UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

WALTER BARBOSA SOARES JÚNIOR

Desenvolvimento de um Aplicativo para Smartphone utilizando o Node-RED para o Protocolo MQTT Aplicado numa Bancada Eletropneumática

Uberlândia
2023

WALTER BARBOSA SOARES JÚNIOR

Desenvolvimento de um Aplicativo em Smartphone utilizando o Node-RED para o Protocolo MQTT Aplicado numa Bancada Eletropneumática

Trabalho de Conclusão de apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica

Orientador: Prof. Dr. José Jean Paul Zanlucchi de Souza Tavares

Uberlândia

2023

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

S676 Soares Junior, Walter Barbosa, 1991-
2023 Desenvolvimento de um Aplicativo em Smartphone utilizando o Node-RED para o Protocolo MQTT Aplicado numa Bancada Eletropneumática [recurso eletrônico] / Walter Barbosa Soares Junior. - 2023.

Orientador: José Jean Paul Zanlucchi de Souza Tavares.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Engenharia Mecatrônica.
Modo de acesso: Internet.
Inclui bibliografia.
Inclui ilustrações.

1. Mecatrônica. I. Tavares, José Jean Paul Zanlucchi de Souza,1972-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Engenharia Mecatrônica. III. Título.

CDU: 621.03

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

WALTER BARBOSA SOARES JÚNIOR

Desenvolvimento de um Aplicativo em Smartphone utilizando o Node-RED para o Protocolo MQTT Aplicado numa Bancada Eletropneumática

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica

Trabalho aprovado. Uberlândia, 30/11/2023

Banca Examinadora:

_____

Prof. Dr. José Jean Paul Zanlucchi de Souza Tavares

_____

Prof. Dr. Pedro Pio Rosa Nishida

_____

Prof. Ms. Werley Rocherter Borges Ferreira

Dedico este trabalho aos meus pais, pelo
estímulo, carinho e compreensão.

AGRADECIMENTOS

Agradeço ao professor e amigo Prof. Dr. José Jean Paul Zanlucchi de Souza Tavares o incentivo, motivação e orientação nesta caminhada acadêmica, ao Laboratório de Ensino de Mecatrônica e ao Projeto FEMEC Maker pelo espaço e materiais cedidos para a realização deste trabalho.

Aos meus pais por todo o apoio e carinho durante a minha vida. Aos colegas Alisson Vasconcelos, Matheus Alves, e Thiago Santana pela ajuda e contribuição durante as atividades no laboratório.

E aos meus amigos Thiago Santana, Matheus Luiz, Rodrigo Souto, Guilherme Victor e Arnon Carrasco feitos durante esta jornada na UFU e sem a ajuda deles não teria chegado até aqui.

RESUMO

O avanço da tecnologia nos tempos atuais tem se tornado um desafio para a formação do engenheiro mecatrônico pois a IoT, Industria 4.0, MQTT, Node-RED entre outras tecnologias emergentes forçam este profissional a estar cada vez mais atualizado. Este trabalho visou o desenvolvimento de uma aplicação WEB utilizando a ferramenta Node-RED para comandar remotamente uma bancada eletropneumática via protocolo de comunicação MQTT. A solução desenhada para este trabalho foi dividida em duas soluções onde uma utilizava uma placa ESP32 para comandar válvula 5/2 de simples solenoide e retorno por mola e sensores fim de curso e outra solução visou em utilizar placas ESP32 específica para o comando da válvula e outra para coleta dos dados enviados pelos sensores. O servidor broker utilizado neste trabalho é hospedado em uma nuvem externa a rede da Universidade Federal de Uberlândia fazendo necessário o tráfego de informações via internet.

Palavras-chave: MQTT; Node-RED; Industria 4.0, IoT, Cloud Computing, Mecatrônica.

ABSTRACT

The advance of technology today has become a challenge for mechatronics engineers, as IoT, Industry 4.0, MQTT, Node-RED and other emerging technologies force them to be increasingly up-to-date. This work aims to develop a WEB application using the Node-RED tool to control an electropneumatic workbench via the MQTT communication protocol. The solution designed for this work was divided into two solutions where one used an ESP32 board to control a 5/2 valve with a simple solenoid and spring return and end-of-travel sensors and the other solution aimed to use ESP32 boards specifically to control the valve and another to collect the data sent by the sensors. The broker server used in this work is hosted in a cloud external to the Federal University of Uberlandia network, making it necessary to traffic information via the internet.

Keywords: MQTT; Node-RED; Industry 4.0, IoT, Cloud Computing, Mechatronics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

MQTT- *Message Queuing Telemetry Transport*

M2M- *Machine to Machine*

IDE- *Integrated Development Environment*

IOT- *Internet of Things*

IIOT- *Industrial Internet of Things*

IP- *Internet Protocol*

GPIO- *General Purpose Input/Output*

SUMÁRIO

1 INTRODUÇÃO

A Indústria 4.0 representa uma nova era de automação e inteligência na manufatura, que se caracteriza pela utilização de tecnologias inteligentes e conectadas. O advento da Internet das Coisas (IoT) possibilitou a integração de máquinas, equipamentos e dispositivos diversos, permitindo a criação de ambientes altamente colaborativos e conectados.

Nesse contexto, o protocolo MQTT (Message Queuing Telemetry Transport) tem se destacado como uma das principais tecnologias utilizadas na Indústria 4.0. O MQTT é um protocolo de mensagens projetado para dispositivos com recursos limitados, permitindo a comunicação de dados em dispositivos IoT de forma eficiente e confiável.

O MQTT utiliza uma arquitetura publish/subscribe, onde os dispositivos conectados enviam e recebem mensagens em tópicos específicos, tornando a comunicação entre eles mais ágil e escalável. Além disso, o protocolo é extremamente seguro, permitindo a criptografia de dados sensíveis.

Um dos principais benefícios do MQTT é a sua capacidade de integração com serviços de nuvem, permitindo o gerenciamento de dados e o processamento em larga escala. A utilização de serviços de nuvem também permite a coleta e análise de dados em tempo real, tornando possível a tomada de decisões mais precisas e rápidas.

A aplicação do MQTT na Indústria 4.0 tem sido ampla e diversa, sendo utilizado em diversas aplicações industriais, como no controle de qualidade, na gestão de produção, no gerenciamento de estoques e na manutenção preditiva. O protocolo também tem sido utilizado em sistemas de monitoramento ambiental, logística e transporte, entre outras áreas.

O uso do MQTT em conjunto com serviços de nuvem tem permitido a criação de soluções altamente escaláveis e personalizáveis, que atendem às necessidades específicas de cada indústria. Além disso, a integração com outras tecnologias como inteligência artificial e aprendizado de máquina têm ampliado ainda mais as possibilidades de aplicação do MQTT na Indústria 4.0, o que está sendo denominado Internet das Coisas Industrial ou IIoT.

1.1 Objetivos

1.1.1 Objetivo Principal

Este projeto de fim de curso tem como objetivo principal explorar a utilização do protocolo MQTT em uma bancada eletropneumática controlada por meio de um aplicativo desenvolvido na plataforma Node-RED através de servidor broker hospedado em uma infraestrutura de cloud computing. A aplicação desenvolvida teria como finalidade a função de comando remoto para o acionamento da bancada eletropneumática poderá ser utilizada como ferramenta de ensino.

1.1.2 Objetivos Específicos

- Estudo do MQTT
- Utilização do Broker em uma infraestrutura de cloud computing.
- Estudo do Node-RED como ferramenta para desenvolvimento
- Explorar o uso do Arduino IDE para programação da ESP32
- Desenvolver uma aplicação WEB de acesso universal
- Demonstrar os potenciais da IIoT
- Explorar a capacidades e versatilidades da Industria 4.0
- Demonstrar as possibilidades do uso deste trabalho como ferramenta de ensino

1.2 Justificativa do Projeto

A justificativa para este trabalho foi a necessidade de adequações a forma de ensino devido as novas tecnologias que estão se tornando cada vez mais comum no dia a dia da indústria e espera que o formando em engenharia mecatrônica tenha conhecimento em tais tecnologias como o IoT, Industria 4.0, Cloud Computing que são pilares da indústria 4.0 e ferramentas de programação como o Node-RED.

Além disso o processo de aprendizados que foi afetado devido e a pandemia no período de 2020 a 2022 com várias disciplinas ofertando seus conteúdos de forma remota e isso acabou impactando na forma de ensino e aprendizado e este trabalho propõem uma alternativa para que pode ser empregada tanto no ensino remoto quanto presencial.

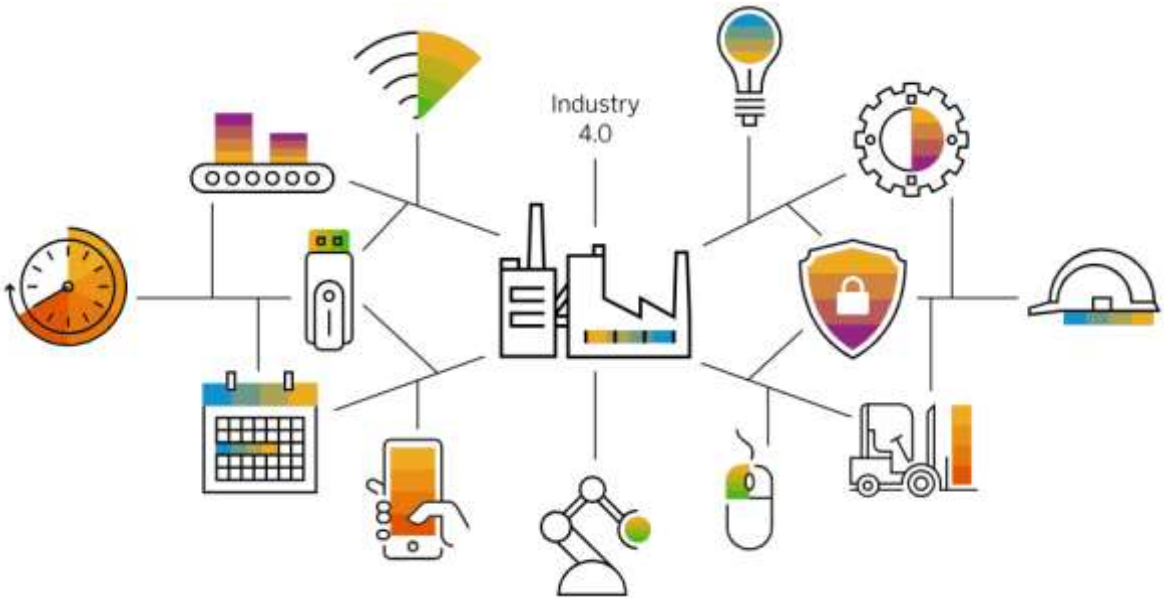
2 REVISÃO BIBLIOGRÁFICA

Neste tópico serão abordados os principais elementos presentes neste projeto como a plataforma ESP32, IOT, protocolo MQTT.

2.1 Industria 4.0

A Indústria 4.0 pode ser definida como a integração de tecnologias digitais inteligentes em processos industriais e de produção. Ela abrange um conjunto de tecnologias que incluem redes industriais de IoT, IA, Big Data, robótica e automação. A Indústria 4.0 permite a produção inteligente e a criação de fábricas inteligentes, conforme apresenta a Figura 1. Ela tem como objetivo aumentar a produtividade, a eficiência e a flexibilidade, possibilitando personalização e tomada de decisões mais inteligentes nas operações de produção e cadeia de suprimentos (SAP).

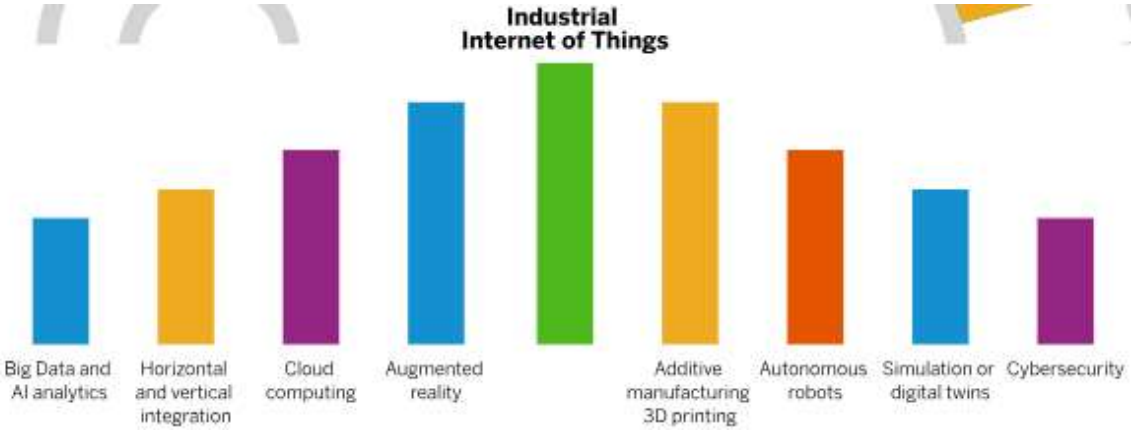
Figura 1:Dispositivos Industriais e Industria 4.0



Fonte: https://www.sap.com/brazil/products/scm/industry-4-0/what-is-iiot.html

A Indústria 4.0 se baseia em nove pilares tecnológicos, como visto na Figura 2 e eles são Big Data, Robôs autônomos, Simulação, Integração de Sistemas, IoT, Cibersegurança, Cloud Computing, Manufatura Aditiva e Realidade Aumentada. Essas inovações ligam os mundos físico e digital e tornam possíveis os sistemas autônomos inteligentes. Empresas e cadeias de suprimentos já usam algumas dessas tecnologias avançadas, mas o potencial pleno da Indústria 4.0 ganha vida quando elas são usadas em conjunto.

