UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mário César Melon Bomfim

O Estado da Arte e as Perspectivas de *IoT* com um Estudo de Caso na Prática

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Mário César Melon Bomfim

O Estado da Arte e as Perspectivas de *IoT* com um Estudo de Caso na Prática

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Orientador: Daniel Duarte Abdala

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2023

Mário César Melon Bomfim

O Estado da Arte e as Perspectivas de *IoT* com um Estudo de Caso na Prática

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 07 de junho de 2023:

Daniel Duarte Abdala
Orientador

Marcia Aparecida Fernandes

Maria Adriana Vidigal de Lima

Uberlândia, Brasil
2023

Agradecimentos

À Deus dedico este trabalho, a este Ser Maior, que me concedeu a vida, determinação e sabedoria no decorrer deste curso.

“Deus escreveu com uma tinta que nunca borra, fala com uma língua que nunca erra, age com uma mão que nunca falha.”

Resumo

Com a constante evolução da Internet das Coisas diversas soluções e produtos surgiram no mercado. Grande parte das soluções presentes estão concentradas em nichos específicos, tais como *smart TVs* e *smart Lamps*. Dispositivos não contemplados pela indústria recaem inevitavelmente em soluções caseiras que vias de regra implicarão em complexidades decorrentes da falta de padronização e especificidade técnica de cada dispositivo. Este trabalho buscou analisar o estado da arte com foco na esfera domiciliar. Com base neste levantamento, identificou-se os pontos cruciais para adaptação de um dispositivo regular ao ecossistema IoT o que resultou na proposta de um *template* que sistematiza os passos necessários do processo. Baseado neste *template* um estudo de caso foi realizado com o intuito de validar os conceitos e ideais apresentados neste trabalho funcionam para um dispositivo qualquer - no caso, um ventilador de mesa padrão. Este trabalho conclui que para IoT se tornar cada vez mais pervasivo, três fatores são suficientes e necessários, nomeadamente, padronização, interfaceamento homem-máquina e banda de comunicação.

Palavras-chave: Internet das coisas, IoT, Automação, Arduino, ESP32, Wi-Fi, Conectividade, 5G, Sofware, arquitetura Cliente-Servidor.

Abstract

With the constant evolution of the Internet of Things several solutions and products have emerged in the market. Most of the present solutions are concentrated in specific niches, such as smart TVs and smart Lamps. Devices not contemplated by the industry inevitably fall into homemade solutions that usually imply complexities due to the lack of standardization and technical specificity of each device. This monography sought to analyze the state of the art focusing on the home(house) sphere. Based on this survey, the crucial points to adapt a regular device to the IoT ecosystem were identified, which resulted in the proposal of a template that systematizes the necessary steps of the process. Based on this template a case study was performed in order to validate that the concepts and ideals presented in this work for any given device - in this case, a standard table fan. It concludes that for IoT to become increasingly pervasive, three factors are sufficient and necessary, namely standardization, human-machine interfacing, and communication bandwidth.

Keywords: Internet of Things, IoT, Automation, Arduino, ESP32, Wi-Fi, 5G, Sofware, Client-Server architecture.

Lista de ilustrações

Sumário

Lista de códigos-fonte

1 Introdução

Internet das coisas é uma tradução literal do termo em inglês Internet of Things - *IoT*. Contudo, ela pode ser melhor descrita como "Internet em Todas as Coisas". O conceito não é novo, ele remonta a ideia de *smart house* (casa inteligente) criado na década de 1960 na estreia do programa espacial norte americano. Ele se refere a uma rede de objetos físicos ou "coisas" que são incorporadas a sensores, softwares e conectividades, o que lhes permite trocar dados com outros dispositivos e sistemas pela Internet. Esses objetos podem ser qualquer coisa, desde dispositivos domésticos inteligentes, vestíveis e veículos até equipamentos e infraestrutura industriais. O objetivo da *IoT* é tornar a vida mais fácil e eficiente, automatizando várias tarefas e reduzindo a intervenção humana.

Embora o termo *smart house* seja vagamente familiar, poucos possuem uma compreensão concreta do que significa. Foi utilizado pela primeira vez de maneira oficial na década de 1984 pela Associação Americana de Construtores de Casa, embora as primeiras "casas com fio" já terem sido construídas por hobistas no início dos anos 1960. Este desenvolvimento é fundamental para definir o termo, lembrando que o programa espacial foi um grande catalisador deste movimento, conduzido por amantes da eletrônica e computação (HARPER, 2003).

A expectativa para a época era a automatização de componentes domésticos. Com o passar das décadas as tecnologias se difundiram e atualmente no século XXI várias tecnologias foram criadas afim de suprir essa expectativa. Já é possível controlar a temperatura de um ambiente com ar-condicionado inteligente ou até mesmo do sistema de aquecimento da casa através de um termostato conectado a sua rede local e controlado pelo seu *smartphone*.

A *IoT* nasce com a expectativa de ser a nova tecnologia que habitará uma vida conectada, extrapolando o conceito de smart house para todos os aspectos da vida moderna e tornando toda a automatização decorrente interconectada por meio da Internet. Ela pode ser vista como precursora que levará o ser humano a um estilo de vida totalmente conectado. Esta descrição define o conceito de computação pervasiva ou ubíqua, um mundo onde a maioria das atividades será auxiliada por dispositivos computadorizados. A expectativa é pautada em como a *IoT* irá transformar esse cenário e causar uma grande mudança no modo de vida atual.

Apesar da *IoT* ser um tópico corrente (PRESS, 2014), o mercado está aquém de oferecer esse tipo serviço em larga escala para o consumidor. Ainda há uma concentração em produtos muito específicos como lâmpadas e tomadas inteligentes ou em soluções *IoT ready* (prontas para uso) onde tudo é configurado numa central de automação ou um

aplicativo proprietário para o controle de luzes, cortinas, temperatura e afins. Entretanto essa segunda solução é economicamente mais cara, e por isso, não tão popular.

Todavia, a *IoT* é muito mais do que suas aplicações no nicho doméstico, o mundo atual já vivencia serviços e tecnologias *IoT* tais como: Sistemas de vigilância conectados a rede onde é possível acessar as imagens de um smartphone com conexão a Internet e ser notificado caso algum movimento suspeito ocorrer. Um exemplo mais próximo são as diversas tags de pedágio que possibilitam a viagem sem interrupções. Por último montadoras como a Audi já disponibilizam um dispositivo que, ao ser conectado no carro, oferece todas as informações de status do veículo como: autonomia, nível de combustível, pressão dos pneus, temperatura do líquido de arrefecimento, etc.

O ganho passível é possibilitar a acessibilidade e a conectividade às informações de sensores, motores, iluminação, de qualquer lugar com conexão a rede, ou seja, possibilitar o conhecimento de por exemplo quantas caixas de leite ainda restam na geladeira ou qual a umidade do solo de uma horta enquanto estamos fora de casa e consequentemente ser possível tomar decisões que nos foram informadas e de maneira semi ou completamente automatizada.

No entanto, existem outros equipamentos que utilizam a conexão da rede para realizar atividades específicas como câmeras de segurança que, por estarem online, permitem o monitoramento residencial de qualquer lugar e a qualquer momento. Outro exemplo mais próximo do dia a dia são as Smart TVs, que oferecem serviços como a Netflix, YouTube, Spotify, etc. A mais nova geração de video games (consoles) também utilizam a conexão para jogos *multiplayer*, além de assinatura de jogos exclusivos.

Nota-se que existe um esforço para tornar uma gama de dispositivos e serviços *IoT ready*, esse esforço pode ser tanto por parte de empresa quanto de indivíduos. Porém, torná-los *IoT ready* não é uma tarefa simples, a ausência de padronização, a carência de banda de comunicação e a interface humano-computador prejudicam o avanço e por isso pesquisas e desenvolvimentos têm sido feitos nesse aspecto.

Este trabalho consiste em apresentar uma revisão sistemática do estado da arte em *IoT*, focando especialmente na adequação de dispositivos para *IoT ready*. Neste contexto também é apresentado um levantamento dos equipamentos disponíveis no mercado. Notar-se-a que a disponibilidade é muito aquém do esperado, o que motiva inúmeros desenvolvedores a criar soluções próprias para a adequação dos seus dispositivos de interesse ao ecossistema da *IoT*.

Este trabalho também apresenta um estudo de caso onde tecnologias comumente disponíveis são utilizadas para a adequação de um ventilador ao ambiente *IoT*. Um estudo de viabilidade é apresentado de modo a avaliar a relação custo/benefício deste tipo de solução caseira.

1.1 Objetivo

O objetivo principal deste trabalho refere-se a pesquisar, comparar e demonstrar prós e contras da automação caseira e a automação *IoT ready* disponíveis no mercado. Utilizou-se um estudo de caso que denota passo-a-passo o processo de adaptação de um dispositivo não *IoT ready* tornando-o *IoT ready* utilizando a rede *wireless* para controlá-lo via Internet e assim fazer parte da rede *IoT*.

1.1.1 Objetivos Específicos

Como objetivos específicos no desenvolvimento deste trabalho elencam-se:

- Pesquisar e apresentar os conceitos que permeiam a área da *IoT*;
- Pesquisar e analisar produtos da *IoT* disponíveis no mercado, bem como comparar as vantagens e limitações frente ao estado da arte;
- Discutir as tecnologias existentes que possibilitam a automatização caseira de dispositivos não *IoT ready*;
- Executar um estudo de caso da automatização de um ventilador usando a tecnologia previamente pesquisada como prova de conceito.

1.2 Justificativa

Ainda que a *IoT* seja um tema relevante, ela não é bem compreendida pela maioria dos praticantes em computação, principalmente os praticantes em início de carreira. A curva de aprendizado é acentuada e fundamentalmente interdisciplinar, são necessários conhecimentos em elétrica, eletrônica básica e uma vasta base em programação. Há ainda outros fatores que dificultam a emersão da *IoT* em larga escala, o que leva a existir todo um segmento atuante em *IoT* que busca produzir soluções caseiras para uma efetiva conversão de dispositivos não *IoT ready* dentro do ambiente *IoT*.

1.3 Hipótese

Embora o trabalho apresente uma visão crítica e revisacional da área de *IoT*, há uma hipótese que percorreu o desenvolvimento do caso de uso, a qual consiste em mensurar o quão complexo seria tornar um dispositivo *IoT ready* quando ele não foi concebido para sê-lo. Sobre este pressuposto postula-se: É possível, usando uma quantidade de recursos aceitável, tornar um dispositivo não *IoT ready* em *IoT ready*.

O restante desta monografia está organizado como segue: O capítulo 2 apresenta a metodologia de pesquisa em conjunto com a revisão do estado da arte e o desenvolvimento adotado neste trabalho. O capítulo 3 descreve toda fundamentação teórica utilizada e ademais conceitos relativos ao tema. O capítulo 4 trata do estudo de caso utilizado como prova de conceito do terceiro, e finalmente no capítulo 5 é apresentado a conclusão e as considerações finais sobre IoT, além de possíveis trabalhos futuros.

2 Fundamentação Teórica

Com base na pesquisa apresentada no capítulo anterior nota-se que muitos dispositivos de automação domiciliar não estão disponíveis ou ainda não são viáveis economicamente. Portanto, resta a possibilidade de empresas ou indivíduos tornarem dispositivos originalmente não prontos para *IoT* em *IoT ready*. Nesse contexto, algumas tecnologias se apresentam como caminhos viáveis para a devida solução.

Para tornar um dispositivo *IoT ready* é necessário um artefato composto de *hardware* e *software*, um *device driver*, que consiste em uma central de processamento para interpretar e controlar o dispositivo via um software. Contudo, o dispositivo a ser transformado pode variar, sendo ele uma cafeteira, um liquidificador, uma lâmpada, etc. E esse controle de hardware que o *device driver* proporciona se torna um ponto problemático pois é necessário um controle específico para cada tipo de dispositivo e também, uma solução de código específica. A figura 2 ilustra essa situação, onde tem-se um dispositivo genérico, o qual terá seu próprio *device driver* para então se tornar parte do mundo *IoT*.

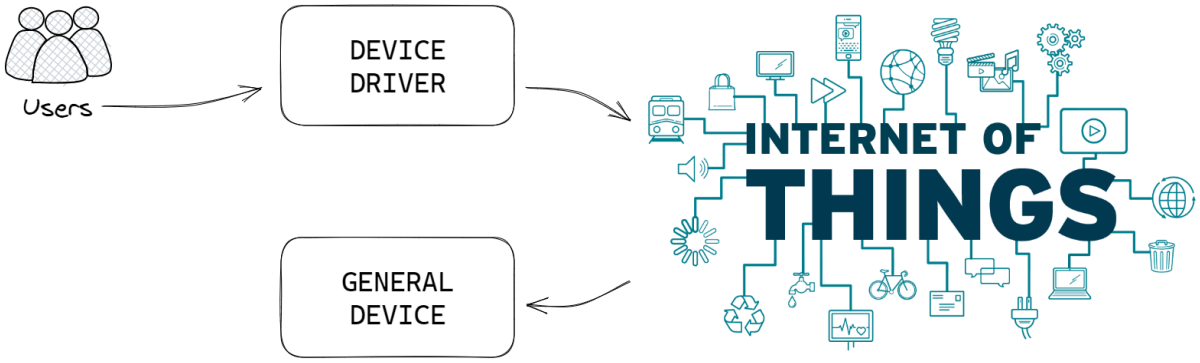

Figura 1 – Visão geral da conversão de um dispositivo em *IoT ready*.(GRATISPNG, 2023)

Alguns dispositivos irão requerer um *driver* analógico, outros, digital, e muito mais comumente tem-se uma parte desse *driver* analógico e uma parte digital, onde se faz necessidade de um acoplamento/conversão analógico-digital. Parte desse *driver* irá realizar tarefas complexas e, com isso, se tornará mais viável utilizar uma solução *open hardware* com um microcontrolador ou um microprocessador para tal ao invés de projetar o circuito de controle utilizando portas lógicas.

Neste contexto surgem algumas opções, dentre as mais comuns cita-se o Arduino, uma plataforma *open hardware* bem difundida e relativamente fácil de programar em uma linguagem própria chamada *Arduino Programming Language*(análoga a C/C++). Essa linguagem é baseada na plataforma de desenvolvimento *Wiring*, que, por sua vez,

é baseada em *Processing*(COPES, 2023). Existem também outras opções mais recentes e com mais recursos embutidos, dos quais o Arduino carece, como é o caso da família ESP, e seu carro-chefe o ESP32. Este capítulo será dividido da seguinte forma: Na primeira sessão será apresentado o diagrama para adaptação dos dispositivos, os microcontroladores mais comuns para isso e uma comparação entre eles, na segunda, a ponte entre o dispositivo e o microcontrolador, comumente chamada de acoplamento analógico-digital ou digital-analógico, na terceira as opções de conexão e níveis de compartilhamento do dispositivo na rede, e por último, conceitos teóricos que serão utilizados no capítulo de estudo de caso para tornar, de fato, um dispositivo em *IoT ready*.

2.1 Adaptação de Dispositivos

A maioria dos dispositivos de automação domiciliar irão necessitar de alguma forma de controle de liga/desliga e a alimentação desses dispositivos é o majoritariamente proveniente da rede de distribuição elétrica, cuja corrente é alternada. Consequentemente, parte do *device driver* e da maioria dos dispositivos deverá ser analógico.

