
UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO (EMC)
FACULDADE DE ENGENHARIA MECÂNICA

GABRIEL LATALISA FERNANDES DE ARAÚJO

# Desenvolvimento de uma aplicação IoT para bancada didática de eletropneumática

Goiânia

2025

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

# TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

**1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)**

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): Gabriel Latalisa Fernandes de Araújo

Título do trabalho: Desenvolvimento de uma aplicação IoT para bancada didática de eletropneumática

**2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento** [ X ] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

**Casos de embargo:**

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

**Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.**

Documento assinado eletronicamente por **Joao Paulo Da Silva Fonseca**, **Professor do Magistério Superior**, em 10/12/2025, às 13:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gabriel Latalisa Fernandes De Araujo**, **Discente**, em 11/12/2025, às 23:16, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

GABRIEL LATALISA FERNANDES DE ARAÚJO

# Desenvolvimento de uma aplicação IoT para bancada didática de eletropneumática

Projeto Final de Curso apresentado à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás como requisito parcial para obtenção do título de bacharel em Engenharia Mecânica.

Área de concentração: Engenharia Mecânica

Orientador: João Paulo da Silva Fonseca

Goiânia

2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Araújo, Gabriel Latalisa Fernandes de
Desenvolvimento de uma aplicação IoT para bancada didática de eletropneumática [manuscrito] / Gabriel Latalisa Fernandes de Araújo. - 2025.
LXII, 62 f.: il.

Orientador: Prof. João Paulo da Silva Fonseca.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Escola de Engenharia Elétrica, Mecânica e de Computação (EMC), Engenharia Mecânica, Goiânia, 2025.

Inclui siglas, abreviaturas, tabelas, lista de figuras, lista de tabelas.

1. Indústria 4.0. 2. IIoT. 3. MQTT. 4. Automação Eletropneumática. 5. ESP8266. I. Fonseca, João Paulo da Silva, orient. II. Título.

CDU 621.3

UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

# ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO

Ao(s) dez dia(s) do mês de dezembro do ano de 2025 iniciou-se a sessão pública de defesa do Trabalho de Conclusão de Curso (TCC) intitulado “Desenvolvimento de uma aplicação IoT para bancada didática de eletropneumática”, de autoria de Gabriel Latalisa Fernandes de Araújo, do curso de Engenharia Mecânica, da Escola de Engenharia Elétrica, Mecânica e de Computação da UFG. Os trabalhos foram instalados pelo Prof. Dr. João Paulo da Silva Fonseca - orientador (EMC/UFG) com a participação dos demais membros da Banca Examinadora: Dr. Gustavo Souto de Sá e Souza (EMC/UFG) e Engenheiro Mecânico e Mestrando Gabriel Alves Costa (PPGMEC/EMC/UFG). Após a apresentação, a banca examinadora realizou a arguição do(a) estudante. Posteriormente, de forma reservada, a Banca Examinadora atribuiu a nota final de 9,9 , tendo sido o TCC considerado aprovado.

Proclamados os resultados, os trabalhos foram encerrados e, para constar, lavrou-se a presente ata que segue assinada pelos Membros da Banca Examinadora.

Documento assinado eletronicamente por **Joao Paulo Da Silva Fonseca**, **Professor do Magistério Superior**, em 10/12/2025, às 15:09, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gustavo Souto De Sa E Souza**, **Técnico de Laboratório**, em 10/12/2025, às 15:10, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Gabriel Alves Costa**, **Discente**, em 10/12/2025, às 15:12, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5850259** e o código CRC **89DE4B88**.

**Referência:** Processo nº 23070.042813/2025-13 SEI nº 5850259

## AGRADECIMENTOS

Agradeço ao meu orientador, Professor João Paulo da Silva Fonseca, pela oportunidade, pela paciência e pela orientação segura ao longo do desenvolvimento deste trabalho. Seu conhecimento técnico e disponibilidade foram fundamentais para a conclusão deste projeto.

Aos meus pais, minha eterna gratidão. Obrigado por serem minha base, pelo incentivo constante aos meus estudos e por todo o sacrifício e amor dedicados a mim, que tornaram essa conquista possível.

À minha parceira, agradeço imensamente pelo companheirismo, pelo apoio incondicional e pela compreensão nos momentos de ausência dedicados a este trabalho. Seu incentivo foi meu combustível nas etapas mais difíceis.

Aos meus amigos, obrigado pela parceria, pelas trocas de conhecimento e pelos momentos de descontração que tornaram a jornada de desenvolvimento do projeto mais leve.

*"Qualquer tecnologia suficientemente avançada é indistinguível da magia."*

(Arthur C. Clarke)

## RESUMO

A evolução da Indústria 4.0 e da Internet Industrial das Coisas (IIoT) tem transformado os ambientes de manufatura e ensino, demandando soluções de conectividade eficientes, flexíveis e acessíveis. Neste contexto, o protocolo *Message Queuing Telemetry Transport* (MQTT) destaca-se como uma ferramenta fundamental por viabilizar a comunicação em tempo real e com baixo consumo de banda entre dispositivos diferentes. Este trabalho apresenta o desenvolvimento de um servidor web para o controle e monitoramento remoto de uma bancada eletropneumática, visando a modernização das práticas laboratoriais da disciplina de Sistemas de Acionamentos Hidráulicos e Pneumáticos. A arquitetura do sistema é composta por uma placa de desenvolvimento NodeMCU ESP8266, responsável pelo acionamento das válvulas e leitura dos sensores da bancada; um microcomputador de placa única Raspberry Pi, que atua como servidor central hospedando a aplicação desenvolvida em Python (Flask) e o broker MQTT; uma interface gráfica responsiva que permite a operação nos modos manual e sequencial. Os resultados obtidos validam a aplicação de tecnologias de *hardware* e protocolos abertos na criação de ferramentas didáticas robustas, demonstrando a viabilidade técnica do controle remoto e do monitoramento via rede de comunicação.



**Palavras-chave:** Indústria 4.0; IIoT; MQTT; Automação Eletropneumática; Servidor; ESP8266.

## ABSTRACT

The evolution of Industry 4.0 and the Industrial Internet of Things (IIoT) has transformed manufacturing and educational environments, demanding efficient, flexible, and accessible connectivity solutions. In this context, the Message Queuing Telemetry Transport (MQTT) protocol stands out as a fundamental tool for enabling real-time communication with low bandwidth consumption between different devices. This work presents the development of a web server for the remote control and monitoring of an electropneumatic test bench, aiming at the modernization of laboratory practices in the Hydraulic and Pneumatic Drive Systems course. The system architecture is composed of a NodeMCU ESP8266 development board, responsible for valve actuation and sensor reading on the bench; a Raspberry Pi single-board computer, acting as the central server hosting the application developed in Python (Flask) and the MQTT broker; and a responsive graphical interface that allows operation in manual and sequential modes. The obtained results validate the application of open hardware technologies and protocols in the creation of robust educational tools, demonstrating the technical feasibility of remote control and monitoring via a communication network.



**Keywords:** Industry 4.0; IIoT; MQTT; Electro-pneumatic Automation; Server; ESP8266.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface* (Interface de Programação de Aplicações)

CSS – *Cascading Style Sheets*

DDNS – *Dynamic Domain Name System*

DHCP – *Dynamic Host Configuration Protocol*

ESP – *Espressif Systems* (referência à família de microcontroladores)

GPIO – *General Purpose Input/Output* (Entrada/Saída de Propósito Geral)

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IHM – Interface Homem-Máquina

IIoT – *Industrial Internet of Things* (Internet Industrial das Coisas)

IoT – *Internet of Things* (Internet das Coisas)

JSON – *JavaScript Object Notation*

LED – *Light Emitting Diode*

MQTT – *Message Queuing Telemetry Transport*

QoS – *Quality of Service*

REST – *Representational State Transfer*

SSH – *Secure Shell*

TCP/IP – *Transmission Control Protocol/Internet Protocol*

URL – *Uniform Resource Locator*

USB – *Universal Serial Bus*

Wi-Fi – *Wireless Fidelity*

**SUMÁRIO**

# 1. INTRODUÇÃO

A quarta revolução industrial, ou Indústria 4.0, inaugurou uma nova era nos processos produtivos, caracterizada pela fusão de tecnologias embarcadas e sistemas ciber-físicos para a criação de fábricas inteligentes. Ao integrar dispositivos conectados em rede, esse paradigma rompe com a lógica tradicional de controle centralizado, fundamentando-se no conceito de Internet das Coisas (IoT) aplicada ao ambiente fabril (GILCHRIST, 2016).

Nesse contexto, a Internet Industrial das Coisas (IIoT) surge como um vetor de transformação, permitindo conectar máquinas, sensores e sistemas de gestão em tempo real. Para viabilizar essa interconexão, o protocolo de comunicação *Message Queuing Telemetry Transport* (MQTT) desempenha um papel central. Conforme a norma oficial da OASIS (214) e as definições de Hanes et al. (2017), o MQTT oferece uma solução leve e eficiente para a troca de dados entre dispositivos com recursos limitados, garantindo robustez e escalabilidade às aplicações industriais.

Diante desse avanço tecnológico, torna-se obrigatório que a formação em engenharia acompanhe tal evolução. O ensino de sistemas e acionamentos eletropneumáticos, tradicionalmente focado em lógica de contato e controladores isolados, precisa incorporar as tecnologias de conectividade que o futuro engenheiro encontrará no mercado de trabalho. A modernização dos laboratórios de ensino, através da implementação de bancadas didáticas conectadas e acessíveis remotamente, apresenta-se como uma estratégia eficaz para qualificar o aprendizado e demonstrar na prática os conceitos de telemetria e teleoperação.

Este projeto propõe o desenvolvimento de um servidor web para o controle e monitoramento de uma bancada eletropneumática, utilizando uma arquitetura baseada em *hardware* de baixo custo e *software open source*. A solução integra placas de desenvolvimento NodeMCU e microcontroladores ESP8266 e um microcomputador Raspberry Pi para criar uma aplicação de IoT completa, demonstrando como protocolos modernos podem ser aplicados para modernizar equipamentos legados e enriquecer a experiência educacional em cursos de engenharia e tecnologia.

## 1.1. OBJETIVOS

### 1.1.1. OBJETIVO GERAL

O objetivo geral desse Projeto Final de Curso (PFC) é desenvolver um sistema web para controle e monitoramento de bancadas didáticas de eletropneumática, aplicando o conceito de IoT.

### 1.1.2. OBJETIVOS ESPECÍFICOS

Além do objetivo geral do projeto, foram estabelecidos alguns objetivos específicos como o estudo e a implementação do protocolo de comunicação MQTT, responsável pela comunicação entre a aplicação web e o microcontrolador ESP8266, conectado à bancada; a configuração do Raspberry Pi 2 como servidor central, hospedando o sistema web e o broker local MQTT, funcionando na rede local da Escola de Engenharia Elétrica, Mecânica e de Computação dentro da Universidade Federal de Goiás (UFG); programação e configuração do microcontrolador ESP8266, para o controle de atuadores pneumáticos e leitura dos sensores de fim de curso; desenvolvimento de uma interface web responsiva, responsável pela origem dos comandos de acionamento do microcontrolador e apresentação de sinais de status do mesmo; estudo e implementação de uma lógica de controle sequencial utilizando a lógica de malha fechada; e demonstrar a escalabilidade das soluções encontradas.

## 2. REVISÃO BIBLIOGRÁFICA

Esta seção fundamenta os conceitos teóricos essenciais para a compreensão do projeto, abordando desde os princípios físicos da pneumática até os componentes de controle e sensoriamento utilizados na automação moderna.

### 2.1. SISTEMAS ELETROPNEUMÁTICOS

A pneumática moderna é definida como o ramo da tecnologia que estuda o movimento e os fenômenos dos gases, sendo aplicada industrialmente através do uso de ar comprimido para a transmissão de energia e realização de trabalho mecânico (PARKER, 2011). Nos sistemas modernos, a união da pneumática com a eletrônica resultou na eletropneumática, uma área que integra a força motriz dos atuadores pneumáticos com a flexibilidade e precisão de controle dos componentes elétricos (FIALHO, 2011).

A automação pneumática oferece vantagens significativas, como a simplicidade dos elementos de comando, a robustez dos componentes e a segurança em ambientes perigosos,

visto que o ar não gera faíscas. No entanto, a integração com a parte elétrica permite superar limitações da pneumática pura, facilitando o processamento de sinais complexos e a comunicação entre máquinas (BONACORSO; NOLL, 2013).

A estrutura básica de um sistema eletropneumático consiste na produção e preparação do ar comprimido, seguida de sua distribuição para os elementos de trabalho (atuadores), que são governados por elementos de sinal (sensores e botoeiras) e de comando (válvulas solenoides) (NATALE, 2011).

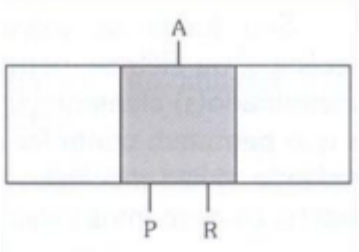
### 2.1.1. VÁLVULAS DE COMANDO

As válvulas direcionais são os elementos responsáveis por controlar o início, a parada e a direção do fluxo de ar comprimido dentro de um circuito. Elas funcionam como as chaves de um circuito elétrico, desviando o ar para diferentes vias de trabalho conforme a necessidade do processo (NATALE, 2011).

Conforme descrevem Bonacorso e Noll (2013), a classificação das válvulas direcionais baseia-se em quatro critérios principais: o número de posições de comutação, o número de vias de fluxo, o tipo de acionamento e o tipo de retorno.

As válvulas necessitam representações em projetos eletropneumáticos, com o intuito de simular o funcionamento interno das mesmas. Tais representações acontecem através de retângulos, que são divididos em quadrados. O número de posições é referente ao número de quadrados justapostos. Pequenos traços colocados de fora do retângulo representam os orifícios (FIALHO, 2011). A Figura 1 demonstra essa representação.