Figura 2: Os 9 Pilares da Industria 4.0



A Indústria 4.0 representa mais do que uma simples atualização tecnológica. Ao eliminar silos e conectar equipes e operações em toda a estrutura de produção, você começa a estabelecer uma maneira mais transparente e holística de fazer negócios – que pode eventualmente se estender por todas as áreas de sua organização.

A eficiência operacional é aprimorada com melhor alocação de recursos, tempo de inatividade reduzido e maior produtividade. Essa eficiência se estende a iniciativas de sustentabilidade, em que funções analíticas e automações inteligentes podem ajudar você a simplificar e otimizar ainda mais o uso de energia, reduzir o desperdício e até mesmo projetar e inovar produtos mais sustentáveis ao longo de todo o ciclo de vida.

2.2 IoT

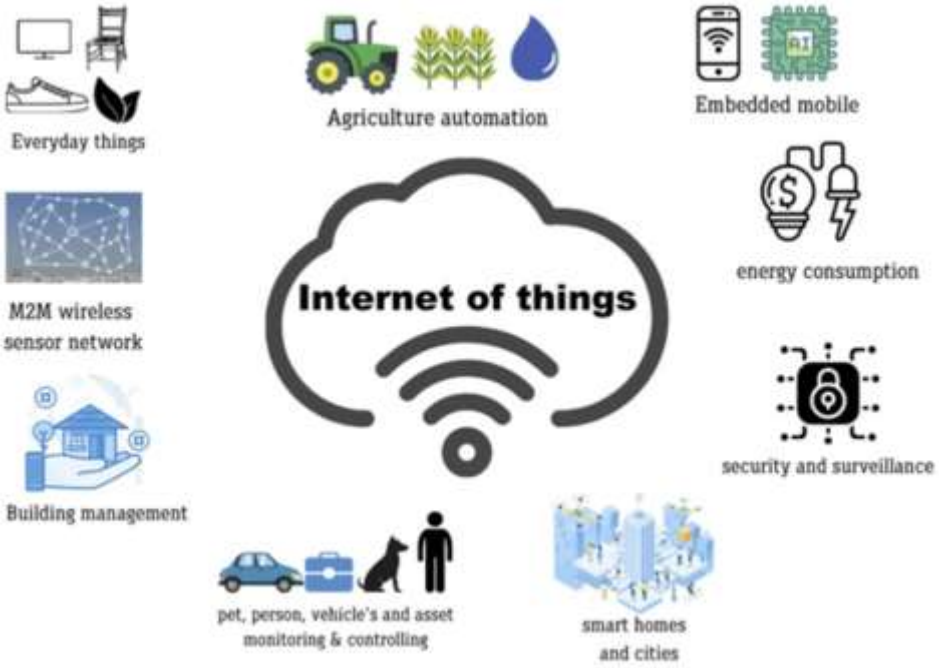
A Internet das Coisas (IoT) descreve a rede de objetos físicos incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas pela internet. Esses dispositivos variam de objetos domésticos comuns a ferramentas industriais sofisticadas. Com mais de 7 bilhões de dispositivos IoT conectados hoje, os especialistas esperam que esse número cresça para 10 bilhões em 2020 e 22 bilhões em 2025 (Oracle).

Um dispositivo com tecnologia IoT nada mais é do que um eletrônico que consegue se comunicar com outros sistemas por meio de uma conexão sem fio (wireless). Em outras palavras, o aparelho é capaz de transmitir dados para uma solução digital, da mesma forma que acontece entre dispositivos conectados à internet. Imagine, por exemplo, um sensor de temperatura de uma máquina. Com a tecnologia IoT, ele pode ser inserido em locais de difícil

acesso. Pela rede sem fio, ele envia os dados coletados em tempo real para um software que faz o monitoramento com alta precisão (TOTVS).

Os benefícios da IoT incluem automação superior que disponibiliza tempo e eficiência de recursos, além de tomada de decisões mais informada com base em maiores quantidades de dados e melhor visibilidade deles. Possíveis preocupações relacionadas à Internet das Coisas incluem riscos de segurança, invasão de privacidade e desafios relacionados à administração de um sistema de dispositivos tão complexo e a Figura 3 a seguir mostra alguns dos dispositivos que utilizam a IoT (VMWARE).

Figura 3: Tipos de dispositivos que utilizam o IoT.



Fonte: JAXEL

2.2.1 IIoT

A Internet Industrial das Coisas (IIoT) é o conjunto de sensores, instrumentos e dispositivos autônomos conectados via Internet a aplicações industriais. Essa rede permite compilar dados, fazer análises e otimizar a produção, aumentando a eficiência e reduzindo os custos do processo de fabricação e prestação de serviços. As aplicações industriais são ecossistemas tecnológicos completos que conectam dispositivos e a estes com as pessoas que gerenciam os processos em linhas de montagem, logística ou distribuição em larga escala (IBERDROLA).

As aplicações atuais da IIoT focam especialmente nos setores manufatureiro, energético e de transporte, com um investimento de mais de 300 bilhões de dólares em todo o mundo em

2019 e espera-se que essa cifra dobre até 2025. No futuro imediato se prevê que a adoção da IIoT se traduza na implantação de mais robôs industriais, como os cobots, sistemas de controle de armazéns e transporte de mercadorias, assim como em sistemas de manutenção preditivos (IBERDROLA).

A diferença entre a Internet das Coisas (IoT) e sua versão industrial (IIoT) é que enquanto a IoT foca nos serviços para os consumidores, a IIoT aumenta a segurança e a eficiência nas fábricas. Por exemplo, as soluções para o consumo se concentram em aparelhos inteligentes para a casa, desde assistentes virtuais até sensores de temperatura ou sistemas de segurança, ou para as pessoas, como os wearables que controlam a saúde. A Figura 4 a seguir mostra a IIoT e alguns exemplos de dispositivos Industriais que a utilizam (IBERDROLA).

Figura 4: IIoT e Dispositivos Industriais

Fonte: https://blog.qbeyond.de/2016/09/iot-vs-iiot-die-besonderheiten-des-industrial-internet/

2.3 Protocolo MQTT

MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação máquina para máquina (M2M - Machine to Machine) com foco IoT que funciona em cima do protocolo TCP/IP. Um sistema MQTT se baseia na comunicação entre cliente e servidor, em que o primeiro pode realizar tanto “postagens” quanto “captação” de informação e o segundo administra os dados a serem recebidos e enviados. Para isso, é utilizado um Paradigma chamado Publish-Subscribe. O protocolo se popularizou pela simplicidade, baixo consumo de dados e pela possibilidade comunicação bilateral (UFRJ).

Nos anos 90 a IBM criou o protocolo MQTT. Sua origem se deu à necessidade de um protocolo simples e leve que conseguisse comunicar várias máquinas entre si, uma

comunicação que ocorreria utilizando microcontroladores para a obtenção de dados que tivesse uma taxa de transmissão leve para a comunicação entre as máquinas e os sensores.

Devido à tão importante funcionalidade de “criar uma conexão” entre pequenos sensores e as máquinas onde esses dados serão tratados, o MQTT é visto como uma das principais tecnologias que podem tanto participar quanto impulsionar o desenvolvimento de uma nova “rede”, a IoT, que já está mostrando sua importância há muito tempo e, com certeza, ganhará ainda mais foco nos próximos anos.

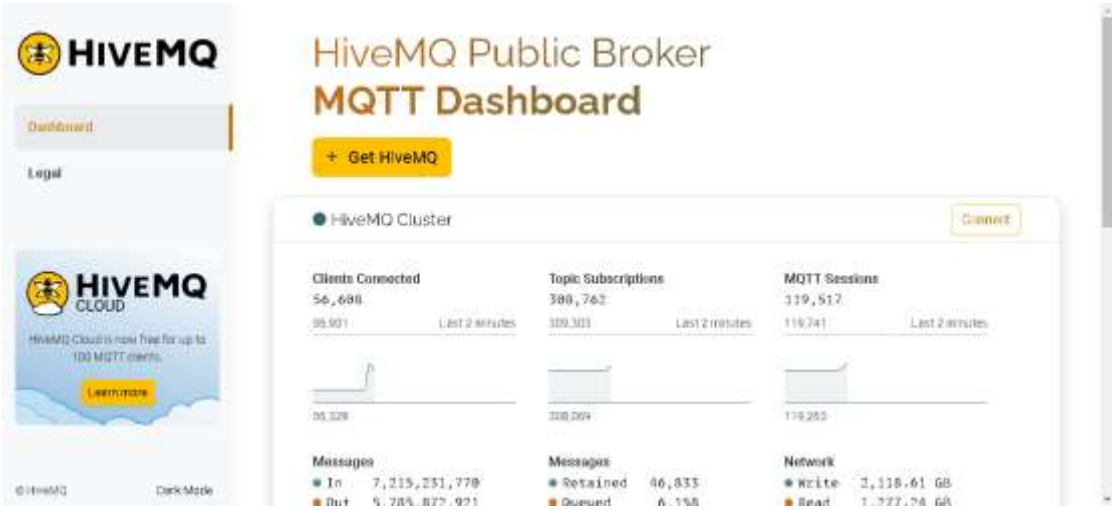
O Broker, que é responsável por filtrar as mensagens e saber exatamente para quem enviar. Dessa forma, o publisher e subscriber não precisam se conhecer diretamente e apenas precisam conhecer o Broker, que é quem fará a notificação da mudança de estados e enviará essa informação para aqueles que tiverem inscritos no tópico referenciado. Por fim, o publisher precisa se preocupar apenas com enviar as informações e estabelecer a conexão exclusivamente com o Broker.

Figura 5: Infraestrutura de Comunicação do MQTT com o *Broker, Publisher e Subscriber*

Fonte: https://emqx.medium.com/a-comprehensive-comparison-of-open-source-mqtt-brokers-2023-e70257cc5b75

É comum em alguns casos os sistemas terem atuações tanto de Publisher quanto de Subscriber. Considerando um caso de sistema que possui um sensor de temperatura e está ligado a um ar-condicionado e um sistema que acompanha a temperatura e envia o sinal para ligar o ar, os dois terão as duas situações, considerando que o sistema 1 precisa publicar as mudanças de temperatura e receber o sinal para ligar o ar, enquanto o sistema 2 precisa receber as informações da temperatura e enviar o sinal para ligar o ar. A Figura 06 mostra um exemplo de servidor broker que está hospedado em uma plataforma de cloud computing.

Figura 6: Servidor Broker HiveMQ

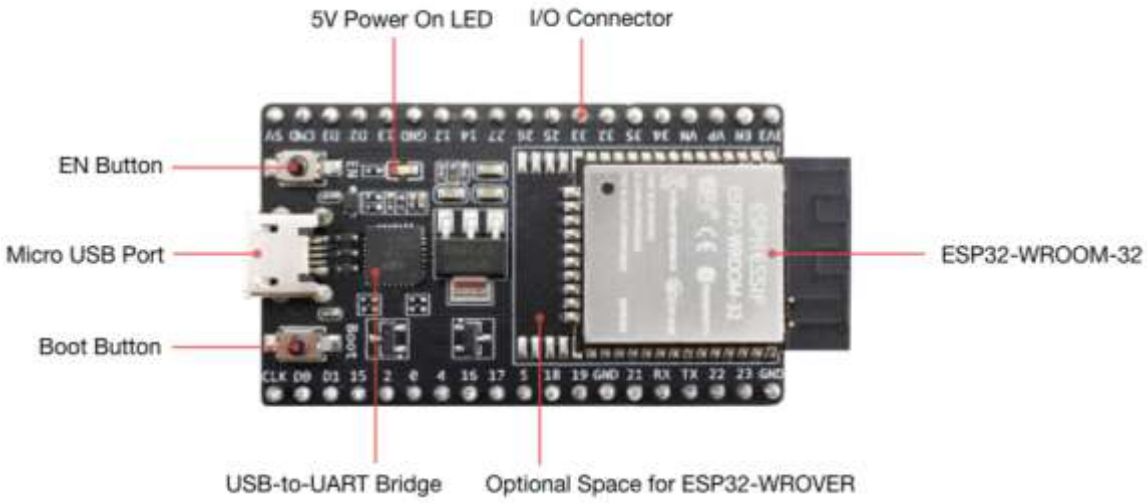


Fonte: https://www.mqtt-dashboard.com/

2.4 Plataforma ESP32

O ESP32 é um dispositivo IoT que consiste em um microprocessador de baixa potência dual core Tensilica Xtensa 32-bit LX6 com suporte embutido à rede WiFi, Bluetooth v4.2 e memória flash integrada. Essa arquitetura permite que ele possa ser programado de forma independente, sem a necessidade de outras dispositivos microcontroladores como o Arduino, por exemplo. Dentre as principais características deste dispositivo, podemos citar: baixo consumo de energia, alto desempenho de potência, amplificador de baixo ruído, robustez, versatilidade e confiabilidade (Masterwalkershop). A Figura 7 mostra os principais componentes presentes na ESP32.

