CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS TIMÓTEO

Marcell Reis Leal

CONTROLE DE DISPOSITIVOS COM INFRAVERMELHO UTILIZANDO O GOOGLE AGENDA

Timóteo

2022

Marcell Reis Leal

CONTROLE DE DISPOSITIVOS COM INFRAVERMELHO UTILIZANDO O GOOGLE AGENDA

Monografia apresentada à Coordenação de Engenharia de Computação do Campus Timóteo do Centro Federal de Educação Tecnológica de Minas Gerais para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Lucas Pantuza Amorim
Coorientador: William Marlon Ferreira

Timóteo

2022

Marcell Reis Leal

Controle de Dispositivos com Infravermelho Utilizando o Google Agenda

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação do Centro Federal de Educação Tecnológica de Minas Gerais, campus Timóteo, como requisito parcial para obtenção do título de Engenheiro de Computação.

Trabalho aprovado. Timóteo, 18 de fevereiro de 2022.

Prof. Dr. Lucas Pantuza Amorim
Orientador

Prof. Me. William Marlon Ferreira
Coorientador

Prof. Me. Alessio Miranda Junior
Professor Convidado

Prof. Dr. Elder de Oliveira Rodrigues
Professor Convidado

Agradecimentos

Gostaria de agradecer a Deus primeiramente por ter me suprido até aqui e me dado a capacidade para finalizar este trabalho.

Agradeço pelas dificuldades e pelos desafios enfrentados nessa caminhada, que me fizeram crescer, amadurecer e me tornar a pessoa na qual me orgulho de ser.

Aos meus pais que sempre me apoiaram e me incentivaram.

Agradeço aos meus colegas de curso e todos que de alguma forma me ajudaram no desenvolvimento deste trabalho.

Agradeço aos professores Lucas Pantuza Amorim e William Ferreira por toda ajuda no desenvolvimento do trabalho.

E por fim, agradeço aos professores que me ajudaram e sempre estiveram dispostos a compartilhar conhecimento.

“Penso 99 vezes e nada descubro.
Deixo de pensar, mergulho no silêncio.
E eis que a verdade se revela“.
Albert Einstein

Resumo

O crescimento em soluções de automações residenciais e da indústria IoT (*Internet of Things*), culminou no crescimento de dispositivos inteligentes conectados à rede mundial de computadores. Sendo assim, o serviço de computação em nuvem surgiu com o intuito de facilitar o acesso e garantir segurança no funcionamento destes aparelhos.Nesse sentido, este trabalho utiliza-se dos conceitos citados para apresentar uma solução de automação residencial por meio de um protótipo que controla o funcionamento de aparelhos com infravermelhos. O calendário Google Agenda foi utilizado para gerenciar os dispositivos que são programados para funcionar em determinados horários. A placa de desenvolvimento NodeMCU foi responsável por controlar o acionamento dos aparelhos. Para implementar a comunicação entre o Google Agenda, o NodeMCU, e o aparelho infravermelho, utilizou-se uma máquina virtual disponibilizada pela Amazon Web Services, a qual foi encarregada de executar um código implementado em Python e um Broker MQTT. Logo, por meio dessa implementação, foi possível realizar a comunicação com a API do Google Agenda e enviar os dados necessários para o NodeMCU via MQTT, com o intuito de controlar os dispositivos.

Palavras-chave: dispositivos inteligentes, casa inteligente, automação residencial, computação em nuvem, NodeMCU.

Abstract

The growth in home automation solutions and the IoT industry, culminated in the increasement of smart devices connected to the world wide web. Therefore, The cloud computing service emerged, with the aim of facilitating access and ensuring safety in the operation of these devices. This work seeks to use the concepts mentioned, to present a home automation solution through a prototype that controls the operation of infrared devices, programmed to operate in certain mined schedules. The calendar used to manage the devices was Google Calendar. THE NodeMCU development board was responsible for controlling the activation of the devices. clocks using leds and sensors. To implement communication between Google Agenda, the NodeMCU, and the infrared device, use was made of a virtual machine available. Made available by Amazon Web Services, in charge of executing code implemented in Python and an MQTT Broker. Through this implementation it was possible to communicate with the Google Calendar API and send the necessary data to NodeMCU via MQTT, with the purpose of controlling the devices.

Keywords: cloud computing, internet of things, smart devices, smart home, NodeMCU.

Lista de ilustrações

Sumário

1 Introdução

A busca por ambientes sustentáveis é crescente, já que a população mundial poderá chegar aos 8,5 bilhões em 2030 (ADRIAN, 2014). CATA (2015) e SANTOS (2020) definem ambientes inteligentes como um ambiente em que dispositivos habilitados por sensores e em rede funcionam de forma contínua e colaborativa para tornar a vida de seus habitantes mais confortável. Sendo assim, a expectativa é que ocorra um enorme crescimento de dispositivos conectados à Internet.

A demanda por tecnologias sustentáveis está atrelada a utilização responsável de recursos essenciais, como água e energia, refletindo na busca por alternativas que reduzam o desperdício e estimulem o consumo consciente (CLEMENTS, 2011). O gasto energético proveniente das lâmpadas e aparelhos de ar-condicionado é de 14% e 20%, respectivamente, em residências (TELLES, 2016). Logo, é notável que diversos problemas existentes, como o gerenciamento de energia elétrica, podem ser solucionados com o uso de aparelhos inteligentes conectados a Internet, que promovem o consumo eficaz dos recursos naturais.

A principal característica dos ambientes inteligentes é a necessidade de responder rapidamente às situações adversas e que a melhoria de um conjunto de fatores intangíveis, como saúde, relações sociais, qualidade do ambiente, prosperidade e condições de vida seja o principal objetivo (FERREIRA, 2018). De acordo com uma previsão da International Data Corporation (IDC), estima-se que haverá 41,6 bilhões de dispositivos IoT (*Internet of Things*), gerando 79,4 *zettabytes* de dados, com uma taxa de crescimento anual composta de 28.7% durante o período de 2018 a 2025 (SECURITY, 2019).

Com o volume de informações produzidas pela humanidade, será possível realizar o tratamento, análise e a computação dos dados para extração de valor, sendo que depois desse processo os próprios dispositivos serão capazes de tomar decisões a partir dos dados obtidos (GONÇALVES; AURELIO, 2019). Portanto, devido à expectativa de desenvolvimento dos ambientes inteligentes, a IoT surgiu como um dos braços mais importantes para a evolução de diversas áreas.

Dessa forma, a implementação de automação residencial, caracterizada por um local com aparelhos que podem ser operados remotamente por meio de um sistema de controle, está cada vez mais evidente (BING et al., 2011). Assim, de acordo com LIMA (2019), a crescente combinação entre conceitos, como IoT e WSN (*Wireless Sensors Network*), faz com que o monitoramento de ambientes e o controle de dispositivos seja feita de modo automatizado.

1.1 Objetivo geral

O projeto proposto realizará a substituição do controle remoto tradicional de dispositivos infravermelhos. A integração do Google Agenda (GOOGLE) com o nodeMCU será responsável por enviar sinais infravermelhos de acionamento para o aparelho, de modo que

seja possível utilizar equipamentos por meio da programação de horários de funcionamento.

1.2 Objetivos específicos

1. Desenvolver um protótipo de *hardware* capaz de enviar sinais infravermelhos aos dispositivos de acordo com comandos recebidos do servidor.
2. Programar *script* para receber informações da API do Google Agenda (GOOGLE);
3. Integrar o módulo eletrônico, a Amazon AWS EC2 (AMAZON) e o Google Agenda (GOOGLE) .

1.3 Justificativa

A crescente preocupação com o uso consciente da matriz energética, em vista da redução de recursos naturais e do aumento da população humana, a qual exerce pressões sobre o meio ambiente, instiga-nos a pesquisar meios de otimizar o consumo de energia nas instalações, já que o consumo não planejado de energia de dispositivos ligados fora do horário de uso é uma parcela importante desse gasto.

A dimensão ambiental é cada vez mais reconhecida como uma base para a sustentabilidade (VOS, 2007), uma vez que a gestão racional desses recursos é fundamental para a manutenção da qualidade de vida (ALVES, 2015).

A automação do sistema de desligamento e gerenciamento de dispostivos, busca sanar problemas relacionados à utilização dos controles remotos, dispositivos esses que são perdidos e sofrem avarias constantes.

Programar horários regulares para o funcionamento do ar-condicionado e manter o ambiente fechado, assim como utilizar as lâmpadas apenas quando existir pessoas presentes no ambiente é uma excelente forma de economizar energia (TELLES, 2016).

2 Fundamentação Téorica

Os assuntos abordados neste capítulo são necessários para a elaboração da proposta e melhor entendimento de sua relevância. São eles: *smart home*, protocolos de aplicação, *cloud computing* e o nodeMCU.

2.1 *Smart home*

Smart home ("Casa inteligente") é uma aplicação para o segmento da IoT que pode ser definida como um ambiente na qual aparelhos automatizam ou facilitam tarefas antes realizadas por pessoas, como por exemplo, automatização do funcionamento das luzes de acordo com a presença em um ambiente, facilitando a tarefa de ligar ou desligar uma lâmpada por meio de um interruptor (SOUZA, 2021). Alguns dos benefícios proporcionados por este conceito, são (RICARDO, 2021):

- **Interconectividade:** a integração de produtos e sistemas domésticos inteligentes permite, por exemplo, que os usuários gerenciem e monitorem remotamente termostatos, analisem imagens de câmeras de vigilância e programem luzes internas e externas;
- **Segurança:** casas inteligentes podem ser gerenciadas de qualquer lugar, a qualquer momento, por meio de monitoramento remoto, de modo a fornecer um alto nível de segurança ao usuário;
- **Gestão de energia:** A automatização no uso dos sistemas de aquecimento, resfriamento e iluminação pode economizar nos custos de energia. Tomadas inteligentes podem gerenciar o desligamento automático de dispositivos eletrônicos, como TVs e sistemas de jogos. As lâmpadas led inteligentes permitem que os usuários acionem e apaguem as luzes em determinados momentos promovendo segurança e conveniência.

2.2 Protocolos de aplicação

As tecnologias IoT apresentam protocolos que têm por finalidade oferecer um modo de habilitar a troca de dados para clientes e servidores, sendo essas, conexões de arquitetura cliente/servidor ou cliente/cliente (PRIYADARSHI; BEHURA, 2018).

Após o surgimento de diferentes metodologias e de vários casos de uso, tornou-se complicada a escolha de um protocolo padronizado (KUKKAMÄKI et al., 2018). Com a IoT sendo uma realidade cada vez mais presente é normal que os protocolos utilizados, em processadores robustos passem a não ser soluções ideais para redes de dispositivos menores, as quais tendem possuir recursos limitados de bateria, memória, largura de banda e processamento (KODALI, 2017). Por conta disso, desejando atender aos requisitos apresentados, começaram a surgir adaptações e novos protocolos, como o MQTT (Transporte de Telemetria

de Enfileiramento de Mensagens), o Constrained Application Protocol (CoAP) e as Application Programming Interfaces (APIs) de transferência de estado representacional ou Representational State Transfer (REST).

2.2.1 MQTT

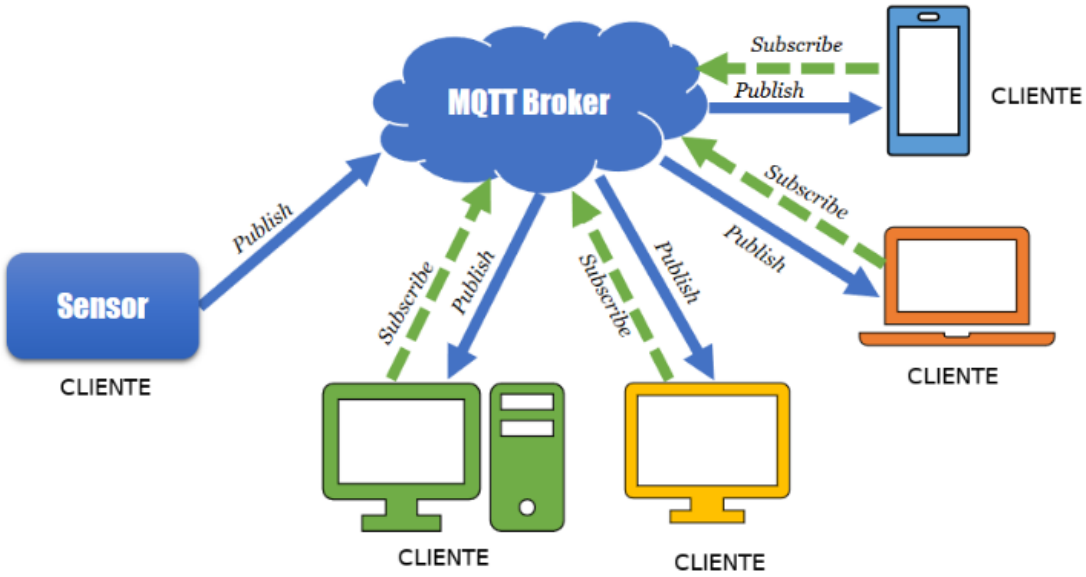
O MQTT desenvolvido pela IBM (IBM) em 1999, reconhecido como padrão pela OASIS (Organization for the Advancement Of Structured Information Standards), é um protocolo de transporte que trabalha sobre o protocolo TCP (Protocolo de Controle de Transmissão), pode ser implementado em vários tipos de redes como conexão cabeada (*Ethernet*) e rede local sem fio (WLAN) (AGHENTA; IQBAL, 2019). Sendo um protocolo orientado por troca de mensagens, é ideal para aplicações IoT, que comumente tem recursos e capacidades limitados (AVELAR, 2011).

O MQTT fornece um protocolo de mensagens no modelo *publish-subcribe* (YASSEIN et al., 2018) e tem os chamados MQTT *clients* e os MQTT *brokers*. Com isso, observa-se que todos os *publishers* e *subscribers* (como sensores, dispositivos *mobile* e demais sistemas que publiquem ou recebam mensagens) são considerados clientes, enquanto os *brokers* são os servidores ou programas encarregados de conectar esses diversos clientes (KODALI, 2017)(Figura 1).

O servidor MQTT, normalmente, é executado na nuvem ou na Internet, é responsável por armazenar os dados publicados em diferentes tópicos e enviar os dados para os clientes corretos (AGHENTA; IQBAL, 2019).

O cliente MQTT é algum tipo de *hardware*, *software*, ou uma combinação dos dois, que conecta ao *broker* com o propósito de compartilhar dados. Quando um dispositivo conectado baixa dados do servidor, esse processo é chamado de subscrição, e esse cliente em particular é chamado de *subscriber*. Por outro lado, caso um dispositivo conectado envie dados para o servidor, o cliente é conhecido como *publisher* e o processo chamado de *Publishing*. O servidor pode suportar diversos *subscribers* e *publishers*, dependendo da capacidade da máquina executando o servidor (AGHENTA; IQBAL, 2019).

Figura 1 – Servidor MQTT



Fonte: (LABORATORY, 2018)

2.3 API RESTFUL

APIs são conjuntos de instruções e padrões de programação que servem para fornecer dados e informações relevantes de uma determinada aplicação (ANTUNES, 2019). APIs são criadas para integrar um *software* a um produto associado, e dessa forma mediar o relacionamento entre os recursos disponíveis no *web service* e o usuário final.

O primeiro cientista da computação a criar essa definição, no ano 2000, foi o Dr. Roy Fielding em sua tese de doutorado (IBM, 2021). O REST proporciona um nível relativamente alto de flexibilidade e liberdade para os desenvolvedores. As APIs de REST surgiram como um método comum para conectar componentes e aplicativos em uma arquitetura de microsserviços (IBM, 2021).

O crescente número de aplicações em nuvem também encoraja a utilização de APIs, afinal, quanto mais os programas conversarem entre si, mais troca de informações haverá e, consequentemente, melhor usabilidade para usuários finais, gestores e tomadores de decisão (THIAGO, 2022).