No entanto, as funções desempenhadas por esses dispositivos variam grandemente, muitos deles vão requerer um controle analógico, como por exemplo um ventilador, mas outros, como televisão, ar-condicionado e home-teathers irão precisar de um controle digital.

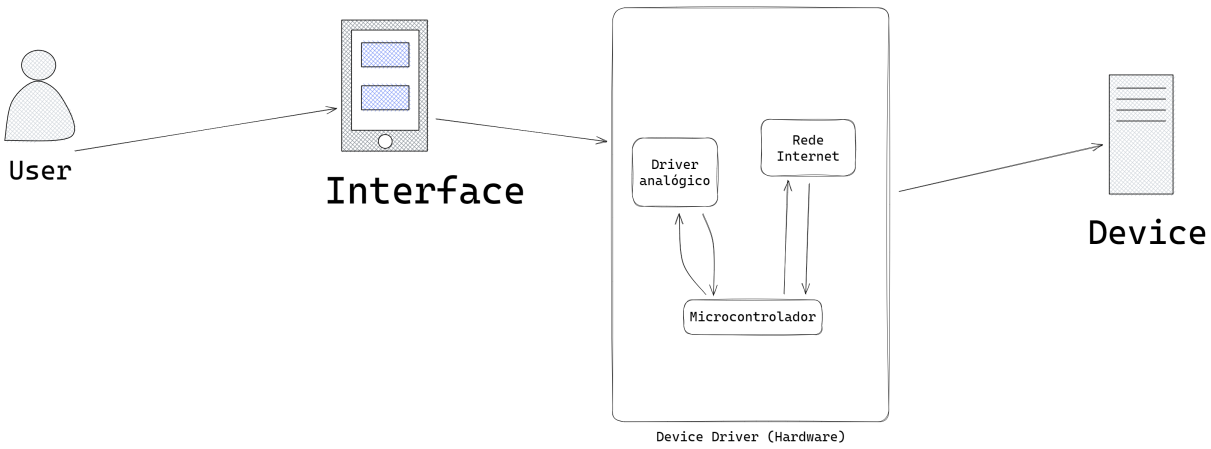

Figura 2 – Módulos de um *device driver*.

Um outro aspecto fundamental de dispositivos *IoT* é a sua conectividade, portanto o *driver* precisa de uma interface para se comunicar via rede (com ou sem fio). A figura 2 ilustra um nível mais detalhado aplicado a figura 1 dentro do bloco *device driver*. E na próxima figura (3) a abstração do microcontrolador é expandida e aprofundada demonstrando que ele se caracteriza por ser o "cérebro" do *device driver*.

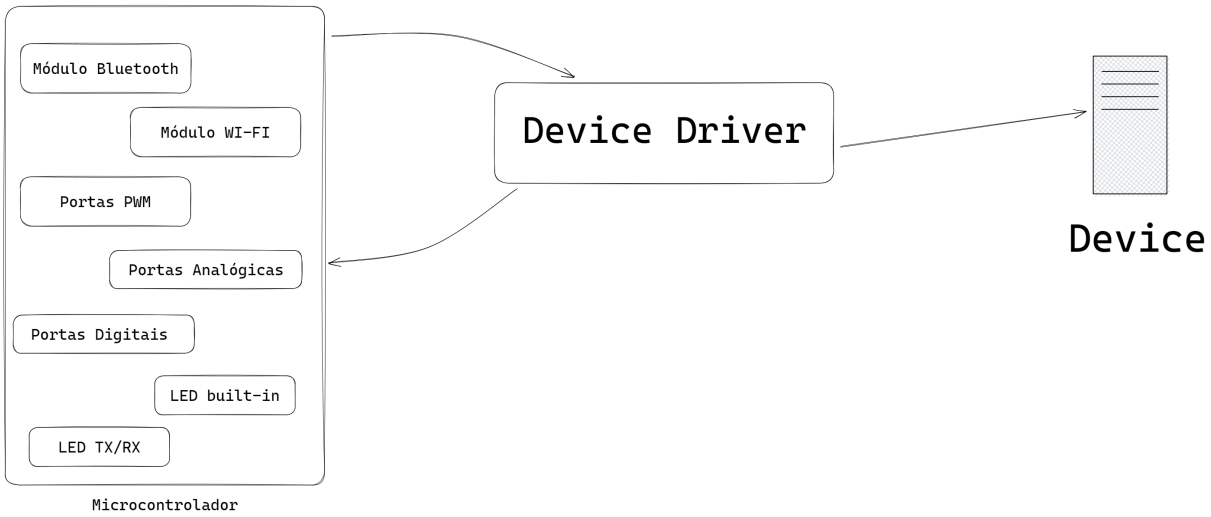

Figura 3 – Módulos de um microcontrolador.

Embora todos estes módulos possam fazer parte de um *device driver* nem todos são sempre necessários. Contudo, alguns são imprescindíveis, como exemplo podemos citar o módulo de rede, seja na configuração mais simples ou na mais complexa, ele precisa ser a ponte entre o dispositivo e a nuvem. Assim como o microcontrolador que faz o papel de cérebro do *device driver*.

Do mesmo modo, o código que o microcontrolador irá executar pode ser analogamente representado como visto na figura 4. Nesta figura observa-se que o código é dividido em partes e que cada módulo é responsável por uma tarefa. Por fim, é importante salientar que apenas os passos finais apontam um para o outro formando uma espécie de *loop*, isso quer dizer que, a partir do passo 8 a única tarefa do microcontrolador é estar pronto para receber requisições e executar o que ele foi programado para fazer, seja acionar um motor, desligar um relé, acender um LED, notificar o usuário, etc. O processo se torna uma máquina de estados possíveis que é acionada a partir de uma requisição.

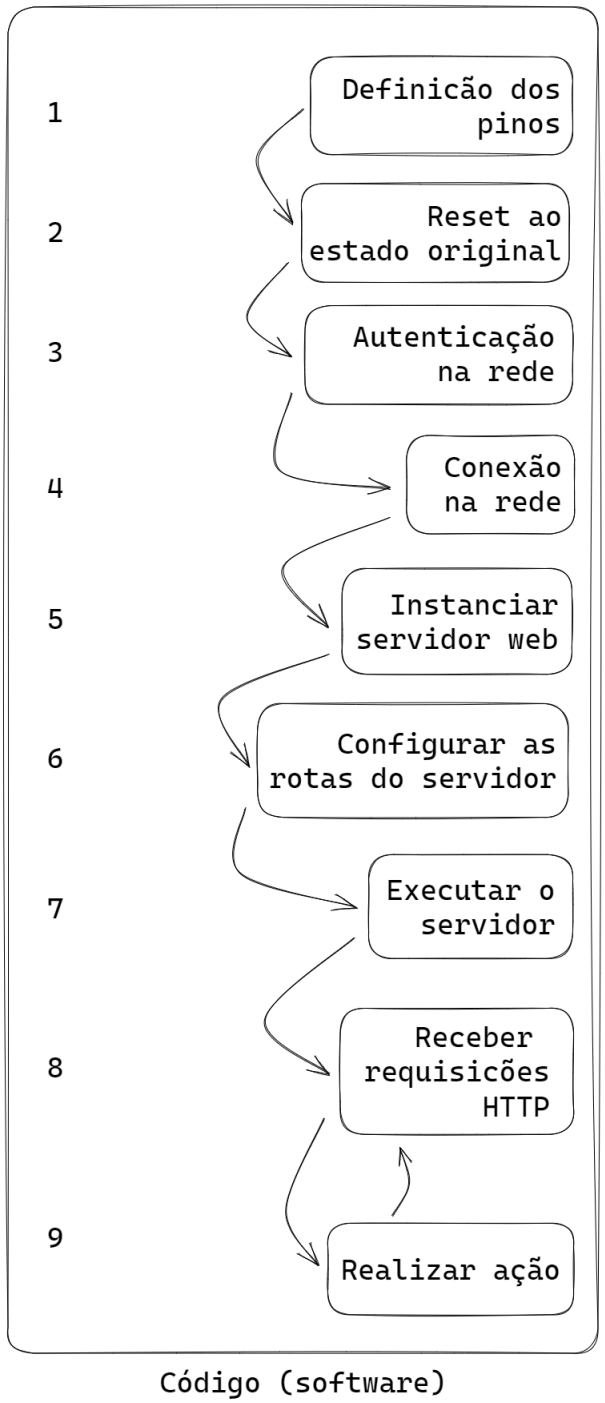

Figura 4 – Módulos do código.

2.2 Microcontroladores

O microcontrolador consiste em um único circuito integrado (CI) que reúne um núcleo de processador, memórias voláteis e não voláteis e diversos periféricos de entrada e de saída de dados. Ele não caracteriza nada além de um computador muito pequeno capaz de realizar determinadas tarefas de maneira eficaz, denotando um tamanho altamente compacto.

O grande precursor da popularização dos microcontroladores é uma plataforma *open hardware* chamada Arduino, construída utilizando o microcontrolador ATMEGA328 desenvolvido pela Atmel. O Arduino foi idealizado em 2005 por um grupo de 5 pesquisadores: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. O objetivo era elaborar um dispositivo que fosse ao mesmo tempo barato e fácil de programar, sendo dessa forma acessível aos estudantes e projetistas amadores. A primeira placa foi composta por um microcontrolador Atmel e programada via Ambiente de Desenvolvimento Integrado (IDE), com linguagem baseada em C/C + +. Já os circuitos de entrada e saída, poderiam ser conectados no computador por cabo USB. A figura 5 mostra toda a linha do tempo do Arduino.

Figura 5 – Linha do tempo do Arduino. (MAKERHERO, 2023a)

Devido ao seu baixo custo, o Arduino se tornou protagonista das soluções de *IoT* caseiras, como por exemplo: um regador automático para plantas com sensores de temperatura e umidade (CAMPBELL, 2023), ou então, acender e apagar uma fita de LED de acordo com a luminosidade do ambiente (SENSORKIT, 2023), até mesmo aleitura de biometria (ENGINEERSGARAGE, 2023) com CI externos (adicionais) chamados de *shields*. A versão mais comum encontrada atualmente do Arduino (Uno) não possui protocolos de comunicação sem fio e então se faz necessário um *shield*.

Shields Arduino são placas complementares que podem ser usadas com uma placa Arduino para estender sua funcionalidade. Tais placas vêm em uma variedade de tipos, variando de placas de sensores simples a módulos de comunicação complexos. A beleza dos *shields* Arduino é que eles permitem que os usuários personalizem facilmente seus projetos Arduino, adicionando os recursos e funcionalidades necessários sem ter que criar hardware personalizado a partir do zero.

Um tipo popular de *shield* Arduino é o *shield* Ethernet (figura 6), que permite a conexão da placa Arduino com a internet. Possuindo esse escudo os usuários podem criar projetos conectados à Internet, como servidores da Web ou dispositivos controlados remotamente. Outro *shield* popular é o *shield* de motor, que pode ser usado para controlar motores e outros dispositivos de alta potência. Este *shield* é ótimo para projetos de robótica ou qualquer projeto que exija controle preciso do motor. Também existem *shields* disponíveis para aplicações específicas, como rastreamento por GPS ou comunicação sem fio.

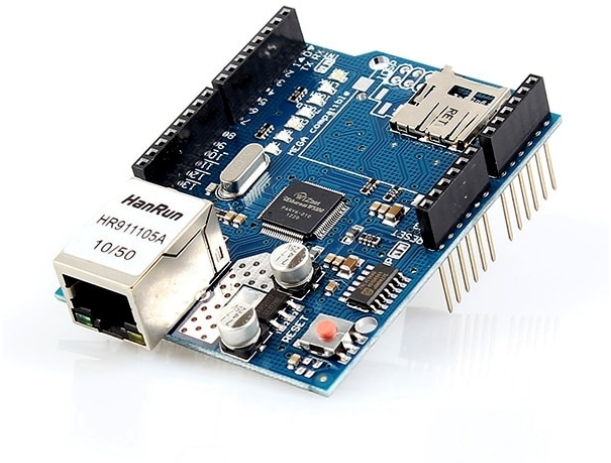

Figura 6 – Shield Arduino Ethernet. (MAKERHERO, 2023b)

Um dos benefícios de usar os *shields* Arduino é que eles são projetados para funcionar perfeitamente com as placas Arduino, tornando mais fácil para os usuários adicionar

recursos e funcionalidades aos seus projetos sem precisar se preocupar com problemas de compatibilidade ou fiação complexa. Muitos *shields* também vêm com bibliotecas e códigos de exemplo, o que torna mais fácil para os usuários começarem seus projetos. No geral, os *shields* do Arduino são uma ótima maneira de estender as capacidades.

Outro ponto já citado é o conceito de *open hardware* (hardware aberto), cuja definição é a que se refere ao design e especificações de hardware e designs sob licenças, as quais permitem que qualquer pessoa estude, modifique, distribua e use o hardware. Este conceito é baseado nos princípios do software de código aberto, onde o código-fonte está disponível para qualquer pessoa visualizar, usar e modificar. No caso de *open hardware*, isso significa que os arquivos de projeto, esquemas e outras especificações técnicas são liberados sob uma licença que permite aos usuários criar projetos, personalizá-los de acordo com suas necessidades ou fabricar o hardware para uso comercial ou pessoal.

O objetivo do *open hardware* é permitir a colaboração e a inovação no desenvolvimento de hardware, removendo barreiras e restrições que limitam o acesso à tecnologia ou desencorajam a experimentação e a inovação. O *open hardware* é visto como uma forma de democratizar o acesso a tecnologia, fornecendo à mais pessoas as ferramentas necessárias para criar suas próprias soluções de hardware. Portanto, o ecossistema de IoT se beneficia de um modo que sempre há colaboração da comunidade para um objetivo em comum a todos(as) entusiastas, usuários ou empresas.

Nos últimos anos outra família de soluções de *open hardware* vem ganhando renome entre os entusiastas de IoT. A família de microcontroladores embutida no ESP32, os quais, na realidade, podem ser classificados como System on Chip (SoC) de baixo custo e baixa alimentação desenvolvido pela Espressif, que inclui Wi-Fi e Bluetooth com um processador *dual-core*, representado na figura 7. Além disso, ainda pode-se utilizar da mesma linguagem de programação do Arduino.

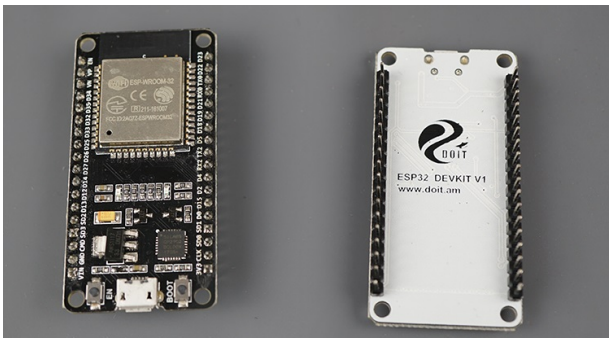

Figura 7 – ESP32 frente e verso. (RANDOMNERDTUTORIALS, 2023a)

Ambas soluções apresentadas foram criadas com o propósito de possuir entradas e saídas que podem ser processadas para atuar em algum sensor, motor e interagir com o ambiente. Dito isso, é perfeito para funcionar como um *device driver* sendo programado

para mimetizar o comportamento humano e, por consequência, automatizar um dispositivo. A figura 8 representa tecnicamente o que cada pino de um ESP32 tem como função. Cada um deles pode representar uma saída ou entrada e ainda existem pinos específicos, como por exemplo, aqueles para uma tela de toque (RANDOMNERDTUTORIALS, 2023a).

