



**MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO
TOCANTINS - *CAMPUS* PALMAS
BACHARELADO EM ENGENHARIA ELÉTRICA**

LUCAS AUGUSTO NUNES DE BARROS

**PROJETO HOLHOOJA: UM SISTEMA IOT PARA AUTOMAÇÃO
LABORATORIAL E CONTROLE DE ACESSO**

**PALMAS - TO
2024**

LUCAS AUGUSTO NUNES DE BARROS

**PROJETO HOLHOOJA: UM SISTEMA IOT PARA AUTOMAÇÃO
LABORATORIAL E CONTROLE DE ACESSO**

Trabalho de Conclusão de Curso apresentado como requisito para obtenção do Título de Bacharel em Engenharia Elétrica do Curso de Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia do Tocantins, *Campus* Palmas.

Orientador: Prof. Marcos Balduino de Alvarenga, Dr.

Coorientador: Prof. Maxwell Moura Costa, Dr.

**PALMAS - TO
2024**

Dados Internacionais de Catalogação na Publicação (CIP)
Bibliotecas do Instituto Federal do Tocantins

B277p Barros, Lucas Augusto Nunes de
 Projeto Holhooja: Um sistema IOT para automação laboratorial e
 controle de acesso / Lucas Augusto Nunes de Barros. – PALMAS,
 TO, 2024.
 132 p. : il. color.

 Trabalho de Conclusão de Curso (Bacharelado em Engenharia
 Elétrica) – Instituto Federal de Educação, Ciência e Tecnologia do
 Tocantins, Campus Palmas, PALMAS, TO, 2024.

 Orientador: Dr. Marcos Balduino de Alvarenga
 Coorientador: Dr. Maxwell Moura Costa

 1. Internet das coisas. 2. Automação de ambientes. 3. Controle de
 acesso. I. Alvarenga, Marcos Balduino de. II. Costa, Maxwell Moura.
 III. Título.

CDD 621

A reprodução total ou parcial, de qualquer forma ou por qualquer meio, deste documento é autorizada para fins de estudo e pesquisa, desde que citada a fonte.

Elaborado pelo sistema de geração automática de ficha catalográfica do IFTO com os dados fornecidos pelo(a) autor(a).

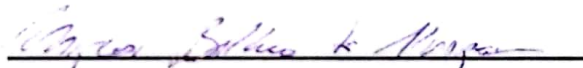
LUCAS AUGUSTO NUNES DE BARROS


**PROJETO HOLHOOJA: UM SISTEMA IOT PARA AUTOMAÇÃO
LABORATORIAL E CONTROLE DE ACESSO**

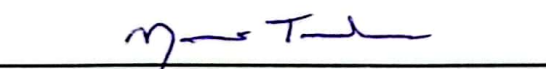
Trabalho de Conclusão de Curso apresentado
como requisito para obtenção do Título de
Bacharel em Engenharia Elétrica do Curso
de Engenharia Elétrica do Instituto Federal
de Educação, Ciência e Tecnologia do Tocan-
tins, *Campus Palmas*

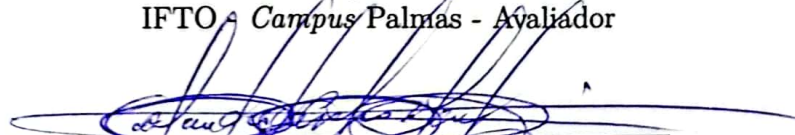
Data da aprovação : 17/09/24

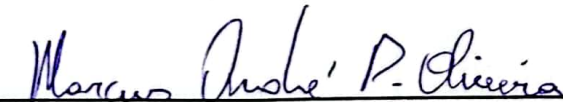
Banca Examinadora:


Prof. Marcos Balduino de Alvarenga, Dr.
IFTO - *Campus Palmas* - Orientador


Prof. Maxwell Moura Costa, Dr.
IFTO - *Campus Palmas* - Coorientador


Prof. Márcio Augusto Tamashiro, Dr.
IFTO - *Campus Palmas* - Avaliador


Prof. Cláudio de Castro Monteiro, Dr.
IFTO - *Campus Palmas* - Avaliador


Prof. Marcus André Pereira Oliveira, Dr
IFTO - *Campus Palmas* - Avaliador

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão a todos que contribuíram de várias maneiras para a realização deste trabalho. Seus esforços foram inestimáveis e tornaram possível a conclusão deste artigo.

Primeiramente, quero agradecer a minha família pelo suporte em todos os momentos de dificuldades, por fornecer recursos essenciais como atenção, compreensão e amor. Sem o suporte deles, este projeto, e muitos outros, não seriam possíveis.

Meu apreço para o professor Dr. Marcos Balduino de Alvarenga, que desempenhou um papel fundamental na minha jornada acadêmica, me orientando e auxiliando em alguns projetos desde o início da graduação e com quem eu tive o prazer de conviver durante bastante tempo dentro do meio acadêmico, sem seus conselhos, paciência e orientação ao longo dos anos este trabalho não seria o que é.

Gostaria de expressar meus agradecimentos também ao Professor Dr. Maxwell Moura Costa por sua paciência, dedicação e constante disponibilidade. Seus ensinamentos não se limitaram ao ambiente acadêmico, estendendo-se para além da sala de aula, contribuindo não apenas para minha jornada acadêmica, mas também para meu desenvolvimento como pessoa. Agradeço pelo privilégio de aprender com alguém tão dedicado e inspirador.

Também sou grato aos membros das equipes de pesquisa as quais fiz parte, todos os que em algum momento me auxiliaram de alguma forma e contribuíram para a realização deste projeto. Apesar de estar desenvolvendo este trabalho de forma individual o trabalho em equipe desempenhou um papel significativo em seu sucesso.

Por fim, agradeço à comunidade acadêmica e a todos que participaram das discussões construtivas e revisões críticas deste trabalho. Suas sugestões contribuíram para a qualidade e rigor deste artigo.

Em suma, este trabalho é o resultado de um esforço coletivo e de apoio que recebi de muitas pessoas, professores e colegas de curso. Expresso minha gratidão profunda a todos vocês.

*“Complicações surgiram, persistiram e
foram superadas”*

Capitão Jack Sparrow

RESUMO

O projeto concentra suas investigações na concepção de uma solução automatizada que aborde as tecnologias da Internet das Coisas (IoT) e na elaboração de um sistema online direcionado ao controle de acesso de ambientes laboratoriais. Este protótipo do sistema integrado foi desenvolvido para o Laboratório de Práticas Autônomas (LPA) e tem como principal atribuição a administração do registro de acesso do LPA, promovendo aprimoramentos em termos de praticidade e controle do uso, ao mesmo tempo em que realiza a verificação do tempo de permanência de cada um dos alunos cadastrados, facilitando a geração de certificados de horas complementares. Paralelamente ao sistema de controle de acesso, o escopo do projeto abrange a implementação de um sistema de automação no laboratório. Este sistema, em conjunto com o controle de acesso, visa disponibilizar a abertura do ambiente, gerenciar os dispositivos de ar condicionado e iluminação, fornecendo informações em tempo real sobre as condições do laboratório.

Palavras-chaves: automação de ambientes, controle de acesso, internet das coisas.

ABSTRACT

The project focuses its research on the design of an automated solution that addresses Internet of Things (IoT) technologies and the development of an online system aimed at controlling access to laboratory environments. This integrated system prototype was developed for the Autonomous Practices Laboratory (LPA) and its main purpose is to manage the access logs of the LPA, improving convenience and control over its use, while also tracking the duration of each registered student's stay, facilitating the generation of certificates for complementary hours. Alongside the access control system, the project scope includes the implementation of an automation system in the laboratory. This system, combined with the access control, aims to enable the opening of the lab, manage air conditioning and lighting devices, and provide real-time information about the laboratory's conditions.

Keywords: access control, environment automation, internet of things.

LISTA DE ILUSTRAÇÕES

Figura 1 – Possibilidades da automação residencial.	16
Figura 2 – Estrutura do servidor	22
Figura 3 – Estrutura de comunicação do protocolo MQTT	24
Figura 4 – Esquema QoS	24
Figura 5 – Microcontrolador ESP32-S3	25
Figura 6 – Diagrama de pinagem do módulo ESP32-S3-WROOM-1.	27
Figura 7 – Kit de Desenvolvimento DevKit C1 ESP-S3-WROOM-1 N16R8.	28
Figura 8 – Sensor de presença SR602.	30
Figura 9 – Sensor DHT11	32
Figura 10 – Sensor SCT13	33
Figura 11 – Esquema do sistema proposto.	36
Figura 12 – Diagrama de ligação do sensor DHT11.	37
Figura 13 – Diagrama de ligação do sensor SR602.	38
Figura 14 – Diagrama de ligação do LED infravermelho utilizando o transistor BC-547.	39
Figura 15 – Diagrama de ligação do LED infravermelho utilizando o transistor BC-547.	39
Figura 16 – Circuito proposto para realizar as medições do SCT.	40
Figura 17 – Circuito de controle da porta.	41
Figura 18 – Estrutura básica da aplicação Flask proposta.	42
Figura 19 – Estrutura do sistema de <i>login</i>	42
Figura 20 – Estrutura de comunicação MQTT.	43
Figura 21 – Sensor MQ135 montado na protoboard	45
Figura 22 – Dispositivos montados em <i>protoboard</i> para a validação	47
Figura 23 – Dados dos sensores apresentados no monitor serial da IDE.	47
Figura 24 – Dados dos sensores apresentados na interface da aplicação.	47
Figura 25 – Tela de <i>login</i> da aplicação.	50
Figura 26 – Diagrama de Entidade de Relacionamento.	52
Figura 27 – Estrutura do Usuário Comum.	53
Figura 28 – Estrutura do Usuário Administrador.	53
Figura 29 – Estrutura de Comunicação MQTT.	58
Figura 30 – Dispositivos soldados	61
Figura 31 – Circuito emissor IR durante teste após confecção da placa	62
Figura 32 – Modelo proposto para confecção do protótipo	62
Figura 33 – Vista superior da placa principal	63
Figura 34 – Vista inferior da placa principal	63

Figura 35 – Placa final com sensores conectados para validação final dos circuitos. .	63
Figura 36 – Circuito emissor IR pronto para ser instalado no LPA	64
Figura 37 – Sensor MQ135 instalado na eletrocalha do LPA	64
Figura 38 – Sensor SR602 instalado próximo a porta do LPA	64
Figura 39 – Sensor SCT13 instalado dentro do módulo condensador	65
Figura 40 – Circuito instalado em série com o SCT13	65
Figura 41 – Conectores de alimentação dos sensores	66
Figura 42 – Circuito principal instalado no LPA	66
Figura 43 – Solicitações pendentes.	67
Figura 44 – <i>Dashboard</i> do usuário administrador	67
Figura 45 – Painel de controle dos usuários	68
Figura 46 – Página de edição do usuário	68
Figura 47 – Painel de controle do laboratório	69
Figura 48 – Lista de presentes no laboratório	69
Figura 49 – Relé novo	71
Figura 50 – Medição de resistência da bobina	72
Figura 51 – Valor de resistência da bobina	72

LISTA DE TABELAS

Tabela 1	–	Comparativo entre microcontroladores similares.	26
Tabela 2	–	Comparativo entre sensores HC-SR501, SR602 e SR505	29
Tabela 3	–	Especificações do sensor SR602.	30
Tabela 4	–	Tipos de sensores de gás.	31
Tabela 5	–	Especificações do sensor MQ-135.	31
Tabela 6	–	Especificações do sensor DHT11.	32
Tabela 7	–	Especificações do relé JQC3F-03VDC-C.	34
Tabela 8	–	Campos da Tabela User	52
Tabela 9	–	Campos da Tabela Acesso	54
Tabela 10	–	Campos da Tabela Sensores	54

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
CPU	<i>Central Processing Unit</i>
EPE	Empresa de Pesquisa Energética
GPIO	<i>General Purpose Input/Output</i> (Entrada/Saída de Propósito Geral)
IDE	<i>Integrated Development Environment</i> (Ambiente de Desenvolvimento Integrado)
IEEE	<i>Institute of Electrical and Electronics Engineers</i> (Instituto de Engenheiros Eletricistas e Eletrônicos)
IFTO	Instituto Federal de Ciência e Tecnologia do Tocantins
INMET	Instituto Nacional de Meteorologia
IoT	<i>Internet of Things</i> (Internet das Coisas)
IR	<i>Infrared</i> (Infravermelho)
I2C	<i>Inter-Integrated Circuit</i>
LDR	<i>Light Dependent Resistor</i> (Resistor Dependente de Luz)
LPA	Laboratório de Práticas Autônomas
LTS	<i>Long Term Supported</i>
MCU	<i>Microcontroller Unit</i>
MQTT	<i>Message Queuing Telemetry Transport</i> (Transporte de Filas de Mensagem de Telemetria)
M2M	<i>Machine to machine</i>
ONS	Operador Nacional do Sistema
PIR	<i>Passive Infrared</i> (Infravermelho Passivo)
PPM	Partículas Por Milhão
PSRAM	<i>Pseudo Static Random Access Memory</i> (Memória Pseudo-Estática de Acesso Aleatório)

PWM	<i>Pulse Width Modulation</i> (Modulação por Largura de Pulso)
QoS	<i>Quality of Service</i> (Qualidade de Serviço)
SGBD	Sistema Gerenciador de Banco de Dados
SPI	<i>Serial Peripheral Interface</i> (Interface Periférica Serial)
SQL	<i>Structured Query Language</i> (Linguagem de Consulta Estruturada)
SSL	<i>Secure Sockets Layer</i>
TLS	<i>Transport Layer Security</i>
UART	<i>Universal Asynchronous Receiver / Transmitter</i> (Receptor / Transmissor Assíncrono Universal)
URL	<i>Uniform Resource Locator</i>
USB	<i>Universal Serial Bus</i>
WSGI	<i>Web Server Gateway Interface</i>

SUMÁRIO

1	INTRODUÇÃO	15
1.1	Contextualização	15
1.2	Problemas da Pesquisa	16
1.3	Justificativa	17
1.4	Objetivos	17
1.4.1	Objetivo geral	17
1.4.2	Objetivos específicos	17
2	FUNDAMENTAÇÃO TEÓRICA	19
2.1	A Aplicação <i>Web</i>	19
2.1.1	Python e o Micro <i>Framework</i> Flask	19
2.1.2	Tecnologias <i>Web</i>	20
2.1.2.1	A linguagem HTML	20
2.1.2.2	CSS - O <i>Framework</i> Bulma	20
2.1.2.3	A linguagem Javascript	21
2.1.3	O Servidor	21
2.1.3.1	Servidor <i>Web</i> - Nginx	21
2.1.3.2	Servidor WSGI - Gunicorn	22
2.1.3.3	O Banco de Dados - MariaDB	23
2.2	Protocolo MQTT	23
2.3	O Microcontrolador Esp 32	25
2.3.1	Arquitetura do Microcontrolador	26
2.3.2	O Ambiente de Desenvolvimento	28
2.3.3	Sensores	29
2.3.3.1	Sensores de Presença <i>PIR</i>	29
2.3.3.2	Sensores de gás	31
2.3.3.3	Sensor de temperatura e umidade	31
2.3.3.4	LDR - <i>Light Dependent Resistor</i>	32
2.3.3.5	Sensor de corrente elétrica	32
2.3.4	Atuadores	34
2.3.4.1	Relé	34
2.3.4.2	Led infravermelho	34
3	PROPOSTA DO PROJETO	36
3.1	Projeto de <i>Hardware</i>	37
3.1.1	Monitoramento da Umidade e Temperatura	37

3.1.2	Monitoramento de Gás	38
3.1.3	Sensor de Presença	38
3.1.4	Circuito de Comunicação Infravermelho	39
3.1.5	LDR - <i>Light Dependent Resistor</i>	39
3.1.6	Monitoramento da Corrente Elétrica do Ar Condicionado	40
3.1.7	Controle da Fechadura Elétrica	40
3.2	Projeto de <i>Software</i>	41
3.2.1	Aplicação	41
4	METODOLOGIA	44
4.1	<i>Hardware</i>	44
4.1.1	Validação do sensores e atuadores	44
4.1.1.1	MQ135	44
4.1.1.2	DHT11	45
4.1.2	LDR - <i>Light Dependent Resistor</i>	45
4.1.2.1	SR602	45
4.1.2.2	SCT 13	45
4.1.2.3	Atuadores: Relé e Led IR	46
4.1.3	Modelo de baixa fidelidade	46
4.2	<i>Software</i>	48
4.2.1	Aplicação Flask: Estrutura, funcionalidades e configurações	48
4.2.1.1	Estrutura da Aplicação	48
4.2.1.2	O Núcleo da Aplicação	49
4.2.1.3	O Banco de Dados	51
4.2.1.4	<i>WebSocket</i>	54
4.2.1.5	Arquivos Complementares	56
4.2.2	Comunicação via MQTT	58
4.2.2.1	Tópicos da Aplicação	58
4.2.3	<i>Firmware</i> do ESP32.	59
5	RESULTADOS E DISCUSSÕES	61
5.1	Modelo final do protótipo	61
5.2	Instalação do sistema no Laboratório de Práticas Autônomas	63
5.3	Validação da Aplicação Python	66
5.4	<i>Firmware</i> do ESP32	69
6	CONSIDERAÇÕES FINAIS	71
6.1	Problemáticas	71
6.2	Trabalho Futuros	72

	REFERÊNCIAS	74
	APÊNDICE A – <i>FIRMWARE</i> DO ESP32	77
A.1	main.ino	77
A.2	holhooja.ino	81
	APÊNDICE B – NÚCLEO DA APLICAÇÃO PYTHON	90
B.1	__init__.py	90
B.2	sockets.py	91
B.3	routes.py	93
B.4	models.py	100
B.5	forms.py	102
	APÊNDICE C – CÓDIGOS COMPLEMENTARES DA APLI- CAÇÃO PYTHON	105
C.1	requirements.txt	105
C.2	mqtt.py	105
	APÊNDICE D – <i>TEMPLATES</i> HTML	110
D.1	base.html	110
D.2	dash_lab.html	114
D.3	editar_senha.html	117
D.4	editar_usuario.html	118
D.5	index_admin.html	122
D.6	index.html	127
D.7	login.html	128
D.8	register.html	128
D.9	user.html	128
D.10	usuários.html	129

1 INTRODUÇÃO

1.1 Contextualização

A automação de ambientes não industriais ganha cada vez mais destaque como uma forma altamente eficiente de otimizar processos e melhorar a gerência de pessoas, equipamentos e espaços. Embora a automação tenha se consolidado primeiramente em ambientes industriais, a adoção em outros setores, como residencial e comercial, tem ocorrido mais lentamente, devido a fatores como custos iniciais elevados e a necessidade de mão de obra especializada ([MURATORI, 2016](#)).

Nos últimos anos, a disseminação de tecnologias relacionadas a Internet das Coisas (IoT - *Internet of Things*) tem facilitado a implementação de sistemas de automação em diversos ambientes fora da indústria. Esses sistemas são conectados e permitem que dispositivos se comuniquem de forma independente, proporcionando maior controle e capacidade de intervenção em tempo real sobre os recursos e funcionalidades ([HOPPEN, 2016](#)).

Neste contexto, a IoT e a automação de ambientes se apresentam como uma solução estratégica para a implementação de sistemas de controle de acesso. A gerência do acesso de ambientes é uma preocupação constante em instituições públicas e privadas, especialmente em estabelecimentos que abrigam equipamentos especializados e de alto valor. Desde laboratórios de pesquisa, depósitos e salas de informática, todos esses ambientes demandam um sistema de controle de acesso eficiente, capaz de monitorar o fluxo de pessoas e impedir a entrada de pessoal não autorizado.

Para que um sistema de controle de acesso automatizado seja eficaz, ele deve ser robusto e confiável, especialmente em ambientes com alta criticidade. A interface deve ser amigável e intuitiva, garantindo que pessoas de diferentes formações acadêmicas e idades possam utilizá-la com facilidade. Ademais, o sistema deve ser pouco invasivo, evitando alterações significativas na estrutura do local e permitindo uma instalação adaptável a diversos ambientes ([LOPES, 2020](#)). A figura 1 mostra algumas possibilidades fornecidas pela automação.

Figura 1 – Possibilidades da automação residencial.



Fonte: (SOARES, 2023)

1.2 Problemas da Pesquisa

Nos últimos anos, observou-se no mercado brasileiro um incremento no consumo de eletricidade (EPE, 2023). Cruzando dados da EPE e da ONS, é possível concluir que no ano de 2022 o Brasil consumiu um total de 611 GWh, onde 36,22% dessa energia foi devido a atividade industrial, enquanto que 48,15%, quase metade, foi proveniente dos setores comerciais e residenciais, o que torna evidente o potencial econômico que é possível atingir quando esses dois setores alcançarem, pelo menos em sua maioria, o patamar de alta eficiência energética.

Apesar dos benefícios associados aos projetos de automação, diversas barreiras têm dificultado sua disseminação em ambientes não industriais. Entre algumas dificuldades encontradas destacam-se a escassez de informação, falta de conscientização por parte da população, bem como os custos elevados. Esses obstáculos têm contribuído para limitar a adoção generalizada de iniciativas eficientes no uso da energia (JANNUZZI, 2001).

Os sistemas de controle de acesso manual enfrentam diversos problemas como a alta suscetibilidade a erros, falsificação de identidades e dificuldades na auditoria dos registros de cada ambiente. A automatização desses sistemas oferece vantagens significativas, incluindo a redução de falhas humanas, maior precisão na verificação de identidades e registros de acessos, além de permitir a implementação de métodos de autenticação avançados, como biometria e cartões RFID, que aumentam a segurança e a eficiência na gestão de acessos.

1.3 Justificativa

No Instituto Federal do Tocantins, *campus* Palmas, o controle de acesso às chaves dos laboratórios do bloco 9, onde se localiza a coordenação da área de indústria é feito de forma manual em uma folha de papel.

Sem tecnologias de automação os ambientes coletivos, como são os espaços educacionais (salas e laboratórios), enfrentam desafios de segurança. A ausência desses sistemas resultam em baixa integridade dos registros, deixando os administradores sem amparo necessário para auditar e intervir, além da incapacidade de integração com outros sistemas existentes.

O controle de acesso no âmbito acadêmico é de especial importância dada a importância desse tipo de instituição para a sociedade e o investimento público depositado nesses ambientes, que muitas vezes possuem equipamentos disponíveis para o desenvolvimento das atividades acadêmicas. Vale pontuar também que algumas salas e laboratórios possuem exigências de segurança e/ou pessoal autorizado para monitorar o uso do ambiente. Esses e outros requisitos podem ser verificados por um sistema automatizado devidamente implementado.

Sistemas de automação integrados à IoT permitem acompanhar e interagir em tempo real com o ambiente, tornando a segurança e o controle mais robustos, bem como proporcionam rastreabilidade e praticidade aos usuários. (OLIVEIRA, 2019).

Medidas como essa não apenas incrementam a segurança, mas também melhoram a imagem institucional, possibilitando o acesso a novas tecnologias e expondo à comunidade interna soluções inovadoras, otimizando a gestão de recursos e infraestrutura da instituição, promovendo um ambiente mais seguro e produtivo.

1.4 Objetivos

1.4.1 Objetivo geral

Implementar um sistema integrado que permita automatizar os condicionadores de ar e a iluminação do ambiente, bem como realizar o controle de acesso via sistema de autenticação online do Laboratório de Práticas Autônomas.

1.4.2 Objetivos específicos

- Monitorar o status do ar condicionado;
- Automatizar o acesso ao laboratório;
- Automatizar as funcionalidades do ar condicionado;

-
- Implementar um sistema de autenticação online;
 - Registrar em tempo real o acesso ao laboratório;
 - Realizar a contagem de horas por usuário;
 - Disponibilizar o sistema de código aberto para implementação em outros ambientes.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 A Aplicação Web

A arquitetura do *framework* Flask oferece ao desenvolvedor a flexibilidade de escolher qual ORM (Object-Relational Mapping) utilizar para manipulação de dados. De acordo com (GARBADE, 2007), a biblioteca SQLAlchemy é a mais amplamente utilizada, exceto em casos onde se opta pelo *framework* Django, que possui seu próprio ORM. Alternativamente, o desenvolvedor pode utilizar outras bibliotecas, como MongoDB ou SQLite.

No que diz respeito aos formulários, biblioteca WTForms é uma das escolhas mais populares, de acordo com (LEIFER, 2015). Ela se destaca por sua simplicidade e documentação abrangente, abstraindo a manipulação de diversos campos e elementos, facilitando o desenvolvimento. Sem a utilização dessa biblioteca, (PICARD, 2016) observa que o tratamento dos campos nos formulários seria provavelmente realizado diretamente nas views. Essa prática possui algumas desvantagens importantes como gerar inconsistências, a manutenção do código torna-se mais difícil, além da falta de reutilização de componentes, que aumenta o esforço de desenvolvimento, pois não há uma maneira eficiente de reaproveitar códigos de validação ou manipulação de formulários.

Para a manipulação de templates, o Flask exige a presença do Jinja, um *template engine* para Python que, conforme destacado na documentação oficial, é necessário para a execução do *framework* (RONACHER, 2010). Embora seja possível integrar outros motores de templates, o Jinja é indispensável para o bom funcionamento do Flask.

Adicionalmente, a biblioteca Flask-User foi utilizada como base para o desenvolvimento da solução de verificação das credenciais, fornecendo um conjunto completo de ferramentas para autenticação segura no Flask. Essa biblioteca integra-se de maneira eficaz com as demais ferramentas populares mencionadas, como o SQLAlchemy, WTForms e Jinja, tornando-se uma escolha robusta para aplicações Flask (RONACHER, 2010).

Por fim, outras ferramentas, como o *Werkzeug*, foram mantidas em suas configurações padrão, visto que já atendem às demandas do projeto sem a necessidade de customizações adicionais. Cada componente selecionado apresenta características que se alinham ao propósito da aplicação.

2.1.1 Python e o Micro *Framework* Flask

Python é uma linguagem de programação de alto nível, conhecida por sua sintaxe clara e legível, que facilita o desenvolvimento e a manutenção de aplicações. No contexto do desenvolvimento *Web*, Python é utilizado devido a *frameworks* como Django e Flask,

que permitem a criação eficiente de *sites* e aplicações *Web* (RONACHER, 2010).

Um *framework* é um conjunto de códigos desenvolvido para facilitar o desenvolvimento de aplicações específicas. A sua principal função é simplificar toda a desenvolvimento, concentrando ferramentas de forma a otimizar esse processo. O Flask, utilizado no desenvolvimento deste projeto, é um micro *framework* Python que se destaca pela leveza e simplicidade. Ele fornece os componentes essenciais para o desenvolvimento de aplicações *Web* baseadas em HTML, e é altamente extensível, contando com um grande acervo de bibliotecas graças a vasta comunidade ativa, o que fornece um amplo espectro de recursos adicionais necessários à cada caso (RONACHER, 2010).

