam o que cada componente *widget* vai apresentar ou qual será o destino para um valor que um componente *widget* irá produzir. Cada componente pode ser um publicador (*publisher*) ou um assinante (*subscriber*) no protocolo MQTT (SOUZA, 2018).

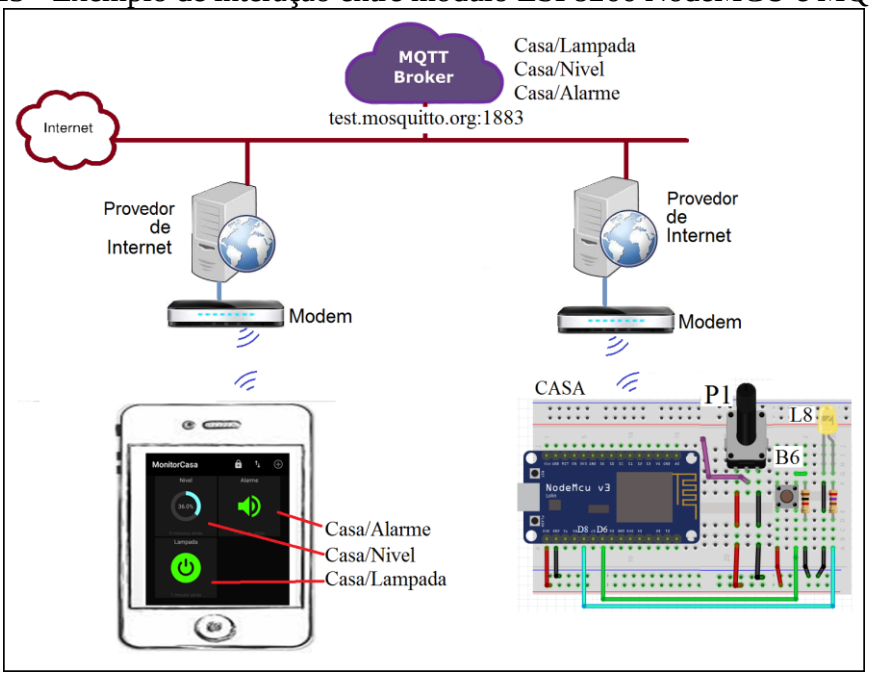
Como exemplo de aplicação, para o estudo da interação entre o módulo ESP8266 NodeMCU e o aplicativo MQTT Dash, é proposto o circuito estrutura, onde o módulo realiza a publicação do valor lido na entrada analógica A0, onde está conectado o potenciômetro P1, em valor convertido a uma escala de 0 a 100, alusivo a um valor de nível de água em um reservatório (tópico: Casa/Nivel). Assim como publica o estado de uma chave conectada ao pino D6, que representa o estado de uma central de alarme, sendo o texto “On” para a condição de alarme ligado e o texto “Off” para o alarme desligado (tópico: Casa/Alarme). E, por fim, o módulo assina o tópico “Casa/Lampada”, que recebe

valores de texto “L” e “D”, indicando para ligar ou desligar o LED conectado ao pino D8, respectivamente.

Do lado do aplicativo MQTT Dash é montado um painel, em que são apresentado de forma gráfica o estado do alarme (tópico: Casa/Alarme) e o valor de nível do reservatório (tópico: Casa/Nivel), sendo que o comando da lâmpada é realizado por um botão que emite valores em texto de “L” e “D”, segundo comando do usuário (tópico: Casa/Lampada).

O *broker* utilizado para esse exemplo é o Mosquitto (*host*: test.mosquitto.org: 1883). A Figura 123 apresenta o exemplo de interação entre o módulo ESP8266 NodeMCU e o aplicativo *mobile* MQTT Dash.

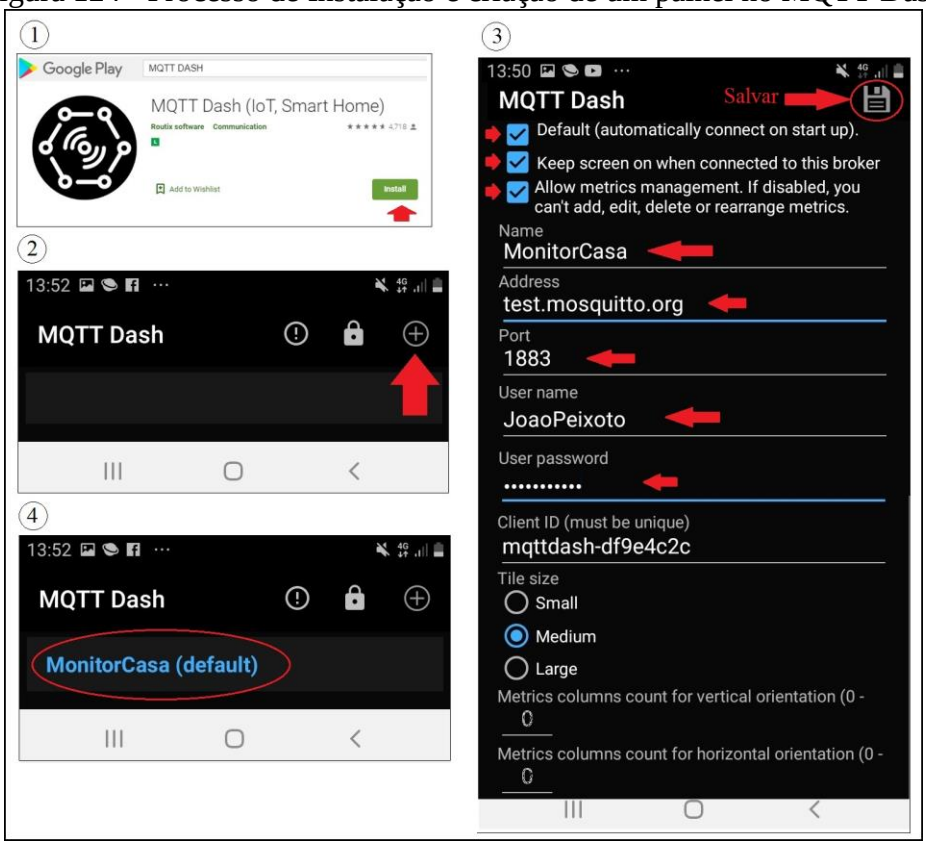
Figura 123 - Exemplo de interação entre módulo ESP8266 NodeMCU e MQTT Dash



Fonte: Autor (2021)

Para instalar o aplicativo MQTT Dash no aparelho mobile e criar o painel de interação, deve-se seguir os passos previstos e apresentados na Figura 124.