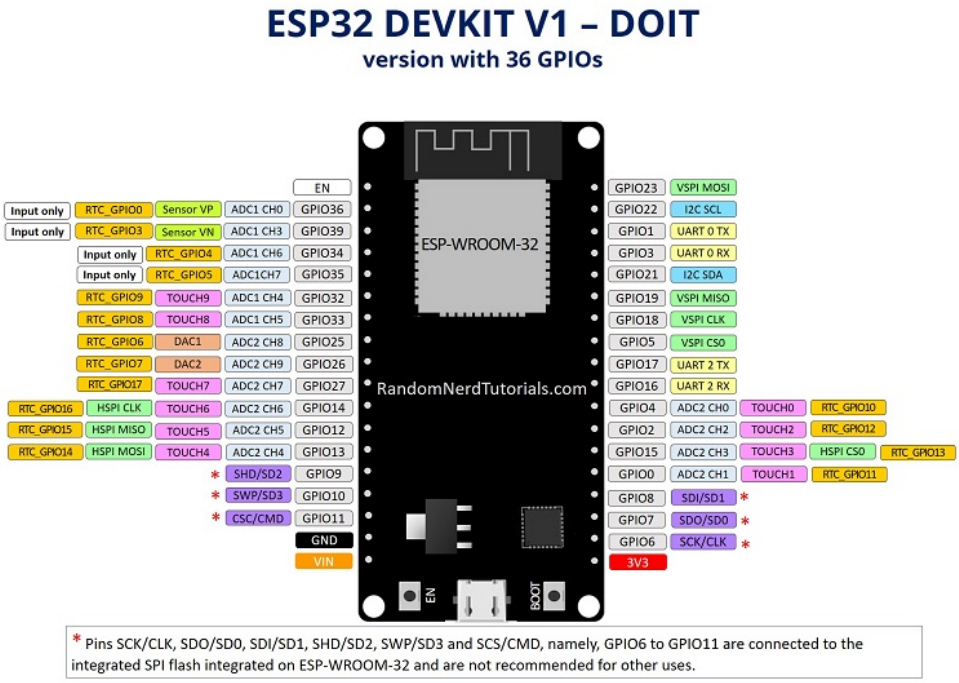

Figura 8 – ESP32 pinagem. (RANDOMNERDTUTORIALS, 2023a)

2.3 Controle Digital

Um dos recursos disponibilizados pelo microcontrolador é o controle digital de certos pinos. Como já foi dito anteriormente, ele possui várias entradas e saídas e uma boa parte delas é por onde o comando/sinal será enviado ao dispositivo a ele conectado. O controle digital funciona com valores discretos, ou seja, a porta (pino) está desligada ou ligada, não há um meio termo. E para isso o Arduino atribui 0V ou 5V, respectivamente.

Sendo o padrão do próprio Arduino uma referência, o código a seguir exemplifica o controle digital de um LED embutido. Na execução do código principal são necessárias duas funções *setup* e *loop* que lidam com todo o comportamento desejado adiante, isto é, utiliza-se ambas funções para definir parâmetros e constantes e repetir para sempre o que ele foi programado para fazer. A função *setup()* é executada uma única vez quando o microcontrolador é ligado, definindo tudo que é necessário para que o microcontrolador inicie com as informações necessárias. Já a função *loop()*, como o nome pressupõe, é executada repetidamente e tem o papel de prender a execução do programa para que a

partir de um estímulo externo ou quando alguma condição seja satisfeita, ela seja capaz de executar.

Este código apresentado abaixo(1) pisca o LED *on-board* a cada segundo. Funciona também com as placas ESP32 e ESP8266 (ambas possuem o LED *on-board*).

```
1  #include <Arduino.h>
2
3  #define LED 2
4
5  void setup() {
6    // put your setup code here, to run once:
7    Serial.begin(115200);
8    pinMode(LED, OUTPUT);
9  }
10
11 void loop() {
12   // put your main code here, to run repeatedly:
13   digitalWrite(LED, HIGH);
14   Serial.println("LED is on");
15   delay(1000);
16   digitalWrite(LED, LOW);
17   Serial.println("LED is off");
18   delay(1000);
19 }
```

Código 1: Controle Digital.

A linha 3 cria uma constante definindo LED com o valor 2, portanto desse ponto em diante o código substitui LED pelo valor 2, na função *setup* a linha 7 será explicada mais detalhadamente no capítulo 4, neste momento institui-se o microcontrolador acerca da taxa de comunicação com o computador ao qual ele está conectado. A função *pinMode* atribui LED como *OUTPUT* (saída). Portanto, o LED se comportará como algo a ser acionado.

Já a função *loop* atribui o valor *HIGH* na porta 2 (LED) na linha 13 e então o LED é aceso, na linha 15 há uma operação de espera *delay* que aguarda 1000 milissegundos (1s) e então, desliga-se o LED com o código da linha 16.

2.4 Controle Analógico

O controle analógico no Arduino permite manipular os níveis de tensão de um pino de entrada analógica. Isso significa que, ao invés de apenas detectar se um sinal está ativado ou desativado, pode-se ler uma faixa de valores que representam a força do sinal.

Usando PWM (*Pulse Width Modulation*), é possível variar os níveis de tensão de um pino de saída digital, simulando assim, uma saída analógica. Isso é útil em aplicações

como controlar o brilho dos LEDs, a velocidade dos motores ou servomotores e o volume da saída de som.

Com a finalidade de usar o controle analógico no Arduino, deve-se usar a função *analogRead()* para ler os níveis de tensão de um pino de entrada analógica e a função *analogWrite()* para escrever um valor analógico em um pino de saída digital habilitado para PWM. Com essas funções, você pode criar projetos complexos que envolvem controle preciso de sinais analógicos.

O código abaixo (2) é semelhante com o anterior, porém, a partir de um potenciômetro é possível controlar a intensidade do brilho do LED girando o cursor do potenciômetro. Quando o cursor for levado até o final da escala, teremos, por exemplo, 0V a ser fornecido ao pino de entrada do Arduino e assim, ao lê-lo, obtém-se 0. Quando se gira o cursor até o outro extremo da escala, haverá 5 e, ao lê-lo, obtêm-se 1023. O Arduino possui um conversor Analógico-Digital embutido de 10 bits, com isso 2^{10} são 1024 valores possíveis de tensão. (SOUZA, 2023)

```
1  int potPin = 0; // selecione o pino de entrada ao potenciômetro
2  int ledPin = 13;   // selecione o pino ao LED
3  int val = 0;    // variável a guardar o valor proveniente do sensor
4
5
6  void setup() {
7
8      pinMode(ledPin, OUTPUT);  // declarar o pino ledPin como saída
9
10 }
11
12
13
14 void loop() {
15
16     val = analogRead(potPin); // ler o valor do potenciômetro
17
18     digitalWrite(ledPin, HIGH);  // ligar o led
19     delay(val);                  // espera tempo ajustado no potenciometro
20     digitalWrite(ledPin, LOW);   // desligar o led
21     delay(val);                  // espera tempo ajustado no potenciometro
22
23 }
```

Código 2: Controle Analógico.

2.5 Acoplamento Analógico-Digital e Digital-Analógico

Levando em consideração um dispositivo não *IoT ready* requisita-se na grande maioria dos casos uma interação humana, isto é, o indivíduo precisa acionar um botão,

girar uma chave ou até utilizar algum tipo de teclado, botão giratório e afins. Alguns desses podem até possuir um controle remoto, porém ainda assim é preciso enviar o sinal da ação desejada. No processo de adaptar esse dispositivo para *IoT ready* se faz necessário componentes que mimetizam a interação humana, e estes componentes se ligam ao microcontrolador para receber um comando esperado e assim ligar/desligar, aumentar/diminuir, alterar o estado que o dispositivo se encontra. Um exemplo clássico é uma cafeteira, a qual tem botões para aumentar ou diminuir a temperatura e iniciar o processo de preparo do café.

Neste contexto, um componente comum é amplamente utilizado. O *relay* também conhecido por relé ou relê (figura 9), é um interruptor eletromecânico projetado por Michael Faraday na década de 1830 com inúmeras aplicações possíveis em comutação de contatos elétricos, servindo para ligar ou desligar dispositivos. De forma superficial, o contato é feito por duas chapas metálicas, uma delas móvel que está conectada a um eletroímã, o qual é energizado para ligar ou desligar o contato. (OMRON, 2023)

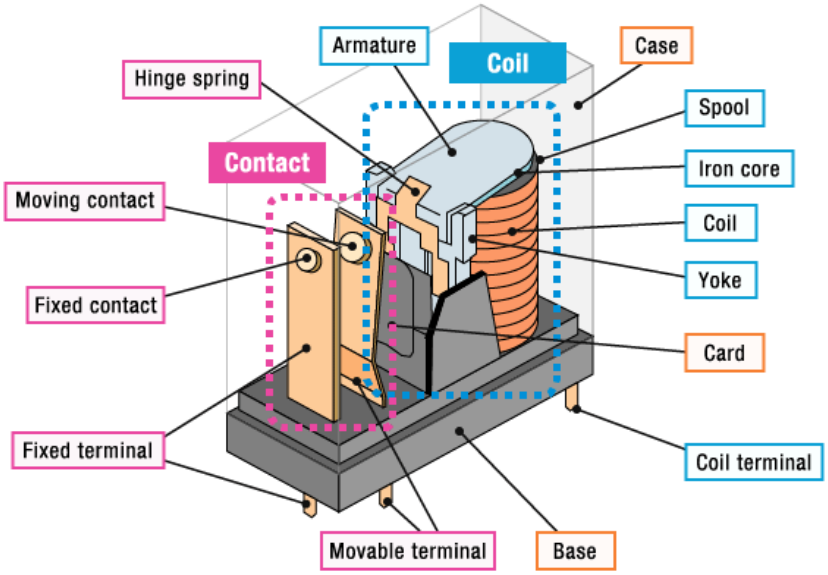

Figura 9 – Esquemático de um relé eletromecânico.(OMRON, 2023)

O módulo relé se faz essencial em algumas configurações pois eletrodomésticos e itens elétricos no geral funcionam com uma tensão comercial padrão de 127V em corrente alternada e as soluções *open hardware* com microcontrolador utilizam 3.3V ou 5V em corrente contínua, portanto para que ocorra o devido funcionamento o relé atua como um interruptor eletromecânico que consegue lidar com alta tensão e ainda receber sinais digitais para ser acionado ou não.

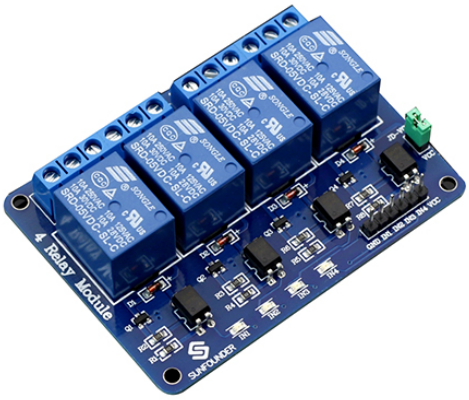

Figura 10 – Módulo relé de 4 canais. (SUNFOUNDER, 2023)

2.6 Conexão do Dispositivo na Rede

É impreterível que o dispositivo esteja conectado para habilitação da *IoT*. O próprio conceito de *IoT* remete a automação de coisas conectadas a Internet. Para isto, a solução *open hardware* precisa de um módulo que se encarregue de se conectar a rede via cabo ou sem fio(*wireless*).

Algumas dependem de um circuito integrado externo para se conectarem, a plataforma Arduino é um exemplo, o shield Ethernet possibilita a comunicação via Internet e uma biblioteca facilita o protocolo de conexão com a rede.

Por outro lado, uma certa gama de soluções já possuem um circuito integrado embutido (*wireless*) capaz de se conectar a redes Wi-Fi e então criar um canal de comunicação entre o usuário e o dispositivo, atingindo um dos pilares e pré-requisitos da *IoT*.

Entrando mais afundo nos possíveis modos de conexão, precisa-se primeiramente categorizar em duas opções, a conexão cabeada (Ethernet) e a conexão wireless (sem fio) que a maioria da população conhece como Wi-Fi. A conexão com fio simplesmente faz a ponte entre o dispositivo e a Internet, permitindo assim que o *open hardware* seja um ponto de acesso (roteador), uma estação ou ambos simultaneamente. Como mostra a lista abaixo:

- Modo Wi-Fi Direct: Neste modo, a solução de *open hardware* pode se conectar a outros dispositivos diretamente, sem a necessidade de um ponto de acesso ou roteador.
- Modo Estação: Neste modo, a solução de *open hardware* se conecta à rede Wi-Fi existente e pode se comunicar com outros dispositivos da rede. (Figura 11)

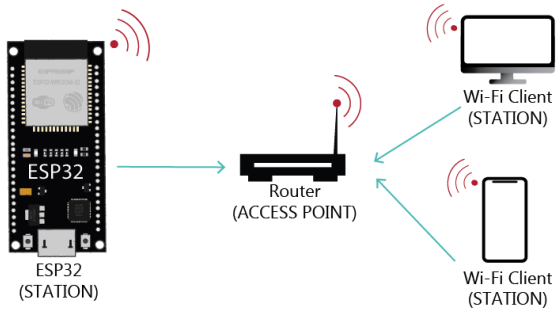

Figura 11 – Modo Access Point. (RANDOMNERDTUTORIALS, 2023b)

- Modo Access Point: Neste modo, a solução de *open hardware* cria sua própria rede Wi-Fi e outros dispositivos podem se conectar a ela. (Figura 12)

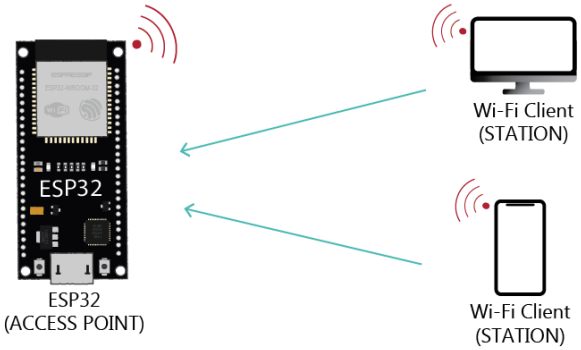

Figura 12 – Modo Access Point. (RANDOMNERDTUTORIALS, 2023b)

- Modo Estação + Access Point: Neste modo, a solução de *open hardware* pode se conectar à rede Wi-Fi existente e criar sua própria rede Wi-Fi ao mesmo tempo.