Figura 7: Principais componentes presentes na ESP32



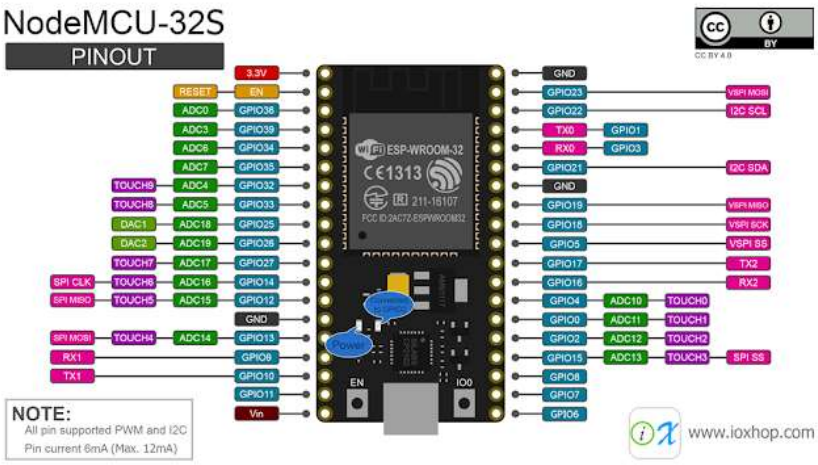
Fonte: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front

A Figura 8 mostra os pinos presentes no ESP32 para a conexão com os dispositivos como sensores e atuadores.

2.5 Computação em Nuvem

A cloud computing é o acesso sob demanda, via internet, a recursos de computação — aplicativos, servidores (físicos e virtuais), armazenamento de dados, ferramentas de desenvolvimento, recursos de rede e muito mais — hospedados em um data center remoto gerenciado por um provedor de serviços em cloud (ou CSP). O CSP disponibiliza esses recursos por uma assinatura mensal ou por um valor cobrado conforme o uso (IBM).

Figura 8: Pinos presentes na ESP32

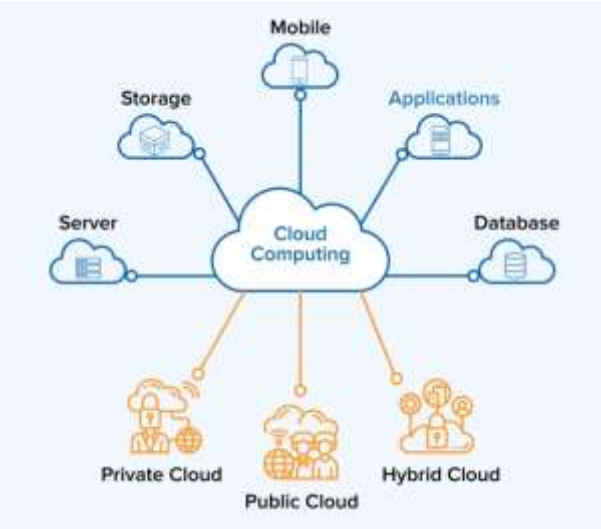


Fonte: https://www.fernandok.com/2018/03/esp32-detalhes-internos-e-pinagem.html

Organizações de todos os tipos, portes e setores usam a nuvem para uma grande variedade de casos de uso, como backup de dados, recuperação de desastres, e-mail, desktops virtuais, desenvolvimento e teste de software, análises de big data e aplicativos web voltados ao cliente. Por exemplo, as empresas do setor de saúde usam a nuvem para desenvolver tratamentos mais personalizados para os pacientes. Empresas de serviços financeiros usam a nuvem como base para detectar e prevenir fraudes em tempo real. E fabricantes de videogames usam a nuvem para entregar jogos online para milhões de jogadores em todo o mundo (AWS).

Para funcionar, a computação em nuvem usa um servidor remoto para que os dispositivos do consumidor ou usuário sejam conectados aos recursos que estão centralizados nesse local. Ele é capaz de armazenar todas as informações e dados necessários para a execução dos programas ou serviços, permitindo a adição, edição ou exclusão de informações e propriedades em qualquer lugar do mundo (INOVA GLOBALWEB). A Figura 9 os principais modelos de serviço que utilizam o cloud computing.

Figura 9: Modelos de serviços que utilizam *Cloud Computing*



Fonte: https://www.tatvasoft.com/blog/cloud-computing-models/

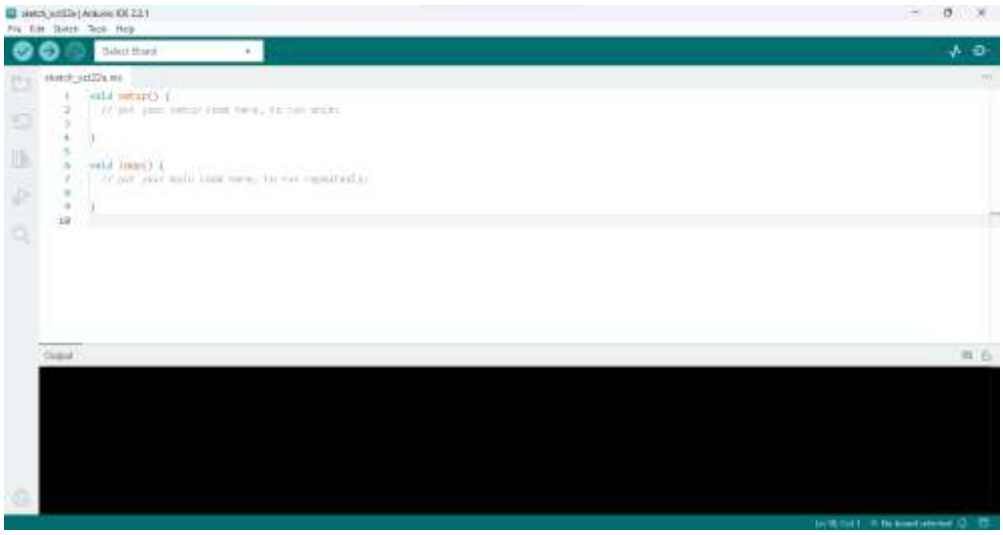
2.6 Arduino IDE

Arduino IDE é o software Arduino gratuito que facilita o desenvolvimento e a gravação de códigos diretamente no microcontrolador. Através deste, é possível realizar o Upload dos códigos para a placa tanto em sistemas operacionais Windows quanto Linux, demonstrando sua funcionalidade e versatilidade.

Além de uma compatibilidade com quase todos os sistemas operacionais, o Arduino IDE torna possível à programação de todos os modelos de placas Arduino e sempre que o projeto apresentar algum problema em sua configuração ou até mesmo em seu código, ele irá gerar uma notificação e elencar onde pode estar o problema.

Atualmente existem dois métodos disponíveis para o Upload de códigos Arduino, seja via Arduino IDE, software instalado junto ao computar, quanto via navegador através de uma plataforma online recentemente desenvolvida pela Empresa Arduino. Além disso a Arduino IDE é compatível com diversas placas além da desenvolvida pela Empresa Arduino como por exemplo as Espressif ESP32 e ESP8266. (Usinainfo) A figura 10 a seguir mostra a interface do usuário do Arduino IDE.

Figura 10: Interface Arduino IDE Versão 2.2.1



Fonte: Autoria Própria

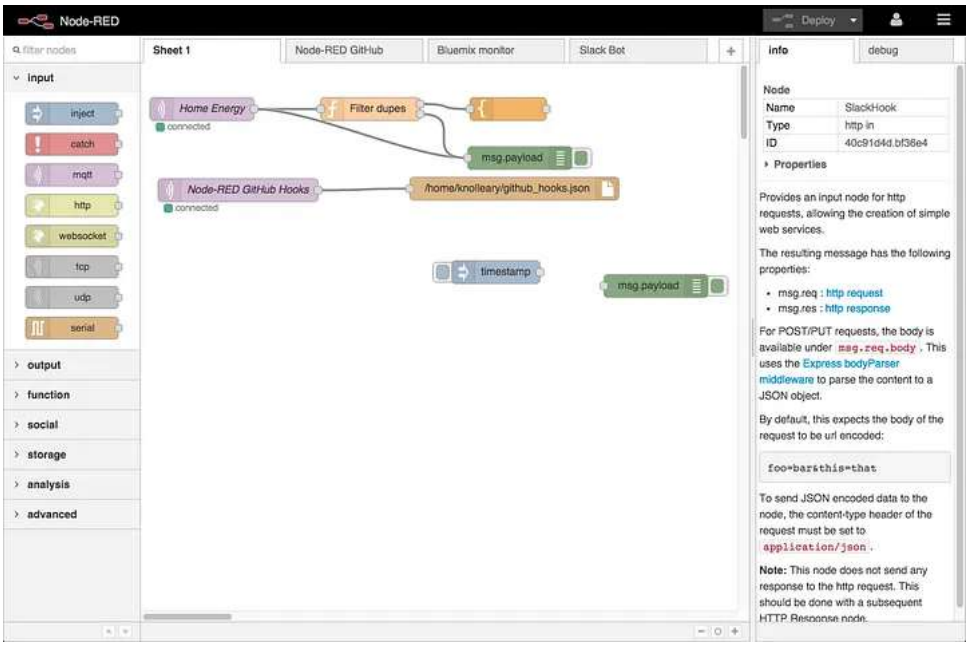
2.7 Node-RED

Node-RED consiste em uma ferramenta de programação bastante versátil e voltada principalmente para o desenvolvimento de soluções de automação baseadas no conceito de IoT. Este recurso utiliza uma abordagem de programação gráfica fazendo com que seja possível criarmos aplicações por meio do estabelecimento de conexões entre blocos (chamados de nós) que possuem funções predefinidas.

O Node-RED possui um editor que deve ser acessado através do navegador de internet do seu computador, portanto, pode-se presumir que ele consiste em uma ferramenta de simples manipulação, não necessitando de etapas de instalação complicadas como vemos em uma série de softwares.

Além da interface de programação, é possível criarmos dashboards contendo as informações pertinentes à sua aplicação e acessá-la, em um primeiro momento, através de qualquer dispositivo existente na rede na qual está o computador em que o Node-RED está instalado. As Figura 11 e 12 mostram um exemplo de programação utilizando os fluxos com os nós de programação e um exemplo de dashboard para monitoramento em tempo real de vários sensores.

Figura 11: Interface de desenvolvimento do Node-RED



Fonte: Autoria Própria

Figura 12: Dashboard em Node-RED

Fonte: https://flows.nodered.org/node/Node-RED-dashboard/in/YSkbb-PzZV4q

3 METODOLOGIA

Neste tópico é descrito o processo pelo qual procedeu o desenvolvimento e a implementação deste trabalho. Nos tópicos seguintes serão descritos os processos de desenvolvimento para a aplicação e a comunicação com a bancada eletropneumática através do protocolo MQTT. Os apêndices A e B tem os detalhes da configuração da Arduino IDE e do Node-RED e os apêndices C, D e E tem os códigos que foram utilizados na ESP32 durante a execução deste trabalho. O código fonte utilizado neste trabalho foi disponibilizado repositório no GitHub do laboratório MAPL-UFU[1].

3.1 Lista de Materiais Utilizados

a) 3 Placas ESP32
b) Protoboard
c) Jumpers
d) 01 módulo relé de um canal
e) 1 atuador de dupla ação
f) 1 válvula eletropneumática 5/2 simples solenoide e retorno por mola
g) Bancada eletropneumática com fonte de ar comprimido
h) 02 sensores fim de curso
i) Um computador com Arduino IDE e Node-RED instalados
j) Servidor *Broker* HIVEMQ

3.2 Aplicação implementada

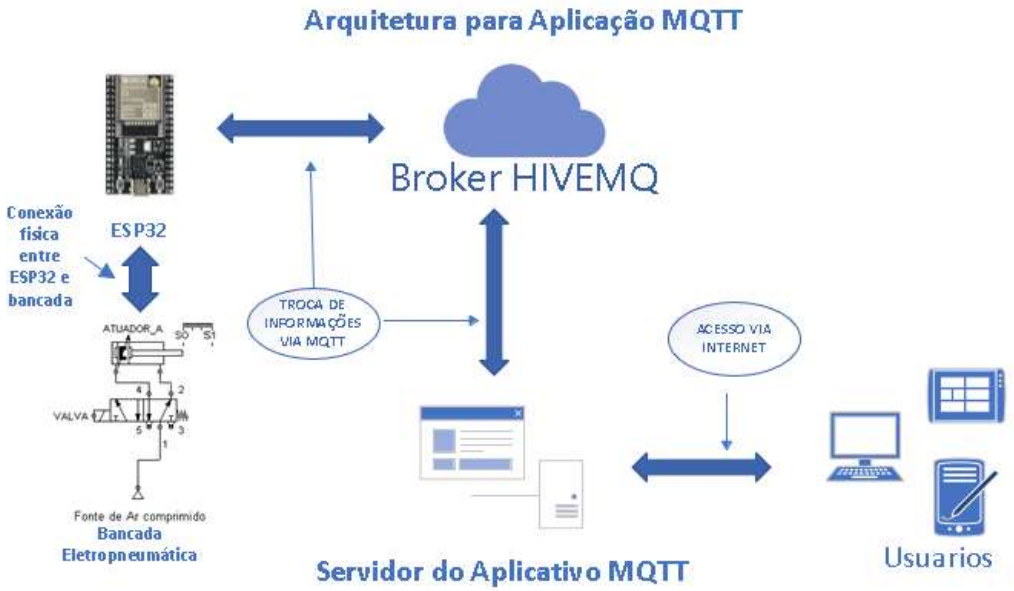
A arquitetura de funcionamento da aplicação será implementada a arquitetura cliente-servidor com o método de comunicação via MQTT onde um cliente é o Publisher e outro é o Subscriber. A aplicação será no estilo dashboard onde possibilita a visualização dos dados enviados pela bancada eletropneumática e o envio de comandos para avançar e recuar o atuador eletropneumático.