Figura 124 - Processo de Instalação e criação de um painel no MQTT Dash

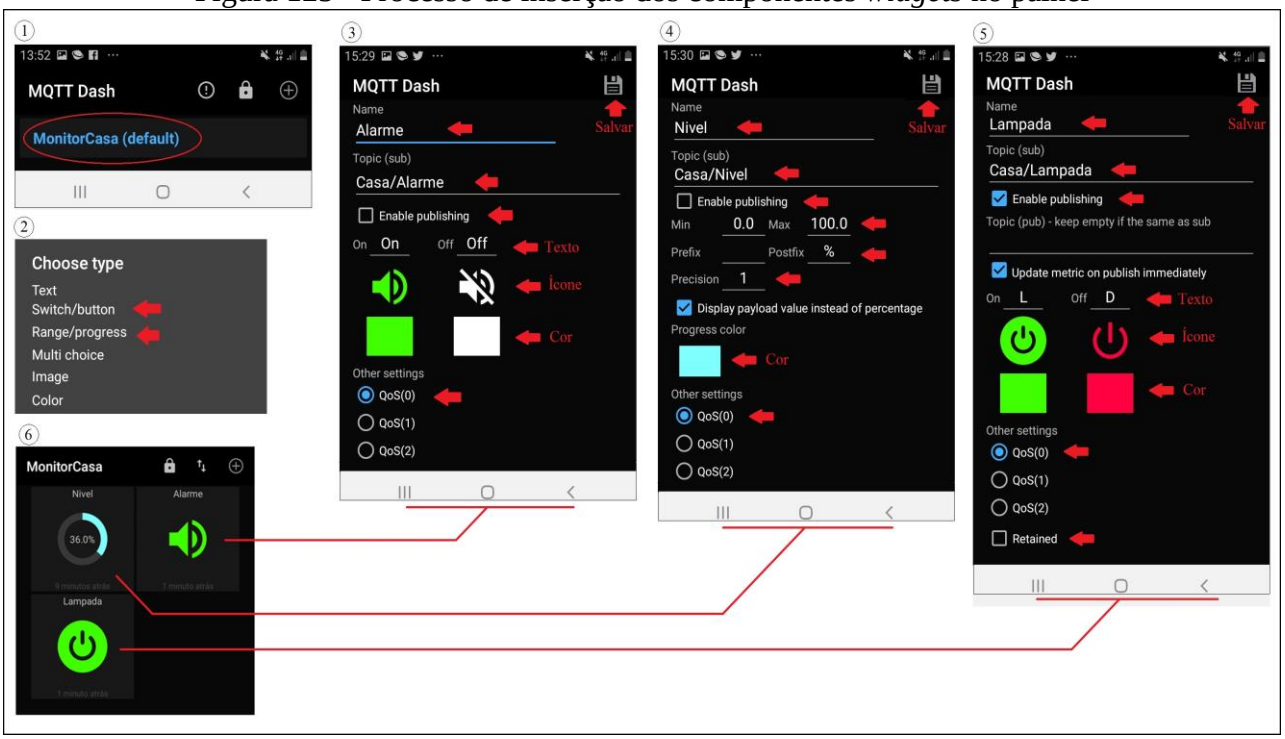


Fonte: Autor (2021)

Inicialmente, deve-se procurar na "Play Store" do aparelho *mobile* pelo aplicativo MQTT Dash, item ①, procedendo sua instalação. Ao abrir o aplicativo MQTT Dash, item ②, deve-se clicar sobre o sinal de "+", para inserir um novo painel. Abrirá uma caixa de parametrização, item ③, onde deve-se indicar o nome do painel (*Name*), o endereço de *host* do *broker* escolhido (*Address*), a porta de conexão ao *host* (*Port*), o nome de usuário (*User name*), indicar uma senha (*User password*) e clicar no ícone superior direito para salvar os parâmetros. A tela principal do aplicativo MQTT Dash apresentará o painel criado, item ④.

Para a inserção dos objetos *widgets* no painel criado, deve-se proceder como o apresentado na Figura 125, que indica o processo de inserção dos componentes *widgets* no painel.

Figura 125 - Processo de inserção dos componentes *widgets* no painel



Fonte: Autor (2021)

Na tela principal do aplicativo MQTT Dash, deve-se clicar sobre o painel criado item ①. Abrirá uma caixa de seleção, com as opções de *widgets*, item ②.

Para o componente que irá indicar o estado do alarme, selecione "*Switch/button*", aparecerá a caixa de parametrização do objeto, item ③, onde é necessário inserir o nome do componente (*Name*) e o tópico de vínculo para assinatura (*Topic(sub)*), desabilitar a caixa "*Enable publishing*" a fim de ter somente a função de assinante e definir o texto que será recebido, o ícone que deverá ser exibido, a cor desse ícone e a qualidade de serviço mqtt (opção "*Other settings"*).