Figura 1 - Representação do número de posições e orifícios

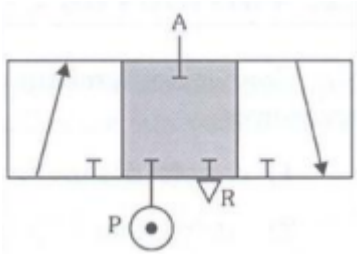



Fonte: Fialho (2012, p. 110)

As vias ou ligações são indicadas por setas ligando os orifícios. Os orifícios fechados são indicados através de um traço curto transversal, formando um T, na parte interna do quadrado. Para reconhecer outras posições, além da normal, o quadrado é deslocado para que

os orifícios fiquem sobre o retângulo apropriado. Além disso, normalmente o conduto de ar comprimido é representado por um pequeno círculo marcado internamente por outro menor e cheio. Já o conduto para a atmosfera tem um pequeno triângulo representando a via de exaustão (PARKER, 2011). A Figura 2 mostra a representação do número de vias e dos condutos de ar.

Figura 2 - Representação do número de vias e dos condutos de ar




Fonte: Fialho (2012, p. 110)

As letras presentes na Figura 2 são formas literais para identificação das vias, pela norma DIN 24300. As letras que a norma define e seus significados estão apresentados abaixo (PARKER, 2011):

- **A, B, C** – linha de trabalho (utilização);
- **P** – Conexão de pressão (alimentação);
- **R, S, T** – escape ao exterior do ar comprimido utilizado pelos equipamentos pneumáticos (escape, exaustão);
- **L** – drenagem de líquido;
- **X, Y, Z** – linha para transmissão da energia de comando (linhas de pilotagem).

Para o funcionamento das válvulas, é necessário um acionamento, para que ocorra a mudança de posição interna. O acionamento pode ser muscular, mecânico, pneumático ou elétrico. Em sistemas eletropneumáticos, as válvulas mais comuns utilizam acionamento por solenoide, onde uma bobina eletromagnética desloca o carretel interno da válvula ao ser energizada, alterando o caminho do ar (BONACORSO; NOLL, 2013).

Uma válvula amplamente utilizada é a direcional 5/2 vias. Ela possui cinco conexões (vias) e duas posições de trabalho. Em uma posição, ela direciona o ar para avançar um atuador e libera o escape da câmara oposta; na outra posição, ela inverte o fluxo, recuando o atuador (FIALHO, 2011).

### 2.1.2. ATUADORES PNEUMÁTICOS

Os atuadores são os dispositivos que convertem a energia potencial do ar comprimido em energia cinética, realizando o trabalho mecânico final do sistema. Eles podem gerar movimentos lineares, rotativos ou oscilantes (PARKER, 2011).

Os atuadores lineares, comumente denominados cilindros, dividem-se em dois grupos principais: cilindro de simples ação e cilindro de dupla ação.

A tabela 1 apresenta um resumo sobre os principais tipos de cilindros.

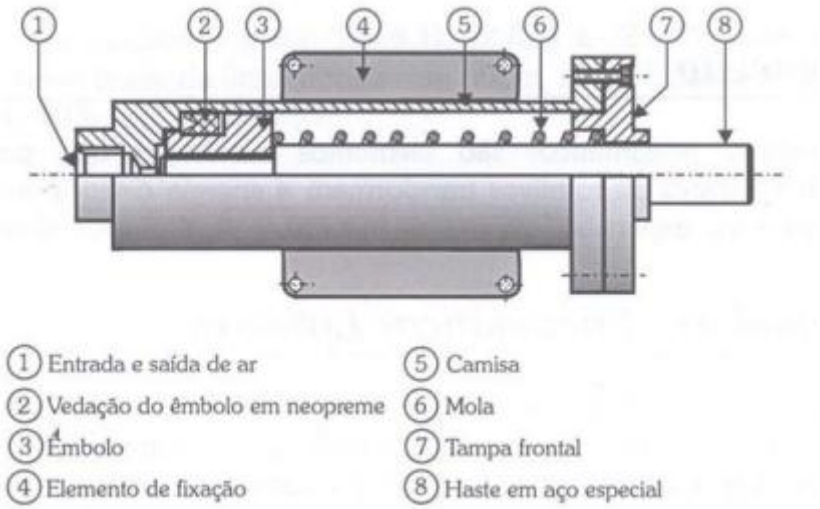
Tabela 1 - Principais tipos de cilindro

| | | |
|---|---|---|
| Cilindros de ação simples | De êmbolo e haste | Retorno por mola ou por força externa |
| | De membrana e haste | |
| | De membrana | |
| Cilindros de ação dupla | Com haste | Simples |
| | | Dupla |
| | Sem haste | De cabo |
| | | De cinta (ou tira) |
| | | Magnético |

Fonte: Bollmann (1997, p. 65)

Os atuadores de simples ação podem realizar trabalho em apenas um sentido de movimento. O avanço ocorre pela ação do ar comprimido, enquanto o retorno é feito por uma força externa ou por uma mola interna incorporada ao cilindro (FIALHO, 2011). A Figura 3 mostra o desenho de uma representação em corte do atuador de simples ação indicando os seus principais componentes.

Figura 3 - Atuador de simples ação




Fonte: Fialho (2012, p. 78)

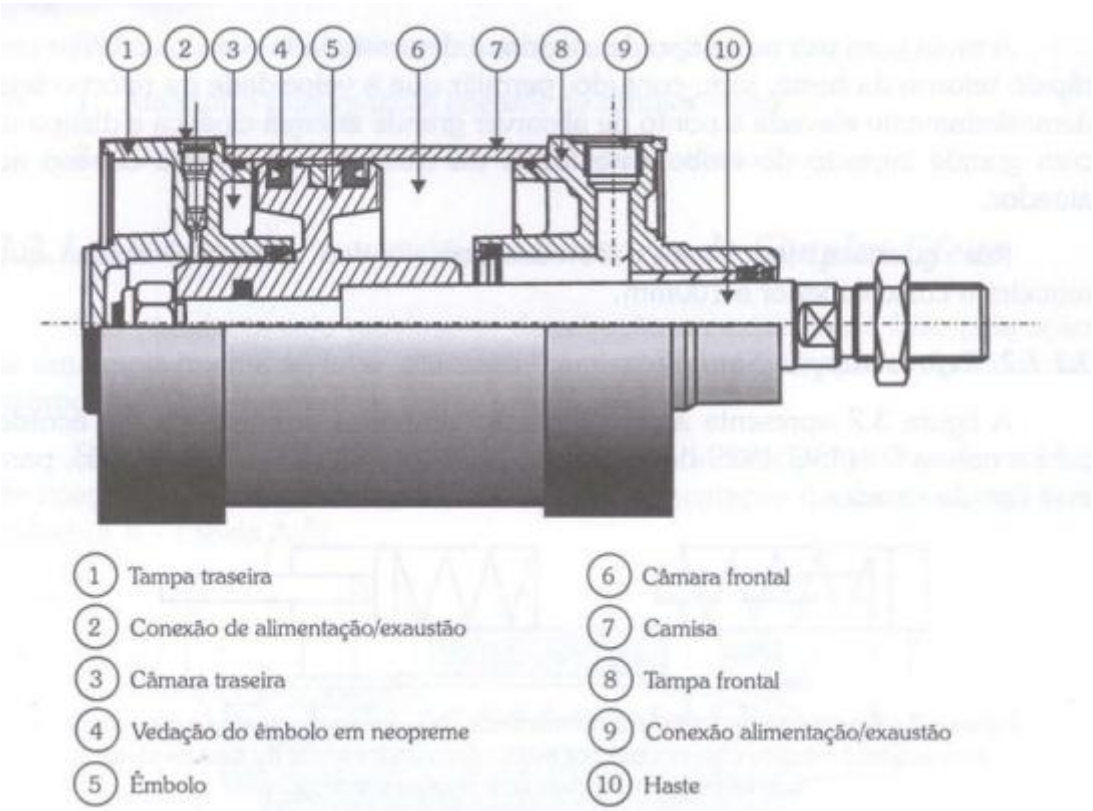
A Figura 4 demonstra a representação simbólica para esse tipo de atuador.

Figura 4 - Representação simbólica para atuador pneumático de simples ação

Fonte: Fialho (2012, p. 79)

Os atuadores de dupla ação realizam trabalho em ambos os sentidos de movimento (avanço e retorno). O ar comprimido é alternado entre as câmaras dianteira e traseira do cilindro para movê-lo em ambas as direções, oferecendo maior controle e força em todo o curso (PARKER, 2011). A Figura 5 mostra o desenho de uma representação em corte do atuador de simples ação indicando os seus principais componentes.

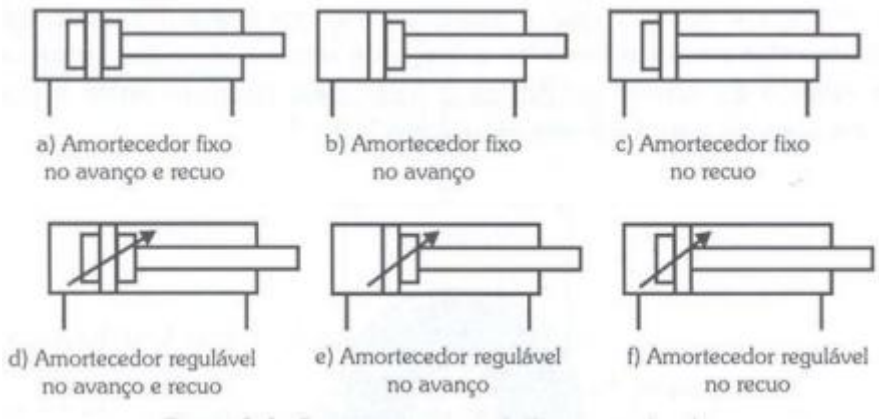
Figura 5 - Atuador de dupla ação




Fonte: Fialho (2012, p. 80)

A Figura 6 demonstra a representação simbólica para esse tipo de atuador.

Figura 6 - Representação simbólica para atuador pneumático de dupla ação




Fonte: Fialho (2012, p. 83)

### 2.1.3. SENSORES

Os sensores de posição são elementos vitais em sistemas de automação, responsáveis por detectar a presença, ausência ou o deslocamento de objetos. Esses dispositivos convertem

uma grandeza física (a posição mecânica) em um sinal elétrico processável pelo controlador (BALBINOT; BRUSAMARELLO, 2011). Para a aplicação em bancadas eletropneumáticas, destacam-se algumas tecnologias de detecção.

#### 2.1.3.1. SENSOR DE FIM DE CURSO ELETROMECÂNICO (ROLETE)

O sensor de fim de curso, ou chave limitadora (*limit switch*), é o dispositivo mais tradicional na indústria. Seu funcionamento baseia-se no contato físico direto. O dispositivo é composto por um atuador mecânico (geralmente um rolete ou alavanca) acoplado a um conjunto de contatos elétricos (THOMAZINI; ALBURQUERQUE, 2011).

Quando a haste de um cilindro pneumático se move e atinge o rolete, a força mecânica desloca o mecanismo interno, comutando os contatos (abrindo um circuito Normalmente Fechado ou fechando um Normalmente Aberto). Apesar de sofrerem desgaste mecânico devido ao contato físico, esses sensores são extremamente robustos a interferências elétricas e simples de implementar, dispensando fontes de alimentação externas para o elemento sensor (THOMAZINI; ALBURQUERQUE, 2011).

#### 2.1.3.2. SENSOR DE PROXIMIDADE INDUTIVO

Os sensores indutivos são dispositivos eletrônicos de estado sólido projetados para a detecção de objetos metálicos sem contato físico. Seu princípio de funcionamento baseia-se na geração de um campo eletromagnético oscilante de alta frequência na face sensora do dispositivo (ROSÁRIO, 2005).

Quando um alvo metálico penetra neste campo, correntes parasitas (Correntes de Foucault) são induzidas na superfície do metal, drenando energia do oscilador interno do sensor e reduzindo a amplitude da oscilação. Um circuito de disparo (*Schmitt Trigger*) detecta essa atenuação e altera o estado da saída lógica. São amplamente utilizados na indústria por sua imunidade a contaminantes não metálicos, como óleo, água e poeira, sendo ideais para ambientes hostis (THOMAZINI; ALBURQUERQUE, 2011).

#### 2.1.3.3. SENSOR DE PROXIMIDADE CAPACITIVO

Diferente dos indutivos, os sensores capacitivos são capazes de detectar tanto materiais condutores (metais) quanto materiais dielétricos (plásticos, vidro, líquidos, madeira). Eles

operam baseados na variação da capacitância entre o eletrodo ativo do sensor e o terra (THOMAZINI; ALBURQUERQUE, 2011).

O sensor gera um campo eletrostático em sua face. A aproximação de qualquer objeto altera a constante dielétrica do meio, aumentando a capacitância do sistema. Quando essa capacitância excede um limiar pré-definido, o oscilador interno entra em ressonância e a saída é ativada. Sua principal vantagem é a versatilidade, permitindo, por exemplo, detectar a presença de fluidos dentro de tubulações plásticas ou a contagem de peças não metálicas em esteiras transportadoras (THOMAZINI; ALBURQUERQUE, 2011).

#### 2.1.3.4. SENSOR DE POSIÇÃO ÓPTICO (FOTOELÉTRICO)

Os sensores ópticos utilizam a propagação da luz (visível ou infravermelha) para detectar objetos. São compostos fundamentalmente por um emissor (LED) e um receptor (fototransistor) (PAHC AUTOMAÇÃO, 2021). Existem três configurações principais de operação:

1. **Barreira (*Through-beam*):** Emissor e receptor estão alinhados frente a frente. A detecção ocorre quando o objeto interrompe o feixe de luz. Oferece o maior alcance e confiabilidade.
2. **Retroreflexivo:** Emissor e receptor estão no mesmo corpo, e um espelho prismático reflete o feixe de volta. O objeto é detectado ao bloquear esse retorno.
3. **Difuso:** O sensor emite luz que é refletida pelo próprio objeto de volta ao receptor. O alcance depende da cor e textura do objeto.

Sua principal vantagem é o longo alcance de detecção e a capacidade de detectar objetos de qualquer material, desde que não sejam transparentes (no caso dos tipos barreira/reflexivo) (THOMAZINI; ALBURQUERQUE, 2011).