Em termos de código, tanto o Arduino quanto o ESP32 e soluções de *open hardware* paralelas possuem abstrações construídas em forma de bibliotecas de código para facilitar a conexão. O ponto positivo está em não precisar reescrever sempre o mesmo código e reaproveitá-lo de acordo com a necessidade específica, seja ela conectar ao Wi-Fi, controlar um motor de passo ou acionar um relé. O trecho de código abaixo (código 3) mostra uma possível função escrita na linguagem C++ para se conectar à rede Wi-Fi.

```
1  #include "WiFi.h"
2
3  void initWiFi() {
4      WiFi.mode(WIFI_STA);
5      WiFi.begin(ssid, password);
6      Serial.print("Connecting to WiFi ..");
7      while (WiFi.status() != WL_CONNECTED) {
8          Serial.print('.');
9          delay(1000);
10     }
11     Serial.println(WiFi.localIP());
12 }
```

Código 3: Exemplo de conexão a uma rede Wi-Fi.

O código inicia incluindo a biblioteca WiFi.h que habilita a utilização das funções a seguir no código. Na linha 4 o modo de conexão é definido com a constante WIFI_STA, que representa o modo estação. A seguir na linha 5 os valores para ssid e password são o nome da rede e senha respectivamente para então a função *begin* iniciar o protocolo de conexão. A estrutura condicional *while* na linha 7 é responsável por validar o status da conexão e, enquanto não estiver ativa, o monitor serial mostrará um ponto (.) a cada 1 segundo. Ao final da conexão o IP local atribuído ao microcontrolador é exibido e a conexão está pronta para ser utilizada.

Como constatou-se durante toda a fundamentação teórica, os conceitos e estruturas utilizadas variam bastante a depender do dispositivo em questão, neste capítulo apresentou-se uma forma mais genérica possível a criação e adaptação de um dispositivo não *IoT ready* em *IoT ready* para que leitores e trabalhos futuros utilizem como uma importante fonte de conhecimento sobre *IoT*.

A seguir, no estudo de caso do capítulo 4, os principais conceitos apresentados no capítulo 3 serão implementados para exemplificação.

3 Metodologia de Pesquisa e Desenvolvimento

O método desta pesquisa tange a hipótese levantada anteriormente. A abordagem escolhida inicia-se com um estudo do estado da arte de *IoT* voltado para o nicho domiciliar e em seguida, pesquisas de conteúdo e materiais que apresentam possíveis soluções tecnológicas de como chegar do ponto A ao ponto B, ou seja, como de fato converter um dispositivo para que ele se torne *IoT ready*. Após isto, uma pesquisa quantitativa dos itens *IoT ready* disponíveis no mercado foi realizada em diferentes categorias. Esta pesquisa inicial foi feita sobretudo, online, consultando sites de vendedores e fabricantes. O levantamento consolidou os dados que podem ser observados na Tabela 3.3. Por fim, as duas últimas seções discutem um modelo que, até certo ponto, é o padrão para qualquer solução *IoT*.

3.1 Estado da Arte de *IoT* para Automação Domiciliar

Embora existam alguns dispositivos para automação domiciliar já disponíveis no mercado, e uma lista mais detalhada desses produtos comerciais serão apresentados na seção 2.3, o mercado ainda carece de muitos deles. Como veremos na seção 2.3, os que estão atualmente disponíveis no mercado são de nichos específicos, caros, e por isso motivam entusiastas da área a criarem soluções caseiras para tornar os artefatos *IoT ready*.

Portanto, nota-se que todo o ecossistema está populado de soluções caseiras para tornar dispositivos *IoT ready*. Como será apresentado no decorrer deste trabalho, existem invenções no mercado e várias soluções caseiras, porém essas soluções não são padronizadas. Este trabalho tem como um dos objetivos apresentar o panorama geral do que seria *IoT* para automação domiciliar, suas partes/componentes e um guia/plano genérico para quem deseja tornar um dispositivo *IoT ready*, apresentando diversas possibilidades e os meios para a correta execução do mesmo.

Para (De Silva; MORIKAWA; PETRA, 2012), os seres humanos interagem com o ambiente que os cerca de inúmeras maneiras. Eles percebem as condições ambientais e agem, reagem ou ajustam-se de acordo. Se o ambiente puder ser feito para retribuir esse comportamento e responde-lo, este trará muitas vantagens. Tal conduta pode automatizar várias tarefas que os humanos devem realizar manualmente e também fornecer novos serviços e facilidades. Uma casa inteligente é um ambiente caseiro que possui inteligência ambiental e controle automático, os quais permitem responder ao comportamento dos moradores e fornecer a eles diversas facilidades. Em síntese, *IoT* de maneira simples, é

interconectar tudo ao nosso redor via a Internet.

Já para (SARACINO, 2020), a automação domiciliar está assumindo um papel de destaque no universo da Internet das Coisas (*IoT*). A importância e o impacto positivo que dispositivos domésticos inteligentes e interconectados podem ter na vida das pessoas, permitiram que produtos caseiros e serviços de casa tenham uma grande difusão tanto no cenário nacional, quanto internacional.

Ainda segundo um levantamento de diversos dispositivos para automação domiciliar feito por (SARACINO, 2020) e o Observatório da Politécnica de Milão, foram criadas 698 novas soluções de *smart home* no mercado entre os anos 2000 e 2020. E uma análise sobre os dados foi feita no sentido de obter um retrato macro do mercado *IoT* domiciliar.

Vários aspectos deste levantamento foram abordados, o primeiro é um aspecto geográfico. Considerando o ocidente e o oriente a distribuição geográfica da Europa, América e Ásia (China), esses são responsáveis por 80% do mercado domiciliar *smart*.

O segundo aspecto foi a distribuição de oferta entre os diferentes tipos de empresas (*Big player, Small and Medium Enterprises (SME), Startups*. A evidência é que a maioria dos produtos/serviços são oferecidos por Gigantes do mercado e *Startups*.

Um outro aspecto interessante no que diz respeito à analise entre as amplas áreas de aplicação existentes, a categoria mais importante no mercado é a **segurança**. Uma vez que corresponde à necessidade primária das pessoas, seguida por gerenciamento de eletrodomésticos e cenários da casa.

3.2 Soluções Tecnológicas para Tornar um Dispositivo *IoT ready*

Com base nessa pesquisa inicial das soluções caseiras para *IoT* estudou-se conceitos e as tecnologias necessárias para atingir o objetivo citado anteriormente, pois assuntos como soluções *open hardware* (Arduino, ESP, Raspberry), interface analógica-digital e comunicação via rede são ou não cobertos em um nível mais raso em um currículo tradicional de Ciência da Computação. Sendo assim, tais assuntos foram pesquisados e maiores informações sobre eles são apresentados no capitulo 3 de fundamentação teórica para embasar e validar todo o desenvolvimento do estudo de caso no capítulo 4.

Tornar um dispositivo *IoT ready* tende a ser um processo técnico, mas pode ser dividido em algumas etapas:

1. Escolher uma solução *open hardware* com um microcontrolador ou um processador (*single-board* placa única): Um microcontrolador é um pequeno computador em um único circuito integrado que pode controlar outros dispositivos. Um computador *single-board* é uma máquina completa em uma única placa de circuito. Ambos podem

ser usados como o cérebro de um dispositivo *IoT*.

2. Adicionar sensores e atuadores: Os sensores podem coletar dados sobre o ambiente ou o próprio dispositivo, enquanto os atuadores podem controlar vários dispositivos ou componentes. Você pode escolher entre uma variedade de sensores e atuadores para atender às suas necessidades específicas.

3. Conectar o dispositivo à Internet: Isso pode ser feito por Wi-Fi ou conectividade celular (4G/5G). Pode-se usar um módulo Wi-Fi ou um módulo celular (SIM/eSIM) para permitir que o dispositivo se conecte à internet.

4. Escolha uma plataforma para criar seu aplicativo de *IoT*: Há muitas plataformas de *IoT* disponíveis, pode-se escolher uma que atenda às necessidades do dispositivo em questão.

5. Desenvolver o software: Precisa-se desenvolver o software para conectar o dispositivo à internet, enviar dados para a nuvem e receber comandos da nuvem.

6. Segurança e Infraestrutura: É necessário preocupar-se com a exposição do dispositivo e os dados trafegados na rede, assim como a infraestrutura que ele depende, como por exemplo: a alimentação, a redundância e a persistência dos dados registrados, além de recuperação de falhas e intempéries inerentes ao dispositivo.

7. Testar e implantar: Realizar testes de acordo com o comportamento esperado e cenários de uso do dispositivo. Gerar e implantar uma versão 1.0, mapeando e gerando um rastro de futuras alterações, correções e melhorias no código.

Diante destes 7 pontos citados há várias trilhas de estudo que podem ser seguidas e aprofundadas caso se deseje uma compreensão mais profunda e completa. Mais uma vez, este trabalho concentrou-se em alguns conceitos importantes que fazem parte destas trilhas. A lista a seguir detalha cada um dos 7 pontos respectivamente.

1. Microcontroladores e computadores (*single-board*): Deve-se ter uma boa compreensão dos diferentes tipos de microcontroladores utilizados em soluções *open hardware* e computadores de placa única disponíveis, como Arduino, Raspberry Pi e ESP32, conhecer os componentes e o funcionamento de cada recurso do dispositivo.

2. Sensores e atuadores: Deve-se conhecer os diferentes tipos de sensores e atuadores que podem ser usados em dispositivos *IoT*, como sensores de temperatura, sensores de luminosidade, sensores de umidade, sensores de movimento e servomotores.

3. Conectividade com a Internet: Deve-se ter conhecimento de como os dispositivos podem se conectar à Internet, incluindo Wi-Fi e conectividade celular, e a função de protocolos como MQTT(opcional) e HTTP.

4. Plataformas *IoT*: Deve-se entender as plataformas *IoT*, como PlatformIO, *AWS IoT*, *Google Cloud IoT* e *Microsoft Azure IoT*, e como elas podem ser usadas para gerenciar e processar dados de dispositivos *IoT*.

5. Linguagens de Programação: Precisa-se ter conhecimento sobre as linguagens de programação, sendo elas: C/C++, Python(opcional) e JavaScript, bem como plataformas como Node-RED(opcional) e Arduino IDE.

6. Segurança e Infraestrutura: Deve-se estar ciente dos riscos de segurança em dispositivos *IoT* e como mitigá-los, incluindo autenticação segura, criptografia e controle de acesso. Em relação à infraestrutura é fundamental garantir a consistência do estado do dispositivo e como se recuperar caso algo fora do controle aconteça, por exemplo, uma queda de energia pode causar um *reset* do dispositivo para seu estado original.

7. Testar variações e casos de uso do dispositivo e gerar uma versão final do código que na verdade ao final de todo processo se tornará um *firmware* pois não será alterado se caso tivessem N dos mesmos dispositivos.

Ao estudar esses tópicos, o leitor estará equipado com o conhecimento, de certa forma avançado, necessário para modificar um dispositivo e torná-lo *IoT ready*.

3.3 Levantamento de Dispositivos *IoT ready*

Como já foi citado, existem dispositivos comercialmente disponíveis e a pesquisa levou tal fato em consideração assim como na construção da tabela 3.3, a qual concentra dispositivos e suas respectivas categorias, em adição a valores praticados no ano corrente.

Tabela 1 – Dispositivos *IoT* disponíveis no mercado.

	Item	Preço (2021)
	Geladeira	
1	Geladeira Smart LG Side by Side Inverter 601 Litros	R$ 9.999,90
2	Refrigerador French Door Samsung Soundbar, Smart Wi-Fi	R$ 28.999,90
	Iluminação	
3	Smart Lâmpada Led Colors, Wi-FI - Elgin, compatível com Alexa	R$ 59,90
4	Philips Hue White & Color Ambiance - Wifi E Bluetooth, compatível com Alexa	R$ 329,90
5	Luminária Inteligente Elgin Wi-Fi, de Embutir, compatível com Alexa	RS 169,90
	Ar condicionado	
6	Ar-Condicionado Split LG Dual Inverter Voice, Wi-Fi	R$ 2.249,00
	Aspirador de pó	
7	Aspirador de Pó Robô KaBuM! Smart 100	R$ 649,90
8	Aspirador de Pó Robô WAP ROBOT W100	R$ 502,55
	Tomada	
9	Smart Plug Max Wi-Fi Positivo Casa Inteligente – Compatível com Alexa	R$ 132,72
10	NOVADIGITAL Tomada Inteligente Wifi Smart Home, Compatível com Alexa	R$ 76,90
	Termostato	
11	Tuya WiFi LCD Display termostato inteligente compatível com Alexa Google Home	R$ 199,90
	TV	
12	Smart TV 4K LG LED 50” Google Assistente, Alexa, Controle Smart Magic e Wi-Fi	R$ 2.699,90
13	Samsung Smart TV UHD 4K 50"4K, Controle Único, Alexa Built in e Wi-Fi	R$ 2.799,90
14	Smart TV 50” 4K UHD QLED TCL Wi-Fi	R$ 2.690,90
	Máquina de lavar	
15	Lava e Seca Samsung WD6000 Smart (Wi-Fi) - 11kg	R$ 4.699,00
16	Lava e Seca Smart Victor 17kg Automática Abertura Frontal LG	R$ 9.344,00
	Câmera de segurança	
17	Smart Câmera 360º Bot Wi-Fi Positivo Casa Inteligente – Compatível com Alexa	R$ 309,90
18	Câmera de Monitoramento 360º, Wi-Fi Full HD, Tapo C200, TP-Link	R$ 449,00
	Fechadura	
19	Fechadura Digital SHS-1321 Samsung Smart Home	R$ 1.199,00
	Assistentes Virtuais	
20	Echo Dot (4ª Geração): Smart Speaker com Alexa	R$ 379,00
21	Nest Mini 2ª geração Smart Speaker - com Google Assistente	R$ 314,10
	Interruptores	
22	Smart Interruptor Wi-Fi, Positivo Casa Inteligente, Compatível com Alexa, Branco	R$ 199,00
23	Interruptor Tuya Wifi Ventilador De Teto Alexa/google Smart	R$ 210,00
24	Interruptor Novadigital WSUS3B	R$ 148,32

Com base na tabela acima, o número de dispositivos *IoT ready* disponíveis no mercado deixa a desejar e não atende o anseio da sociedade em geral. Outro ponto observável são os valores quando comparados a um mesmo item não *IoT ready*.Uma lâmpada

de LED comum (10W) pode variar de dez a vinte reais, enquanto a lâmpada mais barata (item 3) pode chegar a custar até seis vezes mais.

As figuras 13 e 14 abaixo ilustram os valores de duas categorias de itens, lâmpadas e televisores, ambos nas suas versões *IoT ready* e não *IoT ready* sendo esta última, a mais popular com serviços *IoT* como *streamings*, controle por voz, espelhamento de smartphone, entre outros.

