UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JEAN FÜCHTER

DESENVOLVIMENTO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM ACESSO VIA APLICATIVO ANDROID E GERENCIADOR DE TAREFAS COM ATIVAÇÃO AUTOMÁTICA

PATO BRANCO

2022

JEAN FÜCHTER

DESENVOLVIMENTO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM ACESSO VIA APLICATIVO ANDROID E GERENCIADOR DE TAREFAS COM ATIVAÇÃO AUTOMÁTICA

Development of a home automation system with access via android application and task manager with automatic activation

Trabalho de conclusão de curso de graduação apresentada como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Dr. Ricardo Bernardi

PATO BRANCO

2022

JEAN FÜCHTER

DESENVOLVIMENTO DE UM SISTEMA DE AUTOMAÇÃO RESIDENCIAL COM ACESSO VIA APLICATIVO ANDROID E GERENCIADOR DE TAREFAS COM ATIVAÇÃO AUTOMÁTICA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 07 de dezembro de 2022

_____

César Rafael Claure Torrico
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Ricardo Bernardi
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Robison Cris Brito
Doutorado
Universidade Tecnológica Federal do Paraná

PATO BRANCO

2022

RESUMO

A automação surgiu na indústria com o intuito de auxiliar na realização de tarefas diárias, seus notórios benefícios juntamente com a evolução da tecnologia rapidamente deram origem a novas subáreas, uma delas é a automação residencial. A automação residencial visa gerir os recursos habitacionais como a climatização, iluminação, persianas, irrigação dentre outras inúmeras tarefas que podem ser automatizadas em uma moradia, porém, a gestão desses recursos de forma independente, em plataformas separadas, torna sua operação difícil e trabalhosa. Diante disso surge a Domótica, que é um segmento tecnológico com o intuito de unificar a gestão de todos os recursos habitacionais em um único sistema, de forma integrada. Este trabalho apresenta o desenvolvimento de um sistema de automação residencial baseado no conceito Domótica, o sistema consiste em um servidor *web* instalado em uma *Raspberry Pi* que proporciona uma interface de controle via aplicativo *Android* para que o usuário possa gerenciar todos os recursos disponíveis na residência. Os comandos da interface podem ser executados instantaneamente pelo usuário ou agendados em tarefas para serem executadas posteriormente de forma automática pelo sistema. Os comandos enviados pelo servidor são executados pelos dispositivos ESP8266 conectados aos atuadores, que também realizam a aquisição de leituras de sensores e as enviam ao servidor, a comunicação entre o servidor e o ESP8266 é via rede *Wi-Fi,* dispensando o uso de fios para instalação. A interface de usuário pode ser acessada remotamente de qualquer lugar via aplicativo *Android* ou também em qualquer dispositivo que tenha navegador *web* e acesso à internet. Por fim, é demonstrado que o uso desse sistema pode trazer o aumento de conforto, praticidade, descanso e principalmente a otimização de tempo dos usuários, visto que inúmeras tarefas diárias da residência podem ser executadas pelo sistema de forma automática.

Palavras-chave: Domótica. Automação residencial. *Wi-Fi*. Servidor *web*. Microcontrolador.

ABSTRACT

Automation emerged in the industry in order to assist in carrying out daily tasks, its notorious benefits along with the evolution of technology quickly gave rise to new subareas, one of which is home automation. Home automation aims to manage housing resources such as air conditioning, lighting, blinds, irrigation, among many other tasks that can be automated in a home, however, managing these resources independently, on separate platforms, makes its operation difficult and laborious. In view of this, Domotics emerges, which is a technological segment with the aim of unifying the management of all housing resources in a single system, in an integrated manner. This work presents the development of a home automation system based on the Home Automation concept, the system consists of a web server installed on a Raspberry Pi that provides a control interface via an Android application so that the user can manage all the resources available in the home. Interface commands can be executed instantly by the user or scheduled in tasks to be executed later automatically by the system. The commands sent by the server are executed by the ESP8266 devices connected to the actuators, which also acquire sensor readings and send them to the server, the communication between the server and the ESP8266 is via Wi-Fi network, eliminating the use of wires to installation. The user interface can be accessed remotely from anywhere via the Android app or any device that has a web browser and internet access. Finally, it is demonstrated that the use of this system can increase comfort, practicality, rest and, above all, the optimization of users' time, since numerous daily tasks at home can be performed automatically by the system.

Keywords: Domotics. Home automation. Wi-Fi. Web server. Microcontroller.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico/Digital
GPS	*Global Positioning System* (Sistema de Posicionamento Global)
CI	Circuito Integrado
CISC	*Complex Instruction Set Computers* (Computador com um Conjunto Complexo de Instruções)
HTML	*Hypertext Markup Language* (Linguagem de Marcação de Hipertexto)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
IP	*Internet Protocol* (Protocolo de rede)
LAN	*Local Area Network* (Rede de Área Local)
MAN	*Metropolitan Area Network* (Rede de Área Metropolitana)
MQTT	*Message Queue Telemetry Transport* (Transporte de telemetria da fila de mensagens)
RISC	*Reduced Instruction Set Computers* (Computador com um Conjunto Reduzido de Instruções)
SO	Sistema Operacional
URL	*Uniform Resource Locator* (Localizador Uniforme de Recursos)
WAN	*Wide Area Network* (Rede de Área Ampla)

SUMÁRIO

1 INTRODUÇÃO

1.1 Automação residencial e Domótica

A automação apareceu na indústria após o surgimento da eletrônica, em meados da década de 50, e veio com o intuito de auxiliar o homem na realização de suas tarefas diárias, através da integração de dispositivos eletroeletrônicos e eletromecânicos. Seus notórios benefícios ao longo dos anos fizeram a automação crescer rapidamente originando as suas subáreas, sendo uma delas a Automação Residencial.

Segundo Bolzani (2006), a Automação Residencial tem demonstrado benefícios consideráveis, ajudando na redução do consumo de recursos como água e energia elétrica, agregando também conforto e segurança às residências. No geral, os sistemas de automação operam de maneira isolada, tornando custoso o gerenciamento de todos os serviços ao mesmo tempo. Devido a necessidade de integração destes sistemas, capaz de controlar tudo de maneira centralizada é que surge a Domótica.

José Pinheiro (2015) define Domótica como um segmento tecnológico que surgiu com o intuito de unir e gerir todos os recursos habitacionais de uma residência através de comandos remotos, pela internet ou mesmo por aplicativos de celular. Tem como objetivos substituir o ser humano nas atividades de rotina, controlar sistemas de segurança e variáveis ambientais, tudo em propósito de aumentar o grau de conforto e segurança, gerando uma melhora na qualidade de vida de seus ocupantes.

A Domótica pode ser implantada em uma residência para gerenciar sistemas de climatização, iluminação, sonorização, também pode executar tarefas diárias como irrigação do jardim, horta e alimentar os animais. Ainda, é capaz de auxiliar na segurança do imóvel através de informações em tempo real do que está acontecendo no ambiente. A Domótica se propõe a controlar os mais diversos periféricos presentes numa moradia de forma centralizada e automática sem a necessidade de intervenção humana. A Figura 1 ilustra os inúmeros recursos habitacionais e dispositivos que podem ser controlados de forma unificada utilizando o conceito da Domótica.

Figura 1 – Recursos e dispositivos que podem ser controlados utilizando a Domótica

Fonte: Pinheiro (2015).

A comunicação por rádio frequência ou tecnologia *wireless*, como é popularmente conhecida, tem auxiliado no crescimento da Domótica, pois, está presente na maioria dos dispositivos eletrônicos, eliminando o uso de fios para a troca de dados entre os mesmos, dando-lhes "liberdade" de estar em qualquer local onde haja sinal. O *smartphone*, por exemplo, usa a rede *wireless* para acessar a internet. Tecnologias atuais com valor relativamente baixo viabilizam a integração de dispositivos sem fio para que possam comunicar-se remotamente.

Neste contexto de benefícios proporcionados pela Domótica, aliado ao fácil acesso às tecnologias, se insere a proposta deste trabalho, que consiste no desenvolvimento de um sistema de automação dos periféricos de uma residência. Esse controle pode ser realizado remotamente através de qualquer dispositivo que possa acessar uma página *web* através da internet.

1.2 Objetivos

1.2.1 Objetivo geral

Construir uma automação de periféricos residenciais através do gerenciamento de sensores e atuadores a partir da integração com uma placa de aquisição e controle, banco de dados e servidor *web*, que permita controlar os recursos habitacionais de forma unificada e remota a partir de qualquer dispositivo conectado à internet ou automaticamente por meio de um perfil definido pelo usuário.

1.2.2 Objetivos específicos

- Apresentar a utilização de uma placa de aquisição e controle para gerenciar sensores e atuadores através de comandos recebidos do servidor *web*.
- Desenvolver um aplicativo *Android* para *smartphone* que permita acessar a página *web* e enviar comandos para o servidor.
- Criar um servidor *web* para gerenciar o sistema de controle e possibilitar a criação de tarefas periódicas em um perfil de usuário.
- Criar um sistema capaz de executar automaticamente as tarefas baseado na localização do usuário.

1.3 Justificativa

O ambiente residencial está, em geral, relacionado à ideia de descanso, conforto, local para recuperar as energias após um dia de trabalho e encontro de familiares. Isto está ligado a fatores temporais da residência, por exemplo, temperatura e luminosidade, ou ainda, tarefas que precisam ser executadas diariamente como, irrigar o jardim e a horta, alimentar os animais. Através da criação de um perfil é possível realizar tais atividades e ajustes, de maneira que possam ser ativadas automaticamente quando não há usuários na residência.

Deixar a mesma luz acessa durante toda noite ou por vários dias quando se está viajando já não é mais sinônimo de segurança, pelo contrário, acaba evidenciando que não há movimentação na residência durante este período. É interessante acender lâmpadas de cômodos diferentes em horários variados para simular a presença de pessoas na moradia e ainda ajudar a reduzir o consumo de energia elétrica. Tecnologias atuais permitem esse tipo de controle.

Na proposta deste trabalho, a ativação do controle manual ou automático da residência deve ocorrer conforme a rede em que o *smartphone* está conectado, ou seja, rede local (LAN) ou rede remota (WAN). Dessa forma, quando o morador estiver ausente o sistema deve operar de maneira automática, executando as tarefas programadas no perfil. Contudo, na presença do morador ele deve permanecer em modo manual.

1.4 Organização do texto

O texto a seguir está organizado em capítulos. O capítulo 2 apresenta o referencial teórico que explana sobre microcontrolador, sensores e atuadores, plataforma *Android* e redes de computadores, ao final deste capítulo são apresentados os trabalhos relacionados ao

proposto. No capítulo 3 é apresentado os materiais e métodos utilizados para desenvolvimento do projeto. O capítulo 4 apresenta os resultados obtidos e suas discussões, e por fim, o capítulo 5 apresenta as conclusões obtidas com o presente trabalho.

2 REFERENCIAL TEÓRICO

No presente capítulo serão apresentados conceitos que oferecem suporte ao desenvolvimento deste projeto. Estes conceitos estão relacionados com microcontrolador, sensores, atuadores, plataforma *Android* e redes de computadores. Por fim, serão apresentados trabalhos relacionados ao proposto. Caso o leitor já tenha conhecimento sobre os assuntos acima citados pode avançar para o próximo capítulo, Materiais e Métodos.

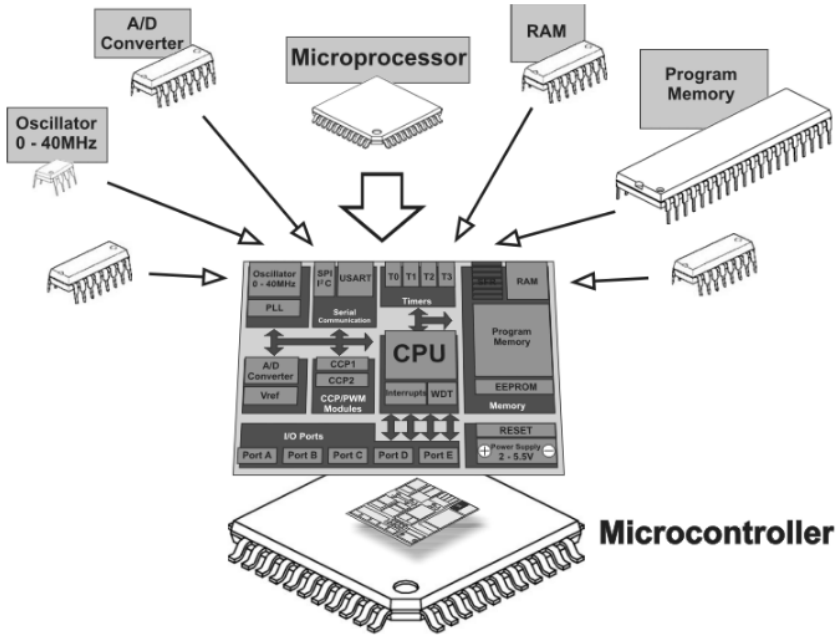
2.1 Microcontrolador

A placa de aquisição e controle é um dispositivo eletrônico que normalmente é composta por vários periféricos, como sensores, *leds*, osciladores, memória *flash*, pinos para conexões externas, entre outros. Mas, seu principal componente é um circuito integrado (CI) chamado de microcontrolador, responsável por gerir todos os recursos da placa. Estas placas normalmente são chamadas de kit de desenvolvimento, pois possibilitam ao usuário programar o microcontrolador a partir do seu próprio computador, através de um cabo conectado entre eles, e posteriormente testá-lo com outros circuitos.

Microcontroladores são computadores com tamanho e capacidade de processamento reduzidos. Diferentemente de um microprocessador que necessita de periféricos externos para seu funcionamento, o microcontrolador tem todos os componentes necessários para operar em um único chip (SILVA JÚNIOR, 1988). Desenvolvido para tarefas específicas, eles são capazes gerenciar diversos *hardwares* conectados a ele, como por exemplo sensores e atuadores.

O Microcontrolador tem seu *hardware* composto basicamente por um microprocessador, memórias, conversor Analógico/Digital (A/D), circuito oscilador e portas de entrada e saída, conforme pode ser visto na Figura 2 (PEREIRA, 2016). Normalmente, as linguagens comumente utilizadas para programação de microcontroladores são C e *Assembly.*

Figura 2 – Componentes de *hardware* que compõem um microcontrolador



Fonte: Pereira (2016, p. 2).

O processador de um microcontrolador é responsável por gerenciar todos os periféricos conectados a ele. Este gerenciamento acontece basicamente através da busca e execução de instruções contidas na memória de programa, o resultado do processamento destas instruções será uma ação que o microcontrolador deverá executar através de seus periféricos de entrada e saída. Tais instruções podem ser do tipo RISC (*Reduced Instruction Set Computers*) ou CISC (*Complex Instruction Set Computers*), ou ainda a combinação destes dois tipos, que permite que instruções mais complexas se tornem mais simples, resultando em uma velocidade maior de processamento e consequentemente um tempo menor de resposta para tal instrução.

Segundo Predko (1999), existem dois ou três tipos de memória nos microcontroladores:

- Memória de programa não volátil;
- Memória volátil;
- Memória de dados não volátil.

A memória de programa não volátil é o ponto de partida quando o microcontrolador é energizado, pois é ela a responsável por armazenar o código do programa que será executado. O termo não volátil significa que ela retém as informações mesmo na ausência de energia, ou seja, ao retirar a alimentação do circuito, a memória deverá manter os dados do programa intactos para serem utilizados quando o microcontrolador for religado.

A memória de dados volátil, também conhecida como memória RAM, é normalmente a mais rápida dentre as memórias presentes num microcontrolador, por isso ela é utilizada para guardar dados do programa em tempo real de execução. Ou seja, quando o microcontrolador é energizado o processador passa a carregar os dados contidos na memória de programa e os salva na memória RAM, para posteriormente buscar, executar e salvar as instruções diretamente na memória RAM. Neste tipo de operação a velocidade de leitura e escrita da memória é crucial para o desempenho do microcontrolador, por isso, essa memória é considerada um complemento do processador, pois o mesmo necessita dela para realizar suas operações.

Opcional, a memória de dados não volátil normalmente tem como objetivo "guardar" dados de transmissões entre dispositivos para utilizar numa próxima execução. Nestes dados geralmente está contido informações para dispositivos de redes, endereços IP (*Internet Protocol*), dados de calibração (PREDKO, 1999).

Por fim, os periféricos de entrada e saída permitem que o microcontrolador se conecte a diversos dispositivos de hardware externos, tais periféricos podendo ser portas, conversor A/D, oscilador, dentre outros.

2.2 Sensores e atuadores

Sensores e atuadores são dispositivos que permitem um equipamento eletrônico interagir com o meio. O sensor é capaz de detectar uma determinada condição do ambiente como, temperatura por exemplo, e transformá-la em um sinal elétrico para que possa ser interpretada por uma placa de aquisição e controle. No caminho inverso o atuador recebe um sinal elétrico da placa controladora e realiza uma ação no ambiente ligando uma lâmpada ou motor por exemplo (BANZI; SHILOH, 2015).

Uma analogia a sensores e atuadores é o corpo humano que "sente" determinadas condições do meio e as envia para o cérebro através de pulsos elétricos, esses impulsos são processados e enviados como ações que os músculos devem realizar (BANZI; SHILOH, 2015).

Os sensores podem operar de forma direta, conhecidos como transdutores, que mensuram uma grandeza física e a transformam em outro tipo de sinal, ou ainda, podem operar de forma indireta, que ao mensurar algum tipo de sinal alteram suas propriedades como resistência, indutância e capacitância de forma mais ou menos proporcional a este sinal detectado (BORGES; DORES, 2010).

Os sensores são classificados em duas categorias, os analógicos e digitais. Nos sensores analógicos o sinal elétrico de saída pode variar infinitesimalmente entre dois valores

de tensão estabelecidos, ou seja, para cada pequena variação na condição do meio haverá um sinal de saída correspondente. Já os sensores digitais trabalham com uma lógica binária, onde só podem alternar entre estados bem definidos, não podendo haver valor intermediário entre estes estados.