#### 2.1.3.5. DETECTORES DE POSIÇÃO MAGNÉTICOS

Estes sensores são acionados pela presença de um campo magnético externo, geralmente proveniente de um ímã permanente instalado no êmbolo de cilindros pneumáticos. Isso permite detectar a posição do atuador através da parede de alumínio ou aço inox, sem necessidade de contato mecânico com a haste (TECNOTRON, 2025).

Destacam-se dois tipos principais:

1. **Reed Switch (Ampola de Vidro):** É um dispositivo eletromecânico simples, composto por duas lâminas ferromagnéticas flexíveis encapsuladas hermeticamente em uma ampola de vidro com gás inerte. Na presença de um campo magnético, as lâminas se atraem e fecham o contato elétrico. É passivo, de baixo custo e amplamente utilizado em cilindros comerciais, embora tenha vida útil limitada pelo desgaste mecânico das lâminas.
2. **Sensor de Efeito Hall:** É um dispositivo eletrônico de estado sólido (sem partes móveis). Baseia-se no princípio físico descoberto por Edwin Hall, onde uma diferença de potencial (tensão Hall) é gerada transversalmente em um condutor percorrido por corrente quando submetido a um campo magnético perpendicular. Ao detectar o ímã do cilindro, o circuito eletrônico comuta uma saída a transistor. Por não possuir contatos móveis, apresenta durabilidade virtualmente ilimitada e é imune a vibrações que poderiam causar falsos disparos no Reed Switch (THOMAZINI; ALBURQUERQUE, 2011).

## 2.2. PROTOCOLO MQTT

O MQTT (*Message Queuing Telemetry Transport*) é um protocolo de comunicação de mensagens leve, criado sobre a pilha TCP/IP, otimizado para redes com largura de banda limitada ou alta latência. Segundo Hanes et al. (2017), ele se tornou o padrão de fato para a Internet das Coisas (IoT) devido à sua eficiência e simplicidade.

Sua arquitetura difere do modelo tradicional cliente-servidor (como o HTTP). No MQTT, a comunicação é desacoplada e baseada em eventos, utilizando um modelo de Publicação/Subscrição (*Publish/Subscribe*) (OASIS, 2014). De acordo com Gilchrist (2016), nessa arquitetura existem três componentes fundamentais:

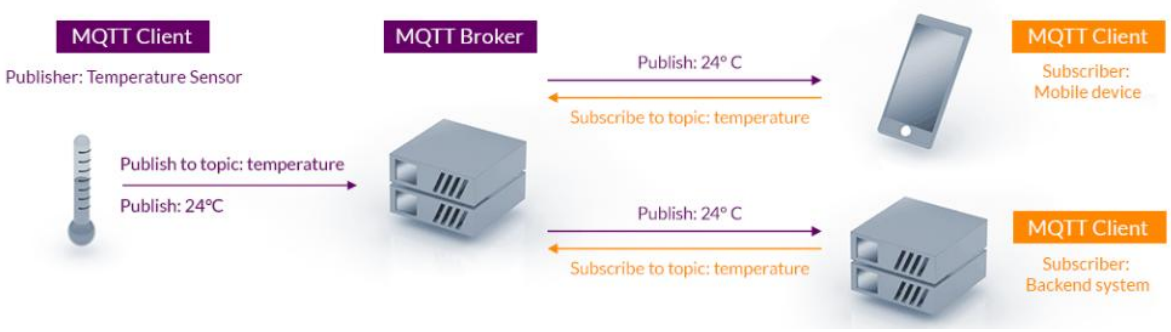
- **Publicador (*Publisher*):** O dispositivo que gera e envia os dados.
- **Subscritor (*Subscriber*):** O dispositivo interessado em receber os dados.
- **Broker:** O servidor central que gerencia o tráfego. Ele recebe as mensagens dos publicadores, filtra com base nos tópicos e as distribui para os assinantes corretos.

A organização dos dados é feita através de Tópicos (*Topics*), que são estruturados hierarquicamente, com níveis separados por barras (/). Por exemplo, um tópico como “atuadores/acao/avancar” indica o acionamento da ação de avançar de um atuador. Além disso, Hanes et al. (2017) destacam a importância dos níveis de QoS (*Quality of Service*), que definem a garantia de entrega:

- **QoS 0:** A mensagem é enviada uma vez, sem garantia de recebimento ("*fire and Forget*").
- **QoS 1:** Garante que a mensagem chegue ao menos uma vez (pode haver duplicidade).
- **QoS 2:** Garante que a mensagem chegue exatamente uma vez (mais lento, porém mais confiável).

A Figura 7 demonstra a arquitetura do protocolo MQTT, com ênfase no modelo de publicação e subscrição.

Figura 7 - Modelo de publicação e subscrição no MQTT




Fonte: https://mqtt.org/

## 2.3. PLATAFORMAS DE HARDWARE

Para viabilizar a comunicação entre o mundo físico e o digital, são necessários dispositivos embarcados capazes de processar lógica local e conectar-se à rede.

### 2.3.1. MICROCONTROLADOR ESP8266

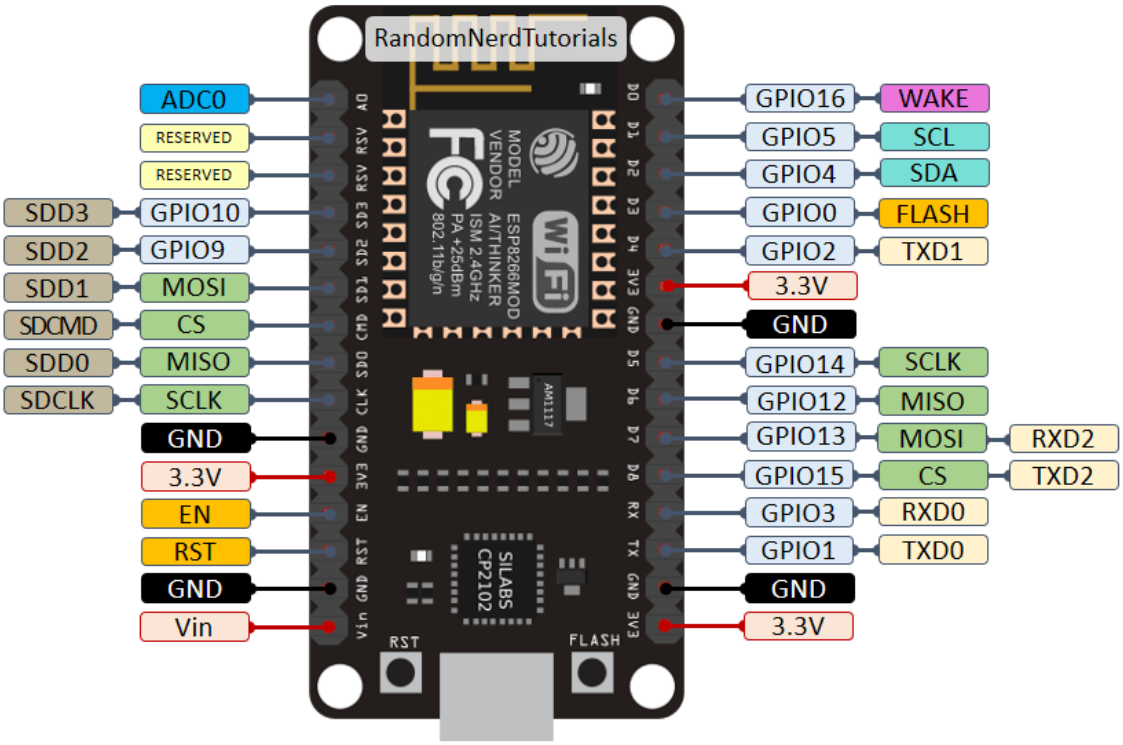
A placa de desenvolvimento NodeMCU ESP8266 é um *System-on-Chip* (SoC) de baixo custo desenvolvido pela Espressif Systems, que integra um microcontrolador de 32 bits (Tensilica L106) e um transceptor Wi-Fi completo (padrão 802.11 b/g/n). Sua principal vantagem é a capacidade de operar de forma autônoma (*standalone*), executando aplicações diretamente em sua memória flash, sem a necessidade de um microcontrolador externo.

Embora o chip ESP8266 seja o componente central, em projetos de prototipagem é comum o uso de placas de desenvolvimento como o NodeMCU. Tal placa facilita o uso do ESP8266 ao integrar um conversor USB-Serial e um regulador de tensão na mesma placa,

permitindo a programação direta via porta USB do computador sem a necessidade de conversores externos.

Uma característica fundamental do ESP8266 para automação é a multiplexação de seus pinos. A maioria dos pinos GPIO (*General Purpose Input/Output*) possui múltiplas funções. Além de operarem como entradas e saídas digitais simples, eles podem ser configurados para interfaces de comunicação serial como I2C (*Inter-Integrated Circuit*), SPI (*Serial Peripheral Interface*) e UART (*Universal Asynchronous Receiver-Transmitter*). A Figura 8 mostra um modelo de ESP8266 e sua pinagem.

Figura 8 - ESP8266 e sua pinagem




Fonte: https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

### 2.3.2. MICROCOMPUTADOR RASPBERRY

O Raspberry Pi é uma série de computadores de placa única (*Single Board Computers – SBC*) desenvolvidos pela Raspberry Pi Foundation, projetados para promover o ensino de ciência da computação e facilitar a prototipagem de sistemas embarcados. No desenvolvimento deste projeto, utilizou-se especificamente o Raspberry Pi 2 Model B, uma plataforma que

marcou um avanço significativo em relação aos seus antecessores ao introduzir a arquitetura *quad-core* (UPTON; HALFACREE, 2014).

O coração desta placa é o *System-on-Chip* (SoC) Broadcom BCM2836. Diferente das gerações anteriores baseadas na arquitetura ARMv6, o Pi 2 utiliza um processador ARM Cortex-A7 de quatro núcleos (*quad-core*) operando a 900 MHz. Esta mudança de arquitetura para ARMv7 não apenas aumentou a capacidade de processamento em aproximadamente seis vezes em comparação ao Modelo B original, mas também ampliou a compatibilidade com uma gama maior de distribuições Linux, incluindo versões robustas do Ubuntu e do Debian (ADAFRUIT, 2015).

Além do processador, o dispositivo conta com 1GB e memória RAM LPDDR2, o que é essencial para executar aplicações de servidor multitarefa sem gargalos de desempenho. Em termos de conectividade e interfaces, o Raspberry Pi 2 Model B oferece:

- 4 portas USB 2.0, permitindo a conexão de periféricos;
- Porta Ethernet 10/100 BaseT, fundamental para a conexão de rede estável necessária em aplicações de servidor;
- Conector GPIO de 40 pinos, que mantém a compatibilidade com sensores e módulos de expansão padrão da indústria.

No contexto de Internet das Coisas (IoT) e deste projeto, o Raspberry Pi 2 desempenha o papel de Gateway e Servidor Central. Sua capacidade de processamento é mais do que suficiente para hospedar o sistema operacional Raspberry Pi OS (baseado em Linux), executar o broker de mensagens Mosquitto e servir a aplicação web desenvolvida em Python/Flask.

Embora modelos mais recentes (como o Pi 3 ou 4) ofereçam Wi-Fi integrado e maior velocidade, o Raspberry Pi 2 continua sendo uma escolha extremamente eficiente e estável para aplicações de automação que operam via conexão cabeada (Ethernet), oferecendo baixo consumo de energia e robustez para operar continuamente como um servidor de borda (*edge server*) (UPTON; HALFACREE, 2014).

A Figura 9 mostra o microcomputador de placa única, Raspberry Pi 2 Model B.

Figura 9 - Raspberry Pi 2 Model B

Fonte: https://www.raspberrypi.com/products/raspberry-pi-2-model-b/

## 2.4. DESENVOLVIMENTO WEB E INTERFACES

A interface Homem-Máquina (IHM) evoluiu significantemente com o advento da Indústria 4.0. Sistemas modernos abandonam painéis de controle físicos e rígidos em favor de aplicações web acessíveis via navegador, permitindo o monitoramento e controle de processos industriais a partir de qualquer dispositivo conectado à rede, como tablets, smartphones e computadores. Para viabilizar essa flexibilidade, utiliza-se um conjunto de tecnologias de *Backend* (servidor) e *Frontend* (interface) integradas.

### 2.4.1. FRAMEWORK FLASK E ARQUITETURA BACKEND

Para o desenvolvimento do servidor de aplicação hospedado no Raspberry Pi, optou-se pelo uso do Flask. O Flask é um *microframework* para Python baseado na especificação WSGI (*Web Server Gateway Interface*). A classificação “micro” refere-se à sua filosofia de manter o núcleo simples e extensível, sem impor decisões de arquitetura ou dependências de banco de dados desnecessárias, o que o torna ideal para aplicações embarcadas onde recursos de memória e processamento são limitados.

O Flask opera através de duas dependências principais: o kit de ferramentas Werkzeug para o roteamento de requisições HTTP e o motor de modelos Jinja2 para a renderização de páginas HTML dinâmicas. No contexto deste projeto, o Flask desempenha três funções críticas:

- **Gerenciamento de Rotas (Routing):** Mapeia URLs específicas (ex: /api/atuador/comando) para funções Python que executam a lógica e controle.
- **API RESTful:** Expõe *endpoints* que recebem comandos via requisições HTTP (POST/GET) contendo cargas de dados em formato JSON (*JavaScript Object Notation*).
- **Serviço de Arquivos Estáticos:** Entrega ao navegador os arquivos de *Frontend* (HTML, CSS, Imagens e Scripts) necessários para a renderização da interface do usuário.

### 2.4.2. COMUNICAÇÃO EM TEMPO REAL COM SOCKET.IO

Em sistemas de supervisão industrial, a latência na atualização das informações é um fator crítico. O protocolo HTTP padrão segue um modelo de "requisição-resposta", onde o servidor só envia dados quando solicitado pelo cliente. Isso é ineficiente para monitoramento em tempo real, pois exigiria que o navegador solicitasse atualizações constantemente (*polling*), sobrecarregando a rede.