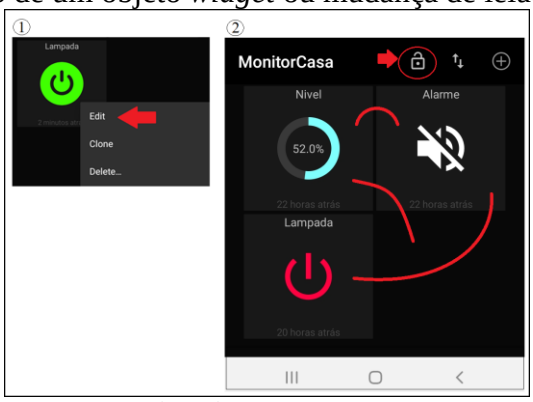
Para o componente que irá indicar o nível do reservatório de água, selecione "*Range/progress*", surgirá a caixa de parametrização do objeto, item ④, onde é necessário inserir o nome do componente (*Name*) e o tópico de vínculo para assinatura (*Topic(sub)*), desabilitar a caixa "*Enable publishing*", a fim de ter somente a função de assinante, indicar os valores mínimos e máximos para geração do gráfico e definir a unidade de medida que se deseja exibir, a precisão de casas decimais para exibição, a cor do gráfico e a qualidade de serviço mqtt (*Other settings*).

Para o componente que irá comandar a lâmpada, selecione "*Switch/button*", aparecerá a caixa de parametrização do objeto, item ⑤, onde é necessário inserir o nome do componente (*Name*) e o tópico de vínculo para assinatura (*Topic(sub)*), habilitar a caixa "*Enable publishing*", a fim de que opere como publicador, e definir o texto que será enviado segundo comando do usuário, o ícone que deverá ser exibido segundo comando do usuário, a cor desses ícone, a qualidade de serviço mqtt (*Other settings*) e se essa mensagem será retentiva no protocolo mqtt.

Por fim, o painel está pronto para uso, item ⑥.

Os componentes foram adicionados ao painel segundo sua ordem de criação. Mas essa ordem pode ser alterada. Basta clicar sobre o "cadeado", habilitando a movimentação dos componentes na tela do painel. Ao clicar novamente sobre o "cadeado", a tela volta a ficar travada. Assim como os componentes podem ser novamente editados, pela necessidade de alteração de algum parâmetro, bastando clicar sobre o componente por aproximadamente 3 segundos, onde será aberta uma aba para selecionar "*Edit*". Essas duas funções estão apresentadas na Figura 126, sendo o item ① para edição de um componente *widget* e item ② para alteração do leiaute.

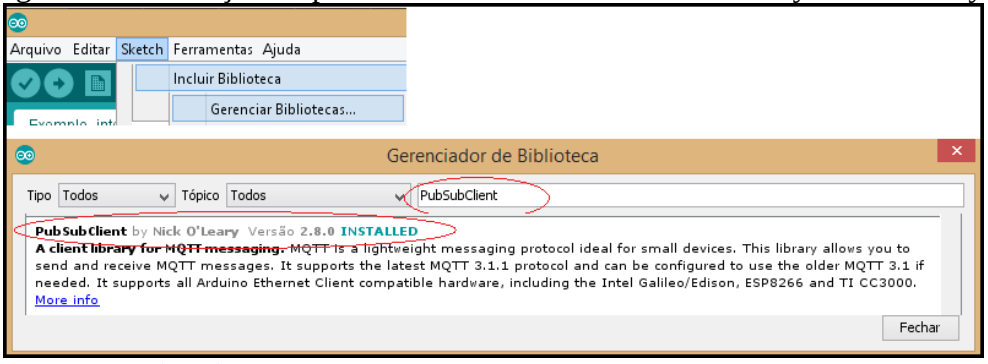
Figura 126 - Edição de um objeto *widget* ou mudança de leiaute no MQTT Dash



Fonte: Autor (2021)

Quanto ao programa a ser inserido no módulo ESP8266 NodeMCU, para essa aplicação exemplo, faz-se necessário, como passo inicial, a instalação do pacote de biblioteca "*PubSubClient by Nick OLeary*" na IDE Arduino. Isso pode ser realizado clicando na aba "*Sketch*", "Incluir Biblioteca" e "Gerenciar Biblioteca...", na IDE Arduino, procurando pela palavra "*PubSubClient*", encontrando e instalando a biblioteca indicada. A Figura 127 apresenta esse processo de inserção do pacote de biblioteca "*PubSubClient by Nick OLeary*".

Figura 127 - Inserção do pacote de biblioteca "PubSubClient by Nick OLeary"



Fonte: Autor (2021)