O Flask é baseado na biblioteca *WSGI - Web Server Gateway Interface - Werkzeug*, que é uma biblioteca que fornece a base para o desenvolvimento de aplicações *Web* utilizando Python. A biblioteca *Werkzeug* facilita a interface entre servidores *Web* e aplicações Python, permitindo que o Flask gerencie solicitações HTTP e respostas de maneira eficiente. Essa integração garante ao Flask um desempenho robusto e compatível com os padrões da *Web*.

A flexibilidade da linguagem Python, combinada com o modelo de desenvolvimento simplificado do Flask, facilita a criação de aplicações *Web*. O uso desse *framework* torna o processo de desenvolvimento mais ágil. Integrando funcionalidades vindas de bibliotecas de terceiros é possível acessar o banco de dados, validação de formulários e diversas outras funcionalidades, enriquecendo a aplicação (GRINBERG, 2024).

2.1.2 Tecnologias Web

2.1.2.1 A linguagem HTML

As principais características do HTML são a simplicidade e flexibilidade. A linguagem permite a inserção de diversos tipos de conteúdo, como textos, imagens, links, vídeos e formulários, de forma estruturada e semântica.

A linguagem ainda é compatível com a maioria dos navegadores modernos, garantindo que as páginas desenvolvidas com essa tecnologia possam ser acessadas pela maior parte dos dispositivos. Suas aplicações vão desde a criação de pequenos *sites* até complexas aplicações *Web*.

2.1.2.2 CSS - O Framework Bulma

Bulma é um *framework* CSS de código aberto e gratuito, criado por Jeremy Thomas e lançado em 2016 sob a licença MIT. Ele fornece uma coleção de componentes prontos para uso, facilitando a criação de interfaces responsivas. A simplicidade do *framework* permite que até mesmo usuários sem conhecimento aprofundado de CSS possam utilizá-

lo, pois exige um mínimo de código HTML para alcançar uma estrutura visualmente agradável e funcional (BULMA, 2020).

O *framework* oferece ampla variedade de componentes prontos para serem integrados na aplicação. Esta biblioteca de componentes permite que os desenvolvedores criem interfaces complexas com um esforço mínimo, garantindo ao mesmo tempo uma aparência coesa e profissional, muito indicado para projetos simples e de prototipagem. A utilização de classes pré-definidas e a adoção de uma sintaxe clara e intuitiva tornam o Bulma uma ferramenta acessível para desenvolvedores de todos os níveis.

2.1.2.3 A linguagem Javascript

JavaScript é uma linguagem de programação amplamente utilizada para adicionar dinamismo às páginas *Web*. Permitindo aos desenvolvedores manipular elementos e eventos, bem como transmitir informações de forma bilateral entre uma página e seu servidor. Possui como principais características a flexibilidade e capacidade de integração com diversas outras tecnologias, sendo amplamente utilizada em desenvolvimento *Web*.

Em muitas aplicações essa linguagem é implementada sob o paradigma da programação orientada a eventos, um estilo de programação onde o fluxo de execução do código é ditado por eventos, como cliques no mouse ou em teclas específicas. Esses eventos desencadeiam a execução de funções em segundo plano durante a exibição da aplicação ao usuário. Por isso o uso de javascript permite que os desenvolvedores tornem as páginas *Web* interativas e dinâmicas, respondendo em tempo real às dinâmicas dos usuários e servidores sem a necessidade de recarregar a página.

2.1.3 O Servidor

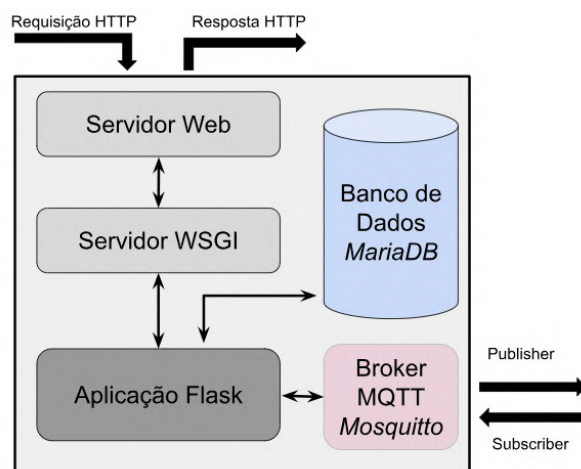
O fluxo de operação básico de um servidor responsável por disponibilizar uma aplicação Python, como as construídas com Flask, envolve uma série de etapas que são executadas sempre que uma solicitação é enviada. Estes processos incluem a interação entre o servidor *Web*, o servidor WSGI e a própria aplicação.

O servidor implementado neste projeto está organizado como representado no diagrama da figura 2:

2.1.3.1 Servidor *Web* - Nginx

O servidor *Web* é responsável por gerenciar requisições HTTP provenientes de clientes e responder a essas requisições entregando recursos estáticos ou dinâmicos, como páginas HTML, arquivos CSS, etc. Ele gerencia as conexões com os clientes, implementa protocolos de segurança bem como realiza o gerenciamento das requisições, e pode também

Figura 2 – Estrutura do servidor



Fonte: Autor (2024)

ser responsável por realizar o balanceamento de carga, distribuindo as requisições entre várias instâncias da aplicação para garantir eficiência e alta disponibilidade.

O Nginx é um servidor *Web* de alta performance, atuando também como servidor proxy reverso e balanceador de carga. Possui alta eficiência e capacidade de lidar com um grande número de conexões simultâneas. O Nginx é comumente escolhido para ambientes de alta demanda por suportar padrões de segurança como SSL/TLS além de poder ser configurado para trabalhar em conjunto com servidores de aplicação WSGI, para entregar conteúdo dinâmico gerado por frameworks como Flask.

2.1.3.2 Servidor WSGI - Gunicorn

Servidores WSGI atuam como na camada intermediária entre o servidor *Web* e a aplicação, traduzindo as requisições HTTP em chamadas de função que a aplicação Python entende. O servidor *Web* envia as informações ao servidor WSGI, que então traduz as solicitações e executa a aplicação para gerar as respostas que são então devolvidas ao servidor *Web* para que então as respostas sejam enviadas aos respectivos clientes. Esses servidores também atuam no gerenciamento garantindo que a aplicação possa atender a diversos clientes ao mesmo tempo.

O Gunicorn - Green Unicorn, é um servidor WSGI altamente eficiente e de fácil implementação, projetado para servir aplicações Python. Ele funciona como um intermediário entre o Nginx e a aplicação Python, gerenciando a execução da aplicação e distribuindo as requisições entre múltiplos processos de trabalho (workers). A capacidade desse servidor em processar diversas requisições simultaneamente permite um alto desempenho e escalabilidade, tornando-o uma escolha popular.

2.1.3.3 O Banco de Dados - MariaDB

Bancos de dados são sistemas de armazenamento de informações que permitem a inserção, atualização, exclusão e consulta de dados de maneira estruturada. Em aplicações *Web*, os bancos de dados são essenciais para gerenciar informações, como dados de usuários, transações, conteúdos e configurações. Tornando-os a espinha dorsal de muitas aplicações modernas, desde pequenas aplicações até plataformas e redes sociais complexas.

Um Sistema de Gerenciamento de Banco de Dados (SGBD) serve para organizar, armazenar e gerenciar grandes volumes de informações de maneira estruturada e eficiente. Ele facilita a inserção, atualização, exclusão e consulta de dados, garantindo a integridade e segurança das informações.

SGBDs são amplamente usados em diversas aplicações, desde *Websites* até serviços financeiros e redes sociais, proporcionando robustez, escalabilidade e suporte a múltiplas linguagens de programação, tornando-os versáteis para uma variedade de aplicativos (QUALHATO, 2023).

Um exemplo de sistema de gerenciamento de banco de dados é o MariaDB, sistema de gerenciamento utilizado no desenvolvimento do presente trabalho. É um sistema desenvolvido como *fork* do MySQL, pelo próprio criador do MySQL, após a aquisição deste pela Oracle. Mantendo a compatibilidade com MySQL, fato que simplifica a adesão por desenvolvedores familiarizados com o sistema original. O MariaDB é amplamente utilizado em aplicações *Web* devido à sua natureza *open source*, permitindo inspeção, modificação e distribuição gratuita.

Entre as vantagens do MariaDB, destaca-se a sua contínua evolução, com frequentes atualizações e sua grande comunidade de usuários e desenvolvedores que contribuem para uma vasta quantidade de recursos de suporte, como documentação e fóruns. A compatibilidade com MySQL também é uma vantagem significativa, permitindo que aplicações existentes façam a transição sem grandes dificuldades.

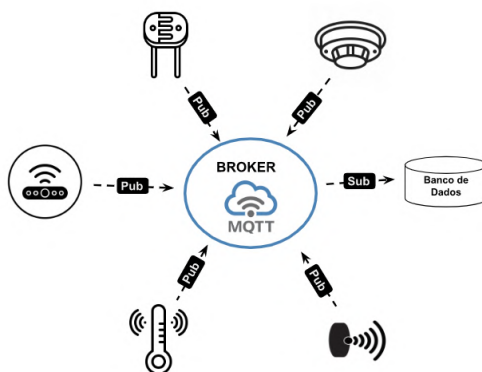
2.2 Protocolo MQTT

MQTT, sigla para *Message Queuing Telemetry Transport*, é um protocolo leve, aberto e de fácil implementação, ideal para aplicativos IoT e sistemas de comunicação *machine to machine* (M2M) (CORREA, 2016). Ele é amplamente utilizado em aplicações que exigem alta eficiência computacional devido a limitação de *hardware* e baixo uso da largura de banda da internet, operando na pilha de protocolos TCP/IP (NERI, 2019).

O servidor MQTT, conhecido como *broker*, atua como intermediário entre clientes que publicam mensagens e clientes que recebem as mensagens. De forma geral, sensores enviam dados ao *broker* publicando em tópicos específicos, os clientes inscritos nesses

tópicos são notificados assim que há novas publicações, Figura 3, recebendo as informações imediatamente do *broker* (LOCATELLI, 2016).

Figura 3 – Estrutura de comunicação do protocolo MQTT

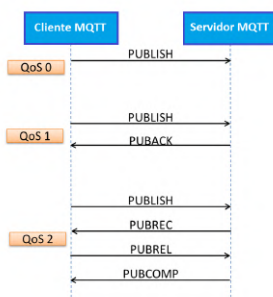


Fonte: Autor (2024)

O protocolo MQTT possui diversos pontos positivos como fácil implementação, baixo consumo de dados e de banda, comunicação bilateral, além de fornecer níveis de relevância às mensagens, garantindo eficiência na entrega. Especialmente útil na comunicação entre dispositivos de borda, destacando-se pela qualidade de serviço na entrega de dados em ambientes com recursos limitados, como é o caso de dispositivos que operam sob supervisão de pequenos sistemas embarcados (YUAN, 2017).

Um recurso diferencial do protocolo MQTT é o *Quality of Service* (QoS), que permite ao cliente escolher um nível de serviço adequado à confiabilidade da rede e às necessidades específicas do aplicativo, conforme ilustrado na figura 4. Esses níveis de serviço são classificados em três categorias distintas, oferecendo uma variedade de opções para atender às diversas exigências dos diferentes contextos de implementação.

Figura 4 – Esquema QoS



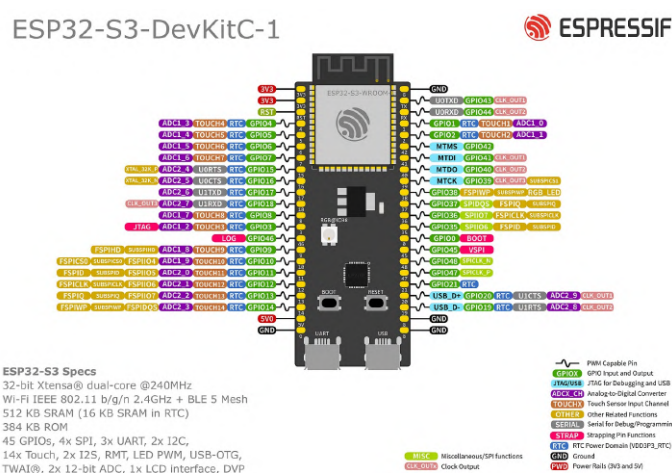
Fonte: (KODALI, 2016)

2.3 O Microcontrolador Esp 32

Os microcontroladores são dispositivos integrados que combinam elementos essenciais de um computador em um único *chip* (SYSTEMS, 2019). Eles são amplamente utilizados em aplicações contíduas desde dispositivos de consumo, como eletrodomésticos, até sistemas industriais e automóveis.

Esses dispositivos compactos incorporam uma unidade central de processamento (CPU), memória, periféricos de entrada/saída e, em alguns casos, interfaces de comunicação, como USB, UART e I2C. Sua arquitetura otimizada para baixo consumo de energia e tamanho compacto os torna ideais para sistemas embarcados e de baixo custo (ESPRESSIF, 2018). A Figura 5 apresenta as especificações dos pinos GPIO do ESP32-S3.

Figura 5 – Microcontrolador ESP32-S3



Fonte: (ESPRESSIF, 2018)

Desenvolvido pela *Espressif Systems*, o ESP32 é um microcontrolador versátil da família ESP. Conhecido por sua eficiência energética e desempenho robusto, o ESP32 é bastante utilizado em projetos de Internet das Coisas (IoT), automação residencial e dispositivos portáteis. Este microcontrolador integra um processador, conectividade Wi-Fi e *Bluetooth*, bem como diversas interfaces de comunicação, tornando-o uma escolha popular para desenvolvedores profissionais e entusiastas.

Uma das principais vantagens do ESP32 é sua conectividade sem fio integrada, que facilita a comunicação em redes locais e com a internet, sem a necessidade de componentes adicionais. Isso reduz o custo e a complexidade do desenvolvimento de projetos. O ESP32 também possui suporte nativo à recursos avançados de criptografia, como SSL/TLS, garantindo a segurança das comunicações, o que é crucial em aplicações IoT.

Os microcontroladores da família ESP, incluindo o ESP8266 e o ESP32, são amplamente aplicados em diversas áreas, como monitoramento remoto, controle de disposi-

tivos, automação industrial e sistemas embarcados. Sua versatilidade é evidenciada pela vasta comunidade de desenvolvedores que contribuem com bibliotecas e projetos de código aberto, facilitando o aprendizado e a implementação de soluções. A combinação de poder de processamento, conectividade e eficiência energética faz dos microcontroladores da família ESP uma escolha ideal para diversas aplicações.

2.3.1 Arquitetura do Microcontrolador

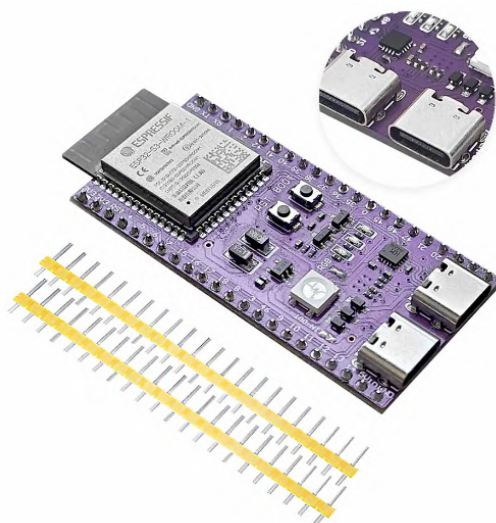
Para a construção do protótipo, a escolha do microcontrolador foi baseada em uma comparação das tecnologias realizada por (SOARES, 2023), além de uma análise em termos financeiros dos MCUs disponíveis no mercado e que se adequassem às necessidades do protótipo.

Tabela 1 – Comparativo entre microcontroladores similares.

Especificações	ESP32-S3	ESP8266	ATmega328P
Nº de Núcleos	2	1	1
Arquitetura	32 bits	32 bits	8 bits
<i>Clock</i>	240 MHz	80 MHz	16 MHz
Wi-Fi	Sim	Sim	Não
<i>Bluetooth</i>	Sim	Não	Não
Memória RAM	512 KB	160 KB	2 KB
Memória Flash	16 MB	16 MB	32 KB
E/S digitais	36	17	14
E/S analógicas	18	1	6

Fonte: (SOARES, 2023)

Figura 7 – Kit de Desenvolvimento DevKit C1 ESP-S3-WROOM-1 N16R8.



Fonte: Mercado Livre, 2024

faces periféricas incluem GPIOs programáveis, interfaces SPI, UARTs full-duplex e uma interface LCD, proporcionando uma ampla gama de possibilidades de conectividade.

Os recursos adicionais do ESP32 incluem interfaces analógicas de 12 bits, sensores de temperatura e interfaces de detecção de toque. O gerenciamento de baixo consumo é feito por uma unidade de energia com cinco modos de operação. Em termos de segurança, o ESP32 oferece boot seguro, criptografia de flash e aceleração de *hardware* criptográfico. Com essas características, o ESP32 é ideal para aplicações que necessitam de dispositivos de borda robustos e eficientes ([ESPRESSIF, 2018](#)).

2.3.2 O Ambiente de Desenvolvimento

O Arduino IDE é um ambiente de desenvolvimento integrado utilizado para facilitar o desenvolvimento de aplicações embarcadas em placas Arduino. Ele oferece uma interface simples e intuitiva, projetada especialmente para facilitar o desenvolvimento de projetos eletrônicos baseados em microcontroladores ([ARDUINO, 2019](#)).

A comunidade em torno do Arduino IDE é conhecida por seu ambiente colaborativo e acessível. Um dos principais benefícios dessa comunidade é o compartilhamento aberto de conhecimento, com isso ela contribui significativamente disponibilizando bibliotecas. Estas bibliotecas são desenvolvidas por membros da comunidade para realizar funções específicas, o que economiza tempo e esforço no desenvolvimento de novos projetos que necessitam das mesmas funcionalidades já desenvolvidas por outro usuário.

Com suporte nativo a uma linguagem de programação baseada em C++ e utilizando bibliotecas de terceiros, é possível utilizar o Arduino IDE para desenvolver e

carregar códigos diretamente nas placas de microcontroladores da família ESP, simplificando tarefas comuns e tornando-o uma escolha popular tanto para iniciantes quanto para profissionais na área de eletrônica e programação ([MCROBERTS, 2011](#)).

2.3.3 Sensores

Segundo ([JUNIOR ; FARINELLI, 2018](#)), os sensores mais empregados na automação residencial são os de luminosidade, temperatura e presença, e ainda em outras situações, menos numerosas, sensores para medição em tempo real de grandezas elétrica, sensores de gás, de umidade, vibração e até de chuva.

Neste trabalho, foram utilizados sensores de presença, de gases nocivos, umidade do ar e temperatura. Uma breve revisão sobre os sensores será apresentada, fundamentando a escolha de cada um dos componentes.

2.3.3.1 Sensores de Presença *PIR*

No ramo da automação os sensores de presença são componentes comuns e amplamente utilizados em sistemas de iluminação, alarmes, dispositivos de monitoramento e controles de acesso.

Os sensores de presença são classificados de acordo com a tecnologia empregada na detecção. Entre os modelos disponíveis no mercado, os mais utilizados são os baseados em chave magnética, fim de curso, fotoelétricos, ultrassônicos, e radiação infravermelho ou PIR, do inglês *Passive Infrared*. Um comparativo entre três sensores PIR amplamente disponíveis no mercado é apresentado na Tabela 2.

Tabela 2 – Comparativo entre sensores HC-SR501, SR602 e SR505

Característica	HC-SR501	SR602	SR505
Alimentação	5 - 20 V	3,3 - 5 V	4,5 - 20 V
Alcance de Detecção	3 - 7 m	3 - 5 m	3 - 5 m
Tipo de Detecção	Ajustável	Fixo	Fixo
Ângulo de Detecção	110º	100º	100º
Tempo de Ativação	5s a 5min	cerca de 2s	cerca de 8s
Tipo de Ativação	Ajustável	Fixo	Fixo
Modos de Operação	Com e sem repetição	N/A	N/A

Fonte: ([SOARES, 2023](#))

Os sensores PIR possuem elementos sensíveis a esse comprimento de onda, de forma que, quando um corpo que emite calor se move, ele altera o equilíbrio da radiação captada pelos elementos sensíveis, gerando um sinal elétrico que indica a presença de movimento.

Um sensor de presença de baixo custo e muito utilizado é o módulo SR602, que funciona baseado na tecnologia PIR. O módulo em questão é apresentado na Figura 8.

Figura 8 – Sensor de presença SR602.



Fonte: (SOARES, 2023)

Como esse tipo de sensor é passivo, ou seja, não emite nenhum tipo de radiação, o seu consumo de energia é baixo, o que juntamente com sua alta sensibilidade o torna o dispositivo adequado à diversas aplicações, oferecendo uma forma eficaz e robusta de detectar movimentação de pessoas em um dado espaço.

O módulo SR602 é uma opção compacta e eficiente, com boa sensibilidade a movimentos e de fácil instalação. Com um raio de detecção de até 5 metros, possui alimentação de 3,3 a 5V_{cc}, tornando-o ideal para aplicações com baixo consumo de energia. Embora possua um tempo de atraso fixo de 2 segundos, o SR602 se destaca em projetos que necessitam de um sensor PIR confiável e de pequeno porte. Na Tabela 3 são apresentadas as principais características do módulo SR602.

Tabela 3 – Especificações do sensor SR602.

Especificação	Valores
Tensão	3,3 - 5 V _{cc}
Ângulo de abertura	100º
Distância de detecção	3 - 5 m
Tempo de atraso	Cerca de 2 s
Tempo de bloqueio	Fixo
Temperatura operacional	-20 até +85°C

Fonte: (SOARES, 2023)

2.3.3.2 Sensores de gás

São sensores utilizados na detecção de gases inflamáveis, como GLP, ou gases nocivos à saúde. Como mostrado na tabela 4 existem diversos sensores semelhantes, cada um com maior sensibilidade para determinados gases.

Tabela 4 – Tipos de sensores de gás.

Sensor	Gases de alta sensibilidade
MQ-02	GLP, Metano, Propano, Butano, Hidrogênio
MQ-03	Álcool e Etanol
MQ-04	Metano, Propano e Butano
MQ-05	GLP e Gás natural
MQ-06	GLP, Isobutano e Propano
MQ-07	Monóxido de carbono
MQ-08	Hidrogênio
MQ-09	Monóxido de carbono, Metano e Propano
MQ-135	Amônia, Dióxido de carbono, Benzeno, Óxido nítrico, Fumaça e Álcool

Fonte: ([SOARES, 2023](#)).

O sensor utilizado para o protótipo foi o MQ-135 devido a sua sensibilidade a fumaça e outros gases nocivos. Na tabela 5 são apresentadas as principais características do sensor MQ-135.

Tabela 5 – Especificações do sensor MQ-135.

Especificações	Detalhes
Tensão de alimentação	3 - 5V _{cc}
Faixa de detecção	50 - 1000 ppm
CI	LM393
Temperatura operacional	-10 a +70 °C

Fonte: ([GODOI, 2018](#))

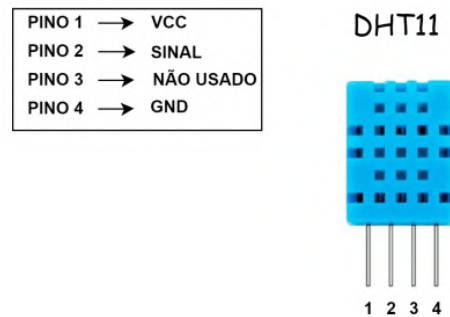
Como o uso do detector de gases no laboratório é motivado pela necessidade de detectar quaisquer registros de fumaça ou outros gases gerados por queima, solda, reações químicas, entre outros, um sensor com amplo espectro de detecção parece ser o mais lógico.

2.3.3.3 Sensor de temperatura e umidade

São utilizados para monitorar a temperatura, segundo Junior e Farinelli (2018) os três mais utilizados são o sensor semicondutor LM35, Termistores e os sensores DHT11 e DHT22. Podem ser utilizados para conforto térmico, controlando não só aparelhos de ar-condicionado, como também diversos equipamentos como chuveiro, banheira, fornos ([OLIVEIRA, 2017](#)).

Na figura 9 são apresentados o DHT11 e seus pinos de saída.

Figura 9 – Sensor DHT11



Fonte: Autor, 2024

Na tabela 6 são apresentadas as especificações desse sensor.

Tabela 6 – Especificações do sensor DHT11.

Especificações	Detalhes
Faixa de medição de umidade	20% - 90%
Faixa de medição de temperatura	0 - 50 °C
Precisão de medida de umidade	±5% UR
Precisão de medida de temperatura	±2 °C

Fonte: (JUNIOR ; FARINELLI, 2018).

2.3.3.4 LDR - *Light Dependent Resistor*

O LDR é um componente eletrônico cuja resistência varia de acordo com a intensidade da luz incidente sobre sua superfície. Em ambientes escuros, sua resistência é alta, enquanto em ambientes iluminados, a resistência diminui significativamente (THOMAZINI, 2005). Essa propriedade torna o LDR ideal para aplicações que exigem detecção de luminosidade, como sistemas de iluminação automática, medição de níveis de luz e controle de dispositivos com base nas condições de iluminação do ambiente.

2.3.3.5 Sensor de corrente elétrica

O sensor não-invasivo SCT13 é um dispositivo capaz de realizar a medição de correntes elétricas de até 100a. A sigla SCT - *Split-core Current Transformer*, refere-se a um transformador de corrente com núcleo dividido. O SCT consiste em um transformador de corrente composto por uma bobina e um núcleo de ferrite dividido (KUROISHI, 2018). O sensor em questão é apresentado na Figura 10.

Figura 10 – Sensor SCT13



Fonte: Autor, 2024

Um transformador de corrente funciona convertendo a corrente elétrica de um circuito primário, geralmente elevada, em uma corrente proporcional, mas reduzida, em um circuito secundário. Quando a corrente alternada flui pelo circuito primário, ela cria um campo magnético variável ao redor do núcleo dividido do transformador que enlaça o condutor do equipamento monitorado. Esse campo magnético induz uma corrente na bobina secundária, que é proporcional à corrente do circuito primário, mas em uma magnitude reduzida e segura para medição.

2.3.4 Atuadores

São dispositivos que possuem a capacidade de produzir uma resposta mecânica ao comando do controlador, gerando ação a partir da conversão de energia. (VIANNA, 2018).

A seguir são apresentados os atuadores utilizados no desenvolvimento deste trabalho.

2.3.4.1 Relé

O relé tem seu funcionamento baseado na passagem de corrente elétrica por uma bobina. Essa bobina cria o campo magnético responsável por acionar os contatos internos do dispositivo. Sua forma de operar é semelhante ao de uma chave liga/desliga, onde os contatos se abrem e fecha conforme a circulação de corrente na bobina de acionamento (AGUIAR, 2021).

Ainda é possível que o relé seja do tipo NA (normalmente aberto) ou NF (normalmente fechado), sendo que a diferença é estabelecida pela posição dos contatos em estado desenergizado. No relé NA, os contatos são mantidos abertos quando o relé está sem energia, permitindo a passagem de corrente apenas quando o relé foi ativado. O relé NF opera de forma análoga porém com as posições dos contatos invertidas (AGUIAR, 2021).