Para solucionar isso, o projeto utiliza a biblioteca Socket.IO, que implementa o protocolo WebSocket. Esse protocolo estabelece um túnel de comunicação bidirecional e persistente (full-duplex) sobre uma única conexão TCP. Isso permite que o servidor envie dados (como a mudança de estado de um sensor) para a interface do usuário proativamente (*push*), garantindo que o painel de controle reflita o estado físico da bancada com latência mínima.

### 2.4.3. TECNOLOGIAS DE FRONTEND (HTML, CSS E JAVASCRIPT)

A camada de apresentação, ou *Frontend*, é responsável pela interação direta com o operador. Ela é construída sobre a tríade fundamental das tecnologias web, conforme padronizado pelo W3C (*World Wide Web Consortium*):

- **HTML5 (*HyperText Markup Language*):** Define a estrutura semântica e o conteúdo da página. No projeto, elementos HTML organizam os painéis de controle, botões de acionamento e áreas de visualizações gráfica.
- **CSS3 (*Cascading Style Sheets*):** Responsável pela camada de apresentação visual. Através do CSS, implementa-se o Design Responsivo utilizando *Media Queries*, que permitem adaptar o layout da interface a diferentes resoluções de tela. Isso assegura

a usabilidade tanto em monitores de desktop quanto em telas sensíveis ao toque de dispositivos móveis.

- **JavaScripts (JS):** É a linguagem de programação executada no lado do cliente (navegador). O JavaScript confere dinamismo à interface, sendo responsável por capturar eventos do usuário (cliques), manipular o DOM (*Document Object Model*) para atualizar elementos visuais sem recarregar a página e gerenciar a comunicação assíncrona com o servidor via Socket.IO.

### 2.4.4. ARQUITETURA REST E FORMATO JSON

A comunicação entre o Frontend (navegador) e o Backend (Flask) segue os princípios da arquitetura REST (*Representational State Transfer*). Em uma API RESTful, o servidor expõe "recursos" (como /api/atuador/comando) que podem ser manipulados através de verbos HTTP.

Para o transporte dos dados nessas requisições, utiliza-se o formato JSON (*JavaScript Object Notation*). O JSON é um padrão leve de intercâmbio de dados, legível por humanos e fácil de ser interpretado por máquinas. No projeto, ele é utilizado tanto para estruturar os comandos enviados ao servidor quanto para as mensagens MQTT trocadas com o hardware, garantindo uma padronização dos dados em todo o sistema. Um exemplo de comando em JSON, para comandar um atuador, é "{"id":"1", "acao": "avancar"} ". Nesse exemplo, o comando faz com que o atuador enumerado de "1" avance.

## 3. DESENVOLVIMENTO

Este capítulo descreve a metodologia aplicada para a construção do sistema de controle eletropneumático via IoT. O desenvolvimento foi estruturado em etapas sequenciais, iniciando-se pela definição da arquitetura de comunicação, seguida pela preparação do hardware servidor, desenvolvimento do firmware de controle e, por fim, a implementação da interface homem-máquina.

Todo o código-fonte desenvolvido, bem como o passo a passo para a configuração do projeto estão disponíveis publicamente no repositório do projeto "iot-pneumatic-control" no GitHub[1].

[1] https://github.com/glatalisa/iot-pneumatic-control

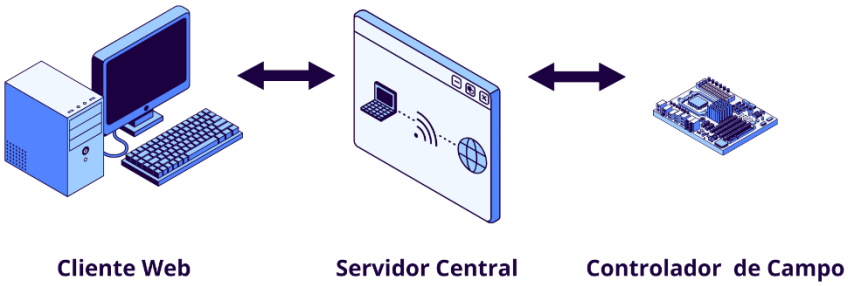
## 3.1. ARQUITETURA DO PROJETO

A arquitetura da solução foi concebida baseada no modelo publicador-subscitor, projetada para garantir baixa latência em rede local e acessibilidade remota via internet. O sistema é composto por três nós principais que interagem de forma assíncrona:

1. **Nó de Aplicação e Gerenciamento (Servidor Central):** Representado pelo microcomputador Raspberry Pi 2. Este nó atua como o "cérebro" do sistema, hospedando o servidor web (*backend*) e o *broker* de mensagens. Ele é responsável por centralizar a lógica de negócios, gerenciar as sessões dos usuários, executar as sequências de automação e servir a interface gráfica. A comunicação interna deste nó utiliza *WebSockets* para prover atualizações em tempo real para o frontend.
2. **Nó de controle e Atuação (Controlador de Campo):** Representado pelo microcontrolador ESP8266. Este dispositivo opera na borda (*edge*), conectado fisicamente aos atuadores e sensores da bancada. Sua função é puramente executiva e de sensoriamento: ele recebe comandos de acionamento via protocolo MQTT e retorna o estado dos sensores para o sistema. Esta separação de responsabilidades garante que a lógica complexa não sobrecarregue o microcontrolador.
3. **Nó de Interface (Cliente Web):** Qualquer dispositivo (smartphone, tablet ou computador) equipado com um navegador web moderno. Este nó não requer instalação de software dedicado, pois a aplicação é carregada via HTTP. Ele atua como a interface de comando, enviando requisições JSON para a API do servidor e renderizando visualmente o estado da bancada através de animações CSS dinâmicas.

A Figura 10 ilustra a arquitetura do projeto.

Figura 10 - Arquitetura do projeto




Fonte: Próprio Autor (2025)

A comunicação entre estes nós ocorre sobre uma rede TCP/IP. Para o acesso externo, foi implementada uma estratégia de tunelamento e DNS Dinâmico (DDNS), permitindo que o servidor local, situado atrás de um NAT (*Network Address Translation*), seja alcançável pela internet pública sem a necessidade de um IP fixo comercial.

## 3.2. MATERIAIS E MÉTODOS

Para a execução do projeto, foram utilizados componentes de hardware e ferramentas de software de código aberto (*open source*), alinhados com a filosofia de acessibilidade da indústria 4.0.

Recursos de Hardware:

- **Servidor:** Raspberry Pi 2 Model B (CPU Quad-Core ARM Cortex-A7 900MHz, 1GB RAM) com um cartão micro SD de 8 GB, escolhido por sua robustez e capacidade de executar um sistema operacional Linux completo (Raspberry Pi OS Lite).
- **Controlador:** Módulo ESP8266 NodeMCU v3, selecionado por integrar microcontrolador e transceptor Wi-Fi em um único chip.
- **Interface de Potência:** Módulos relé de 5V para conversão de potência no acionamento dos atuadores (o ESP gera sinal de atuação em 5VDC e os solenoides devem sem ativados com 24VDC) e módulos relé de 24V para conversão de potência dos sinais provenientes dos sensores de campo (os sensores instalados

geram sinais 24VDC enquanto o ESP deve receber um sinal 5VDC em suas portas de entradas digitais).

- **Bancada Eletropneumática:** Composta por 3 cilindros de dupla ação, 2 válvulas solenoides 5/2 vias monoestáveis, 1 válvula solenoide 5/2 vias biestáveis e 6 sensores de fim de curso (1 rolete eletromecânico, 1 detector de posição indutivo, 1 detector de posição capacitivo, 1 detector de posição ótico e 2 detectores de posição magnéticos).
- **Infraestrutura de Rede:** Roteador Wi-Fi para criação da rede local e conexão com a internet.

Ferramentas de Software:

- **Visual Studio Code (VS Code):** Ambiente de desenvolvimento integrado utilizado para a escrita dos códigos em Python, HTML, CSS e JavaScript.
- **Arduino IDE:** Utilizada para a compilação e *upload* do *firmware* em C++ para o ESP8266.
- **Python 3.9:** Linguagem de programação utilizada no *backend*, escolhida pela sua vasta biblioteca de suporte a redes e servidores web.
- **Flask:** *Microframework* web utilizado para construir a API REST e servir a aplicação.
- **Flask-SocketIO:** Biblioteca para implementação de comunicação *full-duplex* (WebSockets).
- **Eclipse Mosquitto**: *Broker* MQTT leve e eficiente, instalado no Raspberry Pi.
- **MQTT Explorer**: Ferramenta de diagnóstico utilizada para monitorar e depurar as mensagens MQTT em tempo real.
- **Git & GitHub:** Utilizados para versionamento de código e documentação do projeto.

## 3.3. CONFIGURAÇÃO DO SERVIDOR

A implementação do servidor central foi realizada no microcomputador Raspberry Pi 2 Model B. Esta etapa envolveu desde a preparação do sistema operacional até a configuração dos serviços de rede e aplicação. O objetivo foi criar um ambiente estável, seguro e capaz de operar de forma autônoma (*headless*), ou seja, sem a necessidade de monitores ou periféricos conectados diretamente ao dispositivo.

### 3.3.1. INSTALAÇÃO DO SISTEMA OPERACIONAL E REDE

A primeira etapa consistiu na instalação do sistema operacional Raspberry Pi OS Lite (versão baseada em Debian Bullseye, 32-bit). Optou-se pela versão "Lite" devido à ausência de interface gráfica de usuário (GUI), o que reduz significativamente o consumo de memória RAM e processamento, dedicando os recursos do hardware exclusivamente para a execução do *broker* MQTT e do servidor web.

A gravação da imagem do sistema no cartão micro SD foi realizada utilizando a ferramenta **Raspberry Pi Imager**. Durante este processo, foram pré-configuradas as credenciais de acesso SSH (*Secure Shell*) e as configurações iniciais de rede, permitindo o acesso remoto ao terminal do Raspberry Pi imediatamente após a primeira inicialização.

Para garantir que o servidor fosse sempre acessível pelo mesmo endereço na rede local, foi imperativo configurar um endereço IP estático. Diferente da atribuição dinâmica via DHCP, onde o endereço pode mudar a cada reinicialização, o IP fixo assegura a estabilidade das conexões MQTT e HTTP. A configuração foi realizada através da edição do arquivo de sistema **/etc/dhcpcd.conf.**

No arquivo, foram adicionadas as diretivas para a interface de rede Ethernet (eth0), definindo o endereço IP desejado, o *gateway* (roteador) e os servidores DNS, conforme exemplo ilustrado no trecho de código apresentado na Figura 11:

Figura 11 - Exemplo de configuração do endereço de rede atribuído ao servidor local (Raspberry Pi 2 Model B)

```
interface eth0
static ip_address=192.168.0.50/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8
```

Fonte: Próprio Autor (2025)

### 3.3.2. IMPLEMENTAÇÃO DO BROKER MQTT

O elemento central da comunicação assíncrona do projeto é o *broker* MQTT. O software escolhido foi o **Eclipse Mosquitto**, devido à sua leveza e conformidade com os padrões do protocolo. A instalação foi realizada via gerenciador de pacotes do Linux (apt), juntamente com as ferramentas de cliente para testes (mosquitto-clients).

Após a instalação, foi necessário ajustar as configurações padrão de segurança para permitir conexões externas, visto que as versões mais recentes do Mosquitto bloqueiam o acesso remoto por padrão. O arquivo de configuração /etc/mosquitto/mosquitto.conf foi editado para incluir as seguintes diretivas:

- **listener 1883**: Configura o *broker* para ouvir conexões na porta padrão 1883 de qualquer interface de rede.
- **allow_anonymous true**: Permite conexões sem autenticação de usuário e senha, facilitando os testes iniciais e o desenvolvimento didático (em ambientes de produção, recomenda-se alterar para false e configurar credenciais).

Para aplicar as alterações, o serviço foi reiniciado através do gerenciador de sistemas **systemd**. A validação do funcionamento foi feita através do comando **systemctl status mosquitto**, confirmando que o serviço estava ativo e em execução (*active running*).

### 3.3.3. AMBIENTE DE DESENVOLVIMENTO PYTHON E FLASK

A camada de aplicação, responsável pela lógica de controle e pela interface web, foi desenvolvida em **Python 3**. Para isolar as dependências do projeto das bibliotecas do sistema operacional, adotou-se o uso de um ambiente virtual (*virtual environment*).

O processo de configuração do ambiente de *backend* seguiu os seguintes passos:

1. **Criação do Ambiente Virtual:** Utilizando o módulo **venv**, criou-se um diretório isolado (**.venv**) para a instalação dos pacotes específicos do projeto.
2. **Instalação de Dependências:** As bibliotecas necessárias foram instaladas via gerenciador de pacotes **pip**. As principais bibliotecas utilizadas foram:
    a. **Flask:** O *framework* web principal.
    b. **Flask-SocketIO:** Para gerenciar conexões *WebSocket* de baixa latência.
    c. **Paho-MQTT:** Cliente MQTT para Python, permitindo que a aplicação publique comandos e subscreva aos tópicos de status dos sensores.
    d. **Eventlet:** Servidor WSGI assíncrono, necessário para o desempenho adequado do *Socket.IO*.

### 3.3.4. DESENVOLVIMENTO DO SERVIDOR DE APLICAÇÃO

O núcleo lógico do sistema foi desenvolvido em Python, consolidado no arquivo **server.py**. Este *script* é responsável por orquestrar a comunicação entre a interface do usuário

e os controladores de campo. O código foi estruturado utilizando o *framework* Flask para o gerenciamento das rotas web e a biblioteca Flask-SocketIO para a comunicação em tempo real.