Partindo desta base foi considerado durante o processo de desenvolvimento da aplicação o controle de um atuador através de comandos que foram executados uma válvula eletropneumática que fez o atuador avançar e recuar. A aplicação recebia as informações sobre o avanço e recuo do atuador através de sensores fim de curso instalados na bancada eletropneumática e conectados a ESP32. A acesso para a aplicação será via endereço IP do servidor de aplicação no qual ela está executando. Seguindo este processo descrito acima a

[1] Disponível em https://github.com/MAPL-UFU/MQTT_APP_Node-red

aplicação foi desenvolvida para as 02 soluções de conexão do ESP32 com a bancada eletropneumática descritas a seguir. A Figura 13 mostra a arquitetura da aplicação que foi desenvolvida neste trabalho.

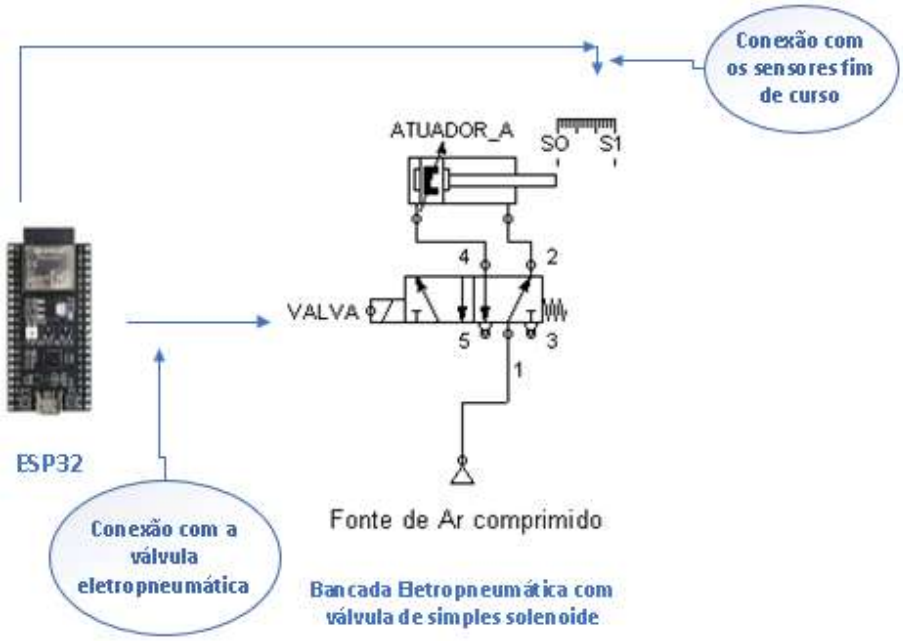
Figura 13: Arquitetura para a aplicação desenvolvida



Fonte: Autoria Própria

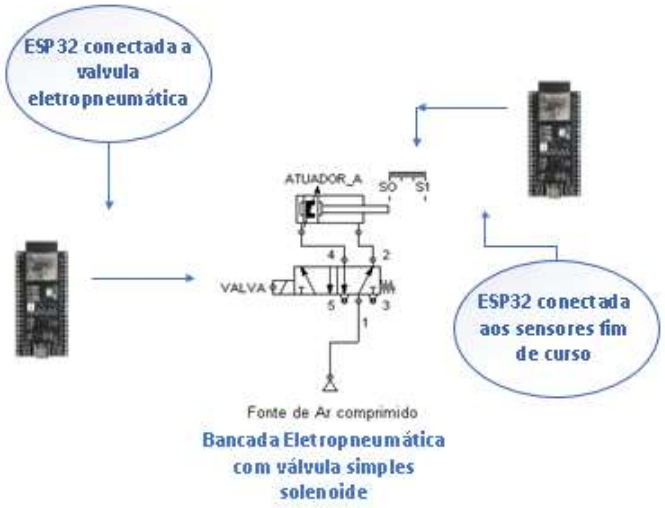
Conforme descrito anteriormente, foram desenvolvidas 2 soluções de conexão e comunicação com a bancada eletropneumática. A solução 01 foi considerando uma válvula 5/2 com simples solenoide e retorno por mola conectada a ESP32 que também gerenciava a informação que era enviada pelos sensores fim de curso (Figura 14). A solução 02 foi com a utilização de dois ESP32 sendo uma para controlar a válvula 5/2 com retorno por mola e outra ESP32 para conectar os sensores fim de curso conforme a indicação da Figura 15 e foi implementada uma versão da aplicação em Node-RED para esta solução onde mostrava o status do sensor fim de curso, ou seja, para mostrar se os sensores estavam conectados com a aplicação via MQTT.

Figura 14: Esquema de implementação da solução 01



Fonte: Autoria Própria

Figura 15: Esquema de implementação da Solução 02



Fonte: Autoria Própria

Toda a comunicação foi realizada através do protocolo MQTT com o servidor Broker HIVEMQ disponível em um ambiente de cloud computing da própria HIVEMQ através do endereço “broker.hivemq.com” e porta 1883 com mostra a Figura 16 a seguir. Os tópicos utilizados para publicação e assinatura dos dados trafegados entre a aplicação e a bancada eletropneumática foram configurados tanto no código da ESP32 e na aplicação no Node-RED.

Figura 16: página do servidor público HIVEMQ

Fonte: Autoria Própria

3.3 Desenvolvimento do código da ESP32

Para o desenvolvimento do código que foi embarcado na ESP32 foi utilizado o Arduino Ide. A lógica empregada na programação do dispositivo foi a que a ESP32 recebesse e enviasse todos os comandos via MQTT. A utilização da IDE do Arduino se deu por ser um ambiente de desenvolvimento bastante didático e de fácil implementação e replicação dos procedimentos utilizados para a programação da ESP32. Nas três soluções desenvolvidas e implementados, a ESP32 foi identificada na IDE como Node32s. No apêndice A consta a configuração da IDE para habilitar o suporte das placas ESP32. A Figura a seguir mostra a IDE do Arduino no processo de desenvolvimento da programação para a ESP32 na solução 01.

Para conexão da ESP32 com a bancada eletropneumática foram utilizadas as GPIOs 2 e 4 para os sensores fim de cursos e para a válvula simples solenoide com retorno por mola foi defina a porta GPIO 05. Esta configuração foi empregada nas soluções 01 e 02, sendo que na solução 02 utiliza-se uma placa ESP32 para controle da válvula eletropneumática e outra para controle dos sensores fim de curso. A Figura 17 mostra um dos códigos fontes desenvolvidos para ser carregado em uma das soluções apresentadas.

Figura 17: Arduino IDE e ESP32

Fonte: Autoria Própria

Para os sensores fim de curso foram utilizadas as GPIOs 02 e 04 foi implementada as mesmas portas digitais de entrada configuradas como INPUT PULLUP utilizadas na solução 01 e 02. Sobre a conexão das GPIO com a válvula eletropneumática é importante frisar que será utilizado um módulo relé para compatibilização dos níveis de tensão elétrica entra a válvula e as portas do ESP 32 uma vez que a ESP32 trabalha com até 5V e a bancada funciona em 24V. a ligação elétrica será realizada conectando a porta solenoide da válvula como positivo e o GND com o relé e da mesma forma a porta 05 e GND da ESP32 com o módulo relé. Essa ligação será implementada também para a solução 02.

Para conexão com a bancada eletropneumática utilizou-se o GND comum entre a placa ESP32, a válvula eletropneumática e os sensores fim de curso para que todos tenham o mesmo referencial de tensão elétrica.

Para conexão com os sensores fim de curso utilizou-se a configuração Normalmente Aberto (NA) para configuração elétrica da comunicação do sensor com a placa ESP32. As Figuras 18 e 19 mostram a configuração das GPIOs para soluções 01 e 02 respectivamente.

Figura 18: Configuração das GPIOs utilizadas na Solução 01

```
// Configuração das portas utilizadas na ESP32
const int S0 = 2;  // Porta para o sensor fim de curso
const int S1 = 4;  // Porta para o sensor fim de curso
const int valvA = 5;  // Porta para a válvula A
```

Fonte: Autoria Própria

Figura 19: Configuração das GPIOs utilizadas na Solução 02 ESP32 válvula e na ESP32 fim de curso

```
PubSubClient client(espClient);  // Objeto para comunicação MQTT
// Configuração das portas utilizadas na ESP32
const int valvA = 5;  // Porta para a válvula A


// Configuração das portas utilizadas na ESP32
const int S0 = 2;  // Porta para o sensor fim de curso
const int S1 = 4;  // Porta para o sensor fim de curso
```

As ações executadas pelo dispositivo foram implementadas através de uma função específica para cada ação que foi executada como mostra a Figura 20, ou seja, foi criada uma função para avançar o atuador eletropneumático, uma função para recuar o atuador, outra para tratar os dados dos sensores fim de curso.

Figura 20: Função Implementadas tanto na solução 01 quanto na 02

```
// Protótipo das funções
void reconnectWiFi();  // Função para reconectar Wi-Fi
void reconnectMQTT();  // Função para reconectar MQTT
void callback(char* topic, byte* message, unsigned int length);  // Função de retorno de chamada MQTT
void publishSensorState(int sensorValue);  // Função para publicar o estado do sensor
void openValve();  // Função para abrir a válvula
void closeValve();  // Função para fechar a válvula
```

Fonte: Autoria Própria

Outras funções foram necessárias e foram implementadas para a conexão com a rede sem fio da UFU e outra conexão com o Broker HIVEMQ que estava alocado na nuvem. Além dessas funções foi implementada também a função callback para obtenção das informações do Broker via o tópico. A Figura 21 mostra a configuração da comunicação da ESP32 utilizando o protocolo MQTT.

Figura 21:Configuração da comunicação via MQTT

```
// Configuração do servidor broker
const char* mqttServer = "broker.hivemq.com";  // Endereço do servidor MQTT
const int mqttPort = 1883;  // Porta do servidor MQTT
```

Fonte: Autoria Própria

Além foram usadas no desenvolvimento dos códigos as bibliotecas “PubSubClient.h” para gerenciamento da comunicação com o Broker para envio e recebimentos das informações e as bibliotecas “esp_wpa2.h” e “WiFi.h” para gerenciamento da comunicação via WI-FI como pode ser visto na Figura 22. Nos anexos A, B e C estão os códigos completos desenvolvidos para a solução 01 e 02. No repositório do GitHub[2] encontra-se o procedimento para conexão da ESP32 com internet através da rede WI-FI.

Figura 22: Bibliotecas utilizadas para a solução 01 e 02

```
// Bibliotecas Usadas
#include <PubSubClient.h>  // Biblioteca para comunicação MQTT
#include <WiFi.h>  // Biblioteca para conexão Wi-Fi
#include "esp_wpa2.h"  // Biblioteca para autenticação WPA2-Enterprise
```

Fonte: Autoria Própria

3.4 Desenvolvimento da Aplicação em Node-RED

Para o desenvolvimento da aplicação foi utilizado o Node-RED como base para a programação dela. A aplicação tem o formato de dashboard e foi implementada com a utilização palete *node-RED-dashboard* tanto para o design quanto para configuração dos botões para envio dos comandos para avançar e recuar o atuador eletropneumático. No apêndice B tem o processo de configuração e instalação do Node-RED utilizado neste trabalho. O servidor da aplicação em Node-RED estava desenvolvida foi o próprio computador utilizado para o desenvolvimento da aplicação. Os demais dispositivos acessavam a aplicação no computador através de endereço IP e porta em um navegador WEB.

Para a comunicação da aplicação com a bancada eletropneumática foram utilizados os nós *MQTT in* e *MQTT out* que foram responsáveis para trocar informações com a bancada. Para exibir o status da bancada, dos sensores fim de curso e alteração do estado do atuador

[2] Disponível em https://github.com/MAPL-UFU/MQTT_APP_Node-red

eletropneumático foi utilizado o nó *ui_text* e para enviar os comandos para recuar e avançar o atuador foi utilizado o nó *ui_button* como pode ser visto nas Figuras 23 e 24.

Figura 23: Nó *ui-button*

Fonte: https://flows.nodered.org/node/Node-RED-dashboard/in/YSkbb-PzZV4q

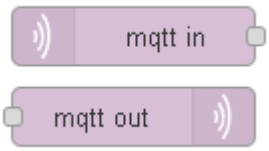
Figura 24: Nó *ui-text*

Fonte: https://flows.nodered.org/node/Node-RED-dashboard/in/YSkbb-PzZV4q

Para a comunicação utilizando o protocolo MQTT foi implementada através da configuração de dois nós o *mqtt in* e o *mqtt out* que são mostrados na Figura 25. Neles foram configurados o endereço do servidor Broker HIVEMQ e a sua porta, o nível de QoS para 0 e o tópico utilizado para enviar ou receber a informação proveniente da bancada como por ser visualizada na Figura 26 a seguir e vale para as duas soluções apresentadas. As tabelas 01 e 02 mostram a relação de tópicos utilizadas na comunicação via MQTT nas soluções 01 e 02.