Facilmente encontrados no mercado, os módulos relés possuem uma ampla faixa de tensões, possuindo valores específicos para uso com microcontroladores. Um exemplo de modelo com essa característica é o JQC3F-03VDC-C, utilizado neste trabalho, suas características são apresentadas na tabela 7.

Tabela 7 – Especificações do relé JQC3F-03VDC-C.

Especificações	Valores
Tempo de acionamento dos contatos	10 ms
Temperatura de operação	-45 a 85°C
Tensão de operação	3V _{cc}
Max. tensão de comutação	250 V _{ca} / 30 V _{cc}
Max. corrente de comutação	10A

Fonte: DB Lectro Inc, 2024.

2.3.4.2 Led infravermelho

O fotodiodo, também conhecido como LED - *Light Emitting Diode*, possui uma variação projetada para emitir luz no comprimento do infravermelho. Utilizando de dispositivos semicondutores, como transistores, é possível chavear esse sinal luminoso de forma

transmitir um código em infravermelho. Esse tipo de transmissão é amplamente utilizado em dispositivos que utilizam controles remotos. Compatível com a maioria dos microcontroladores, este método é amplamente empregado na criação de projetos eletrônicos e de IoT, permitindo a comunicação sem fio em diversos tipos de aplicações ([CAVALCANTE, 2019](#))

No contexto desta aplicação, o uso do LED infravermelho se resume ao envio de comandos para o ar condicionado, de maneira a permitir a automatização dos processos de ligar e desligar, bem como disponibilizar o controle do ar condicionado de forma online para os administradores do sistema.

3 PROPOSTA DO PROJETO

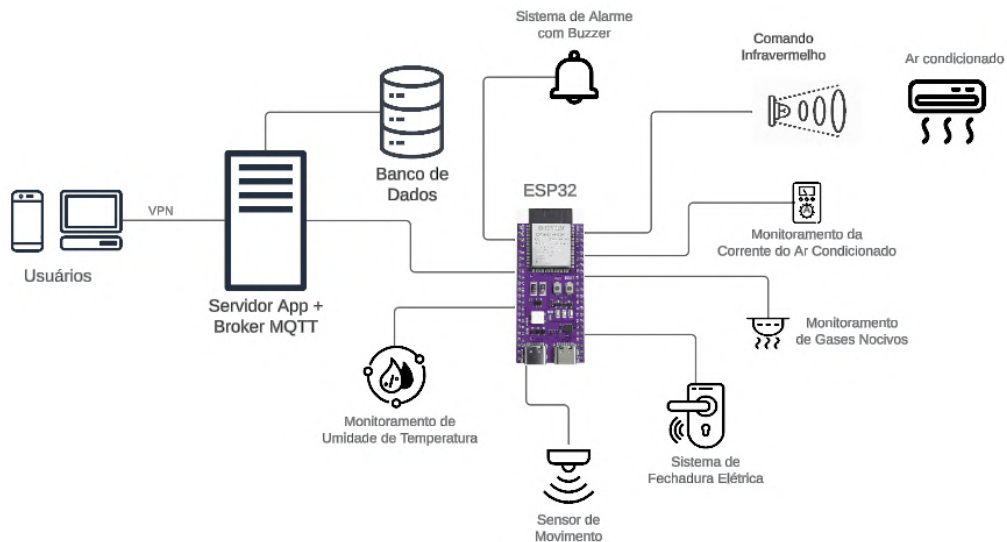
A proposta deste trabalho é o desenvolvimento de um sistema integrado composto por *hardware* e *software* que operam em conjunto para criar uma solução capaz de gerenciar o acesso e os dispositivos de forma automatizada, utilizando uma aplicação *web* que permitirá a comunicação entre os componentes físicos, interface digital e usuários.

O sistema será composto por um dispositivo de *hardware*, responsável pela coleta de dados dos sensores instalados, e por uma aplicação *web* que processará essas informações, permitindo a execução de comandos de controle sobre os atuadores conectados, além de disponibilizar as informações em tempo real para os administradores do sistema.

Ao final do desenvolvimento, o sistema proposto deverá ser capaz de realizar tarefas de monitoramento e automação em tempo real, com a capacidade de se adaptar a diferentes cenários.

A presente seção do trabalho destina-se a abordar de forma detalhada as tecnologias e circuitos eletrônicos propostos. A estrutura proposta para a central de automação é apresentada no diagrama da Figura 11.

Figura 11 – Esquema do sistema proposto.



Fonte: Autor, 2024

3.1 Projeto de *Hardware*

O projeto de *hardware* consiste em uma central de automação baseada no microcontrolador ESP32, no qual os seguintes circuitos de aquisição, comunicação e controle estão conectados:

- Monitoramento da Umidade e Temperatura;
- Monitoramento de Gás;
- Sensor de Movimento;
- Comunicação Infravermelho;
- Controle da Fechadura Eletrônica;
- Circuito de Monitoramento da Corrente Elétrica do Ar Condicionado.

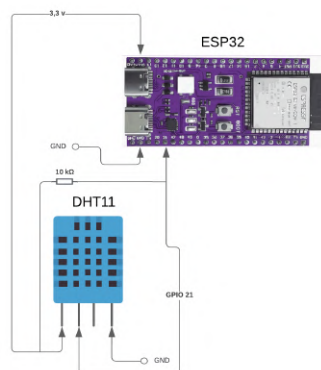
Nas próximas seções serão abordados em detalhes todos os módulos citados, bem como outras informações referentes ao protótipo.

3.1.1 Monitoramento da Umidade e Temperatura

O sensor DHT11 permite o monitoramento da temperatura e umidade de um ambiente. No protótipo, apenas a função de monitoramento importa, apesar dos dados coletados poderem ser utilizados como parâmetro para acionar um outro sistema.

Na Figura 12, é apresentado o esquema de ligação do sensor usado para o monitoramento de temperatura e umidade.

Figura 12 – Diagrama de ligação do sensor DHT11.



Fonte: Autor, 2024

3.1.2 Monitoramento de Gás

Permite o monitoramento da presença de gases nocivos e fumaça, aumentando a segurança do laboratório. O sensor utilizado foi o sensor utilizado foi o MQ135 que é sensível a diversos tipos de gases nocivos à saúde, como é descrito em seu *datasheet* (PIRES, 2018)

Ao detectar a presença de qualquer gás nocivo ou fumaça o sistema aciona imediatamente um sinal sonoro para alertar os presentes sobre o potencial perigo. O monitoramento deve ser contínuo e a resposta automatizada é um diferencial para garantir a segurança dos usuários.

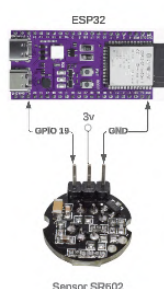
3.1.3 Sensor de Presença

Os sensores discutidos utilizam a tecnologia PIR - *Passive InfraRed* e possuem custo semelhante. No entanto, o sensor SR602 destaca-se por sua simplicidade e eficiência em aplicações que não demandam configurações avançadas. Com um alcance de detecção operando na faixa de 3 a 5 metros e um ângulo de atuação de 100°, o SR602 é uma escolha prática para projetos onde as dimensões e o baixo consumo de energia são prioritários. Diferentemente do HC-SR501, o SR602 não oferece opções de ajuste para o alcance ou tempo de reativação, mas sua confiabilidade o torna adequado para detecções rápidas e precisas em ambientes controlados.

Essas funcionalidades o tornam o sensor mais flexível e adequado para diversas aplicações, garantindo respostas precisas e adaptáveis às condições ambientais que variam ao longo do ano. Devido a essas características ele foi selecionado para ser implementado no projeto.

Na Figura 13 esta representado o esquema de ligação entre o módulo sensor SR602 e o ESP32.

Figura 13 – Diagrama de ligação do sensor SR602.

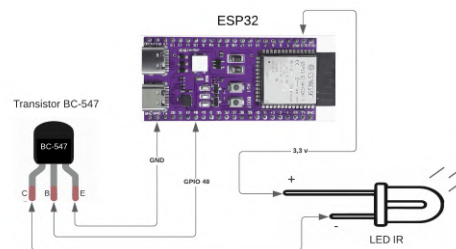


Fonte: Autor, 2024

3.1.4 Circuito de Comunicação Infravermelho

Para controlar a unidade condicionadora de ar de forma autônoma e remota é utilizado um LED IR - *InfraRed* conectado ao ESP32 e a um transistor BC-547. Esse transistor será controlado pelo ESP e usado como chave para modular os pulsos da comunicação, pois no envio de mensagens em infravermelho, os dados são codificados variando a largura da base do sinal, usualmente operando na faixa de 38 kHz. O circuito proposto é apresentado na Figura 14.

Figura 14 – Diagrama de ligação do LED infravermelho utilizando o transistor BC-547.



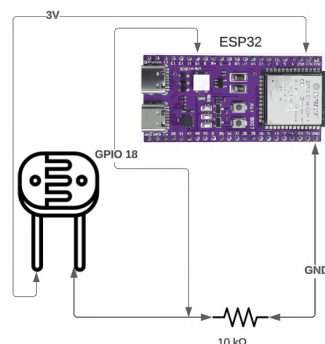
Fonte: Autor, 2024

3.1.5 LDR - *Light Dependent Resistor*

A proposta de utilizar o LDR no projeto é monitorar o estado de iluminação no ambiente, verificando se as luzes estão acesas ou apagadas. Como o LDR é sensível à variação de luminosidade, ele permite ao sistema identificar de forma simples e eficaz a presença de luz no local, sem a necessidade de leituras extremamente precisas.

O esquema de ligação da Figura 15 mostra o circuito proposto para implementação do LDR no projeto.

Figura 15 – Diagrama de ligação do LED infravermelho utilizando o transistor BC-547.



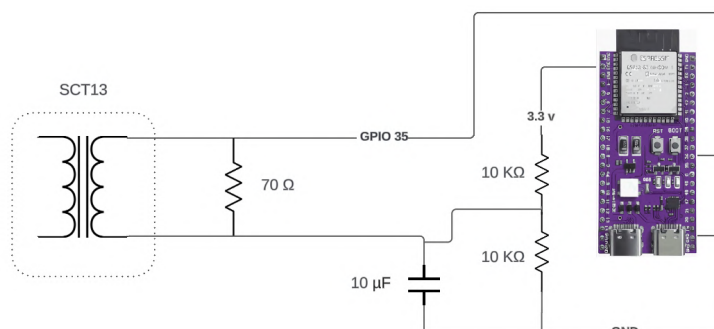
Fonte: Autor, 2024

3.1.6 Monitoramento da Corrente Elétrica do Ar Condicionado

Para realizar a leitura usando um microcontrolador, o sensor de corrente (SCT - *Split-core Current Transformer*) deve operar em série com um circuito auxiliar por onde a corrente é devidamente monitorada e medida. Para realizar essas medições, o sinal de saída do sensor deve ser condicionado à atender as especificações das portas analógicas do ESP32. Como esse microcontrolador realiza a leitura dos níveis de tensão em suas entradas analógicas, operando na faixa de $0V_{cc}$ até $3,3V_{cc}$, é necessário converter o sinal de corrente medido pelo sensor para um sinal de tensão que se enquadre na faixa de operação do ESP32.

Para isso, o circuito em questão deve operar com um resistor de carga, que irá receber a variação de tensão, de forma diretamente proporcional, a variação do sinal de corrente enviado pelo sensor. Para tal é necessário realizar o cálculo do resistor de acordo com a amplitude da corrente monitorada. A Figura 16 apresenta o circuito utilizado em conjunto com o sensor SCT para realizar a medição dos sinais de corrente (FONSECA, 2023).

Figura 16 – Circuito proposto para realizar as medições do SCT.



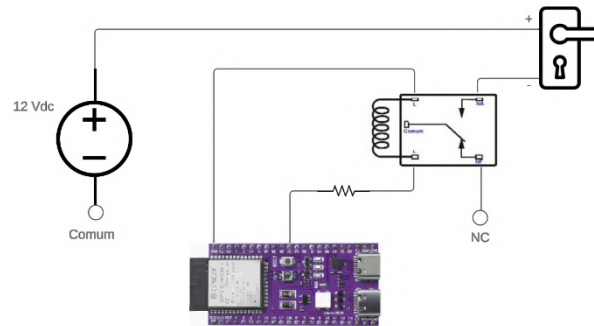
Fonte: Autor, 2024

Os cálculos são detalhados no próximo capítulo. O circuito baseia-se em um divisor de tensão formado por dois resistores de 10k ohms em série (R1 e R2). Também foi adicionado um capacitor de 10uF entre o GND e a saída Vout do divisor de tensão, criando um circuito que funciona como uma bateria de $1,66V_{cc}$. Esse ajuste permite que a forma de onda senoidal seja condicionada à oscilar entre $0V_{cc}$ e $3,3V_{cc}$.

3.1.7 Controle da Fechadura Elétrica

Possibilita ao usuário acessar o laboratório através da aplicação *web* Holhooja sem a necessidade de uma chave física. Na Figura 17 é apresentado o esquema de ligação para o controle da porta.

Figura 17 – Circuito de controle da porta.



Fonte: Autor, 2024

3.2 Projeto de *Software*

O *software* proposto é uma aplicação Flask leve e eficiente para controlar o acesso ao LPA - Laboratório de Práticas Autônomas, além de automatizar funções do laboratório, como controlar o ar condicionado e a abertura automática da porta, mediante *login* de um usuário autorizado. A aplicação é dividida em duas principais vertentes: Python e C++.

No lado Python, temos uma aplicação Flask que gerencia a interface *web*, fornecendo a interação com o usuário e permitindo o acesso às informações do laboratório em tempo real. Essa aplicação se comunica com um *broker* MQTT para enviar e receber informações do ESP32 e também realiza o gerenciamento das informações armazenadas no banco de dados.

Já a linguagem C++, usada no *firmware* desenvolvido para embarcar o ESP32, é responsável por controlar a leitura dos sensores e o acionamento dos atuadores, bem como enviar e receber informações da aplicação Python via *broker* MQTT.

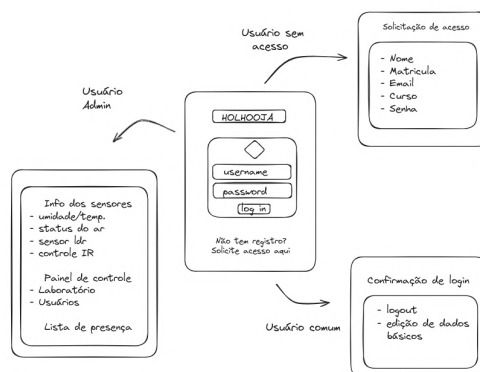
A arquitetura proposta foi elaborada para tornar a aplicação facilmente escalável e o mais versátil possível, de forma que possa ser disponibilizada via servidor físico ou máquina virtual, podendo também ser executada a partir de um contêiner, local ou disponível em nuvem.

3.2.1 Aplicação

No contexto das aplicações *web*, a linguagem Python possui bibliotecas e *frameworks* que facilitam o tratamento de requisições HTTP, integração com bancos de dados e suporte à comunicação utilizando diversos protocolos, incluindo o MQTT. O Flask, por sua vez, é um *framework* minimalista mas eficiente, focado em desenvolvimento de aplicações *web*, sua estrutura modular permite versatilidade e eficiência. Utilizando essas tecnologias é proposta a implementação de uma aplicação para controle de acesso, um

esquema da aplicação é apresentado na Figura 18.

Figura 18 – Estrutura básica da aplicação Flask proposta.

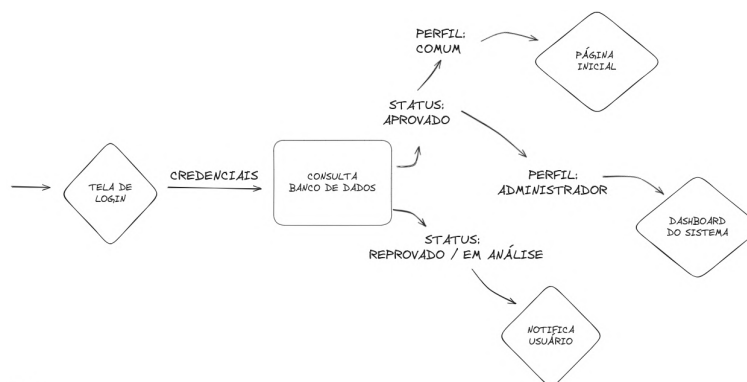


Fonte: Autor, 2024

A aplicação deve ficar disponível à qualquer aluno ou professor vinculado ao IFTO. Caso o usuário não possua registro no banco de dados da aplicação, é possível solicitar acesso ao laboratório. Feita a solicitação de acesso, ela será analisada por um administrador que ficará responsável em fazer a análise. Caso seja aprovada, o usuário poderá acessar o laboratório, independente do perfil associado pelo administrador.

Para realizar o controle de fluxo dos usuário, é necessário que a aplicação tenha acesso ao banco de dados, pois será necessário checar as credenciais de *login*, bem como realizar *logout*, registrando a hora de saída de cada usuário. Para autenticação de usuários, é proposta a estrutura esquematizada na Figura 19.

Figura 19 – Estrutura do sistema de *login*.



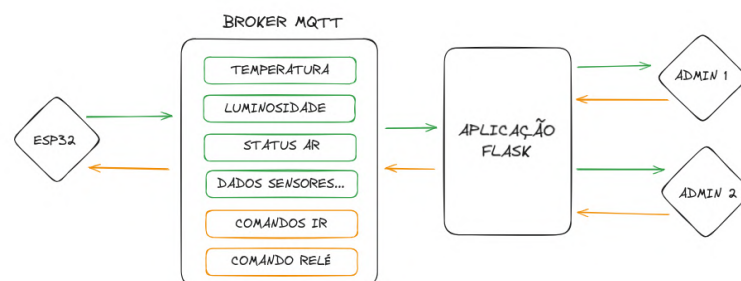
Fonte: Autor, 2024

Para desempenhar sua função, a aplicação Python também deve ser capaz de consumir, atualizar e apresentar informações do banco de dados, para permitir um acesso consistente ao banco, é proposto o uso do *MariaDB Connector/Python*, uma biblioteca

que permite a comunicação direta entre o Flask e o banco de dados MariaDB. Esse conector integra ao Python a execução de comandos SQL diretamente no código fonte, permitindo realizar operações de consulta e manipulação com o SQL e armazenar as saídas em variáveis Python. Com essa solução, a integração é feita de forma eficiente e segura, suportando diversas operações e permitindo que a aplicação Python possa interagir com o banco de dados de forma ágil e confiável.

A aplicação Python ainda deve ser capaz de receber e enviar informações ao ESP32, essa comunicação será realizada via protocolo MQTT. A estrutura apresentada na Figura 20 servirá como base para estabelecer a comunicação MQTT entre a aplicação Python e o *firmware* embarcado no ESP32.

Figura 20 – Estrutura de comunicação MQTT.



Fonte: Autor, 2024

No projeto, o acesso ao banco de dados será realizado exclusivamente pela aplicação Python, garantindo uma camada adicional entre o *firmware* do ESP32 e o banco. O *firmware*, responsável pela coleta de dados dos sensores e o controle dos atuadores, comunica-se com a aplicação Python via protocolo MQTT. A aplicação, por sua vez, fica responsável por armazenar as informações recebidas por MQTT no banco de dados e apresentá-las aos usuários administradores. Essa arquitetura evita que o ESP32 tenha acesso direto ao banco, simplificando o *firmware* e reforçando a integridade da informação.

4 METODOLOGIA

A metodologia deste trabalho foi desenvolvida com o objetivo de detalhar o processo de implementação do *hardware* e *software*, abordando os diversos testes realizados, de forma individual e conjunta.

4.1 *Hardware*

Nas próximas seções são abordadas os procedimentos utilizados na confecção e preparo dos circuitos e componentes durante o desenvolvimento do sistema proposto. Devido à presença de sensores e atuadores no sistema, a opção pela simulação computacional foi descartada, pois além de verificar o funcionamento dos circuitos propostos é necessário também validar os resultados obtidos pelos sensores, bem como testar se os atuadores operam da maneira esperada. Assim, optou-se por realizar testes diretamente no *hardware*, utilizando componentes sobressalente já adquiridos com o propósito de teste e validação.

4.1.1 Validação do sensores e atuadores

Para iniciar a simulação, foi conduzida uma etapa preliminar de validação dos circuitos propostos utilizando uma *protoboard*. Essa abordagem permitiu testar a funcionalidade dos esquemas de ligação e dos dispositivos selecionados. A utilização da *protoboard* nesta fase facilita a identificação e correção de quaisquer inconsistências, assegurando que os componentes selecionados, bem como o circuito proposto funcionem conforme esperado.

Durante os testes foi utilizada a própria conexão USB do computador como alimentação para o ESP32. Na versão final foi dada preferência ao uso de um carregador de celular simples, que fornece $5V_{cc}$ e até $2 A_{cc}$, tanto pela praticidade quanto por apresentar níveis de ruído reduzido, quando comparados com fontes de bancada comuns (CASTRO, 2023).

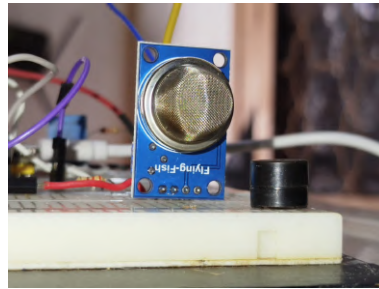
4.1.1.1 MQ135

Na Figura 21 é apresentado o sensor MQ135 durante os testes na *protoboard*.

A validação do sensor MQ135 foi realizada em um recipiente transparente e dividida em três etapas, utilizando a queima de aproximadamente 10g de incenso, papel e polipropileno. Durante cada etapa, os valores medidos pelo sensor foram exibidos no monitor serial da IDE, permitindo o acompanhamento em tempo real.

A calibração do sensor foi feita de forma simples. Ao final de cada queima, o sensor registrava valores entre 400 e 500 ppm, baseado nessas leituras, o *firmware* do ESP32 foi

Figura 21 – Sensor MQ135 montado na protoboard



Fonte: Autor, 2024

configurado para emitir sinais sonoros e visuais sempre que a média das últimas cinco leituras ultrapassasse o valor dos 450 ppm.

4.1.1.2 DHT11

Para validação do sensor DHT11 foram utilizados dois parâmetros de referência, um para a temperatura e outro para a umidade relativa do ar.

A função de termômetro do multímetro digital modelo POL-41A+ foi usado como parâmetro de validação para a leitura de temperatura do sensor, enquanto que os valores de umidade relativa do ar foram comparados com os dados publicados pelo INMET - Instituto Nacional de Meteorologia.

4.1.2 LDR - *Light Dependent Resistor*

A validação do sensor LDR foi realizada de forma qualitativa, uma vez que o objetivo principal do sensor é detectar a presença ou ausência de iluminação no ambiente. A leitura dos valores exatos não foi um critério relevante, já que o propósito do LDR no sistema é distinguir entre os estados de iluminação do ambiente.

4.1.2.1 SR602

A validação do sensor de movimento SR602 foi realizada de maneira simples, verificando a detecção de movimentos em frente ao sensor. Para confirmar seu funcionamento, foi utilizada a função *Serial.println* para apresentar no monitor serial as mudanças no estado do sensor sempre que um movimento era detectado. Esse procedimento permitiu assegurar de forma rápida e objetiva o correto funcionamento do dispositivo.

4.1.2.2 SCT 13

De forma semelhante ao LDR, a validação do sensor SCT-013, também foi baseada em uma análise qualitativa. Esse sensor de corrente, que suporta até 100A, requer a

utilização de um circuito em série para que suas leituras possam ser interpretadas pelo ESP32, o que introduz variações na resistência final, tornando difícil obter leituras precisas, especialmente em correntes baixas. Diante dessas limitações, os valores de referência observados para o sensor ficaram abaixo de 5 quando o ar-condicionado estava desligado e acima de 5 quando ligado. Com base nesses parâmetros, o *firmware* foi configurado para considerar o ar-condicionado ligado sempre que a média das últimas cinco leituras fosse superior a 5, e desligado quando essa média fosse inferior.

4.1.2.3 Atuadores: Relé e Led IR

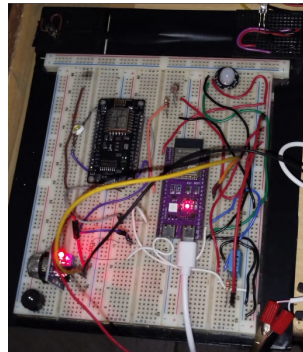
O módulo relé, responsável pela abertura da porta, e o LED emissor infravermelho, utilizado para o envio de comandos ao ar-condicionado, foram validados em uma protoboard. A validação do circuito emissor IR foi realizada utilizando funções da biblioteca *IRremoteESP8266*, que estabelecem a comunicação via sinais infravermelhos. Para isso, foi necessário realizar a captura e gravação dos códigos emitidos pelo controle remoto original do ar-condicionado.

O código utilizado para a obtenção dos sinais foi o exemplo padrão *IRrecvDumpV2* da mesma biblioteca. Esse código ativa o receptor infravermelho, permitindo que ele entre em modo de espera para captar sinais IR e os exiba no monitor serial da IDE em diferentes formatos, facilitando a análise e verificação das informações contidas no código recebido. O formato escolhido para a emissão dos comandos foi o *RAW*, devido à sua capacidade de reproduzir com precisão os sinais capturados do controle remoto original.

O LED emissor infravermelho TIL 78, controlado por um transistor BC547, foi configurado para transmitir os sinais obtidos do controle do ar-condicionado. As funções da biblioteca *IRremoteESP8266* permitem o envio de comandos em diversos formatos, enquanto o transistor amplifica e modula o sinal para garantir que o LED transmita o código IR corretamente. Ao final da validação, constatou-se que o circuito estava emitindo os sinais IR de maneira adequada, controlando o ar-condicionado sem a necessidade do controle remoto original.

4.1.3 Modelo de baixa fidelidade

Após validar os circuitos propostos e os componentes utilizados, foi realizada a montagem de todo o sistema na *protoboard*, com objetivo de verificar o funcionamento dos subsistemas de forma integrada, como mostrado na Figura 22.

Figura 22 – Dispositivos montados em *protoboard* para a validação

Fonte: Autor, 2024

Na Figura 23 são observados os dados da leitura dos sensores transmitidos para o computador via comunicação serial e apresentados diretamente na saída serial da IDE do Arduino.