As APIs utilizam uma rede para o tráfego de informações de uma forma padronizada. Isso pode ser através da Internet ou de uma rede local. Existem três formas de API (ANTUNES, 2019):

1. Local;
2. Baseada em programa;
3. Baseada em *web*.

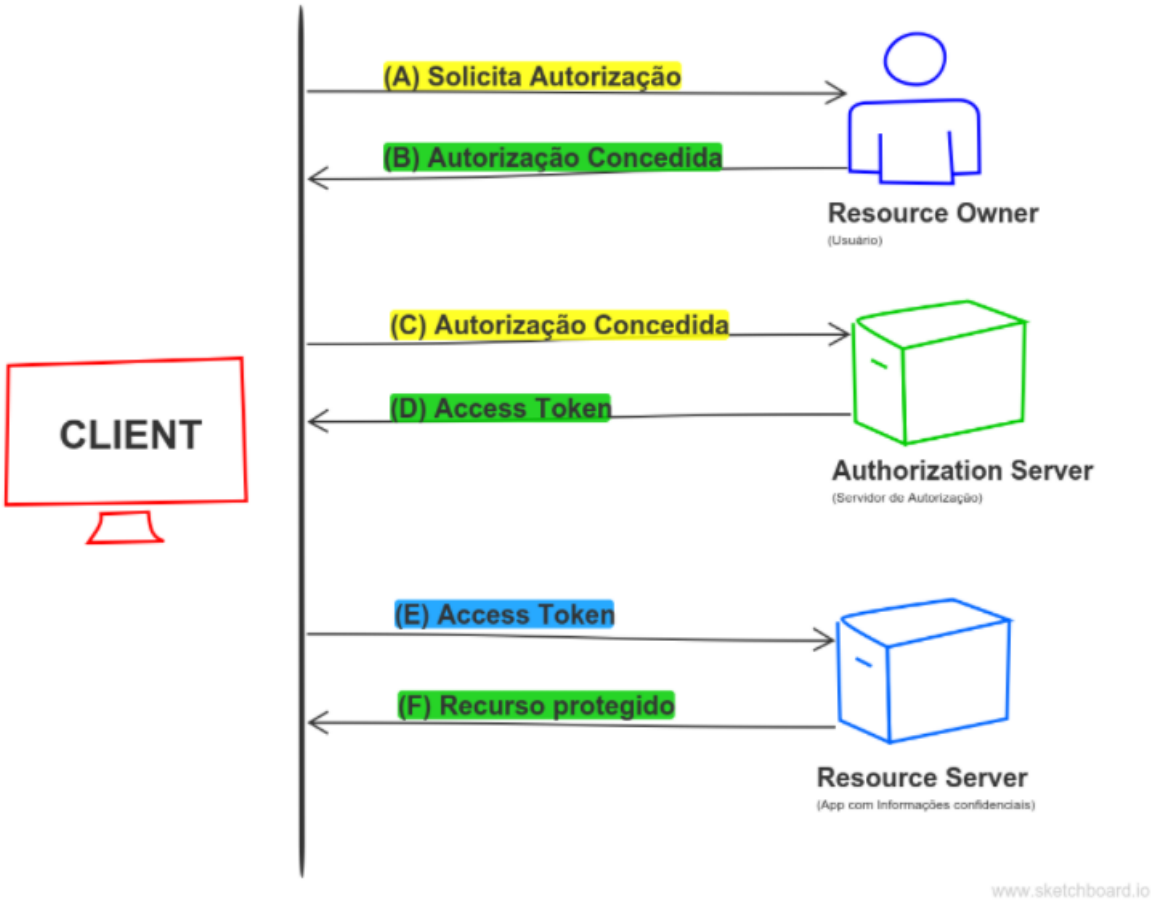
API RESTFUL, é uma API baseada em protocolo HTTP. As APIs RESTFUL aumentam o desempenho em situações de concorrência, ou seja, quando muitas pessoas estão pedindo a mesma coisa ao mesmo tempo. Elas utilizam verbos com o intuito de representar a ação de cada requisição à ser executada. Os verbos são (ANTUNES, 2019):

- `GET`: A requisição é um pedido de dados para a API. A API vai buscar os dados solicitados em algum banco e, provavelmente, vai retornar em formato JSON (formato de notação de objeto JavaScript);
- `POST`: Tipo de requisição utilizada para criar um recurso em uma determinada API. São chamados de recursos o objeto que está sendo tratado naquela API;
- `PUT`: Requisição utilizada para atualizar o recurso indicado com alguma informação;
- `PATCH`: Requisição feita para atualização de somente uma parte de um recurso;
- `DELETE`: Requisição para excluir um dado.

2.3.1 Oauth 2.0

OAuth 2.0 é um protocolo de autorização projetado, por exemplo, para permitir que um site acesse recursos hospedados por outros aplicativos da *web* em nome de um usuário, fazendo uso de tokens de acesso, que é um dado que representa a autorização para acessar recursos em nome do usuário final (OAUTH, 2022). O fluxo de autorização Oauth 2.0, mostrado na Figura 2, é definido por quatro entidades (BRITO, 2019):

Figura 2 – Oauth 2.0



Fonte: (BRITO, 2019)

Resource Owner: entidade que concede acesso aos dados;

Resource Server: API exposta na Internet que contém os dados do usuário;

Authorization Server: responsável por autenticar e emitir os *tokens* de acesso;

Client: aplicação que interage com o Resource Owner.

De maneira prática, o fluxo de autorização pode ser explicado pelas seguintes etapas (BRITO, 2019):

1. O Usuário acessa um *client*. Para ter acesso ao conteúdo protegido da API (*Resource Server*) o *client* (A) Solicita Autorização ao *Resource Owner*;
2. A Autorização é concedida pelo usuário (*Resource Owner*);
3. O *client* solicita um token de acesso ao *Authorization Server*;
4. O Usuário (*Resource Owner*) confirma sua identidade através do usuário e senha ou através de um terceiro (Facebook (FACEBOOK,), Google (GOOGLE,)). Se tudo ocorrer bem, um *access token* será criado e devolvido para o *client* gerenciar.

2.4 *Cloud Computing*

Segundo HASHEM et al. (2015) *cloud computing* é o termo usado para descrever o paradigma que entrega serviços computacionais, no qual os clientes utilizam a Internet e servidores remotos para executar aplicações e armazenar dados. A tecnologia em nuvem permite uma maior eficiência computacional ao remover custos de se manter uma Infraestrutura de TI dentro de uma corporação. Sendo assim, o manejo de recursos torna-se mais fácil, rápido, barato e eficaz, já que não é necessário lidar diretamente com o *hardware* dos serviços utilizados. Dentre os serviços entregues estão:

- Processamento paralelo;
- Recursos virtualizados;
- Segurança;
- Servidores escaláveis.

Atualmente, a computação em nuvem é dividida em 3 modelos (MANDIC, 2022):

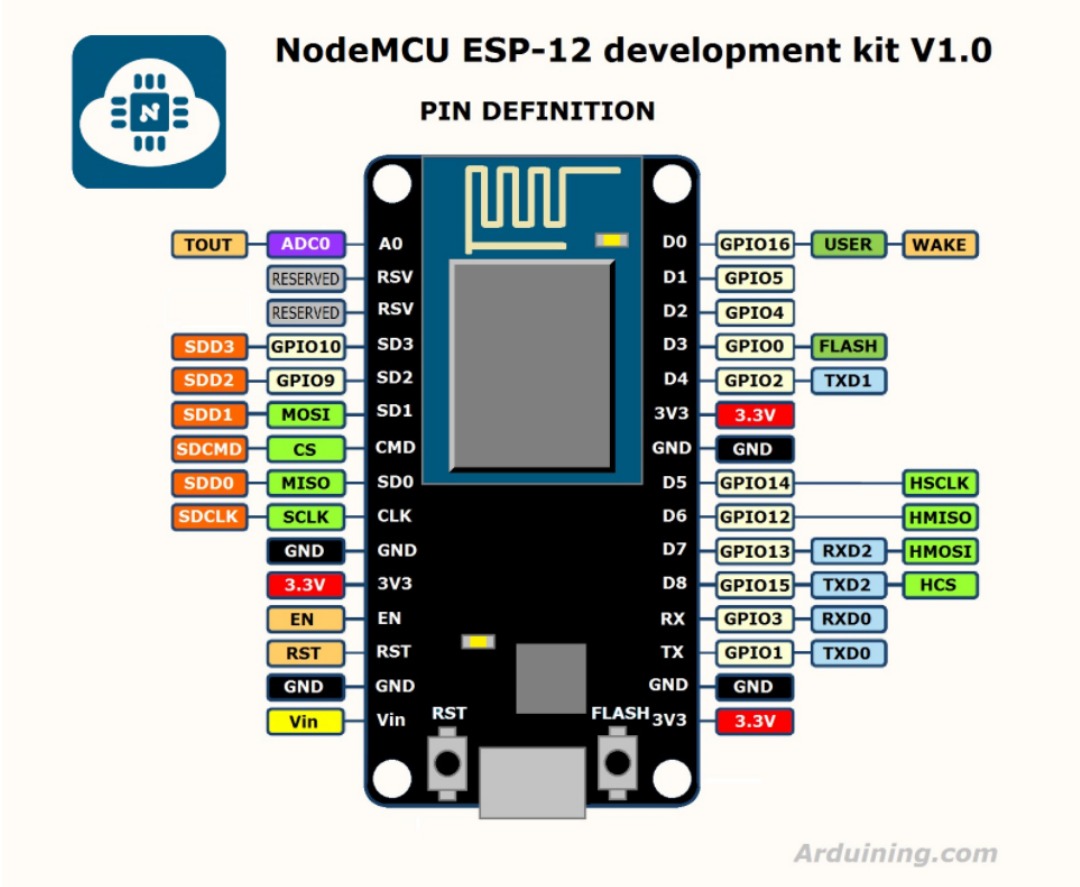
1. SaaS (*Software* como Serviço): disponibiliza o aplicativo de *software* ao usuário por meio de uma interface de navegador ou de programa e faz com que a rede subjacente, o sistema operacional e os recursos funcionem nos bastidores. Não é necessário comprar o *software*, pois o serviço é contratado por meio de assinaturas que dão permissão de acesso. Logo, o uso dos aplicativos pode ser feito de qualquer dispositivo conectado à Internet.
2. PaaS (Plataforma como Serviço): ambiente completo de desenvolvimento on demand, no qual é possível criar, modificar e otimizar softwares e aplicativos. A vantagem do modelo PaaS é que ele inclui sistemas operacionais, ferramentas de desenvolvimento, sistemas de gerenciamento de bancos de dados, serviços de Business Intelligence e muitos outros recursos, além de toda a infraestrutura necessária para executar ou aperfeiçoar aplicações *web* ou móveis.
3. IaaS (Infraestrutura como Serviço): é o modelo de serviço com maior nível de flexibilidade e controle sobre os recursos de tecnologia. Ele proporciona às organizações a capacidade de aproveitar recursos brutos do servidor enquanto o restante do gerenciamento da plataforma e do *software* é de responsabilidade da empresa. Isso permite capacidade extra sem a preocupação com requisitos de *hardware*.

2.5 Módulo ESP8266 nodeMCU

O nodeMCU é uma plataforma *open source* da família ESP8266 criado para ser utilizado no desenvolvimento de projetos IoT. É composta, basicamente, por um *chip* controlador (ESP8266 ESP-12E), porta micro USB para alimentação e comunicação, conversor USB serial integrado, além de possuir WiFi (OLIVEIRA, 2017).

Dentre suas características físicas, as quais podem ser observadas na Figura 3, possui uma faixa de operação entre 5 e 9 V, o que permite a utilização de, por exemplo, uma bateria alcalina de 9 V para alimentação do microcontrolador. Além disso, possui um total de 11 portas GPIO (*General Purpose Input/Output*) (Porta de entrada/saída de propósito geral), as quais podem ser utilizadas para conectar sensores ou outros dispositivos externos (GRAÇA, 2017).

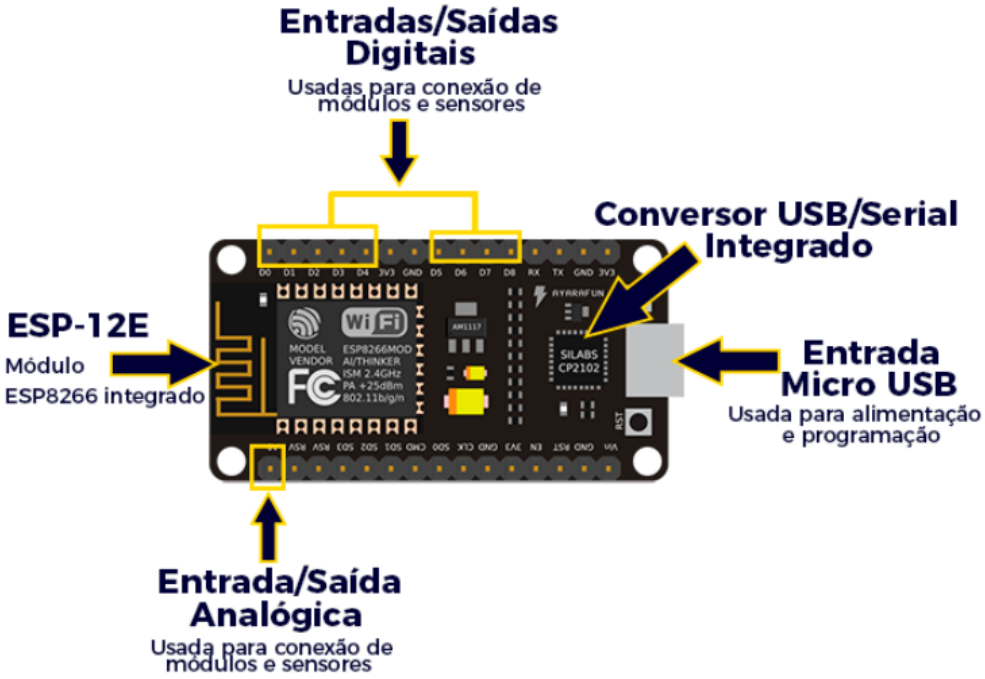
Figura 3 – Pinagem nodeMCU



Fonte: (VEGEY, 2016)

Possui apenas um pino analógico comumente usado para ler valores de componentes, como sensor de temperatura, sensor de gás/fumaça, sensor de álcool, potenciômetro, LDR (OLIVEIRA, 2017), antena embutida e conector micro-usb para conexão ao computador, além de 11 pinos de I/O e conversor analógico-digital (Figura 4). A versão 3 fabricada pela Lolin conta com o conversor USB serial CH340.

Figura 4 – Descrição portas nodeMCU



Fonte: (OLIVEIRA, 2017)

2.5.1 Principais Características

Descrição	Características
Wireless Standard	*IEE802.11b*
Frequency Range	*2.412-2.484 GHz*
Power Transmission & 802.11b	*802.11b:+16 ± 2 dBm (at 11 Mbps)* *802.11g : +14 ± 2 dBm (at 54 Mbps)* *802.11n : +13 ± 2 dBM (at HT20, MCS7)*
Wireless Form	*On-board PCB Antenna*
IO Capability	*UART, I2C, PWM,GPIO,1 ADC*
Electrical Characteristics	*3.3 V Operate* *15 mA output current per GPIO pin* *12 - 200 mA working current* *Less than 200 uA standby current*
Operating Temperature	*-40 to +125 ºC*
Serial Transmission	*110 - 921600 bps, TCP Client 5*
Wireless Network Type	*STA / AP / STA + AP*
Security	*WEP / WPA-PSK / WPA2-PSK*
Encryption	*WEP64 / WEP128 / TKIP / AES*
Firmware Upgrade	*Local Serial Port, OTA Remote Upgrade*
Network Protocol	*IPv4, TCP/ UDP/ FTP/ HTTP*
User Configuration	*AT + Order Set, Web Android/iOS, Smart Link APP*

Fonte: (PAZ, 2018)

3 Trabalhos Correlatos

Este capítulo aborda alguns trabalhos encontrados em que os autores apresentam modelos e pesquisas semelhantes às feitas neste projeto.

3.1 Projeto de Gerenciamento do Sistema de Climatização do Campus Avançado Ipatinga

O projeto de pesquisa aplicada do Instituto Federal de Educação, Ciência e Tecnologia (IFMG) proposto por MARLON (2018) busca controlar aparelhos de ar condicionado substituindo o controle remoto tradicional por um controle programável. Um computador pessoal realiza interação *online* com dispositivos microcontrolados que recebem dados do ambiente e enviam comandos para o aparelho de ar condicionado. Dessa forma, baseado no protótipo, foi desenvolvido um sistema controle e automação do sistema de climatização do Campus Avançado Ipatinga.

Após ser realizada a montagem do sistema utilizando sensores, led, receptor infravermelho e nodeMCU em *protoboard*, foi desenvolvido um projeto de *hardware* (Figura 5) em uma placa de circuito impresso. Esse equipamento (Figura 6) junto ao *dashboard* criado, possibilita o ajuste prévio dos horários disponíveis para se ligar os aparelhos de ar condicionado, além de desligá-los de forma automática fora do horário permitido.

