UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

JOÃO EMANUEL DOS SANTOS COSTA

DESENVOLVIMENTO DE UM MÓDULO WIFI MICROCONTROLADO PARA APLICAÇÕES EM PROJETOS DE INTERNET DAS COISAS

CAMPO MOURÃO

2022

JOÃO EMANUEL DOS SANTOS COSTA

DESENVOLVIMENTO DE UM MÓDULO WIFI MICROCONTROLADO PARA APLICAÇÕES EM PROJETOS DE INTERNET DAS COISAS

.

Development of a microcontrolled WiFi module for applications in internet of things projects

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Marcelo Nanni.
Coorientador: Prof. Dr. Lucio Geronimo Valentin.

CAMPO MOURÃO

2022

JOÃO EMANUEL DOS SANTOS COSTA

DESENVOLVIMENTO DE UM MÓDULO WIFI MICROCONTROLADO PARA APLICAÇÕES EM PROJETOS DE INTERNET DAS COISAS

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

Data de aprovação: 06/junho/2022

Lucas Ricken Garcia
Doutor
Universidade Tecnológica Federal do Paraná

Roberto Wilhelm Krauss Martinez
Doutor
Universidade Tecnológica Federal do Paraná

Marcelo Nanni
Doutor
Universidade Tecnológica Federal do Paraná

CAMPO MOURÃO

2022

Dedico este trabalho aos meus pais, por toda a confiança
que sempre me concederam e pelo exemplo de força,
determinação e integridade que representam.
Dedico também a meus amigos, por me ajudarem durante
todo o percurso.

AGRADECIMENTOS

Acredito que seja impossível agradecer a todas as pessoas que me ajudaram nessa caminhada, pois considero que tudo na minha vida tenha me trazido até esse momento. Deixo aqui apenas uma pequena parcela de todos os obrigados que tenho a dar.

Agradeço, primeiramente, aos meus pais, pois, mesmo a milhares de quilômetros de distância e a vários anos de separação, nunca duvidaram de mim ou do meu propósito, me apoiando a cada decisão.

Agradeço ao meu orientador Prof. Dr. Marcelo Nanni, pelo tempo dedicado a mim e ao meu trabalho.

Agradeço ao meu coorientador Prof. Dr. Lucio Geronimo Valentin, por me ensinar tudo que eu sei sobre programação e pelo tempo em mim investido.

Por fim, agradeço a todos os amigos e demais familiares, por terem me guiado e ajudado durante minha caminhada.

RESUMO

Em um mundo cada vez mais conectado e com a demanda crescente por dados, se aumenta cada vez mais a demanda para que projetos em Engenharia Eletrônica possuam algum mecanismos internos que os possibilite realizar o envio de dados ao exterior. Pensando nisso, o trabalho teve como objetivo o desenvolvimento de um dispositivo eletrônico capaz de realizar o transporte de dados entre protocolos seriais e de internet, por meio da rede Wi-Fi. Dessa forma, o produto criado poderá facilitar a criação de uma série de outros projetos em eletrônica; em particular, projetos que envolvam IoT (do inglês *Internet of Things*). Com esse intuito, foram realizados estudos teóricos para determinar quais os melhores protocolos a serem utilizados, chegando-se a três: MQTT (do inglês *Message Queuing Telemetry Transport*), HTTP (do inglês *Hypertext Transfer Protocol*) e UART (do inglês *Universal Asynchronous Receiver/Transmitter*). Desta forma, foi desenvolvido um *software* em linguagem C/C++, baseado no ambiente de desenvolvimento integrado (IDE) Arduino e embarcado no microcontrolador ESP8266. Também, foi desenvolvida uma página *web*, utilizando HTML (do inglês *HyperText Markup Language*), CSS (do inglês *Cascading Style Sheets*) e JS (do inglês *Java Script*), para que o usuário possa executar a configuração do equipamento. Além disso, foi implantado o suporte a OTA (do inglês Over-the-air), sendo possível atualizar o *firmware* do dispositivo à distância. No escopo do *hardware*, foi projetada e fabricada uma PCB (do inglês *Printed Circuit Board*), para que ocorra a conexão física entre os dois circuitos, com suporte tanto a nível de tensão de 3,3 V, quanto para 5V. Finalmente, com o produto em mãos, o usuário poderá melhorar a comunicação do seu circuito eletrônico com outros meios, sendo eles do tipo *wired* ou *wireless.*

Palavras-chave: IoT; ESP8266; Wi-Fi; Servidores; OTA.

ABSTRACT

In an increasingly connected world and with the growing demand for data, the demand for Electronic Engineering projects to have some internal mechanisms that allow them to send data abroad is increasing. With that in mind, the objective of this work was to develop an electronic device capable of carrying data between serial and internet protocols, through the Wi-Fi network. In this way, the product created will be able to facilitate the creation of a series of other projects in electronics; in particular, projects involving IoT (Internet of Things). To this end, theoretical studies were carried out to determine the best protocols to be used, reaching three: MQTT (Message Queuing Telemetry Transport), HTTP (Hypertext Transfer Protocol) and UART (Universal Asynchronous Receiver/Transmitter). In this way, a software was developed in C/C++ language, based on the Arduino integrated development environment (IDE) and embedded in the ESP8266 microcontroller. Also, a web page was developed, using HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and JS (Java Script), so that the user can configure the equipment. In addition, OTA (Over-the-air) support was implemented, making it possible to update the device's firmware remotely. In terms of hardware, a PCB (Printed Circuit Board) was designed and manufactured, so that the physical connection between the two circuits occurs, with support for both 3.3 V and 5V voltage levels. Finally, with the product in hand, the user will be able to improve the communication of his electronic circuit with other means, whether wired or wireless.

Keywords: IOT; ESP8266; Wi-Fi; Servers; OTA.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AP	*Access Point*
BSS	*Basic Service Set*
BW	*Band With*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
FIFO	*First-In, First-Out*
GPIO	*General Purpose Input/Output*
GPS	*Global Positioning System*
Hi-Fi	*High Fidelity*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
ID	Identidade
IDE	*Integrated Development Environment*
IEEE	Institute of Electrical and Electronics Engineers
IoT	*Internet of Things*
IP	*Internet Protocol*
I^2C	*Inter-Integrated Circuit*
JS	*Java Script*
M2M	*Machine-to-Machine*
MAC	*Media Access Control*
MVP	*Minimum Viable Product*
MQTT	*Message Queuing Telemetry Transport*
NodeMCU	*Node MicroController Unit*
OTA	*Over-the-air*
PCB	*Printed Circuit Board*
PIC	*Peripheral Interface Controller*
RX	Receptor
SoC	*System-on-a-Chip*
SSID	*Service Set Identifier*
STA	*Station*
TCC	Trabalho de Conclusão de Curso
TCP	*Transmission Control Protocol*
TX	Transmissor
UART	*Universal Asynchronous Receiver/Transmitter*
VS	Visual Studio
WECA	Wireless Ethernet Compatibility Alliance
WEP	*Wired Equivalent Privacy*
Wi-Fi	*Wireless Fidelity*
WPS	Wi-Fi *Protected Setup*

SUMÁRIO

1 INTRODUÇÃO

Após o término da Segunda Guerra Mundial, em meados de 1950, o mundo estava perto de vivenciar uma mudança total no padrão de consumo estabelecido, fato que é refletido até os dias atuais. A indústria se encontrava a todo vapor e, para que toda essa energia pudesse ter escoamento, a produção industrial passou de armamentos e equipamentos de guerra para bens de consumo, lazer e beleza, apostando no consumo das famílias, na chamada economia do pós-guerra (BERCHELE et al., 2018; CARMO; COLOMBO; FAVOTO, 2008).

Nesse contexto, com o aumento no tamanho das famílias e com as donas de casa indo trabalhar fora, surgiu o mercado de eletrodomésticos – equipamentos que tinham como objetivo tornar as tarefas de casa mais rápidas e práticas. Dessa forma, surgiu uma necessidade nas pessoas de equipamentos cada vez melhores, a fim de suprir suas necessidades de conforto, segurança e lazer (CARMO; COLOMBO; FAVOTO, 2008; BADESCU; LAZAROIU; BARELLI, 2018).

A fim de atender essas e outras expectativas, por volta da década de 80 foram introduzidos os conceitos de Automação Residencial (Domótica) – uma aplicação das técnicas de Automação Industrial para o contexto das residências. Com a Domótica, passou-se a utilizar sensores e atuadores, de diversos tipos, no contexto domiciliar, de modo os equipamentos verificarem seus próprios funcionamentos e introduzir possíveis correções (BADESCU; LAZAROIU; BARELLI, 2018; SOUZA, 2016).

Em paralelo a esse processo, acontecia também a chamada revolução digital, ao passo que ocorreram várias evoluções, tanto na computação, com a evolução dos microprocessadores; quanto nas comunicações, com a criação e evolução dos protocolos de internet. Graças à disseminação de equipamentos eletrônicos e o aumento da disponibilidade de internet para as pessoas, o mundo contemporâneo é totalmente conectado e automatizado (BADESCU; LAZAROIU; BARELLI, 2018; OLIVEIRA, S., 2017).

Nesse contexto, e com o barateamento de microcontroladores (MCUs, do inglês *microcontroller unit*) , como o ESP8266, que além de possuir um bom poder de processamento, apresentam conexão com redes de internet; volta à tona o conceito de Internet das Coisas(IoT, do inglês *Internet of Things*), onde passa a existir a possibilidade de que objetos do cotidiano, como relógios, lâmpadas e tomadas, possam ter conexão direta com a internet; conceito que há muito é discutido, mas que nunca havia sido possível de pôr em prática (OLIVEIRA, S., 2017).

A partir da união dos conceitos de Domótica, IoT e *Data Science,* é possível executar ações como: a casa passe a identificar quando não houver mais pessoas em um cômodo e apagar

as luzes deste local; mudar a intensidade luminosa das lâmpadas de acordo com a iluminação natural; identificar se todas as portas e janelas estão trancadas e caso algum estranho tentar invadir o local, emita um sinal de alerta; é possível até que a casa receba informações do GPS do proprietário e traçando padrões, a ponto de prever o tempo que ele irá levar para chegar, podendo assim acionar sistemas como os de aquecimento, iluminação e limpeza (BADESCU; LAZAROIU; BARELLI, 2018; OLIVEIRA, S., 2017).