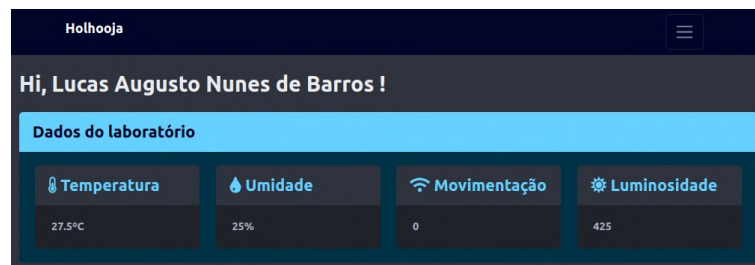
Figura 23 – Dados dos sensores apresentados no monitor serial da IDE.

```
Sem gases nocivos detectados.  
Quantidade de gás detectada: 69  
Umidade: 25.0%, Temperatura: 27.5°C  
Luminosidade: 431
```

Fonte: Autor, 2024

Já na Figura 24 os mesmos valores lidos pelos sensores são apresentados diretamente na interface gráfica da aplicação, atestando o funcionamento perfeito da infraestrutura de comunicação MQTT.

Figura 24 – Dados dos sensores apresentados na interface da aplicação.



Fonte: Autor, 2024

4.2 Software

Para uma maior compreensão sobre os *softwares* utilizados no projeto, o tópico está dividido em três seções. Primeiramente, será explorada a aplicação Flask, incluindo sua estrutura e funcionalidades, bem como sua interação com o banco de dados. Em seguida, o funcionamento do Eclipse Mosquitto MQTT e sua função no desenvolvimento do projeto. Por fim, o *firmware* embarcado no ESP32, o código será parcialmente exposto, cobrindo as partes mais relevantes.

4.2.1 Aplicação Flask: Estrutura, funcionalidades e configurações

Nesta seção é apresentado o processo de desenvolvimento da aplicação Python. Incluindo a elaboração do código, sua estrutura de projeto, o mapeamento do banco de dados, telas do sistema e demais funcionalidades presentes na aplicação.

O desenvolvimento da aplicação foi conduzido em um ambiente virtual Python, executado no sistema operacional Ubuntu 22.04 LTS e utilizando a IDE Visual Studio para implementação da aplicação Flask. O ambiente virtual em questão foi configurado de acordo com o arquivo *requirements.txt*, presente no apêndice C.

Com o objetivo de facilitar a integração com outros sistemas no futuro, bem como alcançar um bom nível de manutenibilidade, foi adotada uma arquitetura com clara distinção entre a interface de usuário (*front-end*) e processamento de dados (*back-end*), bem como as tecnologias foram selecionadas de forma a fomentar integrações com sistemas externos.

A infraestrutura da aplicação foi desenvolvida em Python, utilizando o microframework Flask, enquanto o front-end utiliza tecnologias *web* padrão, como HTML, CSS e JavaScript. Para a comunicação com o servidor foram usadas requisições HTTP, permitindo dessa forma a comunicação entre plataformas desenvolvidas sobre tecnologias distintas.

4.2.1.1 Estrutura da Aplicação

A aplicação foi implementada de acordo com a estrutura apresentada a seguir:

```
Holhooja/  
├── app/  
│   ├── templates/  
│   ├── __init__.py  
│   ├── routes.py  
│   ├── sockets.py  
│   └── forms.py
```

```
├── models.py
├── venv/
├── config.py
├── holhooja.py
├── mqtt.py
└── requirements.txt
```

O arquivos principais encontram-se dentro do diretório *app/*, enquanto os arquivos *config.py*, *holhooja.py* e *mqtt.py* possuem apenas códigos complementares. A função básica de cada um desses arquivos será desenvolvida adiante.

4.2.1.2 O Núcleo da Aplicação

O arquivo `__init__.py` é o cabeçalho do núcleo da aplicação Flask, pois ele contém a importação das principais bibliotecas e declaração dos principais objetos, como por exemplo o objeto *app* do pacote Flask que é o principal em termos de manipulação. Parte do código em questão pode ser visto abaixo.

```
1 from flask import Flask
2 from config import Config
3 from threading import Lock
4 from flask_login import LoginManager
5 from flask_sqlalchemy import SQLAlchemy
6 from flask_socketio import SocketIO
7 import paho.mqtt.client as mqtt
8 from mqtt import on_connect, on_message, on_publish
9
10 async_mode = None # Modo de sincronia do socket
11 app = Flask(__name__) # Objeto principal
12 app.config.from_object(Config) # Configura o objeto app segundo o arquivo
    Config
13 db = SQLAlchemy(app) # Objeto db para manipular o banco de dados
```

Lista de Ilustrações 4.1 – Trecho do código `__init__.py`

Juntamente com o cabeçalho de declarações `__init__.py`, o arquivo *routes.py* é responsável por garantir a interação do usuário com a aplicação. Nele estão contidas as rotas da aplicação, que associam URLs a funções de retorno, que enviam respostas para quem as solicita. Nesse arquivo é definido como a aplicação deve responder diferentes requisições desde a renderização de páginas *web* até o processamento de formulários e a interação com o banco de dados.

O responsável pela declaração e gerenciamento dos formulários da aplicação é o arquivo *forms.py*. Nele são definidos os campos de cada formulário, suas validações e a

lógica de manipulação dos dados fornecidos pelos usuários. Dessa forma é simples criar interfaces de entrada de dados de forma estruturada e segura.

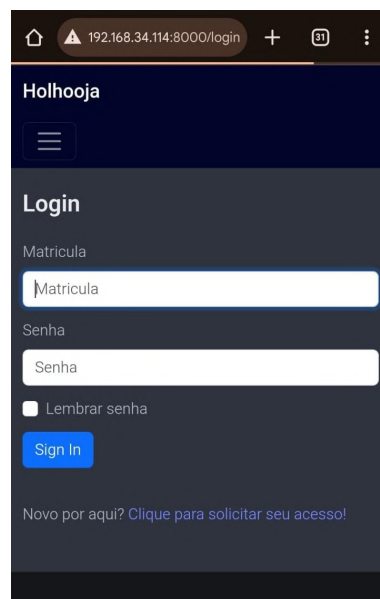
Abaixo é apresentado um trecho de código onde é declarado o formulário de *login* da aplicação.

```
1 class LoginForm(FlaskForm):
2     matricula = StringField(validators=[DataRequired(),
3                                     InputRequired(),
4                                     Length(min=14, max=14)],
5                             render_kw={"placeholder": "
6                                     Matricula"})
7
8     senha = PasswordField(validators=[DataRequired(),
9                                     InputRequired(),
10                                    Length(min=8, max=20)],
11                            render_kw={"placeholder": "Senha"
12                                    })
13
14     remember_me = BooleanField('Lembrar senha')
15
16     submit = SubmitField('Sign In')
```

Lista de Ilustrações 4.2 – Trecho do código *forms.py*

O formulário obtido como resultado é apresentado na Figura 25.

Figura 25 – Tela de *login* da aplicação.



Holhooja

Login

Matricula

Senha

☐ Lembrar senha

Sign In

Novo por aqui? [Clique para solicitar seu acesso!](#)

Fonte: Autor, 2024

De forma similar, o arquivo *models.py* é utilizado para definir as estruturas do banco de dados, as classes que representam as tabelas e suas relações. Para facilitar a interação com o banco de dados é utilizada a biblioteca SQLAlchemy que mapeia as classes Python e as rotula para suas respectivas tabelas do banco de dados, além de possuir uma coleção de métodos próprios para interação com bases de dados, o que simplifica o processo de implementação.

Essa abordagem permite que a interação seja orientada a objetos, o que facilita a manipulação segura de maiores quantidades de dados. A seguir segue um trecho de código contendo a declaração de uma classe Python referente a uma tabela no banco de dados.

```
1 class Acesso(db.Model):
2     __tablename__ = 'acesso'
3     id = Column(Integer, primary_key=True)
4     user_id = Column(Integer, ForeignKey('user.id'))
5     data_criacao = Column(DATETIME, nullable=False, default=datetime.now)
6     register_type = Column(String(5)) # INPUT / OUTPUT
7
8     def __repr__(self):
9         return '<Acesso ID: {}>'.format(self.id)
```

Lista de Ilustrações 4.3 – Trecho do código *models.py*

Neste trecho é possível ver a declaração de uma classe chamada *Acesso* que se relaciona com uma tabela no banco de dados chamada *acesso*, como aparece na linha 2. Fazendo a chamada da classe base *db.Model* na declaração é possível fazer com que a nova classe *Acesso* herde métodos e funcionalidades da sua classe base, incluindo a capacidade de representar uma tabela do banco de dados.

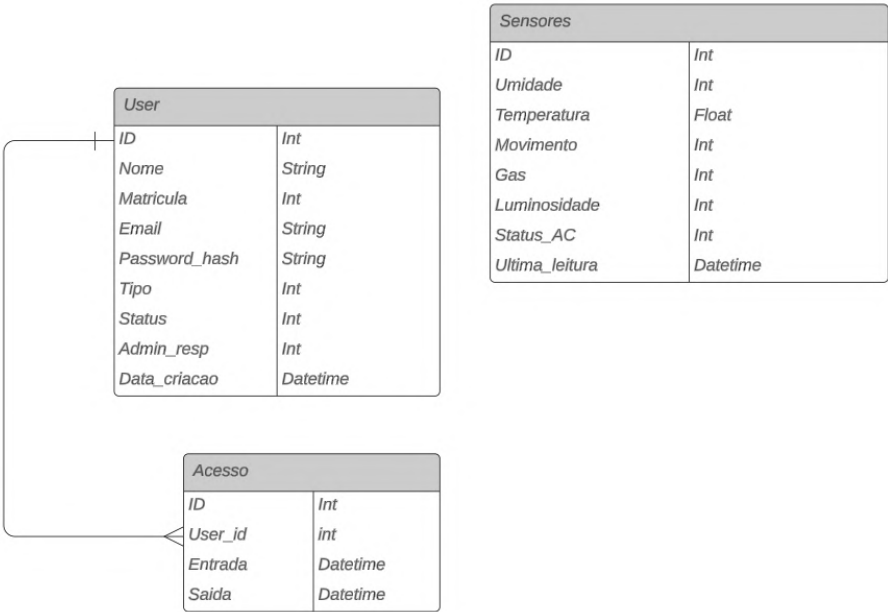
O arquivo *sockets.py* lida com a comunicação em tempo real na aplicação, utilizando a biblioteca *Flask-SocketIO*. Ele gerencia eventos de *WebSocket*, permitindo a troca de mensagens entre o servidor e os clientes conectados em tempo real, o que é particularmente útil para funções de notificações e atualizações dinâmicas dos dados laboratoriais

4.2.1.3 O Banco de Dados

O banco de dados utilizado para armazenar informações referentes ao sistema é formado por três tabelas, sendo elas *User*, *Acesso* e *Sensores*.

O diagrama de entidade de relacionamento do banco utilizado pela aplicação pode ser visto a seguir na Figura 26.

Figura 26 – Diagrama de Entidade de Relacionamento.



Fonte: Autor, 2024

A tabela *User* armazena todas as informações essenciais sobre os usuários que utilizam o sistema, exercendo papel na gestão e autenticação dos mesmos. Esta tabela inclui detalhes como nome, matricula, email, tipo e status, permitindo o controle sobre as permissões de cada usuário dentro do sistema.

Na Tabela 8 é apresentada a estrutura dos dados atribuídos a cada usuário com o nome dos campos e uma breve descrição.

Campo	Descrição
Nome	Nome completo do usuário
Matrícula	Número de matrícula do usuário
Password_hash	Senha utilizada para acessar o sistema
E-mail	E-mail para contato
Curso	Curso ao qual o usuário está vinculado
Tipo	Comum ou Administrador
Status	Aguardando, ativo ou inativo
Data_criação	Dia que a solicitação de acesso foi feita

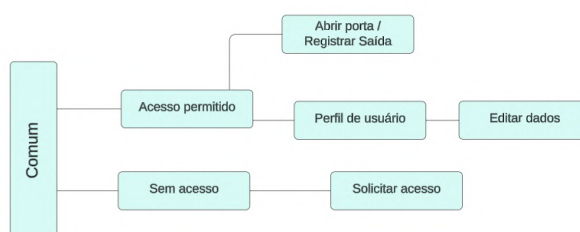
Tabela 8 – Campos da Tabela User

A tabela *User* fará o registro do usuário que solicitar acesso através da aplicação *web*. Os campos *Nome*, *Matricula* e *Curso* são para identificação do usuário, o email é solicitado com o intuito de estabelecer comunicação e os campos *Tipo* e *Status* são referentes às permissões de acesso que cada usuário deve ter.

O campo *Tipo* classifica o usuário em dois perfis: Comum ou Administrador, como será exposto mais adiante. Já o campo *Status* reflete o andamento da solicitação de acesso ao laboratório, com três possíveis valores. O status *Aguardando* indica que a solicitação ainda não foi analisada por um administrador, sendo assim ainda não é permitido o acesso ao laboratório. Quando a solicitação é aprovada, o status muda para *Ativo*, concedendo acesso ao usuário solicitante. O status *Inativo* é atribuído quando a solicitação é reprovada pelo administrador ou quando o usuário deixa de ter vínculo com a instituição.

As estruturas e diferenças de cada tipo de usuário são elucidadas a seguir, nas Figuras 27 e 28.

Figura 27 – Estrutura do Usuário Comum.

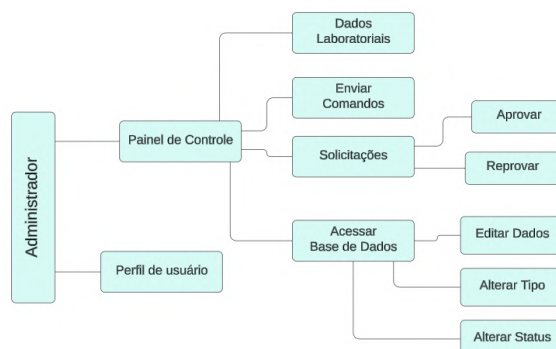


Fonte: Autor, 2024

O usuário do tipo Comum é o padrão e tem permissão apenas de fazer *login* para acessar o laboratório e *logout* para registrar sua saída, dessa forma contabilizando o seu tempo de permanência.

Para ascender ao nível Administrador é necessário ser promovido por alguém que já possua permissões de administrador. Esse tipo de usuário possui permissão de editar dados amplamente, além de poder alterar o *Tipo* e *Status* dos usuários.

Figura 28 – Estrutura do Usuário Administrador.



Fonte: Autor, 2024

Prosseguindo com a análise das demais tabelas que integram a base de dados. A seguir, na tabela 9, é apresentada a tabela *Acesso*, que registra as entradas e saídas do laboratório.

Campo	Descrição
User_id	Número de identificação do usuário
Entrada	Dia e hora da entrada
Saída	Dia e hora da saída

Tabela 9 – Campos da Tabela Acesso

A tabela *Acesso* desempenhará a função do antigo caderno de registros, controlando as entradas e saídas de cada usuário. Dessa forma duas funções são automatizadas: primeiro, o controle de acesso deixa de ser manuscrito para ser realizado via *login* em um aplicativo *web* e segundo, a geração do certificado de horas complementares deixa de ser feita via contagem manual das horas de cada usuário no caderno de registros, para ser auxiliada pela aplicação. Será disponibilizado o total de horas que cada usuário permaneceu no laboratório através do *Painel de Controle*, aba *Usuário*.

Abaixo é retratada a tabela de sensores, onde os dados enviados pelo ESP32 são armazenados.

Campo	Descrição
Umidade	Leitura de umidade do DHT11
Temperatura	Leitura de temperatura do DHT11
Luminosidade	Nível de iluminação medido pelo LDR
Movimento	Leitura do HC-SR501
Gas	Saída do MQ135
Status_AC	Status do ar condicionado, on/off
Ultima_leitura	Dia e hora da última atualização

Tabela 10 – Campos da Tabela Sensores

Os valores obtidos pelos sensores são recebidos pelo ESP32 e enviados ao *broker* em intervalos regulares, que por sua vez, ao receber essas informações, as armazena no banco de dados, permitindo que sejam consumidas pela aplicação e disponibilizadas aos administradores através do painel de controle

4.2.1.4 WebSocket

Devido a necessidade de receber informações em tempo real do servidor e atualizar o *front-end* de forma automática, foi utilizada a biblioteca *Flask-SocketIO* que permite interação da aplicação Python, no *back-end*, com o JavaScript executado no navegador do dispositivo, no *front-end*

A *Flask-SocketIO* adiciona suporte para comunicação bidirecional em tempo real entre clientes e servidores utilizando *WebSockets*. Essa biblioteca permite a atualização instantânea de informação, operando como um recarregamento constante da página.

O trecho de código apresentado a seguir realiza a consulta ao banco de dados periodicamente e envia dos dados consultados para atualizar os valores de cada sensor no *dashboard* do usuário administrador.

```
1 def background_thread():
2     """Example of how to send server generated events to clients."""
3     count = 0
4     while True:
5
6         with app.app_context():
7             dados_sensores = db.first_or_404(sa.select(Sensores).where(
8                 Sensores.id == 1))
9             socketio.sleep(3)
10            count += 1
11            socketio.emit('my_response', {
12                'umidade': dados_sensores.umidade,
13                'temperatura': dados_sensores.
14                    temperatura,
15                'luminosidade': dados_sensores.
16                    luminosidade,
17                'movimento': dados_sensores.
18                    movimento
19            })
```

Lista de Ilustrações 4.4 – Trecho do código *sockets.py*

No *front-end*, onde é executado o JavaScript da aplicação, existe uma função implementada de maneira a consumir as informações da aplicação Flask via *WebSocket*. Essa função é apresentada a seguir.

```
1 socket.on('my_response', function(msg, cb) {
2
3     $('#temperatura').text($('#<div/>').text( msg.temperatura +
4         ' C ').html());
5     $('#umidade').text($('#<div/>').text( msg.umidade + '%').
6         html());
7     $('#movimento').text($('#<div/>').text( msg.movimento).html
8         ());
9     $('#luminosidade').text($('#<div/>').text( msg.luminosidade)
10         .html());
11 })
```

```
8      // Informa o status do ar condicionado baseado no valor do
      sensor
9      const status = msg.movimento === 1 ? 'Ligado' : 'Desligado'
      ;
10     $('#status_ar').text(status);
11
12     if (cb)
13         cb();
14
15     });
```

Lista de Ilustrações 4.5 – Trecho do código JavaScript implementado em *base.html*

4.2.1.5 Arquivos Complementares

Os arquivos complementares são aqueles localizados fora do diretório *app/*, sendo eles *config.py*, *holhooja.py*, *mqtt.py*, *requirements.txt*

O arquivo *config.py* define a classe *Config* que contém duas configurações importantes para a aplicação. Essa classe define o valor padrão de algumas variáveis do ambiente para o caso delas não estarem devidamente configuradas. Isso promove a robustez da aplicação, permitindo que as configurações sejam facilmente alteradas sem modificar o código principal, além de ser uma boa prática para a segurança e manutenção do código.

```
1 import os
2 from datetime import timedelta
3
4 class Config:
5     SECRET_KEY = os.environ.get('SECRET_KEY') or 'uma-chave-muito-secreta'
6
7     PERMANENT_SESSION_LIFETIME = timedelta(hours=4)
8
9     SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or "mariadb+
    mariadbconnector://admin:admin@127.0.0.1:3306/lpa_db"
```

Lista de Ilustrações 4.6 – Código *config.py*

A configuração *SECRET_KEY* é muito importante dentro de diversas aplicações *web*, pois é utilizada na assinatura de cookies, tokens de sessão e em outras formas de garantir a segurança dos dados. Caso a variável de ambiente não esteja devidamente definida, será usado o valor padrão *'uma-chave-muito-secreta'*.

A variável de ambiente *PERMANENT_SESSION_LIFETIME* é usada no Flask para definir o tempo limite de uma sessão. Ela controla o tempo de uma sessão do usuário antes que ele seja desconectado automaticamente por inatividade. O valor é definido

especificando a duração da sessão. Após o tempo configurado, a sessão expira e o usuário precisa realizar o *login* novamente.

A variável `SQLALCHEMY_DATABASE_URI` define o URI do banco de dados para o SQLAlchemy. Ela tenta obter o valor da variável de ambiente `DATABASE_URL` e não estando definida, é usado um conector padrão para uma base de dados MariaDB localizada no *localhost*, com o usuário *augusto*, senha *1234*, e banco de dados *sql_db*. Isso permite flexibilidade para configurar e até mesmo alterar o banco de dados sem a necessidade de modificar o código principal da aplicação.

O próximo arquivo é o *holhooja.py*, se trata de um simples arquivo de configuração para o shell interativo do Flask, não sendo necessário para o funcionamento da aplicação final, mas muito útil durante o desenvolvimento e depuração.

```
1 import sqlalchemy as sa
2 import sqlalchemy.orm as so
3 from app import app, db
4 from app.models import User, Acesso
5
6 @app.shell_context_processor
7 def make_shell_context():
8     return {'sa': sa, 'so': so, 'db': db, 'User': User, 'Acesso': Acesso}
```

Lista de Ilustrações 4.7 – Código *holhooja.py*

A função *make_shell_context* precedida pelo decorador *@app.shell_context_processor*, retorna um dicionário com referências aos módulos 'sqlalchemy' ('sa'), 'sqlalchemy.orm' ('so'), à instância do banco de dados 'db', e aos modelos 'User' e 'Acesso'. Isso configura o contexto do shell interativo do Flask para que esses objetos possam ser acessados diretamente via shell (com o comando *flask shell*), simplificando o processo de importação e tornando esse recurso mais eficiente.

Por último, o arquivo *requirements.txt* possui o nome dos pacotes e suas respectivas versões utilizadas no desenvolvimento desta aplicação, esse arquivo também não é necessário para o funcionamento final da aplicação mas é muito importante para garantir a consistência e a reprodutibilidade do ambiente de desenvolvimento. Esse arquivo além de listar todas as dependências necessárias, também permite realizar a instalação automática com um único comando (*pip install -r requirements.txt*). Isso facilita a colaboração e a integração contínua no desenvolvimento do projeto, além de ajudar na resolução de problemas de compatibilidade e na replicação do ambiente de produção.

4.2.2 Comunicação via MQTT

No projeto, o ESP32 desempenha o papel de coletar dados dos sensores para monitorar e controlar o ambiente, fornecendo essas informações em tempo real aos administradores do sistema. Além disso, ele recebe comandos da aplicação, permitindo o controle remoto dos atuadores. Para garantir que todas as interações ocorram em tempo real, a comunicação entre o ESP32 e a aplicação Python é realizada via protocolo MQTT, com o Eclipse Mosquitto atuando como intermediário.

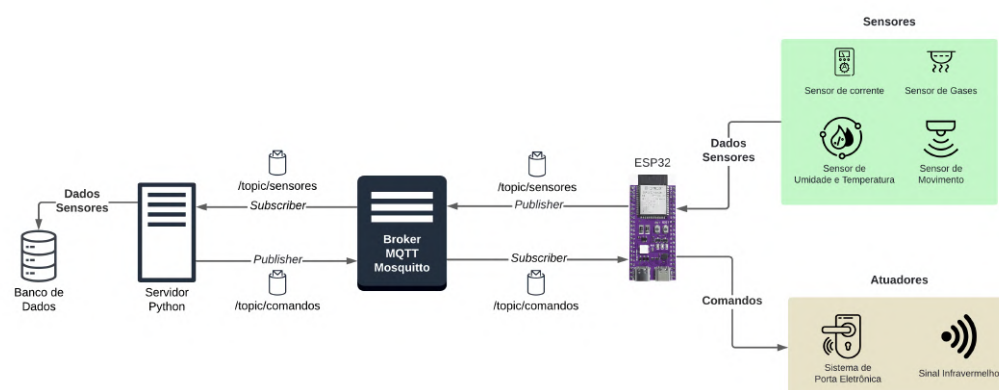
O arquivo *mqtt.py* possui as configurações de conexão com o Eclipse Mosquitto e as funções necessárias para armazenar as informações recebidas no banco de dados. O código completo pode ser visto no apêndice C.

4.2.2.1 Tópicos da Aplicação

Nessa seção são abordadas as configurações dos tópicos para comunicação MQTT, as funções de *callback*, que garantem respostas do sistema, bem como o funcionamento sistemático da comunicação MQTT entre aplicação e ESP32.

Os sensores do laboratório são lidos pelo ESP32, que publica seus dados através do *broker*. A aplicação Flask, por sua vez, consome essas informações e as armazena no banco de dados. Podendo ainda enviar comandos, publicando-os via *broker* enquanto o ESP32 recebe esses comandos e aciona os respectivos atuadores, como é apresentado no diagrama da Figura 29.

Figura 29 – Estrutura de Comunicação MQTT.



Fonte: Autor, 2024

A seguir é apresentado um trecho do *firmware* desenvolvido em C++, onde os tópicos da aplicação são definidos.

```
1 // Topicos MQTT
2 const char* topicEspAr = "/comandos/esp32/AC"; // subscribe
3 const char* topicEspPorta = "/comandos/esp32/porta"; // subscribe
4 const char* topicSensorDHT = "/sensores/DHT"; // publish
5 const char* topicSensorLDR = "/sensores/LDR"; // publish
6 const char* topicSensorSR602 = "/sensores/SR602"; // publish
7 const char* topicSensorSCT = "/sensores/SCT13"; // publish
8 const char* topicSensorMQ135 = "/sensores/MQ135"; // publish
```

Lista de Ilustrações 4.8 – Tópicos MQTT Usados na Aplicação

Vale destacar que, no desenvolvimento deste projeto, o servidor NginX, o Eclipse Mosquitto e o MariaDB foram hospedados na mesma máquina física, localizada no laboratório 907 do IFTO, o mesmo Laboratório de Práticas Autônomas onde esta instalado o ESP32 e o sistema de automação.

No entanto, essa configuração é opcional, pois a arquitetura do sistema foi projetada para permitir flexibilidade na distribuição dos componentes. A aplicação pode ser hospedada em diferentes ambientes, seja uma infraestrutura local ou em nuvem, utilizando máquinas virtuais ou servidores físicos, mantendo a comunicação e o funcionamento corretos da aplicação por meio de redes locais ou da internet, dependendo apenas da devida configuração das rotas de conexão entre as partes da aplicação.

4.2.3 *Firmware* do ESP32.

Para o desenvolvimento do *firmware* foi utilizada a Arduino IDE 2.3.2. Essa nova versão da IDE conta com um sistema de gerenciamento de bibliotecas mais prático que sua versão anterior, permitindo a instalação de bibliotecas externas de forma simplificada. As bibliotecas utilizadas no desenvolvimento do *firmware* são apresentadas abaixo.

- DHTStable;
- EmonLib;
- IRremoteESP8266;
- PubSubClient.

O código desenvolvido para embarcar o ESP32 foi dividido em dois arquivos: o código principal e a biblioteca de funções, para dessa forma modularizar o *firmware* e facilitar a manutenção do sistema. Essa separação organiza o código, tornando-o mais legível e fácil de depurar. Além de permitir atualizações da biblioteca sem a necessidade de alterar o código principal, promovendo uma estrutura mais robusta e eficiente. Ambos

arquivos foram desenvolvidos em linguagem de programação C++ utilizando o Arduino IDE 2.0.