Os atuadores são dispositivos que convertem o comando de um controlador em um parâmetro físico de atuação no processo. Atuadores normalmente são elétricos, mecânicos e pneumáticos (PESSÔA; SPINOLA, 2014). Relé é um dos exemplos mais simples de atuador, este dispositivo eletromecânico é capaz de acionar circuitos de potência elevada através de um pequeno sinal de corrente advindo da placa controladora.

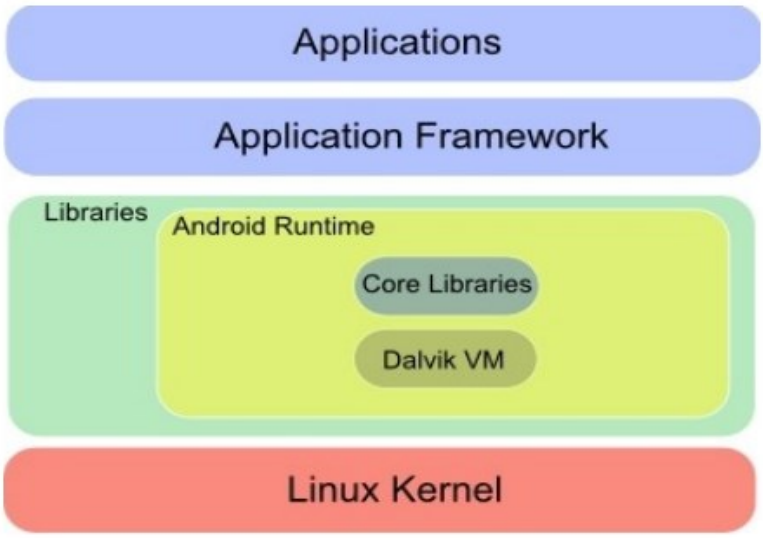
2.3 Plataforma Android

O *Android* é um sistema operacional para dispositivos móveis lançado em 2007. Desde seu lançamento vem crescendo rapidamente, alcançando a marca de 375 milhões de *smartphones* no terceiro trimestre de 2016, o que representa 88% de todos *smartphones* presentes no mundo (SUI, 2016). Atualmente o *Android* está presente em cerca de 72% de todos os *smartphone* existentes (STATCOUNTER GLOBALSTATS, 2022).

Baseado no núcleo do Linux, o *Android* é um sistema operacional *Multi-Thred* capaz de executar vários aplicativos e processos ao mesmo tempo (VOLPATO, 2016). É um *software open Source* desde 2009 e tem seus aplicativos desenvolvidos na linguagem de programação Java, rodando em cima de bibliotecas desenvolvidas pela Google.

A arquitetura *Android* é dividida em quatro subcamadas, como ilustra a Figura 3. De cima para baixo, primeiramente está a camada de aplicação, na sequência *framework* de aplicação, bibliotecas e serviços nativos, e por fim núcleo Linux.

Figura 3 – Arquitetura *Android*



Fonte: Gandhewar e Sheikh (2010, p. 25).

Na camada de aplicação estão os serviços essenciais ao usuário, visto em forma de aplicativos, como e-mail, navegador, calendário, rede social, entre outros. A camada de *Application Framework* representa o *software* de desenvolvimento que padroniza a implementação de aplicativo para sistema operacional *Android*, ou seja, esta é a camada utilizada pelos desenvolvedores para construir suas aplicações. A terceira camada é dividida em duas partes. Na primeira estão as bibliotecas que dão auxílio ao desenvolvedor, escritas em C/C++. A segunda parte contém o *Android Runtime*, onde se encontram as bibliotecas do núcleo Java e a máquina virtual Dalvik, responsável por converter código escrito em linguagem de programação Java para linguagem de máquina, legível ao processador.

A última camada é a base do sistema operacional, baseado na versão 2.6, o núcleo Linux atua no gerenciamento de memória, energia e periféricos de hardware, o que representa a última abstração entre *software* e *hardware*. Ou seja, ele é responsável por gerenciar a execução de instruções no processador e controlar dispositivos como: câmera, G*lobal Positioning System* (GPS), acelerômetro, entre outros. Nesta camada estão contidos todos os *drivers* de *hardwares* necessários para o funcionamento correto do sistema operacional *Android*.

2.4 Redes de computadores

As redes de computadores surgiram na década de 60 com o objetivo de interligar dois computadores e trocar dados entre eles. Segundo Sousa (1999), rede de computadores é um conjunto de dispositivos computacionais interligados de maneira a trocarem informações e

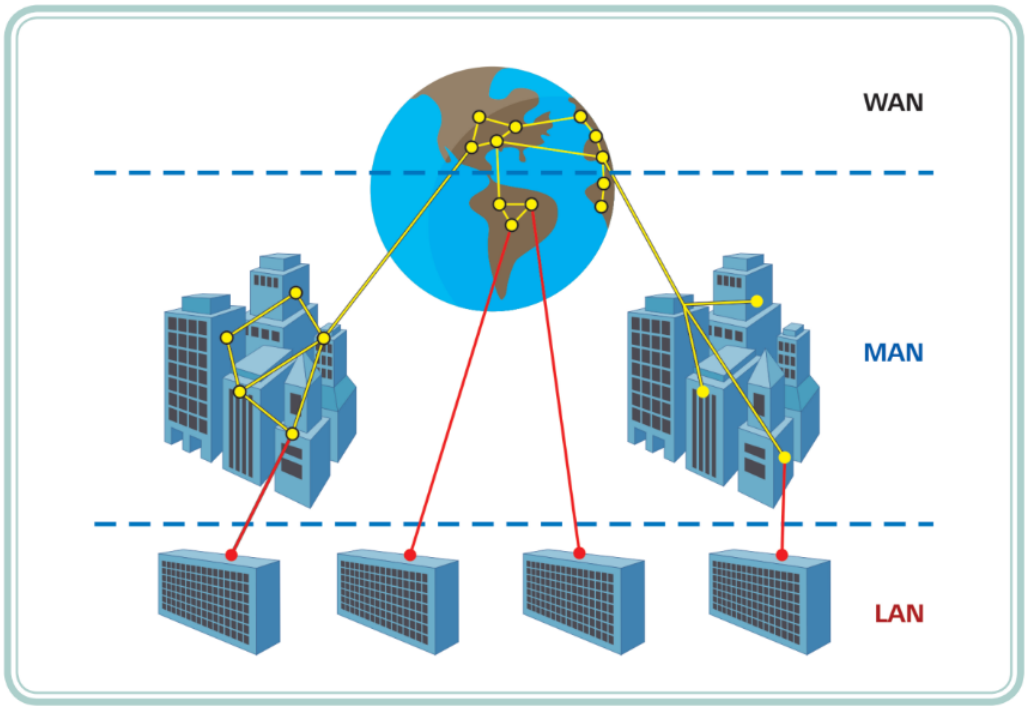
compartilharem recursos como arquivos de dados, impressoras, modems, *softwares* e outros equipamentos. As redes de computadores são classificadas conforme sua dispersão e abrangência geográfica. Assim, é convencional a classificação das redes em locais (LANs - *Local Area Networks*), metropolitanas (MANs - *Metropolitan Area Networks*) e geograficamente distribuídas (WANs - *Wide Area Networks*) (DANTAS, 2002).

A rede LAN é uma rede de abrangência local, ou seja, é uma rede de curto alcance. Esse tipo de rede opera em um ambiente restrito e nas suas proximidades, como em uma residência, escritório, empresa, sua principal característica é ser uma rede privativa gerenciada por uma pessoa ou empresa. Os equipamentos que compõem esse tipo de rede normalmente são computadores, *smartphones*, *switches* e demais dispositivos que têm placa de rede, como é o caso do ESP8266 e a *Raspberry Pi*, utilizados no desenvolvimento da automação residencial do presente trabalho.

A rede MAN é uma rede de abrangência metropolitana, esse tipo de rede interliga várias redes LAN dentro de uma cidade ou região. Como exemplo, a rede MAN é utilizada para interligar unidades de uma mesma empresa que estão localizadas em diferentes pontos da cidade.

A rede WAN é uma rede de longa distância, utilizada para interligar as redes de menor alcance, como as redes MAN e redes LAN, a Figura 4 ilustra essas interligações. O alcance da rede WAN é imensurável, ela pode interligar regiões, cidades, países e até continentes, a interligação das redes WAN formam a rede de internet, que nos possibilita acessar serviços do mundo inteiro a partir de qualquer dispositivo conectado à essa rede. É através da rede de internet que é possível controlar de qualquer local do mundo os dispositivos de automação residencial propostos neste trabalho.

Figura 4 – Exemplo de interligação entre as redes LANs, MANs e WANs



Fonte: Cristo *et al.* (2014, p. 19).

Com a evolução da tecnologia sem fio (*wireless*) surgiram outras nomenclaturas de redes, para tal, basicamente foi adicionado a letra W de *wireless* as siglas já conhecidas e discutidas neste capítulo. A WLAN é uma rede local sem fio, sua função é interligar todos os dispositivos de uma residência que tenham tecnologia *wireless,* por exemplo. Seguindo nesse mesmo contexto, a WMAN é uma rede metropolitana sem fio, comumente utilizada pelas operadoras de telecomunicações para possibilitar a comunicação sem fio entre redes LAN da mesma cidade ou região. A WMAN é a versão sem fio da rede MAN, é uma rede sem fio de longo alcance utilizada pelas operadoras para criar uma rede de transmissão celular.

No presente trabalho é utilizado uma rede WLAN para fornecer acesso sem fio aos dispositivos da residência, como ESP8266, *Raspberry PI*, celular e computador, e, também é utilizada a rede WWAN (internet) para acesso e controle remoto da página *web.*

2.5 Trabalhos relacionados

Dentre os diversos trabalhos relacionados com Automação Residencial e Domótica, a seguir estão descritos alguns deles que foram selecionados por ter uma maior semelhança com o trabalho proposto.

Lucas Bragazza Beghini (2013) apresenta um sistema de automação residencial de baixo custo que utiliza um *Arduino Uno* como central de comando. O projeto permite ao usuário

controlar a alimentação de animais de estimação, iluminação e alarme da residência através de um aplicativo *Android* instalado num dispositivo móvel. Tal controle pode ocorrer de qualquer lugar onde haja conexão com a internet, já que neste caso o *Arduino Uno* se comunica com o dispositivo móvel através da rede.

Eduardo Pereira (2014) descreve seu trabalho como soluções inteligentes de baixo custo para automação residencial, no qual o usuário pode controlar eletrônicos da residência através de um aplicativo de celular com sistema operacional *IOS*. As ações de comando são mensagens de caracteres enviadas a partir do celular para um *Raspberry Pi* via *twitter*, no *Raspberry Pi* as mensagens são processadas e transmitidas via rádio frequência a um *Arduino* Uno, que as executa através de atuadores.

Lucas Longo (2015) apresenta um sistema de automação residencial baseado no paradigma Internet das Coisas, onde o usuário pode escolher através de uma interface *web* ações que devam ser realizadas dentro da residência ou ainda deixar em modo automático para que o próprio sistema as coordene. Tais ações de comando são recebidas via um servidor *web* instalado dentro de um *Raspberry Pi*, o qual é responsável por transmitir estes comandos via rádio frequência a uma placa de aquisição e controle, que por sua vez deve gerir os sensores e atuadores presentes na moradia.

3 MATERIAIS E MÉTODOS

A seguir estão descritas os materiais, ferramentas e tecnologias que foram utilizadas no desenvolvimento do projeto. Também será apresentado a interação entre os *hardwares* do protótipo e o funcionamento do sistema.

3.1 Materiais

O Quadro 1 apresenta as ferramentas de *hardware* utilizadas e suas finalidades no desenvolvimento do protótipo deste projeto.

Quadro 1 – Periféricos utilizados no projeto

Ferramenta	Datasheet	Finalidade
Raspberry Pi B Rev2	https://www.raspberrypi.org/documentation/	Placa de aquisição e controle utilizada como servidor *web* e responsável pelo gerenciamento dos microcontroladores ESP8266.
Microcontrolador ESP8266 12E	https://components101.com/sites/default/files/component_datasheet/ESP12E%20Datasheet.pdf	Microcontrolador *Wi-Fi* responsável por executar ações recebidas do servidor e enviar leituras de sensores.
Adaptador *Wi-Fi* TL-WN722N	https://www.tp-link.com/br/support/download/tl-wn722n/	Adaptador utilizado na *Raspberry Pi* para fornecer conexão *Wi-Fi.*
Sensor de temperatura e umidade DHT22	https://datasheetspdf.com/pdf-file/792211/Aosong/DHT22/1	Mensurar temperatura e umidade da residência.
Relé	https://www.digchip.com/datasheets/parts/datasheet/413/SRU-S-105L-pdf.php	Acionar dispositivos eletroeletrônicos de potência.
Smartphone Motorola G7 Plus		Executar aplicativo *Android* responsável pelo acesso a página *web* e determinar a localidade do usuário.

Fonte: Autoria própria.

O Quadro 2 apresenta as ferramentas de *software* utilizadas e suas finalidades na implementação do sistema.

Quadro 2 – Ferramentas de desenvolvimento utilizadas

Software	Versão	Referência	Finalidade
Raspbian	*Stretch*	https://www.raspberrypi.org/downloads/raspberry-pi-os/	Sistema operacional Linux utilizado no *Raspberry Pi*.

Win32DiskImager		https://win32diskimager.download/	*Software* utilizado para montar a imagem do SO *Raspbian* dentro do cartão SD.
Arduino IDE	1.8.13	https://www.arduino.cc/en/main/software	Ambiente de desenvolvimento para programação dos microcontroladores ESP8266.
Flask	1.1	https://flask.palletsprojects.com/en/1.1.x/	*Framework* responsável por criar o servidor *web* para gerenciamento do sistema.
Eclipse Oxigen	2		Ambiente para desenvolvimento do servidor *web* (*back-end*).
Android Studio	2021.3	https://developer.android.com/studio/index.html	Ambiente de desenvolvimento para aplicativos *Android.*
Linguagem *Python*		https://python.org.br/	Linguagem de programação utilizada no lado do servidor *web (back-end)*.
Linguagem C			Linguagem de programação utilizada nos microcontroladores ESP8266.
HTML			Linguagem de marcação para desenvolvimento da estrutura das páginas *web*.
CSS			Linguagem para desenvolvimento dos elementos estéticos das páginas *web.*
JavaScript			Linguagem de programação para desenvolvimento da interatividade das páginas *web.*
SQLite		https://www.sqlite.org/	Banco de dados para armazenamento das leituras de sensores e perfil de usuário.
Bootstrap		https://getbootstrap.com/	Framework *web* utilizado para desenvolvimento da interface de usuário.
MQTT		https://mqtt.org/	Protocolo de troca de mensagens entre o servidor *web* e os microcontroladores ESP8266.
Mosquitto		https://mosquitto.org/	Gerenciador de mensagens utilizado pelo protocolo MQTT.
Ngrok		https://ngrok.com/	Ferramenta para expor o servidor *web* local na internet.

Fonte: Autoria própria.

3.1.1 Raspberry PI

O *Raspberry Pi* é um computador de tamanho reduzido capaz de desempenhar as mesmas funções que um computador de tamanho normal. Esse minicomputador foi criado originalmente no Reino Unido para auxiliar no ensino de ciência da computação em escolas e

universidades, foi lançado em 2012 como *Raspberry Pi* 1 e atualmente está no seu quarto modelo. Neste projeto foi utilizado o *Raspberry* 1 *Model B Rev2,* com as seguintes especificações:

- Processador: BCM2835, 700 MHz, família ARM11;
- Processamento gráfico: Broadcom VideoCore IV, OpenGL ES 2.0, OpenVG 1080p30 H.264/MPEG-4 AVC High-Profile;
- Memória (SDRAM): 512 MB;
- Saída de vídeo: RCA, HDMI;
- Saída de áudio: 3.5 mm, HDMI;
- Armazenamento: Slot disponível para cartão de memória SD/MMC;
- 2 portas USB2.0;
- Porta Ethernet: 10/100 Ethernet RJ45;
- Alimentação: 5V/700mA DC via micro USB;
- Dimensões: 85.6 mm x 53.98 mm;
- Expansão: 26 pinos;
- Sistema operacional: GNU/Linux *Raspbian Stretch Lite*;

Neste trabalho o *Raspberry Pi* é usado como servidor *web* e gerenciador de microcontroladores ESP8266 que estão conectados a sensores e atuadores.

3.1.2 Microcontrolador ESP8266

A comunicação sem fio entre o *Raspberry Pi* e os dispositivos remotos ESP8266 é de suma importância para instalação e automação de periféricos na residência, visto que não se faz necessário a utilização de cabos de comunicação entre eles.

O ESP8266 é um microcontrolador fabricado pela Espressif com comunicação *Wi-Fi* embutida que permite ao mesmo se conectar a uma rede sem fio utilizando o protocolo TCP/IP. De tamanho reduzido e preço relativamente baixo, existem até doze variantes do módulo ESP8266 no momento, com diferentes tamanhos e quantidades de entradas/saídas disponíveis. No presente trabalho serão utilizados módulos ESP8266-12 com as seguintes características:

- Processador: 32-bit RISC, 80 MHz;
- Memória: 64 KB de memória RAM de instruções, 96 KB de dados;
- Armazenamento: Flash QSPI externo de 512 KB a 4 MB;

- Rede sem fio: IEEE 802.11 b/g/n;
- Frequência *Wi-Fi*: 2.4 GHz – 2.5 GHz;

3.1.3 Sensores

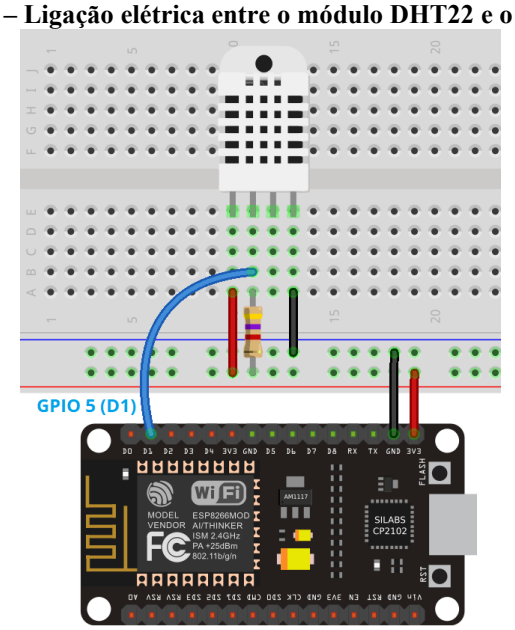
Nesta seção estão descritos os tipos de sensores que foram utilizados no desenvolvimento deste trabalho e suas características.