Figura 5 – *Hardware* do Projeto

Fonte: (MARLON, 2018)

Figura 6 – Equipamento montado

Fonte: (MARLON, 2018)

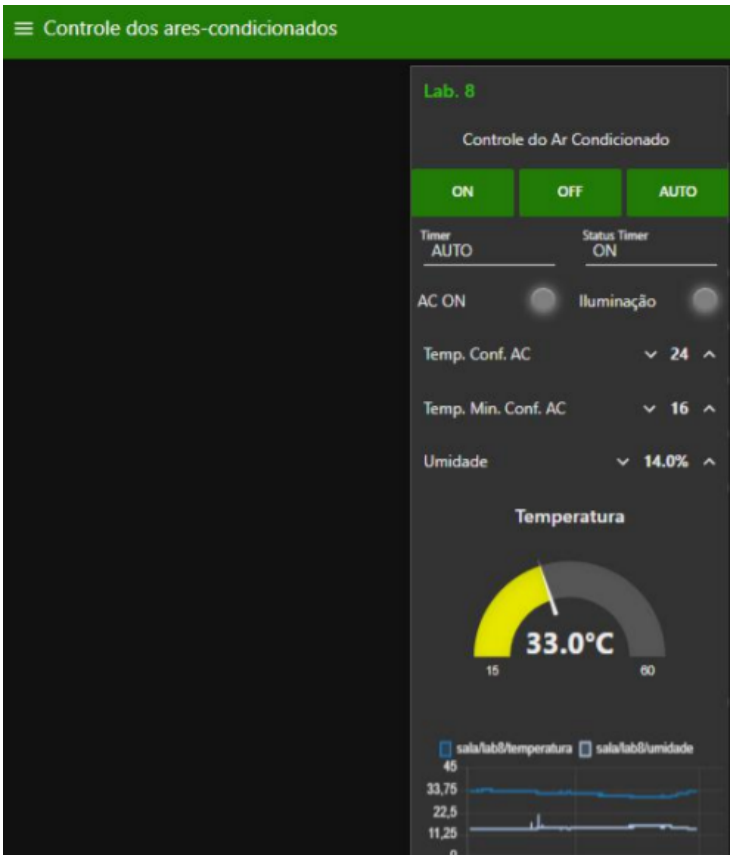
Os usuários da sala/laboratório conseguem ligar e desligar o aparelho dentro dos horários permitidos, além de aumentar ou diminuir a temperatura, através de 2 botões instalados ao lado do interruptor de iluminação (Figura 7).

Figura 7 – Esquema de ativação manual

Fonte: (MARLON, 2018)

Um *dashboard* com a finalidade de monitorar e comandar o sistema junto à ferramenta de desenvolvimento Node-RED foi criado, o que permite controlar o aparelho por meio de gráficos e botões (Figura 8).

Figura 8 – Dashboard utilizado no sistema



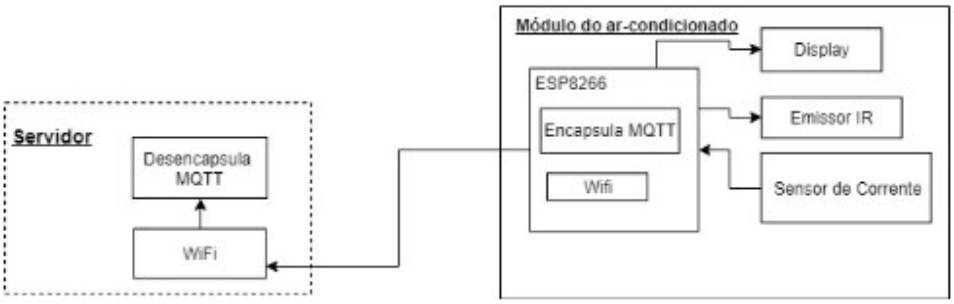
Fonte: (MARLON, 2018)

3.2 Projeto Ubique: Sistema de monitoramento e controle de ar - condicionado

SILVA (2018), em sua tese, busca resolver o consumo excessivo de energia relacionado ao não desligamento e mau funcionamento dos aparelhos de ar- condicionado. Diante disso, foi desenvolvido um sistema capaz de controlar os aparelhos a distância ou localmente.

O sistema de *hardware* foi desenvolvido com o intuito de entregar um protótipo com baixo consumo de energia e capacidade de conexão a rede sem fio (Figura 9). Assim sendo, utilizou-se um microcontrolador (ESP8266EX), para enviar a informação via emissor IR ao ar-condicionado e um sensor de corrente afim de verificar o estado de ligado/desligado do aparelho com o objetivo de mostrar e armazenar essa informação em um servidor *web*.

Figura 9 – Arquitetura do sistema



Fonte: (CAVALCANTE, 2018)

O resultado final deste trabalho foi um protótipo de alta fidelidade (Figura 12). O *software* KiCad foi utilizado para criação de um circuito impresso na qual o microcontrolador, as ligações e os componentes foram posicionados. Além disso, uma página *web* responsável por fornecer uma maneira de ligar/desligar e controlar a temperatura do aparelho via Wi-Fi foi implementada.

Figura 10 – Protótipo de alta fidelidade

Fonte: (CAVALCANTE, 2018)

Figura 11 – Página web ar-condicionado

Fonte: (CAVALCANTE, 2018)

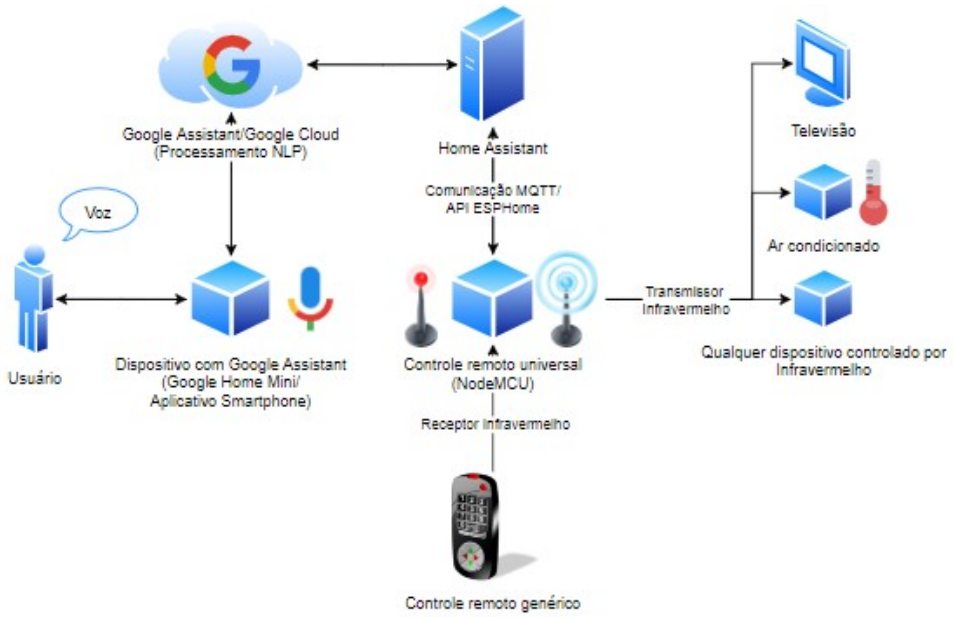
3.3 Projeto de um controle remoto universal infravermelho com suporte a comandos de voz

SOUZA (2021), com o intuito de comandar dispositivos eletrônicos, utiliza conceitos de Casas Inteligentes para modelar e implementar um controle remoto universal infravermelho com suporte a comando de voz. Assim, no desenvolvimento do projeto, utilizou-se um assistente virtual para recepção e tratamento de comandos do Google Assistant (ASSISTANT) , além de um microcontrolador ESP8266.

Para implementar a comunicação entre o assistente de voz e o protótipo foi utilizado um servidor com o sistema operacional de automação residencial Home Assistant (ASSISTANT).

O projeto conta com um assistente de voz composto por um dispositivo com microfone e uma camada de processamento NLP, responsável pela interpretação dos comandos executados pelo usuário. Um servidor intermediário foi utilizado tanto para implementar a comunicação entre o assistente de voz e o controle remoto universal IR quanto para integrar o Google Assistant com o microcontrolador utilizando o protocolo MQTT.

Figura 12 – Arquitetura do projeto



Fonte: (SOUZA, 2021)

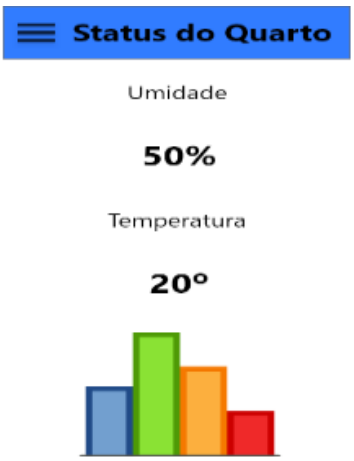
Sendo assim, com o protótipo finalizado foi possível mensurar a temperatura ambiente e controlar aparelhos de televisão de diferentes fabricantes.

3.4 *Smart Room*: Criação de um quarto inteligente com preço acessível

SATALOFF (2018) da Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica, após reconhecer os diversos problemas de automação residencial, criou um quarto inteligente com um arduíno de baixo custo, visando o desenvolvimento de um sistema com um preço mais acessível que os já disponíveis no mercado, cuja finalidade é verificar e controlar de maneira automática a temperatura e humidade de um quarto.

O projeto foi atrelado ao desenvolvimento de um controlador físico dos aparelhos de um quarto utilizando a placa nodeMCU, CloudMQTT (CLOUDMQTT) e sensores, para que os comandos sejam enviados aos dispositivos sem a necessidade de intervenção humana, além de um aplicativo *mobile* (Figura 8) que permite controlar cada um dos aparelhos remotamente e fornece análises em tempo real da temperatura e humidade do ambiente.

Figura 13 – Aplicativo *mobile*



Fonte: (SATALOFF, 2018)

A principal funcionalidade do sistema foi atrelada a ideia de automatizar tarefas de equipamentos presentes em um quarto. Para tal, uma pesquisa para identificação das principais necessidades dos usuários deste ambiente foi feita. Apesar de algumas funcionalidades presentes no aplicativo *mobile* criado não serem integradas com o servidor, esta pesquisa serve de referência para desenvolvimento de trabalhos semelhantes, já que aponta possíveis dificuldades que podem ser enfrentadas.

4 Implementação de agenda automatizada para controle de aparelhos com infravermelho

O presente capítulo apresenta o procedimento utilizado para a implantação dos recursos necessários. Serão apresentados os papéis de cada componente na arquitetura do trabalho, além dos motivos para escolha das ferramentas utilizadas. Essa é uma pesquisa descritiva, aplicada e experimental, envolvendo diferentes dispositivos, linguagens, *softwares* e tecnologias.

O controle Remoto comumente é composto por um microcontrolador, onde permanece armazenado todas as informações de códigos referente ao aparelho, componentes eletrônicos e um led infravermelho, responsável por enviar informação à dispositivos. A comunicação emissor IR e receptor IR, ocorre através de um protocolo, em que determinada empresa cria o seu próprio protocolo, adapta algum já existente ou até mesmo usa um já existente. O código enviado deve ser composto, por no mínimo, quatro subcódigos (CAVALCANTE, 2018):

- **Iniciar** que indica ou avisa o receptor que um determinado comando será enviado;
- **A sequência** indicando o comando a ser executado, (por exemplo, ligar ou desligar);
- **A informação** referente ao fabricante e/ou aparelho;
- **Parar** sinal quando a tecla é solta.

A luz IR é emitida pelo sol, lâmpadas e qualquer outra coisa que produza calor. Isso significa que há muito ruído de luz IR ao nosso redor. Para evitar que esse ruído interfira no sinal IR, é usada uma técnica de modulação de sinal.

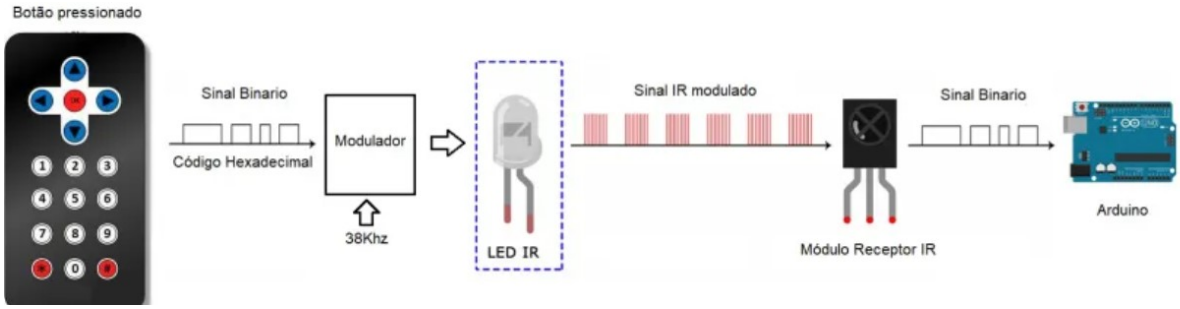
Na modulação de sinal IR, um codificador no controle remoto IR converte um sinal binário (Figura 14) em um sinal elétrico modulado. Este sinal elétrico é enviado para o led transmissor. O led transmissor converte o sinal elétrico modulado em um sinal de luz IR modulado. O receptor IR interpreta o sinal de luz IR e o converte de volta para binário antes de passar a informação para um microcontrolador.

O fotorreceptor IR reconhece diversos protocolos de sinais IR, no entanto, o comando para enviar o código de volta ao aparelho a ser controlado se torna complicado devido a programação complexa para tratar de forma correta o sinal IR antes de ser enviado.

Uma alternativa é a utilização do método RAW para enviar os sinais IR decodificados ao aparelho a ser controlado. Utilizando este método não é necessário saber qual protocolo corresponde ao aparelho que será controlado, necessitando saber somente a linha de códigos

RAW, tamanho da linha e a frequência do sinal. É importante ressaltar que o código RAW precisa ser trabalhado antes de ser enviado ao aparelho a ser controlado (OLIVEIRA, 2017).

Figura 14 – Modulação sinal binário

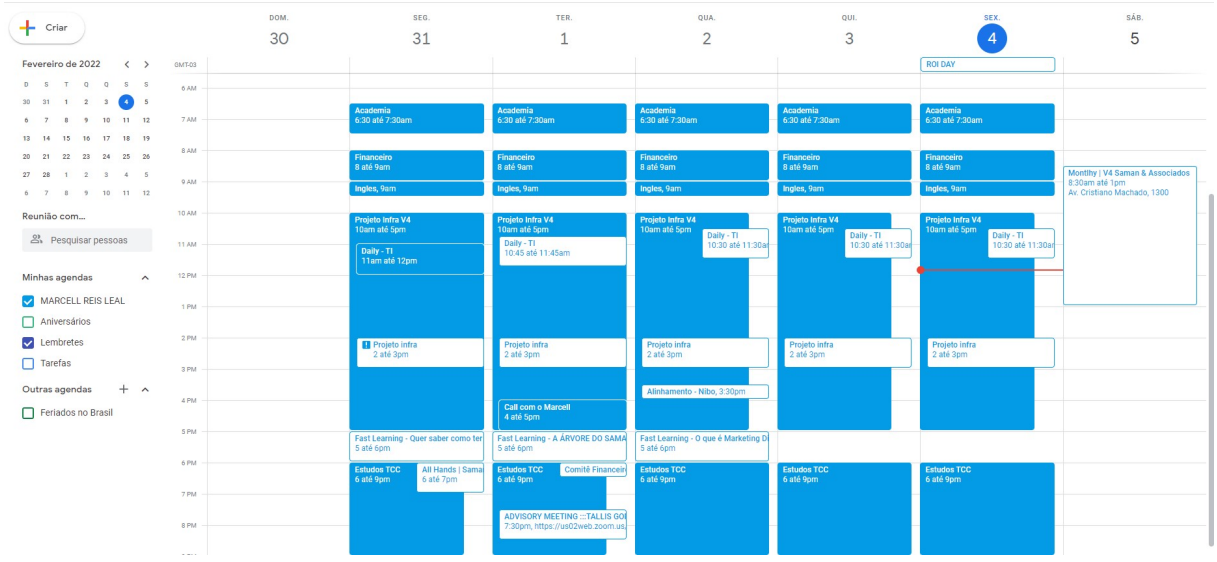


Fonte: (MADEIRA, 2018)

O primeiro integrante do fluxo de funcionamento do sistema desenvolvido é o usuário, o qual irá interagir e criar eventos com horários de início e fim dentro da *web page* do Google Agenda (GOOGLE), a fim de acionar o dispositivo infravermelho.

Google Agenda (GOOGLE) é um calendário grátis feito para *web* e dispositivos *mobile* que permite o compartilhamento de eventos entre os usuários (Figura 15). É a ferramenta ideal para gerenciar calendários pessoais ou profissonais, já que é efetivo e simples de usar (KARCH, 2019).