No código principal encontram-se as configurações das portas GPIO, da comunicação Wi-Fi e MQTT, bem como as declarações e chamadas de funções implementadas na biblioteca, que por sua vez, contém as rotinas utilizadas pelo código principal. Ambos os códigos encontram-se no apêndice A.

5 RESULTADOS E DISCUSSÕES

Para obter resultados quanto ao funcionamento do protótipo, foram realizadas testes para cada um dos circuitos do esquema geral apresentado na Figura 3.1. Validados os circuitos, foram iniciados os testes nos sistemas, de forma individual, tanto a aplicação Python como o *firmware* do ESP32. Somente após a validação de ambos os sistemas operando de forma individual é que foram iniciados os testes conjuntos.

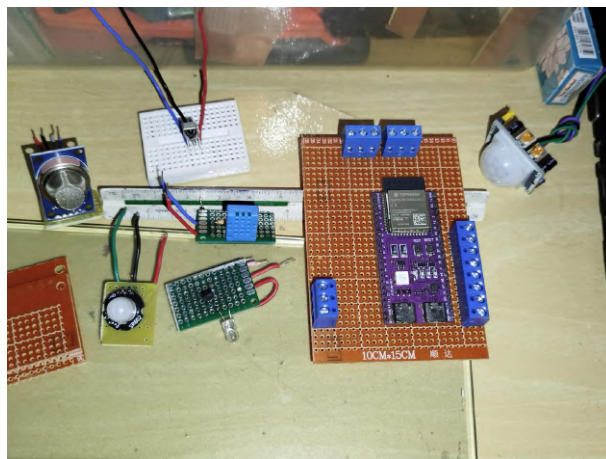
5.1 Modelo final do protótipo

Uma vez testados e validados os circuitos e seus respectivos dispositivos, foi feita a soldagem dos componentes em placas de fenolite ilhadas e uma nova validação foi feita para assegurar que a confecção das placas não comprometeu o funcionamento do circuito.

Para o modelo final, os componentes do circuito foram soldados em placas de fenolite ilhadas, garantindo uma montagem mais robusta. O ESP32, juntamente com os bornes de contato, foi soldado diretamente na placa principal, assegurando uma conexão duradoura. Paralelamente, os dispositivos periféricos foram montados em placas menores, projetadas para facilitar a instalação em posições estratégicas do ambiente, otimizando a coleta de dados e o desempenho do sistema.

Na Figura 30 são apresentados os dispositivos periféricos, já soldados em suas placas.

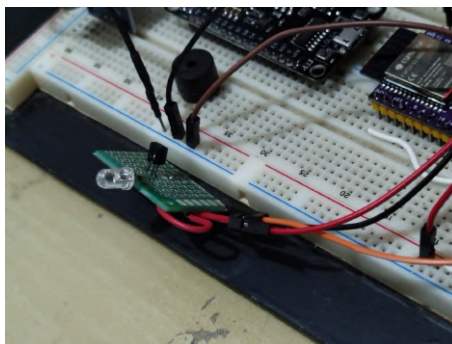
Figura 30 – Dispositivos soldados



Fonte: Autor, 2024

A seguir, na Figura 31 o circuito emissor IR, utilizado para o envio de comandos ao ar-condicionado, soldado em uma placa de fenolite.

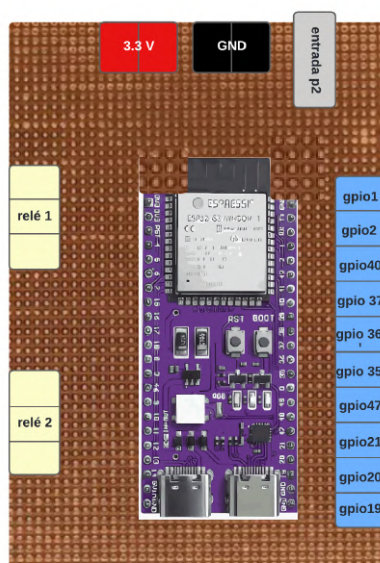
Figura 31 – Circuito emissor IR durante teste após confecção da placa



Fonte: Autor, 2024

Abaixo é possível verificar, na Figura 32, o modelo final do projeto de confecção da placa de circuito.

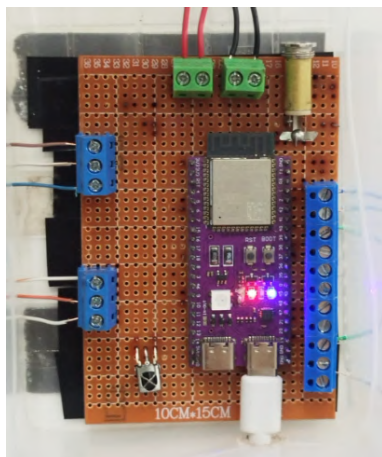
Figura 32 – Modelo proposto para confecção do protótipo



Fonte: Autor, 2024

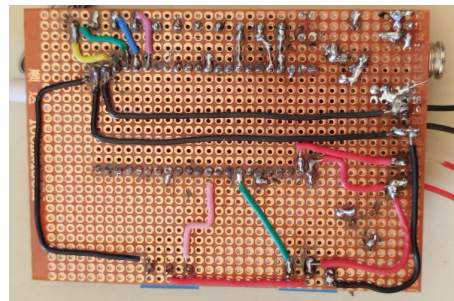
O resultado final da confecção é apresentado na Figura 33 e na Figura 34, onde a placa é exibida em sua vista superior e inferior, respectivamente.

Figura 33 – Vista superior da placa principal



Fonte: Autor, 2024

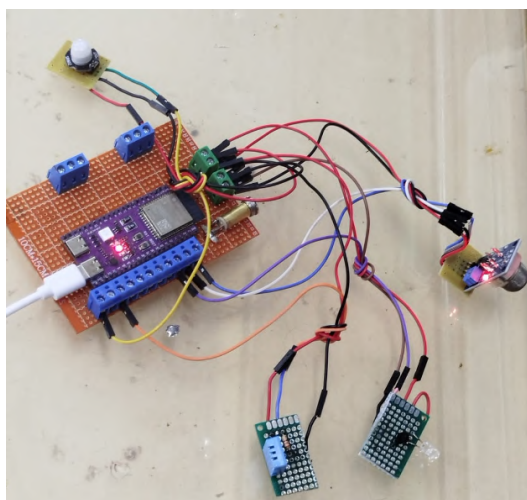
Figura 34 – Vista inferior da placa principal



Fonte: Autor, 2024

Após confeccionada, a placa principal também precisou passar por validação para garantir funcionamento íntegro do dispositivo e conexões. A placa principal conectada com os dispositivos periféricos é apresentada na Figura 35.

Figura 35 – Placa final com sensores conectados para validação final dos circuitos.



Fonte: Autor, 2024

5.2 Instalação do sistema no Laboratório de Práticas Autônomas

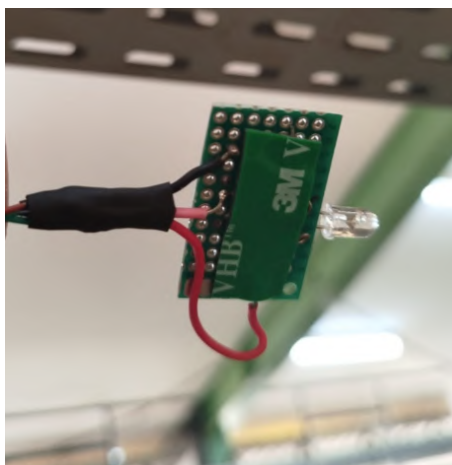
Após confeccionadas e validadas, as placas com os circuitos foram levadas aos LPA para a instalação.

Devido a problemas com o acesso a BIOS das máquinas locais presentes no laboratório, o servidor da primeira versão do sistema foi configurado em um notebook Dell,

com 8GB de memória RAM, 200GB de armazenamento em HD, processador Intel i3 e sistema Linux Ubuntu 22.04 LTS.

Uma vez configurados os serviços no lado servidor, foi preciso atualizar os dados de conexão do *firmware* embarcado no ESP32. Prosseguindo, foi feita a instalação das placas com os circuitos periféricos em locais estratégicos do laboratório. A Figura 36 mostra o circuito emissor IR pronto para ser instalado nas proximidades do ar condicionado.

Figura 36 – Circuito emissor IR pronto para ser instalado no LPA



Fonte: Autor, 2024

As Figuras 37 e 38 mostram, respectivamente, os sensores MQ135 e SR602 instalados. O sensor de movimento SR602 foi instalado nas proximidades da porta, enquanto o sensor de gás MQ135 foi instalado na eletrocalha.

Figura 37 – Sensor MQ135 instalado na eletrocalha do LPA



Fonte: Autor, 2024

Figura 38 – Sensor SR602 instalado próximo a porta do LPA



Fonte: Autor, 2024

Para realizar a leitura do sensor SCT13, foi necessário ter acesso as fios individualizados do ar condicionado, como descascar o fio de alimentação não é recomendado,

a instalação do SCT13 foi feita no interior do condensador, onde os fios de alimentação apresentam-se separadamente e conectados com os demais circuitos. A Figura 39 mostra o local de instalação do sensor SCT13.

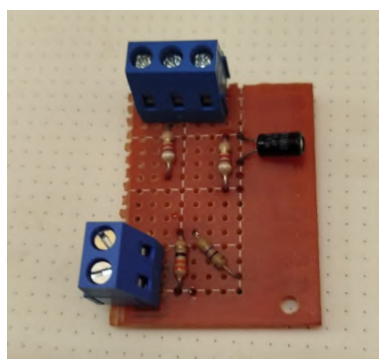
Figura 39 – Sensor SCT13 instalado dentro do módulo condensador



Fonte: Autor, 2024

O circuito desenvolvido para acomodar a leitura do sensor aos valores de operação do ESP32 foi confeccionado separadamente da placa principal, devido a problemas de conexão com o conector P2 que acompanha o sensor. Esse circuito permite identificar se o ar condicionado está ligado ou desligado. O circuito montado é apresentado na Figura 40.

Figura 40 – Circuito instalado em série com o SCT13



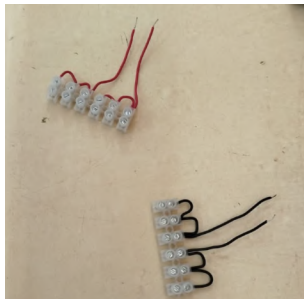
Fonte: Autor, 2024

O borne duplo presente no circuito é a via de entrada do sinal vindo do SCT13, enquanto o borne triplo repassa a saída do circuito para que o ESP32 possa realizar a leitura do sinal de forma segura.

Com todos os componentes do sistema devidamente instalados e validados, os circuitos periféricos foram então conectados na placa principal, bem como os conectores de alimentação dos sensores, Figura 41. Após a conexão, a caixa contendo o circuito principal

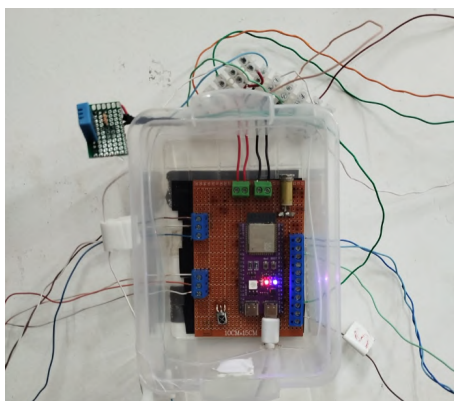
foi fixada na parede do laboratório usando fita dupla face. O protótipo final deste projeto, já instalado no LPA, é apresentado na Figura 42.

Figura 41 – Conectores de alimentação dos sensores



Fonte: Autor, 2024

Figura 42 – Circuito principal instalado no LPA



Fonte: Autor, 2024

5.3 Validação da Aplicação Python

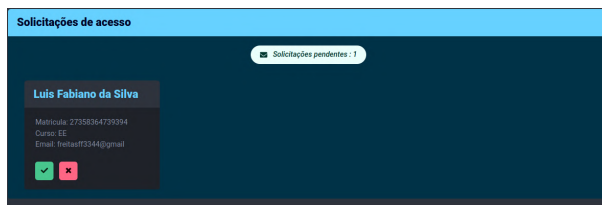
A aplicação Python é a base do sistema *web* que irá receber as informações e permitir o acesso dos usuários ao laboratório, bem como disponibilizará os dados de monitoramento em tempo real para os administradores do sistema.

Ao acessar a aplicação *web* será apresentada a tela de login. A partir de então é possível fazer login na aplicação, caso o usuário já tenha cadastro ativo no banco de dados, ou ainda solicitar acesso ao laboratório preenchendo o formulário de cadastro.

Feito o registro, será necessário aguardar a aprovação da solicitação de acesso por parte de um dos administradores do sistema. Dessa forma, enquanto a solicitação não for aprovada, o status do usuário ficará *Aguardando análise* e não será possível acessar o laboratório. O banco de dados armazenará as informações da solicitação e a aplicação irá

notificar os administradores do sistema que há solicitações pendentes, como é mostrado na Figura 43.

Figura 43 – Solicitações pendentes.



Fonte: Autor, 2024

Uma vez com o usuário ativo, é possível realizar login no sistema para acessar o laboratório, porém as funcionalidades da aplicação variam de acordo com o perfil de usuário.

Usuários comuns são limitados a acessar o laboratório e podem apenas editar suas próprias informações pessoais. Em contrapartida, usuários administradores tem acesso ao *dashboard* com informações tanto dos sensores como dos usuários do laboratório. Além disso, os administradores do sistema também possuem permissão para aprovar ou reprovar solicitações de acesso, editar livremente os dados de qualquer usuário, bem como monitorar e controlar os aparelhos de ar condicionado de forma remota.

Na Figura 44 está apresentado o *dashboard* do usuário administrador, ele possui diferentes áreas, cada uma com sua funcionalidade. Na primeira aba os dados dos sensores são apresentados em tempo real através da conexão *WebSocket* entre cliente-servidor.

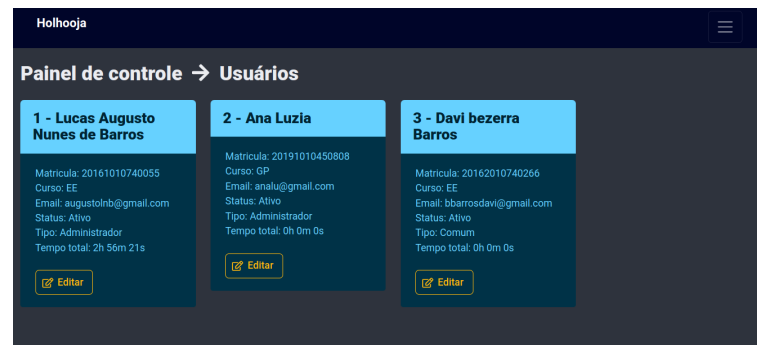
Figura 44 – *Dashboard* do usuário administrador



Fonte: Autor, 2024

Dentro da segunda aba, Painel de Controle, existem duas seções. A seção de *usuários*, apresentada na Figura 45, é responsável pela gerencia dos usuários, sendo possível verificar e editar os dados de qualquer usuário, bem como verificar o total de horas que cada usuário permaneceu no laboratório.

Figura 45 – Painel de controle dos usuários



Fonte: Autor, 2024

Ainda na seção *usuários*, acessando a página de edição de um determinado usuário, Figura 46, é possível fazer alterações em quaisquer informações, como dados pessoais, senha, tipo e *status* desse usuário.

Figura 46 – Página de edição do usuário

A página de edição do usuário no sistema Holhooja, intitulada 'Editar dados de: Lucas Augusto Nunes de Barros', contém um formulário com os seguintes campos e opções:

- Nome completo:
- Matricula:
- Email:
- Tipo do usuário:
- Status:
- Curso atual: Engenharia Elétrica
- Alternar curso: Engenharia Elétrica (menu suspenso)
- Botões de ação: Salvar (verde), Tornar Consumidor (verde), Mudar Senha (verde), Inativar (verde), Voltar (azul).

Fonte: Autor, 2024

Na seção *laboratório*, Figura 47, são disponibilizados os status do ar condicionado e da iluminação, sendo possível também controlar o ar condicionado através da aplicação.

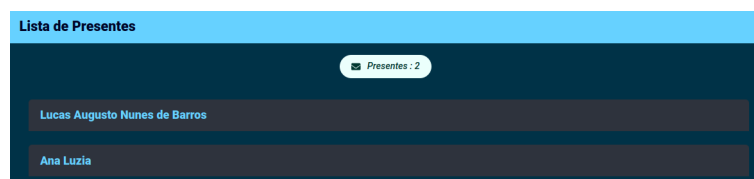
Figura 47 – Painel de controle do laboratório



Fonte: Autor, 2024

Por último, na parte inferior do *dashboard* está a lista de presentes em tempo real, Figura 48, onde é possível para os administradores verificar quais usuários estão conectados no sistema.

Figura 48 – Lista de presentes no laboratório



Fonte: Autor, 2024

5.4 *Firmware do ESP32*

O *firmware* desenvolvido para o ESP32 tem como objetivo principal gerenciar a comunicação dos dispositivos periféricos e estabelecer uma conexão com o *broker* MQTT. Estruturado em dois arquivos, um principal e uma biblioteca, o código foi implementado para configurar o ESP32 à gerenciar a leitura e envio dos dados de sensores e o controle de atuadores. Esta estrutura modular facilita a manutenção e expansão do sistema, conferindo ao projeto grande capacidade de adaptação à outras aplicações envolvendo automação de ambientes.

Para o pleno funcionamento da aplicação, foi necessário utilizar bibliotecas de terceiros, que permitem a leitura e manipulação de dados dos sensores, o controle de dispositivos e a comunicação via rede. Funções específicas à aplicação também foram implementadas neste projeto, incluídas em 'holhooja.h'. Os códigos estão disponíveis no Apêndice A.

Para melhorar a manutenibilidade ainda foram definidas constantes para parametrizar a aplicação, centralizando configurações como tempos de leitura, pinos GPIO e informações sobre conexões Wi-Fi e MQTT. Essa modulação permite realizar ajustes rápidos e melhora a organização do código, tornando-o mais flexível e fácil de manter.

O desenvolvimento do *firmware* para o ESP32, permitiu a criação de uma solução robusta e eficiente para a automação do laboratório. O projeto foi planejado de forma que cada etapa do desenvolvimento foi direcionada para facilitar a modulação e manutenção do sistema. As bibliotecas utilizadas desempenharam um papel fundamental ao fornecer suporte para a comunicação via Wi-Fi e MQTT, bem como para a integração dos diversos sensores e atuadores empregados no projeto.

A implementação do *firmware* demonstra a flexibilidade e capacidade de aplicações do ESP32. O sistema desenvolvido agora proporciona uma base sólida que poderá ser aprimorada em trabalhos futuros e adaptada conforme as necessidades de outros projetos. Desta forma, o trabalho aqui apresentado contribui também no âmbito acadêmico mostrando que com a combinação correta de conhecimentos é possível criar soluções inteligentes e inovadoras.

6 CONSIDERAÇÕES FINAIS

Após descrever os resultados apresentados no Capítulo 4 é possível observar que todas etapas do circuito proposto foram concluídas com êxito.

O circuito de aquisição de dados opera perfeitamente, coletando a leitura dos sensores e as enviando ao broker MQTT, que por sua vez repassa à aplicação que apresenta aos usuários via *dashboard* e armazena as informações no banco de dados.

6.1 Problemáticas

Durante a etapa de instalação do relé na porta do laboratório, ocorreu um problema em que o relé apresentou uma avaria, aparentemente danificando sua bobina. Um relé idêntico foi gentilmente fornecido pelo Prof. Dr. Marcus André Pereira Oliveira, devido a atrasos no prazo de entrega do novo relé, podendo então prosseguir com a substituição do dispositivo danificado, Figura 49.

Figura 49 – Relé novo



Fonte: Autor, 2024

Também foi verificado que a bobina da porta estava avariada, pois apresentava valor de resistência extremamente baixo, como pode ser verificado nas Figuras 50 e 51.

Rapidamente um chamado no sistema do IFTO foi aberto e a fechadura foi consertada.

A aplicação instalada no servidor que se encontra no laboratório sofre alguns problemas de conexão com a rede do IFTO devido a presença dos diversos *firewalls* e procedimentos de segurança inerentes a uma rede institucional do porte da RNP. Por isso os testes foram realizados utilizando uma rede local gerada por um *access point* remoto, que disponibiliza um sinal Wi-Fi para o servidor e o ESP32. Uma forma de contornar essa situação é utilizando um serviço de VPN, porém tais serviços são pagos, o que vai contra

Figura 50 – Medição de resistência da bobina



Fonte: Autor, 2024

Figura 51 – Valor de resistência da bobina



Fonte: Autor, 2024

a ideia principal deste trabalho que é gerar um sistema de automação de baixo custo. Outra possibilidade é obter uma autorização de acesso das portas da conexão servidor e do serviço de comunicação MQTT, ambas bloqueadas pelo *firewall* da instituição.

O controle da iluminação foi projetado no sistema porém não foi executado, sua estrutura de banco de dados e funções por parte da aplicação foram deixados prontos, com o objetivo de facilitar a adição dessa funcionalidade ao sistema.

Após a instalação, o sensor LDR passou a apresentar comportamento similar ao de um curto-circuito. Essa condição pode ter sido causada por atritos e/ou curtos-circuitos envolvendo a eletrocalha. Embora o componente tenha falhado, ele não era parte essencial do projeto, permitindo uma fácil substituição.

Outro componente que apresentou problemas no funcionamento foi o módulo *buzzer*, sendo retirado do projeto e substituído apenas por um sinal luminoso gerado pelo LED *build-in* do ESP32.

6.2 Trabalho Futuros

Foram identificados alguns quesitos em relação ao projeto de *hardware*, observados e também sugeridos para trabalhos futuros sobre o protótipo desenvolvido, de forma a otimizar o sistema e apresentar maior robustez. Considerando alguns aspectos adicionais em relação aos aspectos que são apresentados neste trabalho, algumas das sugestões são:

- Adicionar o controle da iluminação do laboratório;
- Realizar o controle do segundo ar condicionado do laboratório;
- Substituir o sensor LDR e o módulo *buzzer* danificados;

-
- Confeccionar uma versão do *hardware* em placa em PCB;
 - Verificar a disponibilidade de formas alternativas de conexão *web* e MQTT;
 - Realizar a containerização do projeto, de forma a torná-lo ainda mais portátil e escalável;
 - Solicitar uma excessão às portas 8000 e 1883 no *firewall* da instituição;

REFERÊNCIAS

- AGUIAR, V. C. Sistema automatizado de irrigação e monitoramento para plantas em ambientes indoor. 2021. Disponível em: <<https://repositorio.ufu.br/bitstream/123456789/34084/1/SistemaAutomatizadoIriga%C3%A7%C3%A3o.pdf>>. Acesso em: 10 abr. 2024.
- ARDUINO. What is arduino? 2019. Disponível em: <<https://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 23 nov. 2023.
- BULMA. *Documentação Online*. 2020. Disponível em: <<https://bulma.io/>>. Acesso em: 25 mar. 2024.
- CASTRO, G. R. S. S. J. E. C. Desenvolvimento de um protótipo de eletrocardiógrafo de baixo custo. *INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO TOCANTINS*, Palmas - TO, 2023. Disponível em: <<https://ifto.edu.br/palmas/campus-palmas/ensino2/biblioteca/Acervo/trabalhos-academicos>>. Acesso em: 27 jan. 2024.
- CAVALCANTE, A. Projeto ubiquo: Sistema de monitoramento e controle de ar-condicionado. 2019. Disponível em: <<https://repositorio.ufpb.br/jspui/bitstream/123456789/15834/1/ASC23112018.pdf>>. Acesso em: 19 nov. 2023.
- CORREA, M. J. da Cunha; Marcelo Barros de Almeida; Josué Silva de Moraes; Remington Phelipe da S. simulação de aplicações utilizando o protocolo de comunicação mqtt com aplicações em ambientes industriais. *XIV Conferência de Estudos em Engenharia Elétrica*, 2016. Disponível em: <https://www.peteletricaufu.com.br/static/ceel/doc/artigos/artigos2016/ceel2016_artigo108_r01.pdf>. Acesso em: 4 nov. 2023.
- ESPRESSIF. Placas eletrônicas de desenvolvimento 2018a. 2018. Disponível em: <<https://www.espressif.com/en/products/hardware/development-boards>>. Acesso em: 21 set. 2023.
- FONSECA, H. J. da Silva Santos; Juarez da Paz Santos; Marcos José Alef Toloni Moreno de Jesus; Rafael Themístocles Ribeiro de C. J. *IOT PARA CONTROLE E GERENCIAMENTO RESIDENCIAL*. 2023. Disponível em: <<https://revistaft.com.br/iot-para-controle-e-gerenciamento-residencial/>>. Acesso em: 11 mai. 2024.
- GARBADE, D. M. J. *Django, Flask ou Pyramid: Qual é o melhor framework Python para você?* 2007. Disponível em: <<https://educationecosystem.com/blog/django-flask-pyramid-framework-python/>>. Acesso em: 19 mai. 2024.
- GODOI, P. H. D. P. J. C. D. Testador de teor alcoólico para avaliação de segurança de empregados em processos críticos. 2018. Disponível em: <<https://www.eletrica.ufpr.br/p/arquivostccs/505.pdf>>. Acesso em: 05 abr. 2024.
- GRINBERG, M. *The Flask Mega-Tutorial*. 2024. Disponível em: <<https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>>. Acesso em: 2 abr. 2024.
- HOPPEN, A. F. de N. G. M. O. B. T. Sistema supervisorio para monitoramento de consumo de água. *Trabalho de Conclusão de Curso, Universidade Tecnológica Federal do*

- Paraná, Curitiba, 2016. Disponível em: <<http://repositorio.utfpr.edu.br/jspui/handle/1/10055>>. Acesso em: 9 out. 2023.
- JANNUZZI, G. D. M. Aumentando a eficiência nos usos finais de energia no brasil. In: *Trabalho apresentado no evento de Sustentabilidade na geração e no uso de energia. Departamento de Energia, Faculdade de Engenharia Mecânica. UNICAMP.* [s.n.], 2001. Disponível em: <http://www.moretti.agrarias.ufpr.br/eletrificacao_rural/tc_06.pdf>. Acesso em: 19 set. 2023.
- JUNIOR ; FARINELLI, F. A. S. L. S. *Domótica: Automação residencial e casas inteligentes com Arduino e ESP8266*. [S.l.]: Érica, 2018.
- KODALI, S. S. R. K. Mqtt based home automation system using esp8266. *Department of Electronics and Communication Engineering, National Institute Of Technology, Warangal*, 2016. Disponível em: <https://www.researchgate.net/publication/316448543_MQTT_based_home_automation_system_using_ESP8266>. Acesso em: 20 jan. 2024.
- KUROISHI, A. M. Projeto e desenvolvimento de um medidor de fator de potência com arduino uno e arm cortex-m4fmon. *TCC (Graduação) - Curso de Engenharia Elétrica, Universidade Estadual de Londrina*, 2018. Disponível em: <<https://repositorio.ifpb.edu.br/jspui/bitstream/177683/2640/1/TCC%20Fernanda%20Fernandes%20de%20Oliveira.pdf>>. Acesso em: 22 fev. 2024.
- LEIFER, M. C. C. *Learning Flask Framework*. [S.l.]: Packt, 2015.
- LOCATELLI, C. *Introdução ao MQTT*. 2016. Disponível em: <<https://curtocircuito.com.br/blog/Categoria%20IoT/monitoramento-e-controle-por-aplicativo-mqtt>>. Acesso em: 21 mar. 2024.
- LOPES, B. A. L. Plataforma local de controle de acesso para ambientes acadêmicos. *UFPE*, 2020. Acesso em: 19 mai. 2024.
- MCROBERTS. *Arduino Básico*. 1. ed. São Paulo: Novatec, 2011. 443 p.
- MURATORI, P. B. J. R. Automação residencial: Histórico, definições e conceitos. 2016. Disponível em: <https://www.osetoelettrico.com.br/wp-content/uploads/2011/04/Ed62_fasc_automacao_capI.pdf>. Acesso em: 9 out. 2023.
- NERI, M. L. G. B. Mqtt. *UFRJ*, 2019. Disponível em: <<https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/>>. Acesso em: 11 set. 2023.
- OLIVEIRA, I. F. Desenvolvimento de um sistema de automação residencial baseado em iot para controle e monitoramento de dispositivos elétricos. 2019. Disponível em: <https://www.monografias.ufop.br/bitstream/35400000/1795/1/MONOGRAFIA_DesenvolvimentoSistemaAutoma%C3%A7%C3%A3o.pdf>. Acesso em: 3 out. 2023.
- OLIVEIRA, R. R. Uso do microcontrolador esp8266 para automação residencial. *Projeto de Graduação - Curso de Engenharia de Controle e Automação - Universidade Federal do Rio de Janeiro, Escola Politécnica*, Rio de Janeiro, 2017. Disponível em: <<http://repositorio.poli.ufrj.br/monografias/monopoli10019583.pdf>>. Acesso em: 13 out. 2023.
- PICARD, R. *Handling forms*. 2016. Disponível em: <https://explore-flask.readthedocs.io/_/downloads/en/latest/pdf/>. Acesso em: 27 mai. 2024.