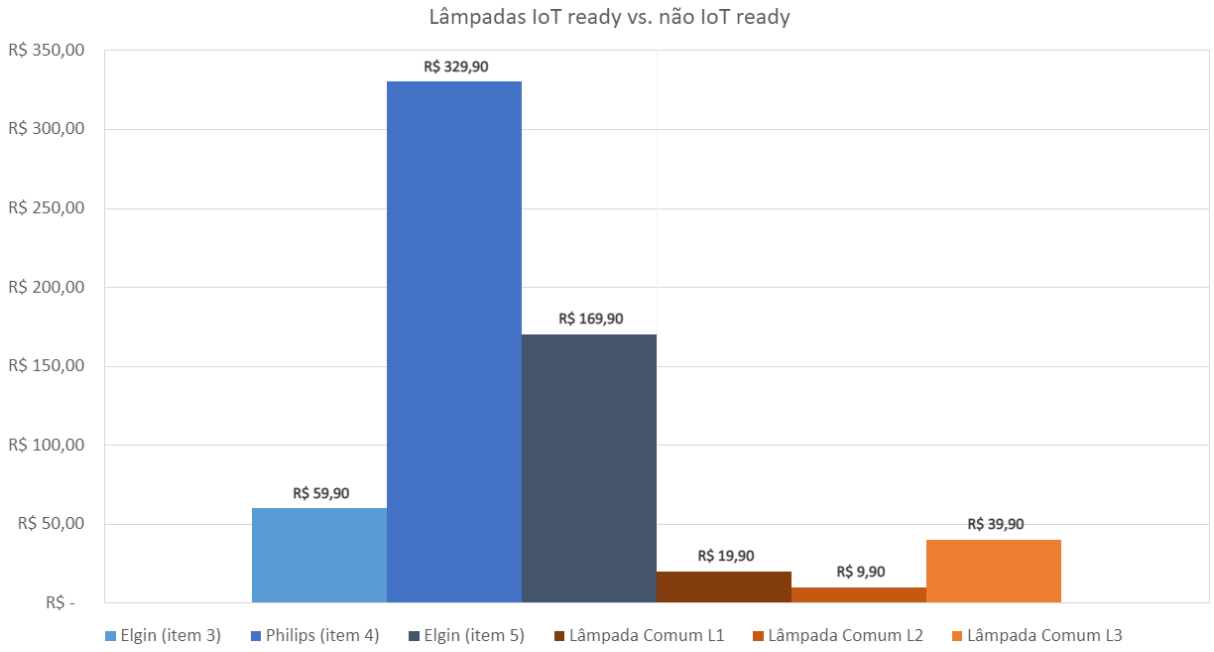

Figura 13 – Gráfico lâmpadas *smart* vs lâmpadas comuns.

A partir dos itens da tabela exibida e diante destes gráficos surge uma pergunta a ser discutida: Como mensurar se compensa pagar a diferença de preço por um dispositivo *IoT ready*? E a resposta deste questionamento é relativo para cada um de nós. Decidir se o alto valor de um dispositivo *IoT* fará uma grande diferença na vida cotidiana é pessoal, entretanto, quanto mais pessoas adquirirem o produto, menor se tornará esta lacuna de valores e maior será a distância da tecnologia inserida cada vez mais no nosso cotidiano, pois essa se tornará cada vez mais acessível ao longo dos anos. Dois exemplos próximos do nosso dia-a-dia são as impressoras e as Smart TVs que, ao decorrer do tempo se tornaram cada vez mais conectadas e inteligentes.

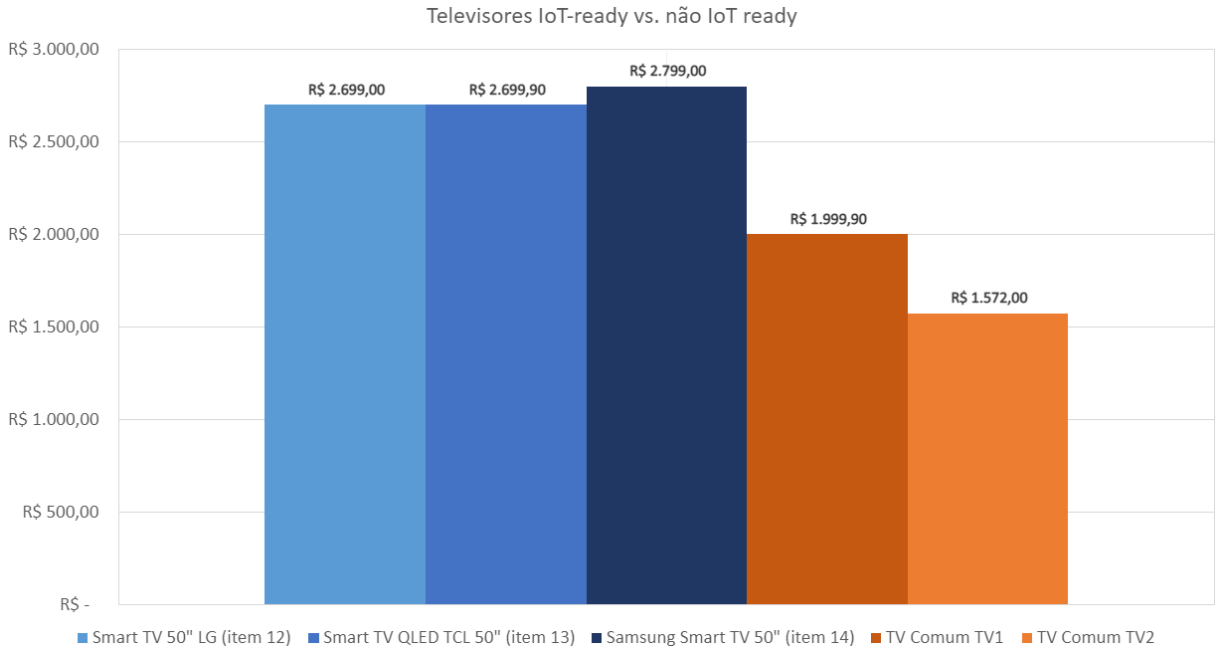

Figura 14 – Gráfico TV *smart* vs TV comuns.

Nota-se que existe uma prevalência de diferentes dispositivos ofertados com tecnologia *IoT*. Os gráficos apontam uma grande discrepância nos valores das lâmpadas, isso pode ser respondido pelo fato do custo de produção delas serem muito baratos e qualquer mudança ou adição no processo incrementaria o custo de produção, além do fato de que os softwares embarcados naquele dispositivo não são escaláveis. Do lado dos televisores, a opção *smart* (*IoT*) é a mais popular e seu preço não se distancia largamente do mesmo item sem recursos inteligentes, isso acontece pois o custo adicional do suporte à *IoT* se dilui no preço total da TV. Embora exista dispositivos *IoT ready* disponíveis no mercado a relação custo/benefício não é atrativa na maioria dos casos, a tabela e os gráficos refletem alguns exemplos. Sendo assim, para a maioria se torna inviável a compra desses dispositivos inteligentes uma vez que o custo pode duplicar, ou até mesmo triplicar.

Provavelmente não será o mercado de eletrodomésticos *IoT ready* que popularizará a *IoT* como um todo. Isso tende a ocorrer em outras frentes primeiro, para que só então o preço da tecnologia diminua e seja possível a produção em escala e a incorporação tecnológica nesse tipo de produto, não elevando de forma drástica o preço do mesmo.

A pesquisa continuou buscando formas pré-estabelecidas para tornar um dispositivo não *IoT ready* em *IoT ready* a fim de descobrir novas formas de tornar um dispositivo comum conectado a Internet e balancear os custos junto a curva de aprendizagem necessária durante o estudo. Para isso, algumas dependências são inerentes ao processo e a principal delas é conectividade à Internet. Em termos de software, duas interfaces precisaram ser escritas, uma que age como um servidor Web e outra que é a tela propriamente

dita, a qual enviará os comandos à este servidor para acionar as devidas ações.

Com base no levantamento feito nas seções 2.1, 2.2 e 2.3, o texto aqui apresentado foi descrito buscando apresentar o quadro geral de *IoT* no âmbito domiciliar no ano corrente. Ademais, apresentamos uma espécie de *blueprint* (diagrama) para que novas soluções de *IoT* desenvolvidas de modo caseiras possam ser feitas de maneira assistiva para o leitor. A fim de materializar a *blueprint* apresentada, um estudo de caso foi realizado em que automatizou-se um ventilador de mesa e todo o custo associado ao processo foi discutido nos sub-capítulos consecutivos ao estudo de caso.

A seguir, no capítulo 3, toda teoria é apresentada com detalhes das tecnologias utilizadas para tornar um dispositivo *IoT ready* com o intuito de criar uma conexão entre a fundamentação teórica e o estudo de caso.

4 Estudo de Caso

O passo-a-passo denotado neste trabalho não é trivial para a maioria do público-alvo, contudo, não significa que é impossível realizá-lo. Se trata de uma abordagem caseira e utiliza conceitos de elétrica, eletrônica e computação para o devido funcionamento. Os conceitos aplicados não precisam necessariamente de uma graduação, um curso técnico, ou um conhecimento aprofundado na área, é totalmente plausível realizar o procedimento sem tais alicerces.

Durante a graduação, o conhecimento adquirido nas disciplinas de Arquitetura de Redes de Computadores, Arquitetura de Redes TCP/IP, Programação para Internet, Algorítimos e estrutura de Dados I e II, Sistemas Digitais, Programação orientada a objeto e Programação Procedimental foram cruciais para a realização deste trabalho. Além disso, a filosofia de aprendizado vista na universidade facilitou a resolução de problemas e a busca por soluções otimizadas sem perder o foco inicial de controlar e tornar um dispositivo *IoT ready*.

Para o estudo de caso abordou-se a adaptação de um ventilador padrão com 3 possíveis velocidades tornando-o *IoT ready*, assim sendo será apresentado na seguinte ordem:

1. Esquemático geral para simular/validar o devido funcionamento;
2. Montagem do circuito eletrônico;
3. Ambiente de desenvolvimento;
4. Criação do projeto;
    - 4.1. Estrutura inicial do código;
        - 4.1.1. Build, Run e Monitor;
    - 4.2. Ligar/desligar ventilador;
    - 4.3. Controle das 3 velocidades;
    - 4.4. Conexão de rede via Wi-Fi (servidor);
        - 4.4.1. SPI Flash File System;
    - 4.5. Página Web para controle do ventilador (cliente);
    - 4.6. Controle via requisição HTTP;
5. Análise do estudo de caso

4.1 Esquemático Geral

A figura 15 representa as conexões entre a ESP32, o módulo relé e o ventilador. Pode-se observar que um dos fios do ventilador (linha vermelha) é conectado diretamente ao relé e na sequência são feitos dois *jumpers* para o reaproveitamento da conexão. Os outros 3 fios que controlam a velocidade do ventilador serão ligados, respectivamente, nos relés. Do lado da ESP32 identifica-se as duas conexões de alimentação para o módulo relé *Vin* e *GND*, e as outras 3 conexões controlam o acionamento de cada relé separadamente.

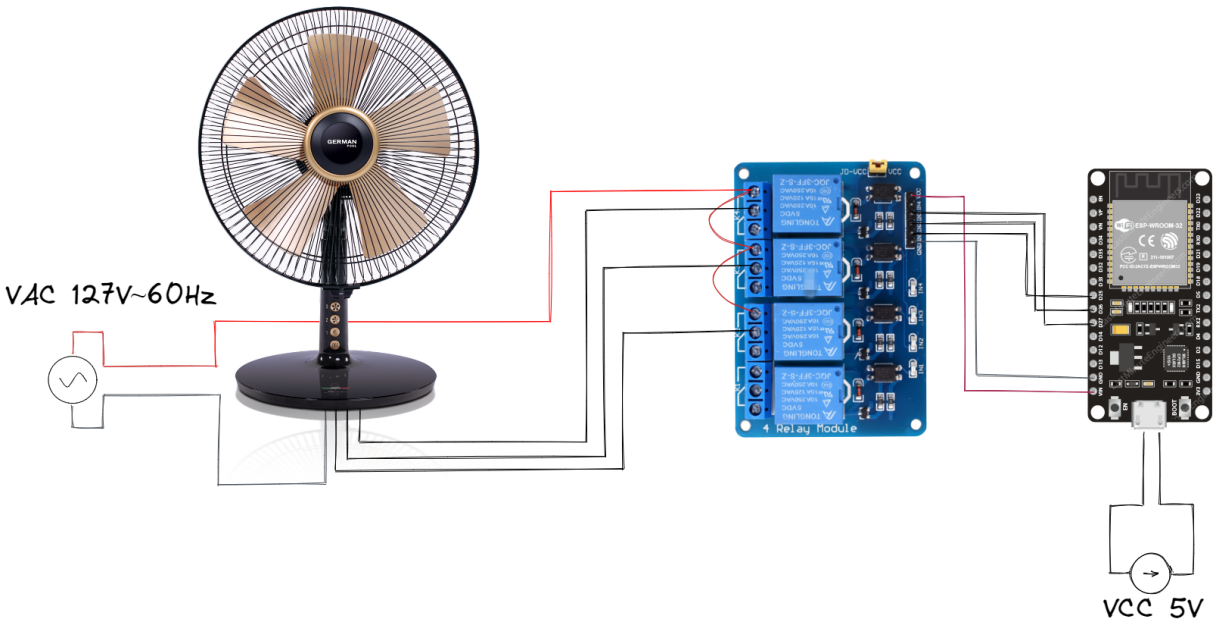

Figura 15 – Esquemático geral.

É válido a ressalva que, devido a energia elétrica no Brasil ser de corrente alternada, a imagem 15 ilustra uma possível opção, isto é, a linha com a cor vermelha nem sempre representará o positivo (fase). Assim como a linha com a cor preta poderá ser, ou não, o negativo (neutro).

4.2 Montagem do Circuito Eletrônico

Utilizou-se uma *protoboard* (placa de prototipação) para facilitar as conexões, sendo a alimentação da ESP32 feita por uma porta microUSB 5V padrão. Todo o conjunto pode ser observado na figura 16. Com todo o circuito montado, o próximo passo é a programação do microcontrolador, ou seja, as instruções necessárias para que ele desligue e ligue cada relé individualmente alternando a velocidade e desligando o ventilador conforme desejado.

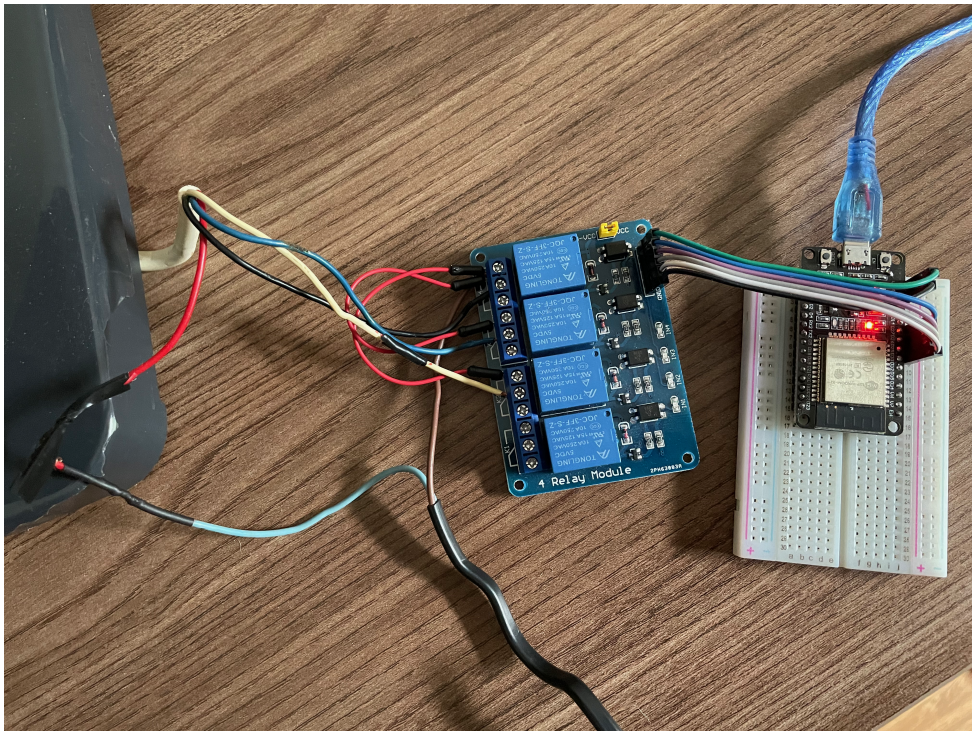

Figura 16 – Esquemático implementado.