Entretanto, a utilização fundamental dada aos princípios de IoT, está na Automação Industrial, na chamada Indústria 4.0. O IoT na indústria torna possível que todas as máquinas de uma planta industrial, troquem informações a partir de uma ou várias redes de comunicação. Desse modo, torna-se possível que as plantas industriais sejam cada vez mais autônomas e que a qualidade dos processos melhore significativamente, visto que todas as máquinas passam a operar em um nível altíssimo de precisão. Isso, em partes, se deve à quantidade de sensores empregados em cada máquina, gerando um montante gigantesco de dados, que acabam por integrar sistemas de controle, baseados em Inteligência Artificial e *Machine Learning* (BERCHELE et al., 2018).

1.1 Objetivos

Em seguida, serão descritos os objetivos gerais, e elencados os objetivos específicos para o presente trabalho.

1.1.1 Objetivo geral

Desenvolver um dispositivo que possibilite a conexão de circuitos eletrônicos digitais entre si e com o meio externo, utilizando de conexão Wi-Fi doméstica em ambientes residenciais.

1.1.2 Objetivos específicos

- Enviar e receber dados via UART;
- Realizar a conexão a redes Wi-Fi;
- Realizar envio e recebimento de dados via protocolo MQTT;
- Projetar uma página Web para configuração do dispositivo;
- Projetar o modelo esquemático, roteamento e modelo tridimensional do circuito;
- Implantar atualização de software OTA;
- Realizar envio e recebimento de dados via protocolo HTTP;

1.2 Justificativas

O termo IoT nunca esteve tão em voga, e a procura por equipamentos desse tipo nunca esteve tão grande como nos dias atuais. Como consequência disso, projetos de dispositivos IoT são fonte constante de estudos por parte de projetistas e entusiastas da área. Muitas vezes o método utilizado para criar novos dispositivos IoT parte da adaptação de alguns circuitos eletrônicos já existentes, como circuitos atuadores, medidores e circuitos de sensoriamento. A partir da união desses e outros circuitos eletrônicos com conexões à internet (direta ou indiretamente), podem surgir novos dispositivos IoT (KOLBAN, 2016).

Entretanto, adicionar uma conexão à internet em um projeto não é tão simples quanto utilizar um relé, uma placa Arduino e algumas poucas linhas de código, para produzir um interruptor digital. Para incorporar a internet em um projeto é necessário que se tenha um grau de conhecimento mais avançado sobre programação em microcontroladores e protocolos de comunicação, tal qual será apresentado mais adiante no texto.

Dito isso, qualquer novo dispositivo que torne esse processo menos trabalhoso, e seja compacto, robusto e, acima de tudo, que seja confiável, é algo que pode atrair bastante atenção; este é um dos motivos para a produção do presente trabalho. Um dispositivo com a descrição acima não somente facilitaria bastante o trabalho de projetistas de dispositivos IoT, quanto poderia tornar possível que pessoas de fora da área pudessem criar seus próprios projetos, podendo inclusive facilitar o aprendizado de tais tecnologias.

Todavia, a ideia por trás do desenvolvimento desse projeto surgiu de uma necessidade prática, do mercado; mais especificamente uma necessidade da empresa Smart Collect – empresa incubada no Hotel Tecnológico da UTFPR de Campo Mourão. A empresa Smart Collect trabalha com medições de dados de consumo elétrico, que são realizadas com a utilização de um dispositivo eletrônico digital que desempenha as funções de wattímetro, frequencímetro, amperímetro e voltímetro. Tal equipamento executa a coleta e o tratamento dos dados elétricos, contudo, ele não possui conexão de internet. Desse modo, foi decidido por construir um módulo para que esse e outros dispositivos semelhantes possam se conectar a redes Wi-Fi (do inglês *Wireless Fidelity*, em português Fidelidade Sem Fio).

2 FUNDAMENTAÇÃO TEÓRICA

Produzir um dispositivo eletrônico que consiga fazer o intermédio entre várias vias de comunicação não é nada simples, visto que isso implica na necessidade de se ter conhecimento sobre vários tipos de protocolos de comunicação, além de ter um bom conhecimento sobre microcontroladores e um domínio do seu funcionamento. Alguns desses conhecimentos serão apresentados no texto adiante.

2.1 Microcontroladores

Em um computador geralmente são executadas três partes primordiais: armazenamento, processador (CPU, do inglês *Central Processing Unit*), e a entrada e saída de dados; todas essas funções são desempenhadas por seus componentes internos que, por sua vez, são interligados por circuitos de comunicação. Um microcontrolador também possui CPU, armazenamento e periféricos de entrada e saída de dados, com a diferença que para o MCU tudo isso está disposto em um só chip (BORBA, 2017; PENIDO; TRINDADE, 2013).

Para que seja possível se ter todos esses componentes em um só lugar, os fabricantes de circuitos integrados optam por diminuir consideravelmente todos os seus atributos. Consequentemente, esses chips também apresentam características como preços e consumo elétrico bastante inferiores ao de um computador; excluídos circuitos integrados como os da família Raspberry, que funcionam tal qual um computador e que, inclusive, rodam seu próprio sistema operacional (HALFACREE, 2020). Microcontroladores, em geral, vêm para suprir necessidades de prototipação e desenvolvimento de projetos mais simples, que exigem menos do hardware (KOLBAN, 2016).

Os microcontroladores tiveram seu surgimento no ano de 1977, com o microcontrolador 8048, produzido pela empresa Intel. Este microcontrolador contava com uma versão modificada da arquitetura Harvard e já continha uma memória interna. Baseado no 8048, surgiu o microcontrolador 8051, bastante conhecido e utilizado até hoje, mesmo com a idade e defasagem, em comparação aos chips atuais (OKI; MANTOVANI, 2013; PENIDO; TRINDADE, 2013).

Após a criação da série 8051, surgiu uma gama de outros circuitos alternativos, dos quais, vale ressaltar os famosos chips da família PIC, nome esse para Controlador de Interface Periférica (do inglês *Peripheral Interface Controller*); produzidos pela empresa Microchip, que

também são bastante utilizados até hoje, pela praticidade de serem programados em linguagem C e facilidade de prototipação (OKI; MANTOVANI, 2013).

Também existem os circuitos integrados utilizados pelas placas de desenvolvimento Arduino. Eles se tornaram bastante famosos entre iniciantes da área, pelo fato de contarem com seu próprio Ambiente Integrado de Desenvolvimento (IDE, do inglês *Integrated Development Environment*), que facilitou bastante a programação e gravação do circuito integrado (OKI; MANTOVANI, 2013).

Nesse sentido, os microcontroladores passaram por várias alterações com o passar dos anos, se tornando cada vez mais robustos e de fácil acesso. Entretanto, esses circuitos integrados, em sua maioria, apresentavam um mesmo gargalo quando se trata de comunicação: eles não apresentavam de forma nativa a capacidade de se conectar com a internet e, muitas vezes, acabavam por depender de periféricos externos para fazer essa comunicação (KOLBAN, 2016).

No ano de 2014, foram lançados os microcontroladores da família ESP8266EX, criados pela empresa Espressif Systems, com a proposta inicial de suprir essa demanda, contando tanto com suporte a redes Wi-Fi quanto com o protocolo TCP/IP (do inglês *Transmission Control Protocol/ Internet Protocol*, em português Protocolo de Controle de Transmissão/ Protocolo de Internet). Isso é possível, a partir da aplicação dos conceitos de sistema em um chip (SoC, do inglês *system-on-a-chip*), que adiciona alguns periféricos extras a circuitos integrados, como a pilha TCP/IP, no caso do ESP8266. (KOLBAN, 2016; STREIF, 2019).

Os chips da família ESP8266EX podem, inclusive, criar sua própria rede Wi-Fi. Esses microcontroladores contam com a mesma facilidade dos utilizados pelas placas de desenvolvimento Arduino, tendo suporte às linguagens de programação mais modernas e podendo, inclusive, serem programados através da mesma interface de desenvolvimento (KOLBAN, 2016; OLIVEIRA, R., 2017).

No ano de 2016, a Espressif Systems realizou o lançamento do chip ESP32. Esse chip, em comparação ao seu predecessor, apresentando um aumento bastante significativo na quantidade de memória interna, na resolução dos terminais de entrada e saída analógicos, no processamento, e na quantidade de portas de entrada e saída de propósito geral (GPIO, do inglês *General Purpose Input/Output*) (RIOS, 2020).

Mesmo com todos esses pontos a favor do microcontrolador ESP32, foi optado pela utilização do ESP8266 pelo fato de que ele é um microcontrolador que possui uma especificação interna (processamento, memória, etc.) satisfatória e apresenta nativamente. Ele

também está a mais tempo no mercado, possuindo um acervo de dados (bibliotecas, fóruns, tutoriais, etc.) maior; possui uma maior variedade e um preço inferior, quando comparado ao ESP32 (RIOS, 2020).

2.1.1 ESP8266

Os SoCs ESP8266EX compõem uma família de microcontroladores, desenvolvidos pela multinacional chinesa Espressif Systems. Eles foram criados com a finalidade de prover chip soluções para IoT, por conta da facilidade de conexão deles com a internet. Esses microcontroladores, hoje em dia, estão vastamente disseminados pelo mercado, a ponto de terem várias plataformas como unidades microcontroladoras em nó (NodeMCU, do inglês *Node MicroController Unit*), que são soluções *open source* e open hardware para prototipação de projetos eletrônicos. O Quadro 01 apresenta as principais características desses chips (ESPRESSIF SYSTEMS, 2021).

Quadro 1 - Especificações ESP8266EX

Definição	Atributo
Tensão de operação	2,5V - 3,6V
Corrente de operação	80mA
Temperatura de operação	-40 °C - 125 °C
Memória flash expansível	16MB máximo (512k normal)
Processador	Tensilica L106 32 bit
Velocidade do processador	80-160MHz
RAM	32K + 80K
GPIOs	17 (multiplexadas com outras funções)
Portas analógicas	1 entrada com resolução de 1024 (10bit)
Suporte IEEE 802.11	b,g,n,d,e,k,r
Máxima de conexões TCP	5

Fonte: Adaptado de Espressif Systems (2020b)

Por mais que os chips da família ESP8266EX possam parecer bastante distintos uns dos outros, à primeira vista, todos eles apresentam a mesma CPU: o chip Tensilica L106, de 32 bits. Esses microcontroladores variam apenas em características como modelo (ou a ausência) de antena WiFi; quantidade de GPIOs e características construtivas como o modelo (ou a ausência) de um *shield* metálico ao redor do chip, e o modulo ao qual o chip possa estar acoplado (ESPRESSIF SYSTEMS, 2020a; KOLBAN, 2016; MODULES, 2021).