- PIRES, E. H. S. Desenvolvimento de medidor de energia de baixo custo aplicado a internet das coisas utilizando esp-32. *UNIVERSIDADE FEDERAL DE UBERLÂNDIA*, Uberlândia - MG, 2018. Disponível em: <<https://repositorio.ufu.br/bitstream/123456789/20601/1/ProjetoUnidadeMonitoramento.pdf>>. Acesso em: 29 jan. 2024.
- QUALHATO, B. S. Desenvolvimento do sistema webcotação: Aplicação na crv industrial. 2023. Disponível em: <<https://repositorio.ifgoiano.edu.br/bitstream/prefix/3733/1/TCC-%20Bruno.pdf>>. Acesso em: 19 set. 2023.
- RONACHER, A. *Flask User's Guide*. 2010. Disponível em: <<https://flask.palletsprojects.com/en/3.0.x/>>. Acesso em: 17 fev. 2024.
- SOARES, F. G. Desenvolvimento de um protÓtipo demonstrativo de automaÇão residencial, utilizando microcontrolador esp32. 2023. Disponível em: <<https://engeletrica-faeng.ufms.br/files/2023/08/TCC-LGJ-AutomacaoResidencial-FernandoGomes-2023.pdf>>. Acesso em: 05 jun. 2024.
- SYSTEMS, E. Esp8266ex datasheet. *Espressif Systems*, 2019. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 2 nov. 2023.
- THOMAZINI, A. *Sensores Industriais – Fundamentos e Aplicações*. 5. ed. São Paulo: Érica, 2005.
- VIANNA, G. P. Domótica: Automação residencial com baixo custo utilizando o arduino. - trabalho de conclusão de curso (bacharel em engenharia elétrica). 2018. Disponível em: <<https://www.unifacvest.edu.br/assets/uploads/files/arquivos/8873f-vianna,-g.-p.-domotica-automacao-residencial-com-baixo-custo-utilizando-o-arduino.-tcc,-2018.pdf>>. Acesso em: 09 abr. 2024.
- YUAN, M. Conhecendo o mqtt. *IBM*, 2017. Disponível em: <<<https://www.ibm.com/developerworks/br/library/iotmqtt-why-good-for-iot/index.html>>>. Acesso em: 9 nov. 2023.

APÊNDICE A – FIRMWARE DO ESP32

A.1 main.ino

```

1  #include "DHTStable.h"
2  #include "holhooja.h"
3  #include <Arduino.h>
4  #include <IRrecv.h>
5  #include <IRutils.h>
6  #include <IRsend.h>
7  #include <WiFi.h>
8  #include <PubSubClient.h>
9
10 #include "EmonLib.h"           // Include Emon Library
11 EnergyMonitor emon1; //
12 //
13 //
14 //  DEFINIÇÃO DAS CONSTANTES
15 //
16 const int MovementTimer = 4000; // 4s
17 const int DHT11_PIN = 21;
18 const int DHTReadTimer = 8000; // 8s
19 const int MQ135ReadTimer = 8000; // 8s
20 const int SCTReadTimer = 8000;
21 const int led = 18;
22 const int MQ135_D_PIN = 2;
23 const int MQ135_A_PIN = 1;
24 const int LDR_PIN = 20;
25 const int LDRReadTimer = 8000; // 8s
26 const int kRecvPin = 10;
27 const int sctPin = 18;
28 const int motionSensor = 19;
29 const uint8_t kTimeout = 50;
30 const uint16_t kSendIRLed = 40;
31 const uint16_t kCaptureBufferSize = 1024;
32 const int RelePin = 15; // pino ao qual o Módulo Relé está conectado
33
34 // ADICIONAR LEITOR DE CORRENTE
35
36
37
38 // CONSTANTES DE CONEXÃO MQTT
39 IPAddress mqtt_server(192, 168, 137, 57);
40 const int mqtt_port = 1883;
41

```

```
42 // CONSTANTES DE CONEX O WIFI
43 const char *ssid = "holhooja";
44 const char *password = "testelpa";
45
46 //
47 // CRIA O DOS OBJETOS DE CADA CLASSE
48 //
49 DHTStable DHT;
50 IRrecv irrecv(kRecvPin, kCaptureBufferSize, kTimeout, true);
51 IRsend irsend(kSendIRLed);
52 WiFiClient wifiClient;
53 PubSubClient mqttClient(wifiClient);
54
55 //
56 // PROT TIPOS DAS FUN ES DEFINIDAS EM <holhooja.h>
57 //
58 void setup_wifi(const char *ssid, const char *password);
59 void connect_MQTT(IPAddress mqtt_server, int mqtt_port, PubSubClient *
    mqttClient);
60 void IRAM_ATTR detectsMovement();
61 void ler_Movimento(int MovementTimer, int led);
62 void ler_SinalIR(IRrecv *irrecv);
63 void ler_DHT(DHTStable *DHT, PubSubClient *mqttClient, int DHT11_PIN, int
    DHTReadTimer);
64 void ler_ldr(int LDR_PIN, PubSubClient *mqttClient, int LDRReadTimer);
65 void verificarConexao(PubSubClient *mqttClient);
66 void ler_sct(int sctPin, EnergyMonitor *emon1, int SCTReadTimer,
    PubSubClient *mqttClient);
67
68 //void ler_sct();
69
70 //
71 //
72 //
73
74 void callback(char *topic, byte *message, unsigned int length) {
75     Serial.print("Mensagem entregue no t pico: ");
76     Serial.print(topic);
77     Serial.print("] ");
78     Serial.print(". Message: ");
79     String messageTemp;
80
81     for (int i = 0; i < length; i++) {
82         Serial.print((char)message[i]);
83         messageTemp += (char)message[i];
84     }
85     Serial.println();
```



```
86
87 if (String(topic) == topicEspAr) {
88     Serial.println("Comandos: ");
89
90     if (messageTemp == "on") {
91         Serial.println("on");
92         enviar_SinalIR(&irsend, 1);
93         //neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, RGB_BRIGHTNESS,
94             RGB_BRIGHTNESS);
95     } else if (messageTemp == "off") {
96         Serial.println("off");
97         enviar_SinalIR(&irsend, 0);
98         //neopixelWrite(RGB_BUILTIN, 0, 0, 0);
99     } else if (messageTemp == "up") {
100         Serial.println("up");
101         neopixelWrite(RGB_BUILTIN, 0, 0, RGB_BRIGHTNESS); // Blue
102     } else if (messageTemp == "down") {
103         Serial.println("down");
104         neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, RGB_BRIGHTNESS, 0); //
105             Yellow
106     } else if (messageTemp == "left") {
107         Serial.println("left");
108         neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, 0, 0);
109     } else if (messageTemp == "right") {
110         Serial.println("right");
111         neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, 0, RGB_BRIGHTNESS);
112     }
113     /*
114     // Vermelho
115     neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, 0, 0);
116
117     // Roxo
118     neopixelWrite(RGB_BUILTIN, RGB_BRIGHTNESS, 0, RGB_BRIGHTNESS);
119
120     */
121 }
122
123 if (String(topic) == topicEspPorta) {
124     if (messageTemp == "porta") {
125         digitalWrite(RelePin, LOW); //aciona o pino
126         delay(100);
127         digitalWrite(RelePin, HIGH); //aciona o pino
128         delay(100);
129         enviar_SinalIR(&irsend, 1);
130     }
131 }
```

```

131
132     }
133
134     if (String(topic) == topicEspLED) {
135         Serial.println("LED");
136     }
137 }
138
139 //
140 //   FUNÇÃO DE CONFIGURAÇÃO
141 //
142 void setup() {
143     Serial.begin(115200);                // Inicia a comunicação serial
144     pinMode(motionSensor, INPUT_PULLUP); // PIR Motion Sensor mode
145     // MQ135 SENSOR GPIO
146     pinMode(MQ135_D_PIN, INPUT);
147     attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement,
148                     RISING); // Defini uma interrupção em caso de leitura do sensor de
149                               movimento
150     pinMode(led, OUTPUT);
151     digitalWrite(led, LOW);              // LED DE TESTE
152     pinMode(kRecvPin, INPUT);            // Configurar o pino
153     // do receptor IR
154     irrecv.enableIRIn();                 // Inicia o receptor
155     // IR
156     irsend.begin();                     // Inicia o emissor
157     // IR
158     setup_wifi(ssid, password);          // Inicia conexão
159     // wifi
160     connect_MQTT(mqtt_server, mqtt_port, &mqttClient); // Estabelece
161     // conexão com o Broker MQTT e faz o SUBSCRIBE dos tópicos
162     verificarConexao(&mqttClient);
163     mqttClient.setCallback(callback);
164     // Sensor de Corrente
165     emon1.current(sctPin, 111.1);        // Current: input pin,
166     // calibration.
167     pinMode(RelePin, OUTPUT); // seta o pino como saída
168     digitalWrite(RelePin, HIGH);
169
170 }
171
172 void loop() {
173     mqttClient.loop();                  // Mantém a conexão com
174     // o broker
175     ler_SinalIR(&irrecv);               // Verificar se um sinal
176     // IR foi recebido

```

```

167 ler_Movimento(MovementTimer, led, &mqttClient); // Aguarda timeSeconds
    para registrar um novo acionamento e no caso de movimento o led liga
168 ler_DHT(&DHT, &mqttClient, DHT11_PIN, DHTReadTimer);
169 ler_ldr(LDR_PIN, &mqttClient, LDRReadTimer);
170 ler_mq135(&mqttClient, MQ135_D_PIN, MQ135_A_PIN, MQ135ReadTimer);
171 ler_sct(sctPin, &emon1, SCTReadTimer, &mqttClient);
172
173 delay(500);
174 }

```

Lista de Ilustrações A.1 – Código principal do *firmware* embarcado no ESP32

A.2 holhooja.ino

```

1
2 #include <IRrecv.h>
3 #include <IRsend.h>
4 #include <WiFi.h>
5 #include <PubSubClient.h>
6 #include "EmonLib.h"
7
8 uint16_t ligar_17[199] = {4376, 4388, 542, 1624, 514, 554, 542, 1624,
    514, 1650, 516, 526, 542, 552, 544, 1622, 514, 552, 544, 526, 540,
    1650, 516, 526, 540, 554, 542, 1624, 514, 1652, 514, 526, 544,
    1650, 516, 524, 544, 554, 540, 1624, 516, 1650, 514, 1650, 514,
    1624, 514, 1650, 516, 1624, 514, 1650, 516, 1624, 516, 552, 544,
    526, 540, 554, 542, 526, 544, 552, 542, 526, 544, 550, 544, 526,
    542, 528, 566, 528, 542, 526, 544, 552, 542, 526, 542, 552, 544,
    1624, 516, 1650, 514, 1622, 542, 1624, 516, 1648, 516, 1622, 516,
    1648, 514, 1624, 516, 5198, 4378, 4388, 542, 1624, 516, 552, 542,
    1622, 516, 1650, 514, 526, 544, 550, 544, 1624, 514, 554, 542,
    526, 542, 1650, 516, 526, 542, 552, 544, 1622, 516, 1648, 516,
    526, 542, 1648, 516, 524, 572, 526, 542, 1650, 516, 1622, 514,
    1650, 514, 1624, 514, 1652, 514, 1622, 516, 1648, 516, 1650, 516,
    526, 542, 554, 542, 526, 542, 528, 544, 550, 542, 526, 542, 552,
    544, 526, 542, 552, 542, 528, 542, 552, 544, 524, 544, 528, 542,
    552, 542, 1650, 514, 1624, 516, 1650, 516, 1624, 516, 1648, 516,
    1622, 516, 1648, 516, 1650, 516};
9
10 uint16_t ligar_20[199] = {4376, 4388, 542, 1624, 514, 552, 544, 1622,
    518, 1648, 516, 526, 544, 552, 542, 1624, 516, 552, 544, 524, 544,
    1648, 514, 528, 542, 552, 544, 1624, 514, 1650, 516, 526, 542,
    1648, 516, 526, 544, 552, 542, 1622, 516, 1650, 516, 1650, 514,
    1624, 516, 1650, 514, 1622, 516, 1652, 512, 1624, 516, 552, 544,
    524, 544, 554, 542, 528, 542, 550, 544, 526, 544, 550, 544, 526,
    542, 1648, 516, 526, 542, 552, 544, 524, 544, 526, 544, 550, 544,
    1624, 514, 1650, 514, 528, 568, 1624, 516, 1650, 516, 1622, 516,

```

```

1648, 516, 1622, 516, 5196, 4378, 4388, 544, 1624, 514, 552, 544,
1622, 514, 1650, 516, 526, 542, 554, 542, 1624, 516, 552, 544,
526, 544, 1648, 516, 526, 542, 552, 542, 1624, 516, 1648, 516,
528, 542, 1648, 516, 526, 568, 526, 544, 1648, 516, 1624, 514,
1650, 516, 1624, 514, 1648, 516, 1624, 514, 1648, 516, 1650, 516,
526, 542, 554, 542, 526, 542, 526, 544, 552, 542, 526, 542, 554,
542, 526, 542, 1648, 516, 528, 542, 552, 542, 528, 540, 554, 540,
526, 544, 1650, 514, 1622, 516, 554, 542, 1620, 514, 1650, 514,
1622, 516, 1648, 516, 1648, 518}; // COOLIX B23F20

11
12 uint16_t desligar[199] = {4376, 4388, 544, 1622, 514, 554, 542, 1622,
516, 1650, 514, 528, 544, 552, 540, 1622, 518, 552, 542, 526, 544,
1650, 514, 526, 544, 552, 544, 1622, 514, 1650, 516, 526, 542,
1648, 516, 528, 540, 1650, 516, 1622, 516, 1648, 516, 1648, 518,
524, 544, 1648, 516, 1622, 516, 1648, 516, 526, 542, 554, 542,
526, 540, 554, 542, 1622, 516, 552, 542, 528, 542, 1650, 516,
1622, 516, 1650, 514, 526, 544, 552, 542, 526, 542, 552, 542, 526,
542, 552, 542, 526, 542, 526, 570, 1622, 516, 1648, 516, 1624,
516, 1650, 516, 1622, 516, 5196, 4378, 4388, 544, 1622, 514, 554,
542, 1622, 516, 1650, 516, 528, 542, 552, 544, 1624, 516, 550,
542, 526, 542, 1648, 518, 524, 542, 556, 540, 1624, 514, 1648,
516, 524, 544, 1648, 516, 526, 568, 1622, 516, 1648, 518, 1622,
516, 1650, 516, 526, 544, 1648, 516, 1622, 516, 1648, 516, 552,
542, 526, 544, 528, 540, 554, 542, 1622, 516, 552, 542, 524, 544,
1650, 516, 1650, 516, 1622, 516, 552, 544, 526, 544, 526, 542,
552, 544, 526, 542, 552, 544, 526, 544, 552, 542, 1622, 516, 1650,
516, 1624, 516, 1648, 516, 1648, 516}; // COOLIX B27BE0

13
14 boolean startTimer = false;
15 int i;
16 unsigned long lastMq135Read;
17 unsigned long lastSctRead;
18 unsigned long lastDhtRead;
19 unsigned long lastLdrRead;
20 unsigned long lastSrRead;
21
22 // T picos MQTT
23 const char* topicEspAr = "/comando/esp32/AC"; // subscribe
24 const char* topicEspPorta = "/comandos/esp32/porta"; // subscribe
25 const char* topicEspLED = "/comando/esp32/lampada"; // subscribe
26 const char* topicSensorDHT = "/sensores/DHT"; // publish
27 const char* topicSensorLDR = "/sensores/LDR"; // publish
28 const char* topicSensorSR602 = "/sensores/SR602"; // publish
29 const char* topicSensorSCT = "/sensores/SCT"; // publish
30 const char* topicSensorMQ135 = "/sensores/MQ135"; // publish
31
32 void setup_wifi(const char *ssid, const char *password){

```

```
33   delay(10);
34   Serial.println("");
35   Serial.print("Conectando a rede");
36   Serial.println(ssid);
37
38   WiFi.mode(WIFI_STA);
39   WiFi.begin(ssid, password);
40
41   while(WiFi.status() != WL_CONNECTED){
42     delay(500);
43     Serial.print(".");
44   }
45
46   randomSeed(micros());
47
48   Serial.println("");
49   Serial.println("WiFi conectado");
50   Serial.print("IP: ");
51   Serial.println(WiFi.localIP());
52 }
53
54 void enviar_SinalIR(IRsend *irsend, int comando){ // 1->liga 17 c    2->
    ligar 20 c    0->desliga
55   Serial.println(comando);
56   if(comando == 0)
57     irsend->sendRaw(desligar, 199, 38); // Send a raw data capture at 38
        kHz.
58
59   if(comando == 1)
60     irsend->sendRaw(ligar_17, 199, 38); // Send a raw data capture at 38
        kHz.
61
62   if(comando == 2)
63     irsend->sendRaw(ligar_20, 199, 38); // Send a raw data capture at 38
        kHz.
64
65   Serial.println("C digo enviado");
66   delay(100);
67 }
68
69 void ler_SinalIR(IRrecv *irrecv){
70   decode_results results;
71   if (irrecv->decode(&results)) {
72     unsigned long hex = results.value;
73
74     //   Serial.println(resultToHumanReadableBasic(&results));
75     /*
```

```
76     if(hex == 0x1FFF807){
77         for(i=0;i<10;i++){
78             digitalWrite(ledIR, HIGH);
79             delay(100);
80             digitalWrite(ledIR, LOW);
81             delay(100);
82         }
83     }
84     */
85     if(hex == 0x1FFC03F){
86         #ifdef RGB_BUILTIN
87             Serial.println("Bot o esquerdista: altera led vermelho");
88             neopixelWrite(RGB_BUILTIN,RGB_BRIGHTNESS,0,0); // Red
89         #endif
90     }
91
92     if(hex == 0x1FF40BF){
93         #ifdef RGB_BUILTIN
94             Serial.println("Bot o direto: altera led azul");
95             neopixelWrite(RGB_BUILTIN,0,0,RGB_BRIGHTNESS); // Blue
96         #endif
97     }
98
99     if(hex == 0x1FF807F){
100         #ifdef RGB_BUILTIN
101             Serial.println("Bot o p/ baixo: led roxo");
102             neopixelWrite(RGB_BUILTIN,RGB_BRIGHTNESS,0,RGB_BRIGHTNESS); //
103                 Purple
104         #endif
105     }
106
107     if(hex == 0x1FF00FF){
108         #ifdef RGB_BUILTIN
109             Serial.println("Bot o p/ cima: led amarelo");
110             neopixelWrite(RGB_BUILTIN,RGB_BRIGHTNESS,RGB_BRIGHTNESS,0); //
111                 Yellow
112         #endif
113     }
114
115     irrecv->resume();
116 }
117
118 void IRAM_ATTR detectsMovement() {
119     int led = 20;
120     digitalWrite(led, HIGH);
```

```
121     startTimer = true;
122     lastSrRead = millis();
123 }
124
125 void ler_Movimento(int MovementTimer, int led, PubSubClient *mqttClient){
126     unsigned long currentMillis = millis();
127     boolean motion = false;
128
129     if((digitalRead(led) == HIGH) && (motion == false)) {
130         motion = true;
131         char jsonBuffer[128];
132         snprintf(jsonBuffer, 128, "{ 'Movimento' : 1, }");
133         mqttClient->publish(topicSensorSR602, jsonBuffer);
134         // enviar_SinalIR();
135     }
136     if(startTimer && (currentMillis - lastSrRead > MovementTimer)) {
137         Serial.println("Sem movimento...");
138         digitalWrite(led, LOW);
139         startTimer = false;
140         motion = false;
141         char jsonBuffer[128];
142         snprintf(jsonBuffer, 128, "{ 'Movimento' : 0, }");
143         mqttClient->publish(topicSensorSR602, jsonBuffer);
144     }
145 }
146
147 void ler_DHT(DHTStable *DHT, PubSubClient *mqttClient, int DHT11_PIN, int
    DHTReadTimer) {
148     // variaveis para leitura do DHT11
149     int chk = DHT->read11(DHT11_PIN);
150     unsigned long currentMillis = millis();
151     float humidity=0.0;
152     float temperature=0.0;
153
154     switch (chk)
155     {
156         case DHTLIB_OK:
157             //Serial.print("OK,\t");
158             break;
159         case DHTLIB_ERROR_CHECKSUM:
160             Serial.print("Checksum error,\t");
161             break;
162         case DHTLIB_ERROR_TIMEOUT:
163             Serial.print("Time out error,\t");
164             break;
165         default:
166             Serial.print("Unknown error,\t");
```

```

167     break;
168 }
169
170 if (currentMillis - lastDhtRead >= DHTReadTimer) {
171     humidity = DHT->getHumidity();
172     temperature = DHT->getTemperature();
173
174     if (isnan(humidity) || isnan(temperature)) {
175         humidity = 0.0;
176         temperature = 0.0;
177     }
178
179     Serial.printf("Umidade: %.1f%%, Temperatura: %.1f C \n", humidity,
180         temperature);
181
182     lastDhtRead = currentMillis;
183
184     // Create JSON string for MQTT publication
185     char jsonBuffer[128];
186     snprintf(jsonBuffer, 128, "{ 'Umidade' : %.1f, 'Temperatura' : %.1f}",
187         humidity, temperature);
188
189     // Publish sensor data to MQTT topic
190     mqttClient->publish(topicSensorDHT, jsonBuffer);
191 }
192
193 void ler_mq135(PubSubClient *mqttClient, int MQ135_D_PIN, int MQ135_A_PIN,
194     int MQ135ReadTimer) {
195     unsigned long currentMillis = millis();
196     float presenca=0.0;
197     float quantidade=0.0;
198
199     if (currentMillis - lastMq135Read >= MQ135ReadTimer) {
200         presenca = digitalRead(MQ135_D_PIN);
201         ///////////////////////////////////////////////////
202         if (presenca == HIGH)
203             Serial.println("Sem gases nocivos detectados.");
204         else
205             Serial.println("Gases nocivos detectados!");
206         ///////////////////////////////////////////////////
207         quantidade = analogRead(MQ135_A_PIN);
208         Serial.print("Quantidade de g s detectada: ");
209         Serial.println(quantidade);
210         ///////////////////////////////////////////////////

```



```
211     if (isnan(presenca) || isnan(quantidade)) {
212         presenca = 0.0;
213         quantidade = 0.0;
214     }
215
216     //Serial.printf("presenca: %.1f, quantidade: %.1f \n", presenca,
217                     quantidade);
218
219     lastMq135Read = currentMillis;
220
221     // Create JSON string for MQTT publication
222     char jsonBuffer[128];
223     snprintf(jsonBuffer, 128, "{ 'presenca' : %.1f, 'quantidade' : %.1f }",
224             presenca, quantidade);
225
226     if(quantidade > 450){
227         for(i=0;i<10;i++){
228             #ifdef RGB_BUILTIN
229                 neopixelWrite(RGB_BUILTIN,RGB_BRIGHTNESS,0,0); // Red
230                 delay(500);
231                 neopixelWrite(RGB_BUILTIN,0,0,0); // off
232                 delay(500);
233             #endif
234         }
235     }
236
237     // Publish sensor data to MQTT topic
238     mqttClient->publish(topicSensorMQ135, jsonBuffer);
239 }
240
241 void connect_MQTT(IPAddress mqtt_server, int mqtt_port, PubSubClient *
242                  mqttClient){
243     mqttClient->setKeepAlive( 90 );
244     mqttClient->setServer(mqtt_server, mqtt_port);
245
246     while (!mqttClient->connected()) {
247         if (mqttClient->connect("ESP32-S3")) {
248             Serial.println("inscri o aqui");
249             mqttClient->subscribe(topicEspAr);
250             mqttClient->subscribe(topicEspPorta);
251
252             Serial.println("MQTT Conectado > ESP32-S3");
253
254             Serial.println("\nInscrito nos t picos:");
255             Serial.println(topicEspAr);
256             Serial.println(topicEspPorta);
257         }
258     }
259 }
```

```

255     delay(500);
256   } else {
257     Serial.println("MQTT connect fail");
258     delay(3000);
259   }
260 }
261 }
262
263
264 void ler_sct(int sctPin, EnergyMonitor *emon1, int SCTReadTimer,
265   PubSubClient *mqttClient){
266   unsigned long currentMillis = millis();
267   float lastSCTRead=0;
268
269   if (currentMillis - lastSCTRead >= SCTReadTimer) {
270     double Irms = emon1->calcIrms(1480); // Calculate Irms only
271     if (isnan(Irms)) {
272       Irms = 0.0;
273     }
274     /*
275     if (Irms < 4) {
276       Serial.printf("Status: Desligado \n");
277       lastSCTRead = currentMillis;
278     }
279
280     if (Irms > 4) {
281       Serial.printf("Status: Ligado \n");
282       lastSCTRead = currentMillis;
283     }
284     */
285
286     lastSCTRead = currentMillis; // Create JSON string for MQTT publication
287
288     char jsonBuffer[128];
289     snprintf(jsonBuffer, 128, "{ 'Status_AC' : %.1f}", Irms);
290
291     // Publish sensor data to MQTT topic
292     mqttClient->publish(topicSensorSCT, jsonBuffer);
293   }
294 }
295
296 void ler_ldr(int LDR_PIN, PubSubClient *mqttClient, int LDRReadTimer){
297   unsigned long currentMillis = millis();
298
299   if (currentMillis - lastLdrRead >= LDRReadTimer) {
300     int ldr_value = analogRead(LDR_PIN);
301     if (isnan(ldr_value)) {