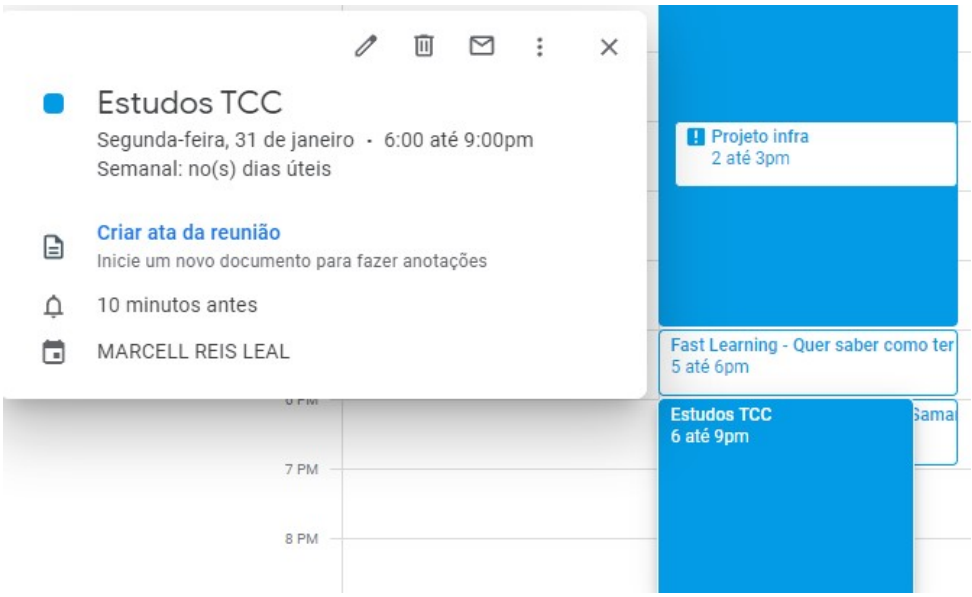
Figura 15 – Captura de tela calendários Google Agenda



Fonte: Elaborada pelo autor

Para adicionar um evento como um aniversário, basta selecionar a data na interface, podendo ser dia de um mês ou hora em um dia ou semana. Após o agendamento, uma caixa de diálogo que contém o dia ou hora é apresentada no interface e, consequentemente, o usuário pode acompanhar os eventos (Figura 16).

Figura 16 – Captura de tela Google Agenda Evento



Fonte: Elaborada pelo autor

A API do Google Agenda (GOOGLE) é uma API RESTFUL que concede a maioria dos recursos disponíveis na interface do Google Agenda (GOOGLE). Pode ser acessada por meio de *requests* HTTP ou através de bibliotecas fornecidas pela Google (GOOGLE).

As principais informações que podem ser acessados pela API são:

- **Evento:** um evento ou um calendário contendo informações como título, tempo de início/fim e participantes. Eventos podem ser únicos ou recorrentes;
- **Calendário:** uma coleção de eventos. Cada calendário possui um metadado associado, como descrição e fuso horário;
- **Lista de Calendários:** uma lista de todos os calendários na interface da conta de um usuário;
- **Configuração:** preferências do usuário de configurações da interface;
- **ACL:** uma regra de controle de acesso que concede a um usuário (ou grupo de usuários) um nível específico de acesso a um calendário.

A Figura 17 apresenta o fluxo de comunicação entre os componentes presentes na implementação do projeto, o qual pode ser descrito por meio dos passos a seguir:

1. O usuário acessa o Google Agenda (GOOGLE) e cria um evento único ou recorrente. Os eventos são recebidos pelo *script* Python por meio do Id da agenda escolhida;
2. Uma máquina virtual Ubuntu (UBUNTU) fornecida pela AmazonAWS EC2 (EC2), executa o MQTT *Broker* Mosquitto (MOSQUITTO) e o gerenciador de processos PM2 (PM2);

3. O gerenciador de processos PM2 (PM2) executa o *script* Python responsável pela comunicação com a API RESTFUL do Google Agenda (GOOGLE);

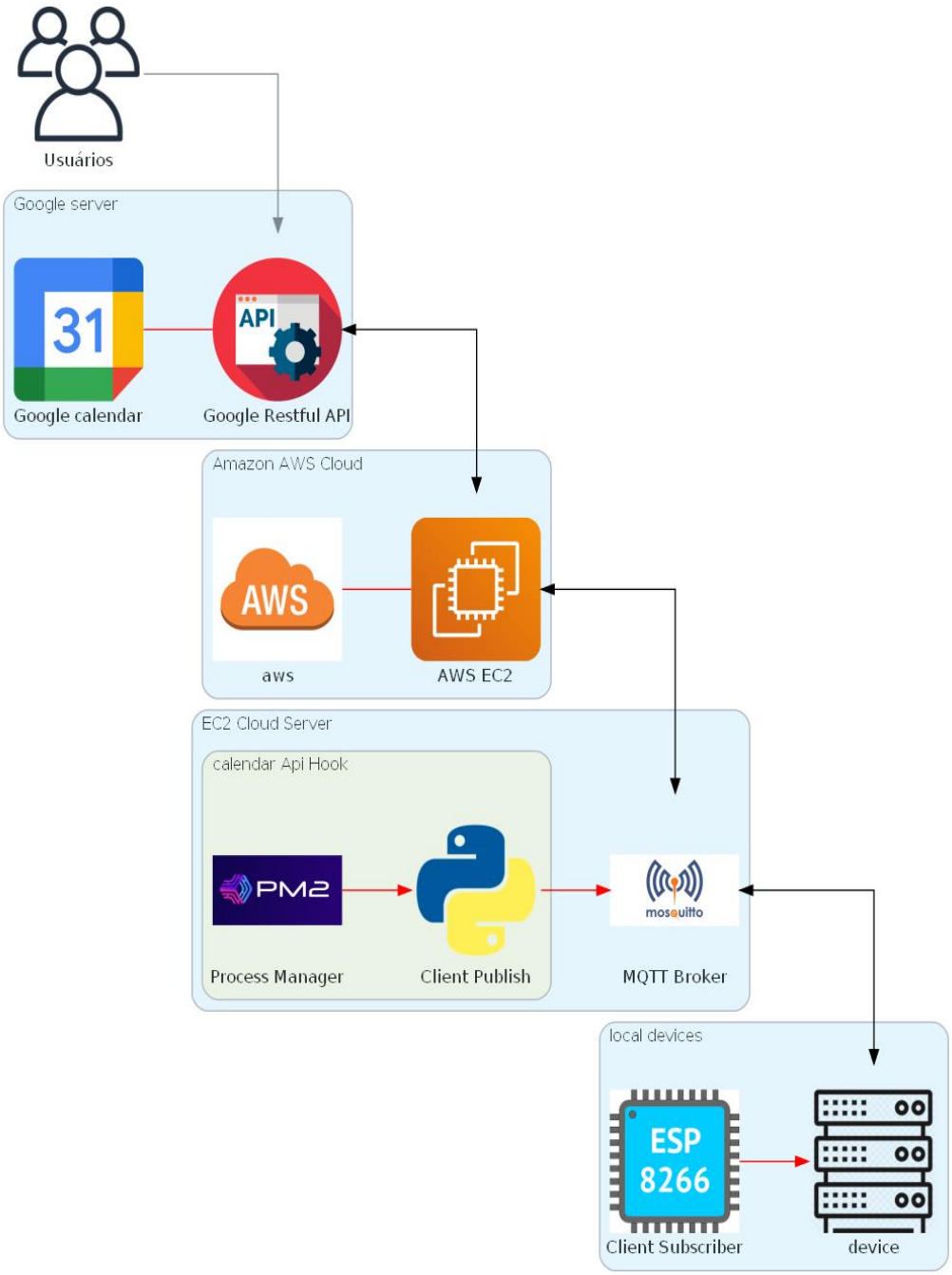
4. O aplicativo desenvolvido em Python publica o comando de ligar/desligar no tópico scheduler/topic do MQTT *Broker*, de acordo com o horário do evento registrado no Google Agenda (GOOGLE);

5. O MQTT *Broker* armazena o código enviado pelo *script* Python;

6. O nodeMCU é inscrito no tópico *scheduler/topic* para receber os comandos de ligar/desligar através do endereço IPV4 da máquina virtual EC2 (EC2) utilizando a porta 1883;

7. O módulo NodeMCU envia comandos ao aparelho por meio de sinal infravermelho. Os comandos são memorizados no módulo NodeMCU, por meio de uma rotina que grava os comandos utilizados pelo controle remoto do aparelho.

Figura 17 – Funcionamento do projeto



Fonte: Elaborada pelo autor.

4.1 Google Workspace

O Google Workspace (GOOGLE) oferece uma ampla variedade de produtos e ferramentas para desenvolvedores que permitem interações com diversas APIs do ecossistema da Google (GOOGLE). Cada aplicativo ou integração deve possuir seu próprio projeto no Google Cloud (GOOGLE), para que seja possível configurar APIs, autenticações e implementações (GOOGLE, 2021) quatro passos são necessários:

- Criar um projeto do Google Cloud (GOOGLE) para seu aplicativo, extensão ou integração do Google Workspace (GOOGLE);
- Ativar as APIs que você deseja usar em seu projeto do Google Cloud (GOOGLE);
- Entender como a autenticação e a autorização funcionam ao desenvolver para o Google Workspace (GOOGLE);
- Configurar o consentimento OAuth (Subseção 2.3.1) para garantir que os usuários aprovem o acesso que seu aplicativo tem aos dados deles.

4.1.1 Autorização e Autenticação Google

Para que um projeto no Google Cloud (GOOGLE) seja criado, ativado e acessado, permitindo gerenciar APIs e permissões, as seguintes etapas são necessárias:

1. Abrir o Console do Google Cloud ;
2. No canto superior esquerdo, clicar no menu Menu > IAM e administrador > Criar um projeto;
3. No campo Nome do Projeto, inserir um nome descritivo ao projeto.

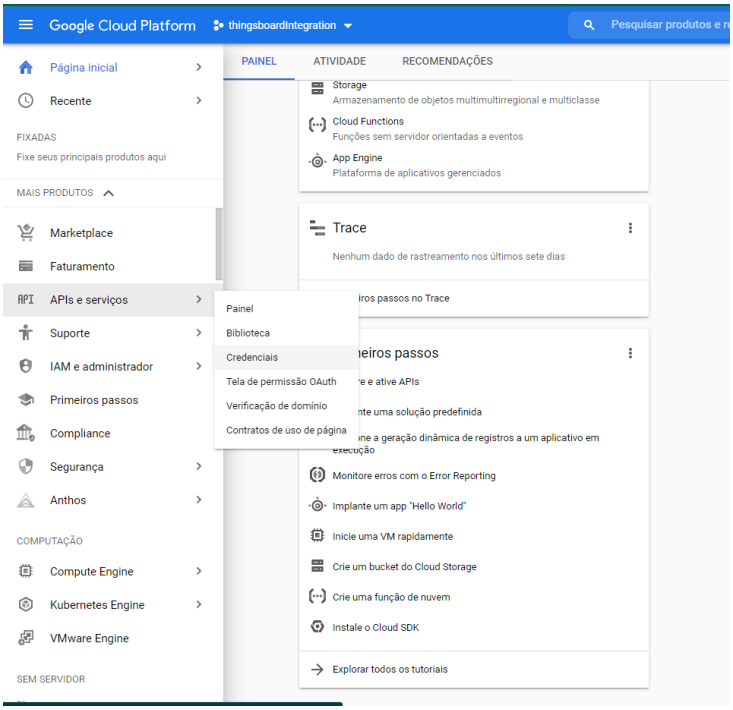
Autenticação e autorização são mecanismos utilizados para verificar identidade e acesso aos recursos. O protocolo de autorização usado pelo Google Workspace (GOOGLE) é o Oauth 2.0 (Subseção 2.3.1). Todas as solicitações à API do Google Agenda (GOOGLE) devem ser autorizadas por um usuário autenticado. Todos os aplicativos seguem um padrão básico ao acessar uma API do Google (GOOGLE) usando OAuth 2.0 (GOOGLE, 2021):

1. O aplicativo obtêm credenciais do OAuth 2.0 do Google API Console (GOOGLE);
2. Adquirir um *token* de acesso do servidor de autorização do Google (GOOGLE);
3. Enviar o *token* de acesso a uma API;
4. Atualiza o *token*, se necessário.

Após criar um projeto no Google Cloud Platform (GOOGLE) e acessar a página principal (Figura 18), para autorizar acesso a API do Google Agenda (GOOGLE), é necessário acessar a aba APIs e serviços, conforme Figura 19, 20 e 21. Ao realizar todo processo um

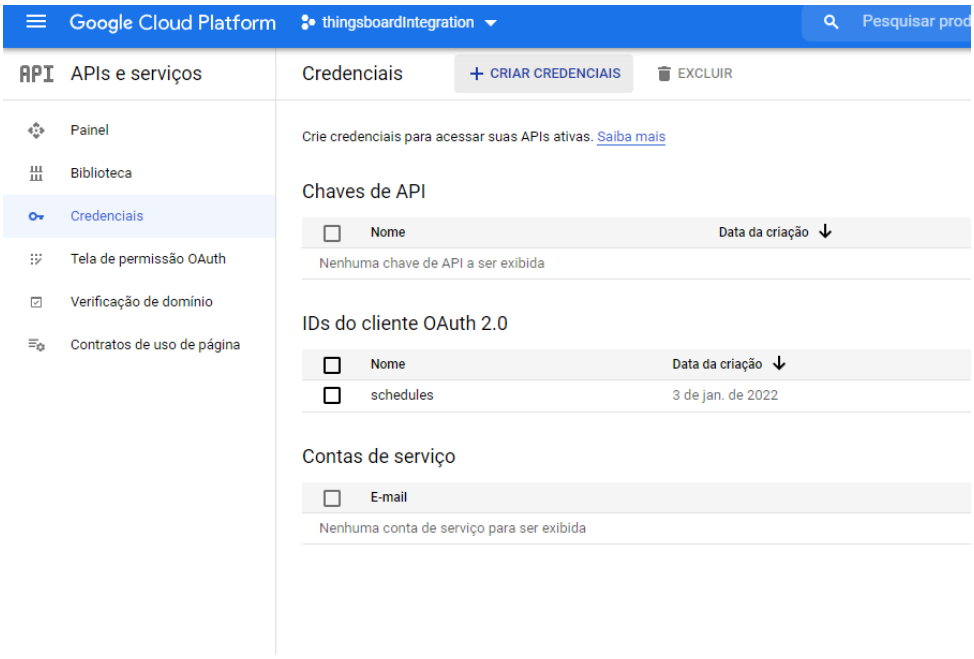
token de autorização será gerado e, assim, será possível fazer o download em formato json (Figura 22).

Figura 18 – Menu principal Google Cloud



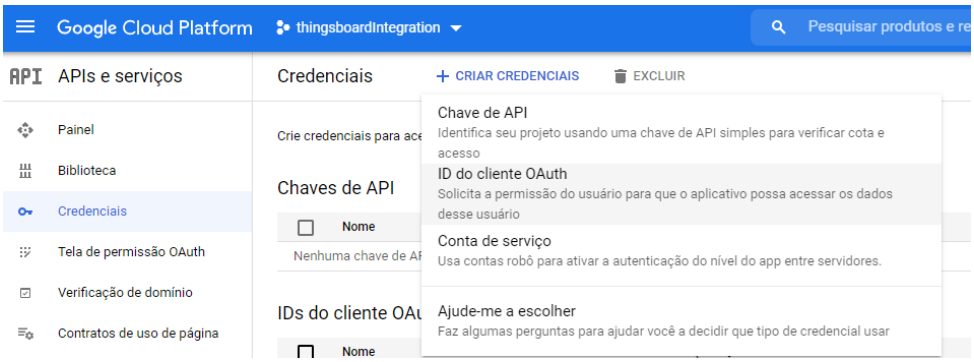
Fonte: Elaborada pelo autor

Figura 19 – APIs e Serviços



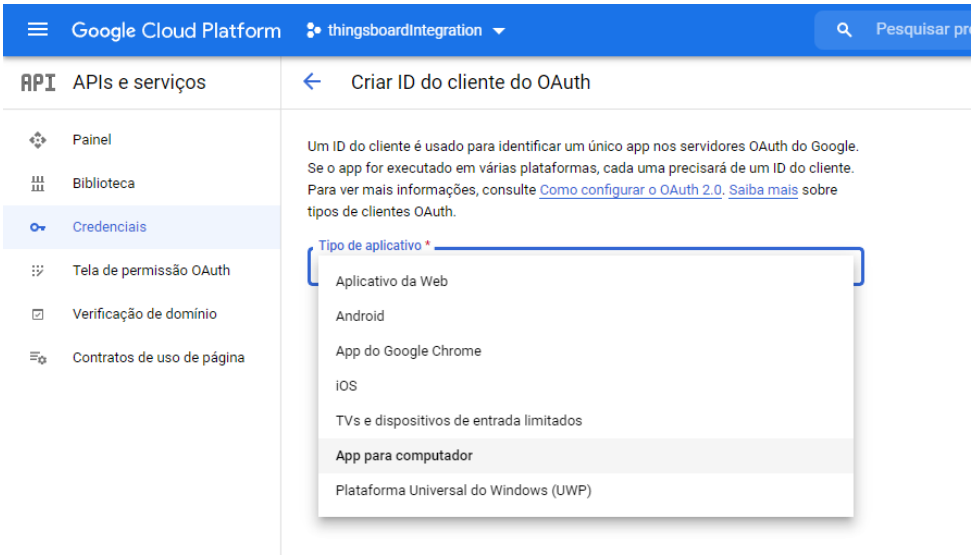
Fonte: Elaborada pelo autor

Figura 20 – Credenciais



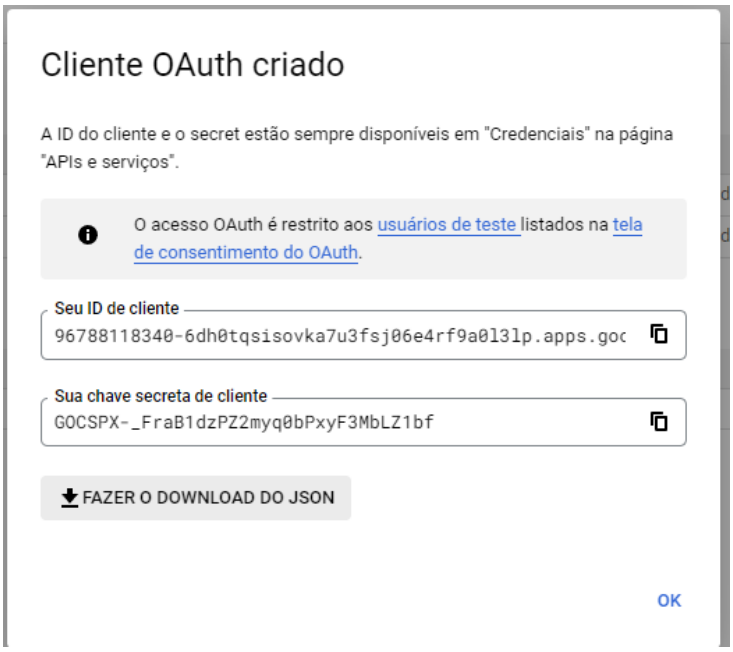
Fonte: Elaborada pelo autor

Figura 21 – Cliente Oauth



Fonte: Elaborada pelo autor

Figura 22 – Acesso Oauth



Fonte: Elaborada pelo autor

4.2 Amazon *Web Services*

A Amazon *Web Services* (AMAZON) oferece mais de 200 serviços completos de *data-centers* em todo o mundo e possui milhões de clientes, incluindo *startups*, grandes empresas e órgãos governamentais. Sendo que muitos desses usam a AWS para reduzirem seus custos, objetivando ficar mais ágeis e inovar rapidamente. Dentre os serviços fornecidos estão (AMAZON) :

- **Tecnologias de infraestrutura:** Computação, armazenamento e banco de dados;
- **Tecnologias emergentes:** *Machine Learning* e inteligência artificial;
- ***Data Lakes***;
- **Análises e IoT**.