3.1.3.1 Sensor de temperatura e sensor de umidade DHT22

O DHT22 é um sensor digital de temperatura e umidade em um único módulo. Para obter a temperatura ele utiliza um termistor com capacidade de medir temperaturas entre -40 a +80 ° Celsius, com precisão de 0,1 graus e para medir umidade utiliza um sensor capacitivo com uma faixa que varia de 0 a 100 %, com precisão de 0,1%.

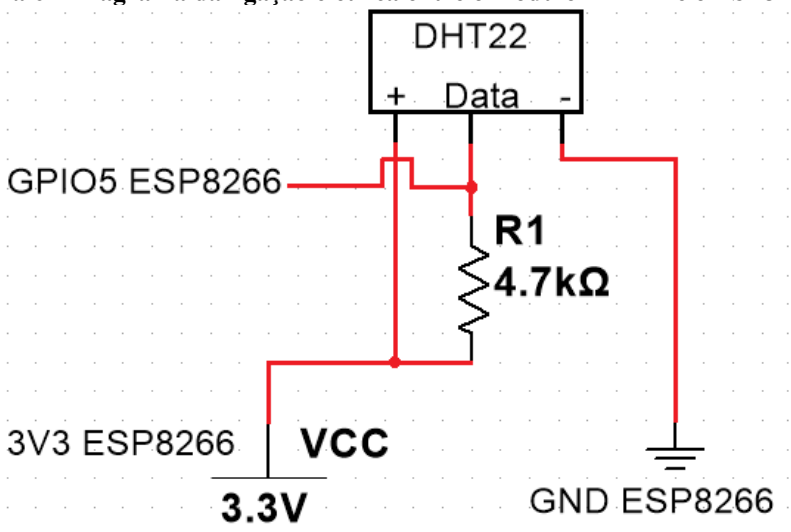
Para realizar a medição com o DHT22 é preciso alimentar o pino 1 com 3,3 V e o pino 4 com 0 V (GND). O pino 2, saída de dados, é conectado a entrada D1 do ESP8266 e também a um resistor de *pullup* de 4,7 kΩ ligado a alimentação, conforme exibe a Figura 5.

Figura 5 – Ligação elétrica entre o módulo DHT22 e o ESP8266



Fonte: Autoria própria

Figura 6 – Diagrama da ligação elétrica entre o módulo DHT22 e o ESP8266



Fonte: Autoria própria

3.1.4 Atuadores

Nesta seção está descrito o tipo de atuador que foram utilizados no desenvolvimento do presente trabalho e suas características.

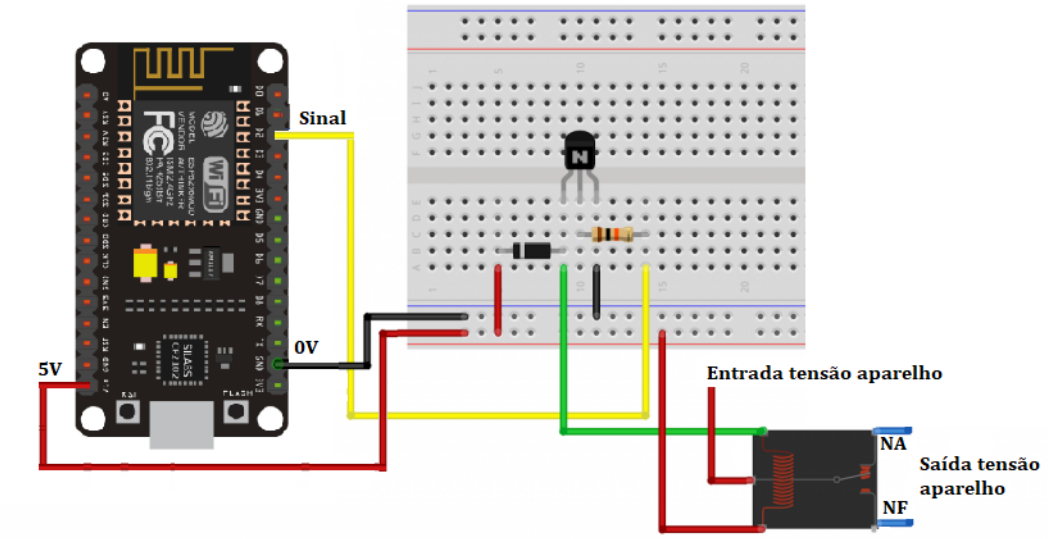
3.1.4.1 Circuito acionador de relé

O circuito acionador é responsável por acionar o relé a partir do sinal de um pino de saída do microcontrolador ESP8266. O relé é um dispositivo eletromecânico que através de um sinal de baixa tensão de entrada consegue acionar equipamentos elétricos que necessitam de tensões elevadas para entrarem em funcionamento como uma lâmpada, por exemplo. Ele também tem a função de isolar eletricamente o circuito acionador do circuito acionado.

O relé utilizado neste projeto necessita de 5V/70mA na entrada para acionar sua saída, o pino de saída do microcontrolador ESP8266 fornece no máximo 3,3V/12mA, por isso a necessidade de utilizar o circuito acionador que está exibido na Figura 6. Os componentes necessários para montagem do circuito são:

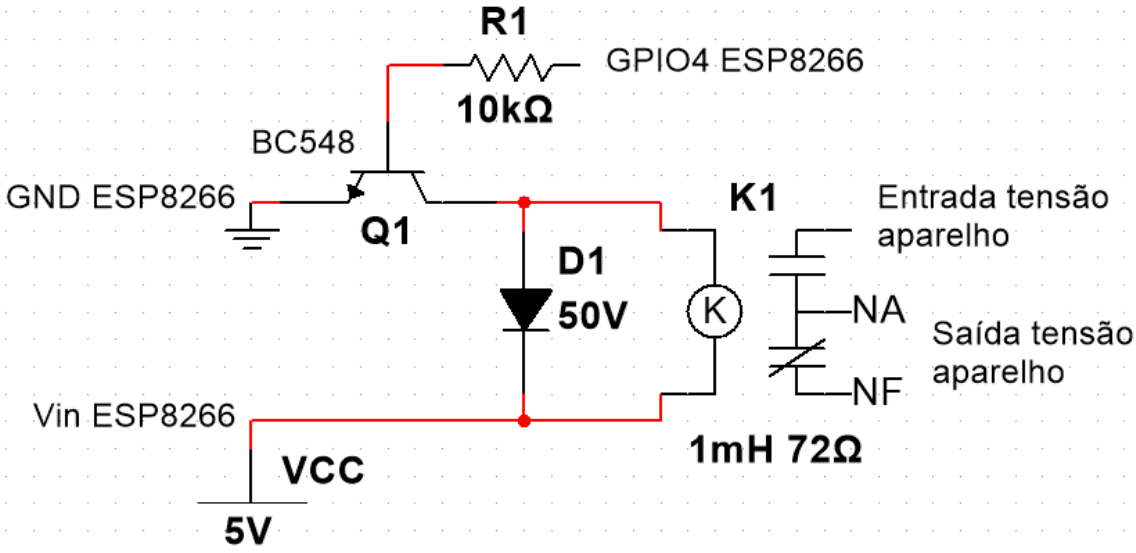
- 1 Relé 5 V;
- 1 Diodo 1N4001;
- 1 Transistor BC548;
- 1 Resistor 10 kΩ.

Figura 7 – Ligação elétrica do circuito acionador de relé



Fonte: Autoria própria

Figura 8 – Diagrama elétrico do circuito acionador de relé



Fonte: Autoria própria

O pino Vin e GND do ESP8266 é responsável por fornecer 5 V e 0 V ao circuito, respectivamente. Quando o microcontrolador aplica 3,3V (nível lógico 1) à base do transistor, por meio do resistor de base, a corrente que circulará será suficiente para saturar o transistor, fazendo com que a bobina do relé seja magnetizada. Quando o microcontrolador aplica 0 V à base do transistor, o transistor permanece em corte e a bobina do relé não é magnetizada. O diodo funciona como diodo de roda livre quando o relé estiver sendo desligado.

3.2 Métodos

Nesta seção é apresentado os métodos que foram utilizados para desenvolvimento do presente trabalho. Inicialmente, de forma simplificada será contextualizado como funciona a interação entre os periféricos utilizados neste projeto e, na sequência, será aprofundado e explicado a instalação e configuração de cada ferramenta utilizada.

A seguir estão descritas as etapas que serão utilizadas para o desenvolvimento do presente trabalho.

a) **Detalhamento de requisitos**

   Nesta etapa será verificado a disponibilidade dos sensores e atuadores contidos na universidade que possam ser utilizados para este trabalho e, verificar a necessidade de possível aquisição por parte do acadêmico de componentes necessários não encontrados.

b) **Análise e modelagem do sistema**

   Analisar cada componente e modelar seu relacionamento com os demais periféricos do sistema através de diagramas.

c) **Implementação dos *softwares***

   Etapa de desenvolvimento do servidor *web*, banco de dados e aplicativo *Android* para interface com o usuário.

d) **Desenvolvimento de *firmware***

   Implementação do *firmware* do microcontrolador para gerenciar sensores, atuadores e comunicar com o servidor.

e) **Construção do protótipo**

   Construção do protótipo da residência para demonstrar o funcionamento dos sensores e atuadores integrados com o sistema de controle.

f) **Testes**

   Nesta etapa serão realizados testes para verificar possíveis erros no projeto e/ou na codificação, e, falhas em componentes elétricos.

g) **Possíveis correções**

   Realizar correções de possíveis erros ou falhas encontradas na etapa de testes.

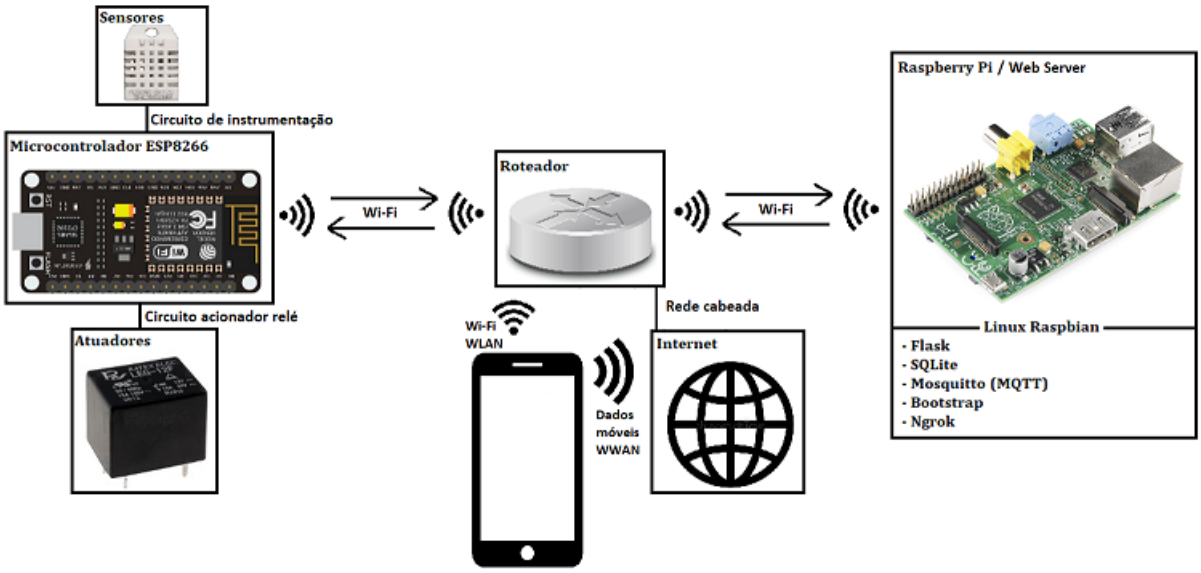
h) **Elaboração da monografia**

   Nesta etapa será redigido e documentado o TCC.

3.2.1 Funcionamento geral

A Figura 8 ilustra como os *hardwares* do protótipo estão conectados entre si. O sistema utiliza um *Raspberry Pi* com Sistema Operacional Linux *Raspbian* e o *framework Flask* com linguagem de programação *Python*. Esse conjunto de ferramentas constitui um servidor capaz de gerar uma página *web* para interação com o usuário. Ele também é responsável por gerir todos os dispositivos remotos que estão conectados a sensores e atuadores espalhados pela residência. A instalação e configuração destas ferramentas está descrito na seção 3.2.3.

Figura 9 - Exemplo da forma de interação entre os dispositivos do projeto



Fonte: Autoria própria.

Na página *web,* com o sistema em modo manual, o usuário tem a possibilidade de escolher uma ação a ser realizada instantemente por um atuador remoto ou então agendar a tarefa para ser executada quando o sistema estiver operando em modo automático. Se o usuário optar por uma ação instantânea, o servidor irá processar essa ação e enviá-la ao dispositivo remoto correspondente através de uma comunicação *wireless*, fornecida e gerenciada por um roteador. Em caso do agendamento ser inserido no perfil, as ações escolhidas geram uma lista de tarefas do usuário que é salvo em uma base de dados para serem executadas quando a residência estiver em modo automático de operação, o salvamento também garante a consistência nos dados em caso de reinício no servidor.

Leituras de sensores de temperatura, umidade e luminosidade são processadas pelo ESP8266 e enviadas ao servidor regularmente, na sequência são armazenadas em tabelas e salvas na base de dados do servidor, para posteriores consultas e exibição de gráficos estáticos.

As informações de perfil de usuário e leituras de sensores são salvas em um banco de dados *SQLite,* instalado no *Raspberry Pi,* conforme descrito na seção 3.2.4.

Os dispositivos remotos ESP8266 são microcontroladores com comunicação *Wi-Fi* embutida que permite ao mesmo se conectar a uma rede sem fio utilizando o protocolo TCP/IP. Esses módulos estão conectados fisicamente a atuadores e executam ações recebidas do servidor. Cada dispositivo remoto ESP8266 é um nó de rede e se mantém conectado ao roteador da residência aguardando comandos e atualizando o servidor com leituras de sensores. Para que a comunicação entre o servidor e os dispositivos remotos ocorram de forma rápida e sem sobrecarregar a rede LAN da residência, foi utilizado o protocolo de comunicação MQTT, que tem como principal vantagem o tamanho reduzido do pacote de dados trocado. A seção 3.2.5 explica como configurar e utilizar esse protocolo.

3.2.2 Modos de operação

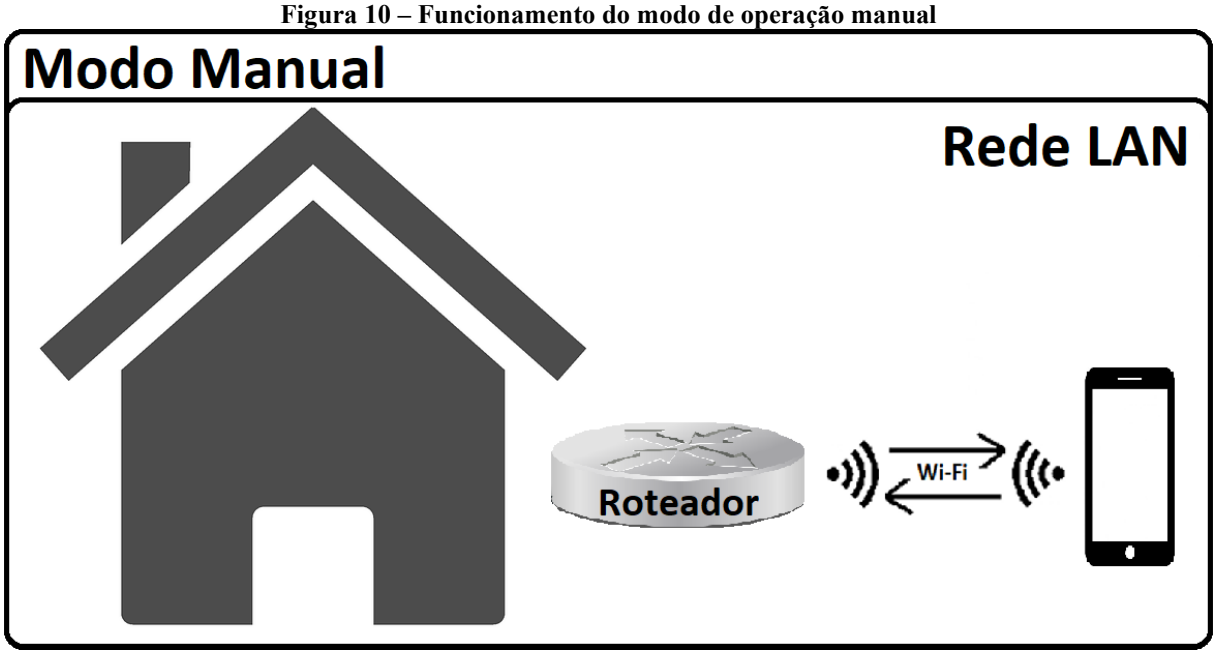
Os modos de operação dizem de que forma e em que momento o sistema irá acionar os dispositivos da residência, o acionamento pode ser realizado de duas maneiras, manual ou automático. Na proposta inicial do presente trabalho foi utilizado a posição geográfica (GPS) do *smartphone* para alternar entre os modos de funcionamento, ou seja, quando o *smartphone* estivesse próximo a residência o modo manual era ativado, quando o *smartphone* estivesse fora da região da residência, o modo automático entrava em funcionamento. Esse método se mostrou pouco eficiente visto que a leitura do GPS nem sempre era possível ser obtida devido a sinal fraco, como ocorre em ambientes enclausurados como prédios e garagens. O consumo da bateria do *smartphone* também seria bastante afetada utilizando o GPS, visto que seria necessário enviar a posição geográfica a cada pequeno intervalo de tempo para atualizar o servidor. Então, foi alterado o método de obter a localidade do usuário, passando a consultar o endereço IP do *smartphone* para identificar se o dispositivo se encontra na rede local (LAN) da residência ou na rede remota.