Como os chips internos de cada ESP8266 são quase o mesmo, geralmente é possível fazer o mesmo código para rodar em todos os modelos – levando em consideração quais portas estão disponíveis, a presença ou ausência de antena Wi-Fi e a memória interna. Essa programação pode ser feita tanto em C/C++, utilizando IDEs como a do Aduino, quanto em linguagem Lua, com gravador dos dispositivos ESP (ESPLoader), utilizado pelas NodeMCUs.

Também existem aplicações que utilizam programação em outras linguagens, como Java Script e Phyton (KODALI, 2016; KOLBAN, 2016).

Por mais que tenha sido lançado apenas em meados de 2014, o ESP8266 é um produto bastante recente, em questão de tecnologia. Isso significa que essa é uma tecnologia ainda não totalmente dominada, tendo presença frequente dentro de fóruns da área. Essa quantidade de pessoas interessadas se deve pelo mesmo se tratar de um microcontrolador bastante potente, versátil e, principalmente, muito barato – tendo valores, inclusive, abaixo de 20 reais, em sites nacionais (KOLBAN, 2016; OLIVEIRA, R., 2017).

2.2 Entradas e saídas

Quando se trata de entrada e saída de dados, de forma "cabeada", basicamente se refere a dois tipos de sinais: sinais analógicos e digitais. Sinais analógicos, como o nome indica, são representações físicas diretas da forma como aquela informação foi obtida da natureza. Entretanto, como é sabido, a grande maioria dos equipamentos eletrônicos modernos é digital, e para que eles possam analisar dados analógicos, os mesmos devem ser digitalizados, sendo então o dado da natureza codificado em agrupamentos de zeros e uns (*bytes*), de forma que um processador possa compreender tais informações (LATHI, 2010).

Desse modo, para evitar que os dados sejam constantemente digitalizados, o que na prática significa gasto, eles muitas vezes já são transmitidos de forma digital; principalmente entre dois dispositivos que já trabalhem dessa forma. Desse fato surgem uma séria de padrões de comunicação digital; a exemplo o padrão universal de transmissão e recepção assíncrona (UART, do inglês *Universal Asynchronous Receiver/Transmitter*). Ele é utilizado para a comunicação entre dispositivos eletrônicos digitais, por exemplo, a comunicação entre dois processadores ou microcontroladores (OLIVEIRA, S., 2017).

2.2.2 UART

O padrão UART é um padrão de troca de dados de forma assíncrona, como consta em seu nome. Isso quer dizer que para ocorrer a comunicação não é necessário que o dispositivo transmissor e o receptor estejam compartilhando do mesmo sinal de *clock*; apenas é necessário que ambos estejam operando com frequências equivalentes (OLIVEIRA, S., 2017; UART, 2014).

Dessa forma, toda a comunicação é executada com a utilização de dois canais: um canal receptor (RX) (receptor) e um transmissor (TX). O transmissor executa a transmissão dos

bits (dado digital) de forma serial, por um único canal compartilhado, e o receptor identifica os dados que estão chegando através de marcadores, que são enviados junto ao dado. A comunicação UART é realizada sobre o paradigma onde todos os dados são lidos na mesma ordem que são enviados (FIFO, do inglês *First-In, First-Out*), com o receptor iniciando a leitura a partir do marcador de início e parando ao identificar o marcador de final (UART, 2014; CAMPBELL, 2016).

Por ser um dos padrões de comunicação mais simples e de fácil aplicação, ao mesmo tempo que robusto, o padrão UART acaba por ser um dos mais utilizados, estando presente em quase todos os microcontroladores atuais. Um exemplo disso está no microcontrolador ESP8266, que apresenta duas interfaces UART de fábrica (UART0 e UART1). É através da interface UART que programas podem ser gravados e depurados nesse tipo de microcontrolador, por exemplo (ESPRESSIF SYSTEMS, 2020a; OLIVEIRA, S., 2017).

2.3 Wi-Fi

A palavra Wi-Fi há muito já se tornou de senso comum, entretanto, pouco se é falado sobre o real significado dela. Ela deriva do termo em inglês *Wireless Fidelity*, inspirado no termo *High Fidelity* (Hi-Fi, Alta Fidelidade em português), utilizado por amplificadores de áudio. O termo Wi-Fi foi dado posteriormente, nomeando um padrão de comunicação sem fio adotado pela *Wireless Ethernet Compatibility Alliance* (WECA) – grupo de trabalho formado por empresas de tecnologia da época, como 3Com, Nokia, Lucent e Aironet; grupo que, posteriormente viria a se chamar de Wi-Fi Alliance (ALECRIM, 2021; WI-FI ALLIANCE, 2021).

Esse padrão de comunicação sem fio se tratava do IEEE 802.11; um derivado do IEEE 802.3, criado com o propósito de possibilitar a transferência de dados sem fio, tal qual o que já era feito de modo cabeado. Dessa forma, um dispositivo chamado de Ponto de Acesso (AP, do inglês *Access Point*) gera sinal radiofrequência, o qual é captado e interpretado por um outro dispositivo, chamado Estação (STA, do inglês *Station*), que passa a se comunicar com o AP, gerando um conjunto de serviços básico (BSS, do inglês *Basic Service Set*). Para que não aja conflito entre duas redes, cada AP utiliza um identificador de conjunto de serviços (SSID, do inglês *Service Set Identifier*) (ALECRIM, 2021).

Ao contrário do que parece, apenas a terceira versão do padrão (IEEE 802.11b) pode ser chamada de Wi-Fi, quando foi ele que foi adotado pela WECA. A partir do padrão IEEE 802.11b vieram vários modelos outros, os quais buscavam sempre evoluir a tecnologia; adaptando suas respectivas frequências de sinal, largura de banda (BW, do inglês *Band With*),

taxa de transmissão, alcance, método de modulação, quantidade de vias de transmissão simultâneas, entre outros. O quadro abaixo mostra a relação entre os principais parâmetros desses modelos de WiFi (ALECRIM, 2021; IEEE 802.11, 2016).

Quadro 2 - Evolução das Redes WiFi

Tecnologia	Ano	Frequência	Velocidade	Modulação	N. vias
Wi-Fi 1: 802.11b	1999	2,4 GHz	11 Mb/s	DSSS	1
Wi-Fi 2: 802.11a	1999	5 GHz	54 Mb/s	OFDM	1
Wi-Fi 3: 802.11g	2003	2,4 GHz	54 Mb/s	OFDM/DSSS	1
Wi-Fi 4: 802.11n	2009	2,4 GHz/5 GHz	600 Mb/s	MIMO/OFDM	4
Wi-Fi 5: 802.11ac	2019	5 GHz	6 Gb/s	MU-MIMO/ OFDM/ 256-QAM	8
Wi-Fi 6: 802.11ax	2021	2,4 GHz/5 GHz	9,6 Gb/s	MU-MIMO/ OFDMA/1024-QAM	8

Fonte: Adaptado de Alecrim (2021)

Apesar de as tecnologias de transmissão das redes Wi-Fi terem evoluído bastante, até o padrão mais atual, o tempo mostrou que evoluir a transmissão não era a única prioridade, e que outra questão também deveria ser igualmente trabalhada: a segurança das redes. Isso se deve ao fato de que, inicialmente, se tendo o SSID de uma rede Wi-Fi qualquer dispositivo (STA) poderia se conectar e, uma vez conectado, ele teria acesso total sobre a rede, podendo desde modificar a rede em si até poder ter acesso a outros dispositivos da mesma rede (ALECRIM, 2021).

Foi pensando nisso que surgiu padrão de privacidade equivalente a fios (WEP, do inglês *Wired Equivalent Privacy*); o primeiro mecanismo de segurança criado para redes Wi-Fi. Esse padrão não mais permitia que quaisquer dispositivos se conectar à rede sem que eles fornecessem uma senha de acesso de antemão. Entretanto, logo esse padrão se mostrou ineficiente, pois era alvo fácil de ataques, sendo rapidamente ultrapassado. Foi então criado o padrão de acesso cabeado protegido (WPA, do inglês *Wired Protected Access*), o qual incorporou à senha da rede também o endereço de controle de acesso por mídia (endereço MAC, do inglês *Media Access Control Address*) do dispositivo em questão. Após o WPA, vieram outros padrões, como o WPA2, sucessor do WPA, e o padrão de configuração de Wi-Fi protegido (WPS, do inglês Wi-Fi *Protected Setup*); uma melhoria do WPA2. Atualmente, está sendo posto em prática o protocolo WPA 3 (ALECRIM, 2021; WI-FI ALLIANCE, 2021).

Em paralelo a esses modelos de redes, chamadas de redes domésticas, existem as redes de Wi-Fi empresariais (*enterprise*), que utilizam tecnologias ainda mais avançadas. Essas redes podem ser dotadas de sistema de *roaming* – sistema que faz com que vários roteadores possam fazer parte de uma mesma rede, com um único SSID, podendo o usuário transitar entre um e

outro sem se desconectar – possibilitam também que um só roteador possa criar mais de uma rede, com diferentes SSIDs; redes essas que são totalmente independentes (HUAWEI, 2018).

As redes empresariais também utilizam protocolos de segurança como o WPA 2 e 3, entretanto, com uma versão diferente, adaptada do padrão IEEE 802.1x, onde os roteadores são conectados a servidores de autenticação remota (RADIUS, do inglês *Remote Authentication Dial-In User Service*) – um servidor Web que armazena todos os usuários que têm acesso livre à rede. Dessa forma, além de os usuários terem que informar a senha da rede, eles também têm que informar um identificador (ID) válido, tornando essas redes ainda mais seguras a ataques (SECURE W2, 2021).

2.4 Servidor *web*

Como já é sabido, celulares, computadores pessoais e até mesmo calculadoras digitais, necessitam de algum mecanismo interno de armazenamento de dados, o qual se convencionou chamar de memória, tal qual a humana. Da mesma forma, aplicações *web* também necessitam de armazenamento (memória), pois, tal qual seu computador, os sites que são encontrados na internet também necessitam de algum lugar para deixar armazenadas fotos, vídeos e afins. Mecanismo semelhante é utilizado para aplicativos como Instagram e Facebook (OLIVEIRA, S., 2017).

O que acontece é que, geralmente, esses dados são armazenados em uma memória externa, chamada de servidor (*web server*), e os mesmos são buscados quando o usuário solicita; por isso a demora para "carregar" uma página um site qualquer. Esse processo de carregamento de uma página *web*, na prática, significa que o site está enviando uma solicitação para que o servidor forneça algum dado – pedidos como esses são realizados com a utilização de algum protocolo de transferência.