O programa para o módulo neste exemplo de aplicação é o seguinte.

```
//************ CLIENTE CASA ********************
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //No gerenciador de bibliotecas "PubSubClient by Nick O'Leary"
//configuração WiFi
const char* SSID = "**********"; //NOME da Rede LAN
const char* PASSWORD = "**********";    //SENHA da rede LAN
WiFiClient wifiClient;
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "Casa" //Nome do Cliente, que deve ser único no Broker.
#define TOPIC_PUB_Nivel "Casa/Nivel" //Topico para publicação do nivel, 0 a 100%
#define TOPIC_PUB_Alarme "Casa/Alarme" //Topico para publicação do estado Alarme (On ou Off)
#define TOPIC_SUB_Lampada "Casa/Lampada" //Topico para Assinatura do comando para Lampada (L ou D)
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient

#define P1 0 //Potenciometro no pino A0(GPIO-0)
#define Alarme 12 //Chave no pino D6(GPIO-12)
#define Lampada 15 //LED no pino D8(GPIO-15)
bool estadoAlarme = LOW;
int anteriorNivel=0;
long tempoAnterior = 0;

void setup(){
  Serial.begin(115200);
  pinMode(P1,INPUT);
  pinMode(Alarme,INPUT);
  pinMode(Lampada,OUTPUT);
  conectaWiFi();//conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT);//configura o cliente
  MQTT.setCallback(mqtt_callback);//ativa a função para receber postagens de tópicos assinados
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
    while(!MQTT.connected()){
        Serial.print("Conectando ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if(MQTT.connect(ID_MQTT)) {
            Serial.println("Conectado ao Broker com sucesso!");
            MQTT.subscribe(TOPIC_SUB_Lampada);//assina o tópico "Casa/Lampada"
        }
        else {
            Serial.println("Falha ao conectar, nova tentativa em 10s");
            delay(10000);
        }
    }
}
void publicaDados(){
  if(millis() - tempoAnterior > 5000){//leitura a cada 5s
    tempoAnterior = millis();
    if(digitalRead(Alarme)==HIGH){
      if(estadoAlarme==LOW){
```

```
      estadoAlarme=HIGH;
      Serial.println("Alarme Ligado");
      MQTT.publish(TOPIC_PUB_Alarme,"On");//publica no tópico "Casa/Alarme"
     }
    }
    if(digitalRead(Alarme)==LOW){
     if(estadoAlarme==HIGH){
      estadoAlarme=LOW;
      Serial.println("Alarme Desligado");
      MQTT.publish(TOPIC_PUB_Alarme,"Off");//publica no tópico "Casa/Alarme"
     }
    }
    int valNivel = map(analogRead(P1),0,1023,0,100);
    char msgNivel[10];
    sprintf(msgNivel,"%.d",valNivel);
    if(valNivel != anteriorNivel){
     anteriorNivel = valNivel;
     MQTT.publish(TOPIC_PUB_Nivel,msgNivel);//publica no tópico "Casa/Nivel"
    }
  }
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
     char c = (char)payload[i];
     mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_Lampada)){//lê o topico "Cliente/Lampada"
    //toma ação dependendo da string recebida:
    if(mensagem.equals("L"))digitalWrite(Lampada,HIGH);
    if(mensagem.equals("D"))digitalWrite(Lampada,LOW);
  }
}
void loop() {
  if(!MQTT.connected())conectaMQTT();//caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  publicaDados();
  MQTT.loop();//keep alive do protocolo, necessário para ler os Callback que chegaram
}
```

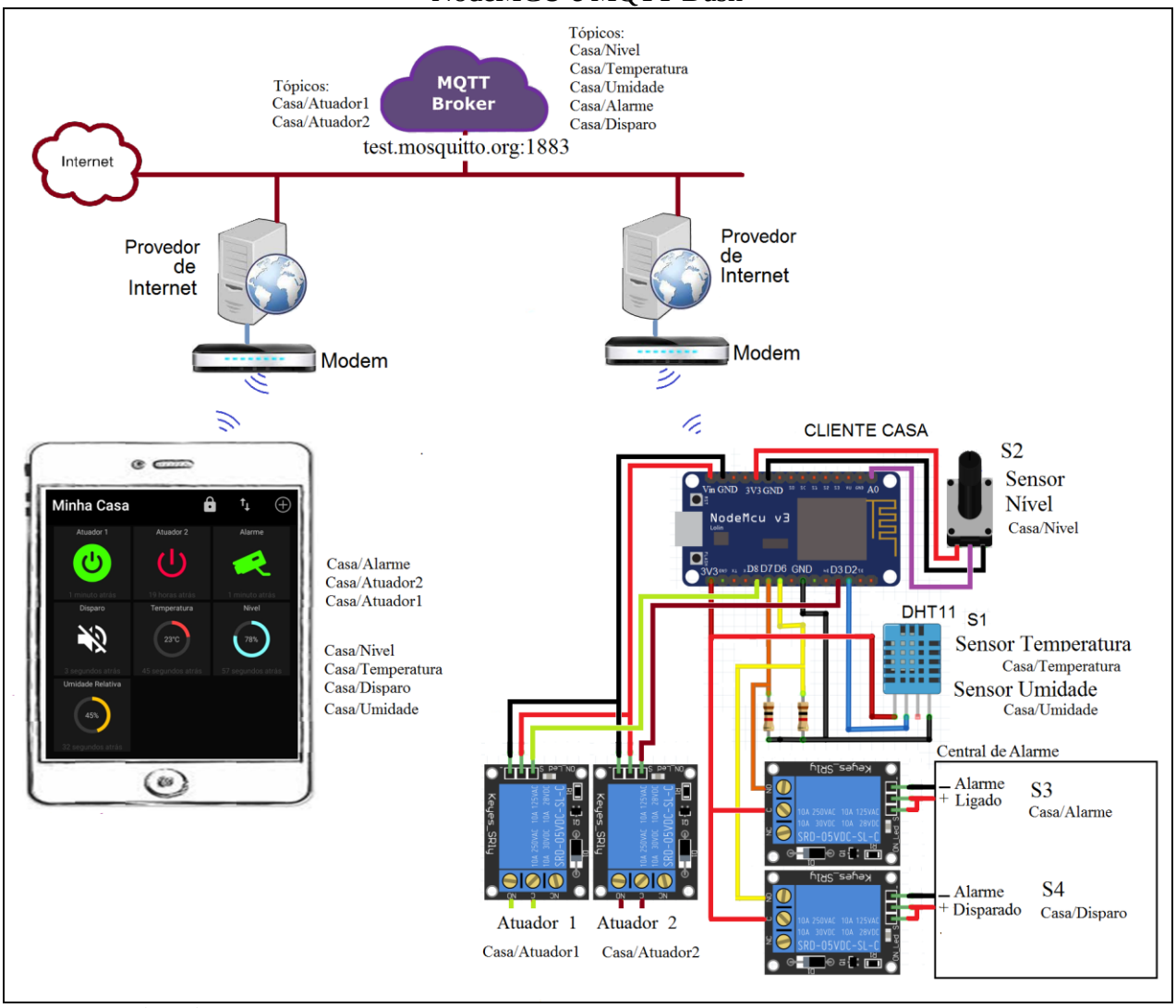
O Projeto 26 apresenta uma aplicação de monitoramento residencial com um módulo ESP8266 NodeMCU, sendo apresentado o estado de variáveis na residência em um painel gráfico de um aplicativo *mobile* do MQTT Dash.