```

```
301     ldr_value = 0.0;
302 }
303 Serial.printf("Luminosidade: %d \n", ldr_value);
304 lastLdrRead = currentMillis;
305
306 char jsonBuffer[128];
307 snprintf(jsonBuffer, 128, "{ 'Luminosidade' : %d, }", ldr_value);
308 mqttClient->publish(topicSensorLDR, jsonBuffer);
309 }
310 }
311
312 void verificarConexao(PubSubClient *mqttClient){
313
314     mqttClient->subscribe("/esp32/verificarConexao");
315     mqttClient->subscribe("/esp32/enviarComando");
316     printf("Inscrito no t pico: '/esp32/verificarConexao'\n");
317     printf("Inscrito no t pico: '/esp32/enviarComando'\n");
318 }
```

Lista de Ilustrações A.2 – Código da biblioteca de funções do *firmware* embarcado no ESP32

APÊNDICE B – NÚCLEO DA APLICAÇÃO PYTHON

B.1 `__init__.py`

```

1
2 from flask import Flask
3 from config import Config
4 from threading import Lock
5 from flask_login import LoginManager
6 from flask_sqlalchemy import SQLAlchemy
7 from flask_socketio import SocketIO
8 import paho.mqtt.client as mqtt
9 from mqtt import on_connect, on_message, on_publish
10
11 async_mode = None # Modo de sincronia do socket
12 app = Flask(__name__) # Declara o do objeto app
13 app.config.from_object(Config) # Configura o do objeto app segundo o
    arquivo Config
14 db = SQLAlchemy(app) # Declara o do objeto db para manipula o do
    banco de dados
15
16 broker_address = "192.168.171.57"
17 broker_port = 1883
18
19 # Configura o do cliente mqtt
20 client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
21 client.on_connect = on_connect
22 client.on_message = on_message
23 client.on_publish = on_publish
24 client.connect(broker_address, broker_port)
25
26 print('Iniciando loop MQTT')
27 client.loop_start()
28
29 print('Iniciando servidor socket')
30 socketio = SocketIO(app, async_mode=async_mode)
31 login = LoginManager(app)
32 thread = None
33 thread_lock = Lock()
34
35 from app import routes, models
36
37 if __name__ == '__main__':
38     socketio.run(app)

```

Lista de Ilustrações B.1 – Código principal da aplicação Flask

B.2 *sockets.py*

```
1
2 from app import db
3 from app import app
4 from app import socketio
5 from app.models import User, Sensores
6 import sqlalchemy as sa
7 from app import thread_lock, thread
8 from flask import Flask, render_template, session, request, \
9     copy_current_request_context
10 from flask_socketio import SocketIO, emit, join_room, leave_room, \
11     close_room, rooms, disconnect
12
13 def background_thread():
14     """Example of how to send server generated events to clients."""
15     count = 0
16     while True:
17
18         with app.app_context():
19             dados_sensores = db.first_or_404(sa.select(Sensores).where(
20                 Sensores.id == 1))
21             socketio.sleep(3)
22             count += 1
23             socketio.emit('my_response', {
24                 'umidade': dados_sensores.umidade,
25                 'temperatura': dados_sensores.
26                     temperatura,
27                 'luminosidade': dados_sensores.
28                     luminosidade,
29                 'movimento': dados_sensores.
30                     movimento
31             })
32
33 @socketio.event
34 def my_event(message):
35     session['receive_count'] = session.get('receive_count', 0) + 1
36     emit('my_response',
37         {'data': message['data'], 'count': session['receive_count']})
38
39 #
40 # EXEMPLO DE RECEBIMENTO DE INFORMAÇÃO VIA SOCKET E MANIPULAÇÃO DO
41 # BANCO DE DADOS
42 #
43 """
```

```
37     user = User(matricula="12345678901234", nome=message['data'], email="
38         teste@gmail.com", curso="cu")
39     user.set_password("123mudar")
40     user.set_tipo(0) # default
41     user.set_status(0) # aguardando
42     db.session.add(user)
43     db.session.commit()
44     """
45 @socketio.event
46 def my_broadcast_event(message):
47     session['receive_count'] = session.get('receive_count', 0) + 1
48     emit('my_response',
49         {'data': message['data'], 'count': session['receive_count']},
50         broadcast=True)
51     print(message)
52
53 @socketio.event
54 def join(message):
55     join_room(message['room'])
56     session['receive_count'] = session.get('receive_count', 0) + 1
57     emit('my_response',
58         {'data': 'In rooms: ' + ', '.join(rooms()),
59          'count': session['receive_count']})
60
61 @socketio.event
62 def leave(message):
63     leave_room(message['room'])
64     session['receive_count'] = session.get('receive_count', 0) + 1
65     emit('my_response',
66         {'data': 'In rooms: ' + ', '.join(rooms()),
67          'count': session['receive_count']})
68
69 @socketio.on('close_room')
70 def on_close_room(message):
71     session['receive_count'] = session.get('receive_count', 0) + 1
72     emit('my_response', {'data': 'Room ' + message['room'] + ' is closing.'
73         ,
74         'count': session['receive_count']},
75         to=message['room'])
76     close_room(message['room'])
77
78 @socketio.event
79 def my_room_event(message):
80     session['receive_count'] = session.get('receive_count', 0) + 1
81     emit('my_response',
82         {'data': message['data'], 'count': session['receive_count']},
```

```

82         to=message['room'])
83
84 @socketio.event
85 def disconnect_request():
86     @copy_current_request_context
87     def can_disconnect():
88         disconnect()
89
90     session['receive_count'] = session.get('receive_count', 0) + 1
91     # for this emit we use a callback function
92     # when the callback function is invoked we know that the message has
93     # been
94     # received and it is safe to disconnect
95     emit('my_response',
96         {'data': 'Disconnected!', 'count': session['receive_count']},
97         callback=can_disconnect)
98
99 @socketio.event
100 def my_ping():
101     emit('my_pong')
102
103 @socketio.event
104 def connect():
105     global thread
106     with thread_lock:
107         if thread is None:
108             thread = socketio.start_background_task(background_thread)
109     emit('my_response', {'data': 'Connected', 'count': 0})
110
111 @socketio.on('disconnect')
112 def test_disconnect():
113     print('Client disconnected', request.sid)

```

Lista de Ilustrações B.2 – Código reponsável pela comunicação *WebSocket* entre cliente e servidor

B.3 routes.py

```

1
2 from app.sockets import *
3 from app import db
4 from app import app
5 from app import socketio
6 from app import client
7 import sqlalchemy as sa
8 from datetime import datetime, timedelta
9 from urllib.parse import urlsplit

```

```
10 from app.models import User, Acesso, Sensores, Comandos
11 from app.forms import LoginForm, RegistrationForm, EditionForm
12 from flask import render_template, flash, redirect, url_for, request
13 from flask_login import current_user, login_user, login_required,
    logout_user
14
15 @app.route('/')
16 def home():
17     return redirect(url_for('login'))
18
19 @app.route('/index')
20 @login_required
21 def index():
22     # USUARIOS ADMIN
23     if current_user.status == 1 and current_user.tipo == 1:
24         sensores = db.session.scalar(sa.select(Sensores).where(Sensores.id
            == 1))
25         notifs = notifs = db.session.query(User).filter(User.status == 0).
            all()
26         users = db.session.query(Acesso, User.nome).join(User, Acesso.
            user_id == User.id).filter(
27
28                                     Acesso.saida == None
29                                     #(Acesso.entrada) == date.
29                                     today()
30                                     ).all()
31
32         online = []
33
34         for acesso, user in users:
35             if acesso.entrada.date() == datetime.now().date():
36                 online.append(user)
37
38         print("teste")
39         print(online)
40
41         return render_template('index_admin.html', title='Dash admin',
42                                 notifs=notifs, users_on=online, async_mode=socketio.async_mode)
43
44     # USUARIOS DEFAULT
45     if current_user.status == 1 and current_user.tipo == 0:
46         return render_template('index.html', title='Dash default')
47
48     return render_template('index.html', title='Home Page')
49
50 @app.route('/login', methods=['GET', 'POST'])
51 def login():
52     if current_user.is_authenticated:
```



```
51     print("\nUsuario autenticado\n")
52     return redirect(url_for('index'))
53 form = LoginForm()
54 if form.validate_on_submit():
55     user = db.session.scalar(sa.select(User).where(User.matricula ==
56         form.matricula.data))
57     if user is None or not user.check_password(form.senha.data):
58         flash('Matricula ou senha invalidas')
59         return redirect(url_for('login'))
60     if user.status == 0:
61         flash('Usu rio ainda sem aprova o')
62         return redirect(url_for('login'))
63     login_user(user, remember=form.remember_me.data)
64     acesso = Acesso(user_id=user.id)
65     db.session.add(acesso)
66     db.session.commit()
67     client.publish("/comandos/esp32/porta", "porta")
68     next_page = request.args.get('next')
69     if not next_page or urlsplit(next_page).netloc != '':
70         next_page = url_for('index')
71     return redirect(next_page)
72
73 return render_template('login.html', title='Sign In', form=form)
74
75 @app.route('/logout/<id>')
76 def logout(id):
77     logout_user()
78     acesso = Acesso.query.filter_by(user_id=id).order_by(Acesso.id.desc()).
79         first()
80     acesso.saida = datetime.now()
81     db.session.commit()
82     return redirect(url_for('login'))
83
84 @app.route('/register', methods=['GET', 'POST'])
85 def register():
86     if current_user.is_authenticated:
87         return redirect(url_for('index'))
88     form = RegistrationForm()
89     if form.validate_on_submit():
90         user = User(matricula=form.matricula.data, nome=form.nome.data,
91             email=form.email.data, curso=form.curso.data)
92         user.set_password(form.senha.data)
93         user.set_tipo(0) # default
94         user.set_status(0) # aguardando
95         db.session.add(user)
96         db.session.commit()
97         flash('Usuario registrado!')
```

```
95         return redirect(url_for('login'))
96     return render_template('register.html', title='Register', form=form)
97
98 @app.route('/user/<matricula>')
99 @login_required
100 def user(matricula):
101     user = db.first_or_404(sa.select(User).where(User.matricula ==
102         matricula))
103     posts = [
104         {'author': user, 'body': 'Test post #1'},
105         {'author': user, 'body': 'Test post #2'}
106     ]
107     return render_template('user.html', user=user, posts=posts)
108
109 @app.route('/aprovar_status/<matricula>')
110 @login_required
111 def aprovar_status(matricula):
112     user = db.first_or_404(sa.select(User).where(User.matricula ==
113         matricula))
114     user.set_status(1) # aprovado
115     db.session.add(user)
116     db.session.commit()
117     flash('Usuario aprovado!')
118
119     return redirect(url_for('index'))
120
121 @app.route('/reprovar_status/<matricula>')
122 @login_required
123 def reprovar_status(matricula):
124     user = db.first_or_404(sa.select(User).where(User.matricula ==
125         matricula))
126     user.set_status(-1) # reprovado
127     db.session.add(user)
128     db.session.commit()
129     flash('Usuario Reprovado!')
130
131     return redirect(url_for('index'))
132
133 @app.route('/usuarios')
134 @login_required
135 def usuarios():
136     users = db.session.query(User).all()
137
138     tempo_total_dict = []
139
140     for user in users:
141         acessos = db.session.query(Acesso).filter(
```

```
139         Acesso.user_id == user.id,
140         Acesso.saida != None # Certifique-se de que o registro tenha
                                um hor rio de sa da
141     ).all()
142
143     tempo_total = timedelta()
144
145     for acesso in acessos:
146         if acesso.saida and acesso.entrada: # Garantir que ambos os
                                valores existam
147             tempo_permanencia = acesso.saida - acesso.entrada #
                                Calcula o tempo de permanencia
148             tempo_total += tempo_permanencia
149
150     # Formata o tempo total para horas, minutos e segundos
151     horas_totais = tempo_total.total_seconds() // 3600
152     minutos_totais = (tempo_total.total_seconds() % 3600) // 60
153     segundos_totais = tempo_total.total_seconds() % 60
154
155     # Armazena os resultados em um dicionario
156     tempo_total_dict.append({
157         "user_id": user.id,
158         "tempo_total": f"{int(horas_totais)}h {int(minutos_totais)}m {
                                int(segundos_totais)}s"
159     })
160
161
162     print(tempo_total_dict)
163     return render_template('usuarios.html', users=users, lista_tempo=
                                tempo_total_dict)
164
165 @app.route('/usuarios/mudar_tipo/<id>')
166 @login_required
167 def mudar_tipo(id):
168     # users = db.session.query(User).all()
169     user = db.first_or_404(sa.select(User).where(User.id == id))
170     form = EditionForm()
171     if user.tipo == 1:
172         user.set_tipo(0)
173     elif user.tipo == 0:
174         user.set_tipo(1)
175     db.session.add(user)
176     db.session.commit()
177     flash('Cadastro atualizado!')
178     return render_template('editar_usuario.html', title="Editar dados",
                                form=form, user=user)
179
```

```
180 @app.route('/usuarios/mudar_status/<id>')
181 @login_required
182 def mudar_status(id):
183     #     users = db.session.query(User).all()
184     user = db.first_or_404(sa.select(User).where(User.id == id))
185     form = EditionForm()
186     if user.status == 1:
187         user.set_status(-1)
188     elif user.status == -1:
189         user.set_status(1)
190     db.session.add(user)
191     db.session.commit()
192     flash('Cadastro atualizado!')
193     return render_template('editar_usuario.html', title="Editar dados",
194                             form=form, user=user)
194
195 @app.route('/usuarios/editar/<id>')
196 @login_required
197 def editar_usuario(id):
198     form = EditionForm()
199     user = db.session.scalar(sa.select(User).where(User.id == id))
200     return render_template('editar_usuario.html', title="Editar dados",
201                             form=form, user=user)
201
202 @app.route('/usuarios/atualizar/<id>', methods=['GET', 'POST'])
203 @login_required
204 def atualizar_cadastro(id):
205     if request.method == 'POST':
206         user = db.first_or_404(sa.select(User).where(User.id == id))
207         user.set_nome(request.form.get('ed_nome'))
208         user.set_matricula(request.form.get('ed_matricula'))
209         user.set_email(request.form.get('ed_email'))
210         user.set_curso(request.form.get('ed_curso'))
211         db.session.add(user)
212         db.session.commit()
213         flash('Cadastro atualizado!')
214
215         return redirect(url_for('usuarios'))
216
217 @app.route('/usuarios/editar_senha/<id>')
218 @login_required
219 def editar_senha(id):
220     user = db.session.scalar(sa.select(User).where(User.id == id))
221     return render_template('editar_senha.html', title="Editar dados", user=
222                             user)
222
223 @app.route('/usuarios/atualizar_senha/<id>', methods=['GET', 'POST'])
```

```
224 @login_required
225 def atualizar_senha(id):
226     if request.method == 'POST':
227         user = db.first_or_404(sa.select(User).where(User.id == id))
228         if request.form.get('ed_senha') == request.form.get('ed_senha2'):
229             user.set_password(request.form.get('ed_senha'))
230             db.session.add(user)
231             db.session.commit()
232             flash('Senha atualizada!')
233         else:
234             flash('Senhas divergentes! Repita o processo.')
235
236     return redirect(url_for('usuarios'))
237
238 @app.route('/dash_lab')
239 @login_required
240 def dash_lab():
241     #users = db.session.query(User).all()
242     user = db.first_or_404(sa.select(User).where(User.id == 1)) # buscar
        todos usuarios
243     #sensores = db.first_or_404(sa.select(Sensores).where(Sensores.id == 1)
        )
244     sensores = db.session.scalar(sa.select(Sensores).where(Sensores.id ==
        1))
245     print('\n\n Status AC: ')
246     print(sensores.status_ac)
247
248     return render_template('dash_lab.html', title="Dash de Controle", user=
        user, sensores=sensores)
249
250 @app.route('/mqtt/<string:comando>')
251 def mqtt(comando):
252     print("Enviar comando aqui:")
253     print(comando)
254     print("\n")
255
256     if comando == "on" or comando == "off" or comando == "up" or comando ==
        "down" or comando == "left" or comando == "right":
257         client.publish("/comando/esp32/AC", comando)
258
259
260
261     return redirect(url_for('dash_lab'))
```

Lista de Ilustrações B.3 – Arquivo com as rotas e funcionalidades da aplicação

B.4 models.py

```
1
2 from app import db
3 from app import login
4 from flask_login import UserMixin
5 from datetime import datetime, timezone
6 from sqlalchemy.orm import declarative_base, relationship
7 from sqlalchemy import create_engine, Column, Integer, String, DATETIME,
8     ForeignKey, FLOAT, DATE
9
10 import sqlalchemy as sa
11 import sqlalchemy.orm as so
12
13 @login.user_loader
14 def load_user(id):
15     return db.session.get(User, int(id))
16
17 class User(UserMixin, db.Model):
18     __tablename__ = 'user'
19     id = Column(Integer, primary_key=True)
20     nome = Column(String(80), nullable=False)
21     matricula = Column(String(20), nullable=False, unique=True)
22     password_hash = Column(String(250))
23     email = Column(String(80), nullable=False, unique=True)
24     curso = Column(String(3), nullable=False)
25     tipo = Column(Integer) # 1 > admin || 0 > default
26     status = Column(Integer) # 0 > aguardando || 1 > aprovado || -1 >
27         reprovado/inativo
28     data_criacao = Column(DATETIME, nullable=False, default=datetime.now)
29
30     def __repr__(self):
31         return '<User {}>'.format(self.nome)
32
33     def set_nome(self, nome):
34         self.nome = nome
35
36     def set_matricula(self, matricula):
37         self.matricula = matricula
38
39     def set_curso(self, curso):
40         self.curso = curso
41
42     def set_email(self, email):
43         self.email = email
```

```
44     def set_password(self, password):
45         self.password_hash = generate_password_hash(password)
46
47     def check_password(self, password):
48         return check_password_hash(self.password_hash, password)
49
50     def set_tipo(self, tipo):
51         self.tipo = tipo
52
53     def set_status(self, status):
54         self.status = status
55
56 class Acesso(db.Model):
57     __tablename__ = 'acesso'
58     id = Column(Integer, primary_key=True)
59     user_id = Column(Integer, ForeignKey('user.id'))
60     entrada = Column(DATETIME, nullable=False, default=datetime.now)
61     saida = Column(DATETIME)
62
63     def __repr__(self):
64         return '<Acesso ID: {}>'.format(self.id)
65
66 class Sensores(db.Model):
67     __tablename__ = 'sensores'
68     id = Column(Integer, primary_key=True)
69     #amb_id = Column(Integer, nullable=False)
70     umidade = Column(FLOAT)
71     temperatura = Column(FLOAT)
72     luminosidade = Column(Integer)
73     gas = Column(Integer)
74     movimento = Column(Integer)
75     status_ac = Column(Integer)
76     ultima_leitura = Column(DATETIME, nullable=False, default=datetime.now)
77
78     def __repr__(self):
79         return '<Sensores ID: {}>'.format(self.id)
80
81 class Comandos(db.Model):
82     __tablename__ = 'comandos'
83     id = Column(Integer, primary_key=True)
84     id_user = Column(Integer)
85     comando = Column(String(25))
86     data = Column(DATETIME, default=datetime.now)
87
88     def __init__(self, id_user, comando, data):
89         self.id_user = id_user
90         self.comando = comando
```

```

91         self.data = data
92
93     def __repr__(self):
94         return '<Comandos ID: {}>'.format(self.id)

```

Lista de Ilustrações B.4 – Arquivo com as definições das classes que representam as tabelas do banco de dados

B.5 forms.py

```

1
2 from flask_wtf import FlaskForm
3 from wtforms import StringField, PasswordField, SubmitField, SelectField,
   BooleanField
4 from wtforms.validators import InputRequired, Length, ValidationError,
   DataRequired, Email, EqualTo
5
6 import sqlalchemy as sa
7 from app import db
8 from app.models import User
9
10 class LoginForm(FlaskForm):
11     matricula = StringField(validators=[DataRequired(),
12                                       InputRequired(),
13                                       Length(min=14, max=14)],
14                             render_kw={"placeholder": "
15                                     Matricula"})
16
17     senha = PasswordField(validators=[DataRequired(),
18                                       InputRequired(),
19                                       Length(min=8, max=20)],
20                             render_kw={"placeholder": "Senha"
21                                     })
22
23     remember_me = BooleanField('Lembrar senha')
24
25     submit = SubmitField('Sign In')
26
27 class RegistrationForm(FlaskForm):
28     nome = StringField(validators=[DataRequired(),
29                                   InputRequired(),
30                                   Length(min=3, max=40)],
31                             render_kw={"placeholder": "Nome
32                                     Completo"})
33
34     matricula = StringField(validators=[DataRequired(),

```



```

33         InputRequired(),
34         Length(min=14, max=14)],
35         render_kw={"placeholder": "
36                     Matricula"})
37
38     senha = PasswordField(validators=[DataRequired(),
39                                 InputRequired(),
40                                 Length(min=8, max=20)],
41                             render_kw={"placeholder": "Senha"
42                                         })
43
44     confirme_senha = PasswordField(validators=[DataRequired(),
45                                                 InputRequired(),
46                                                 Length(min=8, max=20),
47                                                 EqualTo('senha')],
48                                     render_kw={"placeholder": "
49                                                 Confirme a senha"})
50
51     email = StringField(validators=[InputRequired(),
52                                    Length(min=10, max=40)],
53                            render_kw={"placeholder": "Email"})
54
55     curso = SelectField(u'Curso', choices=[('EE', 'Engenharia El trica'),
56                                           ('EC', 'Engenharia Civil'),
57                                           ('EA', 'Engenharia Agron mica'
58                                           ),
59                                           ('SI', 'Sistemas para Internet'
60                                           ),
61                                           ('FI', 'F sica'),
62                                           ('MA', 'Matem tica'),
63                                           ('LE', 'Letras'),
64                                           ('GP', 'Gest o P blica')],
65                           validators=[InputRequired()])
66
67     submit = SubmitField('Registrar')
68
69     def validate_matricula(self, matricula):
70         user = db.session.scalar(sa.select(User).where(
71             User.matricula == matricula.data))
72         if user is not None:
73             raise ValidationError('Matricula ja registrada.\nVerifique o
74                                     numero de matricula ou procure um administrador')
75
76     """
77
78     def validate_email(self, email):
79         user = db.session.scalar(sa.select(User).where(

```

```

74         User.email == email.data))
75     if user is not None:
76         raise ValidationError('Please use a different email address.')
77     """
78
79 class EditionForm(FlaskForm):
80     nome = StringField(validators=[DataRequired(),
81                                 InputRequired(),
82                                 Length(min=3, max=40)],
83                        render_kw={"placeholder": "Nome
84                                    Completo"})
85
86     matricula = StringField(validators=[DataRequired(),
87                                       InputRequired(),
88                                       Length(min=14, max=14)],
89                             render_kw={"placeholder": "
90                                         Matricula"})
91
92     senha = PasswordField(validators=[DataRequired(),
93                                       InputRequired(),
94                                       Length(min=8, max=20)],
95                             render_kw={"placeholder": "Senha
96                                         })
97
98     email = StringField(validators=[InputRequired(),
99                                   Length(min=10, max=40)],
100                          render_kw={"placeholder": "Email"})
101
102     curso = SelectField(u'Curso', choices=[('EE', 'Engenharia El trica'),
103                                           ('EC', 'Engenharia Civil'),
104                                           ('EA', 'Engenharia Agron mica'
105                                           ),
106                                           ('SI', 'Sistemas para Internet'
107                                           ),
108                                           ('FI', 'F sica'),
109                                           ('MA', 'Matem tica'),
110                                           ('LE', 'Letras'),
111                                           ('GP', 'Gest o P blica')],
112                          validators=[InputRequired()])
113
114     submit = SubmitField('Salvar edi es')

```

Lista de Ilustrações B.5 – Arquivo com as declarações dos formulários da aplicação

APÊNDICE C – CÓDIGOS COMPLEMENTARES DA APLICAÇÃO PYTHON

C.1 *requirements.txt*

```
1
2 asgiref==3.5.0
3 bcrypt==3.2.0
4 blinker==1.7.0
5 cffi==1.15.0
6 click==8.1.2
7 dnspython==2.2.1
8 Flask==2.2.2
9 Flask-Bcrypt==1.0.1
10 Flask-Login==0.6.0
11 Flask-MQTT==1.2.1
12 Flask-Session==0.4.0
13 Flask-SocketIO==5.3.6
14 Flask-SQLAlchemy==3.0.3
15 Flask-WTF==1.0.1
16 greenlet==1.1.2
17 gunicorn==21.2.0
18 idna==3.3
19 importlib-metadata==4.11.3
20 itsdangerous==2.1.2
21 Jinja2==3.1.1
22 mariadb==1.1.10
23 MarkupSafe==2.1.1
24 paho-mqtt==2.0.0
25 pycparser==2.21
26 simple-websocket==1.0.0
27 six==1.16.0
28 SQLAlchemy==1.4.35
29 sqlparse==0.4.2
30 Werkzeug==2.2.2
31 wheel==0.37.1
32 WTForms==3.0.1
33 zipp==3.8.0
```

Lista de Ilustrações C.1 – Lista de dependências para que a aplicação funcione corretamente

C.2 *mqtt.py*

```
1
2 import mariadb
3 from datetime import datetime
```

```

4
5 # Database connection details
6 DB_HOST = "localhost"
7 DB_NAME = "sql_db"
8 DB_USER = "augusto"
9 DB_PASSWORD = "1234"
10
11 # MQTT connection details
12 broker_address = "192.168.100.212"
13 broker_port = 1883
14
15 # Topics MQTT
16 #topic = '/esp32/verificarConexao'
17 #topic2 = '/esp32/enviarComando'
18 topicSensorDHT = "/sensores/DHT"; # subscribe
19 topicSensorLDR = "/sensores/LDR"; # subscribe
20 topicSensorSR602 = "/sensores/SR602"; # subscribe
21 topicSensorSCT = "/sensores/SCT"; # subscribe
22
23 ### Estabelece conexão com o BD
24 def connect_to_database():
25     """Connects to the MariaDB database and returns the connection object.
26     """
27     try:
28         conn = mariadb.connect(
29             host=DB_HOST,
30             user=DB_USER,
31             password=DB_PASSWORD,
32             database=DB_NAME,
33         )
34         return conn
35     except mariadb.Error as e:
36         print(f"Error connecting to database: {e}")
37         return None
38
39 ##### MANIPULA O DO BD #####
40 #####
41 def store_dht_in_database(umidade, temperatura):
42
43     conn = connect_to_database()
44
45     if conn:
46         cursor = conn.cursor()
47
48         sql_update = "UPDATE sensores SET umidade = ?, temperatura = ?,
49                     ultima_leitura = ? WHERE id = 1"