2.4.1 HTTP vs. MQTT

O protocolo de transferência de hipertexto (HTTP, do inglês *Hypertext Transfer Protocol*), como o próprio nome já diz, é um protocolo que efetua a transferência de hipertextos. Isso significa, que esse protocolo trabalha a partir do transporte de documentos em linguagem de marcação de hipertexto (HTML, do inglês *HyperText Markup Language*), que são a base para a construção de qualquer página web. Dessa forma, a partir de requisições geradas no navegador, o HTTP faz com que esses comandos sejam traduzidos na forma de um novo

documento HTML, composto de dados buscados no servidor *web*, formando os chamados sites (NIELSEN, 1997).

Por outro lado, o protocolo de transporte por telemetria em fila (MQTT, do inglês *Message Queuing Telemetry Transport*), que é baseado na transferência de dados e não de documentos, o que significa que não é mais necessária a utilização de documentos HTML nem de um navegador para fazer a transferência de dados. A transferência é realizada de uma "máquina" para outra (M2M, do inglês *Machine-to-Machine*), a partir da interação indireta entre quem envia e quem recebe o dado, com o intermédio de um servidor especial, chamado de MQTT Broker – os arquivos são transportados na forma de vetores, a partir de publicações e subscrições dos clientes com o servidor (OLIVEIRA, S., 2017; SEROZHENKO, 2017).

Desse modo, nota-se uma clara diferença entre MQTT E HTTP, de modo que o protocolo HTTP apresenta uma complexidade e, consequentemente, um tamanho bem maior, quando comparados ao MQTT. Isso afeta diretamente a velocidade e o consumo que são necessários para que seja realizado o transporte de dados, de modo que é estimado que o protocolo MQTT tenha de 20 a 93 vezes mais rapidez que o HTTP; isso tudo com um tráfego de dados 50 vezes menor (BARTNITSKY, 2018; SEROZHENKO, 2017).

Tendo isso tudo em mente, o protocolo HTTP continua tendo sua hegemonia, quando se trata de aplicações *web*, mas para aplicações onde é necessário apenas que dados sejam transferidos de um lugar a outro, o MQTT é a melhor escolha. É por esses fatos que o MQTT é o protocolo mais utilizado para aplicações de IoT, as quais necessitam apenas que os dados sejam transportados do dispositivo para o servidor, e vice-versa; isso é bastante interessante, devido ao baixo poder computacional utilizado (OLIVEIRA, S., 2017; SEROZHENKO, 2017).

3 METODOLOGIA

A metodologia utilizada partiu da divisão do período de projeto em três ciclos consecutivos de produção. Cada ciclo foi composto de etapas de pesquisa, planejamento, prototipação, teste e adequação. Dessa forma, o desenvolvimento ocorreu de maneira continua, e as funcionalidades foram adicionadas com o tempo, tendo, ao final do processo, um produto finalizado, já com todas as funcionalidades disponíveis, testadas e melhoradas.

O primeiro ciclo teve a finalidade de desenvolver uma primeira versão produto (MVP). Essa versão era constituída de um *software,* produzido em linguagem de programação C/C++; sendo ele, inicialmente, embarcado em uma placa NodeMCU ESP32S, escolhida por motivos de praticidade. Essa versão já era capaz de enviar e receber dados via UART, realizar conexão a redes Wi-Fi, realizar envio e recebimento de dados via protocolo MQTT; ainda que de forma simplificada.

Na segunda rodada de produção foram feitas alterações no código, de modo que ele se tornasse compatível com os chips da família ESP8266EX, utilizando uma placa Nodemcu ESP12E para a realização dos testes. Também foi realizado o projeto do *hardware* do produto, de modo que não mais fosse necessária a utilização da plataforma NodeMCU. Foi produzido o desenho esquemático do circuito, o roteamento das trilhas da placa de circuito impresso e o *plot* do modelo tridimensional.

Na terceira foram feitos os ajustes necessários, tanto de *software*, com a possível adoção de novas bibliotecas, quanto de *hardware*, com a mudança das terminações a serem utilizadas; tudo isso de modo que seja possível utilizar os chips ESP12S. Ao final dessa etapa foi produzido o primeiro protótipo físico do circuito.

Como as etapas anteriores foram bem-sucedidas, foi realizada um último ciclo, de modo que o dispositivo também pudesse ter suporte ao protocolo HTTP. Também foi finalizada a plataforma *web* de configuração do dispositivo, que foi desenvolvida em paralelo ao desenvolvimento das etapas anteriores; nela o usuário pode realizar a configuração do dispositivo, alterando os parâmetros internos do *firmware*.

Por último, já todo o programa finalizado, foi enfim adicionada a capacidade alterar o *firmware* utilizado no dispositivo, e os arquivos gravados no chip, à distância. Isso foi feito a partir da adição ao projeto da tecnologia atualização pelo ar (OTA, do inglês *Over-the-air),* sendo posta de forma atrelada a página de configuração do dispositivo.

4 DESENVOLVIMENTO

Tal qual já foi previamente explicitado, o projeto em questão é constituído da produção de um módulo de transferência de dados, sendo ele embarcado em um microcontrolador ESP8266. Sendo assim, o projeto do módulo é composto de duas partes principais: uma parte representada pelo software a ser embarcado, e outra, do hardware físico.

Para a realização do projeto e do desenvolvimento do software, foi optado pela utilização do editor de código *Visual Studio Code* (VS *Code*). Esse é um ambiente de desenvolvimento gratuito que contém as mais diversas linguagens de programação e suporte para os sistemas operacionais Windows, Linux e MacOS.

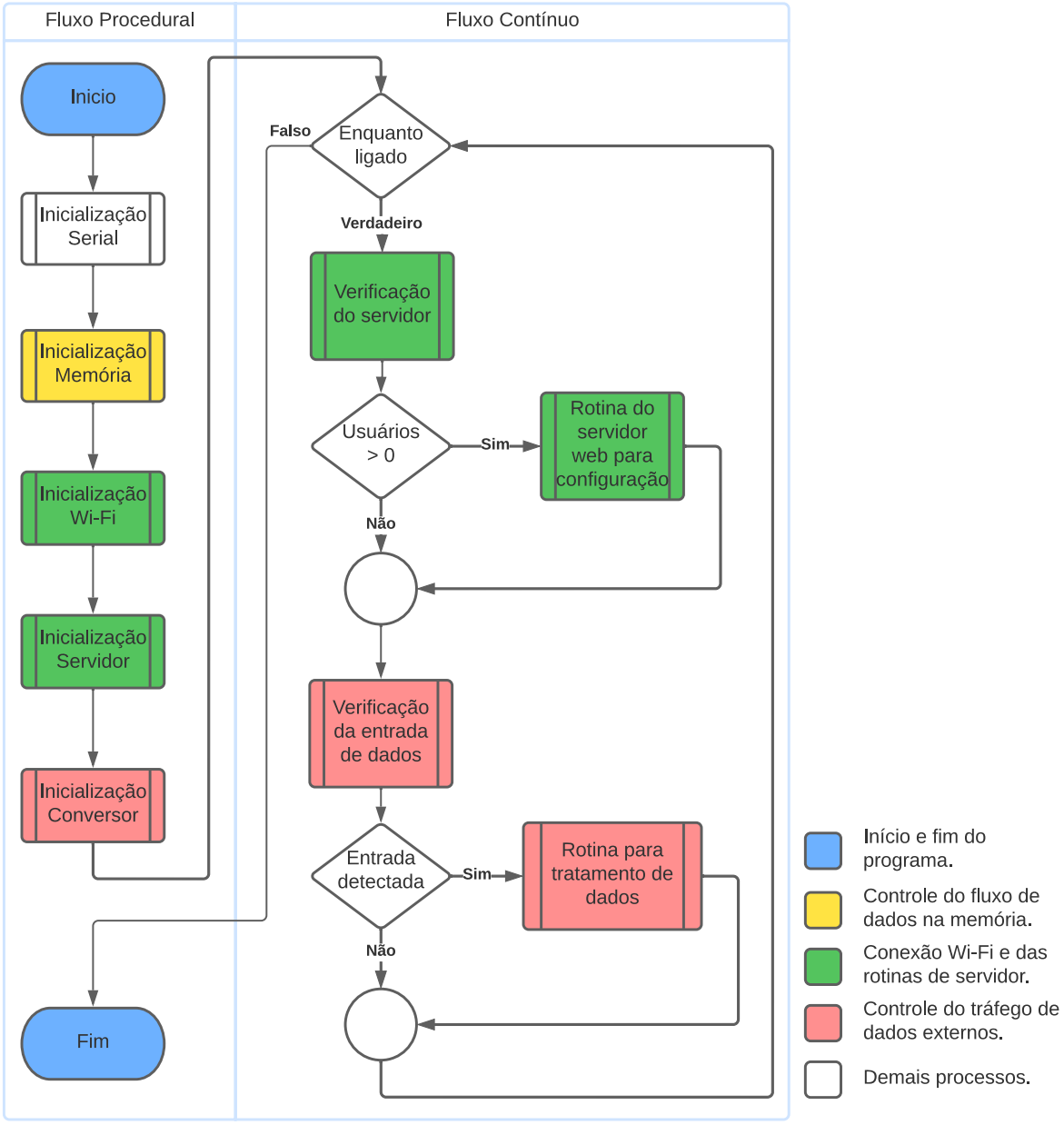
Esse software ainda conta com a uma vasta biblioteca de extensões, que podem ser adicionadas a seu ambiente de desenvolvimento, aprimorando a experiência do usuário; em especial a extensão PlatformIO, que foi bastante utilizada durante o desenvolvimento. Ela é uma ferramenta gratuita que, ao ser adicionada ao ambiente do VS Code, possibilita a programação para mais de mil placas comerciais.

Por último, para a realização do projeto do hardware (desenho esquemático, roteamento e reprodução tridimensional) utilizado, foi optado pela utilização da plataforma EasyEDA. Essa é uma plataforma online utilizada para o design de circuitos eletrônicos. Ela conta com mais de 1 milhão de bibliotecas disponíveis e atende mais de 2 milhões de engenheiros, ao redor do mundo.

4.1 *Firmware*

Visando uma melhor organização do software em questão, visto a complexidade e extensão dos processos operados, ele foi organizado de forma dividida. Nele são delimitadas 3 principais partes, sendo a primeira responsável pelo controle da conexão Wi-Fi e das rotinas de servidor; a segunda responsável pelo controle do fluxo de dados na memória e a última pelo controle do tráfego de dados externos.