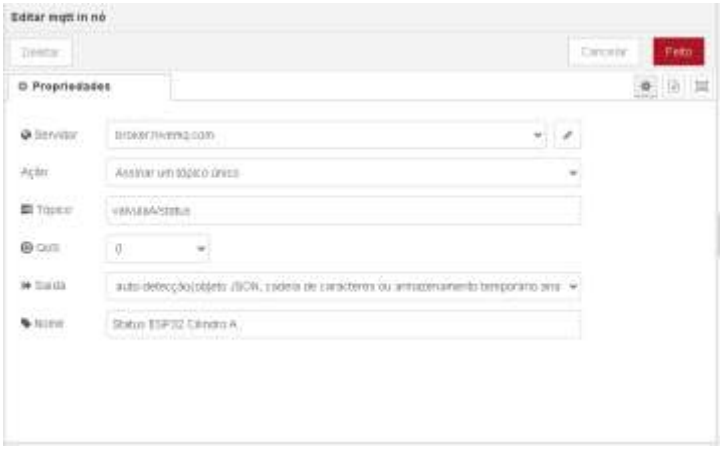
O processo de programação consiste em apenas ligar o nós de comunicação via MQTT com os nós repensáveis para a exibição da informação proveniente da bancada e com os nós responsáveis pelo envio dos comandos para avançar e recuar o atuador eletropneumático. Esse tipo de ligação foi o mesmo implementado nas duas soluções que foram desenvolvidas e a seguir temos as ligações entre os nós para a solução 01 e 02 que mostradas nas Figuras 27 e 28.

Figura 25: Nós para comunicação via MQTT



Fonte: Autoria Própria

Figura 26: ConFiguração da comunicação via MQTT



Fonte: Autoria Própria

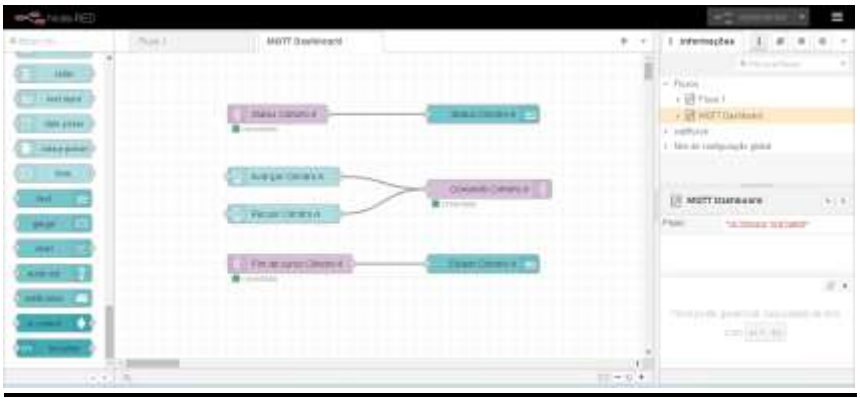
Tabela 1: Relação de Repositórios utilizados na solução 01

Repositório	Função
valvulaA/status	Informar se a ESP32 ligada a bancada eletropneumática está conectada à rede
valvA/comando	Enviar Comando para avançar e recuar
valvulaA/sensorA0A1	Receber atualização quanto a posição do sensor fim de curso

Tabela 2: Relação de Repositórios utilizados na solução 02

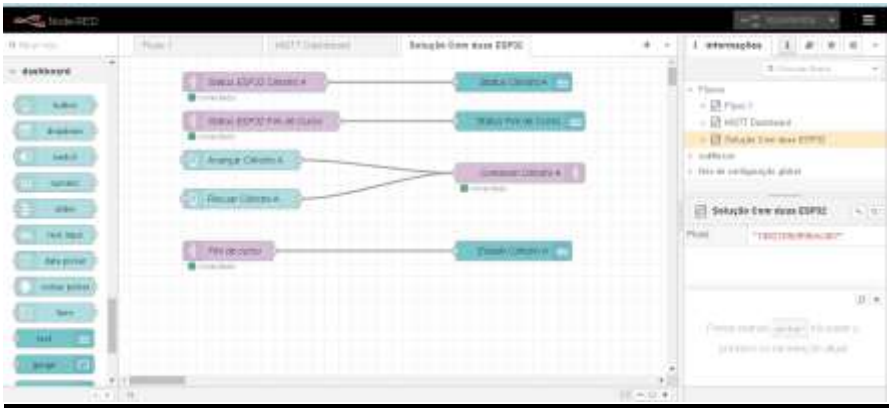
Repositório	Função
valvulaA/status	Informar se a ESP32 ligada a bancada eletropneumática está conectada à rede
Sensores_A/status	Informar se a ESP32 ligado ao sensor fim de curso está conectada à rede
valvA/comando	Enviar Comando para avançar e recuar
Sensores/sensorA0A1	Receber atualização da ESP32 quanto a posição do sensor fim de curso

Figura 27: Ligação dos nós para a aplicação da solução 01



Fonte: Autoria Própria

Figura 28: Ligação dos nós para a aplicação da solução 02

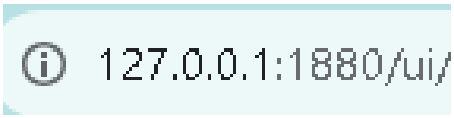


Fonte: Autoria Própria

4 RESULTADOS E DISCUSSÃO

Neste item serão apresentados os resultados obtidos com o desenvolvimento de uma aplicação utilizando o Node-RED para o protocolo MQTT. Para acessar a aplicação devemos utilizar o endereço IP juntamente com a porta 1880 do dispositivo no qual o Node-RED está instalado e após isso a aplicação estará acessível tanto para computadores e para dispositivos móveis como tabletes e smartphones com acesso à internet. A Figura 29 mostra o exemplo de acesso a uma das aplicações desenvolvidas utilizando o endereço IP da placa de rede do computador no qual serviu como servidor da aplicação.

Figura 29: Endereço IP e Porta da Aplicação em Node-RED



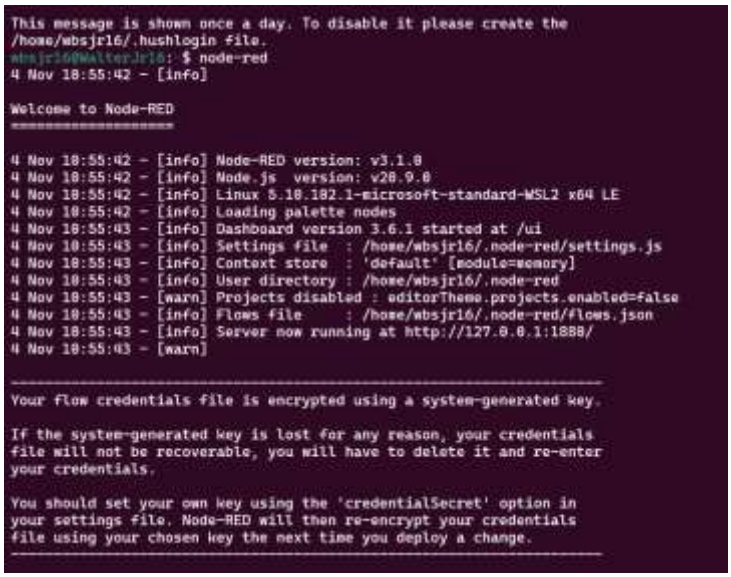
Fonte: Autoria Própria

A aplicação desenvolvida é multiplataforma ele pode ser portado tanto para Linux, Windows, Macintosh e sistemas operacionais mobiles como o Android bastando apenas a instalação da base do nodejs e por seguinte instalar o Node-RED com mostrado no Apêndice B que para efeitos práticos foi utilizado o sistema operacional Windows e após finalizado o seu desenvolvimento a aplicação foi portada para uma máquina virtual com o Linux instalado para testar a portabilidade multiplataforma citada anteriormente e foi seguido os passos descrito no Apêndice B onde após a inicialização do terminal do Linux foi digitado o comando Node-RED para carregar o ambiente de desenvolvimento e por seguinte a aplicação como pode ser observado na Figura 30.

4.1 Montagem da Bancada Eletropneumática:

Para a realização dos testes com as soluções 01 e 02 desenvolvidas neste trabalho foram utilizado bancada eletropneumática mostrada na Figura 31 com fonte de ar comprimido e fonte elétrica de corrente contínua de 24V e demais materiais como válvula eletropneumática 5/2 de simples solenoide e com retorno por mola, atuador eletropneumática de dupla-ação e sensores fim de cursos presente no Laboratório de Ensino de Mecatrônica-LEM 04 que fica localizado na Faculdade de Engenharia Mecânica no Bloco 1DCG no Campus Glória.

Figura 30: Ambiente do Node-RED carregado no Linux.



Fonte: Autoria Própria

Figura 31:Bancada Eletropneumática disponível no Laboratório LEM-04.

Fonte: Autoria Própria

À válvula eletropneumática e os sensores fim de curso foram alimentados pela fonte de 24V mostrada na Figura a seguir e a fonte fornecia energia para a alimentação do módulo relé para o acionamento da válvula através dos comandos enviados pela ESP32 que foram recebidos através do protocolo MQTT. Essa implementação para alimentação foi a mesma para as soluções apresentadas neste trabalho.

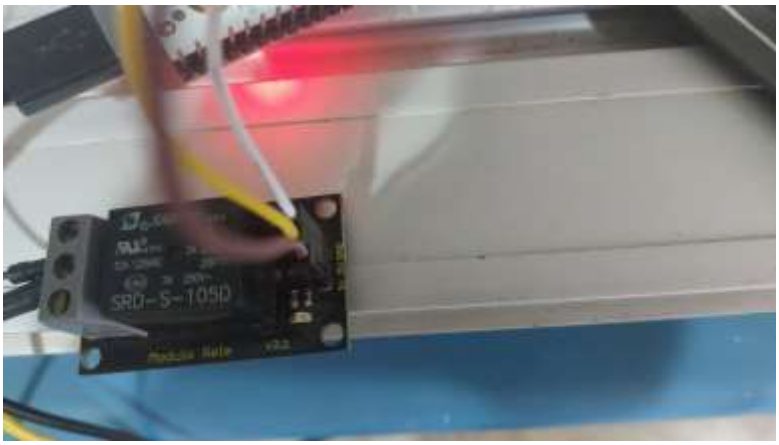
Figura 32:Parte Superior da Bancada Eletropneumática com os Relés e a fonte de 24V

Fonte: Autoria Própria

Figura 33:Ligação da Válvula 5x2 simples solenoide com os Sensores fim de curso.

Fonte: Autoria Própria

Figura 34: Módulo relé utilizado no durante o trabalho.



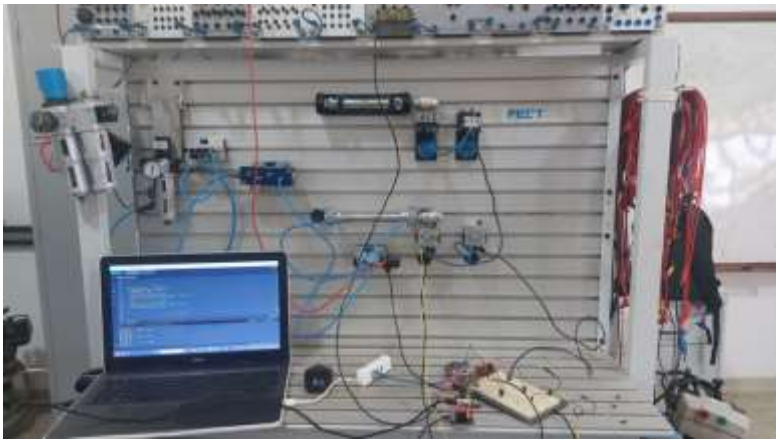
Fonte: Autoria Própria

Após a alimentação da válvula e sensores com a fonte da bancada foi realizada a conexão da bancada com as ESP32 como descrita no item 4.3 relativos ao desenvolvimento do código que foi embarcada nas soluções desenvolvidas e as Figuras 35 e 36 mostram a configuração da bancada eletropneumática para solução 01 e 02 respectivamente. Lembrando que a primeira solução da Figura 35 trouxe uma ESP32 conectada a válvula eletropneumática e aos sensores fim de curso enquanto a Figura 36 mostra a montagem utilizando duas ESP32 sendo uma somente para a válvula e outra para os sensores fim de curso. Os tópicos que foram utilizados para cada solução estão nas Tabelas 01 e 02 no item 4.4 que descreve o processo de desenvolvimento da aplicação em Node-RED.