A arquitetura do software servidor pode ser dividida em quatro blocos funcionais principais:

1. **Gerenciamento de Rotas e Sessões:** O Flask utiliza decoradores de rota (ex: **@app.route**) para mapear URLs a funções Python. Implementou-se um sistema de sessões no lado do servidor (Flask-Session) para armazenar as configurações do usuário (número de atuadores e modo de operação) de forma persistente durante a navegação. Isso permite que a aplicação mantenha o estado do sistema consistente entre diferentes acessos.
2. **Interface de Comunicação MQTT:** A integração com o protocolo MQTT foi realizada através da biblioteca Paho-MQTT. O servidor atua como um cliente MQTT que:
    a. **Publica Comandos:** Envia instruções de acionamento para os tópicos específicos (ex: projeto/atuadores/comando) com QoS 1, garantindo a entrega.
    b. **Subscreve Status:** Escuta o tópico de retorno (projeto/atuadores/status) para receber confirmações de ação e leituras de sensores enviadas pelo ESP8266.
    c. **Processa Mensagens (Callback):** A função de *callback* **on_message** processa os payloads JSON recebidos e, crucialmente, retransmite essas informações para o *frontend* via *WebSocket*, fechando o ciclo de feedback em tempo real.
3. **Motor de Automação (Sequenciador):** Para o modo de controle sequencial, desenvolveu-se um algoritmo de interpretação de comandos (*parser*) capaz de processar strings complexas (ex: 1+ T2 [2+ 2-]*3). A execução ocorre em uma *thread* de segundo plano para não bloquear o servidor principal. A lógica de controle emprega um mecanismo de sincronização de *threads* (threading.Condition). O sequenciador envia um comando e entra em estado de espera; ele só prossegue para o próximo passo quando a *thread* do MQTT confirma o recebimento do sinal do sensor correspondente, caracterizando um controle em malha fechada robusto.
4. **Interface de Tempo Real (WebSockets):** Utilizando a biblioteca Socket.IO, o servidor estabelece um canal bidirecional com o navegador. Eventos como **update_status** (mudança de estado de um atuador) ou **sequencer_status** (progresso da sequência) são emitidos proativamente do servidor para o cliente (*Server-Push*),

garantindo que o *dashboard* reflita o estado físico da bancada instantaneamente, sem a necessidade de recarregamento da página.

## 3.4. DESENVOLVIMENTO DO FIRMWARE (ESP8266)

O desenvolvimento do software embarcado (*firmware*) para o controlador de campo foi realizado utilizando a IDE do Arduino (*Integrated Development Environment*), configurada para a plataforma ESP8266 (modelo NodeMCU 1.0). A linguagem utilizada é o C++, com a adição de bibliotecas específicas para conectividade e manipulação de dados.

O código-fonte foi estruturado para operar de maneira assíncrona e não-bloqueante, garantindo que o microcontrolador possa gerenciar a comunicação de rede e o controle de hardware simultaneamente, sem latências perceptíveis.

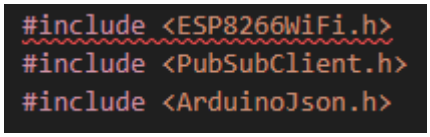
### 3.4.1. BIBLIOTECAS E DEPENDÊNCIAS

A primeira etapa do código consiste na inclusão das bibliotecas que fornecem as funcionalidades básicas de rede e processamento de dados. Foram utilizadas três bibliotecas externas fundamentais:

1. **ESP8266WiFi.h:** Biblioteca nativa do núcleo do ESP8266, responsável por gerenciar a pilha TCP/IP e a conexão com a rede sem fio (Wi-Fi). Ela permite a configuração dos modos de operação (Station Mode) e o gerenciamento de reconexões automáticas.
2. **PubSubClient.h:** Biblioteca que implementa um cliente leve para o protocolo MQTT. Ela gerencia a conexão com o *broker*, a subscrição em tópicos de comando e a publicação de mensagens de status.
3. **ArduinoJson.h:** Biblioteca utilizada para a serialização e desserialização de objetos JSON. Como o protocolo de comunicação do sistema padroniza o uso de JSON para o intercâmbio de dados (ex: {"id": "1", "acao": "avancar"}), esta biblioteca é essencial para interpretar as mensagens recebidas do servidor e formatar as respostas dos sensores.

A Figura 12 apresenta o processo de inclusão das bibliotecas.

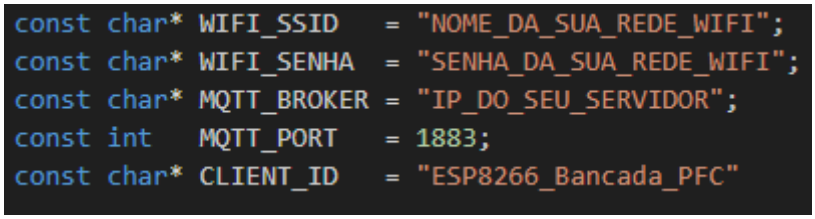
Figura 12 - Definição de bibliotecas necessárias para execução da computação de borda com ESP8266




Fonte: Próprio Autor (2025)

Logo após as inclusões, são definidas as constantes de configuração da rede, que armazenam as credenciais de acesso ao Wi-Fi (SSID e SENHA) e o endereço IP do servidor *broker* MQTT. Estas foram declaradas como constantes globais (**const char***) para facilitar a alteração caso a infraestrutura de rede mude, como mostra a Figura 13.

Figura 13 – Trecho do código em execução no ESP8266 para declaração das credenciais de acesso ao Wi-Fi e Broker MQTT




Fonte: Próprio Autor (2025)

### 3.4.2. MAPEAMENTO DE HARDWARE E VARIÁVEIS DE ATUAÇÃO

Para garantir que o software interaja corretamente com o mundo físico, realizou-se um mapeamento rigoroso dos pinos GPIO (*General Purpose Input/Output*). Em vez de utilizar os números brutos dos pinos no meio do código, foram criadas constantes descritivas para cada atuador:

- **Relés (Saídas):** Foram definidas constantes como RELAY_PIN_1, RELAY_PIN_2, etc., associando-as aos pinos físicos correspondentes (D1, D2, D5 e D6). Isso abstrai a complexidade do hardware e facilita a manutenção.
- **Lógica de Ativação:** Para tornar o código adaptável a diferentes módulos de relé (que podem ser acionados por nível lógico alto ou baixo), foram criadas as definições RELAY_ACTIVE_LEVEL (configurado como LOW ou HIGH) e RELAY_INACTIVE_LEVEL. Todas as funções de controle utilizam essas definições em vez de HIGH/LOW diretamente, prevenindo erros de lógica invertida.

A Figura 14 mostra o trecho do código para o mapeamentos dos pinos de saída lógica de ativação.

Figura 14 – Trecho do código em execução no ESP8266 para mapeamento de pinos de saída e lógica de ativação

```
// --- Relés (Saídas) ---
const int RELAY_PIN_1     = D1; // Atuador 1 (Mola)
const int RELAY_PIN_2     = D2; // Atuador 2 (Mola)
const int RELAY_PIN_3_ADV = D5; // Atuador 3 (Duplo - Avanço)
const int RELAY_PIN_3_REC = D6; // Atuador 3 (Duplo - Recuo)
const int RELAY_PIN_4_ADV = -1; // Atuador 4 (Duplo - Avanço) *Desativado
const int RELAY_PIN_4_REC = -1; // Atuador 4 (Duplo - Recuo) *Desativado

// Lógica do Relé (Ajustar conforme módulo utilizado)
#define RELAY_ACTIVE_LEVEL LOW
#define RELAY_INACTIVE_LEVEL HIGH
#define PULSE_DURATION 200
```

Fonte: Próprio Autor (2025)

Para os sensores, adotou-se uma abordagem orientada a objetos simplificada. Foi criada uma estrutura de dados (struct) chamada **Sensor**, que agrupa todas as propriedades de um sensor físico:

1. **int pin**: O pino físico onde o sensor está conectado.
2. **const char* id**: O nome textual do sensor (ex: "1_avancado"), usado nas mensagens MQTT.
3. **int lastState**: Uma variável de memória que armazena a última leitura (ativado/desativado).

Um vetor (array) de objetos Sensor foi então instanciado, contendo a configuração de todos os 8 sensores possíveis. Pinos não utilizados foram marcados com o valor -1, instruindo a lógica do programa a ignorá-los dinamicamente. O trecho do código é demonstrado na Figura 15.

Figura 15 – Trecho do código em execução no ESP8266 para mapeamento eficiente dos sensores

```
const int SENSOR_PIN_1_ADV = D7;  // GPIO13
const int SENSOR_PIN_1_REC = D3;  // GPIO0
const int SENSOR_PIN_2_ADV = D0;  // GPIO16
const int SENSOR_PIN_2_REC = D4;  // GPIO2
const int SENSOR_PIN_3_ADV = 3;   // GPIO3 (RX)
const int SENSOR_PIN_3_REC = 1;   // GPIO1 (TX)
const int SENSOR_PIN_4_ADV = -1;  // *Desativado
const int SENSOR_PIN_4_REC = -1;  // *Desativado

// Estrutura para gerenciamento eficiente dos sensores
struct Sensor {
  int pin;
  const char* id;
  int lastState;
};

// Array de configuração dos sensores
// Pinos definidos como -1 são ignorados pelo sistema
Sensor sensors[] = {
  {SENSOR_PIN_1_ADV, "1_avancado", -1}, {SENSOR_PIN_1_REC, "1_recuado", -1},
  {SENSOR_PIN_2_ADV, "2_avancado", -1}, {SENSOR_PIN_2_REC, "2_recuado", -1},
  {SENSOR_PIN_3_ADV, "3_avancado", -1}, {SENSOR_PIN_3_REC, "3_recuado", -1},
  {SENSOR_PIN_4_ADV, "4_avancado", -1}, {SENSOR_PIN_4_REC, "4_recuado", -1}
};
const int NUM_SENSORS = sizeof(sensors) / sizeof(sensors[0]);
```

Fonte: Próprio Autor (2025)

### 3.4.3. INSTANCIAÇÃO DE CLIENTES E TEMPORIZADORES

Para gerenciar a comunicação, foram instanciados dois objetos globais:

- **WiFiClient wifiClient**: Cria a camada de transporte TCP.
- **PubSubClient mqttClient(wifiClient)**: Cria o cliente MQTT utilizando a camada TCP criada anteriormente.

Além disso, para garantir que o sistema opere de forma não-bloqueante (sem travar enquanto espera por eventos), foram declaradas variáveis do tipo **unsigned long** para controle de tempo: **lastReconnectAttempt** (controla o intervalo de tentativas de reconexão) e **lastSensorReadTime** (controla a frequência de leitura dos sensores para evitar ruído/rebote), conforme ilustrado na Figura 16.

Figura 16 – Trecho do código em execução no ESP8266 para declaração de variáveis globais e objetos

```
WiFiClient wifiClient;
PubSubClient mqttClient(wifiClient);

unsigned long lastReconnectAttempt = 0;
unsigned long lastSensorReadTime = 0;
const long SENSOR_READ_INTERVAL = 50; // Debounce de 50ms
unsigned long lastWifiStatusTime = 0;
```

Fonte: Próprio Autor (2025)

### 3.4.4. FUNÇÕES DE CONTROLE E LÓGICA DE ATUAÇÃO

A função de *callback* MQTT é o coração da atuação. Ela é executada automaticamente sempre que uma nova mensagem chega. Sua lógica foi detalhada da seguinte forma:

1. **Recepção:** A mensagem bruta (payload) é convertida para uma *String*.
2. **Interpretação:** O ArduinoJson extrai os campos "id" e "acao".
3. **Decisão:** Uma estrutura condicional (if/else) verifica o ID do atuador e aplica a lógica correspondente:
    a. Para válvulas direcionais monoestáveis (acionamento por solenoide e retorno por mola) (IDs 1 e 2), a função auxiliar **setRelay()** é chamada para manter o nível lógico constante.
    b. Para válvulas direcionais biestáveis (duplo solenoide**) (IDs 3 e 4)**, implementou-se uma lógica de segurança que desliga a bobina oposta antes de ligar a bobina desejada, evitando acionamento duplo.
4. **Feedback:** Imediatamente após o acionamento físico, uma mensagem de confirmação (**atuador_cmd**) é publicada de volta para o servidor.

### 3.4.5. MONITORAMENTO DE SENSORES E DEBOUNCE

A leitura dos sensores é realizada pela função **checkAllSensors()**. Diferente de uma leitura simples, esta função implementa um algoritmo de detecção de borda (mudança de estado). A cada ciclo de leitura (definido pelo temporizador **SENSOR_READ_INTERVAL**

de 50ms), a função percorre o vetor de sensores. Ela compara o valor lido atualmente no pino físico com o valor armazenado na variável **lastState** daquele sensor específico.

Se os valores forem iguais, nada acontece. Porém, se os valores forem diferentes, significa que houve uma mudança física (o cilindro alcançou ou saiu de um determinado marcador de curso). Nesse momento, o novo estado é atualizado na memória e uma mensagem MQTT (**sensor_estado**) é enviada ao servidor.

Esse método economiza largura de banda da rede, pois transmite dados apenas quando eventos reais ocorrem, em vez de enviar o estado continuamente. O código responsável por esta lógica é apresentado na Figura 17.

Figura 17 – Trecho do código em execução no ESP8266 para verificação eficiente dos sensores

```
void checkAllSensors() {
  for (int i = 0; i < NUM_SENSORS; i++) {
    if (sensors[i].pin == -1) continue;

    int currentState = digitalRead(sensors[i].pin);
    if (currentState != sensors[i].lastState) {
      const char* estadoStr = (currentState == LOW) ? "ativado" : "desativado";

      String debugMsg = "Sensor " + String(sensors[i].id) + ": " + String(estadoStr);
      publishDebug(debugMsg);

      publishStatus("sensor_estado", sensors[i].id, estadoStr);

      sensors[i].lastState = currentState;
    }
  }
}
```

Fonte: Próprio Autor (2025)

### 3.4.6. GERENCIAMENTO DE CICLO DE VIDA (SETUP E LOOP)

A função **Setup** é executada uma única vez ao ligar. Configura os pinos dos relés como saída (garantindo que iniciem desligados para segurança), configura os pinos dos sensores como entrada com resistor de *pull-up* interno (INPUT_PULLUP), e inicia a conexão Wi-Fi.

Já a função **Loop** é executada continuamente. Sua principal responsabilidade é manter a conexão MQTT viva através da função **mqttClient.loop()** e gerenciar os temporizadores de software para a leitura dos sensores e reconexão automática em caso de falha na rede, garantindo que o dispositivo opere de forma autônoma e resiliente.