Figura 1 - Fluxograma Geral do Programa



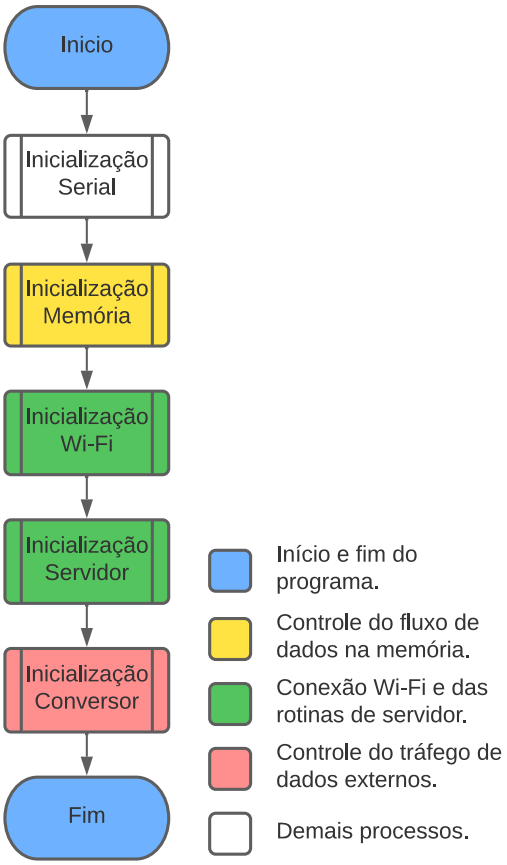
Fonte: Autoria Própria (2022).

O Fluxograma apresentado na Figura 1, se trata de uma representação simplificada de como o programa desenvolvido funciona internamente. Para a produção do código, foi tido como base a estrutura de projeto utilizada dentro do ambiente de desenvolvimento dos dispositivos Arduino, então o código apresenta uma divisão evidente entre os processos, sendo a parte de fluxo procedural, onde são inicializados os processos, e a outra parte é de fluxo contínuo, onde são realizados os principais processos em *loop*. Para facilitar a familiarização com a divisão presente no código, o diagrama também foi organizado da mesma forma.

4.1.1 Fluxo Procedural

Fluxos do tipo procedural são aqueles que são executados no formato de fila de execução, sendo os processos executados de forma sequencial e ordenada, a partir de um estímulo inicial (entrada). No contexto do projeto, os processos que compreendem o fluxo procedural são os de inicialização, que ocorrem assim que o dispositivo é "ligado". A Figura 2 apresenta apenas a parcela da figura 1 responsável pelo fluxo procedural.

Figura 2 - Fluxo Procedural



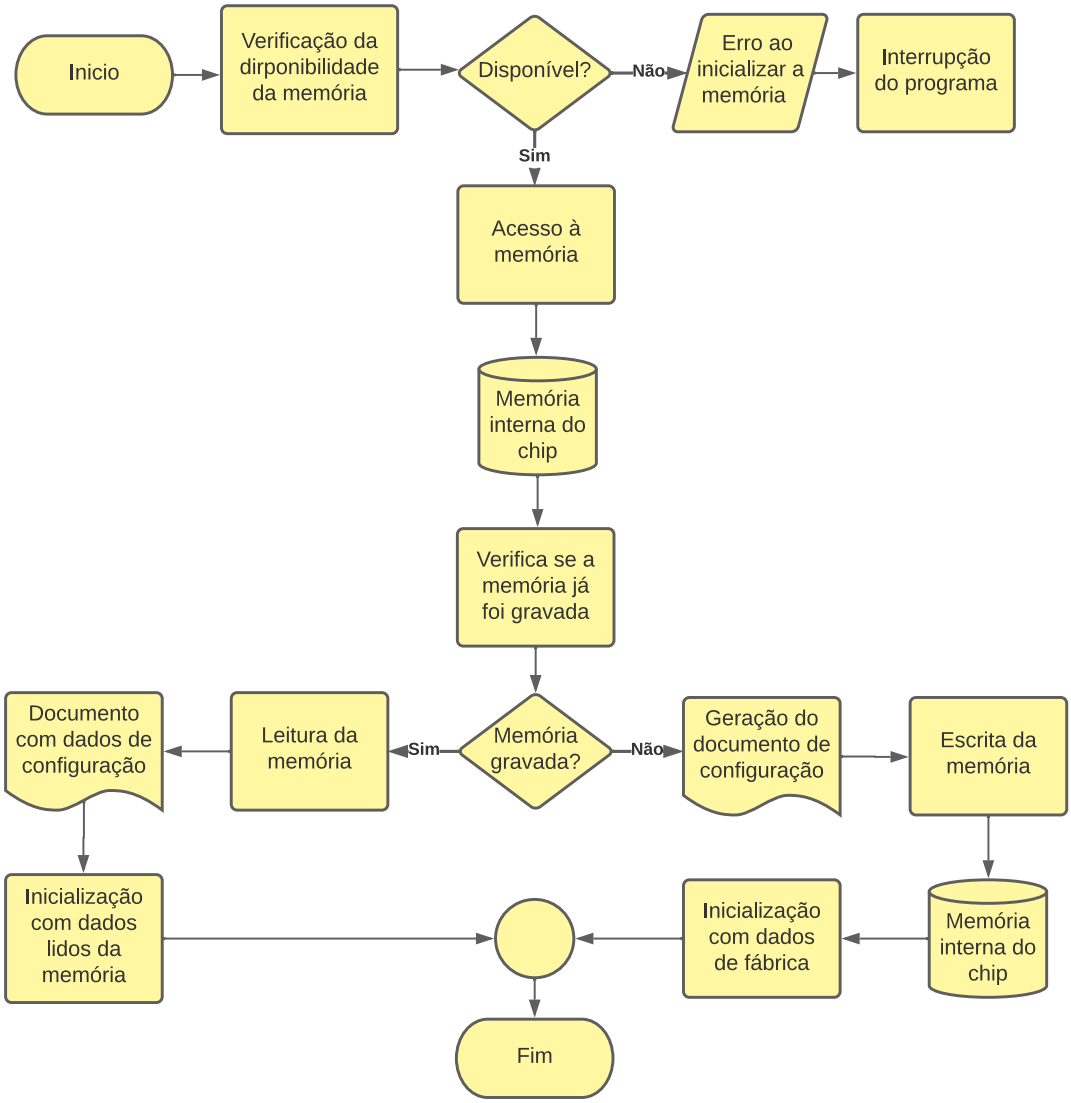
Fonte: Autoria Própria (2022)

Primeiramente, é realizada a inicialização das portas seriais, que são necessárias pelos demais processos. O processo de inicialização serial consta, como processo predefinido pelo fato de que ele é realizado majoritariamente pela própria biblioteca Serial, herdada do Arduino, cabendo ao desenvolvedor, apenas fornecer alguns parâmetros básicos.

O próximo processo a ser executado é o de inicialização da memória. Essa parte do programa, como o nome indica, tem como função acessar a memória interna do chip e resgatar os dados que foram gravados durante a última utilização do módulo. Esse processo por completo é apresentado na Figura 2. Primeiro é verificada a disponibilidade da memória e caso

a mesma não esteja disponível representa um erro no dispositivo, sendo então encerrado programa.

Figura 3 - Inicialização da Memória



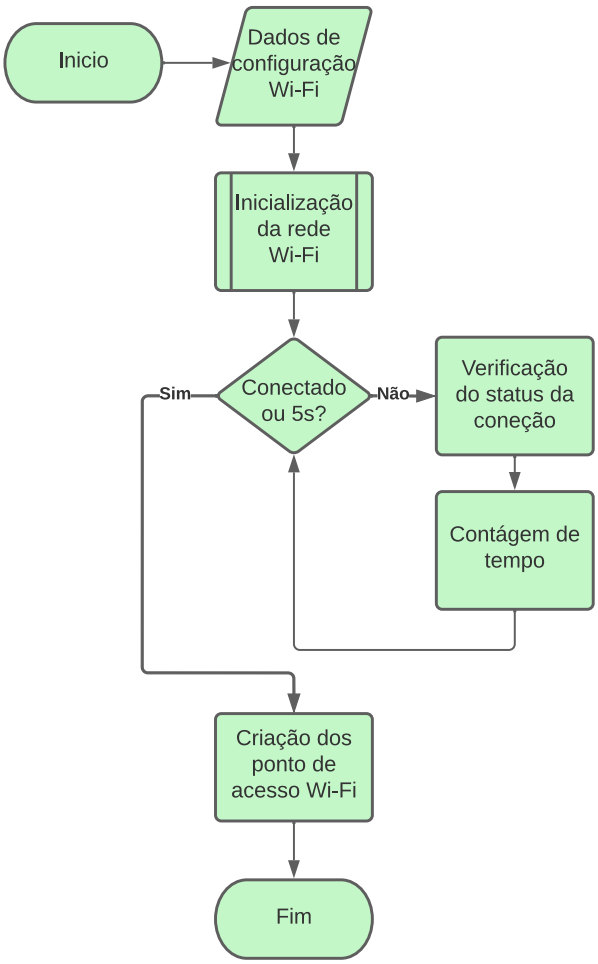
Fonte: Autoria Própria (2022)

Caso a memória esteja disponível, finalmente ela pode ser acessada e é verificado se a mesma já foi gravada. Caso sim, significa que o dispositivo já foi utilizado e que já apresenta uma configuração prévia. Desse modo, é realizada uma leitura na memória e é acessado o documento de texto que contém última configuração salva. Sendo assim, os dados acessados são inicializados no programa na forma de variáveis de código; variáveis essas que serão utilizadas pelo restante do software.

Ainda no processo de verificação descrita anteriormente, caso não seja encontrado nada gravado na memória, é gerado e gravado na memória um novo documento de

configuração, utilizando os dados de configuração de fábrica – uma série de variáveis com valores pré-estabelecidos. Em seguida é realizado o processo de inicialização.

Figura 4 - Inicialização Wi-Fi



Fonte: Autoria Própria (2022)

Voltando no diagrama da Figura 2, o próximo processo a ser realizado é o de inicialização Wi-Fi; o processo completo é mostrado na Figura 4. Primeiramente são verificados os dados de configuração do dispositivo, a fim de descobrir se existe alguma rede WiFi pré-definida. Em seguida é inicializada a conexão à rede Wi-Fi pretendida, caso o usuário tenha executado uma configuração prévia da mesma. Após inicializada a conexão, o código aguarda dentro de um loop até que uma das duas possíveis condições sejam satisfeitas: que a conexão Wi-Fi seja estabelecida, ou que o tempo limite seja alcançado (5 segundos).