Figura 35: Montagem da Solução 01

Fonte: Autoria Própria

Figura 36:Montagem da Solução 02

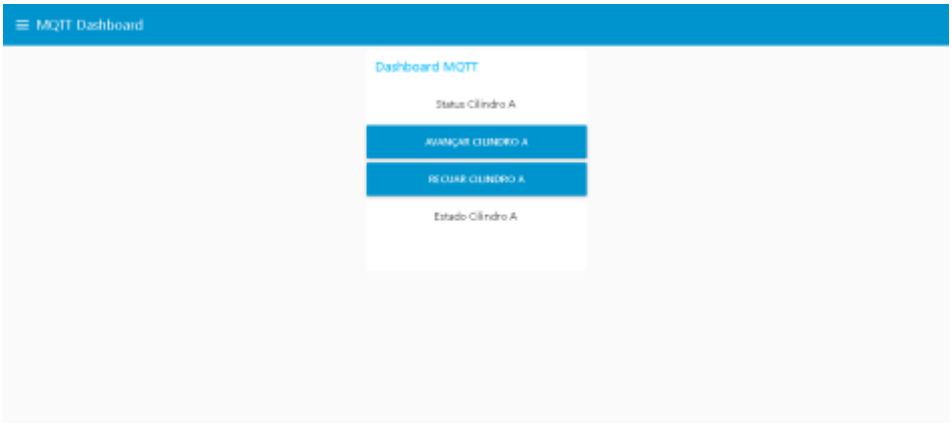


Fonte: Autoria Própria

4.2 Resultados obtidos com a solução 01:

Após a montagem e configuração da bancada eletropneumática mostrada na Figura 35 anteriormente foi realizado os testes de funcionamento tanto da configuração da bancada e da aplicação que foi desenvolvida no estilo de dashboard como citado no item 4.4 e a Figura a seguir mostra a aplicação desenvolvida rodando em navegador WEB com acessado o endereço IP e porta 1880 como citado anteriormente.

Figura 37: Aplicação da Solução 01



Fonte: Autoria Própria

Após acessar a aplicação foram realizados vários testes de envio de comando para a bancada eletropneumática através de smartphones conectados à rede de dados que poderia ser tanto WIFI ou a rede móvel de dados providas por uma operadora de telecomunicações e em ambos os casos os dados foram trafegados e houve o avanço e o recuo do atuador eletropneumático e por conseguinte o envio da informação da bancada para a aplicação como mostrado na Figura 38.

Figura 38:Aplicação da Solução 01 em funcionamento

Fonte: Autoria Própria

A aplicação mostrou as informações de que a válvula estava conectada a internet e disponível para receber os comandos para avançar ou recuar o atuador eletropneumático de dupla ação. A aplicação pode informar sem nenhum tipo de atraso que as atualizações quanto a posição do atuador, ou seja, quando o atuador avançava ou recuava a aplicação pode mostrar esta informação para o usuário sem nenhum tipo de atraso e a Figura 39 a seguir mostra a troca de informação no monitor serial do Arduino IDE relativo as mensagens dos comandos para avançar e recuar o atuador e a conexão da ESP32 ao servidor Broker, o endereço IP da placa e a rede WIFI da UFU no qual estava conectada.

Figura 39: Histórico de informações enviadas para a ESP32

```
Connecting to network: eduroam
.....
WiFi connected
IP address set:
10.14.63.120
Conectando ao Broker
Conectado ao Broker
Message arrived on topic: valvA/comando. Message: Avançar Cilindro A
Message arrived on topic: valvA/comando. Message: Recuar Cilindro A
Message arrived on topic: valvA/comando. Message: Avançar Cilindro A
Message arrived on topic: valvA/comando. Message: Recuar Cilindro A
```

Fonte: Autoria Própria

4.3 Resultados obtidos com a solução 02:

A aplicação desenvolvida para este caso como mostra a Figura 40 apresenta algumas alterações quanto a configuração da bancada eletropneumática pois para este caso foi testado a efetividade de se ter uma ESP32 para sensores fim de curso e outra para a válvula eletropneumática como é mostrado na Figura 36 vista anteriormente e assim como na solução 01 a aplicação foi acessada utilizando o endereço IP e porta do computador no qual a aplicação foi desenvolvida.

Figura 40: Aplicação desenvolvida para solução 02

Fonte: Autoria Própria

Assim como mostrado na solução 01 os resultados obtidos na solução 02 foram os mesmos. Para esta solução como mostrado na Figura 41 a seguir temos os status de conexão tanto da ESP32 conectada a válvula eletropneumática e a que está conectada aos sensores fim de curso além da atualização da posição do atuador que é enviada pelos sensores.

Também foi observado que assim como na solução 01 não nenhum tipo de atraso no envio ou no recebimento da informação que estavam sendo trafegados entre a bancada e a aplicação, ou seja, toda a atualização do posicionamento do atuador como avançado ou recuado era mostrado na aplicação para o usuário sem nenhum tipo de atraso sendo que o servidor broker utilizado estava rodando na nuvem em um domínio foram da rede da UFU. E como mostrado na Figura 39 pode-se observar no monitor serial do Arduino IDE a troca de informação entre a aplicação e a ESP32 conectada a válvula para avançar ou recuar o atuador eletropneumático.

Figura 41: Aplicação da Solução 02 em funcionamento

Fonte: Autoria Própria

5 CONCLUSÃO

O estudante de engenharia mecatrônica durante a sua graduação tem diversos desafios relativos ao surgimento de novas tecnologias e escolher um caminha no qual deve seguir. Este trabalho pode proporcionar o contato com diversas destas novas tecnologias que estão emergindo e com interface direta com o setor industrial.

Após a realização dos testes e estudos realizados durante o desenvolvimento deste trabalho podemos concluir que ele obteve sucesso em desenvolver uma solução de aplicação para o protocolo MQTT utilizando o Node-RED como base para o comando remoto com a bancada eletropneumática do laboratório LEM-04 presente na UFU.

Na solução 01 com a utilização de apenas uma placa ESP32 pode concluir que ao combinar a aplicação como processamento das informações disponibilizadas pela bancada eletropneumática através da ESP32 via protocolo MQTT provou ser uma solução barata e bastante eficaz pois pode se provar a eficiência da descentralização do controle da tomada de decisão de um processo em diversos dispositivos e meios como foi apresentado nesta solução para este trabalho.

Na solução 02 onde foram utilizadas duas placas ESP32 sendo uma para conectar com a válvula eletropneumática e outra para conexão com os sensores fim de curso também pode concluir que a combinação desta solução com a aplicação desenvolvida obteve sucesso assim como na solução 01 e ao utilizar duas placas para o gerenciamento da conexão com a bancada pode-se comprovar também a eficácia na descentralização do controle na tomada de decisão como descrita anteriormente.

Outra conclusão importante deste trabalho é que a utilização de tecnologias emergentes como a IIoT via o protocolo MQTT e a cloud computing que são pilares da indústria 4.0 e pode ser de grande ajuda para a construção da formação do estudante de engenharia mecatrônica em meio ao surgimento de novas tecnologias e a metodologia como a desenvolvida neste trabalho pode ser de grande ajuda na construção do conhecimento nas disciplinas do curso de graduação como Redes Industriais, Automação Industrial e até mesmo a disciplina de Instrumentação Industrial.

6 REFERÊNCIAS

AWS, Disponível em <https://aws.amazon.com/pt/what-is-cloud-computing/> acessado em 02/08/2022

Materwakershop, disponívelem<https://blogmasterwalkershop.com.br/embarcados/esp32/conhecendo-o-nodemcu-32s-esp32>acessado em 02/08/2022 as 22:36

Computer weekly, disponível em <https://www.computerweekly.com/br/definicoe/Internet-dascoisasindustrialIIoT> acessado em 02/08/2022 às 22:27

Curto-circuito, disponível em<https://www.curtocircuito.com.br/blog/Categoria%20IoT/conhecendo-esp32> acessado em 02/08/2022 as 22:44

ESPRESSIF, Disponível em < https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front > acessado em 20/08/2023

FACECONTROL, Disponível em < https://facecontrol.com.br/automacao-industrial-tudo-o-que-voce-precisa-saber/> acessado em 22/10/2023

HIVEMQ, Disponível em < https://www.hivemq.com/public-mqtt-broker/ > acessado em 20/08/2023

IBERDROLA, Disponível <https://www.iberdrola.com/inovacao/o-que-e-iiot> acessado em 22/10/2023

IBM, Disponível em <https://www.ibm.com/br-pt/cloud/learn/cloud-computing> acessado em 02/08/2022

INOVA GLOBALWEB, Disponível em <https://inova.globalweb.com.br/post/cloud-computing> acessado em 02/08/2022 as 22:57

JAXEL, Disponível em < https://jaxel.com/3-iot-applications-that-will-save-your-business-money/> acessado em 22/10/2023

Node-RED-Dashboard, Disponível em <https://flows.nodered.org/node/Node-RED-dashboard/in/YSkbb-PzZV4q> acessado em 02/11/2023

OPENCADD, Disponível em <https://www.opencadd.com.br/blog/9-pilares-da-industria-4-0> acessado em 15/11/2023

Oracle, Disponível em <https://www.oracle.com/br/internet-of-things/what-is-iot/> acessado em 02/08/2022 as 22:12;

SAP, Disponível em <https://www.sap.com/brazil/products/scm/industry-4-0/what-is-industry-4-0.html> acessado em 21/10/2023

TECHTUDO, Disponível em < https://www.techtudo.com.br/noticias/2012/03/o-que-e-cloud-computing.ghtml> acessado em 22/10/2023

TOTVS, Disponível <https://www.totvs.com/blog/inovacoes/aplicacoes-da-internet-das-coisas/> em acessado em 02/08/2022 22:30

UFRJ, Disponível em < https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/> acessado em 22/10/2023

VMWARE, Disponível em <https://www.vmware.com/br/topics/glossary/content/internet-things-iot.html> acessado em 02/08/2022 às 22:16;

APÊNDICE A – Configuração do Arduino IDE para Programação do ESP32

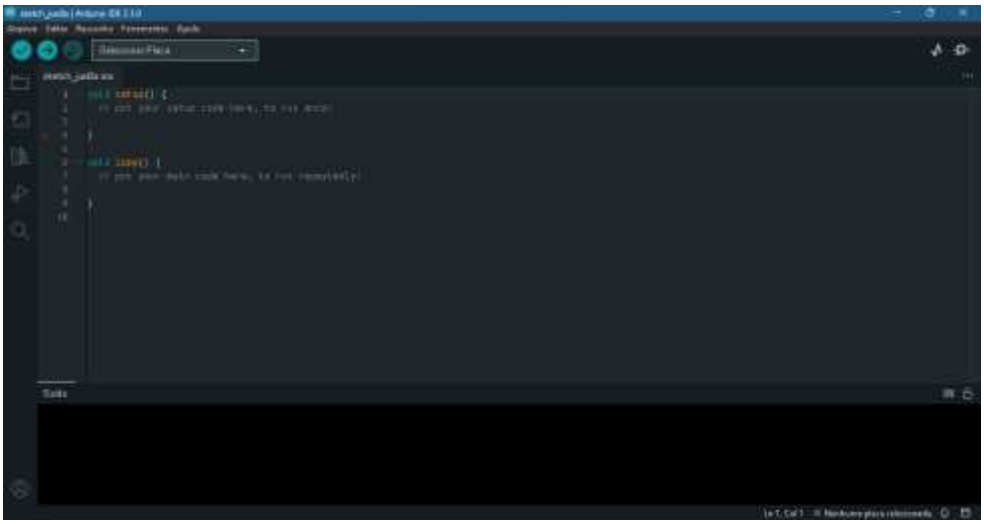
A tarefa inicial foi a configuração o Arduino IDE no notebook instalado com o sistema operacional MS-Windows para a programação do ESP32. O primeiro passo foi após a instalação do Arduino IDE. Nesta esta irá configurar o ambiente para a programação da placa ESP32. Para isto bastamos adicionar o repositório oficial da Espressif no IDE e em seguida instalar e configurar os arquivos que virão do repositório.

Para adicionar o suporte da Espressif primeiro deve adicionar o endereço do repositório no Arduino IDE no menu arquivo e no submenu de preferencias. Dentro de preferências há um campo onde deve-se inserir o endereço web disponível em "https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html" e depois confirmar a inserção do endereço. Deve-se observar que há duas versões do arquivo de configuração disponíveis a *Stable* e *a Development* release como poderá ser observado na imagem a seguir. Para este projeto foi utilizado a versão *Stable*. As imagens a seguir mostram a evolução da configuração do ESP32 no Windows.