3.2.2.1 Modo manual

O modo manual é ativado sempre que o usuário estiver conectado à rede local (LAN) da residência, nesse modo as tarefas agendadas no perfil do usuário não são executadas. Seguindo o mesmo princípio de verificação do endereço IP do modo automático, descrito anteriormente, quando o IP consultado pelo servidor *web* pertence a rede local, sabe-se que o

usuário se encontra na residência, logo, o sistema entra em modo manual de funcionamento e executa somente comandos ordenados manualmente pelo usuário.

Figura 10 – Funcionamento do modo de operação manual

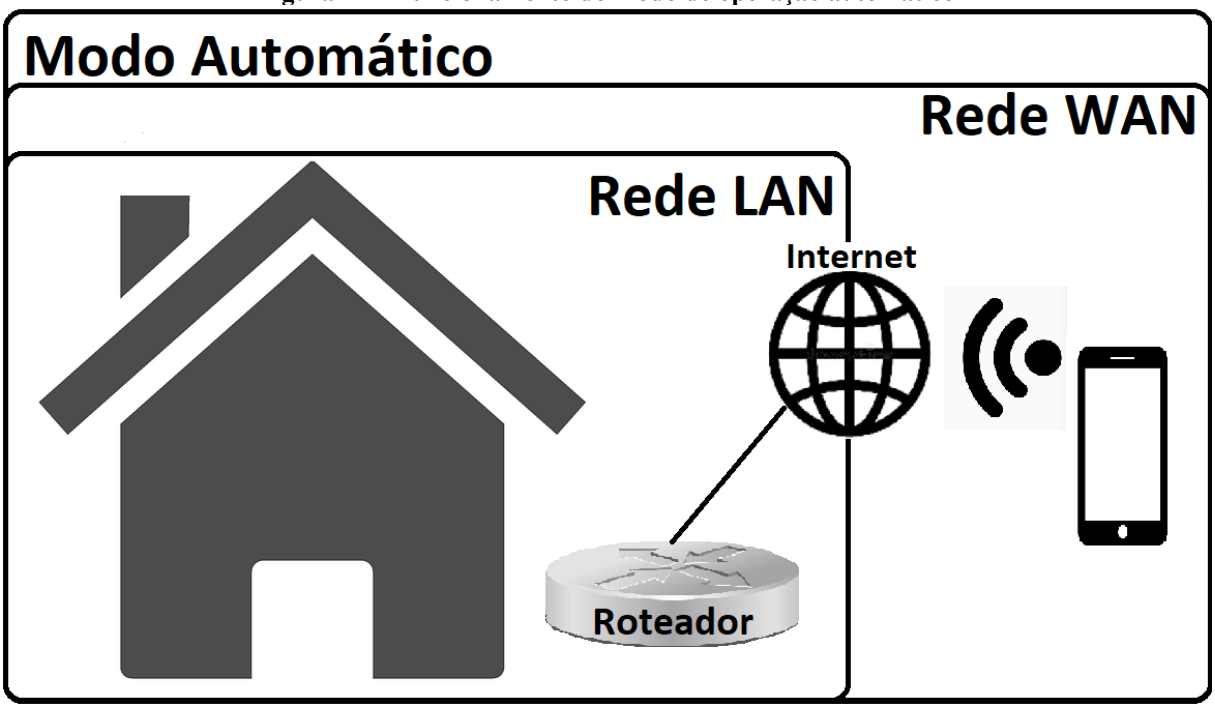


Fonte: Autoria própria.

3.2.2.2 Modo automático

O modo de operação automático executa as tarefas previamente agendadas pelo usuário conforme a rede que o *smartphone* está conectado. Quando o usuário cria um agendamento periódico e o adiciona no perfil, essa tarefa será executada sempre que o usuário estiver ausente da residência, ou seja, sempre que o usuário não estiver conectado a rede local (LAN). Toda vez que o usuário faz uma requisição ao servidor através da página *web,* o IP do dispositivo requisitante é verificado pelo servidor, se ele pertencer ao domínio público, o dispositivo encontra-se em uma rede remota (WAN) indicando que o usuário não está na residência, neste caso, o modo automático entra em operação e executa as tarefas que estão agendadas no perfil do usuário.

Figura 11 – Funcionamento do modo de operação automático



Fonte: Autoria própria.

3.2.3 Instalação e configuração do servidor *web* no *Raspberry Pi*

Para que seja possível instalar o *framework Flask* no *Raspberry Pi* e torná-lo um servidor *web,* primeiro é necessário colocar o Sistema Operacional (SO) *Raspbian* em um cartão de memória para ser executado no *Raspberry Pi*.

O *Raspbian* é uma variante da distribuição *GNU/Linux Debian* otimizada para o hardware do *Raspberry Pi.* Neste SO foram ajustadas configurações de compilação e otimizado instruções, a fim de fornecer melhor desempenho quando executado na arquitetura ARM que compõe o *Raspberry Pi*. Para um melhor desempenho do servidor *web* foi utilizado a versão *Raspbian Stretch* sem interface gráfica, desta forma, todos os programas necessários para este projeto foram instalados no SO através de linha de comando.

Com um cartão SD inserido no computador, formatado em FAT32 e a imagem do *Raspbian* baixada*,* abrir o *software Win32DiskImager*, selecionar o caminho onde está a imagem e a unidade onde deseja mantê-la, por fim, clicar em *Write*. Após o software finalizar o processo, deve-se remover o cartão SD do computador e inserir no *slot* do *Raspberry Pi*.

Agora, de posse de um monitor, teclado e um adaptador *Wi-Fi*, conecta-los ao *Raspberry Pi* e ligar a fonte de alimentação. Ao iniciar o sistema operacional, o *login* e senha são solicitados, sendo "**pi**" e "**raspberry**", respectivamente. É necessário configurar a conexão com a internet para ter acesso externo ao servidor *web*. Ao executar o comando na sequência,

o arquivo de configuração wpa_supplicant.conf é aberto pelo editor nano, conforme exibe a Figura 10. No final deste arquivo é possível inserir o nome da rede *Wi-Fi* e a senha dela.

pi@raspberrypi~ $ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

Figura 12 – Arquivo de configuração wpa_supplicant.conf

Fonte: Autoria própria.

Após inserir os parâmetros é necessário salvá-los e reiniciar a interface de rede com o comando:

pi@raspberrypi~ $ sudo wpa_cli reconfigure

Para facilitar o desenvolvimento foi habilitada a conexão SSH do *Raspberry Pi.* Desta forma, é possível acessá-lo por outro computador que esteja na mesma rede *LAN*, não sendo mais necessário a utilização de monitor e teclado conectados ao *Raspberry Pi*. Ao executar o comando a seguir será exibido a tela de configurações do *Raspberry Pi*, selecionar a opção *Interfacing Options,* na tela seguinte selecionar SSH e habilitar.

pi@raspberrypi~ $ sudo raspi-config

A partir desse ponto já é possível acessar o *Raspberry Pi* remotamente pelo programa *Putty,* digitando o IP e selecionando o tipo da conexão como SSH.

O *Flask* é um micro *framework* escrito em *Python* baseado na biblioteca *Jinja2* e nas ferramentas *Werkzeug WSGI* (*Web Server Gateway Interface*). Ele oferece soluções simples para criação de pequenos aplicativos *web*, porém, também suporta aplicações robustas com eficiência.

Antes de instalar *Flask* no ambiente Linux é necessário instalar o PIP, sistema responsável por gerenciar os pacotes de *software* escritos na linguagem *Python*. Para garantir

que a lista de pacotes do sistema operacional esteja atualizada, é necessário primeiramente executar os comandos:

pi@raspberrypi~ $ sudo apt-get update

pi@raspberrypi~ $ sudo apt-get upgrade

O comando na sequência fará a instalação do PIP:

pi@raspberrypi~ $ sudo apt-get install python-pip

E, por fim, os próximos comandos instalam o servidor *web Flask* no *Raspberry Pi*:

pi@raspberrypi~ $ sudo apt-get install python-flask git-core

pi@raspberrypi~ $ sudo pip install flask

Para permitir que a página *web* seja atualizada automaticamente e exiba as últimas leituras dos sensores de forma assíncrona, é necessário instalar o pacote SocktIO do *Flask*, com o comando a seguir:

pi@raspberrypi~ $ sudo pip install flask-socketio

Também foi necessário configurar o roteador *Wi-Fi* da residência para atribuir um endereço de IP fixo a placa de rede da *RaspberryPI*, para garantir que o IP do servidor *Flask* e servidor de mensagens MQTT seja sempre o mesmo e possibilitar a comunicação com os módulos ESP8266.

3.2.4 Instalação do banco de dados

Para que seja possível armazenar dados temporais coletados por sensores presentes na residência, como temperatura e umidade, e posteriormente gerar estatísticas para o usuário, é necessário a utilização de um banco de dados para armazenamento. Além do mais, o perfil de usuário deve ser definido pelo mesmo com base em algumas de suas escolhas que devem ser salvas em uma tabela para posteriores consultas e tomadas de decisão por parte do sistema de controle.

Como o *Raspberry Pi* é um pequeno computador com poder de processamento reduzido, o banco de dados a ser executado nele deve ser leve e rápido o suficiente para garantir a confiabilidade no salvamento e leitura de dados.

O SQLite é um banco de dados simples e rápido, desenvolvido em linguagem C e amplamente utilizado em sistemas embarcados. Sua arquitetura de servidor faz com que não seja necessário dependências externas para seu funcionamento, ele lê e escreve diretamente em um arquivo em disco, não é preciso executar um processo SGBD separado, como é o caso de banco de dados mais robustos.

Para instalar o SQLite no *Raspberry Pi* é preciso somente executar o comando:

pi@raspberrypi~ $ sudo apt-get install sqlite3

Para testar seu funcionamento é possível criar uma base de dados qualquer chamada "teste" com o seguinte comando:

pi@raspberrypi~ $ sqlite3 teste.db

A partir do comando anterior, o *Shell* do SQLite é invocado e já é possível criar e ler e tabelas utilizando a linguagem SQL.

3.2.5 Configuração do MQTT

O MQTT é um protocolo de mensagens com foco em IOT que funciona em cima do protocolo TCP/IP e se baseia na comunicação cliente e servidor. Ele proporciona comunicação bilateral entre dispositivos com mínimos recursos de largura de banda (baixo consumo de dados) e poder de processamento, por isso pode ser utilizado em microcontroladores pequenos.

Foi projetado para transmitir mensagens pelo modelo de publicação e assinatura, denominado paradigma *Publish-Subscribe*. Nesse modelo um publicador publica a mensagem em um tópico e o assinante deve se inscrever neste tópico para visualizar a mensagem, não há conexão direta entre o emissor e o assinante. Esse modelo de comunicação permite enviar mensagens que podem ser lidas por nenhum, um ou vários clientes e, também, todos os clientes podem transmitir e receber mensagens.

Para que esse protocolo funcione é necessário a utilização de um *Broker*, que tem a função de filtrar as mensagens com base no tópico em que foram publicadas e, em seguida, distribuí-las aos assinantes do tópico correspondente. Existem vários *Brokers* gratuitos disponíveis, dentre eles os públicos e privados. No *Broker* privado somente dispositivos definidos/permitidos podem publicar e assinar tópicos. Já o *Broker* público, que é o caso do *Mosquitto,* o qual foi utilizado neste trabalho, qualquer dispositivo pode publicar e assinar tópicos, não há privacidade.

Para a instalação do *Mosquitto* é necessário executar o comando:

pi@raspberrypi~ $ sudo apt-get install mosquitto

Após sua instalação, para colocá-lo em funcionamento *background* é preciso executar o seguinte comando:

pi@raspberrypi~ $ mosquitto -d

Para testar o funcionamento do *Broker Mosquitto* é possível se inscrever em um tópico:

pi@raspberrypi~ $ mosquito_sub -d -t topicoTeste

E, em seguida com outra linha de comando publicar uma mensagem nesse tópico:

pi@raspberrypi~ $ mosquito_pub -d -t topicoTeste -m "Testando o *Broker Mosquitto.*"

Se tudo estiver correto com a instalação e comandos, será recebido a mensagem "Testando o *Broker Mosquitto.*" no monitoramento do tópico inscrito.

3.2.6 Bootstrap

Para uma página *web* proporcionar uma boa experiência visual ao usuário (*front-end*), é necessário utilizar diversas ferramentas e tecnologias no seu desenvolvimento, o HTML, CSS e JavaScript são as principais tecnologias para esse desenvolvimento. O HTML é a linguagem responsável por estruturar a página *web*, é ele quem cria o esqueleto do site. O CSS é a linguagem para adicionar elementos estéticos a página *web*, como cores, fontes, espaçamentos e assim por diante, em conjunto com o HTML eles formam uma página *web* do tipo estática. Para adicionar interatividade ao site é necessário utilizar a linguagem de programação JavaScript, ele permite criar diversas funcionalidades que dão movimento a página, como animações, janelas, menus deslizantes, atualização automática de valores sem a necessidade de recarregar a página, entre outras.

Utilizar HTML, CSS e JavaScript é o suficiente para criar uma página *web*, porém, a codificação de um site utilizando essas três linguagens é trabalhosa e repetitiva, pois é necessário avaliar inúmeras situações para que a página fique bonita como, cores, fontes, margens e menus, o que torna o desenvolvimento cansativo. Para facilitar essa tarefa será utilizado o *framework* Bootstrap.

O Bootstrap é uma ferramenta para o desenvolvimento do *front-end* de uma página *web*, ele fornece uma estrutura completa de código aberto HTML, CSS e JavaScript para criação de sites e aplicativos *web* responsivos. Com esse framework é possível criar páginas *web* de forma mais fácil a partir de componentes referência já prontos, como botões, menus, formulários, gráficos, *templates*, etc.

O *framework* Bootstrap foi utilizado na criação das páginas *web* do presente trabalho para facilitar o desenvolvimento e também proporcionar uma interface gráfica responsiva, que se adapte ao tamanho da tela do dispositivo em que está sendo utilizado, seja ele celular, tablet ou computador, por exemplo.

Para utilizar os componentes e funcionalidades do Bootstrap é necessário adicionar as suas dependências na criação da página *web*, as dependências/bibliotecas podem estar

hospedadas em servidores remotos ou então localmente no próprio servidor, que é o caso do presente trabalho. Dessa forma, o funcionamento da interface *web* independe da conexão com a internet, se acaso a residência perder a conexão com a internet, os componentes e funcionalidades da interface *web* continuam funcionando normalmente.

Para adicionar as dependências no servidor local foi necessário acessar os sites onde as bibliotecas estavam hospedadas e copiá-las para o servidor local. Na criação de cada página *web* também é necessário apontar o local onde cada biblioteca utilizada na criação dos componentes está armazenada, como exibido no início do código do Apêndice A.

3.2.7 Páginas e servidor web

As funcionalidades do presente trabalho foram divididas em quatro menus, sendo eles: Principal, Agendamentos, Gráficos e Leituras. Cada menu tem suas respectivas páginas *web*.

A página Principal permite ligar e desligar os dispositivos de cada cômodo da residência através de botões animados, o envio desses comandos ao servidor é realizado através do método *post.* No lado do servidor esses comandos são recebidos e processados pela lógica condicional para verificar qual dispositivo ESP8266 deve receber o comando e atuar.

A página Principal também permite visualizar a leitura em tempo real dos sensores instalados em cada um dos cômodos, essa informação é atualizada automaticamente por um Socket.IO a cada nova leitura do sensor, sem a necessidade de recarregar a página. A leitura dos sensores é realizada pelo ESP8266 a cada determinado intervalo de tempo e enviada ao servidor *web*, o servidor recebe a informação, processa e a envia para o Socket.IO da página *web*, a leitura também é armazenada em uma tabela do banco de dados para posterior consulta. O Apêndice A exibe o código do *front-end* da página *web* Principal e o Apêndice B exibe o código do lado do servidor *(back-end)*.

A página de Agendamentos permite agendar a execução de uma tarefa para qualquer um dos dispositivos da residência, é possível agendar uma única execução ou execuções periódicas. As informações do agendamento são enviadas ao servidor *web* através da criação de formulários HTML utilizando o método *post.* No lado do servidor as informações do agendamento são recebidas e inseridas em uma tabela do banco de dados, juntamente é criado o novo agendamento no APScheduler. O APScheduler é uma biblioteca *Python* que permite agendar execuções de tarefas, apenas uma vez ou periodicamente, neste trabalho ele tem a função de executar a tarefa agendada quando a data e hora forem alcançadas. A tabela de

agendamentos salva no banco de dados permite consultar os agendamentos periódicos e excluí-los, se necessário.

Na página de Gráficos é possível visualizar de forma gráfica as leituras do sensor de temperatura e umidade da residência. Os dados exibidos no gráfico são resultado de uma consulta *SELECT* que busca as leituras dos últimos dois dias ordenado por hora, da menor para a maior.

A página de Leituras permite consultar as leituras de temperatura e umidade armazenadas no banco de dados. As leituras exibidas na página *web* são o retorno de um comando *SELECT* que consulta os últimos 1500 registros, ordenados do mais recente para o mais antigo.

3.2.8 Aplicativo Android

O aplicativo *Android* tem a função de facilitar o acesso ao sistema de automação residencial criado neste trabalho, ele aponta a url de acesso à página *web* conforme a rede que o usuário está conectado. Também é responsável por realizar requisições periódicas ao servidor *web* para atualizar a localidade do usuário e ativar/desativar o modo de operação automático.

Na proposta inicial deste trabalho, a localização do usuário seria verificada utilizando a coordenada GPS do *smartphone* e informado ao servidor *web* se o usuário está dentro da região da residência ou não, mas conforme explicado na seção 3.2.2 Modos de Operação, esse método se mostrou pouco eficiente e foi necessário modificá-lo. Agora, a verificação se o usuário está ou não na residência é feita pelo aplicativo *Android* através de uma requisição http ao servidor *web,* utilizando a biblioteca *Volley* do *Android*.