Prosseguindo, é realizada a criação de um ponto de acesso Wi-Fi próprio do dispositivo, seguindo as configurações previamente estabelecidas. Esse ponto de acesso Wi-Fi,

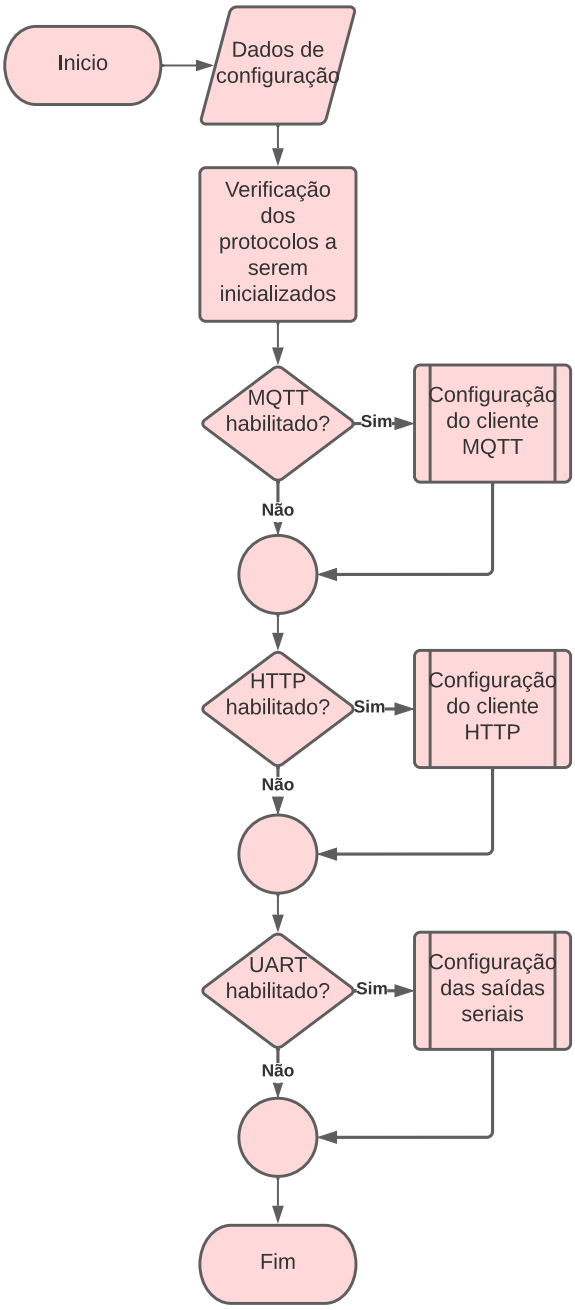
futuramente, servirá para que o usuário possa acessar o módulo utilizando outros dispositivos com Wi-Fi.

Retornando ao diagrama geral, mostrado na Figura 2, o próximo processo a ser desempenhado é o de inicialização do servidor. Esse bloco é o responsável por fazer a ligação entre o servidor e o dispositivo, possibilitando a comunicação entre ambos. Ele determina qual parte do código deve ser invocada em resposta a cada uma das requisições que o navegador pode fazer ao dispositivo, que passa a funcionar como um servidor, enviando e recebendo dados conforme a demanda.

Ainda na Figura 2, o próximo processo a ser executado é o de inicialização do tratamento de dados (conversor). Ele diz respeito à inicialização dos protocolos que futuramente serão utilizados para a conversão dos dados que o módulo se propõe a fazer, sendo esses os protocolos MQTT, HTTP e UART.

A Figura 5 apresenta uma visão mais ampliada de como a tal inicialização é feita. Primeiramente, é verificado quais protocolos devem ser inicializados, de acordo com a configuração definida previamente pelo usuário. Em seguida, é verificado um a um quais protocolos estão habilitado para uso, sendo realizada a inicialização de cada um deles, seguindo os processos definidos por cada uma das respectivas bibliotecas.

Figura 5 - Inicialização do Conversor



Fonte: Autoria Própria (2022)

4.1.2 Fluxo Contínuo

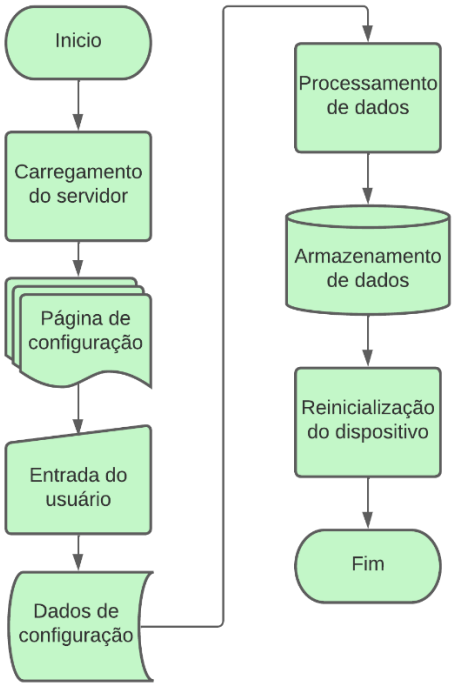
Finalizadas todos os processos de inicialização, é dado início ao fluxo contínuo. Ele é constituído *looping* “infinito”, chamado de ciclo de programa; conceito bastante utilizado por chips microcontroladores como o utilizado no projeto. Como o chip em questão apresenta apenas 1 núcleo, todo o processamento dele fica concentrado nesse loop, sendo necessário

trabalhar com interrupções para fazer executar processos concorrentes, voltando em seguida ao *loop* principal.

No interior do ciclo do programa são primordialmente desempenhadas duas atividades que se assemelham em nome. Elas são a verificação do servidor e a verificação da chegada de dados externos. Esses dois processos ocorrem alternada e continuamente durante a maior parte do tempo em que o dispositivo se encontra ligado.

A verificação do servidor é um processo desempenhado por uma rotina da própria biblioteca utilizada, onde é realizada uma verificação constante da existência de *requests* provenientes de algum navegador ao tentar acessar a página de configuração do dispositivo. No caso da ocorrência de algum pedido, é colocada em uso a estrutura previamente criada durante o processo de inicialização do servidor.

Figura 6 - Rotina do Servidor para Configuração



Fonte: Autoria Própria (2022)

O processo descrito anteriormente pode ser melhor entendido a partir da Figura 6. Ao tentar acessar a página de configuração, o navegador envia as primeiras *requests* ao dispositivo, que, em resposta a tais pedidos, executa um processo de leitura em sua memória interna, retornando ao navegador os arquivos por ele solicitados. Tais arquivos se encontram, até então, gravados na memória interna do chip – arquivos dos tipos HTML, CSS e Java Script. Da união desses arquivos é composta a página de configuração do dispositivo.

Em seguida, após o carregamento, o usuário pode, então, entrar com as configurações que ele deseja para o dispositivo. Após isso, o usuário solicita o salvamento dos dados ao navegador, que repassa essa solicitação ao dispositivo. Nesse momento, são acionados os métodos responsáveis por armazenar esses dados na forma de variáveis. Finalmente, após o fornecimento de todas as informações pelo usuário, ocorre a escrita desses dados no arquivo de configuração do dispositivo, citado anteriormente, e ocorre a reinicialização do *hardware*.

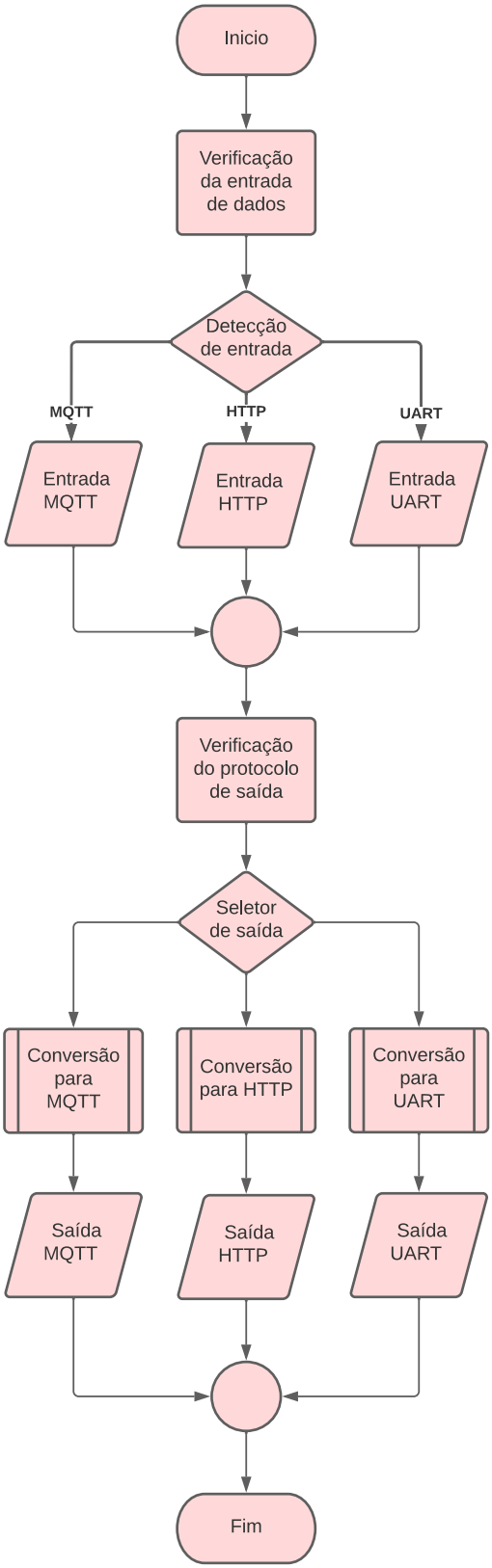
O processo de verificação da entrada de dados, ao contrário do que se imagina, se difere bastante do processo de verificação do servidor. Nele, primeiramente é realizada uma verificação inicial para identificar quais protocolos se encontram habilitados, mediante a configuração definida pelo usuário durante o processo apresentado anteriormente. Cada protocolo habilitado corresponde a uma entrada externa e uma saída externa para o dispositivo.

Desse modo, já com a entrada definida, agora é executada a verificação da entrada selecionada para saber da chegada de dados externos. Os processos de verificação para os protocolos MQTT e UART se assemelham em conceito do descrito para a verificação do servidor, pois todos são processos assíncronos – ocorrem sem necessidade de comando. Entretanto, em método eles se diferem, pois enquanto o protocolo MQTT utiliza uma metodologia de *callback*, semelhante ao utilizado pela página de configuração, o UART utiliza de um mecanismo de *buffer*, armazenando os dados que chegam ao dispositivo para posteriormente serem lidos.