PROJETO 26 – MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E MQTT DASH

"O projeto consiste em um módulo ESP8266 NodeMCU, conectado a um modem wifi, que acessa um broker Mosquitto, por protocolo MQTT, e interage com o aplicativo mobile MQTT Dash, que exibe em um painel gráfico as informações de temperatura, umidade e nível de reservatório, somadas às variáveis de estado do alarme (ligado/desligado, normal/disparado) e a dois botões para acionamento de atuadores no módulo, como comando remotos de dispositivo na residência, tais como: irrigadores, iluminação, alimentadores de animais, entre outros."

A Figura 128 apresenta o diagrama e estrutura do projeto, onde os tópicos MQTT realizam a interação entre o móduolo ESP8266 NodeMCU e o aplicativo mobile MQTT Dash.

Figura 128 – Estrutura e diagrama do projeto monitoramento de residência com ESP8266 NodeMCU e MQTT Dash



Fonte: Autor (2021)

O código para inserção no módulo ESP8266NodeMCU a fim de executar o projeto monitoramento de residência com ESP8266 NodeMCU e MQTT Dash é o seguinte.

```
//****PROJETO 26 - MONITORAMENTO DE RESIDÊNCIA COM ESP8266 NODEMCU E MQTT DASH****
//placa "NodeMCU 1.0(ESP-12E Module)"
#include <ESP8266WiFi.h> //No gerenciador de bibliotecas "ESP8266WIFI by Ivan Grokhotkov"
#include <PubSubClient.h> //No gerenciador de bibliotecas "PubSubClient by Nick O'Leary"
//configuração WiFi
const char* SSID = "**********"; //NOME da Rede LAN
const char* PASSWORD = "**********";    //SENHA da rede LAN
WiFiClient wifiClient;
//Configuração do Cliente MQTT
const char* BROKER_MQTT = "test.mosquitto.org"; //URL do broker MQTT que se deseja utilizar
int BROKER_PORT = 1883; // Porta do Broker MQTT
#define ID_MQTT  "Casa" //Nome do Cliente, que deve ser único no Broker.
#define TOPIC_PUB_Nivel "Casa/Nivel" //Topico para publicação do nivel, 0 a 100%
#define TOPIC_PUB_Temperatura "Casa/Temperatura" //Topico para publicação da Temperatura, 0 a 100Graus
#define TOPIC_PUB_Umidade "Casa/Umidade" //Topico para publicação da Umidade, 0 a 100%UR
#define TOPIC_PUB_Alarme "Casa/Alarme" //Topico para publicação do estado Alarme (On ou Off)
#define TOPIC_PUB_Disparo "Casa/Disparo" //Topico para publicação do Disparo Alarme (On ou Off)
#define TOPIC_SUB_Atuador1 "Casa/Atuador1" //Topico para Assinatura do comando para Atuador 1 (On ou Off)
#define TOPIC_SUB_Atuador2 "Casa/Atuador2" //Topico para Assinatura do comando para Atuador 2 (On ou Off)
PubSubClient MQTT(wifiClient); // Instancia o Cliente MQTT passando o objeto espClient
#include <DHT.h> //Biblioteca .zip disponível em https://drive.google.com/file/d/16brDLNmOVzmQs6sFh-
_CAMivQu-cn6kg/view
#define DHTPIN 4 // pino do sensor em D2 - GPIO4 - SDA
#define DHTTYPE DHT11 //tipo de sensor
DHT dht(DHTPIN, DHTTYPE);
#define P1 0 //Potenciometro no pino A0(GPIO-0)
#define Alarme 12 //Central Alarme Ligado, no pino D6(GPIO-12)
#define Disparo 13 //Central Alarme Disparo, no pino D7(GPIO-13)
#define Atuador1 0 //Chave no pino D3(GPIO-0)
#define Atuador2 15 //LED no pino D8(GPIO-15)
bool estadoAlarme = LOW;
bool estadoDisparo = LOW;
int anteriorNivel=0;
int anteriorTemperatura=0;
int anteriorUmidade=0;
long tempoAnterior = 0;

void setup(){
  Serial.begin(115200);
  pinMode(P1,INPUT);
  pinMode(Alarme,INPUT);
  pinMode(Disparo,INPUT);
  pinMode(Atuador1,OUTPUT);
  pinMode(Atuador2,OUTPUT);
  conectaWiFi();//conecta ao wifi
  MQTT.setServer(BROKER_MQTT, BROKER_PORT);//configura o cliente
  MQTT.setCallback(mqtt_callback);//ativa a função para receber postagens de tópicos assinados
}
void conectaWiFi(){
  Serial.println("");
  Serial.print("Conectando a Rede: ");
  WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI
  while (WiFi.status() != WL_CONNECTED){delay(100);Serial.print(".");}
  Serial.println();
  Serial.print("Conectado com sucesso, na rede: ");
  Serial.print(SSID);
  Serial.print("  IP obtido: ");
  Serial.println(WiFi.localIP());
}
void conectaMQTT(){
   while(!MQTT.connected()){
```

```
      Serial.print("Conectando ao Broker MQTT: ");
      Serial.println(BROKER_MQTT);
      if(MQTT.connect(ID_MQTT)) {
         Serial.println("Conectado ao Broker com sucesso!");
         MQTT.subscribe(TOPIC_SUB_Atuador1);//assina o tópico "Casa/Atuador1"
         MQTT.subscribe(TOPIC_SUB_Atuador2);//assina o tópico "Casa/Atuador2"
      }
      else {
         Serial.println("Falha ao conectar, nova tentativa em 10s");
         delay(10000);
      }
   }
}
void publicaDados(){
 if(millis() - tempoAnterior > 5000){//leitura a cada 5s
   tempoAnterior = millis();
   if(digitalRead(Alarme)==HIGH){
    if(estadoAlarme==LOW){
     estadoAlarme=HIGH;
     Serial.println("Alarme Ligado");
     MQTT.publish(TOPIC_PUB_Alarme,"On");//publica no tópico "Casa/Alarme"
    }
   }
   if(digitalRead(Alarme)==LOW){
    if(estadoAlarme==HIGH){
     estadoAlarme=LOW;
     Serial.println("Alarme Desligado");
     MQTT.publish(TOPIC_PUB_Alarme,"Off");//publica no tópico "Casa/Alarme"
    }
   }
   if(digitalRead(Disparo)==HIGH){
    if(estadoDisparo==LOW){
     estadoDisparo=HIGH;
     Serial.println("Alarme Disparado");
     MQTT.publish(TOPIC_PUB_Disparo,"On");//publica no tópico "Casa/Disparo"
    }
   }
   if(digitalRead(Disparo)==LOW){
    if(estadoDisparo==HIGH){
     estadoDisparo=LOW;
     Serial.println("Alarme Normal");
     MQTT.publish(TOPIC_PUB_Disparo,"Off");//publica no tópico "Casa/Disparo"
    }
   }
   int valNivel = map(analogRead(P1),0,1023,0,100);
   char msgNivel[10];
   sprintf(msgNivel,"%.d",valNivel);
   if(valNivel != anteriorNivel){
    anteriorNivel = valNivel;
    MQTT.publish(TOPIC_PUB_Nivel,msgNivel);//publica no tópico "Casa/Nivel"
   }
  int valTemperatura = (int) dht.readTemperature(); //Le a temperatura
  char msgTemperatura[10];
   sprintf(msgTemperatura,"%.d",valTemperatura);
  if(valTemperatura != anteriorTemperatura){
    anteriorTemperatura = valTemperatura;
    MQTT.publish(TOPIC_PUB_Temperatura,msgTemperatura);//publica no tópico "Casa/Temperatura"
    Serial.print("Temperatura em: ");
    Serial.println(valTemperatura);
  }
  int valUmidade = (int) dht.readHumidity(); //Le a temperatura
  char msgUmidade[10];
   sprintf(msgUmidade,"%.d",valUmidade);
  if(valUmidade != anteriorUmidade){
    anteriorUmidade = valUmidade;
```

```
      MQTT.publish(TOPIC_PUB_Umidade,msgUmidade);//publica no tópico "Casa/Umidade"
      Serial.print("Umidade em: ");
      Serial.println(valUmidade);
    }
}
}
//função que recebe mensagens dos tópicos assinados
void mqtt_callback(char* topic, byte* payload, unsigned int length){
  String topico = topic;
  String mensagem;//obtem a string do payload recebido
  for(int i = 0; i < length; i++){
     char c = (char)payload[i];
     mensagem += c;
  }
  if(topico.equals(TOPIC_SUB_Atuador1)){//lê o topico "Casa/Atuador1" e toma ação dependendo da string recebida
    if(mensagem.equals("On"))digitalWrite(Atuador1,HIGH);
    if(mensagem.equals("Off"))digitalWrite(Atuador1,LOW);
  }
  if(topico.equals(TOPIC_SUB_Atuador2)){//lê o topico "Casa/Atuador2" e toma ação dependendo da string recebida.
    if(mensagem.equals("On"))digitalWrite(Atuador2,HIGH);
    if(mensagem.equals("Off"))digitalWrite(Atuador2,LOW);
  }
}
void loop() {
  if(!MQTT.connected())conectaMQTT();//caso não haja conexão, refaz
  if (WiFi.status() != WL_CONNECTED) conectaWiFi(); //caso não haja conexão, refaz
  publicaDados();
  MQTT.loop();//keep alive do protocolo, necessário para ler os Callback que chegaram
}
```