A requisição http sempre é realizada primeiramente apontando para a url do servidor local, essa url é uma rota criada dentro do servidor *web* que retorna a palavra “ControlHome” ao aplicativo, se o resultado da requisição http a url local for bem sucedida e a palavra retornada for correta, o usuário está conectado à rede local (WLAN) e o servidor *web* passa a operar em modo manual. Se a requisição local for mal sucedida ou a resposta é incorreta, o aplicativo aponta para a url remota (endereço *web*) e realiza uma segunda requisição, quando o servidor *web* recebe a requisição da rede remota (WAN), ele identifica pelo endereço IP do requisitante que a conexão não é local, indicando que o usuário não está na residência, então, o servidor passa a operar em modo automático. As requisições ao servidor são periódicas e ocorrem a cada um minuto, mesmo com o aplicativo operando em segundo plano, isso garante que o servidor seja atualizado frequentemente sobre o modo de operação que deve ativar.

A interface do usuário no aplicativo é exatamente as páginas *web* explicadas na seção anterior. O aplicativo foi desenvolvido com o componente *WebView*, que é uma ferramenta do *Android* para exibir conteúdos *web* dentro de aplicativos. Quando o usuário abre o aplicativo, o componente *WebView* carrega o conteúdo da url especificada. A url que o *WebView* deve carregar depende da rede que o usuário está conectado, seguindo o mesmo método explicado nos parágrafos anteriores, a url é definida a partir da resposta da requisição http local, ou seja, se a requisição local for bem sucedida, a url local será carregada no *WebView*, se a requisição local for mal sucedida, a url remota será carregada no *WebView*. Sempre que o aplicativo é aberto, uma nova requisição http é executada e retorna a página *web* Principal a tela. O Apêndice C exibe o código-fonte do aplicativo *Android*.

3.2.9 ESP8266

O ESP8266 tem a função de adquirir as leituras do sensor e enviá-las ao servidor, e também receber os comandos do servidor e executá-los no atuador (relé). Neste capítulo será explicado como o código do ESP8266 foi desenvolvido para realizar tais funções.

Para que o ESP8266 comunique com o servidor *web* via rede *wi-fi* foi utilizado a biblioteca “ESP8266WiFi.h” disponível no repositório da IDE do Arduino, para ele se conectar a rede *wi-fi* da residência foi necessário inserir as credenciais da *wi-fi* no código-fonte, a partir disso, quando o ESP8266 for energizado ele sempre procurará pelo ssid da *wi-fi* para se conectar.

Após estabelecido o meio físico de comunicação, foi necessário incluir a biblioteca “PubSubClient.h” que implementa um *client* do protocolo de comunicação MQTT, esse protocolo é utilizado para trocar mensagens leves entre o servidor *web* e os módulos ESP8266, conforme já explicado no capítulo anterior. Para que o ESP8266 se inscreva e/ou publique nos tópicos é preciso definir o IP servidor de mensagens MQTT (*broker*) no código fonte do ESP8266, nesse caso, o IP do *broker* MQTT é o mesmo IP do servidor *web,* já que ambas ferramentas estão instaladas dentro da *Raspberry Pi* e seu IP é fixo.

Para realizar a leitura da temperatura e da umidade foi conectado o sensor DHT22 ao ESP8266, conforme a ligação elétrica já exemplificada no capítulo anterior, também foi adicionado a biblioteca “DHT.h” e definido os pinos de conexão no código. O atuador relé também foi conectado ao módulo do ESP8266 conforme conexão elétrica já exemplificada anteriormente e definido os pinos de conexão no código fonte.

Após realizado os passos anteriores foi possível adquirir as leituras do sensor e publicá-las no tópico pré-definido, o servidor *web* inscrito nesse mesmo tópico consegue receber os dados e processá-los. No fluxo contrário, os comandos de atuação recebidos do usuário via página *web* são publicados pelo servidor em um tópico pré-definido e o ESP8266 inscrito nesse mesmo tópico consegue ler os dados e atuar conforme a necessidade. As leituras do sensor de temperatura e umidade são adquiridas e enviadas ao servidor a cada minuto ou a cada necessidade de atualização da página *web.* O Apêndice D exibe o código-fonte do ESP8266.

3.2.10 Código-fonte

No endereço <https://github.com/jeanfuchter/ControlHome> é possível acessar o código-fonte desenvolvido neste projeto.

4 RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados os resultados e discussões obtidos no desenvolvimento do presente trabalho. Primeiramente será explicado e exibido as páginas *web* em funcionamento conjunto com o aplicativo *Android* e na sequência é abordado sobre os sensores e atuadores, por fim, as discussões.

4.1 Visão geral

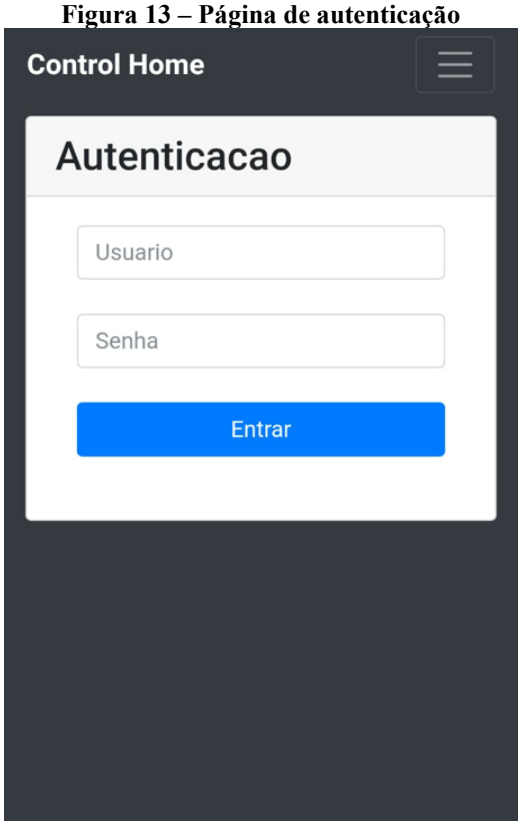
Neste trabalho foi desenvolvido uma automação de periféricos residenciais com base no conceito Domótica, que preconiza a gestão de todos os recursos habitacionais de forma integrada a fim de simplificar e centralizar o controle de todos os dispositivos em apenas um só sistema.

4.2 Página web e aplicativo Android

Ao abrir o aplicativo *Android* para acessar o sistema, se o usuário estiver com seu dispositivo na rede remota (WAN), a página de autenticação será exibida, conforme mostra a Figura 11. Ao informar as credenciais de acesso corretas o usuário será redirecionado para página Principal com total acesso às funcionalidades do sistema, do contrário, uma mensagem de erro será exibida e o usuário permanece na tela de autenticação. Se o usuário estiver acessando o sistema diretamente da rede local (LAN), a página Principal será exibida dispensando a autenticação. O modo de navegação local ou remoto é exibido no canto superior esquerdo da página para indicar a rede que o usuário se encontra, conforme é possível visualizar na Figura 12.

Como a página *web* fica exposta na internet, sendo necessário apenas o link para acessá-la, foi implementado o sistema de autenticação para evitar que usuários não autorizados tenham o controle do sistema.

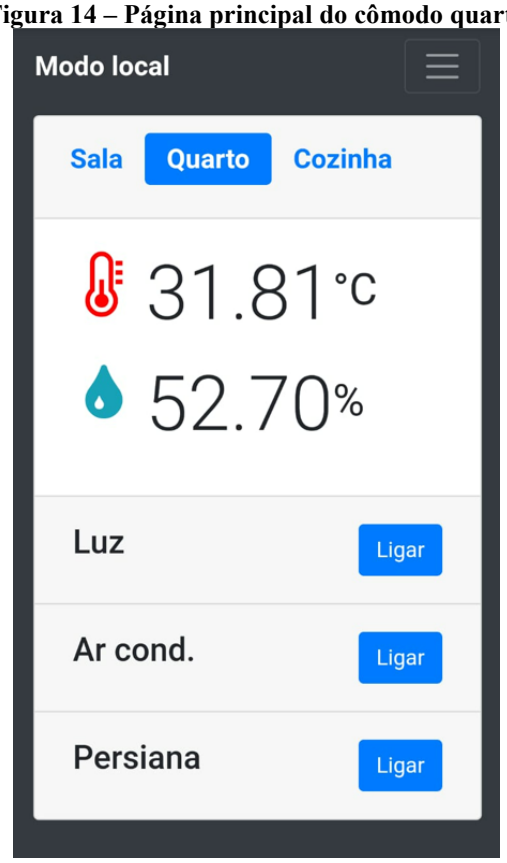
Figura 13 – Página de autenticação



Fonte: Autoria própria.

A página Principal exibe as informações temporais de cada cômodo da residência, como temperatura e umidade. Juntamente com as informações do cômodo estão as ações que o usuário pode executar de forma manual e instantânea para cada dispositivo daquele ambiente. A Figura 12 exibe a página Principal, nela é possível escolher o cômodo que deseja visualizar, cada cômodo tem seus dispositivos que podem ser ligados ou desligados manualmente a qualquer momento, as leituras de temperatura e umidade são atualizadas a cada minuto ou a cada atualização da página.

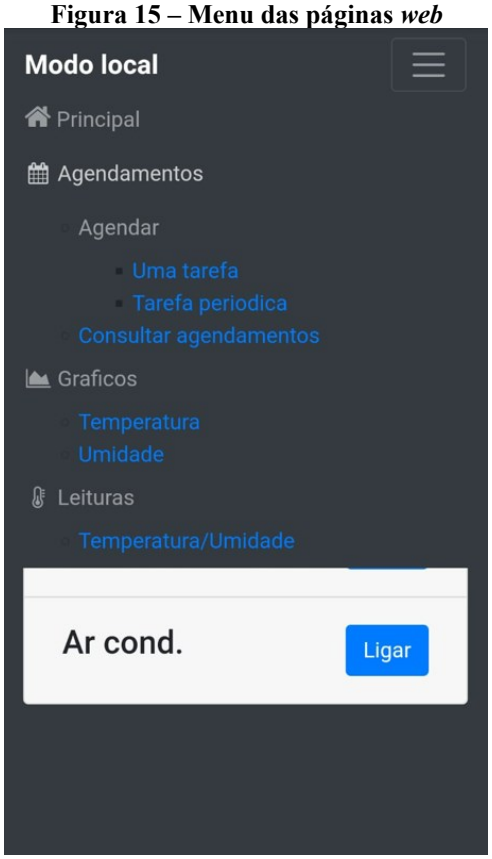
Figura 14 – Página principal do cômodo quarto



Fonte: Autoria própria.

Clicando no menu no canto superior direito é possível alternar entre as páginas do sistema, conforme exibe a Figura 13.

Figura 15 – Menu das páginas *web*

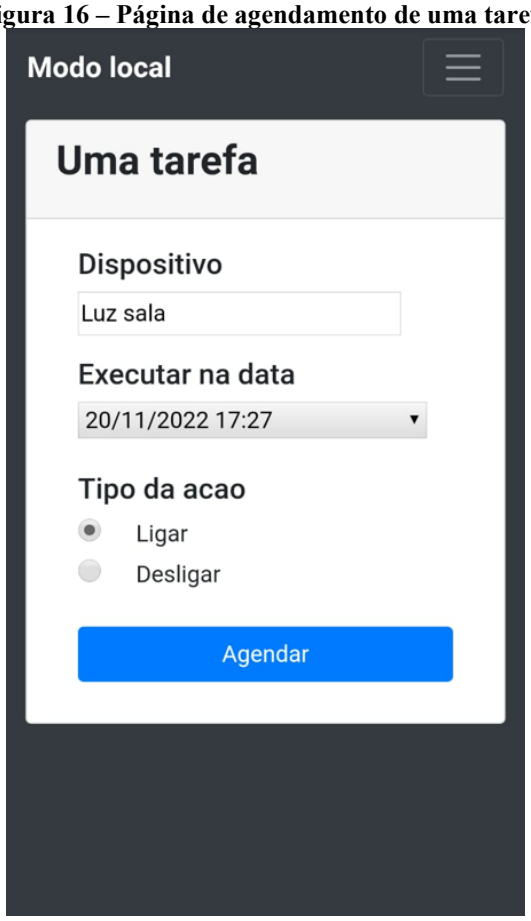


Fonte: Autoria própria.

Na seção Agendamentos é possível agendar uma tarefa para ser executada uma única vez, ou é possível agendar tarefas periódicas, também é possível consultar e excluir as tarefas periódicas agendadas na opção “Consultar agendamentos”.

Na opção “Uma tarefa” do submenu Agendar da seção Agendamentos é possível realizar o agendamento de tarefas de execução única, conforme exibe a Figura 14. Essa tela possibilita ao usuário escolher o dispositivo que deve atuar, a data e hora da execução e o tipo da ação. Ao clicar em agendar, se todos os preenchimentos da tela estiverem corretos, uma mensagem de agendamento realizado com sucesso será retornada.

Figura 16 – Página de agendamento de uma tarefa



Fonte: Autoria própria.

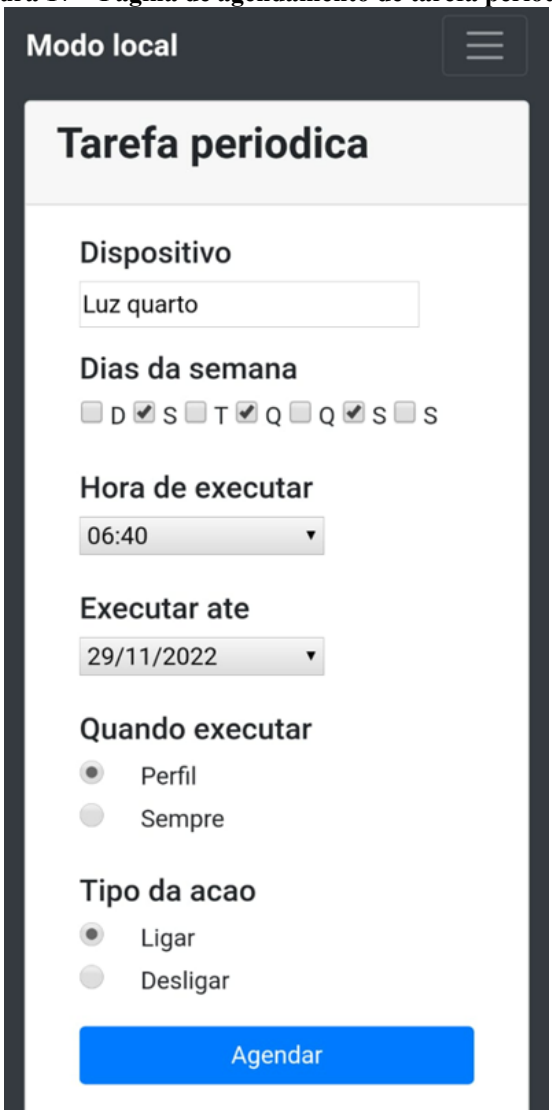
Na opção “Tarefa periódica” do Submenu Agendar da seção de Agendamentos é possível realizar o agendamento de tarefas periódicas, essas tarefas podem executar quantas vezes for necessário conforme as escolhas do usuário. A tela exibida na Figura 15 possibilita ao usuário escolher o dispositivo que deve atuar, os dias da semana que a ação deve ser executada, a hora de executar, o tipo da ação, e até quando o agendamento deve permanecer ativo.

Na opção “Quando executar” é possível escolher entre “Perfil” e “Sempre”, se a tarefa agendada for do tipo “Perfil”, o sistema vai executar a ação somente se o usuário estiver fora da rede local da residência, ou seja, somente se o sistema estiver operando em modo automático, conforme explicado no capítulo anterior. Um exemplo para esse tipo de tarefa é simular a presença do usuário na residência, onde é possível acionar as luzes de diferentes cômodos em variados horários e passar a impressão que tem pessoas na casa.

Se a tarefa agendada for do tipo “Sempre”, o sistema vai executar a tarefa indiferente da localidade do usuário, ou seja, nesse caso o sistema desconsidera o modo de operação manual

ou automático, a tarefa será sempre executada. Irrigar o jardim é um exemplo de tarefa que deve ser sempre executada indiferente da localidade do usuário.

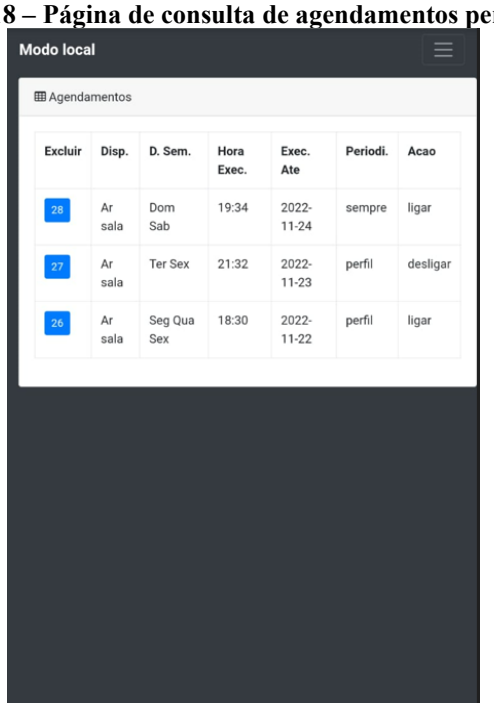
Figura 17 – Página de agendamento de tarefa periódica



Fonte: Autoria própria.

A opção “Consultar agendamentos” da seção Agendamentos possibilita ao usuário consultar todos os agendamentos periódicos realizados, também é possível excluir os agendamentos periódicos a qualquer momento clicando no botão azul da respectiva linha. A Figura 16 exibe a tela de consulta com três agendamentos periódicos ativos.

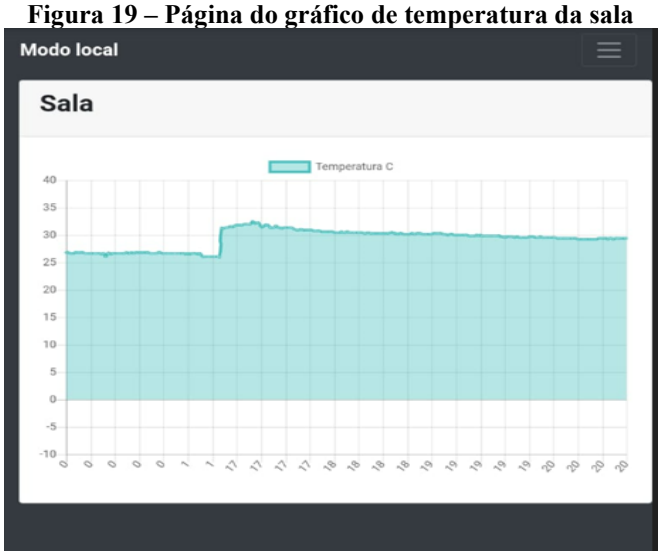
Figura 18 – Página de consulta de agendamentos periódicos



Fonte: Autoria própria.