4.3 Ambiente de Desenvolvimento

O Arduino citado anteriormente, possui uma IDE própria e largamente difundida entre os entusiastas desse ecossistema. Entretanto, neste trabalho utilizou-se outra IDE, que vem ganhando espaço entre os programadores, o Visual Studio Code, abreviando-se para VS Code. Desenvolvido pela Microsoft, essa IDE consiste em um editor de texto com recursos que auxiliam o desenvolvimento e entendimento do código em si, além de possuir extensões desenvolvidas por empresas e pela comunidade para linguagens e aplicações específicas.

No caso deste trabalho, instalou-se a PlatformIO que é um conjunto de ferramentas para desenvolvimento de sistemas embarcados em C/C++. Nas primeiras versões disponibilizadas se tratava de uma plataforma que cobrava para utilizar todas as funcionalidades, porém em junho de 2019 foi liberado como código *open source* e com todas as funcionalidades gratuitas. O PlatformIO é mais comumente utilizado na forma de extensão para o VS Code.

A instalação do VS Code é indexada nos diversos buscadores da Internet como Google, Bing e etc, ou através do site oficial[1]. A partir desse ponto é necessário realizar o download para o respectivo sistema operacional (Windows, Linux ou Mac) e prosseguir normalmente com a instalação.

[1] https://code.visualstudio.com/

Assim que o VS Code for instalado, clique no botão de extensões e digite no campo de busca "platformio". Selecione a extensão do PlatformIO e clique no pequeno botão verde instalar. Essa etapa é denotada na figura 17.

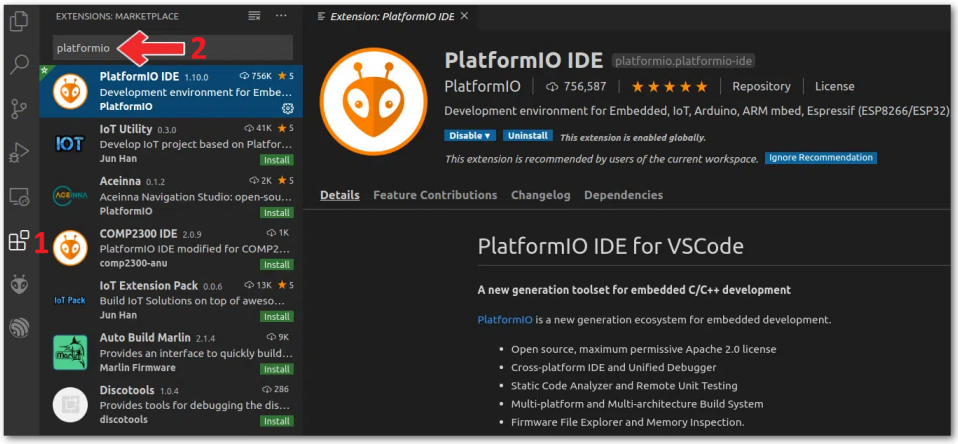

Figura 17 – Instalação da extensão PlatformIO no VS Code.

4.4 Criação do Projeto na IDE

Para criar o projeto, a PlatformIO oferece um ferramenta que permite a inserção das informações da placa (Arduino, ESP32, ESP8266, etc) sendo então gerado uma estrutura pré-definida de pastas. O passo-a-passo é ilustrado nas figuras a seguir.

Primeiramente, deve-se clicar no menu lateral à esquerda com o ícone do PlatformIO e em seguida, no novo menu com a opção *QUICK ACCESS* escolha a opção *Open* como pode ser visto na figura 18.

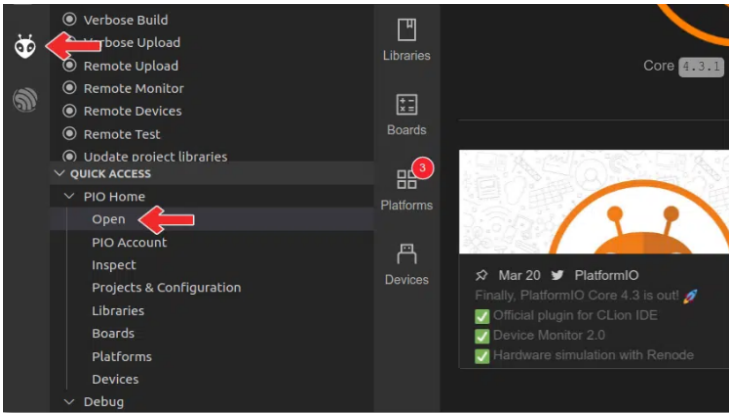

Figura 18 – Criação de projeto via PlatformIO.

Após acessar a página inicial do PlatformIO, clique no botão *+ New Project*. A figura 19 pode ser utilizada como referência.

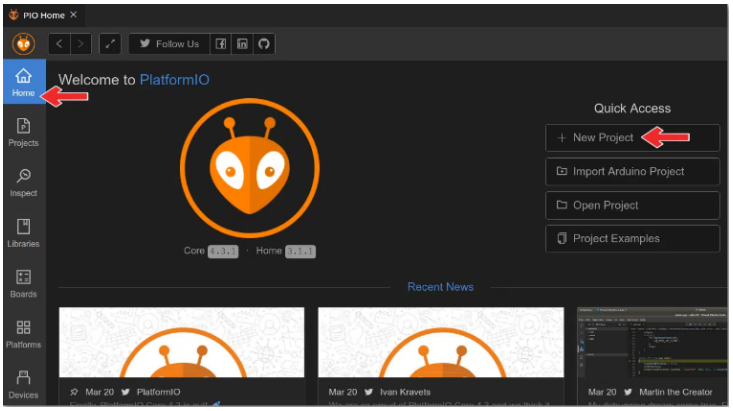

Figura 19 – Menu PlatformIO.

O Visual Studio Code abrirá então um *popup* com os parâmetros citados anteriormente. Preencha os campos como a figura 20.

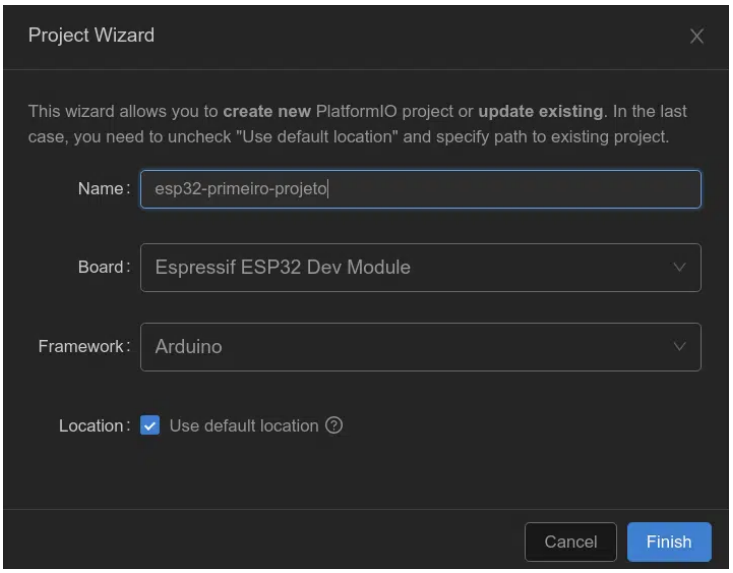

Figura 20 – Parâmetros de entrada do projeto.

Após clicar em *Finish*, os arquivos e pastas necessários serão criados no *workspace* (figura 21). O mais importante desse procedimento é o arquivo *main.cpp* que conterá seu código inicial e está dentro da pasta */src*.

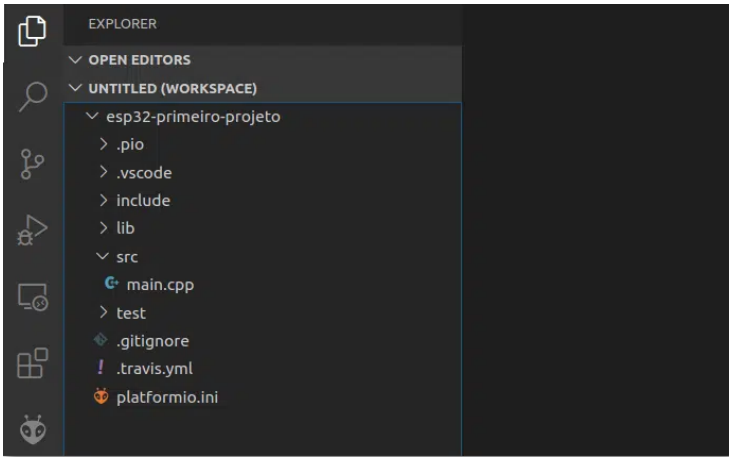

Figura 21 – Estrutura de pastas criada.

4.4.0.1 Build, Run e Monitor

O conjunto de ferramentas do PlatformIO auxilia no encapsulamento das tarefas de compilação e upload do código para a ESP32, para isso é necessário editar o arquivo *platformio.ini*, na raiz do projeto, lá define-se três parâmetros fundamentais, sendo estes: porta serial, velocidade de upload e velocidade de escrita e leitura do monitor serial. Como pode ser visto na lista 4.

```
1  ; PlatformIO Project Configuration File
2  ;
3  ;   Build options: build flags, source filter
4  ;   Upload options: custom upload port, speed and extra flags
5  ;   Library options: dependencies, extra library storages
6  ;   Advanced options: extra scripting
7  ;
8  ; Please visit documentation for the other options and examples
9  ; https://docs.platformio.org/page/projectconf.html
10
11 [env:esp32dev]
12 platform = espressif32
13 board = esp32dev
14 framework = arduino
15 monitor_speed = 115200
16 upload_speed = 921600
17 upload_port = /dev/cu.usbserial-0001
```

Código 4: Arquivo platformio.ini.

A depender do sistema operacional a porta serial pode ser diferente, no Windows pode ser visualizado no **Gerenciador de dispositivos**, já em sistemas baseados em UNIX como Linux e MacOS o comando **dmesg** no terminal revela a porta, como algo desta forma: /dev/ttyUSB0 ou dev/cu.usbserial-0001. As linhas 15 e 16, dizem respeito

à taxa de comunicação entre o ESP32 e o PC, conhecido como *baud rate*[2], os valores são medidos em *bps* (bits por segundo).

Para compilar o código (Build), fazer upload e utilizar o monitor serial, existem botões de comando na parte inferior do VS Code exibidos na imagem 22.

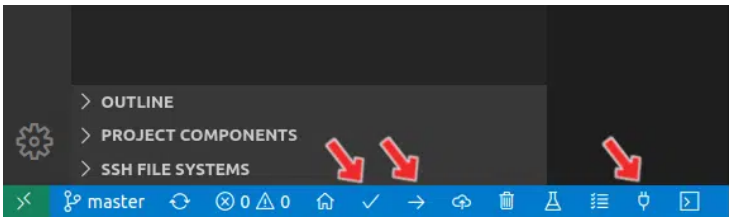

Figura 22 – Barra de ferrramentas VS Code.

O build (compilação) pode ser verificado no terminal do próprio VS Code (Figura 23).

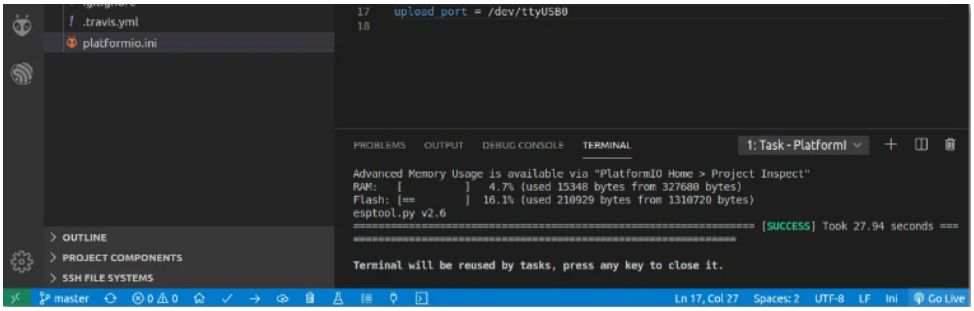

Figura 23 – Compilação executada com sucesso.

Com a placa ESP32 conectada ao computador via cabo USB, usou-se o botão Upload para gravar o programa pisca LED. De acordo com o passo-a-passo, reparou-se que o LED da ESP32 começou a piscar indefinidamente com intervalos de 1 segundo.

4.4.1 Ligar/Desligar Ventilador

O próximo passo se resume na integração do módulo relé com a ESP32 e o código responsável por ligar e desligar o ventilador. Com as conexões da Figura 15 realizadas, a alteração se dá somente via código. Portanto, teremos a seguinte mudança no código 4.

[2] Baud rate é o número de vezes que um sinal em um canal de comunicação muda seu estado, ou varia. Por exemplo, 2400 baud rate, significa que o canal pode mudar o estado até 2400 vezes por segundo. O termo "mudar estado" significa que ele pode variar de 0 para 1 ou de 1 para 0 até X vezes (nesse caso 2400) por segundo. Isto também refere-se ao estado atual da conexão, como voltagem, frequência ou nível de fase.

```
1  #include <Arduino.h>
2
3  // Set RELAY GPIO
4  const int onePin = 27;
5  const int twoPin = 26;
6  const int threePin = 25;
7
8  void reset()
9  {
10   digitalWrite(onePin, HIGH);
11 }
12
13 void setup()
14 {
15   // Serial port for debugging purposes
16   Serial.begin(115200);
17   pinMode(onePin, OUTPUT);
18
19   reset();
20 }
21
22 void loop() {
23   digitalWrite(onePin, LOW);
24   delay(2000);
25   reset();
26 }
```

Código 5: Ligar/desligar ventilador.

O código 5 adiciona a parametrização das portas de comunicação da ESP32 com o módulo relé, isto é, as variáveis das linhas 4, 5 e 6 representam as saídas digitais dos pinos 27, 26 e 25 respectivamente, a função *reset()* é responsável por enviar o sinal para desligar o ventilador. Dentro da função *setup()* designamos o pino 27 como saída digital e na função *loop()* liga-se e desliga-se o ventilador a cada 2 segundos.