REFERÊNCIAS

ADAFRUIT. **The internet of things for everyone**. [2021]. Disponível em: https://io.adafruit.com/. Acesso em: 28 maio 2021.

ALBUQUERQUE, Y. ESP8266 – Cadastro RFID (MFRC522) com webserver. **Smartkits Blog**, [2020]. Disponível em: https://blog.smartkits.com.br/esp8266-cadastro-rfid-mfrc522-com-webserver/. Acesso em: 23 abr. 2021.

ARCHANJO, C. Revit: passo-a-passo de um projeto de visualização 3D. **Fluxo Blog de Engenharia**, 15 de julho de 2018. Disponível em: https://fluxoconsultoria.poli.ufrj.br/blog/revit-passo-a-passo-de-um-projeto-de-visualizacao-3d/. Acesso em: 14 maio 2021.

ARDUINING. **Physical Computing Mini-Projects**. c2021. Disponível em: https://arduining.com/. Acesso em: 9 mar. 2021.

ARDUINO. **Documentação de Referência da Linguagem Arduino**. c2021. Disponível em: https://www.arduino.cc/reference/pt/. Acesso em: 12 mar. 2021.

ARDUINO e Cia. **Como usar o módulo I2C com Arduino e display LCD 16x2**. 2014. Disponível em: https://www.arduinoecia.com.br/modulo-i2c-display-16x2-arduino/. Acesso em: 9 abr. 2021.

ARDUINO e Cia. **Como funciona o conversor de nível lógico 3,3 – 5V**. 2015. Disponível em: https://www.arduinoecia.com.br/conversor-de-nivel-logico-33-5v-arduino/. Acesso em: 13 abr. 2021.