Por outro lado, o protocolo HTTP possui uma mecânica de funcionamento síncrona, o que significa que os dados apenas são recebidos a partir da ação direta do usuário. Isso na prática, acontece de modo que, como o HTTP utiliza uma mecânica de requisição, os dados apenas são recebidos pelo dispositivo a partir de uma solicitação prévia do servidor. Para isso, foi desenvolvido um mecanismo de *clock* onde as requisições são realizadas a uma frequência definida pelo usuário.

Sob um panorama geral, o trabalho executado pelo dispositivo com os protocolos utilizados muito se assemelha ao utilizado por um par multiplexador/demultiplexador. Isso pode ser bem observado a partir da Figura 7, que representa bem o processo realizado para que ocorra a conversão dos dados.

Figura 7 - Fluxo de Trabalho do Conversor



Fonte: Autoria Própria (2022)

Primeiramente, é verificado se ocorreu a entrada em algum dos protocolos utilizados, e, caso seja identificada a presença de dados em quaisquer delas, é então executada a leitura dos mesmos. Após isso ocorre a verificação de qual protocolo a ser utilizado para a saída – todos menos o utilizado durante a leitura do dado recebido.

Por fim, ocorre o tratamento e conversão do dado para o protocolo de saída, de acordo com o que é estabelecido pelas suas respectivas bibliotecas. Após a conversão, ocorre o envio dos dados, finalizando o processo de transferência. Todo esse processo é repetido constantemente, a cada dado que chega, e ele representa a principal finalidade do projeto. O *firmware* embarcado no dispositivo está exposto no Apêndice B.

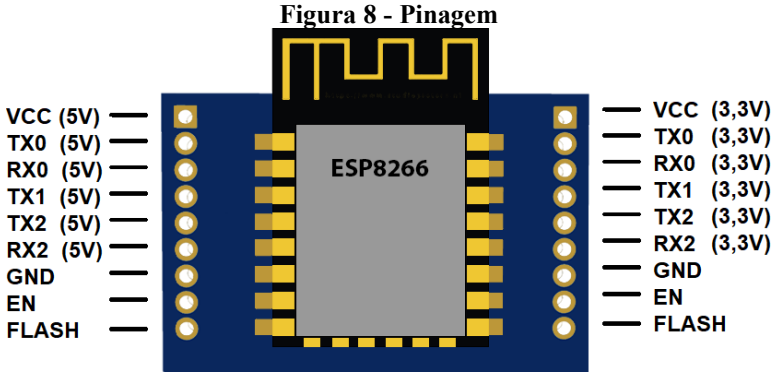
4.2 *Hardware*

Como exposto, previamente, o intento final do hardware proposto é o de compor um módulo microcontrolado. Nesse âmbito, foram traçados alguns parâmetros para serem tomados como base, durante o desenvolvimento do hardware: ele deve possuir um tamanho reduzido, um design simples e de fácil utilização.

A escolha da utilização do microcontrolador ESP-12S deriva do fato de ele ser uma versão melhorada do seu "irmão" ESP-12E, possuindo um designe de antena melhorado. Inicialmente, era pretendido utilizar a versão ESP-07, pelo fato de ele possuir uma antena cerâmica e conector U-FL, o que lhe concede um melhor desempenho em aplicações *wireless*. Entretanto, assim como o ESP-01, ele tem apenas 1 MB de memória flash, o que se provou insuficiente para a gravação de arquivos.

Pensando no quesito aplicabilidade, quando focando em uma maior adaptação ao mercado, foi adicionado ao projeto final o circuito integrado TXB0108PWR; um conversor de nível lógico bidirecional e de 8 bits, produzido pela empresa Texas Instruments, juntamente ao regulador de tensão AMS1117, produzido pela empresa Advanced Monolithic Systems (AMS). Graças a essas adições, torna-se possível fazer com que o módulo seja compatível tanto com aplicações em 3,3 volts quanto em 5 volts, o que melhora sua utilização. A Figura 8 apresenta bem os terminais do módulo produzido.

Figura 8 - Pinagem



Fonte: Autoria Própria (2022)

O roteamento das trilhas da placa de circuito impresso (PCB, do inglês *Printed Circuit Board*), foi projetado de modo a diminuir ao máximo o tamanho do protótipo final, sendo reduzidas praticamente às dimensões do microcontrolador, graças à utilização de uma placa com camada dupla de cobre. Outras características importantes são a presença do chip microcontrolador na parte superior, visando manter livre os canais de transmissão *wireless*. O designe espelhado, de um lado tendo as conexões para 5V e do outro para 3,3V, foi feito de modo a facilitar a utilização.

Circuitos como esse, que trabalham com transmissão de dados (sem fio e cabeados), necessitam de um maior cuidado no projeto, para diminuir a influência de ruídos. Sendo assim, foram utilizados filtros capacitivos nos circuitos de alimentação e foi deixada livre a área nas imediações da antena; tal qual especificado pelos respectivos fabricantes dos componentes. Também foi tomado o cuidado para não se ter trilhas em 90º e foram utilizados planos de aterramento, de modo a barrar tanto interferências internas quanto externas.

Por fim, foi criado o layout tridimensional da placa de circuito impresso. Nele foi optado pela utilização apenas de componentes de montagem em superfície (SMD, do inglês *Surface Mounted Device*), localizados, em sua maioria, na parte inferior da placa. Todos os diagramas supracitados, produzidos para a fabricação da placa estão disponíveis no Apêndice A do trabalho, e a versão final da placa que foi encomendada para a produção industrial é apresentada na Figura 9.

O designe do módulo físico foi elaborado de forma ao dispositivo ser o mais compacto e prático, tendo o tamanho próximo ao de uma foto 3x4. O projeto da placa foi realizado com o intuito de tornar a experiência do usuário o mais simples possível, podendo ser facilmente incorporado a outros projetos.

Figura 9 - Vistas Inferior e Superior do Produto

Fonte: Autoria Própria (2022)

5 RESULTADOS E DISCUSSÕES

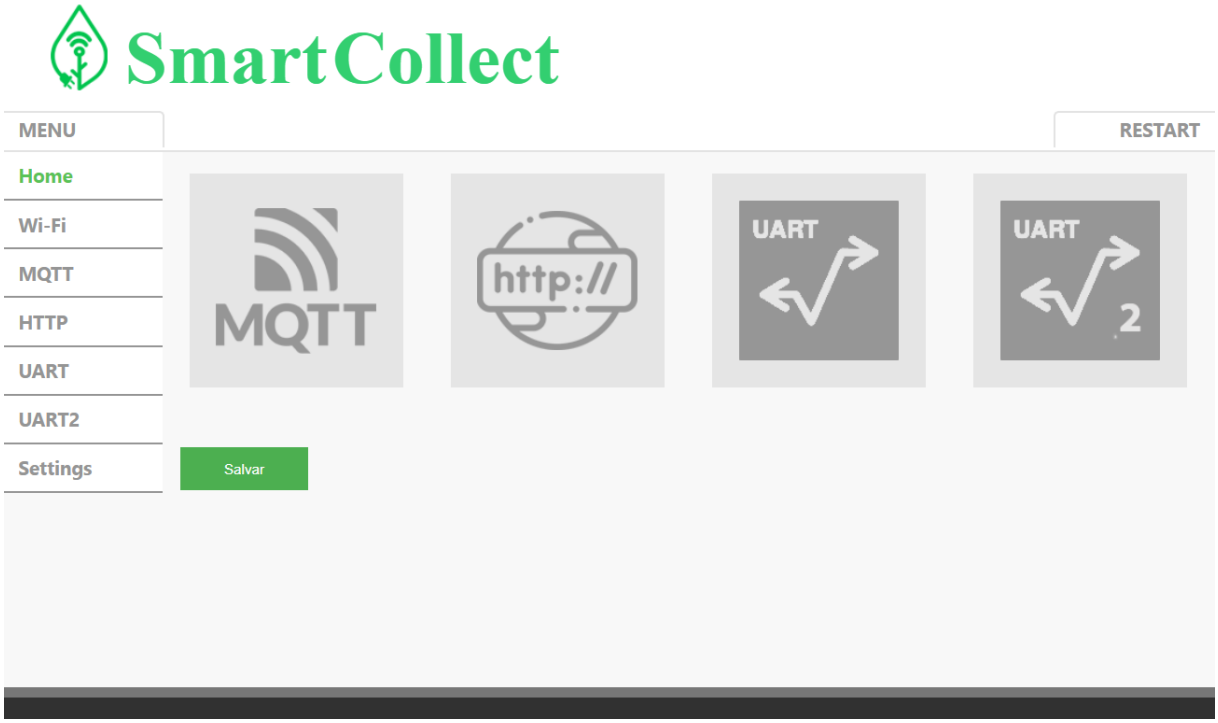
A seguir, será apresentada a seção de teste do dispositivo implementado, descrevendo os principais resultados obtidos durante sua operação.

Ao ser ligado pela primeira vez, pelo fato de não ter ocorrido nenhuma configuração prévia pelo usuário, o dispositivo inicia ainda com suas configurações de fábrica. Desse modo, ele inicia sem conexão à internet e sem nenhum protocolo de comunicação definido. Sendo assim, a única ação externa que se inicia ao utilizar o dispositivo pela primeira vez é a criação de um ponto de acesso Wi-Fi.

Ao se conectar na rede criada pelo módulo, poderá surgir algum alerta no aparelho utilizado pelo usuário, ao se conectar (celular, notebook, etc.), afirmando que não é possível se conectar à internet. Isso acontece porque essa não é uma conexão de internet e sim de configuração. Desse modo, a única página que ela dá acesso é a página de configuração do dispositivo, acessível ao inserir o endereço do protocolo de internet (IP, do inglês Internet Protocol) do dispositivo no campo de busca de algum navegador. O endereço de IP de fábrica do dispositivo é 192.168.11.1.

Inicialmente, ao acessar a página de configuração a partir do endereço de IP, é apresentada a página inicial da plataforma, tendo presente ao lado esquerdo a presença de um menu para a navegação pelo restante da plataforma. Essa primeira página é destinada para que o usuário escolha quais protocolos ele deseja utilizar em seu dispositivo, e, em seguida, prosseguir ao salvamento dos dados, utilizando o botão “salvar”. A Figura 9 apresenta um recorte da página descrita.

Figura 10 - Página Inicial de Configuração, Versão Desktop



Fonte: Autoria Própria (2022)

Utilizando o menu de navegação e clicando no título “WIFI” acontece o redirecionamento automático para a página de configuração da conexão Wi-Fi. Esse campo é destinado para que a pessoa que utilizar o dispositivo consiga fornecer ao sistema os dados de conexão da rede à qual se deseja que o aparelho efetue a conexão, a fim de que ele consiga se conectar à internet. É solicitado que o usuário forneça o SSID e a senha da rede Wi-Fi. Para melhor entendimento, a Figura 10; fornece um recorte dos campos supracitados.