## 3.5. DESENVOLVIMENTO DA INTERFACE

A interface do usuário foi desenvolvida como uma *Single Page Application* (SPA) híbrida, utilizando as tecnologias nativas da web: **HTML5** para estruturação semântica, **CSS3** para estilização e **JavaScript (ES6)** para a lógica de interação e comunicação assíncrona. Não foram utilizados *frameworks* pesados (como React ou Angular), optando-se por uma abordagem leve (*Vanilla JS*) que garante alto desempenho no carregamento e execução, mesmo em dispositivos móveis com recursos limitados.

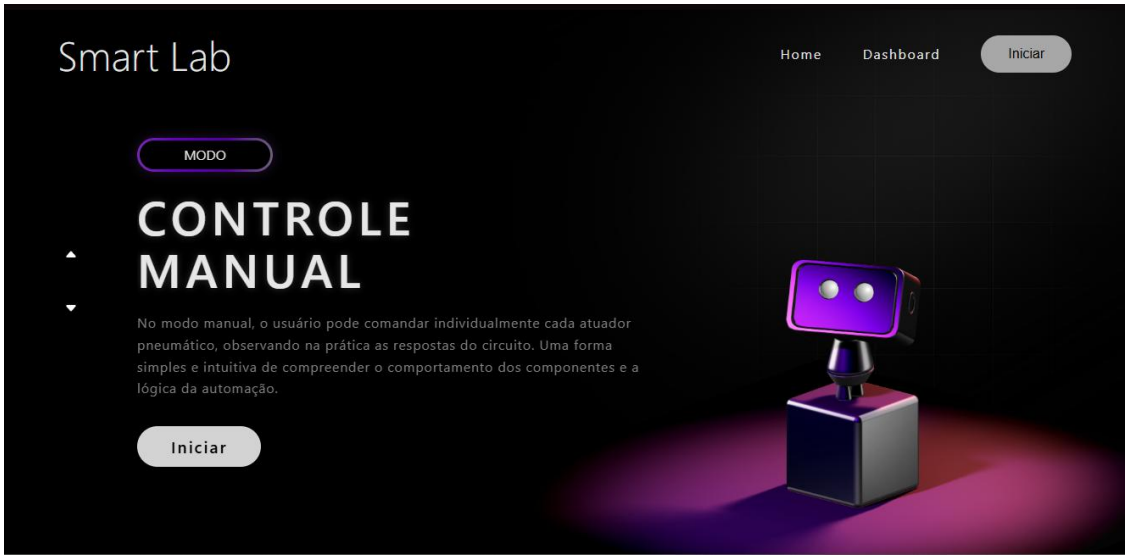
O *design* visual adotou o padrão "Dark Mode" (Modo Escuro) com elementos de *Glassmorphism* (efeito de vidro fosco), proporcionando uma estética moderna e reduzindo a fadiga visual do operador. A responsividade foi assegurada através de *CSS Flexbox* e *Media Queries*, permitindo que a interface se adapte automaticamente a telas de desktops, tablets e smartphones.

### 3.5.1. ARQUITETURA DE NAVEGAÇÃO E SESSÃO

A navegação do sistema foi projetada em um fluxo linear de três etapas para garantir a configuração correta do hardware antes da operação.

A tela de boas-vindas apresenta o sistema e permite a seleção inicial do modo de operação através de um carrossel interativo, como pode ser visto na Figura 18.

Figura 18 - Tela de Boas-Vindas do Servidor Web




Fonte: Próprio Autor, extraídas do servidor web desenvolvido

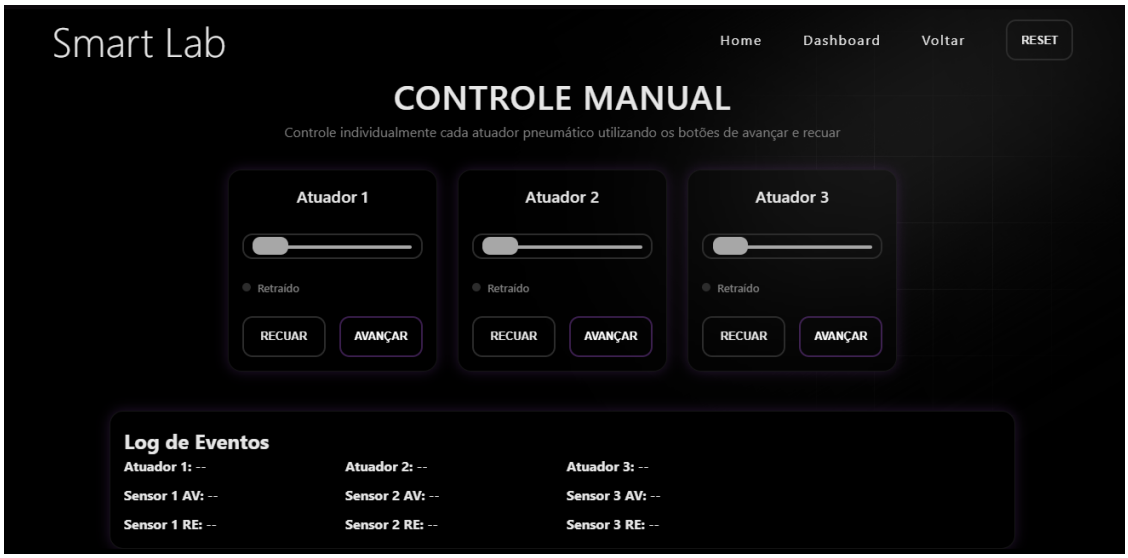
Na tela de configuração, o usuário define o número de atuadores fisicamente presentes na bancada. O *script* captura essa informação e a envia para o servidor via requisição POST. O servidor *Flask* armazena esses dados em uma **Sessão de Usuário** no *backend*. Isso permite que a interface de controle seja gerada dinamicamente, exibindo apenas os controles necessários para a configuração atual (ver Figura 19).

Figura 19 - Tela de configuração do Servidor Web


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

Após realizada a configuração do sistema, o usuário pode seguir por dois caminhos: a página de controle manual (Figura 20) ou a página de controle sequencial (Figura 21).

Figura 20 - Tela de controle manual do Servidor Web




Fonte: Próprio Autor, extraídas do servidor web desenvolvido

Figura 21 - Tela de controle sequencial do Servidor Web


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

### 3.5.2. RENDERIZAÇÃO DINÂMICA E ANIMAÇÕES

Uma inovação deste projeto é a ausência de elementos estáticos para os atuadores. Em vez de desenhar fixamente quatro cilindros na tela, desenvolveu-se um algoritmo em JavaScript que, ao carregar a página, consulta a variável de sessão **num_atuadores** e injeta o código HTML correspondente no DOM (*Document Object Model*).

Para a visualização do estado, criou-se uma representação vetorial dos cilindros utilizando apenas CSS. A animação de avanço e recuo não utiliza vídeos ou GIFs, mas sim a transição de propriedades CSS (**transition: all 0.5s**).

No estado recuado a div que representa a haste do cilindro possui a propriedade left: 5%, representando o estado retraído (Figura 22).

Figura 22 – Representação do atuador retraído no servidor web


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

No estado avançado, ao receber a confirmação de movimento, o JavaScript adiciona a classe **.estado-avancado** ao elemento, alterando a propriedade para left: 65%. O navegador interpola essa mudança, gerando uma animação suave que simula o movimento físico do pistão, resultando no estado visual apresentado na Figura 23.

Figura 23 – Representação do atuador estendido no servidor web


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

### 3.5.3. MECANISMOS DE ACIONAMENTO E FEEDBACK

A operação de controle via interface web não ocorre de forma direta, mas sim através de um ciclo de comando e confirmação que envolve todas as camadas da arquitetura. O processo ocorre da seguinte maneira:

1. **Envio do Comando (Frontend -> Backend):** Ao clicar no botão "Avançar", o JavaScript captura o evento e utiliza a **Fetch API** para enviar uma requisição HTTP assíncrona (AJAX) para a rota **/api/atuador/comando** do servidor, contendo um *payload* JSON (ex: {"id": "1", "acao": "avancar"}). Isso ocorre sem recarregar a página.
2. **Processamento e Envio MQTT (Backend -> Hardware):** O servidor Python recebe a requisição, valida os dados e publica a mensagem no tópico MQTT correspondente.
3. **Execução Física e Retorno (Hardware -> Backend):** O ESP8266 aciona a válvula e, após confirmar a mudança de estado (via sensor ou lógica interna), publica uma mensagem de *status* de volta para o broker.
4. **Atualização em Tempo Real (Backend -> Frontend):** O servidor recebe a mensagem de confirmação do *status* via MQTT e o retransmite imediatamente para o navegador através de um canal **WebSocket** (biblioteca *Socket.IO*).
5. **Atualização Visual (Frontend):** O cliente JavaScript, que mantém uma escuta ativa no evento **update_status** do *Socket.IO*, recebe os dados. Uma função de *callback* identifica qual atuador mudou de estado e aplica a classe CSS de animação correspondente.

Este ciclo garante que a animação na tela só ocorra se o hardware realmente confirmar a ação, proporcionando um *feedback* visual confiável para o operador.

### 3.5.4. MODO DE CONTROLE SEQUENCIAL

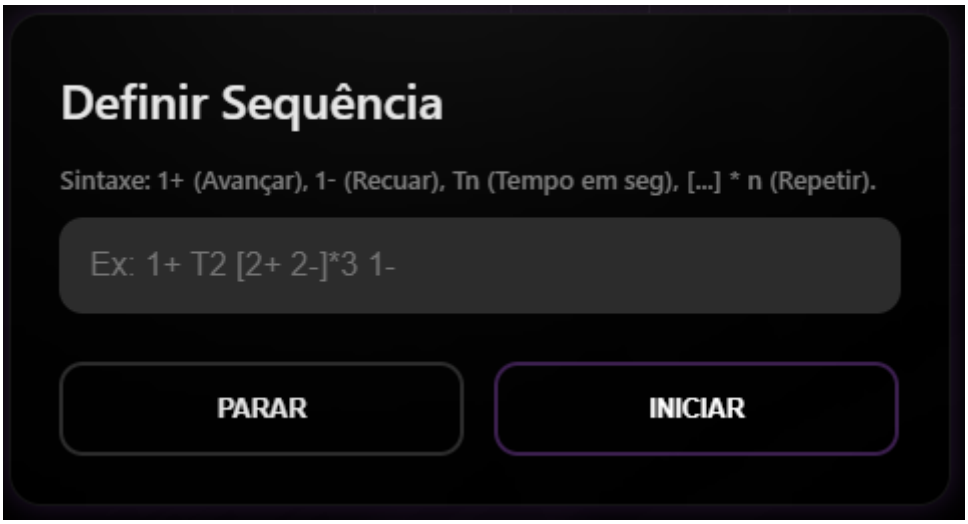
A interface do modo sequencial foi desenvolvida para permitir a programação de automação sem necessidade de reescrever o código do microcontrolador. Ela fornece um campo de entrada de texto onde o usuário pode digitar instruções utilizando uma sintaxe própria desenvolvida para o projeto:

- **Ações Simples:** 1+ (Avança atuador 1), 2- (Recua atuador 2).

- **Temporizadores:** T2.5 (Pausa a execução por 2,5 segundos).
- **Ações Simultâneas:** (1+ 2+) (Executa ambos os comandos em paralelo).
- **Loops de Repetição:** [1+ 1-]*3 (Repete o bloco interno 3 vezes).

A Figura 24 demonstra o campo de inserção destes comandos na interface.

Figura 24 - Campo de definição da sequência de acionamento




Fonte: Próprio Autor, extraídas do servidor web desenvolvido

O JavaScript desta página valida se o campo não está vazio e envia a *string* completa para a API **/api/sequencer/start**, que inicia a sequência de acionamento.

### 3.5.5. DASHBOARD DE DIAGNÓSTICO

Para monitoramento da saúde do sistema, implementou-se uma janela flutuante (*modal*) acessível de qualquer tela. Este painel exibe métricas críticas de infraestrutura que não dizem respeito ao processo mecânico, mas sim à conectividade:

- **Status do Servidor:** Indica se a conexão *Socket.IO* está ativa (Online/Offline).
- **Status do MQTT:** Confirma se o fluxo de mensagens entre o servidor e o *broker* está funcional.
- **Telemetria Wi-Fi:** Exibe a força do sinal (RSSI em dBm) do controlador ESP8266, permitindo diagnosticar problemas de alcance de rede na bancada.

A visualização deste painel encontra-se na Figura 25.

Figura 25 - Dashboard do Servidor Web


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

## 3.6. MONTAGEM EXPERIMENTAL E CONEXÕES FÍSICAS

A etapa final de desenvolvimento consistiu na integração física entre a unidade de controle (Microcontrolador) e a planta industrial (Bancada Eletropneumática). Esta seção detalha os circuitos de interface desenvolvidos para permitir que o ESP8266, que opera com sinais lógicos de 3.3V, controle válvulas de potência de 24V e leia sensores industriais.