ASCII. In: Wikipédia: a enciclopédia livre. Disponível em: https://pt.wikipedia.org/wiki/ASCII. Acesso em: 12 jun. 2021.

BARROS, M. MQTT – Protocolos para IoT. **Embarcados**, 15 de junho de 2015. Disponível em: https://www.embarcados.com.br/mqtt-protocolos-para-iot/. Acesso em: 14 maio 2021.

BAUERMEISTER, G. Biblioteca Arduino: aprenda como criar a sua - Parte 1. **Filipeflop**, 10 de abril de 2018. Disponível em: https://www.filipeflop.com/blog/desenvolvendo-uma-biblioteca-arduino/. Acesso em: 21 abr. 2021.

BERNARDO, J. O Efeito Bouncing e as técnicas para Debouncing. **EletronWord** , 21 de junho de 2016. Disponível em: https://eletronworld.com.br/eletronica/efeito-bounce/. Acesso em: 19 mar. 2021.

BERTOLETI, P. Controle e Monitoramento IoT com NodeMCU e MQTT. **Filipeflop,** 30 de maio de 2016. Disponível em: https://www.filipeflop.com/blog/controle-monitoramento-iot-nodemcu-e-mqtt/. Acesso em: 14 maio 2021.

BLYNK. **Intro**. [2021]. Disponível em: http://docs.blynk.cc/. Acesso em: 22 maio 2021.

BRAGA, N. C. **Relés: Circuitos e Aplicações**. São Paulo: Instituto Newton C. Braga, 2012. Disponível em: https://www.newtoncbraga.com.br/arquivos/reles_previa.pdf. Acesso em: 11 abr. 2021.

CADERNO DE LABORATÓRIO. **Entrada analógica no NodeMCU**. [2021]. Disponível em: https://cadernodelaboratorio.com.br/entrada-analogica-no-nodemcu/. Acesso em: 23 mar. 2021.

CITISYSTEMS. **Curso prático de sensores industriais gratuito**. 2021. Disponível em: https://treinamentos.citisystems.com.br/produto/sensores-industriais-gratuito/. Acesso em: 12 abr. 2021.

CODE LEARN. **Automação em ESP8266 e Telegram: tutorial completo**. Vídeo postado no canal

do youtube Code Learn, [2020]. Disponível em: https://www.youtube.com/watch?v=R9QCM1x375A. Acesso em: 6 fev. 2021.

CURTO CIRCUITO. Estação Meteorológica com NodeMCU. **Blog da Curto**, 24 de outubro de 2018. Disponível em: https://www.curtocircuito.com.br/blog/Categoria IoT/estacao-meteorologica-com-nodemcu. Acesso em: 22 maio 2021.

GUIMARÃES, F. **Como Programar o ESP8266 para ser um WEB SERVER**. Vídeo postado no canal do youtube Brincando com Ideias, 2018. Disponível em: https://www.youtube.com/watch?v=kCjWYQEMvwc. Acesso em: 8 maio 2021.

GUSE, R. Estação Meteorológica com ESP8266 NodeMCU. **Filipeflop**, 20 de novembro de 2020. Disponível em: https://www.filipeflop.com/blog/estacao-meteorologica-com-esp8266-nodemcu/. Acesso em: 12 mar. 2021.

HIVEMQ TEAM. Quality of Service 0,1 & 2 - MQTT Essentials: Part 6. **HiveMQ**, 16 de fevereiro de 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. Acesso em: 14 maio 2021.

LAST MINUTE ENGINEERS. **Interface L298N DC Motor Dirver Module with Arduino**. c2021. Disponível em: https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/. Acesso em: 22 abr. 2021.

LAVRATTI, F. Infográfico de alguns protocolos de rede para IoT. **Embarcados**, 15 de junho de 2016. Disponível em: https://www.embarcados.com.br/infografico-protocolos-de-rede-para-iot/. Acesso em: 28 abr. 2021.

LOCATELLI, C. INTRODUÇÃO AO MQTT. **Blog da Curto**, 14 de janeiro 2020a. Disponível em: https://www.curtocircuito.com.br/blog/introducao-ao-mqtt. Acesso em: 14 maio 2021.

LOCATELLI, C. Desenvolvimento de Dashboard MQTT com Adafruit.IO. **Blog da Curto**, 6 de agosto de 2020b. Disponível em: https://www.curtocircuito.com.br/blog/desenvolvimento-de-dashboard-mqtt-com-adafruitio. Acesso em: 28 maio 2021.

LUA. **A linguagem de programação Lua**. Rio de Janeiro: PUCRIO, 2021. Disponível em: http://www.lua.org/. Acesso em: 9 mar. 2021.

MADEIRA, D. Primeiros passos com o ESP8266 NodeMCU: utilizando sinais PWM. **Portal Vida de Silício**, [2019]. Disponível em: https://portal.vidadesilicio.com.br/primeiros-passos-com-o-esp8266-nodemcu-utilizando-sinais-pwm/#:~:text=Simulando saídas analógicas com PWM,-Apesar de possuir&text=Esta porcentagem é chamada de,no caso do ESP8266 NodeMCU. Acesso em: 19 mar. 2021.

MADEIRA, D. Protocolo I2C - Comunicação entre Arduinos. **Portal Vida de Silício**, [2018]. Disponível em: https://portal.vidadesilicio.com.br/i2c-comunicacao-entre-arduinos/#:~:text=4 Considerações finais-,O protocolo de comunicação I2C,e os demais serão escravos. Acesso em: 9 abr. 2021.

MARQUES, J. Comunicação e Controle entre Dois ESPs8266 Ponto a Ponto - Peer-to-Peer - Com Roteador. **FVM Learning**, 9 de julho de 2019. Disponível em: http://www.fvml.com.br/2019/07/comunicacao-e-controle-entre-dois.html. Acesso em: 30 abr. 2021.