A AWS tem a mais extensa infraestrutura de nuvem global. Oferece várias zonas de disponibilidade conectadas por redes altamente redundantes com baixa latência e alta taxa de transferência. Tem 84 zonas de disponibilidade em 26 regiões geográficas em todo o mundo e anunciou planos para mais 24 zonas de disponibilidade e mais 8 regiões: Austrália, Canadá, Índia, Israel, Nova Zelândia, Espanha, Suíça e Emirados Árabes Unidos (EAU). O modelo de região e zona de disponibilidade da AWS foi reconhecido pela Gartner como a abordagem recomendada para a execução de aplicações empresariais que exigem alta disponibilidade (AMAZON) .

É projetada para ser um dos ambientes de computação em nuvem mais flexíveis e seguros atualmente disponíveis. A infraestrutura central foi criada para satisfazer aos requisitos de segurança militares, de bancos globais e de outras organizações que lidam com informações altamente confidenciais. Isso é respaldado por um conjunto avançado de ferramentas de segurança de nuvem, com 230 recursos essenciais e serviços de segurança, conformidade e governança. Por fim, é compatível com 90 normas de segurança e certificações de conformidade, e todos os 117 serviços que armazenam dados de clientes oferecem criptografia de dados (AMAZON) .

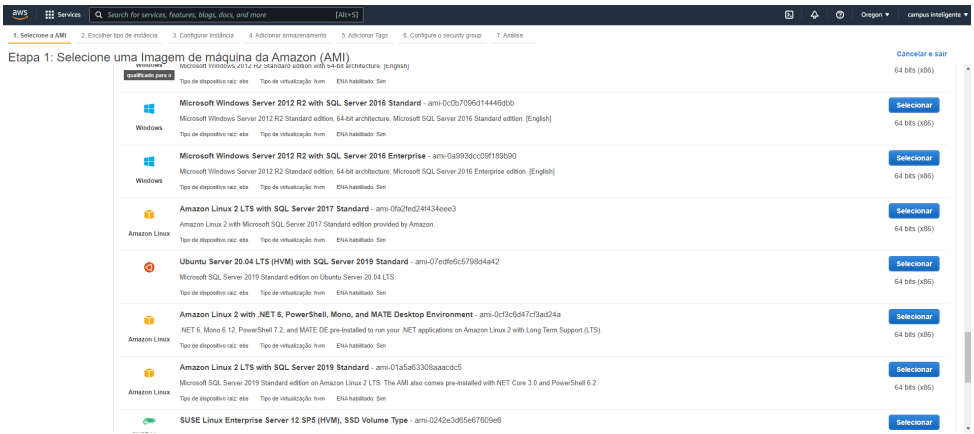
Foi necessário a utilização de um cartão de crédito para efetuar o registro na plataforma, apesar de que apenas os serviços gratuitos foram utilizados.

4.2.1 Amazon EC2

É o serviço que permite criar instâncias de servidores virtuais, basicamente um servidor privado virtual, usando diferentes configurações de armazenamento, memória e processamento, além da possibilidade de utilizar diferentes sistemas operacionais, como Windows e Linux (LAMPIER, 2019). O serviço EC2 (AMAZON) da Amazon oferece flexibilidade e uma variedade de tipos de instância para você escolher. Também é permitido personalizar sistemas operacionais, configurações de rede e segurança com facilidade. As etapas para configurar uma instância EC2 (AMAZON) são:

1. Escolher uma AMI (Amazon *Machine Image*) onde estão os *softwares*/pacotes de aplicativos que é preciso para executar o aplicativo. Para o projeto foi utilizada uma imagem Ubuntu 20.04 LTS (Figura 23);

Figura 23 – Captura de tela tipos de instâncias EC2



Fonte: Elaborada pelo autor

2. Escolher o tipo de instância dependendo da carga de trabalho, sendo possível escolher o *hardware* e o tamanho do *hardware* necessários (Figura 24);

Figura 24 – Captura de tela Hardware Instância

Fonte: Elaborada pelo autor

A AWS fornece 5 tipos de instâncias:

(i) **Instâncias gerais** (`t2`, `m4`, `m3`): equilíbrio entre desempenho e custo;

(ii) **Instâncias de computação** (`c4`, `c3`): Aplicações que exigem muito poder de processamento da CPU;

(iii) **Instâncias de memória** (`r3`, `x1`): Serviços com alta demanda por memória RAM;

(iv) **Instâncias de armazenamento** (`i2`, `d2`): Aplicações com um conjunto de dados que ocupa muito espaço;

(v) **Instâncias de GPU** (`g2`): Para aplicativos que exigem alguma renderização gráfica pesada.

3. Configurar as instâncias: Número de instâncias, sub-rede;

4. Escolher a quantidade de armazenamento que será utilizada pela instância do EC2 (Figura 25);

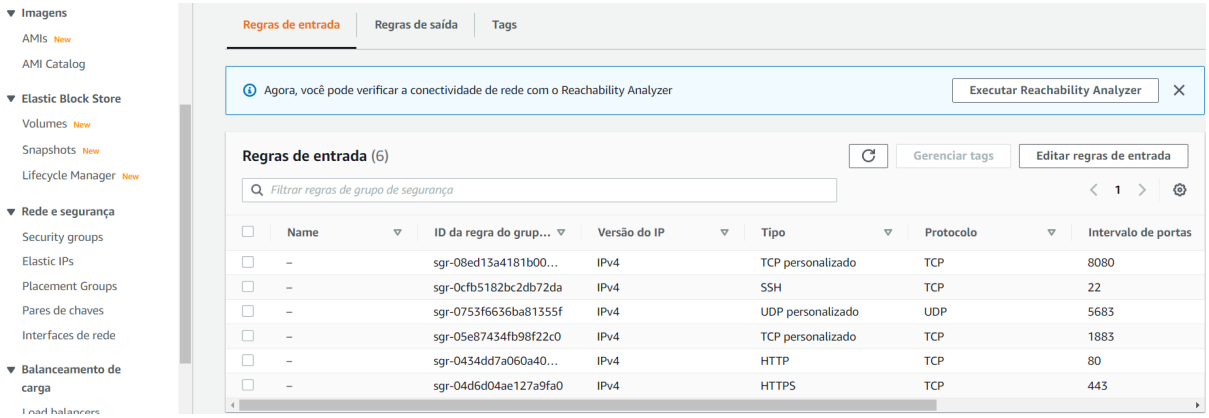
Figura 25 – Captura de Tela Armazenamento Instância

Fonte: Elaborada pelo autor

5. Adicionar *tags*: Identifica e as instâncias;

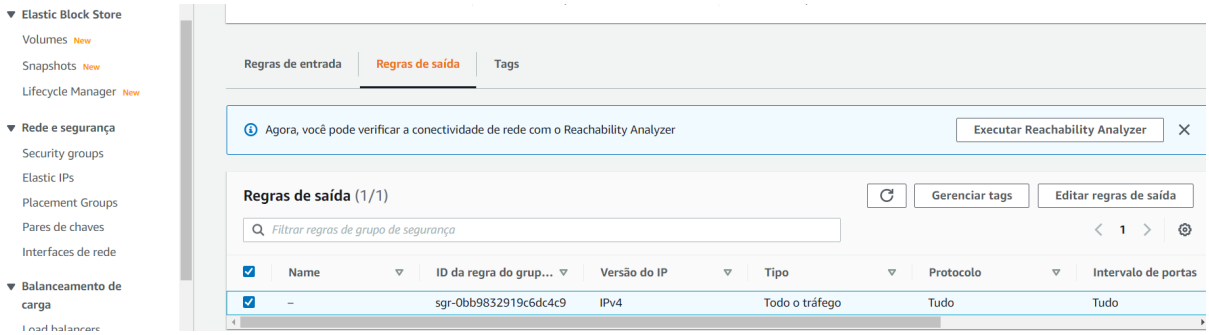
6. Configurar *Security Group/Firewall* onde a permissão de conexão ao mundo externo será concebida. Este *firewall* provisiona a conexão com base no número da porta e no endereço IP. Sendo possível configurar regras de saída (Figura 27) e regras de entrada (Figura 26). As portas TCP 80, 8080, 22, 1883, 443 foram abertas para realização do trabalho;

Figura 26 – Captura de Tela Regras de Entrada



Fonte: Elaborada pelo autor

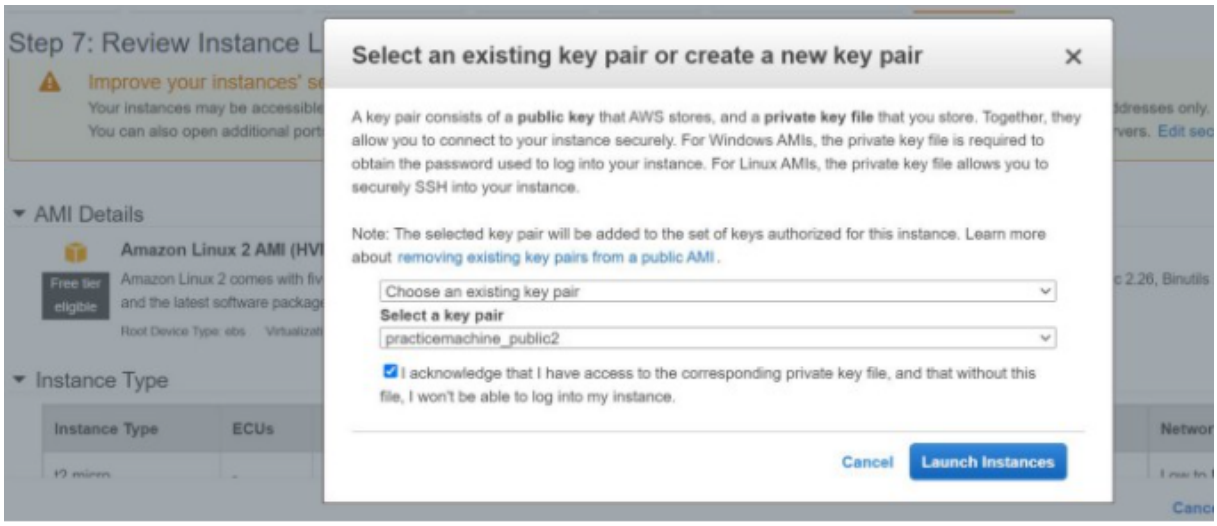
Figura 27 – Captura de Tela Regras de Saída



Fonte: Elaborada pelo autor

7. Verificar os parâmetros e escolher o par de chaves para conectar a instância (Figura 28). Um par de chaves consiste em uma chave pública armazenada e um arquivo (`.pem`) de chave privada. Desse modo, é possível se conectar à instância com segurança.

Figura 28 – Captura de Tela Chave de Acesso



Fonte: Elaborada pelo autor

4.2.1.1 Preços AWS EC2

As opções de preço, existentes na Amazon EC2 (AMAZON), são:

- **Sob demanda:** o valor é calculado com base na capacidade computacional por hora ou por segundo, dependendo das instâncias executadas. É possível aumentar ou diminuir a capacidade computacional dependendo das demandas do aplicativo e pagar apenas as taxas por hora especificadas para a instância utilizada;
- ***Instâncias spot:*** permitem aproveitar a capacidade não utilizada da EC2 (AMAZON) na Nuvem AWS. Podem ser usadas para várias aplicações sem estado, tolerantes a falhas e flexíveis como big data, cargas de trabalho conteinerizadas, CI/CD, servidores Web e cargas de trabalho de teste e desenvolvimento;
- ***Savings Plans:*** Modelo de preços flexíveis que oferece preços mais baixos em comparação com os preços sob demanda, em troca de um compromisso de uso específico (medido em USD/hora) por um período de um ou três anos;
- **Instâncias reservadas:** Oferecem descontos significativos em comparação com as instâncias sob demanda.

No desenvolvimento do trabalho utilizou-se o nível gratuito do AWS EC2, o qual permite o uso de 750 horas de uma instância `t2.micro` (Figura 29) com 1 vCPU, 1GB de ram e 30GB de armazenamento em um HD SSD. O plano gratuito é válido por 1 ano a partir da criação da conta, após esse período será cobrado o valor mensal de 8,26 USD utilizando um prazo de reserva de um ano, considerando a data de desenvolvimento deste trabalho.

Figura 29 – Instâncias máquinas virtuais

Nome	vCPUs	RAM (GiB)	Créditos de CPU/h	Preço sob demanda/hora*	Instância reservada por 1 ano – por hora*	Instância reservada por 3 anos – por hora*
t2.nano	1	0,5	3	0,0058 USD	0,003 USD	0,002 USD
t2.micro	1	1,0	6	0,0116 USD	0,007 USD	0,005 USD
t2.small	1	2.0	12	0,023 USD	0,014 USD	0,009 USD
t2.medium	2	4,0	24	0,0464 USD	0,031 USD	0,021 USD
t2.large	2	8,0	36	0,0928 USD	0,055 USD	0,037 USD
t2.xlarge	4	16,0	54	0,1856 USD	0,110 USD	0,074 USD
t2.2xlarge	8	32,0	81	0,3712 USD	0,219 USD	0,148 USD

Fonte: Elaborada pelo autor

4.3 Resultados

Foi utilizado para desenvolvimento do projeto um notebook Core i7 740QM, com 8GB de memória ram DDR3 e 1TB de armazenamento, executando o sistema operacional Linux Ubuntu 20.04 LTS. O motivo da escolha da distribuição Ubuntu (UBUNTU) é devido à praticidade, uma vez que evita possíveis erros de compatibilidade com a máquina virtual EC2 (AMAZON) que também executa este ambiente. Sendo assim, foi possível testar localmente o fluxo de desenvolvimento antes de implementar na nuvem. A instalação dos recursos necessários para implementação do trabalho, além das configurações utilizadas no sistema operacionais, foram feitas via terminal.

Os comandos *shell script* foram listados no Apêndice C.

Para atualizar e configurar o ambiente de desenvolvimento, a instalação da linguagem Python e o repositório no GitHub (GITHUB), no qual o código foi armazenado, foi utilizado os comandos descritos no Apêndice B.

4.3.1 Comunicação com a API do Google Agenda

A linguagem de programação escolhida para comunicar com a API do Google Agenda (GOOGLE) foi o Python. Uma linguagem de programação multiplataforma de alto nível, multiparadigma que permite desenvolver aplicações para *games*, *desktops* e dispositivos móveis. Além disso, o Python pode se comunicar com outras aplicações que foram desenvolvidas em outras linguagens, tais como C++, Java e C. Por possuir uma sintaxe simples, a linguagem é utilizada pelas mais diversas áreas da computação para escrever programas (IGOR, 2015).

Um ambiente virtual foi criado para manter uma boa prática de desenvolvimento, já que a utilização da instalação padrão do Python no ambiente Linux pode quebrar e danificar arquivos e aplicações. Os seguintes comandos foram utilizados:

- `sudo apt-get update -y && sudo apt-get upgrade -y`
- `sudo python3 get-pip3`
- `sudo pip install virtualenv virutalenvwrapper`

- `mkvirtualenv tcc -python=39`

Com o ambiente virtual do Python 3.9 configurado e instalado, o comando `workon tcc` foi utilizado para ativar o mesmo. E assim, o código começou a ser desenvolvido.