### 3.6.1. INTERFACE DE POTÊNCIA DE ACIONAMENTO

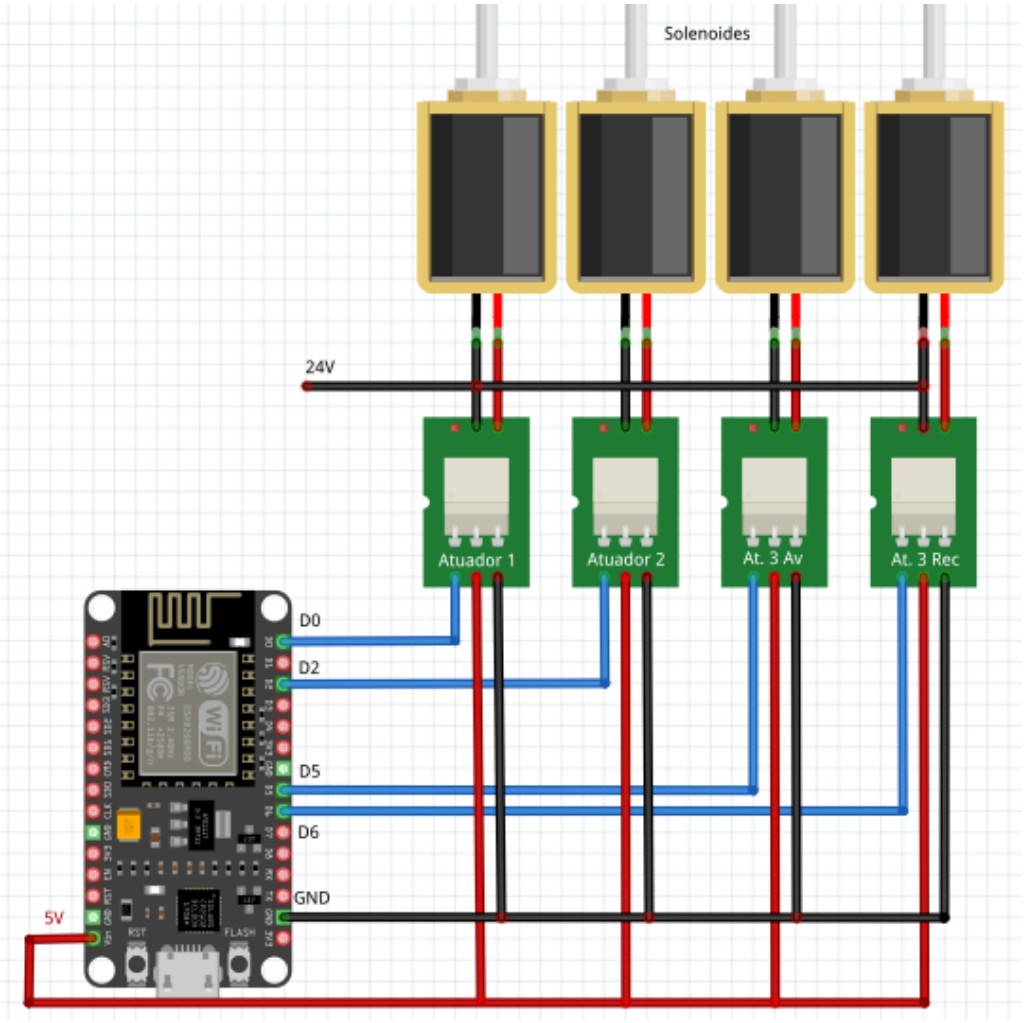
As válvulas solenoides da bancada operam com tensão de 24V DC, incompatível com as portas de saída do microcontrolador. Para realizar o acionamento seguro e isolado, utilizou-se um conjunto de **Módulos Relé de 5V**. O relé atua como um interruptor eletromecânico, garantindo o isolamento galvânico entre o circuito de controle (baixa tensão) e o circuito de carga (alta tensão).

A conexão elétrica foi estruturada em dois estágios:

1. **Estágio de Controle (Lógica):** Os pinos de sinal (IN) dos módulos relé foram conectados diretamente às saídas digitais (GPIOs) do ESP8266. A alimentação das bobinas dos relés (VCC e GND) foi derivada da fonte de 5V que alimenta o microcontrolador.
2. **Estágio de Potência (Carga):** O circuito das válvulas foi montado em série com os contatos do relé. O polo positivo da fonte de 24V foi conectado diretamente ao solenoide da válvula. O polo negativo (0V) foi conectado ao terminal Comum (COM) do relé, e o terminal Normalmente Aberto (NA ou NO) foi conectado ao retorno da válvula.

Desta forma, quando o ESP8266 envia um sinal lógico para o módulo, o relé fecha o contato COM-NA, energizando a válvula e provocando o movimento do atuador. A Figura 26 mostra o esquema de ligação do ESP8266 com os módulos relés que acionam os atuadores.

Figura 26 - Esquema de conexão para acionamento dos solenoides das válvulas direcionais



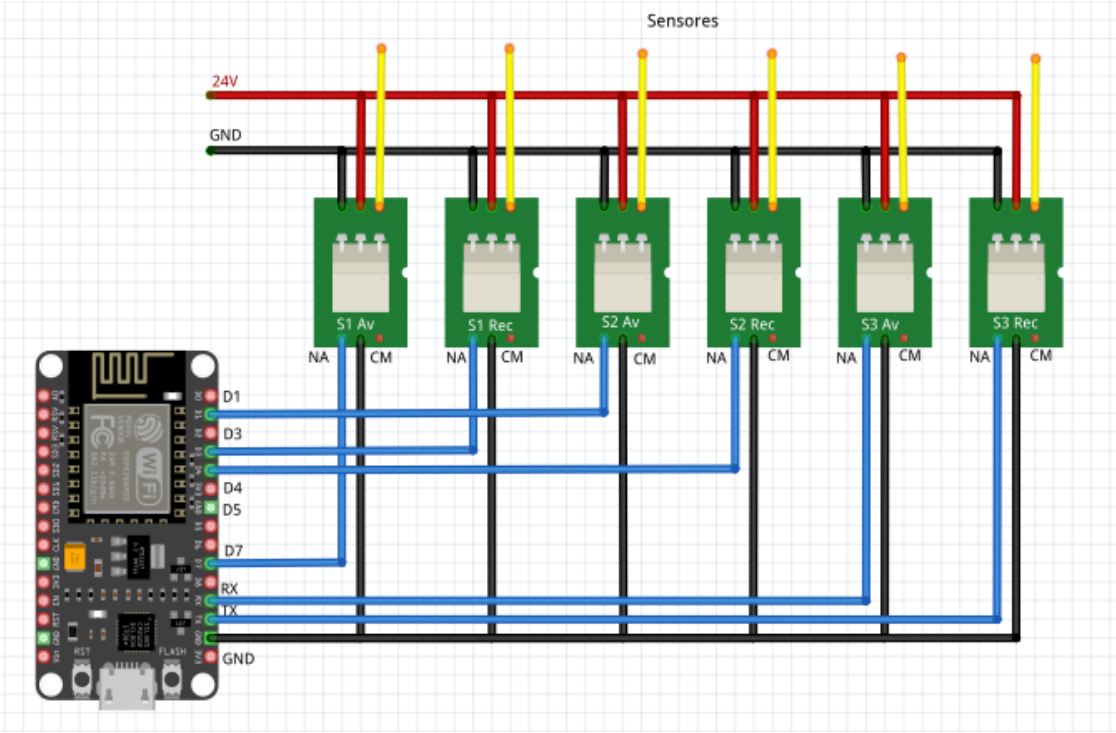
Fonte: Próprio Autor (2025)

### 3.6.2. INTERFACE DE SENSORIAMENTO

O monitoramento da posição dos atuadores é realizado por sensores de fim de curso. Para a leitura dos sinais, adotou-se a lógica inversa do acionamento dos atuadores, onde os sensores enviam um sinal de 24V para o módulo relé, que assim fecha a chave e aciona um pino digital designado para cada sensor.

No *firmware*, os pinos de entrada foram configurados com resistores de *pull-up* internos (INPUT_PULLUP). Esta configuração mantém o nível lógico alto (3.3V) quando o sensor está desativado. Quando o atuador atinge o fim do curso, o sensor ativa e fecha o circuito para o GND, alterando o nível lógico para baixo (0V), o que é interpretado pelo software como "sensor ativado". O esquema elétrico completo desta interface é detalhado na Figura 27.

Figura 27 - Esquema de conexão para leitura dos sensores




Fonte: Próprio Autor (2025)

### 3.6.3. INTEGRAÇÃO DA BANCADA

A montagem final foi realizada na bancada didática, organizando os componentes para evitar choques, movimentações e manter a janela de visualização o mais organizada possível.

Utilizou-se uma fonte de alimentação chaveada de 24V para o circuito pneumático e fontes independentes de 5V para o Raspberry Pi e o ESP8266, unificando-se apenas os referenciais de terra (GND) para garantir a estabilidade dos sinais.

A disposição física dos componentes permitiu o acesso visual aos LEDs indicadores dos módulos relé e do próprio microcontrolador, facilitando o diagnóstico visual durante os testes de operação.

# 4. RESULTADOS

Este capítulo apresenta os resultados obtidos a partir da integração entre a interface web, o servidor central e a bancada eletropneumática. São detalhados a montagem física final, os testes de funcionamento dos diferentes modos de operação e uma análise do desempenho do sistema em condições reais de uso. Um vídeo com exemplos de aplicação do sistema integrado à bancada de acionamentos eletropneumáticos está disponível na pasta “PFC – Gabriel Latalisa” no Google Drive[2].

## 4.1. MONTAGEM EXPERIMENTAL

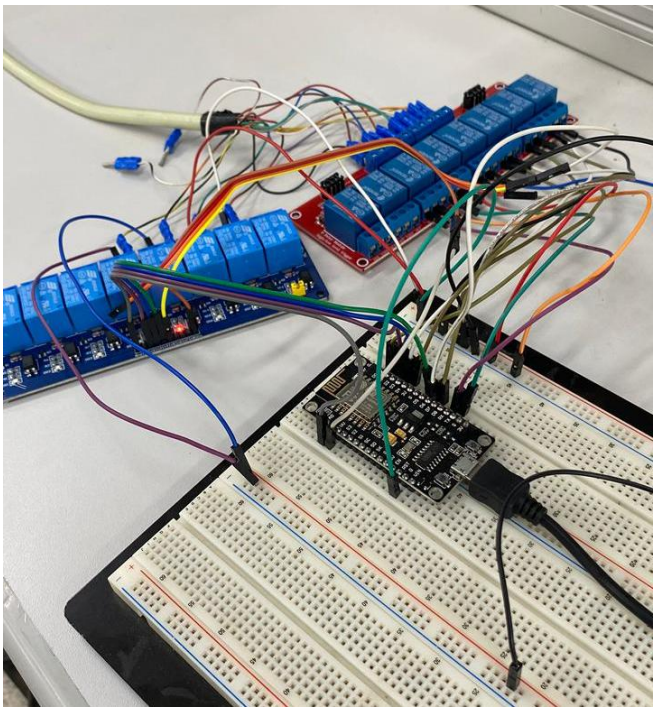
A validação do projeto iniciou-se com a montagem física dos componentes na bancada didática. O microcontrolador ESP8266 e os módulos de relé foram fixados e conectados conforme o esquema elétrico planejado. As válvulas solenoides e os sensores de fim de curso foram conectados aos módulos de interface (relés), garantindo a separação entre os circuitos de potência (24V) e de controle (3.3V/5V).

A disposição dos componentes priorizou a organização dos cabos para evitar interferências e facilitar a manutenção. O Raspberry Pi foi conectado à rede local via cabo Ethernet para garantir a estabilidade do servidor, enquanto o ESP8266 conectou-se via Wi-Fi.

O resultado da montagem dos componentes elétricos e da bancada completa pode ser observado, respectivamente, nas Figuras 28 e 29.

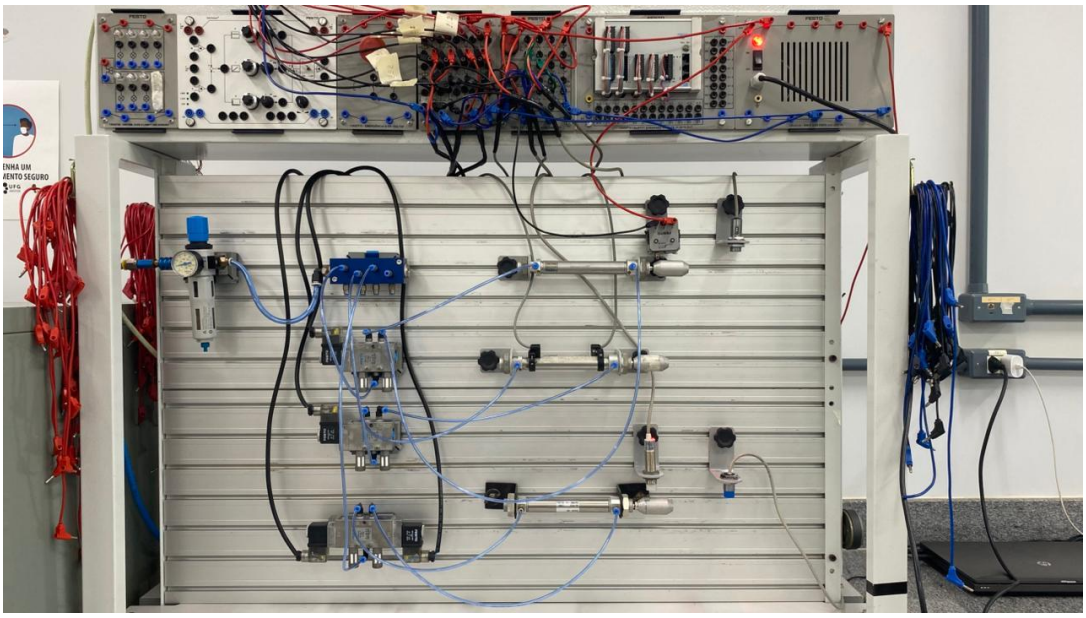
[2] https://drive.google.com/file/d/1jmciT_fpesmQfStz7C4o7drP4mVKqTw3/view?usp=sharing

Figura 28 - Componentes elétricos montados



Fonte: Próprio Autor (2025)

Figura 29 - Bancada eletropneumática montada



Fonte: Próprio Autor (2025)

## 4.2. TESTES DE FUNCIONAMENTO

Após a verificação das conexões elétricas, iniciaram-se os testes de software e comunicação. O sistema foi operado através da interface web, acessada tanto por um

computador quanto por um dispositivo móvel (smartphone), comprovando a responsividade do *design*.

### 4.2.1. MODO DE CONTROLE MANUAL

No primeiro teste, utilizou-se o modo "Controle Manual" para verificar o acionamento individual de cada atuador. Ao clicar nos botões "Avançar" e "Recuar" na interface, observou-se o acionamento dos respectivos relés e o movimento correspondente dos cilindros na bancada.