```

```
49     values = (umidade, temperatura, datetime.now())
50
51     try:
52         cursor.execute(sql_update, values)
53         conn.commit()
54         print("Data stored successfully!")
55     except mariadb.Error as e:
56         print(f"Error storing data: {e}")
57
58     conn.close()
59
60 def store_ldr_in_database(luminosidade):
61
62     conn = connect_to_database()
63
64     if conn:
65         cursor = conn.cursor()
66
67         sql_update = "UPDATE sensores SET luminosidade = ? WHERE id = 1"
68         values = (luminosidade)
69
70         try:
71             cursor.execute(sql_update, values)
72             conn.commit()
73             print("Data updated successfully!")
74         except mariadb.Error as e:
75             print(f"Error updating data: {e}")
76
77         conn.close()
78
79 def store_sr602_in_database(movimento):
80     conn = connect_to_database()
81
82     if conn:
83         cursor = conn.cursor()
84
85         sql_update = "UPDATE sensores SET movimento = ? WHERE id = 1"
86         values = (movimento)
87
88         try:
89             cursor.execute(sql_update, values)
90             conn.commit()
91             print("Data updated successfully!")
92         except mariadb.Error as e:
93             print(f"Error updating data: {e}")
94
95         conn.close()
```

```
96
97 def store_sct_in_database(status_ac):
98     conn = connect_to_database()
99
100     if conn:
101         cursor = conn.cursor()
102
103         sql_update = "UPDATE sensores SET status_ac = ? WHERE id = 1"
104
105         if status_ac[0] > 5:
106             status_ac[0] = 1
107         else:
108             status_ac[0] = 0
109
110         values = (status_ac)
111
112         try:
113             cursor.execute(sql_update, values)
114             conn.commit()
115             print("Data updated successfully!")
116         except mariadb.Error as e:
117             print(f"Error updating data: {e}")
118
119         conn.close()
120
121
122 #####
123 #####          CALLBACK DO MQTT          #####
124 #####
125 def on_message(client, userdata, msg):
126     topic = msg.topic
127     payload = msg.payload.decode()
128     print("Mensagem recebida no t pico: ", topic)
129
130     if topic == topicSensorDHT:
131         payload = eval(payload)
132         valores = []
133
134         for i in payload.values():
135             valores.append(i)
136
137         store_dht_in_database(valores[0], valores[1])
138
139     elif topic == topicSensorLDR:
140         # Converte a string recebida em um dicionario paitu
141         payload = eval(payload)
142         valores = []
```

```
143
144     # Anexa os valores enviados em uma lista
145     for i in payload.values():
146         valores.append(i)
147
148     store_ldr_in_database(valores)
149
150 elif topic == topicSensorSR602:
151     # Converte a string recebida em um dicionario paitu
152     payload = eval(payload)
153     valores = []
154
155     # Anexa os valores enviados em uma lista
156     for i in payload.values():
157         valores.append(i)
158
159     store_sr602_in_database(valores)
160
161 elif topic == topicSensorSCT:
162     payload = eval(payload)
163     valores = []
164
165     for i in payload.values():
166         valores.append(i)
167
168     store_sct_in_database(valores)
169
170 def on_connect(client, userdata, flags, reason_code, properties):
171     if reason_code == 0:
172         print("Conectado ao broker MQTT")
173         client.subscribe(topicSensorDHT)
174         client.subscribe(topicSensorLDR)
175         client.subscribe(topicSensorSR602)
176         client.subscribe(topicSensorSCT)
177
178         print("Inscrito nos t picos: ")
179         print(topicSensorDHT + '\n' + topicSensorLDR + '\n' + topicSensorSR602 +
180               '\n' + topicSensorSCT + '\n')
181     else:
182         print("Conex o falhou, reason_code: " + str(reason_code))
183
184 def on_publish(client, userdata, mid, reason_code, properties):
185     print("Dados publicados \n")
186     pass
```

Lista de Ilustrações C.2 – Arquivo com as definições para comunicação MQTT

APÊNDICE D – TEMPLATES HTML

D.1 base.html

```

1
2 <!doctype html>
3 <html>
4   <head>
5     {% if title %}
6     <title>{{ title }} - Holhooja</title>
7     {% else %}
8     <title>SISTEMA HOLHOOJA</title>
9     {% endif %}
10    <link
11    href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/
12      bootstrap.min.css"
13    rel="stylesheet"
14    integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/
15      DwWykc2MPK8M2HN"
16    crossorigin="anonymous">
17
18    <meta charset="utf-8">
19    <meta name="viewport" content="width=device-width, initial-scale=1"
20      >
21    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@1
22      .0.0/css/bulma.min.css">
23
24    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/
25      jquery.min.js" integrity="sha512-
26      bLT0Qm9VnAYZDflyKcBaQ2gg0hSYNQrJ8RilYldYQ1FxQYoCLtUjuuRuZo+fjqhx
27      /qtq/1itJ0C2ejDxltZVFg==" crossorigin="anonymous"></script>
28
29    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io
30      /3.0.4/socket.io.js" integrity="sha512-aMGMvNYu8Ue4G+
31      fHa359jcPb1u+ytAF+P2SCb+
32      PxrjCd03n3ZTxJ30zuH39rimUggmTwmh2u7wvQsDTHESnmfQ==" crossorigin=
33      "anonymous"></script>
34
35    <script type="text/javascript" charset="utf-8">
36      $(document).ready(function() {
37        // Connect to the Socket.IO server.
38        // The connection URL has the following format, relative to
39        // the current page:
40        // http[s]://<domain>:<port>[/<namespace>]
41        var socket = io();

```



```
30
31 // Event handler for new connections.
32 // The callback function is invoked when a connection with
33 // the
34 // server is established.
35 socket.on('connect', function() {
36     socket.emit('my_event', {data: 'I\'m connected!'});
37 });
38
39 // Event handler for server sent data.
40 // The callback function is invoked whenever the server emits
41 // data
42 // to the client. The data is then displayed in the "Received
43 // "
44 // section of the page.
45 socket.on('my_response', function(msg, cb) {
46
47     $('#temperatura').text($('<div/>').text( msg.temperatura +
48         ' C ').html());
49     $('#umidade').text($('<div/>').text( msg.umidade + '%').
50         html());
51     $('#movimento').text($('<div/>').text( msg.movimento).html
52         ());
53     $('#luminosidade').text($('<div/>').text( msg.luminosidade)
54         .html());
55
56     // Informa o status do ar condicionado baseado no valor do
57     // sensor
58     const status = msg.movimento === 1 ? 'Ligado' : 'Desligado
59         ';
60     $('#status_ar').text(status);
61
62     if (cb)
63         cb();
64
65 });
66
67 // Interval function that tests message latency by sending a
68 // "ping"
69 // message. The server then responds with a "pong" message
70 // and the
71 // round trip time is measured.
72 var ping_pong_times = [];
73 var start_time;
74 window.setInterval(function() {
75     start_time = (new Date).getTime();
76     $('#transport').text(socket.io.engine.transport.name);
```

```
66         socket.emit('my_ping');
67     }, 1000);
68
69     // Handler for the "pong" message. When the pong is received,
70     the
71     // time from the ping is stored, and the average of the last
72     30
73     // samples is average and displayed.
74     socket.on('my_pong', function() {
75         var latency = (new Date).getTime() - start_time;
76         ping_pong_times.push(latency);
77         ping_pong_times = ping_pong_times.slice(-30); // keep
78         last 30 samples
79         var sum = 0;
80         for (var i = 0; i < ping_pong_times.length; i++)
81             sum += ping_pong_times[i];
82         $('#ping-pong').text(Math.round(10 * sum /
83             ping_pong_times.length) / 10);
84     });
85
86     // Handlers for the different forms in the page.
87     // These accept data from the user and send it to the server
88     in a
89     // variety of ways
90     $('form#emit').submit(function(event) {
91         socket.emit('my_event', {data: $('#emit_data').val()});
92         return false;
93     });
94     $('form#broadcast').submit(function(event) {
95         socket.emit('my_broadcast_event', {data: $('#
96             broadcast_data').val()});
97         return false;
98     });
99     $('form#join').submit(function(event) {
100         socket.emit('join', {room: $('#join_room').val()});
101         return false;
102     });
103     $('form#leave').submit(function(event) {
104         socket.emit('leave', {room: $('#leave_room').val()});
105         return false;
106     });
107     $('form#send_room').submit(function(event) {
108         socket.emit('my_room_event', {room: $('#room_name').val()
109             , data: $('#room_data').val()});
110         return false;
111     });
112     $('form#close').submit(function(event) {
```

```

106         socket.emit('close_room', {room: $('#close_room').val()})
107         ;
108         return false;
109     });
110     $('form#disconnect').submit(function(event) {
111         socket.emit('disconnect_request');
112         return false;
113     });
114
115     });
116     </script>
117
118 </head>
119 <body class="has-background-grey-darker">
120
121     <nav class="navbar navbar-dark has-background-link-dark">
122         <div class="container">
123             <b><a class="navbar-brand" href="{{ url_for('index') }}">Holhooja
124                 </a></b>
125
126             <button class="navbar-toggler" type="button" data-bs-toggle="
127                 collapse" data-bs-target="#navbarSupportedContent" aria-
128                 controls="navbarSupportedContent" aria-expanded="false" aria-
129                 label="Toggle navigation">
130                 <span class="navbar-toggler-icon"></span>
131             </button>
132
133             <div class="collapse navbar-collapse" id="navbarSupportedContent"
134                 >
135                 <ul class="navbar-nav me-auto mb-2 mb-lg-0">
136                     <li class="nav-item">
137                         <a class="nav-link" aria-current="page" href="{{ url_for('
138                             login') }}">Login</a>
139                     </li>
140                     <li class="nav-item">
141                         <a class="nav-link" aria-current="page" href="{{ url_for('
142                             user', matricula=current_user.matricula) }}">Profile</a>
143                     </li>
144                     <li class="nav-item">
145                         <a class="nav-link" aria-current="page" href="{{ url_for('
146                             logout', id=current_user.id) }}">Logout</a>
147                     </li>
148                     <li class="nav-item">
149                     </li>
150                 </ul>
151             </div>

```

```

144     </div>
145 </nav>
146
147 <div class="container is-fullhd ">
148     {% with messages = get_flashed_messages() %}
149     {% if messages %}
150         {% for message in messages %}
151             <div class="alert alert-info" role="alert">{{ message }}</div>
152         {% endfor %}
153     {% endif %}
154     {% endwith %}
155     <br>
156     {% block content %}{% endblock %}
157     <br>
158     <!--<p>Async mode is: <b>{{ async_mode }}</b></p>
159     <p>Average ping/pong latency: <b><span id="ping-pong"></span>ms</b>
160         </p>-->
161     <br>
162 </div>
163
164 <script defer src="https://use.fontawesome.com/releases/v5.0.7/js/all
165     .js"></script>
166 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/
167     bootstrap.bundle.min.js" integrity="sha384-
168     YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz "
169     crossorigin="anonymous"></script>
170 </body>
171 </html>

```

Lista de Ilustrações D.1 – Arquivo base para manipulação do *template* utilizando Jinja

D.2 dash_lab.html

```

1
2 {% extends "base.html" %}
3
4 {% block content %}
5     <h1 class="title">Painel de controle&nbsp;&nbsp;&nbsp;<i class="fas fa-arrow-
6         right"></i> &nbsp;&nbsp;&nbsp;Laborat rio</h1>
7
8     <div class="columns ">
9         <!-- CONTROLE DO AR CONDICIONADO -->
10         <div class="column is-half has-text-centered">
11             <article class="message is-info">
12                 <div class="message-header">
13                     <h4 class="title is-4 has-text-info-dark"> Ar-
14                         condicionado </h4>

```

```

13     </div>
14     <div class="message-body has-text-info">
15         <b><p>Status</p></b>
16         <div>
17             {% if sensores.status_ac == 1 %}
18                 Ligado
19             {% elif sensores.status_ac == 0 %}
20                 Desligado
21             {% endif %}
22         </div> <br>
23
24         <!-- BOT O ON/OFF DO AR CONDICIONADO -->
25         <p class="buttons is-centered">
26             <a href="/mqtt/on">
27                 <button id="onButton" class="button is-success"
28                     >
29                     <span class="icon is-small">
30                         <i class="fas fa-power-off"></i>
31                     </span>
32                 </button>
33             </a>
34             <a href="/mqtt/off">
35                 <button id="offButton" class="button is-danger"
36                     >
37                     <span class="icon is-small">
38                         <i class="fas fa-power-off"></i>
39                     </span>
40                 </button>
41             </a>
42         </p>
43         <br>
44         <!--CONTROLE DE TEMPERATURA DO AR CONDICIONADO-->
45         <p class="buttons is-centered">
46             <a href="/mqtt/up">
47                 <button class="button is-info">
48                     &nbsp;<span class="icon is-small">
49                         <i class="fas fa-chevron-up"></i>
50                     </span>&nbsp;
51                 </button>
52             </a>
53         </p>
54
55         <p class="buttons is-centered">
56             <a href="/mqtt/left">
57                 <button class="button is-info">
58                     &nbsp;<span class="icon is-small">

```

[illegible]

```

101         <i class="fas fa-power-off"></i>
102     </span>
103 </button>
104 </a>
105 <a href="/mqtt/off">
106     <button id="offButton" class="button is-danger"
107         >
108         <span class="icon is-small">
109             <i class="fas fa-power-off"></i>
110         </span>
111     </button>
112 </a>
113 </p>
114 <br>
115 </div>
116 </article>
117 </div>
118 {% endblock %}

```

Lista de Ilustrações D.2 – Estrutura do painel de controle do laboratório

D.3 editar_senha.html

[illegible]

```

20     <br>
21
22     <button class="button is-primary is-dark" type="submit">
23         <span class="icon is-small">
24             <i class="far fa-save"></i>
25         </span> &nbsp;
26         Salvar
27     </button>
28 </form>
29
30 <br>
31 <p class="buttons" style="display: flex; justify-content: flex-end; ">
32     <a href="/usuarios">
33         <button class="button is-info ">
34             <span class="icon is-small">
35                 <i class="far fa-arrow-alt-circle-left"></i>
36             </span>&nbsp;
37             Voltar
38         </button>
39     </a>
40 </p>
41 {% endblock %}

```

Lista de Ilustrações D.3 – Formulário para edição de senha do usuário no painel de controle dos administradores

D.4 editar_usuario.html

```
1 {% extends "base.html" %}
2 {% import 'bootstrap_wtf.html' as wtf %}
3 
4 
5 {% block content %}
6     <h1 class="title">Editar dados de: {{ user.nome }}</h1><br>
7 
8     <form action="/usuarios/atualizar/{{ user.id }}" method="POST">
9         <label>Nome completo</label>
10        <div class="input-group mb-3">
11            <input type="text" class="form-control rounded" name="ed_nome"
12                id="editar_nome" value="{{ user.nome }}" require >&nbsp;&nbsp;&nbsp;&
13                &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&~
14            </div>
15 
16        <label>Matricula</label>
17        <div class="input-group mb-3">
18            <input type="text" class="form-control rounded" name="
19                ed_matricula" id="editar_matricula" value="{{ user.matricula
```


[illegible]

```

60     {% elif user.curso == "LE" %}
61         <p>Curso atual: <b>Letras</p></b>
62     {% elif user.curso == "GP" %}
63         <p>Curso atual: <b>Gest o P blica</p></b>
64     {% elif user.curso == "cu" %}
65         <p>Curso atual: <b>Caraio de curso</p></b>
66     {% endif %}
67
68     <br>
69     <label for="ed_curso">Alterar curso:</label>
70     <select id="editar_curso" name="ed_curso" >
71         <option value="--" selected>--</option>
72         <option value="EE">Engenharia El trica</option>
73         <option value="EC">Engenharia Civil</option>
74         <option value="EA">Engenharia Agron mica</option>
75         <option value="SI">Sistemas para Internet</option>
76         <option value="FI">F sica</option>
77         <option value="MA">Matem tica</option>
78         <option value="LE">Letras</option>
79         <option value="GP">Gest o P blica</option>
80     </select>
81 </div>
82 <br>
83 <br>
84
85     <button class="button is-primary is-dark" type="submit" >
86         <span class="icon is-small">
87             <i class="far fa-save"></i>
88         </span> &nbsp;
89         Salvar
90     </button>
91 </form>
92 <br>
93 <p class="buttons">
94
95
96     {% if user.tipo == 1 %} <!-- USUARIOS ADMIN PODEM SE TORNAR DEFAULT
97         -->
98         <a href="/usuarios/mudar_tipo/{{user.id}}">
99             <button class="button is-primary ">
100                 <span class="icon is-small">
101                     <i class="far fa-user"></i>
102                 </span>&nbsp;
103                 Tornar Comum
104             </button>
105         </a>

```

```

105     {% elif user.tipo == 0 %}      <!-- USUARIOS DEFAULT PODEM SE TORNAR
        ADMIN -->
106     <a href="/usuarios/mudar_tipo/{{user.id}}">
107         <button class="button is-primary ">
108             <span class="icon is-small">
109                 <i class="fas fa-user-secret"></i>
110             </span>&nbsp;
111             Tornar Admin
112         </button>
113     </a>
114 {% endif %}
115
116 <a href="/usuarios/editar_senha/{{ user.id }}">
117     <button class="button is-primary ">
118         <span class="icon is-small">
119             <i class="fas fa-key"></i>
120         </span>&nbsp;
121         Mudar Senha
122     </button>
123 </a>
124
125 {% if user.status == 1 %} <!-- USUARIOS ATIVOS PODEM SER
        DESATIVADOS -->
126     <a href="/usuarios/mudar_status/{{user.id}}">
127         <button class="button is-primary ">
128             <span class="icon is-small">
129                 <i class="far fa-times-circle"></i>
130             </span>&nbsp;
131             Inativar
132         </button>
133     </a>
134 {% elif user.status == -1 %} <!-- USUARIOS INATIVOS PODEM SER
        REATIVADOS -->
135     <a href="/usuarios/mudar_status/{{user.id}}">
136         <button class="button is-primary ">
137             <span class="icon is-small">
138                 <i class="far fa-check-circle"></i>
139             </span>&nbsp;
140             Ativar
141         </button>
142     </a>
143 {% elif user.status == 0 %} <!-- USUARIOS QUE AGUARDAM ANALISE -->
144     <a href="/aprovar_status/{{user.matricula}}">
145         <button class="button is-success">
146             <span class="icon is-small">
147                 <i class="fas fa-check"></i>
148             </span>

```

```

149         </button>
150     </a>
151     <a href="/reprovar_status/{{user.matricula}}">
152         <button class="button is-danger">
153             <span class="icon is-small">
154                 <i class="fas fa-times"></i>
155             </span>
156         </button>
157     </a>
158 {% endif %}
159
160
161
162 <p class="buttons" style="display: flex; justify-content: flex-end; ">
163     <a href="/usuarios">
164         <button class="button is-info ">
165             <span class="icon is-small">
166                 <i class="far fa-arrow-alt-circle-left"></i>
167             </span>&nbsp;Voltar
168         </button>
169     </a>
170 </p>
171 </p>
172 </p>
173 {% endblock %}

```

Lista de Ilustrações D.4 – Estrutura do formulário de edição de dados dos usuário no painel de controle dos administradores

D.5 index_admin.html

```

1
2 {% extends "base.html" %}
3
4 {% block content %}
5     <h1 class="title">Hi, {{ current_user.nome }}!</h1>
6
7     <!-- dados do lab -->
8     <div class="columns">
9         <div class="column">
10             <article class="message is-info">
11                 <div class="message-header">
12                     <h4 class="title is-4 has-text-link-dark">Dados do
13                         laborat rio</h4>
14                 </div>
15                 <div class="message-body">

```

```

16         <div class="columns">
17             <div class="column is-one-quarter">
18                 <article class="message">
19                     <div class="message-header">
20                         <h4 class="title is-4 has-text-info"><i
21                             class="fas fa-thermometer-half"></i>
22                             >&nbsp;Temperatura</h4>
23                     </div>
24                     <div class="message-body">
25                         <b> <div id="temperatura"></div> </b>
26                     </div>
27                 </article>
28             </div>
29
30             <div class="column is-one-quarter">
31                 <article class="message ">
32                     <div class="message-header">
33                         <h4 class="title is-4 has-text-info"><i
34                             class="fas fa-tint"></i>&nbsp;
35                             Umidade</h4>
36                     </div>
37                     <div class="message-body">
38                         <b> <div id="umidade"></div></b>
39                     </div>
40                 </article>
41             </div>
42
43             <div class="column is-one-quarter">
44                 <article class="message">
45                     <div class="message-header">
46                         <h4 class="title is-4 has-text-info"><i
47                             class="fas fa-wifi"></i>&nbsp;
48                             Movimenta o</h4>
49                     </div>
50                     <div class="message-body">
51                         <b> <div id="movimento"></div> </b>
52                     </div>
53                 </article>
54             </div>
55
56             <div class="column is-one-quarter">
57                 <article class="message">
58                     <div class="message-header">
59                         <h4 class="title is-4 has-text-info"><i
60                             class="fas fa-sun"></i>&nbsp;
61                             Luminosidade</h4>

```

```

55         </div>
56         <div class="message-body">
57             <b> <div id="luminosidade"></div> </b>
58         </div>
59     </article>
60 </div>
61 </div>
62 </div>
63 </article>
64 </div>
65 </div>
66
67
68 <!-- PAINEL DE CONTROLE -->
69 <div class="columns">
70     <div class="column">
71         <article class="message is-info">
72             <div class="message-header">
73                 <h4 class="title is-4 has-text-link-dark">Painel de
74                     controle</h4>
75             </div>
76             <div class="message-body">
77                 <div class="buttons is-centered">
78                     <a href="{ { url_for('dash_lab') } }"><button class="
79                         button is-primary is-light is-rounded"><i class="
80                             fas fa-bolt"></i>&nbsp;&nbsp;&nbsp;Laborat rio</
81                         button></a>
82                     <a href="{ { url_for('usuarios') } }"><button class="
83                         button is-primary is-light is-rounded"><i class="
84                             fas fa-users"></i>&nbsp;&nbsp;&nbsp;Usu rios</button
85                         ></a>
86                 </div>
87             </div>
88         </article>
89     </div>
90 </div>
91
92 <!-- SOLICITA ES DE ACESSO -->
93 <div class="columns">
94     <div class="column">
95         <article class="message is-info">
96             <div class="message-header">
97                 <h4 class="title is-4 has-text-link-dark">Solicita es de
98                     acesso</h4>
99             </div>
100             <div class="message-body">

```

[illegible]

```

118                                     check"><
119                                     /i>
120                                     </span>
121                                     </button>
122                                     </a>
123                                     <a href="/
124                                     reprovar_status
125                                     /{{notif.
126                                     matricula}}">
127                                     <button class="
128                                     button is-
129                                     danger">
130                                     <span class
131                                     ="icon
132                                     is-small
133                                     ">
134                                     <i class="
135                                     fas fa-
136                                     times"><
137                                     /i>
138                                     </span>
139                                     </button>
140                                     </a>
141                                     </p>
142                                     </div>
143                                     </article>
144                                     </div>
145                                     {% endfor %}
146                                     </div>
147                                     </div>
148                                     </div>
149                                     </div>
150                                     </article>
151                                     </div>
152                                     </div>
153                                     <!-- LISTA DE PRESENTES -->
154                                     <div class="columns">
155                                     <div class="column">
156                                     <article class="message is-info">
157                                     <div class="message-header">
158                                     <h4 class="title is-4 has-text-link-dark">Lista de
159                                     Presentes</h4>
160                                     </div>
161                                     <div class="message-body">
162                                     <div class="buttons is-centered">

```


[illegible]

Lista de Ilustrações D.5 – Estrutura da página inicial do usuário administrador

D.6 index.html

```
1
2 {% extends "base.html" %}
3
```

```

4 {% block content %}
5     <h1>Hi, {{ current_user.matricula }}!</h1>
6     {% for post in posts %}
7     <div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>
8     {% endfor %}
9 {% endblock %}

```

Lista de Ilustrações D.6 – Estrutura da página inicial do usuário comum

D.7 login.html

```

1
2 {% extends "base.html" %}
3 {% import 'bootstrap_wtf.html' as wtf %}
4
5 {% block content %}
6     <h1 class="title is-4">Login</h1>
7     {{ wtf.quick_form(form) }}
8
9     <br>
10    <p>Novo por aqui? <a href="{{ url_for('register') }}">Clique para
        solicitar seu acesso!</a></p>
11 {% endblock %}

```

Lista de Ilustrações D.7 – Estrutura para chamada do formulário de *login*.

D.8 register.html

```

1
2 {% extends "base.html" %}
3 {% import 'bootstrap_wtf.html' as wtf %}
4
5 {% block content %}
6     <h3><b>Solicita o de acesso</b></h3>
7     <br>
8
9     {{ wtf.quick_form(form) }}
10
11 {% endblock %}

```

Lista de Ilustrações D.8 – Estrutura para chamada do formulário de solicitação de acesso.

D.9 user.html

```

1
2 {% extends "base.html" %}
3
4 {% block content %}
5     <h1>User: {{ user.nome }}</h1>
6     <hr>
7     {% for post in posts %}
8     <p>
9         {{ post.author.nome }} says: <b>{{ post.body }}</b>
10    </p>
11    {% endfor %}
12 {% endblock %}

```

Lista de Ilustrações D.9 – *Template* do perfil do usuário.

D.10 usuários.html

```

1
2 {% extends "base.html" %}
3
4 {% block content %}
5 <h1 class="title">Painel de controle&nbsp;&nbsp;&nbsp;<i class="fas fa-arrow-
6   right"></i> &nbsp;&nbsp;&nbsp;Usu rios</h1>
7
8 <div class="columns is-multiline">
9     {% for user in users %}
10         <div class="column is-one-quarter">
11             <article class="message is-info">
12                 <div class="message-header">
13                     <h4 class="title is-4 has-text-info-dark">{{user.id
14                       }} - {{user.nome}} </h4>
15                 </div>
16                 <div class="message-body has-text-info">
17                     Matricula: {{user.matricula}} <br>
18                     Curso: {{user.curso}} <br>
19                     Email: {{user.email}} <br>
20                     Status:
21                         {% if user.status == 1 %}
22                             Ativo
23                         {% elif user.status == -1 %}
24                             Inativo
25                         {% elif user.status == 0 %}
26                             Aguardando an lise
27                         {% endif %} <br>
28                     Tipo:
29                         {% if user.tipo == 1 %}

```