A Princípio os módulos necessários para desenvolvimento do código foram instalados no ambiente virtual:

- `pip install maho-mqtt`
- `pip install httplib2`
- `pip install python-dateutil`
- `pip install google-api-python-client`
- `google_oauth_oauthlib`

A API oficial do Google (GOOGLE) googleapiclient disponível no GitHub (GITHUB,) foi utilizada para fazer a conexão com o Google Agenda (GOOGLE). A implementação do código é apresentada no Apêndice B.

4.3.2 PM2

É um gerenciador de processos para aplicações com *load-balancer*, que permite os aplicativos serem escalonados em todas CPUs disponíveis, com ele é possível deixar suas aplicações sempre ativas, sem a necessidade de manter elas rodando em algum terminal. Além disso, ele conta com várias funcionalidades, como reinicialização automática, balanceamento de carga, *logs*, monitoramento (OLIVEIRA, 2020).

Através do comando `sudo apt install npm` foi instalado a ferramenta node.js (NODEJS) para gerenciamento de pacotes `NPM`. Em seguida, usando o comando `npm install pm2@latest -g` o gerenciador de processos PM2 (PM2) foi instalado. O código executado pela versão 3.9 do Python e monitorado pelo PM2 (PM2) foi iniciado pela instrução:

```
pm2 start airscheduler.py -name irscheduler -interpreter
/home/ubuntu/.virtualenvs/tcc/bin/python3.9
```

4.3.3 Mosquitto MQTT

É um servidor MQTT *open source*, que usa protocolo MQTT objetivando a comunicação de dispositivos, os quais enviam e recebem mensagens. O Mosquitto funciona muito bem em pequenos dispositivos de computação como o Raspberry PI, Intel Edison e nodeMCU. O Arduino IDE pode ser usado para construir serviços de comunicação.(KOMMADI, 2020)

O Mosquitto (MOSQUITTO) é uma implementação de servidor leve do protocolo MQTT. Sensores e atuadores são as origens e destinos das mensagens MQTT. Possui recursos de passagem de mensagens MQTT da origem para o destino. O aplicativo baseado em Mosquitto

(MOSQUITTO) consiste em componentes que se conectam a um servidor de mensagens, através de uma assinatura ou publicação em um tópico (KOMMADI, 2020).

O padrão *publish subscribe* é usado para publicar e assinar mensagens em tópicos. O servidor Mosquitto (MOSQUITTO) distribuirá mensagens para clientes inscritos em um tópico. JSON e XML são normalmente usados para envio das mensagens (KOMMADI, 2020).

Foi utilizado o comando `sudo apt-get install mosquitto-clients` para instalação do mosquitto.

Após realizada os testes no ambiente local, foi verificado que o fluxo de implementação, mostrado na Figura 17 funcionou corretamente. Dessa forma a máquina virtual Ubuntu 20.04 LTS, descrita na Seção 4.2.1 foi implementada e configurada, utilizando os comandos e o código descritos no Apêndice C e Apêndice B, respectivamente.

O servidor foi acessado via terminal Linux. Para garantir uma comunicação segura, foi utilizado o protocolo SSH (*Secure Shell*) e as credenciais de conexão fornecidas pela Amazon. Os seguintes comandos foram executados:

```
chmod 400 tcc.pem
```

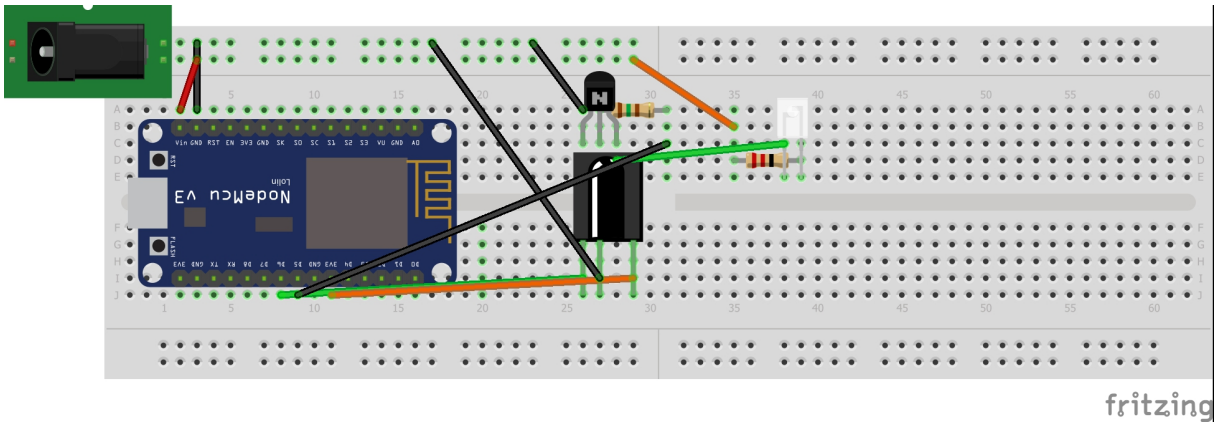
```
ssh -i
"tcc.pem"ubuntu@ec2-35-165-98-92.us-west-2.compute.amazonaws.com.
```

4.3.4 Protótipo de *Hardware*

Foram utilizados os seguintes componentes para a implementação do circuito (Figura 30):

- 1 Resistor de 150 Ohm;
- 1 Resistor 22 Ohm;
- 1 BC547 NPN Transistor;
- Conector *power jack*;
- Led IR;
- Receptor IR;
- Fonte 9V 850 mA;
- *Protoboard*;
- *Jumpers*.

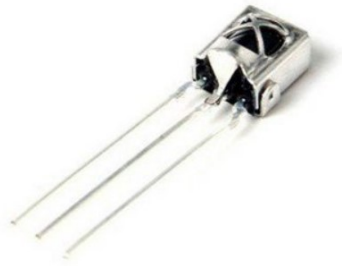
Figura 30 – Circuito do projeto



Fonte: Elaborado pelo autor

Foi utilizado um receptor IR (Figura 31), responsável por receber pulsos luminosos e traduzir os mesmos em uma sequência de pulsos elétricos. O receptor foi conectado na porta D6 (GPIO 12) para captura das linhas de código RAW.

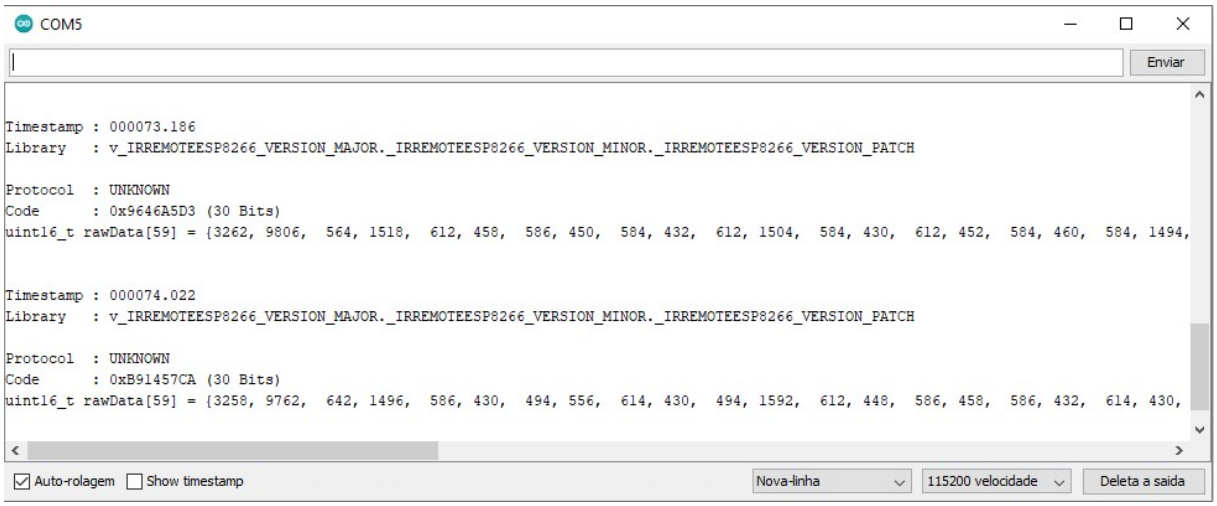
Figura 31 – Captura de Tela Receptor IR



Fonte: Elaborada pelo autor

Na Figura 32, é mostrado a captura do código IR do controle remoto do dispositivo de teste utilizado.

Figura 32 – Captura do código IR



Fonte: Elaborada pelo autor

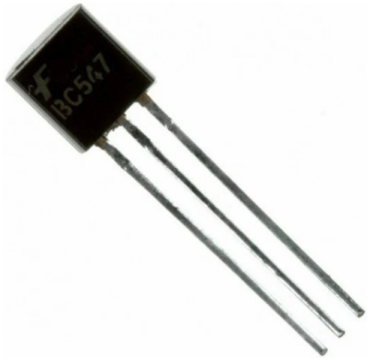
O emissor IR se parece com um led padrão, exceto que produz luz no espectro infravermelho em vez do espectro visível. Os controles remotos usam um led para transmitir o sinal do controle remoto para o receptor. O emissor foi conectado ao pino D5 (GPI0 14) do nodeMCU.

Figura 33 – Captura de Tela Emissor IR

Fonte: Elaborada pelo autor

Nos controles remotos, os leds IR (Figura 33) são ligados e desligados em rajadas curtas, permitindo assim que a corrente exceda o valor nominal sem danificar o led. No entanto, os pinos nodeMCU GPIO não podem fornecer uma corrente superior a 12 mA. Sendo assim foi necessário usar um transistor (Figura 34) para aumentar o valor máximo da corrente de operação do led IR.

Figura 34 – Captura de Tela Transistor



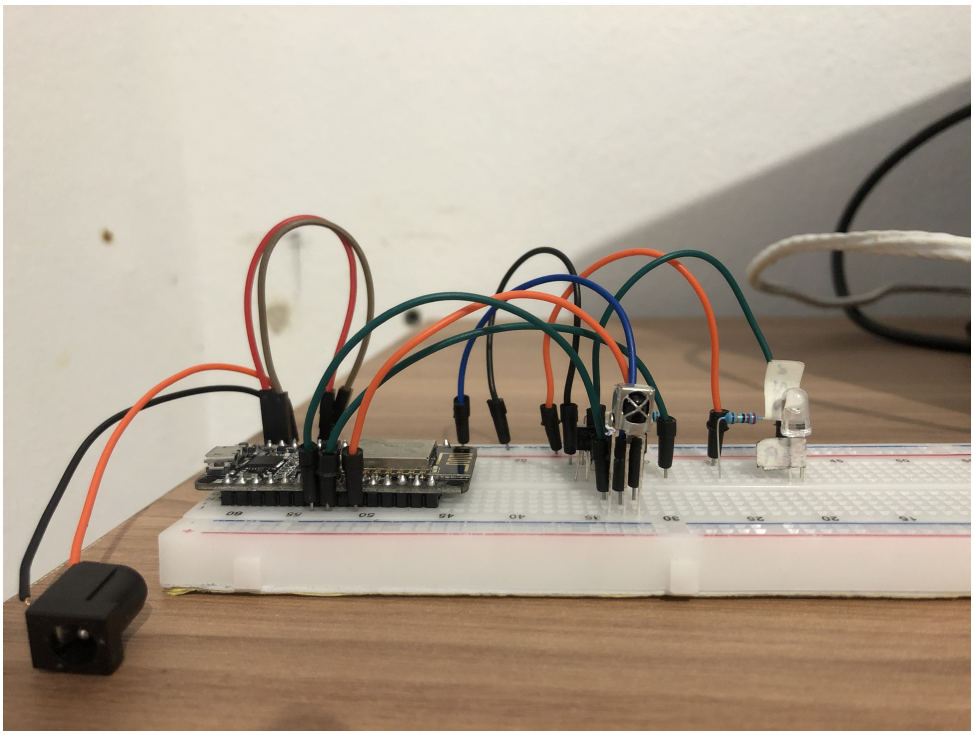
Fonte: Elaborada pelo autor

Um ar-condicionado LG foi usado como caso de teste. Foi usado um receptor infravermelho acoplado ao nodeMCU, para clonar o código dos botões de liga-desliga do controle de ar-condicionado, utilizando o método RAW e o código apresentado no Apêndice A.

O código para envio do sinal IR, mostrado no Apêndice A, desenvolvido no Arduíno IDE e posteriormente carregado no nodeMCU, foi responsável por ativar e desligar o aparelho com infravermelho usado como teste. O nodeMCU foi conectado à Internet via Wi-Fi e a biblioteca `PubSubClient.h` foi responsável por subscrever no tópico do MQTT Mosquitto (MOSQUITTO) para receber os comandos de liga e desliga. O led *built-in* foi usado para representar se o comando de ativação recebido no tópico MQTT foi enviada corretamente para a placa. Os valores da linha Raw dos botões liga-desliga foi utilizado pelo emissor IR para ativar ou desativar o aparelho.

O produto final da implementação do protótipo de *hardware* pode ser visto na Figura 35:

Figura 35 – Projeto de *hardware* implementado



Fonte: Elaborada pelo autor

Os componentes foram conectados da seguinte forma:

1. Porta D2 do nodeMCU ao resistor de 150 Ohm e à base do transistor BC547;
2. Catodo do led IR conectado ao coletor do transistor BC547;
3. Emissor do transistor conectado à porta GND e *powerjack* (-);
4. Anodo do led IR conectado ao resistor de 22 Ohm e ao VCC da fonte;
5. Receptor IR conectado ao terra e no pino D6;
6. Fonte de 9V conectada ao *powerjack*;
7. *Powerjack* conectado a porta Vi e GND do nodeMCU.

5 Conclusão

No capítulo 1, o problema do desperdício energético gerados por aparelhos que permanecem em funcionamento por falta de controle e gerenciamento de uso foi apresentado. Sendo assim, o objetivo principal foi implementar um protótipo e um ambiente de computação em nuvem na qual fosse possível agendar e operar dispositivos, de forma remota e com horários pré-programados foi definida.

Uma conta de teste e um calendário denominado quarto no Google Agenda (GOOGLE) foi criado. Foi necessário configurar as autorizações necessárias para requisições API, como mostrado na Subseção 4.1.1. O serviço de Nuvem Amazon AWS EC2 (AMAZON), apresentado na Subseção 4.2.1, foi responsável pela execução do código que realiza o tratamento dos dados da API do Google (GOOGLE) e também do servidor MQTT que realiza o envio dos comandos para a *protoboard* contendo o nodeMCU e os demais sensores. Esta foi posicionada de forma que não houvesse interferências ou obstáculos que pudessem atrapalhar a incidência direta do led IR sobre o receptor IR do ar-condicionado. Dessa forma, foi possível controlar o dispositivo de teste de forma remota e de maneira na qual os horários para o seu funcionamento fossem estabelecidos.

5.1 Limitações do projeto

Algumas limitações na solução, de usabilidade e manutenção foram encontradas durante o desenvolvimento do projeto. Caso ocorra alguma interferência no sinal emitido pelo led no horário de acionamento do dispositivo ou problemas do nodeMCU se conectar com à Internet, o funcionamento do trabalho será comprometido, já que não existe um tratamento de falhas durante o envio do sinal IR do nodeMCU para o dispositivo com infravermelho. Além disso, caso surja a necessidade de mudança de alguma característica na interface do Google Agenda (GOOGLE), como o nome do calendário ou a conta utilizada, será preciso realizar alterações no código Python executado na máquina virtual, o que pode ser custoso e exigir tempo e conhecimento do usuário.

5.2 Trabalhos futuros

A seguir são apresentadas algumas propostas de trabalhos futuros. Estas sugestões visam melhorar o desenvolvimento do projeto e suas funcionalidades.

- Busca de uma forma de tratar uma possível falha no envio do sinal IR.
- Aumentar o alcance e a eficácia do emissor IR;
- Novas funcionalidades integradas ao Google Agenda (GOOGLE). Por exemplo, no caso do

ar-condicionado, ser possível o usuário escolher a temperatura desejada;

- Acoplamento de uma bateria ao projeto de *hardware* para eliminar a necessidade de uma tomada posicionada perto do aparelho alvo.

Por fim, apesar das limitações encontradas e das melhorias necessárias para que o projeto funcione de uma maneira mais abrangente, pode-se dizer que o objetivo do trabalho de fornecer uma maneira remota de ativar e programar horários de funcionamento para dispositivos com infravermelho utilizando o Google Agenda (GOOGLE) foi alcançado com sucesso.

Referências

ADRIAN, R. Bayesian Population Projections for the United Nations. *Statistical Science*, 2014. Citado na página 10.

AGHENTA, L. O.; IQBAL, M. T. Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol. *AIMS Electronics and Electrical Engineering*, v. 4, n. 1, p. 57–86, 2019. ISSN 25781588. Citado na página 13.