Figura 42: Endereços dos repositórios da Espressif

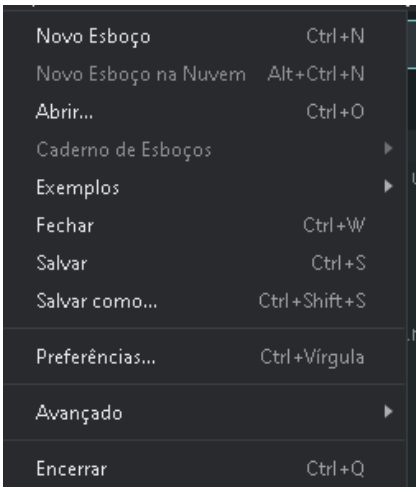
Fonte: Autoria Própria

Figura 43: Interface do Arduino IDE



Fonte: Autoria Própria

Figura 44: Menu de Opções do Arduino IDE



Fonte: Autoria Própria

Figura 45: Menu de Preferências do Arduino IDE

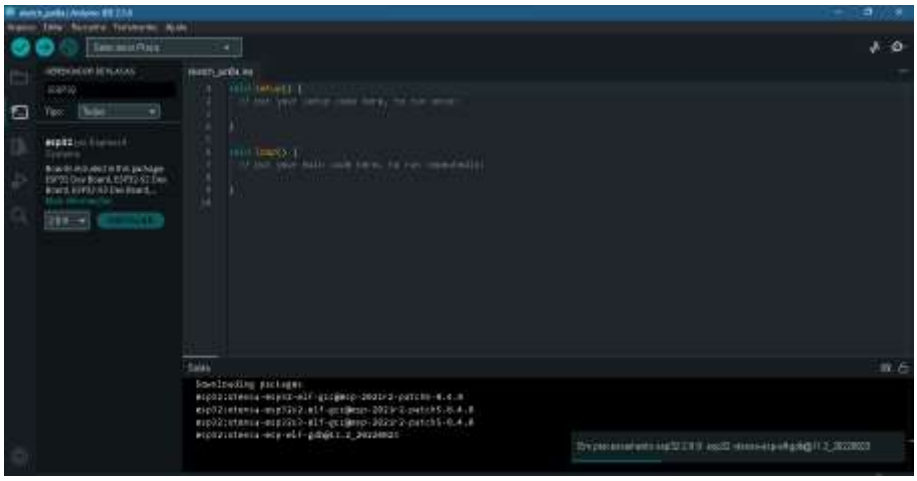
Fonte: Autoria Própria

O segundo passo foi a instalação dos arquivos para o Arduino IDE consiga enxergar a placa dentro do software. Para isto foi em gerenciamento de placa foi realizada a busca pelo arquivo de configuração ESP32 fornecido pela Espressif. Para este trabalho foi utilizada a versão 2.0.9. as imagens a diante mostram o processo de instalação da configuração da placa.

Figura 46:Gerenciador de Placas do Arduino IDE

Fonte: Autoria Própria

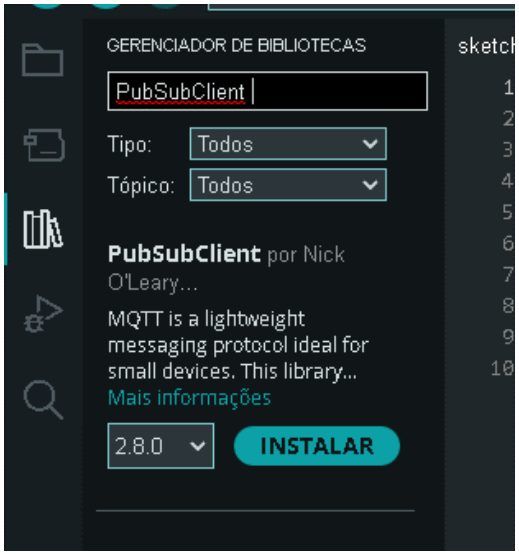
Figura 47: Instalação do Pacote da ESP32 no Arduino IDE



Fonte: Autoria Própria

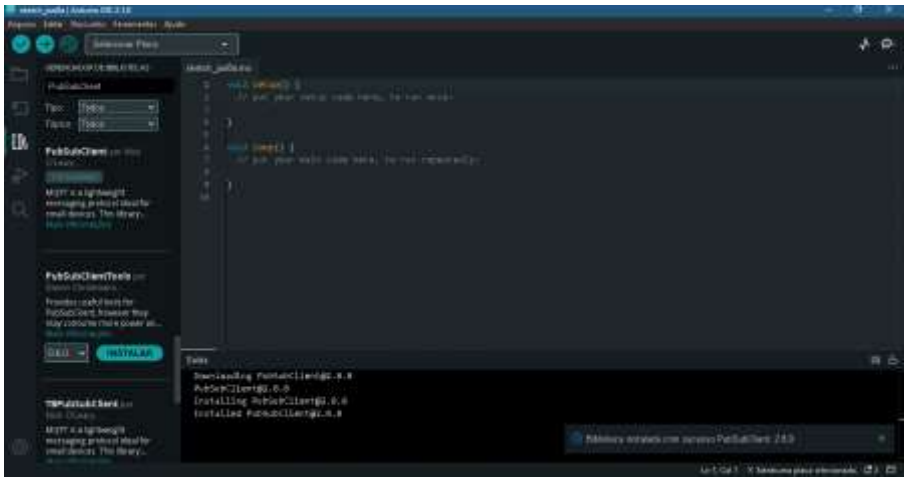
E por último foi realizada a instalação da biblioteca de comunicação para o protocolo MQTT. Para este trabalho utilizou a biblioteca PubSubClient desenvolvida por Nick O'leary na versão 2.8.0 disponível para instalação no gerenciador de biblioteca do ambiente desenvolvimento do Arduino. Para a instalação foi feita uma pesquisa pela biblioteca dentro do gerenciador de bibliotecas e em seguida a instalação e a seguir temos a visualização deste processo.

Figura 48:Gerenciador de Bibliotecas do Arduino IDE



Fonte: Autoria Própria

Figura 49: Instalação da Biblioteca PubSubClient



Fonte: Autoria Própria

APÊNDICE B – Instalação e Configuração do Node-RED

Para instalar e configurar o Node-RED no sistema Linux é preciso instalar o nodejs para poder o roder o Node-RED como servidor da aplicação e o processo de instalação é via terminal. Para instalar o nodejs deve executar o seguinte comando no terminal do Linux “curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -” para baixar os arquivos no repositório do Node-RED e para em seguida instalar via o comando “sudo apt-get install -y nodejs”. As Figuras 50 e 51 a seguir mostram o processo de instalação do nodejs.

Figura 50: Configuração do repositório da Nodejs no Linux

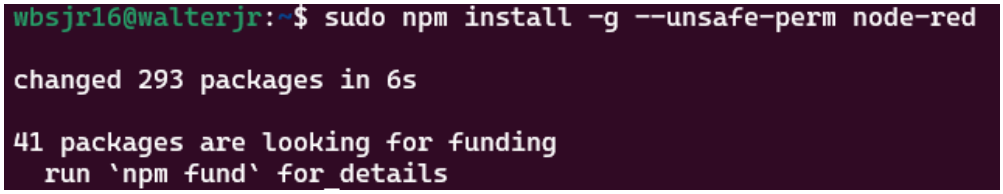
Fonte: Autoria Própria

Figura 51: Instalação do Nodejs no Linux via terminal

Fonte: Autoria Própria

Após a instalação do nodejs seguimos com a instalação do Node-RED via o comando “sudo npm install -g --unsafe-perm Node-RED” que é feita instalação e configuração do Node-RED e pode ser vista na Figura 52.

Figura 52: instalando o Node-RED via terminal

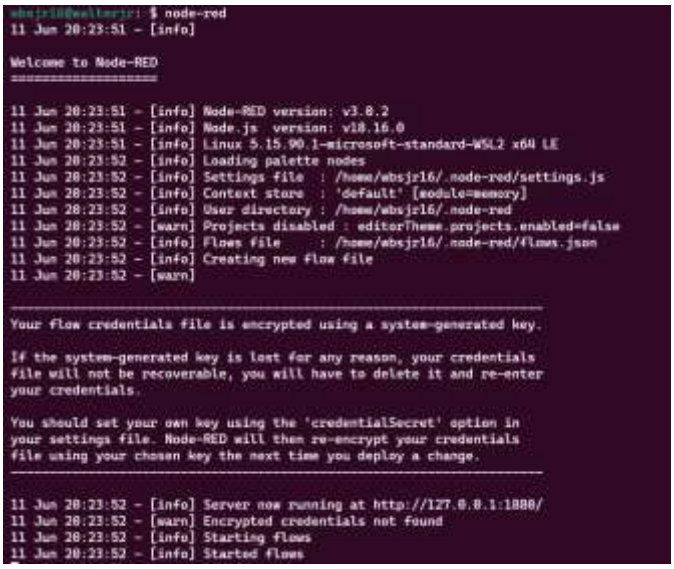


Fonte: Autoria Própria

Feita a instalação podemos testar se o Node-RED está rodando localmente na máquina via comando no terminal Node-RED onde será chamado o software para execução e onde podemos acessar via o endereço IP http://127.0.0.1:1880/ utilizando a porta 1880 ou o endereço local localhost:1880. Para acessar o Node-RED instalado via outra máquina é só conhecer o

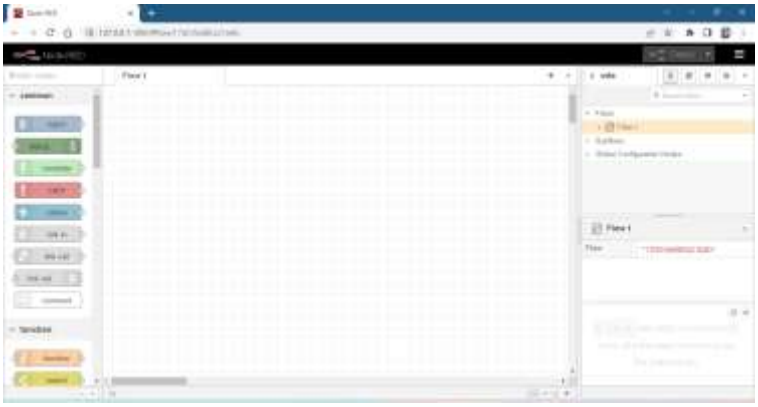
endereço IP dela e utilizar a porta 1880 para acessar a aplicação com mostrado nas Figuras 53 e 54.

Figura 53: Execução do comando node-red no terminal do Linux.



Fonte: Autoria Própria

Figura 54: Interface para programação com os nós do Node-RED



Fonte: Autoria Própria

APÊNDICE C – Código da ESP32 para a Solução 01

```
// Bibliotecas Usadas
#include <PubSubClient.h>  // Biblioteca para comunicação MQTT
#include <WiFi.h>  // Biblioteca para conexão Wi-Fi
#include "esp_wpa2.h"  // Biblioteca para autenticação WPA2-Enterprise

// ConFiguração da conexão wireless do ESP32 no modo PEAP sem autenticação
#define EAP_IDENTITY "xxxx"  // Identidade para autenticação PEAP
#define EAP_USERNAME "xxxx"  // Nome de usuário para autenticação PEAP
#define EAP_PASSWORD "xxxx"  // Senha para autenticação PEAP
const char* ssid = "UFU-Institucional";  // SSID da rede Wi-Fi

// Inicialização dos objetos das Classes das bibliotecas utilizadas
WiFiClient espClient;  // Objeto para conexão Wi-Fi
PubSubClient client(espClient);  // Objeto para comunicação MQTT

// ConFiguração das portas utilizadas na ESP32
const int S0 = 2;  // Porta para o sensor fim de curso
const int S1 = 4;  // Porta para o sensor fim de curso
const int valvA = 5;  // Porta para a válvula A

// ConFiguração do servidor broker
const char* mqttServer = "broker.hivemq.com";  // Endereço do servidor
MQTT
const int mqttPort = 1883;  // Porta do servidor MQTT

// Protótipo das funções
void reconnectWiFi();  // Função para reconectar Wi-Fi
void reconnectMQTT();  // Função para reconectar MQTT
void callback(char* topic, byte* message, unsigned int length);  // Função
de retorno de chamada MQTT
void publishSensorState(int sensorValue);  // Função para publicar o
estado do sensor
void openValve();  // Função para abrir a válvula
void closeValve();  // Função para fechar a válvula

void setup() {
  Serial.begin(115200);  // Inicia a comunicação serial em 115200 bauds
  delay(10);
  Serial.println();
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
  WiFi.disconnect(true);  // Desconecta do Wi-Fi para conFigurar nova
conexão Wi-Fi
  WiFi.mode(WIFI_STA);  // Inicializa o modo Wi-Fi

  // Conexão à rede Wi-Fi usando o protocolo WPA2-Enterprise com PEAP
```

```
          WiFi.begin(ssid,    WPA2_AUTH_PEAP,    EAP_IDENTITY,    EAP_USERNAME,
EAP_PASSWORD);

        pinMode(S0, INPUT_PULLUP);  // Define o sensor fim de curso S0 como
INPUT_PULLUP
        pinMode(S1, INPUT_PULLUP);  // Define o sensor fim de curso S1 como
INPUT_PULLUP
       pinMode(valvA, OUTPUT);  // Define a saída para a válvula A como OUTPUT
        digitalWrite(valvA, LOW);  // Inicialmente, a válvula A está fechada
(LOW)

        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
             if  (WiFi.status()  ==  WL_CONNECT_FAILED  ||  WiFi.status()  ==
WL_CONNECTION_LOST) {
            reconnectWiFi();  // Reconecta Wi-Fi em caso de falha
          }
        }

        Serial.println("");
        Serial.println("WiFi connected");
        Serial.println("IP address:");
        Serial.println(WiFi.localIP());

        client.setServer(mqttServer, mqttPort);  // ConFigura o servidor MQTT
         client.setCallback(callback);  // ConFigura a função de retorno de
chamada MQTT

        reconnectMQTT();  // Reconecta MQTT
      }

      void loop() {
        if (!client.connected()) {
          reconnectMQTT();  // Reconecta MQTT se a conexão foi perdida
        }

        client.loop();  // Mantém a comunicação MQTT ativa

        int sensorValue = digitalRead(S1);  // Lê o estado do sensor fim de
curso S1
        publishSensorState(sensorValue);  // Publica o estado do sensor

        delay(10);
      }

      // Função para reconectar Wi-Fi
      void reconnectWiFi() {
```

```
        Serial.println("WiFi connection lost or failed. Reconnecting...");

        int counter = 0;
        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
          counter++;

          if (counter >= 60) {
            ESP.restart();  // Reinicia o ESP32 após 30 segundos de tentativas
de conexão
          }
        }

        Serial.println("");
        Serial.println("WiFi reconnected");
        Serial.println("IP address:");
        Serial.println(WiFi.localIP());
      }

      // Função para reconectar MQTT
      void reconnectMQTT() {
        while (!client.connected()) {
          Serial.println("Connecting to MQTT broker...");

          if (client.connect("Esp32ValvulaA")) {
            Serial.println("Connected to MQTT broker");

            const char* msgvalA_status = "ESP32A Connectado";
             client.publish("valvulaA/status", msgvalA_status);  // Publica o
status do ESP32 na válvula A

           client.subscribe("valvA/comando");  // Subscreve no tópico MQTT para
receber comandos
          } else {
             Serial.print("Failed to connect to MQTT broker. Retrying in 5
seconds...");
            delay(5000);
          }
        }
      }

      // Função de retorno de chamada MQTT
      void callback(char* topic, byte* message, unsigned int length) {
        Serial.print("Message arrived on topic: ");
        Serial.print(topic);
        Serial.print(". Message: ");
```

```
        for (int i = 0; i < length; i++) {
          Serial.print((char)message[i]);
        }
        Serial.println();

        int command = message[0] - '0';  // Converte o primeiro caractere da
mensagem em um valor inteiro

        if (command == 1) {
          openValve();  // Avança o atuador
           publishSensorState(digitalRead(S1));  // Publica o estado do sensor
fim de curso S1
        } else if (command == 0) {
          closeValve();  // Recua o atuador
           publishSensorState(digitalRead(S0));  // Publica o estado do sensor
fim de curso S0
        }
      }

      // Função para publicar o estado do sensor
      void publishSensorState(int sensorValue) {
        if (sensorValue == LOW) {
          //Serial.println("Atuador A Avançado");
          client.publish("valvulaA/sensorA0A1", "Atuador A Avançado");
          Serial.println();
        } else {
          //Serial.println("Atuador A Recuado");
          client.publish("valvulaA/sensorA0A1", "Atuador A Recuado");
        }
      }

      // Função para abrir a válvula
      void openValve() {
        digitalWrite(valvA, HIGH);  // Define o pino da válvula A como HIGH
(aberto)
      }

      // Função para fechar a válvula
      void closeValve() {
         digitalWrite(valvA, LOW);  // Define o pino da válvula A como LOW
(fechado)
      }
```