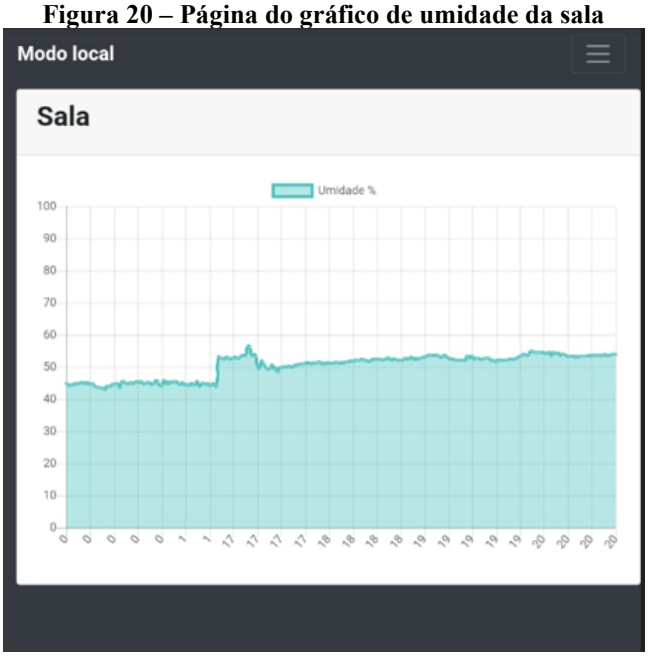
A seção Gráficos permite visualizar os gráficos de temperatura e de umidade originados das leituras do sensor, que são armazenadas no banco de dados do servidor *web.* A Figura 17 exibe página *web* com o gráfico de temperatura e a Figura 18 exibe a página *web* com o gráfico de umidade da sala, as leituras exibidas nos gráficos são do dia corrente. O degrau exibido na curva de temperatura e na curva de umidade de ambos os gráficos se deve a paralisação das leituras do sensor entre as 01h00 e 17h00 do dia corrente, ou seja, nesse intervalo de tempo houve a variação da temperatura e umidade do ambiente, mas não houve leituras do sensor porque o sistema encontrava-se desligado.

Figura 19 – Página do gráfico de temperatura da sala



Fonte: Autoria própria.

Figura 20 – Página do gráfico de umidade da sala



Fonte: Autoria própria.

Na seção Leituras a opção Temperatura/Umidade permite ao usuário verificar as últimas 1500 leituras do sensor de temperatura e umidade da sala. Conforme exibe a Figura 19, a tabela possibilita visualizar os valores de temperatura, umidade, data e hora que a leitura foi obtida.

Figura 21 – Página das leituras de temperatura e umidade da sala

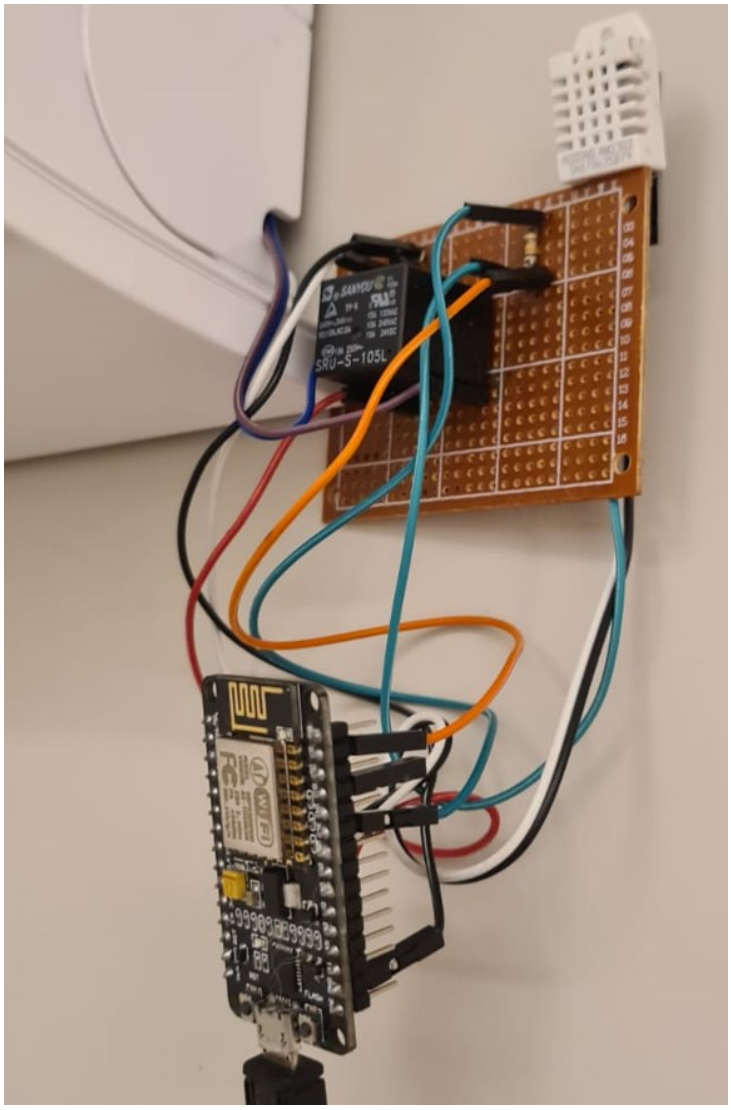
Fonte: Autoria própria.

Todas as páginas *web* exibidas e explicadas neste capítulo são carregadas diretamente dentro do aplicativo *Android* para evitar que o usuário tenha que digitar ou carregar a url no navegador de internet toda vez que deseja acessar o sistema. Desta forma, o usuário só necessita abrir o aplicativo, o direcionamento para a url correta é realizado de maneira automática. Porém, o acesso ao sistema não fica restrito a dispositivos com aplicativo *Android*, o usuário pode acessá-lo em qualquer dispositivo que tenha conexão com a rede local ou com a internet e que tenha um navegador *web*, sendo necessário apenas colocar a url local ou url remota no navegador e carregá-la.

4.3 Sensores e atuadores

Neste trabalho foi utilizado o módulo DHT22 para adquirir temperatura e a umidade da residência. O circuito de ligação elétrica é simples conforme já explicado no capítulo anterior e ilustrado na Figura 5, sendo necessário apenas um resistor de *pullup* ligado a alimentação e a conexão elétrica com o ESP8266-ESP-12E. Para a montagem do circuito foi utilizado uma placa perfurada de fenolite, cabos macho/fêmea e solda de estanho para fixar os componentes na placa. Como a versão 12E do ESP8266 tem várias entradas e saídas digitais, foi utilizado o mesmo módulo e a mesma placa de fenolite para a montagem da ligação elétrica do atuador também. O circuito de ligação elétrica do atuador também é simples conforme já foi explicado no capítulo anterior e ilustrado na Figura 6, ele é composto por relé, transistor, resistor, diodo, e cabos macho fêmea. A Figura 20 exibe os dois circuitos montados, sendo que o atuador (relé) está conectado ao ar condicionado da residência.

Figura 22 – Circuito de leitura do sensor DHT22 e acionamento do ar condicionado



Fonte: Autoria própria.

Como exemplo de aplicação prática, foi utilizado o circuito de acionamento para ligar e desligar o ar condicionado da minha própria residência remotamente, para proporcionar um conforto térmico ao meu cachorro em dias de temperaturas elevadas em que não havia pessoas na casa. O acionamento remoto do ar condicionado era realizado manualmente a partir da temperatura lida pelo sensor, ou seja, o usuário precisa entrar na página *web* e verificar a temperatura do ambiente e realizar o acionamento do ar condicionado, se necessário.

Como melhoria, essa tarefa poderia ser realizada de maneira automática pelo sistema, ou seja, o ar condicionado poderia ser acionado automaticamente pelo sistema a partir de uma temperatura pré-configurada pelo usuário na página *web.* De forma mais genérica, qualquer atuador disponível no sistema poderia ser acionado a partir de um evento configurado

pelo usuário. Neste trabalho, o acionamento de um atuador é possível de ser realizado de maneira manual ou configurando uma data e hora de execução.

No presente trabalho foi utilizado somente um sensor de temperatura e umidade e um atuador relé para exemplificar o funcionamento do sistema, mas há possibilidade de utilizar inúmeros tipos de sensores que podem ser úteis no dia a dia, como o sensor de luminosidade, gás, presença, umidade do solo, consumo de energia elétrica e assim por diante. Nesse mesmo viés, também poderia ser utilizado outros tipos e atuadores. Porém, para a adição de novos sensores e atuadores neste trabalho, é necessário a modificação do código-fonte e a configuração de novos dispositivos ESP8266 para se comunicar com o servidor e com os sensores e atuadores. Poderia ser desenvolvido melhorias no sistema para deixá-lo genérico, onde o usuário possa cadastrar novos dispositivos de maneira simples e sem a necessidade de modificação do código-fonte. Também, poderia possibilitar categorizar por ambiente/cômodo os novos dispositivos, ou seja, cada novo dispositivo inserido no sistema, o usuário pode definir a qual cômodo da residência ele pertence de maneira a organizar a exibição dos dispositivos por ambiente.

5 CONCLUSÃO

Neste trabalho foi demonstrado que a automação residencial juntamente com as presentes tecnologias pode ajudar em várias tarefas do dia a dia de uma residência e proporcionar mais conforto, praticidade e descanso aos seus usuários. Com o uso da comunicação sem fio entre os dispositivos, esse sistema pode ser aplicado em qualquer moradia sem a necessidade da passagem de cabos por toda a casa, reduzindo os custos e incômodos com instalações e reformas.

O agendamento de tarefas periódicas em conjunto com o modo automático de funcionamento, permite que o sistema controle os recursos habitacionais quando o usuário não está em casa e evita que ele precise se preocupar com a execução desses comandos. Ele também proporciona economia de recursos naturais como água e energia elétrica, já que as tarefas vão ser executadas somente o tempo necessário e após serão finalizadas. Ainda, o agendamento possibilita criar tarefas para ligar e desligar as luzes da residência para simular que há pessoas em casa e proporcionar um aumento na segurança.

O aplicativo *Android* oferece uma maior facilidade de acesso à interface de usuário, porém, o acesso via página *web* torna o sistema multiplataforma e permite que o usuário acesse a interface de qualquer lugar do mundo com qualquer dispositivo que tenha um navegador *web* e conexão com a internet, não ficando restrito somente a plataforma *Android.*

A continuidade no desenvolvimento deste trabalho pode proporcionar melhorias que permitam o usuário adicionar novos dispositivos ESP8266 e consequentemente novos tipos sensores e atuadores de maneira simplificada, sem a necessidade de mexer no código-fonte, juntamente com isso pode ser implementado a opção para cadastrar novos ambientes na residência e possibilitar organização dos novos dispositivos por cômodos. O sistema de execução de tarefas também pode ser melhorado para permitir outras possibilidades de disparo além do já existente agendamento por data e hora, pode ser adicionado a possibilidade da leitura de um sensor disparar a execução de um atuador, isso torna o sistema mais versátil e adiciona inúmeras possibilidades para execução de tarefas cadastradas pelo usuário.

REFERÊNCIAS

BANZI, Massimo; SHILOH, Michael. **Primeiros Passos com o Arduino**: A plataforma de prototipagem eletrônica open source. 2.ed. São Paulo: Novatec Editora Ltda. 2015. Disponível em: <https://books.google.com.br/books?hl=pt-BR&lr=&id=otfECQAAQBAJ&oi=fnd&pg=PA21&dq=atuadores+s%C3%A3o&ots=rAVzM7edvd&sig=gJMBsO4ohnCWAssQI1stqHiSEcY#v=onepage&q=atuadores&f=false>. Acesso em: 15 nov. 2016.

BEGHINI, Lucas B. **Automação residencial de baixo custo por meio de dispositivos moveis com sistema operacional Android.** 2013. 76 f. Monografia de Trabalho de Conclusão de Curso– Curso Superior de Engenharia Elétrica com ênfase em Eletrônica. Universidade de São Paulo, São Carlos. São Paulo, 2013. Disponível em: < http://www.tcc.sc.usp.br/tce/disponiveis/18/180450/tce-04022014-152853/?&lang=br>. Acesso em: 22 out. 2016.

BOLZANI, Caio A. M. Desmistificando a Domótica. **I Semana da Automação do Cefet-**SP. São Paulo, out. 2006. Disponível em: <http://www.bolzani.com.br/artigos/art01_07.pdf>. Acesso em: 15 out. 2016.

BORGES, L. P.; Dores, R. C. **Automação predial sem fio utilizando bacnet/zigbee com foco em economia de energia**. 2010, 76f. Trabalho de conclusão de curso – Curso de Graduação em Engenharia de Controle e Automação – UNB, Brasília, 2010.

CRISTO, Fernando de; FRANCISCATTO, Roberto; PERLIN, Tiago. **Redes de Computadores**. 2014. 116 f. Disponível em: <https://drive.google.com/file/d/1fqTnDzmtEyTVW8uE6cH6LJPl2PKagvU8/view>. Acesso em: 20 nov. 2022.

DANTAS, Mario. **Tecnologias de Redes de Comunicação e Computadores**. Rio do Sul: Axcel Books, 2002.

GANDHEWAR, Nisarg; SHEIKH, Rahila. **Google Android**: An Emerging Software Platform For Mobile Devices. In: / International Journal on Computer Science and Engineering, Chandrapur, 2010, p.1-17. Disponível em: <http://www.enggjournals.com/ijcse/doc/003-IJCSESP24.pdf>. Acesso em: 01 nov. 2016.

GLOBALSTATS, StatCounter. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-202111-202211>. Acesso em: 04 nov. 2022.

LONGO, Lucas. **Internet das Coisas: uso de sensores e atuadores na automação de um protótipo residencial.** 2015. 100 f. Monografia de Trabalho de Conclusão de Curso– Curso Superior de Engenharia de Computação. Universidade Tecnológica Federal do Paraná, Pato Branco, 2015.

PEREIRA, Eduardo H. H. **Soluções inteligentes e de baixo custo pra a automação residencial utilizando smartphones.** 2014. 67 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Engenharia Elétrica com ênfase em Sistemas de Energia e Automação. Universidade de São Paulo, São Carlos, 2014. Disponível em:<http://www.tcc.sc.usp.br/tce/disponiveis/18/180500/tce-13012015-162609/?&lang=br>. Acesso em 26 out. 2016.

PEREIRA, Fagner de A. **O mundo dos microcontroladores.** 2016**.** Disponível em: <https://app.box.com/s/kexzcv4fvrq3p913yuni>. Acesso em: 01 dez. 2016.

PESSÔA, Marcelo S. de P.; SPINOLA, Mauro de M. **Introdução à automação para cursos de engenharia e gestão**. CIDADE: Elsevier, 2014.Disponível em: <https://books.google.com.br/books?id=YU0aBQAAQBAJ&pg=SA7-PA11&lpg=SA7-PA11&focus=viewport&dq=o+que+s%C3%A3o+atuadores&hl=pt-BR#v=onepage&q=atuadores&f=false>. Acesso em: 02 nov. 2016.

PINHEIRO, José Mauricio S. Domótica. Projeto de Redes, 21 out. 2015. Disponível em: <http://www.projetoderedes.com.br/artigos/artigo-domotica.php>. Acesso em: 03 nov. 2016.

PREDKO, Myke. **Programming and Customizing the PIC® Microcontroller**. 3. ed. New York: Mcgraw-hill Companies, Inc., 2008. 1263 p. Disponível em: <http://enggate.net/content/uploads/2015/07/programming-and-customizing-pic-microcontroller.pdf>. Acesso em: 02 nov. 2016.

SILVA JUNIOR, Vidal P. da. **Microcontroladores.** São Paulo: Érica, 1998.

SOUSA, Lindeberg Barros de. **Redes de Computadores**: dados, voz e imagem. São Paulo: Érica, 1999.

SUI, Linda. Android Capture Record 88 Percent Share of Global Smartphone Shipments in Q3 2016. Strategy Analytics, 02 nov. 2016. Disponível em: <https://www.strategyanalytics.com/strategy-analytics/blogs/devices/smartphones/smart-phones/2016/11/02/android-captures-record-88-percent-share-of-global-smartphone-shipments-in-q3-2016?slid=68060&spg=3#.WCHSofkrK00>. Acesso em: 03 nov. 2016.

VOLPATO, Luan Cesar S. **Sistema de segurança residencial integrado com aplicativo para smartphone.** 2012. 78 f. Trabalho de Conclusão de Curso (Graduação) – Curso de Engenharia de Controle e Automação. Universidade Federal de Santa Catarina, Florianópolis, 2012. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/166363/PFC-20121-LuanCesarSouzaVolpato.pdf?sequence=1&isAllowed=y>. Acesso em: 20 out. 2016.