4.4.2 Controle das 3 Velocidades

Para o devido controle das três velocidades possíveis, a mudança no Código 6 é pequena, basta expandir os pinos e controlá-los via código, note que o Código 6 apresentado é semelhante ao passo anterior.

```
#include <Arduino.h>

// Set RELAY GPIO
const int onePin = 27;
const int twoPin = 26;
const int threePin = 25;

void reset()
{
  digitalWrite(onePin, HIGH);
  digitalWrite(twoPin, HIGH);
  digitalWrite(threePin, HIGH);
}

void setup()
{
  // Serial port for debugging purposes
  Serial.begin(115200);
  pinMode(onePin, OUTPUT);
  pinMode(twoPin, OUTPUT);
  pinMode(threePin, OUTPUT);

  reset();
}

void loop() {
  digitalWrite(onePin, LOW);
  delay(2000);
  reset();
  digitalWrite(twoPin, LOW);
  delay(2000);
  reset();
  digitalWrite(threePin, LOW);
  delay(2000);
  reset();
}
```

Código 6: Controle das 3 velocidades.

É importante ressaltar que até o momento o código simplesmente ligou e desligou o ventilador com um preciso intervalo de tempo. Nas duas próximas sessões a ESP32 será conectada à Internet e uma página Web será utilizada para controlar cada velocidade independentemente.

4.4.3 Conexão de Rede Via Wi-Fi (Servidor)

A ESP32 possui recurso de conexão Wi-Fi embutido, portanto não precisamos de nenhum circuito integrado (CI) adicional para conectá-la à rede. Neste estudo de caso utilizou-se uma arquitetura cliente-servidor, onde a ESP32 se comporta como um servidor WEB e recebe requisições(HTTP) dos clientes através de um navegador web

como o Google Chrome, Microsoft Edge ou Mozilla Firefox. A Figura 24 ilustra o modo de conexão da ESP32 à rede.

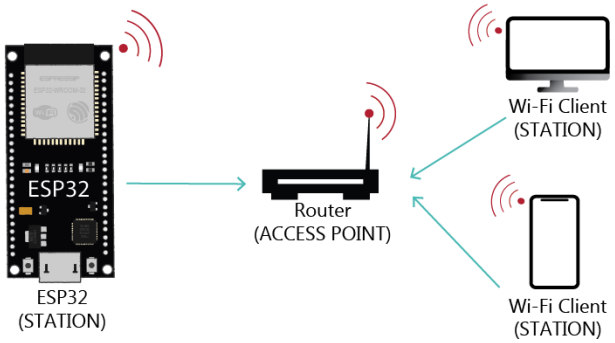

Figura 24 – Conexão Wi-Fi ESP32.

Para utilizar o Wi-Fi da ESP32 é necessário incluir uma biblioteca, a *WiFi.h*. E para a criação do servidor utilizou-se a *ESPAsyncWebServer.h* que abstrai a complexidade e disponibiliza uma *Application Interface Program*(API) para criar um servidor Web assíncrono, permitindo atender mais de uma requisição simultaneamente. Paralelamente, outra biblioteca tornou-se parte do estudo de caso, auxiliando o uso do sistema de arquivos flash da ESP32, chamado SPIFFS.

4.4.3.1 SPI Flash File System

O ESP32 contém um *Serial Peripheral Interface Flash File System* (SPIFFS), traduzido como sistema de arquivos flash de interface periférica serial. SPIFFS é um sistema de arquivos leve criado para microcontroladores com um chip flash, conectado pelo barramento SPI, da mesma forma que a memória flash da ESP32. O SPIFFS permite com que você acesse a memória flash, como faria em um sistema de arquivos usual em seu computador, mesmo sendo de forma mais simples e limitada. Você pode ler, gravar, fechar e excluir arquivos. O SPIFFS não suporta diretórios, então tudo é salvo em uma estrutura plana. O uso do SPIFFS com a placa ESP32 é especialmente útil para:

- Criar arquivos de configuração;
- Salvar dados permanentemente;
- Criar arquivos para salvar pequenas quantidades de dados em vez de usar um cartão microSD;
- Salvar arquivos HTML, CSS e JavaScript para construir um servidor web;
- Salvar imagens, figuras e ícones;

Para isso é necessário a alteração do arquivo *platformio.ini* afim de contemplar estas bibliotecas citadas como dependências do projeto. Vide o arquivo *platformio.ino* final no código 7.

```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter
;   Upload options: custom upload port, speed and extra flags
;   Library options: dependencies, extra library storages
;   Advanced options: extra scripting
;
; Please visit documentation for the other options and examples
; https://docs.platformio.org/page/projectconf.html

[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200
upload_speed = 921600
upload_port = /dev/cu.usbserial-0001
lib_deps=
    Wifi
    FS
    SPIFFS
    https://github.com/me-no-dev/ESPAsyncTCP.git
    https://github.com/me-no-dev/ESPAsyncWebServer.git
lib_ldf_mode = deep
```

Código 7: Arquivo platformio.ini.

Para fins didáticos o código final do servidor foi dividido em partes. Nesta primeira parte (código 8) nota-se a inclusão das bibliotecas e novas variáveis com o nome e senha da rede Wi-Fi, vale ressaltar a instanciação do servidor na linha 16.

```
#include <Arduino.h>
#include <SPIFFS.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char *ssid = "WiFi";
const char *password = "senhaWifi";

// Set RELAY GPIO
const int onePin = 27;
const int twoPin = 26;
const int threePin = 25;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);
```

Código 8: Código do servidor - parte 1.

Na segunda parte (código 9) observa-se uma nova função que trata as informações não encontradas no servidor, geralmente retornando o erro 404. A função *reset()* continua a mesma do código 6.

```
void notFound(AsyncWebServerRequest *request)
{
  request->send(404, "text/plain", "Not found");
}

void reset()
{
  digitalWrite(onePin, HIGH);
  digitalWrite(twoPin, HIGH);
  digitalWrite(threePin, HIGH);
}
```

Código 9: Código do servidor - parte 2.

A terceira parte (código 10) possui várias atualizações dentro da função *setup()*, a inicialização do SPIFFS e a conexão a rede Wi-Fi, que realiza uma tentativa a cada um segundo e quando a conexão é fechada, o IP local é impresso no terminal.

```
30  void setup()
31  {
32    // Serial port for debugging purposes
33    Serial.begin(115200);
34    pinMode(onePin, OUTPUT);
35    pinMode(twoPin, OUTPUT);
36    pinMode(threePin, OUTPUT);
37
38    reset();
39
40    // Initialize SPIFFS
41    if (!SPIFFS.begin(true))
42    {
43      Serial.println("An Error has occurred while mounting SPIFFS");
44      return;
45    }
46
47    // Connect to Wi-Fi
48    WiFi.begin(ssid, password);
49    while (WiFi.status() != WL_CONNECTED)
50    {
51      delay(1000);
52      Serial.println("Connecting to WiFi..");
53    }
54
55    // Print ESP32 Local IP Address
56    Serial.println(WiFi.localIP());
```

Código 10: Código do servidor - parte 3.

Ainda dentro da função *setup()* a parte quatro (código 11) é, de fato, onde o servidor lida com as requisições e altera o estado do módulo relé de acordo com cada velocidade requisitada. As quatro primeiras rotas (index.html, style.css, index.js e favicon.ico) carregam as informações necessárias para o navegador carregar a página que o cliente irá acessar, essa denominada, rota raiz do servidor.

```
58    // Route for root / web page
59    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
60              { request->send(SPIFFS, "/index.html"); });
61
62    // Route to load style.css file
63    server.on("/css/style.css", HTTP_GET, [](AsyncWebServerRequest *request)
64              { request->send(SPIFFS, "/css/style.css", "text/css"); });
65
66    // Route to load index.js file
67    server.on("/js/index.js", HTTP_GET, [](AsyncWebServerRequest *request)
68              { request->send(SPIFFS, "/js/index.js", "text/javascript"); });
69
70    server.on("/favicon.ico", HTTP_GET, [](AsyncWebServerRequest *request)
71              { request->send(SPIFFS, "/favicon.ico"); });
72
73    // Route to set GPIO to LOW
74    server.on("/off", HTTP_PUT, [](AsyncWebServerRequest *request)
75              {
76                reset();
77                request->send(SPIFFS, "/index.html");
78              });
79
80    server.on("/one", HTTP_PUT, [](AsyncWebServerRequest *request)
81              {
82                reset();
83                digitalWrite(onePin, LOW);
84                request->send(SPIFFS, "/index.html");
85              });
86
87    server.on("/two", HTTP_PUT, [](AsyncWebServerRequest *request)
88              {
89                reset();
90                digitalWrite(twoPin, LOW);
91                request->send(SPIFFS, "/index.html");
92              });
93
94    server.on("/three", HTTP_PUT, [](AsyncWebServerRequest *request)
95              {
96                reset();
97                digitalWrite(threePin, LOW);
98                request->send(SPIFFS, "/index.html");
99              });
100
101   server.onNotFound(notFound);
102   // Start server
103   server.begin();
104 }
105
106 void loop() {}
```

Código 11: Código do servidor - parte 4.

A partir da linha 74, cria-se as quatro novas rotas para que o servidor esteja apto a receber e responder requisições, via protocolo HTTP, que o cliente enviará para alternar

o estado do módulo relé. Nota-se que o conceito inicial de ligar, trocar a velocidade ou desligar está implementado dentro de cada rota (linhas 83, 90, 97). Adicionou-se apenas o protocolo de comunicação entre os dispositivos. Na próxima seção, o código que será executado na parte cliente do modelo cliente-servidor complementa e garante sentido à todos os códigos apresentados. Na figura 25 observa-se a execução do servidor e o IP local no terminal de saída do VS Code.

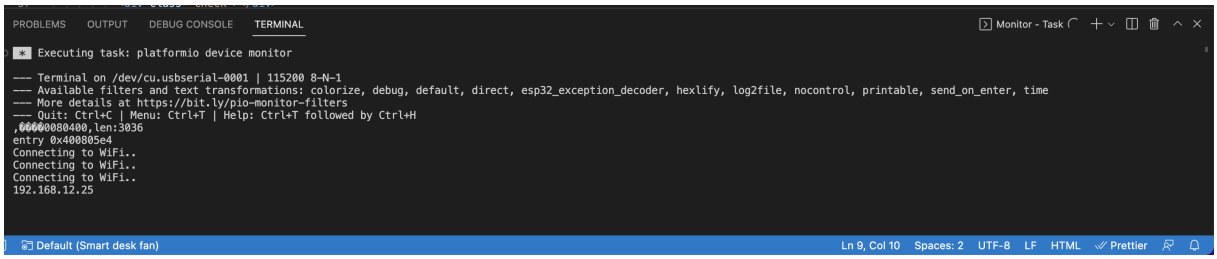

Figura 25 – Execução do servidor.

4.4.4 Página Web para Controle do Ventilador (Cliente)

Tendo em vista todo o código do servidor implementado, desenvolveu-se uma página web que envia ao servidor requisições para atender o desejo de controlar o ventilador à distância, levando em conta a seguinte ressalva: Neste estudo de caso, o funcionamento é limitado à rede local, ou seja, a ESP32 e o aparelho (cliente) que irá controlar o ventilador devem estar conectados à mesma rede Wi-Fi. A expansão para a Internet em geral é assunto para trabalhos futuros, pois a complexidade adicional se concentra em configuração de roteamento e Domain Name System (DNS) do ponto de acesso em questão.

No próximo passo desenvolveu-se três novos arquivos que o navegador necessita para renderizar uma página web, são eles: *data/index.html*, *data/style.css* e *data/index.js*. O servidor envia estes arquivos quando a rota raiz é requisitada, em outras palavras, ao acessar o IP do servidor uma página web é carregada e montada no navegador. Criou-se uma pasta data na estrutura do projeto contendo os arquivos citados, vide Figura 26.

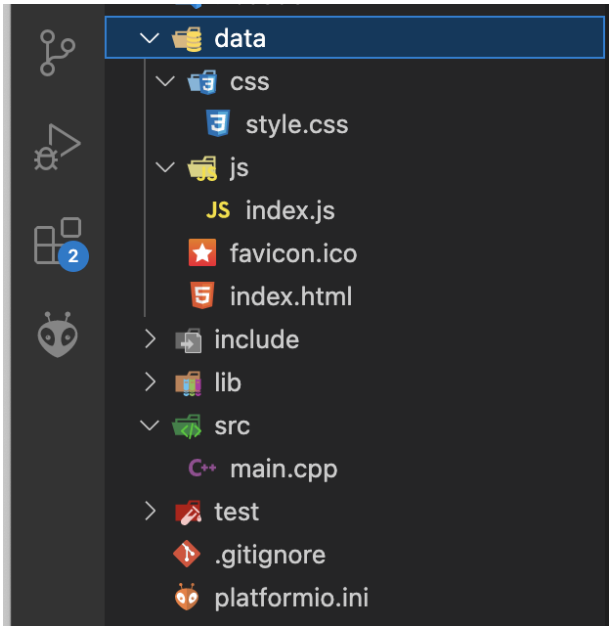

Figura 26 – Estrutura com o novo diretório data.

O *index.html* apresentado no código 12 é responsável pelos componentes, botões e itens na tela que o navegador irá exibir. E o *styles.css* tem o papel de diagramar e estilizar os itens citados (vide apêndice A).

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Smart Fan Desk</title>
    <link href="css/style.css" rel="stylesheet" type="text/css" />
  </head>

  <body>
    <div class="container">
      <h2>Smart desk fan</h2>

      <ul>
        <li>
          <input type="radio" id="off" name="selector" checked />
          <label for="off">Off</label>
          <div class="check"></div>
        </li>

        <li>
          <input type="radio" id="one" name="selector" />
          <label for="one">One</label>
          <div class="check"></div>
        </li>

        <li>
          <input type="radio" id="two" name="selector" />
          <label for="two">Two</label>
          <div class="check"></div>
        </li>

        <li>
          <input type="radio" id="three" name="selector" />
          <label for="three">Three</label>
          <div class="check"></div>
        </li>
      </ul>
    </div>
    <script src="js/index.js"></script>
  </body>
</html>
```

Código 12: Código do cliente - index.html.

Executando o build e o upload do projeto, acessou-se, via smartphone, o endereço IP do servidor, como pode ser visto na Figura 27. Esta página será a interface de comunicação com o servidor para controlar o ventilador em questão. Na sequência, a próxima subseção irá explicar como deu-se a comunicação entre a interface Web e o servidor na ESP32.

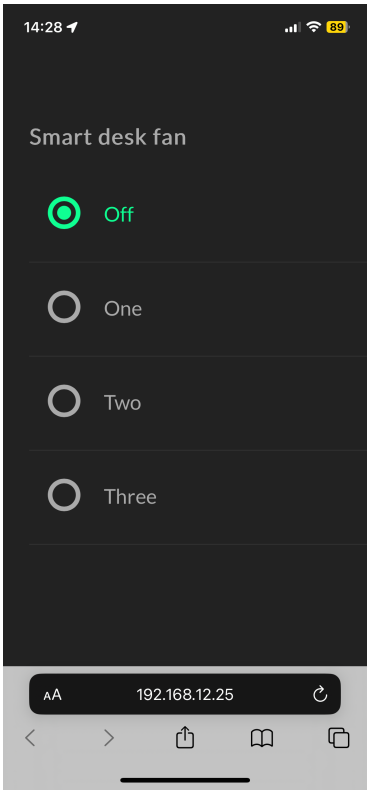

Figura 27 – Página Web.

4.4.5 Controle Via Requisição HTTP

O último dos três arquivos citados é o agente principal responsável por enviar as requisições ao servidor, alterando o estado do módulo relé. Utilizou-se a linguagem *Javascript* para montar requisições suportadas no protocolo Hypertext Transfer Protocol (HTTP).O HTTP é um protocolo cliente-servidor: as requisições são enviados por uma entidade, o agente-usuário, onde na maior parte do tempo, esse é um navegador da Web (MDN, 2022).