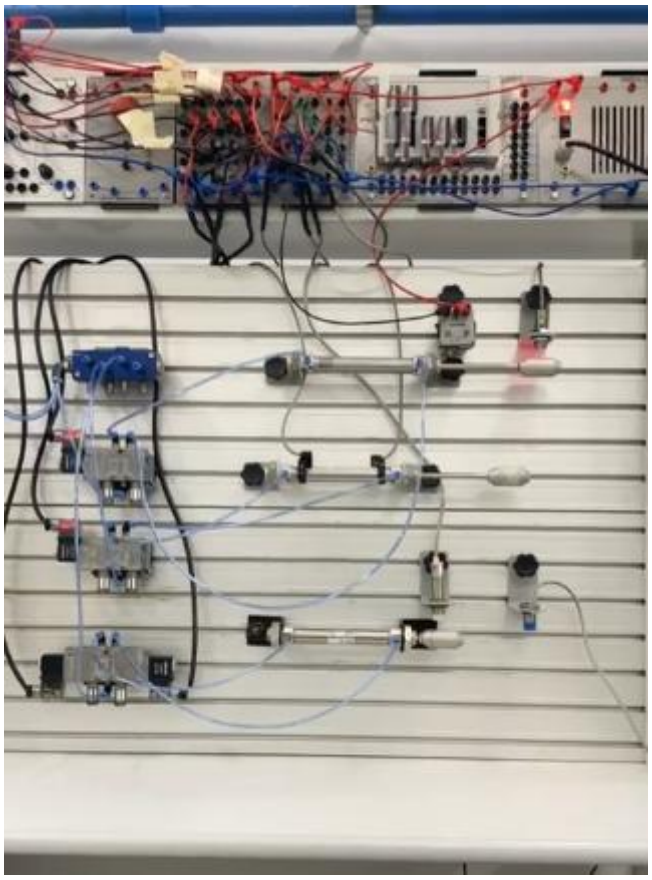
O *feedback* visual na tela, representado pela animação dos cilindros e pela atualização do "Painel de Status", ocorreu de forma adequada com o movimento físico, validando a comunicação de via dupla (comando e retorno de sensor).

### 4.2.2. MODO DE CONTROLE SEQUENCIAL

Para a avaliação da operação do sistema no modo "Controle Sequencial". Foram inseridas sequências complexas de movimentos no campo de texto, utilizando a sintaxe desenvolvida (ex: 1+ 2+ (1- 2-) T2).

O sistema executou os movimentos na ordem correta, respeitando os tempos de pausa programados. O mecanismo de malha fechada demonstrou-se eficaz: o sistema aguardou a confirmação física dos sensores de fim de curso antes de iniciar o passo seguinte, garantindo que a sequência não fosse dessincronizada mesmo diante de variações na velocidade dos atuadores. A Figura 30 ilustra a bancada durante a execução de uma sequência.

Figura 30 - Bancada executando sequência de acionamento



Fonte: Próprio Autor (2025)

### 4.2.3. MONITORAMENTO VIA DASHBOARD

A funcionalidade de monitoramento foi validada através do painel de diagnóstico (modal). Durante a operação, foi possível visualizar em tempo real o status da conexão do servidor, a integridade do *broker* MQTT e a qualidade do sinal Wi-Fi (RSSI) do controlador, permitindo uma análise rápida da saúde do sistema.

Na Figura 31 é apresentada a dashboard no momento em que o servidor está em operação, porém o ESP8266 não está ligado, mostrando que o servidor e o Broker MQTT estão online, mas o Wi-Fi do ESP8266 está desconectado.

Figura 31 - Dashboard em operação


Fonte: Próprio Autor, extraídas do servidor web desenvolvido

## 4.3. ANÁLISE DE DESEMPENHO

Durante os ensaios, o sistema demonstrou alta estabilidade e robustez. A latência entre o envio do comando na interface web e o acionamento da válvula foi satisfatórios para a operação humana.

A utilização do protocolo MQTT com QoS 1 (*Quality of Service* 1) garantiu que nenhum comando fosse perdido, mesmo em momentos de oscilação pontual da rede sem fio. Além disso, a implementação de *WebSockets* eliminou a necessidade de recarregamento da página, proporcionando uma experiência de uso fluida e contínua, similar a de aplicativos nativos.

A solução de acesso remoto via túnel seguro permitiu o controle da bancada através da rede de dados móveis (4G) externa à universidade, comprovando a viabilidade do projeto para aplicações de ensino à distância e teleoperação industrial.

# 5. CONCLUSÃO

O desenvolvimento deste trabalho consolidou a aplicação prática dos conceitos da Indústria 4.0 e da Internet das Coisas (IoT) no contexto educacional e de automação. A implementação do servidor web demonstrou que a integração entre *hardwares* de baixo custo, como o Raspberry Pi e o ESP8266, e protocolos de comunicação robustos, como o MQTT, é capaz de produzir soluções eficientes e acessíveis para o controle remoto de processos industriais em escala educacional.

Conclui-se que o objetivo geral de desenvolver uma ferramenta para o controle e monitoramento remoto de uma bancada eletropneumática foi plenamente alcançado. O sistema permitiu não apenas o acionamento manual dos atuadores, mas também a execução de sequências complexas de automação com *feedback* em tempo real, validando a arquitetura publicador-subscritor proposta.

Para atingir esse resultado, foi necessário aprofundar conhecimentos multidisciplinares que abrangem desde a eletrônica e instrumentação até o desenvolvimento de *software* e configuração de redes. A escolha da linguagem Python e do *framework* Flask para o *backend* mostrou-se acertada, oferecendo a flexibilidade necessária para implementar lógicas de controle avançadas, como o sequenciador em malha fechada, que garantiu a sincronia entre o comando digital e a ação física, mitigando erros operacionais comuns em sistemas de malha aberta.

Os testes realizados comprovaram a estabilidade do sistema. A comunicação via *WebSockets* reduziu a latência de atualização da interface, proporcionando uma experiência de uso fluida e imediata, enquanto o uso do protocolo MQTT com QoS 1 assegurou a entrega confiável dos comandos ao controlador de campo, mesmo diante de oscilações na rede sem fio. Além disso, a viabilidade do acesso externo através de tunelamento seguro (Ngrok) e DDNS demonstrou o potencial da ferramenta para aplicações de ensino à distância (EaD), permitindo que alunos interajam com equipamentos reais de laboratório a partir de qualquer localização.

É importante ressaltar que, embora o projeto seja funcional e robusto, seu escopo é primordialmente didático e de prototipagem. Em um cenário de aplicação industrial crítica, seria necessário incorporar camadas adicionais de segurança, como criptografia TLS/SSL na comunicação MQTT, autenticação robusta de usuários e redundância de hardware, aspectos que não foram o foco principal deste desenvolvimento, mas que são essenciais para a integridade de sistemas em produção.

Em suma, o projeto não apenas entregou um produto funcional, mas serviu como uma prova de conceito da capacidade das tecnologias *open source* em modernizar ambientes de manufatura e ensino, preparando o caminho para futuras inovações na área da Mecânica e Automação.

# 6. TRABALHOS FUTUROS

Embora os objetivos propostos para este trabalho tenham sido plenamente alcançados, o desenvolvimento do sistema revelou diversas oportunidades de expansão e aprimoramento. A arquitetura modular baseada em microsserviços (MQTT e API REST) facilita a incorporação de novas tecnologias sem a necessidade de refazer o núcleo do sistema.

Na sequência, são apresentadas sugestões para a continuidade e evolução desta pesquisa.

## 6.1. INTEGRAÇÃO COM INTELIGÊNCIA ARTIFICIAL (IA)

Uma das evoluções mais promissoras para este projeto é a implementação de algoritmos de Processamento de Linguagem Natural (PLN). Atualmente, o modo sequencial exige que o operador conheça uma sintaxe específica (ex: 1+ T2 1-).

Propõe-se o desenvolvimento de um módulo de IA no servidor *backend*, utilizando bibliotecas como *NLTK*, *Spacy* ou integração com APIs de LLMs (*Large Language Models*), capaz de interpretar comandos em linguagem natural. Isso permitiria que o operador digitasse ou ditasse instruções como: *"Avance o atuador 1, aguarde 3 segundos e depois acione o atuador 2 e 3 simultaneamente"*. O sistema interpretaria a intenção do usuário e converteria automaticamente para a sintaxe de controle da máquina, tornando a operação mais intuitiva e acessível.

## 6.2. IMPLEMENTAÇÃO DE BANCO DE DADOS E HISTÓRICO

A versão atual do sistema opera em memória, o que significa que os registros de eventos e as configurações são perdidos quando o servidor é reiniciado. Sugere-se a integração de um banco de dados relacional (como SQLite ou PostgreSQL) ou de séries temporais (como InfluxDB) para:

- **Persistência de Dados:** Salvar sequências complexas criadas pelos usuários para uso posterior.
- **Histórico de Operação:** Armazenar *logs* detalhados de acionamentos e respostas dos sensores para fins de auditoria e manutenção preditiva.

- **Análise de Desempenho:** Monitorar o tempo de resposta dos atuadores ao longo do tempo para identificar desgaste mecânico.

## 6.3. SEGURANÇA E AUTENTICAÇÃO DE USUÁRIOS

Como o foco inicial foi a viabilidade técnica e didática, o sistema atual é aberto a qualquer usuário na rede. Para uma aplicação em ambiente real ou laboratorial compartilhado, é imprescindível a implementação de camadas de segurança:

- **Autenticação:** Criação de um sistema de *login* com níveis de permissão (Administrador, Operador, Observador).
- **Criptografia:** Implementação de SSL/TLS tanto na comunicação HTTP (HTTPS) quanto no protocolo MQTT (MQTTS), protegendo os dados contra interceptação.

## 6.4. INTERFACE GRÁFICA DE PROGRAMAÇÃO (LOW-CODE)

Para facilitar o aprendizado de alunos iniciantes em lógica de automação, sugere-se a evolução da interface do "Modo Sequencial" para um ambiente visual de programação em blocos, similar ao *Scratch* ou *Google Blockly*. Isso permitiria que o usuário arrastasse e conectasse blocos lógicos ("Avançar", "Esperar", "Repetir") para construir a sequência visualmente, eliminando a possibilidade de erros de sintaxe na digitação dos comandos.

# REFERÊNCIAS

ADAFRUIT. Raspberry Pi 2, Model B. Disponível em: https://cdn-shop.adafruit.com/pdfs/raspberrypi2modelb.pdf. Acesso em: 2025.

BALBINOT, Alexandre; BRUSAMARELLO, Valner João. Instrumentação e Fundamentos de Medidas. v. 1. 2. ed. Rio de Janeiro: LTC, 2011.

BOLLMANN, A. Fundamentos da Automação Industrial Pneutrônica: Projeto de comandos binários eletropneumáticos. São Paulo: ABHP, 1997.

BONACORSO, Nelso Gauze; NOLL, Valdir. Automação Eletropneumática. 12. ed. São Paulo: Érica, 2013.

DUCKETT, Jon. Web Design with HTML, CSS, JavaScript and jQuery Set. 1. ed. Indianapolis: Wiley, 2014.

FIALHO, Arivelto Bustamante. Automação Pneumática: Projetos, Dimensionamento e Análise de Circuitos. 7. ed. São Paulo: Érica, 2011.

FLANAGAN, David. JavaScript: The Definitive Guide. 7. ed. Sebastopol: O'Reilly Media, 2020.

GILCHRIST, Alasdair**.** Industry 4.0: The Industrial Internet of Things. 1. ed. New York: Apress, 2016.

GRINBERG, Miguel. Flask Web Development: Developing Web Applications with Python. 2. ed. Sebastopol: O'Reilly Media, 2018.

HANES, David; SALGUEIRO, Gonzalo; GROSSETETE, Patrick; BARTON, Rob; HENRY, Jerome. IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things. Indianapolis: Cisco Press, 2017.

HIVEMQ. MQTT Essentials. Disponível em: https://www.hivemq.com/mqtt-essentials/. Acesso em: 2025.

KUROSE, James F.; ROSS, Keith W. Redes de Computadores e a Internet: Uma Abordagem Top-Down. 6. ed. São Paulo: Pearson, 2013.

MAKERHERO. *Como funciona o sensor indutivo*. Disponível em: https://www.makerhero.com/blog. Acesso em: 2025.

MONK, Simon. Programming the ESP8266: Build Projects with the Arduino IDE. New York: McGraw-Hill Education, 2016.

NATALE, Ferdinando. Automação Industrial. 10. ed. São Paulo: Érica, 2011.

OASIS. MQTT Version 3.1.1. OASIS Standard, 2014. Disponível em: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Acesso em: 2025.

PARKER HANNIFIN. Tecnologia Pneumática Industrial. Jacareí: Parker Hannifin Ind. Com. Ltda. Training, 2011.

PAVANI, S. A. Comandos Pneumáticos e Hidráulicos. Santa Maria: e-Tec Brasil, 2011.

PAHC AUTOMAÇÃO. Entenda o funcionamento dos sensores ópticos. 2021. Disponível em: https://pahcautomacao.com.br. Acesso em: 2025.

RASPBERRY PI FOUNDATION. Raspberry Pi 2 Model B. Disponível em: https://www.raspberrypi.com/products/raspberry-pi-2-model-b/. Acesso em: 2025.

ROSÁRIO, João Maurício. Princípios de Mecatrônica. São Paulo: Pearson Prentice Hall, 2005.

TECNOTRON. Sensores magnéticos Reed Switch: Detectando com Simplicidade. Disponível em: https://tecnotron.com.br. Acesso em: 2025.

THOMAZINI, Daniel; ALBUQUERQUE, Pedro U. B. Sensores Industriais: Fundamentos e Aplicações. 8. ed. São Paulo: Érica, 2011.

UPTON, Eben; HALFACREE, Gareth. Raspberry Pi User Guide. 3. ed. Indianapolis: John Wiley & Sons, 2014.

W3C. HTML & CSS Standards. Disponível em: https://www.w3.org/standards/webdesign/htmlcss. Acesso em: 2025.

WANG, V.; SALIM, F.; MOSKOVITS, P. The Definitive Guide to HTML5 WebSocket. Apress, 2013.