MATTEDE, H. O que é Servo motor e como funciona? **Mundo da Elétrica,** c2021. Disponível em: https://www.mundodaeletrica.com.br/o-que-e-servo-motor-e-como-funciona/. Acesso em: 21 abr. 2021.

MOSTOVOY, V. V. Routix. **Routix** NetCom, c2021. Disponível em: http://www.routix.net/. Acesso em: 30 maio 2021.

MURTA, G. NodeMCU – ESP12: Guia completo – Introdução (Parte 1). **Blog Eletrogate**, 1 de março de 2018a. Disponível em: https://blog.eletrogate.com/nodemcu-esp12-introducao-1/#:~:text=Mas se for usar a,nunca ultrapasse os 500 mA.&text=A alimentação dessa placa pode,alimentação regulada de 5%2C0V. Acesso em: 23 mar. 2021.

MURTA, J. G. A. M. Guia completo do display LCD - Arduino. **Blog Eletrogate**, 13 de abril de 2018b. Disponível em: https://blog.eletrogate.com/guia-completo-do-display-lcd-arduino/. Acesso em: 23 mar. 2021.

NERI, R.; LOMBA, M.; BULHÕES, G. **MQTT**. Rio de Janeiro: Escola Politecnica da UFRJ, 2019. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/. Acesso em: 14 maio 2021.

OLIVEIRA, S. de. **Internet das coisas com ESP8266, Arduino e Raspberry PI**. São Paulo: [s.n.], 2017.

PAVEL. How to display any sensor data in Blynk App. **Blynk help center**, c2021. Disponível em: http://help.blynk.cc/en/articles/512056-how-to-display-any-sensor-data-in-blynk-app. Acesso em: 22 maio 2021.

ROCHA, N. Como controlar LED através de página de HTML com NodeMCU. **Arduino Portugal**, 26 de abril de 2017. Disponível em: https://www.arduinoportugal.pt/controlo-leds-atraves-pagina-html-nodemcu-marco-2017/. Acesso em: 9 maio 2021.

SACCO, F. Comunicação SPI - Parte 1. **Embarcados**, 5 de maio de 2014. Disponível em: https://www.embarcados.com.br/spi-parte-1/. Acesso em: 24 abr. 2021.

SERRANO, T. M. Introdução ao Blynk App. **Embarcados**, 8 de maio de 2018. Disponível em: https://www.embarcados.com.br/introducao-ao-blynk-app/. Acesso em: 22 maio 2021.

SILVA, L. F. C. IoT: Contexto geral, presente e perspectiva futura – Parte 1. **Embarcados**, 1 de fevereiro de 2017a. Disponível em: https://www.embarcados.com.br/iot-contexto-geral-perspectiva-parte-1/. Acesso em: 28 abr. 2021.

SILVA, L. F. C. IoT: Contexto geral, presente e perspectiva futura – Parte 2. **Embarcados**, 6 de fevereiro de 2017a. Disponível em: https://www.embarcados.com.br/iot-contexto-geral-perspectiva-parte-2/. Acesso em: 28 abr. 2021.

SOUZA, F. Use o MQTT DASH para controlar uma lâmpada remotamente. **Embarcados**, 19 de fevereiro de 2018. Disponível em: https://www.embarcados.com.br/mqtt-dash/. Acesso em: 20 maio 2021.

ADVANCED MONOLITHIC SYSTEMS Inc. **AMS1117**. [2021]. Disponível em: http://www.advanced-monolithic.com/pdf/ds1117.pdf. Acesso em: 9 mar. 2021.

TELEGRAM. **Telegram APIs**. [2021]. Disponível em: https://core.telegram.org/api. Acesso em: 6 abr. 2021.

THOMSEN, A. Como conectar o Sensor Ultrassônico HC-SR04 ao Arduino. **Filipeflop**, 23 de julho de 2011. Disponível em: https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/. Acesso em 12 abr. 2021.

THOMSEN, A. Controle de Acesso usando Leitor RFID com Arduino. **Filipeflop**, 23 de abril de 2014. Disponível em: https://www.filipeflop.com/blog/controle-acesso-leitor-rfid-arduino/. Acesso em: 23 abr. 2021.

VIANA, C. C. Monitor de Temperatura e Umidade com NODEMCU ESP8266 como Servidor Web. **Blog da Robótica**, 11 de agosto de 2020. Disponível em: http://www.blogdarobotica.com/2020/08/11/monitor-de-temperatura-e-umidade-com-nodemcu-esp8266-como-servidor-web/. Acesso em: 21 abr. 2021.

SOBRE O AUTOR

Prof. Dr. João Alvarez Peixoto

Possui graduação em Engenharia Elétrica pela Universidade do Vale do Rio dos Sinos (2005). MBA em Gestão estratégica de instituições de educação profissional e tecnológica pela Faculdade de Tecnologia SENAI Florianópolis. Mestre em Controle e Automação pela Universidade Federal do Rio Grande do Sul (2012). Doutor em Controle e Automação pela Universidade Federal do Rio Grande do Sul. Atuou no Serviço Nacional de Aprendizagem Industrial SENAI com instrutor de nível técnico, como supervisor de educação e tecnologia. Atualmente atua como Professor Adjunto na Universidade Estadual do Rio Grande do Sul - UERGS. Tem experiência na área de Engenharia Elétrica, com ênfase em Automação Eletrônica de Processos Elétricos e Industriais, sendo a especialidade em sistemas de manufatura automatizadas e virtuais.

E-mail: joao-peixoto@uergs.edu.br - joao.alvarez.peixoto@gmail.com
Skype: joao.alvarez.peixoto
Currículo lattes: http://lattes.cnpq.br/3242194031865969
Currículo ORCID: http://orcid.org/0000-0003-1218-0363
Web Site: https://joao-peixoto.webnode.com.br