APÊNDICE D – Código da ESP32 para a Solução 02 – Válvula

```
// Bibliotecas Usadas
#include <PubSubClient.h>  // Biblioteca para comunicação MQTT
#include <WiFi.h>  // Biblioteca para conexão Wi-Fi
#include "esp_wpa2.h"  // Biblioteca para autenticação WPA2-Enterprise
// ConFiguração da conexão wireless do ESP32 no modo PEAP sem autenticação
#define EAP_IDENTITY "xxxx"  // Identidade para autenticação PEAP
#define EAP_USERNAME "xxxx"  // Nome de usuário para autenticação PEAP
#define EAP_PASSWORD "xxxxx"  // Senha para autenticação PEAP
const char* ssid = "eduroam";  // SSID da rede Wi-Fi
// Inicialização dos objetos das Classes das bibliotecas utilizadas
WiFiClient espClient;  // Objeto para conexão Wi-Fi
PubSubClient client(espClient);  // Objeto para comunicação MQTT
// ConFiguração das portas utilizadas na ESP32
const int valvA = 5;  // Porta para a válvula A
// ConFiguração do servidor broker
const char* mqttServer = "broker.hivemq.com";  // Endereço do servidor
MQTT
const int mqttPort = 1883;  // Porta do servidor MQTT

// Protótipo das funções
void reconnectWiFi();  // Função para reconectar Wi-Fi
void reconnectMQTT();  // Função para reconectar MQTT
void callback(char* topic, byte* message, unsigned int length);  // Função
de retorno de chamada MQTT
void openValve();  // Função para abrir a válvula
void closeValve();  // Função para fechar a válvula
void setup() {
  Serial.begin(115200);  // Inicia a comunicação serial em 115200 bauds
  delay(10);
  Serial.println();
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
   WiFi.disconnect(true);  // Desconecta do Wi-Fi para conFigurar nova
conexão Wi-Fi
  WiFi.mode(WIFI_STA);  // Inicializa o modo Wi-Fi
  // Conexão à rede Wi-Fi usando o protocolo WPA2-Enterprise com PEAP
     WiFi.begin(ssid,    WPA2_AUTH_PEAP,    EAP_IDENTITY,    EAP_USERNAME,
EAP_PASSWORD);
  pinMode(valvA, OUTPUT);  // Define a saída para a válvula A como OUTPUT
   digitalWrite(valvA, LOW);  // Inicialmente, a válvula A está fechada
(LOW)
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
       if  (WiFi.status()  ==  WL_CONNECT_FAILED  ||  WiFi.status()  ==
WL_CONNECTION_LOST) {
      reconnectWiFi();  // Reconecta Wi-Fi em caso de falha
```

```
          }
        }
        Serial.println("");
        Serial.println("WiFi connected");
        Serial.println("IP address:");
        Serial.println(WiFi.localIP());
        client.setServer(mqttServer, mqttPort);  // ConFigura o servidor MQTT
         client.setCallback(callback);  // ConFigura a função de retorno de
chamada MQTT
        reconnectMQTT();  // Reconecta MQTT
      }
      void loop() {
        if (!client.connected()) {
          reconnectMQTT();  // Reconecta MQTT se a conexão foi perdida
        }
        client.loop();  // Mantém a comunicação MQTT ativa
        delay(10);
      }
      // Função para reconectar Wi-Fi
      void reconnectWiFi() {
        Serial.println("WiFi connection lost or failed. Reconnecting...");
        int counter = 0;
        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
          counter++;
          if (counter >= 60) {
            ESP.restart();  // Reinicia o ESP32 após 30 segundos de tentativas
de conexão
          }
        }
        Serial.println("");
        Serial.println("WiFi reconnected");
        Serial.println("IP address:");
        Serial.println(WiFi.localIP());
      }
      // Função para reconectar MQTT
      void reconnectMQTT() {
        while (!client.connected()) {
          Serial.println("Connecting to MQTT broker...");
          if (client.connect("Esp32ValvulaA")) {
            Serial.println("Connected to MQTT broker");
            const char* msgvalA_status = "ESP32_Valvula";
             client.publish("valvulaA/status", msgvalA_status);  // Publica o
status do ESP32 na válvula A
            client.subscribe("valvA/comando");  // Subscreve no tópico MQTT para
receber comandos
          } else {
```

```
            Serial.print("Failed to connect to MQTT broker. Retrying in 5
seconds...");
            delay(5000);
          }
        }
      }
      // Função de retorno de chamada MQTT
      void callback(char* topic, byte* message, unsigned int length) {
        Serial.print("Message arrived on topic: ");
        Serial.print(topic);
        Serial.print(". Message: ");
        for (int i = 0; i < length; i++) {
          Serial.print((char)message[i]);
        }
        Serial.println();
        int command = message[0] - '0';  // Converte o primeiro caractere da
mensagem em um valor inteiro
        if (command == 1) {
          openValve();  // Avança o atuador
        } else if (command == 0) {
          closeValve();  // Recua o atuador
        }
      }
      // Função para abrir a válvula
      void openValve() {
        digitalWrite(valvA, HIGH);  // Define o pino da válvula A como HIGH
(aberto)
      }
      // Função para fechar a válvula
      void closeValve() {
        digitalWrite(valvA, LOW);  // Define o pino da válvula A como LOW
(fechado)
      }
```

APÊNDICE E – Código da ESP32 para a Solução 02 Fim de Curso

```
// Bibliotecas Usadas
#include <PubSubClient.h>  // Biblioteca para comunicação MQTT
#include <WiFi.h>  // Biblioteca para conexão Wi-Fi
#include "esp_wpa2.h"  // Biblioteca para autenticação WPA2-Enterprise

// ConFiguração da conexão wireless do ESP32 no modo PEAP sem autenticação
#define EAP_IDENTITY "xxxx"  // Identidade para autenticação PEAP
#define EAP_USERNAME "xxxx"  // Nome de usuário para autenticação PEAP
#define EAP_PASSWORD "xxxx"  // Senha para autenticação PEAP
const char* ssid = "xxxx";  // SSID da rede Wi-Fi

// Inicialização dos objetos das Classes das bibliotecas utilizadas
WiFiClient espClient;  // Objeto para conexão Wi-Fi
PubSubClient client(espClient);  // Objeto para comunicação MQTT

// ConFiguração das portas utilizadas na ESP32
const int S0 = 2;  // Porta para o sensor fim de curso
const int S1 = 4;  // Porta para o sensor fim de curso

// ConFiguração do servidor broker
const char* mqttServer = "broker.hivemq.com";  // Endereço do servidor
MQTT
const int mqttPort = 1883;  // Porta do servidor MQTT

// Protótipo das funções
void reconnectWiFi();  // Função para reconectar Wi-Fi
void reconnectMQTT();  // Função para reconectar MQTT
void publishSensorState(int sensorValue);  // Função para publicar o
estado do sensor

void setup() {
  Serial.begin(115200);  // Inicia a comunicação serial em 115200 bauds
  delay(10);
  Serial.println();
  Serial.print("Conectando ao WIFI: ");
  Serial.println(ssid);
   WiFi.disconnect(true);  // Desconecta do Wi-Fi para conFigurar nova
conexão Wi-Fi
  WiFi.mode(WIFI_STA);  // Inicializa o modo Wi-Fi

  // Conexão à rede Wi-Fi usando o protocolo WPA2-Enterprise com PEAP
    WiFi.begin(ssid,    WPA2_AUTH_PEAP,    EAP_IDENTITY,    EAP_USERNAME,
EAP_PASSWORD);


   pinMode(S0, INPUT_PULLUP);  // Define o sensor fim de curso S0 como
INPUT_PULLUP
```

```
        pinMode(S1, INPUT_PULLUP);  // Define o sensor fim de curso S1 como
INPUT_PULLUP

        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
            if  (WiFi.status()  ==  WL_CONNECT_FAILED  ||  WiFi.status()  ==
WL_CONNECTION_LOST) {
            reconnectWiFi();  // Reconecta Wi-Fi em caso de falha
          }
        }

        Serial.println("");
        Serial.println("ESP32 conectado");
        Serial.println("IP:");
        Serial.println(WiFi.localIP());

        client.setServer(mqttServer, mqttPort);  // ConFigura o servidor MQTT
        reconnectMQTT();  // Reconecta MQTT
      }

      void loop(){
       if (!client.connected()) {
          reconnectMQTT();  // Reconecta MQTT se a conexão foi perdida
        }

        client.loop();  // Mantém a comunicação MQTT ativa

        //variavel auxilar para tratar dados do sensores
         int sensorValue = digitalRead(S1);  // Lê o estado do sensor fim de
curso S1
        publishSensorState(sensorValue);  // Publica o estado do sensor

        delay(1000); // intervalo de 1 segundo entre as leituras
      }

      // Função para reconectar Wi-Fi
      void reconnectWiFi() {
        Serial.println("WiFi: Falha na conexão. Reconectando a ESP32...");

        int counter = 0;
        while (WiFi.status() != WL_CONNECTED) {
          delay(500);
          Serial.print(".");
          counter++;

          if (counter >= 60) {
```

```
            ESP.restart();  // Reinicia o ESP32 após 30 segundos de tentativas
de conexão
          }
        }

        Serial.println("");
        Serial.println("ESP32 reconectado no WIFI");
        Serial.println("IP address:");
        Serial.println(WiFi.localIP());
      }

      // Função para reconectar MQTT
      void reconnectMQTT() {
        while (!client.connected()) {
          Serial.println("Conectando ao Servidor MQTT broker...");

          if (client.connect("Esp32Sensores")) {
            Serial.println("Conectado ao Servidor MQTT broker");

            const char* msgvalA_status = "ESP32_Sensores Conectado";
            client.publish("Sensores_A/status", msgvalA_status);  // Publica o
status do ESP32

             //client.subscribe("valvA/comando");  // Subscreve no tópico MQTT
para receber comandos
          } else {
              Serial.print("Falha  na  conexão  com  o  Servidor  MQTT  broker.
Reconectando em 5 seconds...");
            delay(5000);
          }
        }
      }

      // Função para publicar o estado do sensor
      void publishSensorState(int sensorValue) {
        if (sensorValue == LOW) {
          //Serial.println("Atuador A Avançado");
          client.publish("Sensores/sensorA0A1", "Atuador A Avançado");
          Serial.println("\n Atuador A Avançado \n");
        } else {
          //Serial.println("Atuador A Recuado");
          client.publish("Sensores/sensorA0A1", "Atuador A Recuado");
          Serial.println("\n Atuador A Recuado \n");
        }
      }
```