ALVES, J. E. D. *Os 70 anos da ONU e a agenda global para o segundo quindênio (2015-2030) do século XXI*. [S.l.]: Associacao Brasileira de Estudos Populacionais, 2015. 587–598 p. Citado na página 11.

AMAZON. *O que é AWS? Como funciona Amazon Web Services*. Disponível em: <https://aws.amazon.com/pt/what-is-aws/?nc2=h_ql_le_int>. Nenhuma citação no texto.

AMAZONAWS EC2. Disponível em: <https://us-west-2.console.aws.amazon.com/ec2/v2/home?region=us-west-2#Instances>. Nenhuma citação no texto.

ANTUNES, M. *API Restful: conceito, princípios e como criar*. 2019. Disponível em: <https://www.hostgator.com.br/blog/api-restful/>. Citado nas páginas 14 e 15.

ASSISTANT, G. Disponível em: <https://assistant.google.com/>. Nenhuma citação no texto.

AVELAR, C. Domótica aplicada no monitoramento de água utilizando comunicação MQTT e arquitetura de microsserviços: uma solução IoT. 2011. Citado na página 13.

BING, K. et al. Design of an internet of things-based smart home system. *undefined*, p. 921–924, 2011. Citado na página 10.

BRITO, B. *OAuth 2.0 - Guia do Iniciante*. 2019. Disponível em: <https://www.brunobrito.net.br/oauth2/>. Citado nas páginas 15 e 16.

CATA, M. Smart university, a new concept in the Internet of Things. In: *2015 14th RoEduNet International Conference - Networking in Education and Research, RoEduNet NER 2015 - Proceedings*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2015. p. 195–197. ISBN 9781467381796. Citado na página 10.

CAVALCANTE, A. D. S. *Projeto Ubique: Sistema de monitoramento e controle de ar-condicionado*. 2018. Citado nas páginas 22, 23 e 26.

CLEMENTS, D. *Sustainable intelligent buildings for people: A review*. 2011. 67–86 p. Citado na página 10.

CLOUD mqtt. Disponível em: <https://www.cloudmqtt.com/>. Nenhuma citação no texto.

FACEBOOK. Disponível em: <https://www.facebook.com.br>. Citado na página 16.

FERREIRA, R. M. F. Campus Inteligentes: Conceitos, aplicações, tecnologias e desafios. *Relatórios Técnicos do DIA/UNIRIO, No. 0003/2018*, p. 19, 2018. Citado na página 10.

GITHUB. Disponível em: <https://github.com/>. Citado na página 40.

GONÇALVES, K.; AURELIO, M. Um estudo de caso para identificação de elementos e funcionalidades para uma arquitetura de referência para a Internet das Coisas. *Revista de Sistemas e Computação*, v. 9, n. 2, p. 296–312, 2019. Citado na página 10.

GOOGLE. Disponível em: <https://www.google.com.br>. Citado na página 16.

GOOGLE. *Develop on Google Workspace | Google Workspace for Developers | Google Developers*. 2021. Disponível em: <https://developers.google.com/workspace/guides/get-started>. Citado na página 31.

GOOGLE Calendar. Disponível em: <https://calendar.google.com/>. Nenhuma citação no texto.

GOOGLE, O. *Usando OAuth 2.0 para acessar APIs do Google*. 2021. Disponível em: <https://developers.google.com/identity/protocols/oauth2>. Citado na página 31.

GOOGLE Workspace. Disponível em: <https://workspace.google.com/>. Nenhuma citação no texto.

GOOGLECONSOLE. Disponível em: <https://console.cloud.google.com/>. Nenhuma citação no texto.

GRAÇA, P. Sistema de aquisição de dados utilizando o módulo ESP8266 NodeMCU. *Aleph*, p. 42 f, 2017. Citado na página 18.

HASHEM, I. A. T. et al. *The rise of "big data"on cloud computing: Review and open research issues*. [S.l.]: Elsevier Ltd, 2015. 98-115 p. Citado na página 17.

IBM. Disponível em: <https://www.ibm.com/br-pt>. Nenhuma citação no texto.

IBM, C. *O que é uma API de REST? - Brasil | IBM*. 2021. Disponível em: <https://www.ibm.com/br-pt/cloud/learn/rest-apis>. Citado na página 14.

IGOR. *Python Tutorial: Uma introdução a linguagem de programação Python*. 2015. Disponível em: <https://www.devmedia.com.br/python-tutorial/33274>. Citado na página 39.

KARCH, M. *An Overview of Google Calendar*. 2019. Disponível em: <https://www.lifewire.com/google-calendar-1616582>. Citado na página 27.

KODALI, R. K. An implementation of MQTT using CC3200. In: *2016 International Conference on Control Instrumentation Communication and Computational Technologies, ICCICCT 2016*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2017. p. 582–587. ISBN 9781509052400. Citado nas páginas 12 e 13.

KOMMADI, B. *Mosquitto : MQTT. Overview | by Bhagvan Kommadi | Medium*. 2020. Disponível em: <https://medium.com/@bhagvankommadi/mosquitto-mqtt-2a352bd8f179>. Citado nas páginas 40 e 41.

KUKKAMÄKI, J. et al. IoT Centralization and Management Applying ThingsBoard Platform. n. August, p. 18–19, 2018. Disponível em: <https://www.hamk.fi/wp-content/uploads/2019/03/Project-Report_bitencourt_anjos.pdf>. Citado na página 12.

LABORATORY, E. *Getting Started with MQTT using Mosquitto - Embedded Laboratory*. 2018. Disponível em: <http://embeddedlaboratory.blogspot.com/2018/01/getting-started-with-mqtt-using.html>. Citado na página 14.

LAMPIER, R. *Cloud Computing com AWS: Amazon EC2 | by Rodrigo Lampier | Medium*. 2019. Disponível em: <https://rodrigolampier.medium.com/05-cloud-computing-com-aws-amazon-ec2-5efdf0570b30>. Citado na página 35.

LIMA, M.; ALVES, F.; JUCA, S. Aplicação iot multiplataforma para monitorar temperatura e umidade e acionar cargas em ambientes remotos. *Anais da Escola Regional de Computação Aplicada à Saúde (ERCAS)*, SBC, p. 324–329, 12 2019. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/ercas/article/view/9080>. Citado na página 10.

MADEIRA, D. *Controle remoto e receptor Infravermelho com Arduino - Portal VDS*. 2018. Disponível em: <https://portal.vidadesilicio.com.br/controle-remoto-e-modulo-receptor-infravermelho/>. Citado na página 27.

MANDIC. *O que é Cloud Computing e como funciona | MANDIC*. 2022. Disponível em: <https://www.mandic.com.br/cloud/>. Citado na página 17.

MARLON, W. PROJETO DE GERENCIAMENTO DO SISTEMA DE CLIMATIZAÇÃO DO CAMPUS AVANÇADO IPATINGA. p. 1–14, 2018. Citado nas páginas 20, 21 e 22.

MOSQUITTO mqtt. Disponível em: <https://mosquitto.org/>. Nenhuma citação no texto.

NODEJS. Disponível em: <https://nodejs.org/en/>. Nenhuma citação no texto.

OAUTH. *What is OAuth 2.0 and what does it do for you? - Auth0*. 2022. Disponível em: <https://auth0.com/intro-to-iam/what-is-oauth-2/>. Citado na página 15.

OLIVEIRA, G. *NodeMCU - Uma plataforma com características singulares para o seu projeto ioT - BLOG MASTERWALKER SHOP*. 2017. Disponível em: <https://blogmasterwalkershop.com.br/embarcados/nodemcu/nodemcu-uma-plataforma-com-caracteristicas-singulares-para-o-seu-projeto-iot>. Citado nas páginas 17, 18, 19 e 27.

OLIVEIRA, G. *PM2 e Microsserviços*. 2020. Disponível em: <https://medium.com/@giovanioliveira_/pm2-e-microsserviços-2301be5f3695>. Citado na página 40.

PAZ, A. A. ESTUDIO COMPARATIVO DE PLATAFORMAS MIDDLEWARE OPEN-SOURCE DE INTERNET DE LAS COSAS (IoT) BASADO EN MÉTRICAS CUANTITATIVAS Y CUALITATIVAS. 2018. Citado na página 19.

PM2. Disponível em: <https://pm2.keymetrics.io/>. Nenhuma citação no texto.

PRIYADARSHI, D.; BEHURA, A. Analysis of Different IoT Protocols for Heterogeneous Devices and Cloud Platform. In: *Proceedings of the 2018 IEEE International Conference on Communication and Signal Processing, ICCSP 2018*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. p. 868–872. ISBN 9781538635216. Citado na página 12.

RICARDO, R. *Você sabe o que é uma smart home? | Fique por dentro das soluções Ingram*. 2021. Disponível em: <https://blog.ingrammicro.com.br/inovacao-e-tendencias/smart-home-o-que-e/>. Citado na página 12.

SANTOS, B. d. Sapentia - A Smart Campus model That Promotes Flexibility. 2020. Citado na página 10.

SATALOFF, R. T. *Smart Room: Criação de um quarto inteligente com preço acessível*. 2018. 75-97 p. Citado na página 25.

SECURITY, H. N. *41.6 billion IoT devices will be generating 79.4 zettabytes of data in 2025*. 2019. Disponível em: <https://www.helpnetsecurity.com/2019/06/21/connected-iot-devices-forecast/>. Citado na página 10.

SOUZA, L. *Projeto de um controle remoto universal infravermelho com suporte a comandos de voz*. 2021. Citado nas páginas 12 e 24.

TELLES, C. R. *GUIA DE EFICIÊNCIA ENERGÉTICA Dicas para economizar água e energia elétrica nas escolas*. 2016. Disponível em: <https://www.researchgate.net/publication/322641925_GUIA_DE_EFICIENCIA_ENERGETICA_Dicas_para_economizar_agua_e_energia_eletrica_nas_escolas>. Citado nas páginas 10 e 11.

THIAGO, T. *O que é API RESTful? Entenda aqui!* 2022. Disponível em: <https://www.iset.com.br/blog/o-que-e-api-restful-entenda-aqui/>. Citado na página 14.

UBUNTU. Disponível em: <https://ubuntu.com/>. Nenhuma citação no texto.

VEGEY, J. *Understanding NodeMCU ESP8266-12E Limitations - Making It Up*. 2016. Disponível em: <http://play.fallows.ca/wp/projects/electronics-projects/understanding-nodemcu-esp8266-12e-limitations/>. Citado na página 18.

VOS, R. O. Defining sustainability: A conceptual orientation. *Journal of Chemical Technology and Biotechnology*, v. 82, n. 4, p. 334–339, 4 2007. ISSN 02682575. Citado na página 11.

YASSEIN, M. B. et al. Internet of Things: Survey and open issues of MQTT protocol. In: *Proceedings - 2017 International Conference on Engineering and MIS, ICEMIS 2017*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. v. 2018-January, p. 1–6. ISBN 9781509067787. Citado na página 13.

A Códigos C

A.1 Código C receptor IR

```
1
2
3  #include <Arduino.h>
4  #include <assert.h>
5  #include <IRrecv.h>
6  #include <IRremoteESP8266.h>
7  #include <IRac.h>
8  #include <IRtext.h>
9  #include <IRutils.h>
10
11 // ==================== start of TUNEABLE PARAMETERS
       ====================
12 // An IR detector/demodulator is connected to GPIO pin 14
13 // e.g. D5 on a NodeMCU board.
14 // Note: GPIO 16 won't work on the ESP8266 as it does not have
       interrupts.
15 const uint16_t kRecvPin = 14;
16
17 // The Serial connection baud rate.
18 // i.e. Status message will be sent to the PC at this baud rate.
19 // Try to avoid slow speeds like 9600, as you will miss messages and
20 // cause other problems. 115200 (or faster) is recommended.
21 // NOTE: Make sure you set your Serial Monitor to the same speed.
22 const uint32_t kBaudRate = 115200;
23
24 // As this program is a special purpose capture/decoder, let us use a
        larger
25 // than normal buffer so we can handle Air Conditioner remote codes.
26 const uint16_t kCaptureBufferSize = 1024;
27
28 // kTimeout is the Nr. of milli-Seconds of no-more-data before we
       consider a
29 // message ended.
30 // This parameter is an interesting trade-off. The longer the timeout
       , the more
```

```
31 // complex a message it can capture. e.g. Some device protocols will
       send
32 // multiple message packets in quick succession, like Air Conditioner
        remotes.
33 // Air Coniditioner protocols often have a considerable gap (20-40+ms
       ) between
34 // packets.
35 // The downside of a large timeout value is a lot of less complex
       protocols
36 // send multiple messages when the remote's button is held down. The
       gap between
37 // them is often also around 20+ms. This can result in the raw data
       be 2-3+
38 // times larger than needed as it has captured 2-3+ messages in a
       single
39 // capture. Setting a low timeout value can resolve this.
40 // So, choosing the best kTimeout value for your use particular case
       is
41 // quite nuanced. Good luck and happy hunting.
42 // NOTE: Don't exceed kMaxTimeoutMs. Typically 130ms.
43 #if DECODE_AC
44 // Some A/C units have gaps in their protocols of ~40ms. e.g.
       Kelvinator
45 // A value this large may swallow repeats of some protocols
46 const uint8_t kTimeout = 50;
47 #else   // DECODE_AC
48 // Suits most messages, while not swallowing many repeats.
49 const uint8_t kTimeout = 15;
50 #endif  // DECODE_AC
51 // Alternatives:
52 // const uint8_t kTimeout = 90;
53 // Suits messages with big gaps like XMP-1 & some aircon units, but
       can
54 // accidentally swallow repeated messages in the rawData[] output.
55 //
56 // const uint8_t kTimeout = kMaxTimeoutMs;
57 // This will set it to our currently allowed maximum.
58 // Values this high are problematic because it is roughly the typical
        boundary
59 // where most messages repeat.
60 // e.g. It will stop decoding a message and start sending it to
       serial at
```

```
61 //      precisely the time when the next message is likely to be
       transmitted,
62 //      and may miss it.
63
64 // Set the smallest sized "UNKNOWN" message packets we actually care
       about.
65 // This value helps reduce the false-positive detection rate of IR
       background
66 // noise as real messages. The chances of background IR noise getting
        detected
67 // as a message increases with the length of the kTimeout value. (See
        above)
68 // The downside of setting this message too large is you can miss
       some valid
69 // short messages for protocols that this library doesn't yet decode.
70 //
71 // Set higher if you get lots of random short UNKNOWN messages when
       nothing
72 // should be sending a message.
73 // Set lower if you are sure your setup is working, but it doesn't
       see messages
74 // from your device. (e.g. Other IR remotes work.)
75 // NOTE: Set this value very high to effectively turn off UNKNOWN
       detection.
76 const uint16_t kMinUnknownSize = 12;
77
78 // How much percentage lee way do we give to incoming signals in
       order to match
79 // it?
80 // e.g. +/- 25% (default) to an expected value of 500 would mean
       matching a
81 //      value between 375 & 625 inclusive.
82 // Note: Default is 25(%). Going to a value >= 50(%) will cause some
       protocols
83 //       to no longer match correctly. In normal situations you
       probably do not
84 //       need to adjust this value. Typically that's when the library
        detects
85 //       your remote's message some of the time, but not all of the
       time.
86 const uint8_t kTolerancePercentage = kTolerance;  // kTolerance is
       normally 25%
87
```

```
88 // Legacy (No longer supported!)
89 //
90 // Change to `true` if you miss/need the old "Raw Timing[]" display.
91 #define LEGACY_TIMING_INFO false
92 // ==================== end of TUNEABLE PARAMETERS
       ====================
93
94 // Use turn on the save buffer feature for more complete capture
       coverage.
95 IRrecv irrecv(kRecvPin, kCaptureBufferSize, kTimeout, true);
96 decode_results results;  // Somewhere to store the results
97
98 // This section of code runs only once at start-up.
99 void setup() {
100 #if defined(ESP8266)
101   Serial.begin(kBaudRate, SERIAL_8N1, SERIAL_TX_ONLY);
102 #else  // ESP8266
103   Serial.begin(kBaudRate, SERIAL_8N1);
104 #endif  // ESP8266
105   while (!Serial)  // Wait for the serial connection to be establised
          .
106     delay(50);
107   // Perform a low level sanity checks that the compiler performs bit
          field
108   // packing as we expect and Endianness is as we expect.
109   assert(irutils::lowLevelSanityCheck() == 0);
110
111   Serial.printf("\n" D_STR_IRRECVDUMP_STARTUP "\n", kRecvPin);
112 #if DECODE_HASH
113   // Ignore messages with less than minimum on or off pulses.
114   irrecv.setUnknownThreshold(kMinUnknownSize);
115 #endif  // DECODE_HASH
116   irrecv.setTolerance(kTolerancePercentage);  // Override the default
          tolerance.
117   irrecv.enableIRIn();  // Start the receiver
118 }
119
120 // The repeating section of the code
121 void loop() {
122   // Check if the IR code has been received.
123   if (irrecv.decode(&results)) {
124     // Display a crude timestamp.
125     uint32_t now = millis();
```

```
126     Serial.printf(D_STR_TIMESTAMP " : %06u.%03u\n", now / 1000, now %
            1000);
127     // Check if we got an IR message that was to big for our capture
            buffer.
128     if (results.overflow)
129       Serial.printf(D_WARN_BUFFERFULL "\n", kCaptureBufferSize);
130     // Display the library version the message was captured with.
131     Serial.println(D_STR_LIBRARY "   :  v"
            _IRREMOTEESP8266_VERSION_STR "\n");
132     // Display the tolerance percentage if it has been change from
            the default.
133     if (kTolerancePercentage != kTolerance)
134       Serial.printf(D_STR_TOLERANCE " : %d%%\n", kTolerancePercentage
              );
135     // Display the basic output of what we found.
136     Serial.print(resultToHumanReadableBasic(&results));
137     // Display any extra A/C info if we have it.
138     String description = IRAcUtils::resultAcToString(&results);
139     if (description.length()) Serial.println(D_STR_MESGDESC ": " +
            description);
140     yield();  // Feed the WDT as the text output can take a while to
            print.
141 #if LEGACY_TIMING_INFO
142     // Output legacy RAW timing info of the result.
143     Serial.println(resultToTimingInfo(&results));
144     yield();  // Feed the WDT (again)
145 #endif  // LEGACY_TIMING_INFO
146     // Output the results as source code
147     Serial.println(resultToSourceCode(&results));
148     Serial.println();     // Blank line between entries
149     yield();              // Feed the WDT (again)
150   }
151 }
```