```
1  // Locate the 3 radio buttons
2  const off = document.querySelector("#off");
3  const one = document.querySelector("#one");
4  const two = document.querySelector("#two");
5  const three = document.querySelector("#three");
6
7  // Add event Listener to the 3 radio buttons
8  off.addEventListener("change", toggleStatus);
9  one.addEventListener("change", toggleStatus);
10 two.addEventListener("change", toggleStatus);
11 three.addEventListener("change", toggleStatus);
12
13 async function sendRequestToServer(url) {
14   try {
15     let res = await fetch(url, {
16       method: "PUT",
17     });
18     return res;
19   } catch (error) {
20     console.log(error);
21   }
22 }
23
24 async function toggleStatus(e) {
25   let sourceElementName = e.target.id;
26   let url = `${sourceElementName}`;
27   console.log("Sending to " + url);
28
29   let response = await sendRequestToServer(url);
30
31   console.log(response);
32 }
```

Código 13: Código do cliente - index.js.

As primeiras linhas (até a linha 5) do código 13 são as variáveis de cada botão, entre as linhas 8 e 11 cada botão é referenciado e, caso ocorra algum evento de clique, a função *toggleStatus()* é invocada. Esta função escrita entre as linhas 24 e 32 tem o papel de extrair o respectivo botão clicado e enviar à função *sendRequestToServer()* (linhas 13 a 22) que, por sua vez, realiza a chamada HTTP ao servidor utilizando o método PUT do protocolo. As linhas que contém *console.log()* são auxiliadores de desenvolvimento, como mostra a figura 28, as ferramentas de desenvolvimento do navegar web Google Chrome.

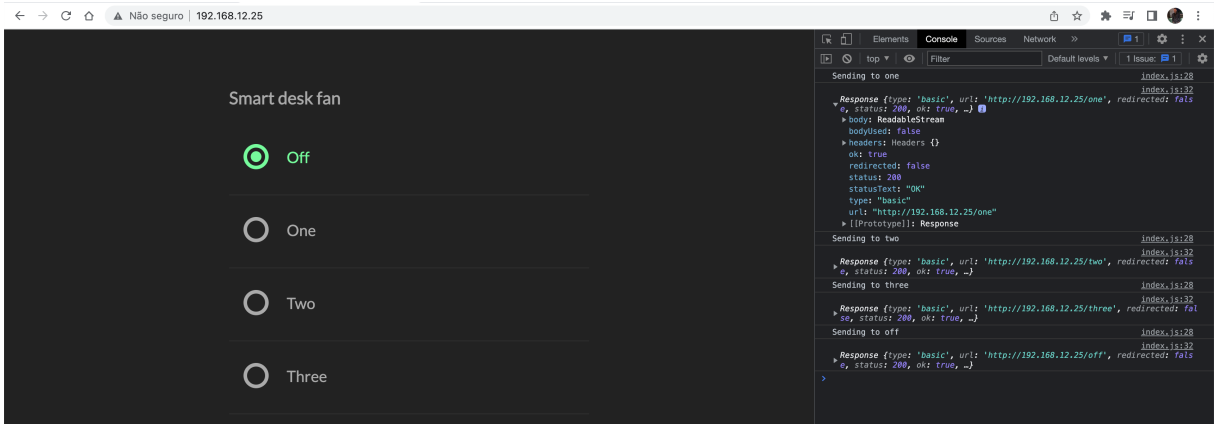

Figura 28 – Ferramenta de desenvolvimento do navegador.

Finalmente toda integração foi feita e agora o ventilador é totalmente controlado através dessa interface e de vários dispositivos diferentes na mesma rede.

4.5 Análise do Estudo de Caso

Até a sessão anterior foi descrito os aspectos técnicos para o desenvolvimento dessa solução *IoT*. No entanto, vale analisar o custo da mão-de-obra, o esforço necessário e a infraestrutura para tornar esse dispositivo *IoT ready*. Em termos de custos a opção escolhida (ESP32) e o módulo relé foram as mais baratas tendo em vista todos os recursos nativos (Wi-Fi, Bluetooth, PWM, etc) prontos.

Figura 29 – Custo do estudo de caso

Comparativamente, se caso optado por uma solução, utilizando Arduino com o módulo Ethernet (uma vez que ele não possuí comunicação sem fio por si só) a pesquisa calculou valores médios que giram em cerca de 199,27 reais (em 2023) e, como pode ser

visto na Figura 29 os valores com o ESP32 foram relativamente abaixo da alternativa, com Arduino.

Uma outra visão aplicada à análise do estudo de caso é a de **esforço** e **know-how (conhecimento)** os quais juntos concatenam e fazem parte da complexidade de saída do ponto A (dispositivo não *IoT ready*) para o ponto B (dispositivo *IoT ready*).

O esforço é subjetivamente calculado pois para quem já realizou algumas vezes o procedimento pode tornar-se simples à medida que os passos são repetidos. Todavia, a situação que representa o primeiro contato com o passo-a-passo apresentado no capítulo pode ser trabalhoso, sinuoso, e altamente complexo. Todos esses pontos implicam em horas de estudo e dedicação para chegar ao objetivo final. O que nos leva à questão do quanto se estaria disposto a empregar o próprio tempo para aprender e realizar o método discutido.

Para com relação ao conhecimento, muitos conceitos foram apresentados durante a técnica do cap. 4 e a maioria deles não são ensinados corriqueiramente, a não ser em cursos mais específicos das áreas de tecnologia e computação. Ainda assim, nessas áreas, é preciso buscar fontes de conhecimento que complementam o quadro de habilidades e disciplinas aprendidas durante os cursos. Para realizar o procedimento é necessário conhecer alguns conceitos básicos e aqui vale citar alguns deles: sistemas digitais, programação e redes de computadores, tais conhecimentos são só parcialmente ofertados num curso de bacharelado em ciência da computação. No entanto, assistindo vídeos no YouTube, vendo tutoriais na internet, consultando livros textos, um bom autodidata pode adquirir o conhecimento necessário. Portanto, novamente a discussão nos leva a questão do preço que se paga para tornar um dispositivo *IoT ready* versus o preço de um dispositivo pronto para ser utilizado no mercado.

5 Conclusão

Neste trabalho uma pesquisa extensiva foi executada acerca dos dispositivos *IoT* disponíveis para comercialização no mercado no ano corrente. Os resultados foram condensados sob a forma de uma tabela comparativa demonstrando que produtos *IoT ready* possuem uma discrepância de valores desproporcional em comparação à um dispositivo similar que não seja *IoT ready*. Possivelmente a demora na adoção das tecnologias *IoT* advém destes fatos supracitados.

Para facilitar o desenvolvimento de soluções de *IoT* caseiras este trabalho também apresentou uma revisão sistemática dos principais conceitos que permeiam a área. Uma estrutura geral de uma aplicação *IoT* foi dividida em módulos e abordada através de alguns conceitos, tais como: controle analógico-digital, conectividade à Internet, arquitetura Cliente-Servidor, requisições HTTP, e demais.

As principais soluções de hardware para dar suporte ao processo de adaptação de dispositivos disponíveis atualmente foram apresentadas. Nomeadamente soluções baseadas em Arduino e ESP32. Além dos componentes e conceitos utilizados, tais como: os diferentes microcontroladores, linguagens de programação, *frameworks* disponíveis e as opções de conectividade do dispositivo com a Internet.

O modelo geral proposto neste trabalho foi posto em prática em um estudo de caso, onde automatizou-se um ventilador de mesa padrão. Para isto, utilizou-se uma solução *open hardware* da família ESP32 desenvolvido pela Espressif Systems, que conta com conexão Wi-Fi embutida, mitigando o pilar da conexão entre o dispositivo e a rede. Além disso, um circuito de relés foi utilizado para realizar o controle analógico de cada possível velocidade do ventilador. Para permitir o controle via *smartphone* o software conta com um um servidor web local, o qual recebe e responde requisições alterando a velocidade do ventilador de forma remota.

Durante a revisão do estado da arte, as soluções de *IoT* caseiras encontradas e exemplificadas utilizavam uma vasta lista de tecnologias. No entanto notou-se que dentre as opções disponíveis existiam múltiplos prós e contras. Portanto neste trabalho apresentou-se um guia com os passos para automatizar dispositivos utilizando soluções *open hardware*, *frameworks* de software, opções de conectividade, desenvolvimento de software cliente. Tal guia visa facilitar o retrabalho de quem deseja automatizar um dispositivo qualquer, pois os elementos passíveis de automatização foram discriminados ao longo deste conteúdo.

Uma das perguntas em aberto paralela ao tema é o motivo pelo qual *IoT* ainda não está tão difundida quanto deveria. Um suposto caminho para se chegar próximo a

uma resposta a esta pergunta seria possivelmente analisar 3 fatores que influenciam nesta inércia relacionada a adoção de soluções *IoT*.

A conectividade é uma delas. A dificuldade de conexão de um dispositivo a Internet é grande como demonstrado neste trabalho, contudo a rede tende a tornar-se cada vez mais móvel e a chegada de tecnologias como o 5G/6G irá ultrapassar essa barreira, permitindo que empresas desenvolvam produtos com conexão móvel em larga escala, facilitando e diluindo o valor de produtos *IoT ready*.

O segundo fator é a forma de interação com o usuário final, o que é conhecido como Interação Humano-Computador. Nos tempos atuais, ainda utilizam-se interfaces como o teclado e o mouse, telas *touchscreen* com botões que limitam a vazão de entrada de dados, pois eles convertem a forma com que nos expressamos em um devido formato que a máquina e o sistema entendam. Assim, cada vez mais dispositivos com Processamento de Linguagem Natural estão presentes na nossa realidade. Siri, Google Assistente, Google Bard, e Alexa são alguns exemplos que possibilitam uma forma mais natural e intuitiva de interação através a nossa própria fala.

O terceiro fator é a falta de padronização, assim como em toda tecnologia emergente há uma certa dificuldade em padronizar a forma de construção, o ambiente de programação e até mesmo a linguagem que estes dispositivos utilizam para se comunicar, o que culmina em softwares proprietários, falta de intercomunicação entre diferentes dispositivos de diferentes marcas e protocolos distintos que realizam a mesma função.

5.1 Perspectivas futuras

Uma análise do quadro geral denota, quase que inequivocamente, que *IoT* é o futuro. Em breve, todos os nossos dispositivos estarão conectados, não só dentro de casa, mas também no trabalho, na cidade, no carro, ubiquamente.

Com a implementação em curso do 5G no ano corrente e a discussão do desenvolvimento da tecnologia 6G em andamento, a internet muito rápida e barata vai tornar-se largamente disponível. O que derrubará a barreira da conectividade e criará novas oportunidades para impulsionar *IoT*.

Outro personagem emergente nos últimos tempos são as inteligências artificiais capazes de interpretar e conversar no mesmo nível que o ser humano, *e.g.* ChatGPT, isso propiciará cada vez mais interação com dispositivos inteligentes.

Como vimos durante toda a discussão acima, fica claro que não é uma dificuldade tecnológica intransponível para as empresas e o mercado darem uma maior atenção ao universo *IoT*. E conclui-se que o motivo de não ter tanto espaço é que, além dos fatores citados acima, o mercado ainda não enxerga *IoT*, principalmente na esfera domiciliar,

como uma tendência principal e dominante. Enquanto não houver holofotes nessa área, ela ainda será vista de forma embrionária e hobista, ou desfrutada para quem, curiosamente, gosta se aventurar neste mundo.

Referências

CAMPBELL, S. 2023. Disponível em: <https://www.circuitbasics.com/how-to-set-up-the-dht11-humidity-sensor-on-an-arduino/>. Acesso em: 17 mai. 2023. Citado na página 19.

COPES, F. 2023. Disponível em: <https://flaviocopes.com/arduino-programming-language/>. Acesso em: 02 fev. 2023. Citado na página 15.

De Silva, L. C.; MORIKAWA, C.; PETRA, I. M. State of the art of smart homes. *Engineering Applications of Artificial Intelligence*, v. 25, n. 7, p. 1313–1321, 2012. ISSN 0952-1976. Advanced issues in Artificial Intelligence and Pattern Recognition for Intelligent Surveillance System in Smart Home Environment. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095219761200098X>. Citado na página 28.

ENGINEERSGARAGE. 2023. Disponível em: <https://www.engineersgarage.com/biometric-sensor-with-arduino/>. Acesso em: 17 mai. 2023. Citado na página 19.

GRATISPNG. 2023. Disponível em: <https://www.gratispng.com/png-kkffbq/>. Acesso em: 30 jan. 2023. Citado 2 vezes nas páginas 6 e 14.

HARPER, R. *Inside the Smart Home*. Bristol, UK: Springer, 2003. Citado na página 10.

MAKERHERO. 2023. Disponível em: <https://www.makerhero.com/blog/o-que-e-arduino/>. Acesso em: 31 jan. 2023. Citado 2 vezes nas páginas 6 e 18.

MAKERHERO. 2023. Disponível em: <https://www.makerhero.com/produto/ethernet-shield-w5100-para-arduino/>. Acesso em: 29 jan. 2023. Citado 2 vezes nas páginas 6 e 19.

MDN. 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview>. Acesso em: 04 dez. 2022. Citado na página 53.

OMRON. 2023. Disponível em: <https://components.omron.com/sg-en/products/basic-knowledge/relays/basics>. Acesso em: 18 jan. 2023. Citado 2 vezes nas páginas 6 e 24.

PRESS, G. 2014. Disponível em: <https://www.forbes.com/sites/gilpress/2014/08/22/internet-of-things-by-the-numbers-market-estimates-and-forecasts/?sh=3f194df7b919>. Acesso em: 15 fev. 2021. Citado na página 10.

RANDOMNERDTUTORIALS. 2023. Disponível em: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>. Acesso em: 28 jan. 2023. Citado 3 vezes nas páginas 6, 20 e 21.

RANDOMNERDTUTORIALS. 2023. Disponível em: <https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/>. Acesso em: 01 mai. 2023. Citado 2 vezes nas páginas 6 e 26.

SARACINO, E. *SMART HOME: STATE OF THE ART AND KEY FACTORS FOR VALUE CREATION*. Dissertação (Mestrado) — POLITECNICO DI MILANO, Milão, 2020. Citado na página 29.

SENSORKIT. 2023. Disponível em: <https://sensorkit.arduino.cc/sensorkit/module/lessons/lesson/05-the-light-sensor>. Acesso em: 17 mai. 2023. Citado na página 19.

SOUZA, F. 2023. Disponível em: <https://embarcados.com.br/arduino-entradas-analogicas/>. Acesso em: 23 abr. 2023. Citado na página 23.

SUNFOUNDER. 2023. Disponível em: <http://wiki.sunfounder.cc/index.php?title=4_Channel_5V_Relay_Module>. Acesso em: 01 fev. 2023. Citado 2 vezes nas páginas 6 e 25.