APÊNDICE A – Código da página web da tela Principal (*front-end*)

CÓDIGO DA PÁGINA *WEB* DA TELA PRINCIPAL (*FRONT-END*)

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Control Home</title>
  <!-- Bootstrap core CSS-->
  <link href="static/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
  <!-- Custom fonts for this template-->
  <link href="static/vendor/font-awesome/css/font-awesome.min.css"
   rel="stylesheet" type="text/css">
  <!-- Page level plugin CSS-->
  <link href="static/vendor/datatables/dataTables.bootstrap4.css" rel="stylesheet">
  <!-- Custom styles for this template-->
  <link href="static/css/sb-admin.css" rel="stylesheet">
  <script src="static/vendor/jquery/jquery-3.1.1.min.js"></script>
  <script type="text/javascript" src="static/vendor/ajax/socket.io.min.js"></script>
  <script type="text/javascript" charset="utf-8">
      $(document).ready(function()
      {
           $('#currentTemperature').text('22.50').html();
           $('#currentHumidity').text('81.00').html();
           var socket = io.connect('http://' + document.domain + ':' + location.port);
           socket.on('connect', function()
           {
               socket.emit('my event', {data: 'Connected!'});
           });
           socket.on('dht_temperature', function(msg)
           {
             var nDate = new Date();
             $('#currentTemperature').text(msg.data).html();
           });
           socket.on('dht_humidity', function(msg)
           {
             var nDate = new Date();
            $('#currentHumidity').text(msg.data).html();
           });
      });
   </script>
</head>

<body class="fixed-nav sticky-footer bg-dark" id="page-top">
  <!-- Navigation-->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top" id="mainNav">
    {% if userRemote == 'true' %}
        <a class="navbar-brand font-weight-bold" href="/">Modo remoto</a>
    {% else %}
        <a class="navbar-brand font-weight-bold" href="/">Modo local</a>
    {% endif %}
    <button class="navbar-toggler navbar-toggler-right" type="button" data-toggle="collapse"
      data-target="#navbarResponsive" aria-controls="navbarResponsive"
      aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav navbar-sidenav" id="exampleAccordion">
        <li class="nav-item" data-toggle="tooltip" data-placement="right" title="Principal">
          <a class="nav-link" href="/">
            <i style="font-size: 16pt" class="fa fa-home"></i>
            <span class="nav-link-text">Principal</span>
          </a>
       </li>
       <li class="nav-item" data-toggle="tooltip" data-placement="right" title="Agendamentos">
        <a class="nav-link nav-link-collapse collapsed" data-toggle="collapse"
        href="#collapseComponents3" data-parent="#exampleAccordion2">
                <i class="fa fa-fw fa-calendar"></i>
                 <span class="nav-link-text">Agendamentos</span>
        </a>
```

```
      <ul class="sidenav-second-level collapse" id="collapseComponents3">
        <li class="nav-item" data-toggle="tooltip" data-placement="right"
        title="Agendamentos">
          <a  class="nav-link nav-link-collapse collapsed" data-toggle="collapse"
          href="#collapseComponents4" data-parent="#exampleAccordion4">
          <span class="nav-link-text">Agendar</span>
          </a>
          <ul class="sidenav-third-level collapse" id="collapseComponents4">
             <li>
             <a href="taskSchedulerOne"> Uma tarefa</a>
             </li>
             <li>
             <a href="taskSchedulerDays">Tarefa periodica </a>
             </li>
          </ul>
        </li>
        <li>
          <a href="schedules">Consultar agendamentos</a>
        </li>
       </ul>
       </li>
       <li class="nav-item" data-toggle="tooltip" data-placement="right" title="GrÃ¡ficos">
         <a class="nav-link nav-link-collapse collapsed"
         data-toggle="collapse" href="#collapseComponents1" data-parent="#exampleAccordion1">
           <i class="fa fa-fw fa-area-chart"></i>
           <span class="nav-link-text">Graficos</span>
         </a>
         <ul class="sidenav-second-level collapse" id="collapseComponents1">
           <li>
             <a href="chartOfTemperature">Temperatura</a>
           </li>
           <li>
             <a href="chartOfHumidity">Umidade</a>
           </li>
         </ul>
       </li>
       <li class="nav-item" data-toggle="tooltip" data-placement="right" title="Leituras">
         <a class="nav-link nav-link-collapse collapsed"
         data-toggle="collapse" href="#collapseComponents2" data-parent="#exampleAccordion">
           <i class="fa fa-fw fa-thermometer-half"></i>
           <span class="nav-link-text">Leituras</span>
         </a>
         <ul class="sidenav-second-level collapse" id="collapseComponents2">
           <li>
             <a href="readingsTH">Temperatura/Umidade</a>
           </li>
         </ul>
       </li>

    {% if logged == 'True' %}
             <li class="nav-item" data-toggle="tooltip"
             data-placement="right" title="Sair">
             <a class="nav-link" href="logout">
             <i class="fa fa-fw fa-user-o"></i>
             <span class="nav-link-text">Sair</span>
             </a>
             </li>
    {% endif %}

    </ul>
  </div>
</nav>

<div class="content-wrapper">
  <div class="container-fluid">
    <!-- Breadcrumbs-->
    <ol class="breadcrumb">
      <li class="breadcrumb-item">
        <a href="/">Home</a>
      </li>
      <li class="breadcrumb-item active">Principal</li>
    </ol>

       <!-- Title -->

  <!-- Text -->
```

```
    <div class="card" style="width:330px">
    <div class="card-header">
           <ul class="nav nav-pills card-header-pills">
          <li class="nav-item">
            <h5 class="card-title font-weight-bold" >
            <a class="nav-link active" href="/">Sala</a></h5>
          </li>
          <li class="nav-item">
            <h5 class="card-title font-weight-bold" >
            <a class="nav-link" href="bedroom">Quarto</a></h5>
          </li>
          <li class="nav-item">
            <h5 class="card-title font-weight-bold" >
            <a class="nav-link" href="kitchen">Cozinha</a></h5>
          </li>
        </ul>
    </div>
    <div class="card-body">

<div class="d-flex justify-content-left">
  <i style="color:red" class="fa fa-fw fa-3x pt-1 amber-text fa-thermometer-half"></i>
  <p class="display-4 degree pt-1"><span id="currentTemperature"></span></p><sup
  class="units pt-5 fa-2x"><span>&#176;</span>C </sup>
</div>

<div class="d-flex justify-content-left">
  <i class="fa fa-fw fa-3x text-info pt-1 amber-text fa-tint"></i>
  <p class="display-4 degree pt-1"><span id="currentHumidity"></span></p><sup
  class="units pt-5 fa-2x">%</sup>
</div>
</div>

<div class="card-footer">
    <form action="" method="post">
            {% if pins[4].state == 'True' %}
                    <div class="d-flex">
                    <div><h4 class="p-2">Luz</h4></div>
                    <div class="ml-auto p-2"><input class="btn btn-secondary"
                    type="submit" name="luzSala" value="Desligar"></div>
                    </div>
            {% else %}
                    <div class="d-flex">
                    <div><h4 class="p-2">Luz</h4></div>
                    <div class="ml-auto p-2"><input class="btn btn-primary"
                    type="submit" name="luzSala" value="Ligar"></div>
                    </div>
            {% endif %}
     </form>
 </div>
 <div class="card-footer">
    <form action="" method="post">
            {% if pins[5].state == 'True' %}
                    <div class="d-flex">
                    <div><h4 class="p-2">Ar cond.</h4></div>
                    <div class="ml-auto p-2"><input class="btn btn-secondary"
                    type="submit" name="arSala" value="Desligar"></div>
                    </div>
            {% else %}
                    <div class="d-flex">
                    <div><h4 class="p-2">Ar cond.</h4></div>
                    <div class="ml-auto p-2"><input class="btn btn-primary"
                    type="submit" name="arSala" value="Ligar"></div>
                    </div>
            {% endif %}
    </form>
  </div>
    </div>

<!-- Bootstrap core JavaScript-->
<script src="static/vendor/jquery/jquery.min.js"></script>
<script src="static/vendor/popper/popper.min.js"></script>
<script src="static/vendor/bootstrap/js/bootstrap.min.js"></script>
    <script src="static/js/weather.js"></script>
<!-- Core plugin JavaScript-->
<script src="static/vendor/jquery-easing/jquery.easing.min.js"></script>
<!-- Page level plugin JavaScript-->
```

```
    <script src="static/vendor/chart.js/Chart.min.js"></script>
    <script src="static/vendor/datatables/jquery.dataTables.js"></script>
    <script src="static/vendor/datatables/dataTables.bootstrap4.js"></script>
    <!-- Custom scripts for all pages-->
    <script src="static/js/sb-admin.min.js"></script>
    <!-- Custom scripts for this page-->
    <script src="static/js/sb-admin-datatables.min.js"></script>
    <script src="static/js/sb-admin-charts.min.js"></script>
  </div>
</body>

</html>
```

APÊNDICE B – Código do servidor *web* (*back-end*)

CÓDIGO DO SERVIDOR WEB (*BACK-END*)

```
import paho.mqtt.client as mqtt
from flask import Flask, jsonify, render_template, redirect, url_for, request, session
from functools import wraps
from flask_socketio import SocketIO, emit
from ipaddress import IPv4Address
import json
import sqlite3
from apscheduler.schedulers.background import BackgroundScheduler
from datetime import datetime

app = Flask(__name__)
app.config['SECRET_KEY'] = 'jean'
socketio = SocketIO(app)

#Login required
def login_required(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if (request.remote_addr and request.remote_addr != '127.0.0.1') or ('logged_in' in
session):
            return f(*args, **kwargs)
        else:
            return redirect(url_for('login'))
    return wrap

def dict_factory(cursor, row):
    d = {}
    for idx, col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d

def on_connect(client, userdata, flags, rc):
    print("Connected: "+str(rc))

    client.subscribe("/esp8266/temperature")
    client.subscribe("/esp8266/humidity")
    client.subscribe("/esp8266/dhtreadings")

# Waiting to receive callback msg ESP8266
def on_message(client, userdata, message):

    if message.topic == "/esp8266/dhtreadings":
        print("DHT readings update")
        dhtreadings_json = json.loads(message.payload)
        conn=sqlite3.connect('home-database.db')
        c=conn.cursor()
        c.execute("""INSERT INTO readingsTH (temperature,
            humidity, date, time, device) VALUES((?), (?),
date(CURRENT_TIMESTAMP,'localtime'),
            time(CURRENT_TIMESTAMP,'localtime'), (?))""", (dhtreadings_json['temperature'],
            dhtreadings_json['humidity'], 'esp8266_1') )
        conn.commit()
        conn.close()

    if message.topic == "/esp8266/temperature":
        print("temperature update")
        socketio.emit('dht_temperature', {'data': message.payload})
    if message.topic == "/esp8266/humidity":
        print("humidity update")
        socketio.emit('dht_humidity', {'data': message.payload})

mqttc=mqtt.Client()
mqttc.on_connect = on_connect
mqttc.on_message = on_message
mqttc.connect("localhost",1883,60)
mqttc.loop_start()

# Create dictionary of devices
pins = {
   1 : {'name' : 'Luz quarto',      'board' : 'esp8266', 'topic' : '/esp8266/1', 'state' :
'False'},
   2 : {'name' : 'Ar quarto',       'board' : 'esp8266', 'topic' : '/esp8266/2', 'state' :
'False'},
```

```
   3 : {'name' : 'Persiana quarto', 'board' : 'esp8266', 'topic' : '/esp8266/3', 'state' :
'False'},
   4 : {'name' : 'Luz sala',        'board' : 'esp8266', 'topic' : '/esp8266/4', 'state' :
'False'},
   5 : {'name' : 'Ar sala',         'board' : 'esp8266', 'topic' : '/esp8266/5', 'state' :
'False'},
   6 : {'name' : 'UpdateTH',        'board' : 'esp8266', 'topic' : '/esp8266/6', 'state' :
'False'},
   7 : {'name' : 'Luz cozinha',     'board' : 'esp8266', 'topic' : '/esp8266/7', 'state' :
'False'},
   8 : {'name' : 'Ar cozinha',      'board' : 'esp8266', 'topic' : '/esp8266/8', 'state' :
'False'}
       }

# Send dictionary to pins:
templateData = {
   'pins' : pins
   }

# Page login and password
@app.route('/login', methods=['GET', 'POST'])
def login():

    if (request.remote_addr and request.remote_addr != '127.0.0.1') or ('logged_in' in
session):
        return redirect(url_for('index'))
    else:
        error = None
        if request.method == 'POST':
            if request.form['username'] != 'jean' or request.form['password'] != 'jean':
                error = 'Usuario e senha invalidos, tente novamente.'
            else:
                session['logged_in'] = True
                return redirect(url_for('index'))
        return render_template('login.html', error=error)

@app.route('/logout')
def logout():
    session.pop('logged_in', None)
    return redirect(url_for('login'))

@app.route('/', methods=['GET', 'POST'])
@login_required
def index():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    #Show button logout
    if 'logged_in' in session:
        logged = 'True'
    else:
        logged = 'False'

    if request.method == 'POST':

        if request.form.get('luzSala'):
            if request.form['luzSala'] == 'Desligar':
                mqttc.publish(pins[4]['topic'],"0")
                pins[4]['state'] = 'False'
            elif request.form['luzSala'] == 'Ligar':
                mqttc.publish(pins[4]['topic'],"1")
                pins[4]['state'] = 'True'
        elif request.form.get('arSala'):
            if request.form['arSala'] == 'Desligar':
                mqttc.publish(pins[5]['topic'],"0")
                pins[5]['state'] = 'False'
            elif request.form['arSala'] == 'Ligar':
                mqttc.publish(pins[5]['topic'],"1")
                pins[5]['state'] = 'True'
    else:
        mqttc.publish('/esp8266/6',"0")
        #Force update temperature and humidity
```

```
    templateData = {
        'pins' : pins
    }

    return render_template('index.html', userRemote=userRemote, logged=logged,
async_mode=socketio.async_mode, **templateData)

@app.route('/bedroom', methods=['GET', 'POST'])
@login_required
def bedroom():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    #Show button logout
    if 'logged_in' in session:
        logged = 'True'
    else:
        logged = 'False'

    if request.method == 'POST':
        if request.form.get('luzQuarto'):
          if request.form['luzQuarto'] == 'Desligar':
               mqttc.publish(pins[1]['topic'],"0")
               pins[1]['state'] = 'False'
          elif request.form['luzQuarto'] == 'Ligar':
               mqttc.publish(pins[1]['topic'],"1")
               pins[1]['state'] = 'True'
        if request.form.get('arQuarto'):
          if request.form['arQuarto'] == 'Desligar':
               mqttc.publish(pins[2]['topic'],"0")
               pins[2]['state'] = 'False'
          elif request.form['arQuarto'] == 'Ligar':
               mqttc.publish(pins[2]['topic'],"1")
               pins[2]['state'] = 'True'
        if request.form.get('persianaQuarto'):
          if request.form['persianaQuarto'] == 'Desligar':
               mqttc.publish(pins[3]['topic'],"0")
               pins[3]['state'] = 'False'
          elif request.form['persianaQuarto'] == 'Ligar':
               mqttc.publish(pins[3]['topic'],"1")
               pins[3]['state'] = 'True'
    else:
        mqttc.publish('/esp8266/6',"0")
        #Force update temperature and humidity

    templateData = {
        'pins' : pins
    }

    return render_template('bedroom.html', userRemote=userRemote, logged=logged,
async_mode=socketio.async_mode, **templateData)

@app.route('/kitchen', methods=['GET', 'POST'])
@login_required
def kitchen():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    #Show button logout
    if 'logged_in' in session:
        logged = 'True'
    else:
        logged = 'False'

    if request.method == 'POST':
        if request.form.get('luzCozinha'):
```

```
            if request.form['luzCozinha'] == 'Desligar':
                mqttc.publish(pins[7]['topic'],"0")
                pins[7]['state'] = 'False'
            elif request.form['luzCozinha'] == 'Ligar':
                mqttc.publish(pins[7]['topic'],"1")
                pins[7]['state'] = 'True'
        if request.form.get('arCozinha'):
            if request.form['arCozinha'] == 'Desligar':
                mqttc.publish(pins[8]['topic'],"0")
                pins[8]['state'] = 'False'
            elif request.form['arCozinha'] == 'Ligar':
                mqttc.publish(pins[8]['topic'],"1")
                pins[8]['state'] = 'True'
    else:
        mqttc.publish('/esp8266/6',"0")
        #Force update temperature and humidity

    templateData = {
        'pins' : pins
    }

    return render_template('kitchen.html', userRemote=userRemote, logged=logged,
async_mode=socketio.async_mode, **templateData)

# Function for on/off devices
def powerAir(whenRun, action, idDelete, device):

    global userRemote

    print('Quando: ' + whenRun)
    print('Acao: ' + action)
    print('Id: ' + idDelete)
    print('Dispositivo: ' + device)

    #Execute action
    if ((whenRun == 'perfil' and userRemote == 'true') or (whenRun == 'sempre')):
        if (action == 'ligar'):
            if (device == 'Luz quarto'):
                mqttc.publish(pins[1]['topic'],"1")
                pins[1]['state'] = 'True'
            elif (device == 'Ar quarto'):
                mqttc.publish(pins[2]['topic'],"1")
                pins[2]['state'] = 'True'
            elif (device == 'Persiana quarto'):
                mqttc.publish(pins[3]['topic'],"1")
                pins[3]['state'] = 'True'
            elif (device == 'Luz sala'):
                mqttc.publish(pins[4]['topic'],"1")
                pins[4]['state'] = 'True'
            elif (device == 'Ar sala'):
                mqttc.publish(pins[5]['topic'],"1")
                pins[5]['state'] = 'True'
        elif (action == 'desligar'):
            if (device == 'Luz quarto'):
                mqttc.publish(pins[1]['topic'],"0")
                pins[1]['state'] = 'False'
            elif (device == 'Ar quarto'):
                mqttc.publish(pins[2]['topic'],"0")
                pins[2]['state'] = 'False'
            elif (device == 'Persiana quarto'):
                mqttc.publish(pins[3]['topic'],"0")
                pins[3]['state'] = 'False'
            elif (device == 'Luz sala'):
                mqttc.publish(pins[4]['topic'],"0")
                pins[4]['state'] = 'False'
            elif (device == 'Ar sala'):
                mqttc.publish(pins[5]['topic'],"0")
                pins[5]['state'] = 'False'


    if idDelete != '' and idDelete != '0':

        #Delete base schedule
        idDelete = int(idDelete)
        conn=sqlite3.connect('home-database.db')
        c=conn.cursor()
        c.execute("""DELETE FROM schedules WHERE id = ?""", (idDelete,))
```

```
        conn.commit()
        conn.close()


    templateData = {
        'pins' : pins
    }


@app.route("/taskSchedulerOne",  methods=['GET', 'POST'])
@login_required
def taskSchedulerOne():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    msg = None
    action = None

    if request.method == 'POST':
       dateTimeHTML = request.form['dateHourStart']
       device = request.form['devices']

       if dateTimeHTML != '' and device != '':
           dateTimeRun = dateTimeHTML[0:10] + ' ' + dateTimeHTML[11:16] + ':00'
           action = request.form['action']

           global sched
           sched = BackgroundScheduler(daemon=True)
           sched.add_job(powerAir,'date', run_date=dateTimeRun, args=['sempre', action, '0',
device])
           sched.start()
           msg = 'Agendamento realizado com sucesso.'
       else:
            msg = 'Nao foi possivel realizar o agendamento.'