A.2 Código C para comunicação com dispositivo IR

```
152
153 /*
154  Basic ESP8266 MQTT example
155  This sketch demonstrates the capabilities of the pubsub library in
        combination
156  with the ESP8266 board/library.
```

```
157  It connects to an MQTT server then:
158   - publishes "hello world" to the topic "outTopic" every two seconds
159   - subscribes to the topic "inTopic", printing out any messages
160     it receives. NB - it assumes the received payloads are strings
          not binary
161   - If the first character of the topic "inTopic" is an 1, switch ON
       the ESP Led,
162     else switch it off
163  It will reconnect to the server if the connection is lost using a
     blocking
164  reconnect function. See the 'mqtt_reconnect_nonblocking' example for
       how to
165  achieve the same result without blocking the main loop.
166  To install the ESP8266 board, (using Arduino 1.6.4+):
167   - Add the following 3rd party board manager under "File ->
       Preferences -> Additional Boards Manager URLs":
168       http://arduino.esp8266.com/stable/package_esp8266com_index.
             json
169   - Open the "Tools -> Board -> Board Manager" and click install for
       the ESP8266"
170   - Select your ESP8266 in "Tools -> Board"
171 */
172
173 #include <ESP8266WiFi.h>
174 #include <PubSubClient.h>
175 #include <IRremoteESP8266.h>
176 // Update these with values suitable for your network.
177 #include <IRsend.h>
178
179 const char* ssid = "";
180 const char* password = "";
181 const char* mqtt_server = "";
182 const uint16_t kIrLed = 14;
183 IRsend irsend(kIrLed);
184 int tamanho=59;
185 int frequencia=32;
186 uint16_t LIGA[59] = {3200,9900, 550,1600, 550,500, 550,500, 550,550,
     550,1600, 500,550, 550,550, 550,550, 550,500, 550,550, 550,500,
     550,550, 550,500, 550,550, 550,500, 550,550, 550,1550, 550,500,
     550,500, 550,550, 550,550, 550,1550, 550,550, 550,500, 550,1600,
     550,1600, 550,550, 550,500, 550};
187 uint16_t DESLIGA[59] = {3200,9900, 550,1600, 550,500, 550,500,
     550,550, 550,1600, 500,550, 550,550, 550,550, 550,500, 550,550,
```

```
    550,500, 550,550, 550,500, 550,550, 550,500, 550,550, 550,1550,
    550,500, 550,500, 550,550, 550,550, 550,1550, 550,550, 550,500,
    550,1600, 550,1600, 550,550, 550,500, 550};
188
189 WiFiClient espClient;
190 PubSubClient client(espClient);
191 unsigned long lastMsg = 0;
192 #define MSG_BUFFER_SIZE (50)
193 char msg[MSG_BUFFER_SIZE];
194 int value = 0;
195
196 void setup_wifi() {
197
198   delay(10);
199   // We start by connecting to a WiFi network
200   Serial.println();
201   Serial.print("Connecting␣to␣");
202   Serial.println(ssid);
203
204   WiFi.mode(WIFI_STA);
205   WiFi.begin(ssid, password);
206
207   while (WiFi.status() != WL_CONNECTED) {
208     delay(500);
209     Serial.print(".");
210   }
211
212   randomSeed(micros());
213
214   Serial.println("");
215   Serial.println("WiFi␣connected");
216   Serial.println("IP␣address:␣");
217   Serial.println(WiFi.localIP());
218 }
219
220 void callback(char* topic, byte* payload, unsigned int length) {
221   Serial.print("Message␣arrived␣[");
222   Serial.print(topic);
223   Serial.print("]␣");
224   for (int i = 0; i < length; i++) {
225     Serial.print((char)payload[i]);
226   }
227   Serial.println();
```

```
228
229   // Switch on the LED if an 1 was received as first character
230   if ((char)payload[0] == '1') {
231     digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note that
          LOW is the voltage level
232     irsend.sendRaw(LIGA,tamanho,frequencia);
233
234     // but actually the LED is on; this is because
235     // it is active low on the ESP-01)
236   } else {
237     digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off by making
          the voltage HIGH
238     irsend.sendRaw(DESLIGA,tamanho,frequencia);
239   }
240
241 }
242
243 void reconnect() {
244   // Loop until we're reconnected
245   while (!client.connected()) {
246     Serial.print("Attempting␣MQTT␣connection...");
247     // Create a random client ID
248     String clientId = "ESP8266Client-";
249     clientId += String(random(0xffff), HEX);
250     // Attempt to connect
251     if (client.connect(clientId.c_str())) {
252       Serial.println("connected");
253       // Once connected, publish an announcement...
254       client.publish("outTopic", "hello␣world");
255       // ... and resubscribe
256       client.subscribe("scheduler/topic");
257     } else {
258       Serial.print("failed,␣rc=");
259       Serial.print(client.state());
260       Serial.println("␣try␣again␣in␣5␣seconds");
261       // Wait 5 seconds before retrying
262       delay(5000);
263     }
264   }
265 }
266
267 void setup() {
```

```
268   pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin
          as an output
269   Serial.begin(115200);
270   irsend.begin();
271
272   setup_wifi();
273   client.setServer(mqtt_server, 1883);
274   client.setCallback(callback);
275   digitalWrite(BUILTIN_LED, HIGH);
276 }
277
278 void loop() {
279
280   if (!client.connected()) {
281     reconnect();
282   }
283   client.loop();
284
285 }
286    2022 GitHub, Inc.
287 Terms
288 Privacy
289 Security
290 Status
291 Docs
292 Contact GitHub
293 Pricing
294 API
295 Training
296 Blog
297 About
298 Loading complete
```

B Códigos Python

B.1 Comunicação Api Google

```
import pickle
import os
import datetime
from google_auth_oauthlib.flow import Flow, InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.http import MediaFileUpload, MediaIoBaseDownload
from google.auth.transport.requests import Request


def Create_Service(client_secret_file, api_name, api_version, *scopes):
    print(client_secret_file, api_name, api_version, scopes, sep='-')
    CLIENT_SECRET_FILE = client_secret_file
    API_SERVICE_NAME = api_name
    API_VERSION = api_version
    SCOPES = [scope for scope in scopes[0]]
    print(SCOPES)

    cred = None
    working_dir = os.getcwd()
    token_dir = 'token files'

    pickle_file = f'token_{API_SERVICE_NAME}_{API_VERSION}.pickle'
    # print(pickle_file)

    ### Check if token dir exists first, if not, create the folder
    if not os.path.exists(os.path.join(working_dir, token_dir)):
        os.mkdir(os.path.join(working_dir, token_dir))

    if os.path.exists(os.path.join(working_dir, token_dir, pickle_file)):
        with open(os.path.join(working_dir, token_dir, pickle_file), 'rb') as token:
            cred = pickle.load(token)

    if not cred or not cred.valid:
        if cred and cred.expired and cred.refresh_token:
            cred.refresh(Request())
        else:
            flow = Installed
            AppFlow.from_client_secrets_file(CLIENT_SECRET_FILE, SCOPES)
            cred = flow.run_local_server(port=5034)

        with open(os.path.join(working_dir, token_dir, pickle_file), 'wb') as token:
            pickle.dump(cred, token)

    try:
        service = build(API_SERVICE_NAME, API_VERSION, credentials=cred)
```

```
        print(API_SERVICE_NAME, 'service created successfully')
        return service
    except Exception as e:
        print(e)
        print(f'Failed to create service instance for {API_SERVICE_NAME}')
        os.remove(os.path.join(working_dir, token_dir, pickle_file))
        return None

def convert_to_RFC_datetime(year=1900, month=1, day=1, hour=0, minute=0):
    dt = datetime.datetime(year, month, day, hour, minute, 0).isoformat() + 'Z'
    return dt
```

B.1.1 Código principal

```
from Google import Create_Service
import datetime
from dateutil import rrule
import dateutil.parser
from paho.mqtt import client as mqtt_client
import random
from time import sleep

#Porta para comunica  o MQTT
port = 1883
#Endere o do Broker MQTT
broker="172.31.31.123"
#T pico MQTT
topic="scheduler/topic"
#id do cliente
client_id = f'python-mqtt-{random.randint(0, 1000)}'
#Nome do arquivo com as credenciais para acessar A API
CLIENT_SECRET_FILE='client_secrets.json'
#Nome da Api
API_NAME = 'calendar'
#Versao da API
API_VERSION='v3'
#Escopo
SCOPES=['https://www.googleapis.com/auth/calendar']

#Conex o mqtt
def connect_mqtt():
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker!")
        else:
            print("Failed to connect, return code %d\n", rc)
    # Set Connecting Client ID
    client = mqtt_client.Client()
    # client.username_pw_set(username, password)
    client.on_connect = on_connect
    client.connect(broker, port)
    return client
```

```
#Publica no t pico MQTT
def publish(mode):
    client=connect_mqtt()
    # result: [0, 1]

    while True:
        print(f"Send `{mode}` to topic `{topic}`")
        client=connect_mqtt()
        result = client.publish(topic, mode)
        status = result[0]
        if status==0:
            return True
        else:
            sleep(1)


#autentica o na API do Google Agenda
service = Create_Service(CLIENT_SECRET_FILE,API_NAME,API_VERSION,SCOPES)
#%%
#estado de desligado=0 ou ligado=1 para eventos recorrentes
rec_state=0
#Estado de desligado=0 ou ligado=1 para eventos simples
single_state=0

#Laco para ativar ou desativar dispositivo de acordo com o horario
while True:
    try:
#lista de eventos
        events_list=service.events().list(calendarId='
                                                    5u8mdsherc4mp7fcg3kj5hvcmo@group.
                                                    calendar.google.com').execute()

    #Caso ocorra Connection Reset Error a conexao e estabelecida novamente
    except(ConnectionResetError):
        CLIENT_SECRET_FILE='client_secrets.json'
        API_NAME = 'calendar'
        API_VERSION='v3'
        SCOPES=['https://www.googleapis.com/auth/calendar']
        service = Create_Service(CLIENT_SECRET_FILE,API_NAME,API_VERSION,SCOPES)

    else:
        #eventos filtrados por recorrencia
        rec_conf_events=list(filter(lambda x:'confirmed' in x['status'] and x.get('
                                                    recurrence'),events_list['items'])
                                                    )
        #eventos unicos

        single_conf_events=list(filter(lambda x:'confirmed' in x['status'] and not x
                                                    .get('recurringEventId') and not x
                                                    .get('recurrence'),events_list['
                                                    items']))
```

```
        #eventos cancelados
        cancelled_events=list(filter(lambda x:'cancelled' in x['status'],events_list
                                                    ['items']))
        #eventos atualizados
        update_event=list(filter(lambda x:'confirmed' in x['status'] and x.get('
                                                    originalStartTime'),events_list['
                                                    items']))

    #se existir eventos recorrentes
        if rec_conf_events:
            #percorre os eventos recorrentes
            for calendar in rec_conf_events:
                days_rec=rrule.rrulestr(calendar['recurrence'][0])._byweekday
                #lista com o numero da semana dos dias recorrentes
                days_rec=list(days_rec)
                #id do evento recorrente
                id_rec=calendar.get('id')
                #horario de comeco do evento
                start_event=dateutil.parser.isoparse(calendar.get('start')['dateTime
                                                        ']).replace(tzinfo=None)
                #horario de fim do evento
                end_event=dateutil.parser.isoparse(calendar.get('end')['dateTime']).
                                                        replace(tzinfo=None)
                #Caso possuir eventos cancelados
                if cancelled_events:
                    for cancelled in cancelled_events:
                        id_cancelled=cancelled.get('recurringEventId')
                        cancelled_start=dateutil.parser.isoparse(cancelled.get('
                                                                originalStartTime'
                                                                )['dateTime']).
                                                                replace(tzinfo=
                                                                None)
                        if id_rec==id_cancelled:
                            cancelled_day=cancelled_start.weekday()
                            #remove da lista dos dias de eventos recorrentes
                            days_rec.remove(cancelled_day)
                #Se existe eventos atualizados
                if update_event:
                    #Percorre os eventos atualizados
                    for update in update_event:

                        id_update=update.get('recurringEventId')
                        start_up=dateutil.parser.isoparse(update.get('start')['
                                                                dateTime']).
                                                                replace(tzinfo=
                                                                None)
                        end_up=dateutil.parser.isoparse(update.get('end')['dateTime'
                                                                ]).replace(tzinfo=
                                                                None)

                        if id_rec==id_update:
```

```
                    start_event=start_up
                    end_event=end_up
            #data atual
            now_date=datetime.now().utcnow().replace(second=0, microsecond=0)
            #dia da semana atual
            weekday_now=now_date.weekday()
            #se o dia atual estiver na lista de dias de eventos recorrentes
            if weekday_now in days_rec:
                #se a data atual sem fuso-horario for igual a data de inicio do
                                                        evento e o dispositivo
                                                         estiver desligado
                if now_date == start_event and not rec_state:
                    print('Evento Recorrenteiniciado')
                    #Publica no T pico MQTT
                    publish(1)
                    rec_state=1
                #Se a data de fim sem fuso-horario for igual a data de fim do
                                                        evento e o dispositivo
                                                         estiver ligado
                elif now_date == end_event and rec_state:
                    print('Evento Recorrente Encerrado')
                    #Publica no topico MQTT
                    publish(0)
                    rec_state=0
    #Se possui eventos unicos
    if single_conf_events:
        #Percorre eventos unicos
        for calendar in single_conf_events:
            #Inicio do evento
            start_event = dateutil.parser.isoparse(calendar.get('start')['
                                                    dateTime']).replace(tzinfo
                                                    =None)
            #Fim do evento
            end_event = dateutil.parser.isoparse(calendar.get('end')['dateTime']
                                                    ).replace(tzinfo=None)
            #Data atual
            now_date=datetime.now().utcnow().replace(second=0, microsecond=0)
            #se a data de inicio for igual a data atual e o dispositivo estiver
                                                    desligado
            if now_date==start_event and not single_state:
                print('Evento unico iniciado')
                publish(1)
                single_state=1
            #Se a data de fim for igual a data atual e o dispositivo estiver
                                                    ligado
            elif now_date==end_event and single_state:
                print('Evento unico encerrado')
                publish(0)
                single_state=0
```

C Comandos Linux

- sudo apt install npm
- npm install pm2@latest -g
- sudo npm install pm2@latest -g
- sudo apt-get update
- sudo apt-get install python3.9
- sudo python3 get-pip.py
- sudo python3 get-pip3.py
- pip3 install virtualenv
- sudo apt install python3-pip
- pip3 install virtualenv
- sudo pip install virtualenv virtualenvwrapper
- sudo rm -rf /.cache/pip get-pip.py
- sudo nano .bashric
- sudo nano .bashrc
- source .bashrc
- mkvirtualenv tcc -python=39
- sudo apt install git
- git clone https://github.com/xelel/Tcc.git
- pip install maho-mqtt
- pip install paho-mqtt
- pip install httplib2
- pip install dateutil
- pip install google-api-python-client
- pip install google_auth_oauthlib
- python3 airscheduler.py

- `pip install pytz`
- `python3 airscheduler.py`
- `pip install python-dateutil`
- `python3 airscheduler.py`
- `sudo apt-get update -y  sudo apt-get upgrade -y`
- `sudo apt-get install mosquitto`
- `sudo apt-get install mosquitto-clients`
- `sudo apt install net-tools`
- `sudo nano airscheduler.py`
- `mosquitto_sub -t scheduler/topic`
- `mosquitto_pub -m 'TEste' -t scheduler/topic`
- `pm2 start airscheduler.py -name irscheduler -interpreter /home/ubuntu/.virtualenvs/tcc/bin/python3.9`