    return render_template('taskSchedulerOne.html', msg=msg, userRemote=userRemote)

@app.route("/taskSchedulerDays",  methods=['GET', 'POST'])
@login_required
def taskSchedulerDays():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    msg = None
    whenRun = None
    action = None

    if request.method == 'POST':
       TimeStart = request.form['hourStart']
       EndDate = request.form['endDate']
       device = request.form['devices']
       DaysOfWeek = ''
       DiasSemana = ''
       if request.form.get('sun'):
           DaysOfWeek = 'sun'
           DiasSemana = 'Dom'
       if request.form.get('mon'):
           DaysOfWeek = DaysOfWeek + ' mon'
           DiasSemana = DiasSemana + ' Seg'
       if request.form.get('tue'):
           DaysOfWeek = DaysOfWeek + ' tue'
           DiasSemana = DiasSemana + ' Ter'
       if request.form.get('wed'):
           DaysOfWeek = DaysOfWeek + ' wed'
           DiasSemana = DiasSemana + ' Qua'
       if request.form.get('thu'):
           DaysOfWeek = DaysOfWeek + ' thu'
```

```
            DiasSemana = DiasSemana + ' Qui'
        if request.form.get('fri'):
            DaysOfWeek = DaysOfWeek + ' fri'
            DiasSemana = DiasSemana + ' Sex'
        if request.form.get('sat'):
            DaysOfWeek = DaysOfWeek + ' sat'
            DiasSemana = DiasSemana + ' Sab'

        if TimeStart != '' and EndDate != '' and DaysOfWeek != '' and device != '':

            Hour = TimeStart[0:2]
            Minute = TimeStart[3:5]

            whenRun = request.form['whenRun']
            action = request.form['action']

            conn=sqlite3.connect('home-database.db')
            c=conn.cursor()
            c.execute("SELECT max(id) FROM schedules")
            record = c.fetchone()
            if record[0] != None:
                newId = str(record[0] + 1)
            else:
                newId = str(1)

            c.close()

            #Agenda a tarefa no agendador
            global sched
            sched = BackgroundScheduler(daemon=True)
            sched.add_job(powerAir,'cron', day_of_week=DaysOfWeek, hour=Hour, minute=Minute ,
end_date=EndDate, id=newId, args=[whenRun, action, newId, device])
            sched.start()

            print('IDDDD DO AGENDAMENTO: ' + newId)
            #Insere a tarefa na tabela de agendamentos
            conn=sqlite3.connect('home-database.db')
            c=conn.cursor()
            c.execute("""INSERT INTO schedules (device, daysWeek, timeStart, endDate, whenRun,
action)
            VALUES((?), (?), (?), (?), (?), (?))""", ('Ar sala', DiasSemana, TimeStart,
EndDate, whenRun, action))
            conn.commit()
            conn.close()

            msg = 'Agendamento realizado com sucesso.'
        else:
             msg = 'Nao foi possivel realizar o agendamento.'

    return render_template('taskSchedulerDays.html', msg=msg, userRemote=userRemote)

@app.route("/schedules", methods=['GET', 'POST'])
@login_required
def schedules():
   #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    conn=sqlite3.connect('home-database.db')
    conn.row_factory = dict_factory
    c=conn.cursor()
    c.execute("SELECT * FROM schedules ORDER BY id DESC")
    records = c.fetchall()
    c.close()

    if request.method == 'POST':
       idDelete = request.form['excluir']
       idDeleteInt = int(idDelete)

       conn=sqlite3.connect('home-database.db')
       c=conn.cursor()
       c.execute("""DELETE FROM schedules WHERE id = ?""", (idDeleteInt,))
       conn.commit()
```

```
        conn.close()

        global sched
        sched = BackgroundScheduler(daemon=True)

        try:
         sched.remove_job(idDelete)
         print('Agendamento removido')

        except:
         print('Agendamento não removido')

        return redirect(url_for('schedules'))

    return render_template('schedules.html', records=records, userRemote=userRemote)

@app.route("/updateServer")
@login_required
def updateServer():
    #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    return ('ControlHome', 200)

@app.route("/chartOfTemperature")
@login_required
def chartOfTemperature():
   #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    legend = 'Temperatura C'
    conn=sqlite3.connect('home-database.db')
    conn.row_factory = dict_factory
    c=conn.cursor()
    c.execute("SELECT temperature, strftime('%H', time) AS hour FROM readingsTH WHERE date >=
datetime('now','-2 day') ORDER BY id LIMIT 300")
    #c.execute("SELECT temperature, strftime('%H', time) AS hour FROM readingsTH WHERE
strftime('%d', date) = strftime('%d', 'now') ORDER BY id LIMIT 1500")
    #c.execute("SELECT temperature, strftime('%H', time) AS hour FROM readingsTH ORDER BY id
DESC LIMIT 1500")
    readings = c.fetchall()
    c.close()

    return render_template('chartOfTemperature.html', readings=readings,  legend=legend,
userRemote=userRemote)

@app.route("/chartOfHumidity")
@login_required
def chartOfHumidity():
   #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    legend = 'Umidade %'
    conn=sqlite3.connect('home-database.db')
    conn.row_factory = dict_factory
    c=conn.cursor()
    c.execute("SELECT humidity, strftime('%H', time) AS hour FROM readingsTH WHERE date >=
datetime('now','-2 day') ORDER BY id LIMIT 300")
    readings = c.fetchall()
    c.close()
```

```
    return render_template('chartOfHumidity.html', readings=readings,  legend=legend,
userRemote=userRemote)

@app.route("/readingsTH")
@login_required
def readingsTH():
   #Verify user request
    global userRemote

    if request.remote_addr and (request.remote_addr != '127.0.0.1'):
        userRemote = 'false'
    else:
        userRemote = 'true'

    conn=sqlite3.connect('home-database.db')
    conn.row_factory = dict_factory
    c=conn.cursor()
    c.execute("SELECT * FROM readingsTH WHERE date >= datetime('now','-2 day') ORDER BY id
DESC LIMIT 1500")
    readings = c.fetchall()
    c.close()

    return render_template('readingsTH.html', readings=readings, userRemote=userRemote)

@socketio.on('my event')
def handle_my_custom_event(json):
    print('received json data here: ' + str(json))

if __name__ == "__main__":
                    socketio.run(app, host='0.0.0.0', port=80, debug=0)
```

APÊNDICE C – Código do aplicativo Android

CÓDIGO DO APLICATIVO ANDROID

```
package com.example.controlhome2;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.Lifecycle;
import androidx.lifecycle.OnLifecycleEvent;

import android.os.Handler;
import android.webkit.WebView;
import android.webkit.WebViewClient;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;

import java.io.BufferedInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class MainActivity extends AppCompatActivity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        WebView webView = findViewById(R.id.webview);
        webView.setWebViewClient(new WebViewClient());

        //Enable JavaScript
        webView.getSettings().setJavaScriptEnabled(true);
        //Enable Cache
        webView.getSettings().setDomStorageEnabled(true);
        // Instantiate the RequestQueue.
        RequestQueue queue = Volley.newRequestQueue(this);
        String url = "http://192.168.0.110:80/updateServer";

        // Request a string response from the provided URL.
        StringRequest stringRequest =
            new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    System.out.println(response);
                    if(response.equals("ControlHome")){
                        webView.loadUrl("http://192.168.0.110:80/");
                    }else{
                        System.out.println("Caiu no else do onCreate");
                        webView.loadUrl("http://7b00.ngrok.io");
                    }
                }
```

```
        }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            System.out.println("Servidor nao encontrado");
            webView.loadUrl("http://7b00.ngrok.io");
        }
    });

    // Add the request to the RequestQueue.
    queue.add(stringRequest);

    // Create the Handler object (on the main thread by default)
    Handler handler = new Handler();
    // Define the code block to be executed
    Runnable runnableCode = new Runnable() {
        @Override
        public void run() {
        System.out.println("Requisitou o servidor");
        String url = "http://192.168.0.110:80/updateServer";

        // Request a string response from the provided URL.
        StringRequest stringRequest =
                new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        System.out.println("Request local");
                    }
                }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
        System.out.println("Request remoto");
        webView.loadUrl("http://7b00.ngrok.io/updateServer");
        }
        });
        // Add the request to the RequestQueue.
        queue.add(stringRequest);
        handler.postDelayed(this, 60000);
        }
    };
    // Start the initial runnable task by posting through the handler
    handler.post(runnableCode);
}

@Override
public void onResume(){
    super.onResume();
    System.out.println("Retomou a pagina1");

    setContentView(R.layout.activity_main);
    WebView webView = findViewById(R.id.webview);
    webView.setWebViewClient(new WebViewClient());

    //Enable JavaScript
    webView.getSettings().setJavaScriptEnabled(true);
    // Instantiate the RequestQueue
    RequestQueue queue = Volley.newRequestQueue(this);
    String url = "http://192.168.0.110:80/updateServer";

    // Request a string response from the provided URL.
    StringRequest stringRequest =
```

```
            new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                System.out.println("Antes:"+ response+":Depois");
                if(response.equals("ControlHome")){
                    webView.loadUrl("http://192.168.0.110:80/");
                }else{
                    System.out.println("Caiu no else do onResume");
                    webView.loadUrl("http://7b00.ngrok.io");
                }}
            }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                System.out.println("Servidor nao encontrado");
                webView.loadUrl("http://7b00.ngrok.io");
            }
        });
        queue.add(stringRequest);
    }
}
```

APÊNDICE D – Código do ESP8266

CÓDIGO DO ESP8266

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

#define DHTTYPE DHT22

const char* ssid = "JS_House";
const char* password = "bartolomeu2020";
int UpdateTempHumidt = 0;

const char* mqtt_server = "192.168.0.110";

WiFiClient espClient;
PubSubClient client(espClient);

const int ledGPIO0 = 0;
const int ledGPIO16 = 16;

// DHT Sensor
const int DHTPin = 5;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;

char data[80];


void setup_wifi() {
  delay(10);

  Serial.println();
  Serial.print("Conectando ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Conectado ao IP: ");
  Serial.println(WiFi.localIP());
}

//Callback message receiving routine
void callback(String topic, byte* message, unsigned int length) {
  Serial.print("Recebeu msg no topico: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String messageTemp;

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

  if(topic=="/esp8266/4"){
      UpdateTempHumidt = 1;

      if(messageTemp == "1"){
        digitalWrite(ledGPIO0, HIGH);
        Serial.print(" On");
      }
      else if(messageTemp == "0"){
        digitalWrite(ledGPIO0, LOW);
        Serial.print(" Off");
      }
  }
  if(topic=="/esp8266/5"){
```

```
      Serial.print("Changing GPIO 16 to ");
      if(messageTemp == "1"){
        digitalWrite(ledGPIO16, HIGH);
        Serial.print("On");
      }
      else if(messageTemp == "0"){
        digitalWrite(ledGPIO16, LOW);
        Serial.print("Off");
      }
  }
  if(topic=="/esp8266/6"){
      UpdateTempHumidt = 1;
  }

  Serial.println();
}


void reconnect() {

  while (!client.connected()) {
    Serial.print("Tentando conectar ao broker MQTT");

    if (client.connect("ESP8266Client")) {
      Serial.println("Conectado");
      client.subscribe("/esp8266/4");
      client.subscribe("/esp8266/5");
      client.subscribe("/esp8266/6");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println("Tentando novamente em 5 segundos");
      delay(5000);
    }
  }
}

void setup() {
  dht.begin();
  pinMode(ledGPIO16, OUTPUT);
  pinMode(ledGPIO0, OUTPUT);

  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  if(!client.loop())

    client.connect("ESP8266Client");

    //Updates temperature and humidity on every page reload
    if (UpdateTempHumidt == 1) {
      delay(1500);
      UpdateTempHumidt = 0;

      float h = dht.readHumidity();
      float t = dht.readTemperature();

      if (isnan(h) || isnan(t)) {
        Serial.println("Leitura do sensor falhou");
        return;
      }

      float hic = dht.computeHeatIndex(t, h, false);
      static char temperatureTemp[7];
      dtostrf(hic, 6, 2, temperatureTemp);

      static char humidityTemp[7];
      dtostrf(h, 6, 2, humidityTemp);

      // Publish temperature and humidity
```

```
      client.publish("/esp8266/temperature", temperatureTemp);
      client.publish("/esp8266/humidity", humidityTemp);
    }

    now = millis();

    //New reading every minute
    if (now - lastMeasure > 60000) {
      lastMeasure = now;

      float h = dht.readHumidity();
      float t = dht.readTemperature();

      if (isnan(h) || isnan(t)) {
        Serial.println("Leitura do sensor falhou");
        return;
      }

      float hic = dht.computeHeatIndex(t, h, false);
      static char temperatureTemp[7];
      dtostrf(hic, 6, 2, temperatureTemp);

      static char humidityTemp[7];
      dtostrf(h, 6, 2, humidityTemp);

      String dhtReadings = "{ \"temperature\": \"" + String(temperatureTemp) + "\",
\"humidity\" : \"" + String(humidityTemp) + "\"}";
      dhtReadings.toCharArray(data, (dhtReadings.length() + 1));

      // Publishes Temperature and Humidity values
      client.publish("/esp8266/temperature", temperatureTemp);
      client.publish("/esp8266/humidity", humidityTemp);
      client.publish("/esp8266/dhtreadings", data);

    }
}
```

ANEXO A - Lei n. 9.610, de 19 de fevereiro de 1998

Presidência da República
Casa Civil
Subchefia para Assuntos Jurídicos

LEI Nº 9.610, DE 19 DE FEVEREIRO DE 1998[1].

Altera, atualiza e consolida a legislação sobre direitos autorais e dá outras providências.

O PRESIDENTE DA REPÚBLICA Faço saber que o Congresso Nacional decreta e eu sanciono a seguinte Lei:

Título I - Disposições Preliminares

Art. 1º Esta Lei regula os direitos autorais, entendendo-se sob esta denominação os direitos de autor e os que lhes são conexos.

Art. 2º Os estrangeiros domiciliados no exterior gozarão da proteção assegurada nos acordos, convenções e tratados em vigor no Brasil.

Parágrafo único. Aplica-se o disposto nesta Lei aos nacionais ou pessoas domiciliadas em país que assegure aos brasileiros ou pessoas domiciliadas no Brasil a reciprocidade na proteção aos direitos autorais ou equivalentes.

Art. 3º Os direitos autorais reputam-se, para os efeitos legais, bens móveis.

Art. 4º Interpretam-se restritivamente os negócios jurídicos sobre os direitos autorais.

Art. 5º Para os efeitos desta Lei, considera-se:

I - publicação - o oferecimento de obra literária, artística ou científica ao conhecimento do público, com o consentimento do autor, ou de qualquer outro titular de direito de autor, por qualquer forma ou processo;

II - transmissão ou emissão - a difusão de sons ou de sons e imagens, por meio de ondas radioelétricas; sinais de satélite; fio, cabo ou outro condutor; meios óticos ou qualquer outro processo eletromagnético;

III - retransmissão - a emissão simultânea da transmissão de uma empresa por outra;

IV - distribuição - a colocação à disposição do público do original ou cópia de obras literárias, artísticas ou científicas, interpretações ou execuções fixadas e fonogramas, mediante a venda, locação ou qualquer outra forma de transferência de propriedade ou posse;

V - comunicação ao público - ato mediante o qual a obra é colocada ao alcance do público, por qualquer meio ou procedimento e que não consista na distribuição de exemplares;

VI - reprodução - a cópia de um ou vários exemplares de uma obra literária, artística ou científica ou de um fonograma, de qualquer forma tangível, incluindo qualquer armazenamento permanente ou temporário por meios eletrônicos ou qualquer outro meio de fixação que venha a ser desenvolvido;

VII - contrafação - a reprodução não autorizada;

VIII - obra:

a) em co-autoria - quando é criada em comum, por dois ou mais autores;

b) anônima - quando não se indica o nome do autor, por sua vontade ou por ser desconhecido;

c) pseudônima - quando o autor se oculta sob nome suposto;

d) inédita - a que não haja sido objeto de publicação;

e) póstuma - a que se publique após a morte do autor;

f) originária - a criação primígena;

g) derivada - a que, constituindo criação intelectual nova, resulta da transformação de obra originária;

h) coletiva - a criada por iniciativa, organização e responsabilidade de uma pessoa física ou jurídica, que a publica sob seu nome ou marca e que é constituída pela participação de diferentes autores, cujas contribuições se fundem numa criação autônoma;

i) audiovisual - a que resulta da fixação de imagens com ou sem som, que tenha a finalidade de criar, por meio de sua reprodução, a impressão de movimento, independentemente dos processos de sua captação, do suporte usado inicial ou posteriormente para fixá-lo, bem como dos meios utilizados para sua veiculação;

IX - fonograma - toda fixação de sons de uma execução ou interpretação ou de outros sons, ou de uma representação de sons que não seja uma fixação incluída em uma obra audiovisual;

X - editor - a pessoa física ou jurídica à qual se atribui o direito exclusivo de reprodução da obra e o dever de divulgá-la, nos limites previstos no contrato de edição;

XI - produtor - a pessoa física ou jurídica que toma a iniciativa e tem a responsabilidade econômica da primeira fixação do fonograma ou da obra audiovisual, qualquer que seja a natureza do suporte utilizado;

XII - radiodifusão - a transmissão sem fio, inclusive por satélites, de sons ou imagens e sons ou das representações desses, para recepção ao público e a transmissão de sinais codificados, quando os meios de decodificação sejam oferecidos ao público pelo organismo de radiodifusão ou com seu consentimento;

XIII - artistas intérpretes ou executantes - todos os atores, cantores, músicos, bailarinos ou outras pessoas que representem um papel, cantem, recitem, declamem, interpretem ou executem em qualquer forma obras literárias ou artísticas ou expressões do folclore.

Art. 6º Não serão de domínio da União, dos Estados, do Distrito Federal ou dos Municípios as obras por eles simplesmente subvencionadas.

---

[1] Disponível em: http://www.planalto.gov.br/ccivil_03/leis/l9610.htm.