Figura 11 - Página de Configuração Wi-Fi, Versão Móvel

Fonte: Autoria Própria (2022)

O terceiro campo do menu, onde está escrito “MQTT”, como é de se imaginar, dá acesso à página destinada à configuração dos parâmetros do protocolo MQTT. São apresentados vários campos de dados, nos quais é solicitado o fornecimento da URL do servidor broker MQTT o qual o módulo deve se conectar, o tópico onde os dados devem ser publicados, o tópico que o dispositivo deve assinar para receber os pacotes, assim como também é solicitado o identificador (ID MQTT), o usuário e a senha, caso sejam necessários. Também existe a possibilidade de fornecer o IP do servidor, para o caso de aplicações locais; a porta da rede, um tópico para a publicação do status do dispositivo e uma mensagem de “último desejo”. Todas essas informações podem ser conferidas na Figura 11.

Figura 12 - Página de Configuração MQTT, Versão Móvel

Fonte: Autoria Própria (2022)

Ainda existe um campo “HTTP”, para a página de configuração do servidor HTTP onde são solicitados como o *Host* do servidor e os identificadores de recursos (URI, do inglês *Uniform Resource Identifier*) de requisição; um campo “UART”, para a definição dos parâmetros da conexão Serial, como o *baud rate* da comunicação – a especificação utilizada para os pacotes é de 8 *bits* de dados com um *stop bit* e sem paridade, que é a especificação padrão da biblioteca utilizada.

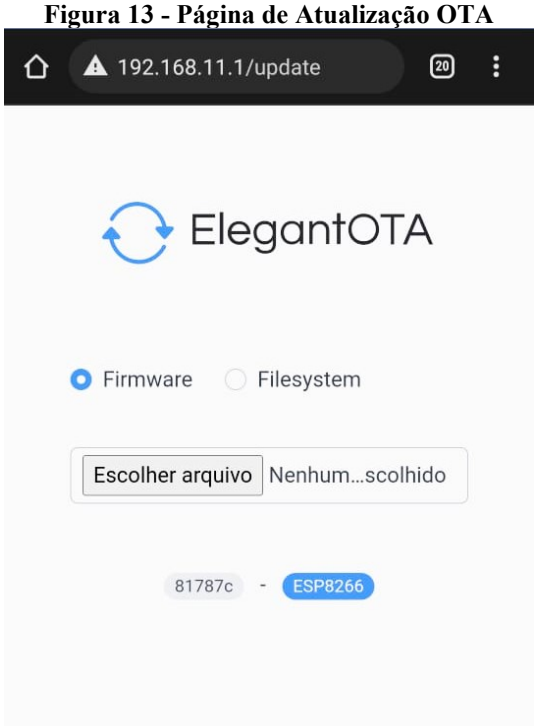
Por último, existe ainda, um campo “SETTINGS”, onde o usuário pode redefinir as configurações do *access point* WiFi criado pelo módulo, podendo alterar, como o SSID da rede gerada e inserir uma senha para o controle do acesso de usuários à rede. Também é possível configurar qual o IP do dispositivo, o *gateway* e a máscara de rede.

Após o término de configuração do descrito, tendo o operador já fornecido ao sistema todos os parâmetros desejados, e tendo efetuado o salvamento para cada página, o próximo

passo a ser tomado é o de acionar o botão presente no canto superior direito da tela, com a inscrição “RESTART”. Após esse botão ser acionado, o dispositivo finaliza o salvamento dos dados fornecidos e executa a reiniciação *firmware* para que ocorra uma nova inicialização, já com os novos parâmetros em vigor.

Continuando na página de configuração, ainda existem três URIs ocultas, que não podem ser acessadas diretamente da página de configuração: “/reset” e “/update”. A primeira delas tem a função de fazer com que o dispositivo retorne às configurações de fábrica; já a segunda leva o usuário à página apresentada na Figura 12. Essa é uma página disponibilizada pela biblioteca ElegantOTA, e permite que tanto o *firmware* do dispositivo quanto os arquivos armazenados nele sejam alterados a distância.

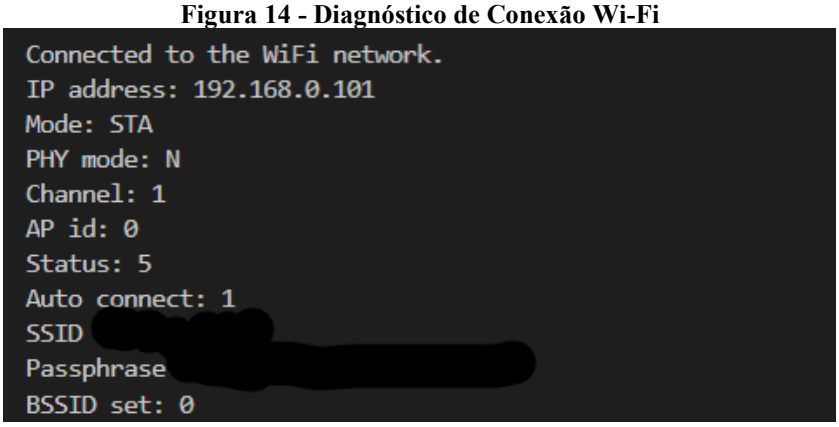
Figura 13 - Página de Atualização OTA



Fonte: Autoria Própria (2022)

Sendo assim, após o dispositivo receber os dados de configuração do usuário, o sistema enfim poderá exercer suas funcionalidades por completo. Tal qual comentado anteriormente, uma das primeiras execuções do sistema é de tentar se conectar à internet, a partir de alguma rede Wi-Fi. Como não se trata mais de uma primeira inicialização, é suposto que o usuário já tenha fornecido os dados de conexão a alguma rede do tipo. A Figura 12 apresenta um recorte do *status* do dispositivo conectado a uma rede Wi-Fi convencional, retirado diretamente do monitor serial do PlatformIO, rodando a função “printDiag()” da biblioteca ESP8266WiFi.

Figura 14 - Diagnóstico de Conexão Wi-Fi



Fonte: Autoria Própria (2022)

Com a conexão à internet estabelecida, e contando que não ocorra nenhum problema de execução o módulo está apto a fazer o transporte de dados entre qualquer um dos três protocolos (MQTT, HTTP e/ou UART) disponíveis. Dessa forma, apenas é necessário que o usuário tenha selecionado com quais ele deseja trabalhar e tenha disponibilizado os seus respectivos parâmetros, corretamente.

Inicialmente foi avaliada a implementação do protocolo de integração de circuitos (I^2C, *Inter-Integrated Circuit*), mas a implantação dele não se demonstrou viável para o projeto em questão. Isso foi devido ao fato de que boa parte das aplicações do protocolo I^2C utilizam bibliotecas próprias, que fornecem ao dispositivo com o qual se deseja comunicar as os dados necessários para o funcionamento do dispositivo. Na prática isso significa que o usuário também seria responsável enviar tais dados; tal fato foge da proposta inicial do trabalho, pois torna o trabalho do projetista ainda mais complexo.

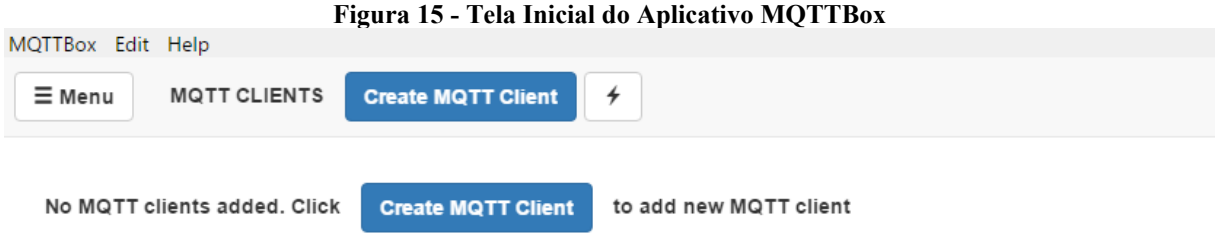
Visto isso, para que o projeto inteiro não tivesse que ser refeito, devido à ausência do protocolo, a porta I^2C, presente no projeto inicial, foi convertida em uma porta UART extra, chamada de UART2, e os terminais SDA e SCL foram transformados em RX2 e TX2. Isso também irá auxiliar o usuário, que agora terá duas portas UART disponíveis (além do terminal TX1, por onde ele tem acesso aos logs do programa), podendo conectar ao mesmo módulo dois microcontroladores diferentes, inclusive com taxas de transferências diferentes, o que antes seria possível apenas a partir da utilização de dois módulos.

Entretanto, como foi informado na Fundamentação Teórica, o microcontrolador ESP8266 apresenta nativamente apenas uma porta UART, além de uma meia UART, com apenas o pino de transmissão (TX) disponível. Visto isso, a utilização da porta I^2C como UART é inicialmente inválida. Por outro lado, essa conversão é possível por software, o que foi feito, a partir da utilização da biblioteca SoftwareSerial. Essa porta, a princípio, apresenta as mesmas

funcionalidades que uma convencional, mas com algumas deficiências, como o fato ela possuir um *baud rate* máximo de 115200 símbolos por segundo. Por isso, é aconselhável que o usuário dê prioridade à primeira porta.

Para que ocorresse as trocas de dados utilizando o protocolo MQTT foi utilizada a plataforma MQTTBox. Essa é uma aplicação gratuita para computador, de simulação para o protocolo, funcionando como um cliente MQTT. Dessa forma, é possível fazer a simulação da comunicação entro dois clientes, sendo um deles o MQTTBox, e o outro, o próprio módulo. A Figura 13 apresenta a tela inicial do aplicativo MQTTBox, em sua versão disponível para Windows 10, pela Microsoft Store.

Figura 15 - Tela Inicial do Aplicativo MQTTBox



Fonte: Autoria Própria (2022)

Porém, a comunicação por MQTT não ocorre diretamente entre os dois clientes, sendo necessária a utilização de um *broker*, para armazenar os envios e redistribuir os dados. O *broker* escolhido para desempenhar essa função foi o Mosquitto, disponível em "mqtt://test.mosquitto.org". Nele, foram criados dois tópicos: um nomeado de "/módulo" e o outro chamado "/box", para onde são enviados os dados advindos do dispositivo e do MQTTBox, respectivamente. Também foi criado um tópico "/status", onde é sinalizado para a rede, em caso de desconexão do módulo.

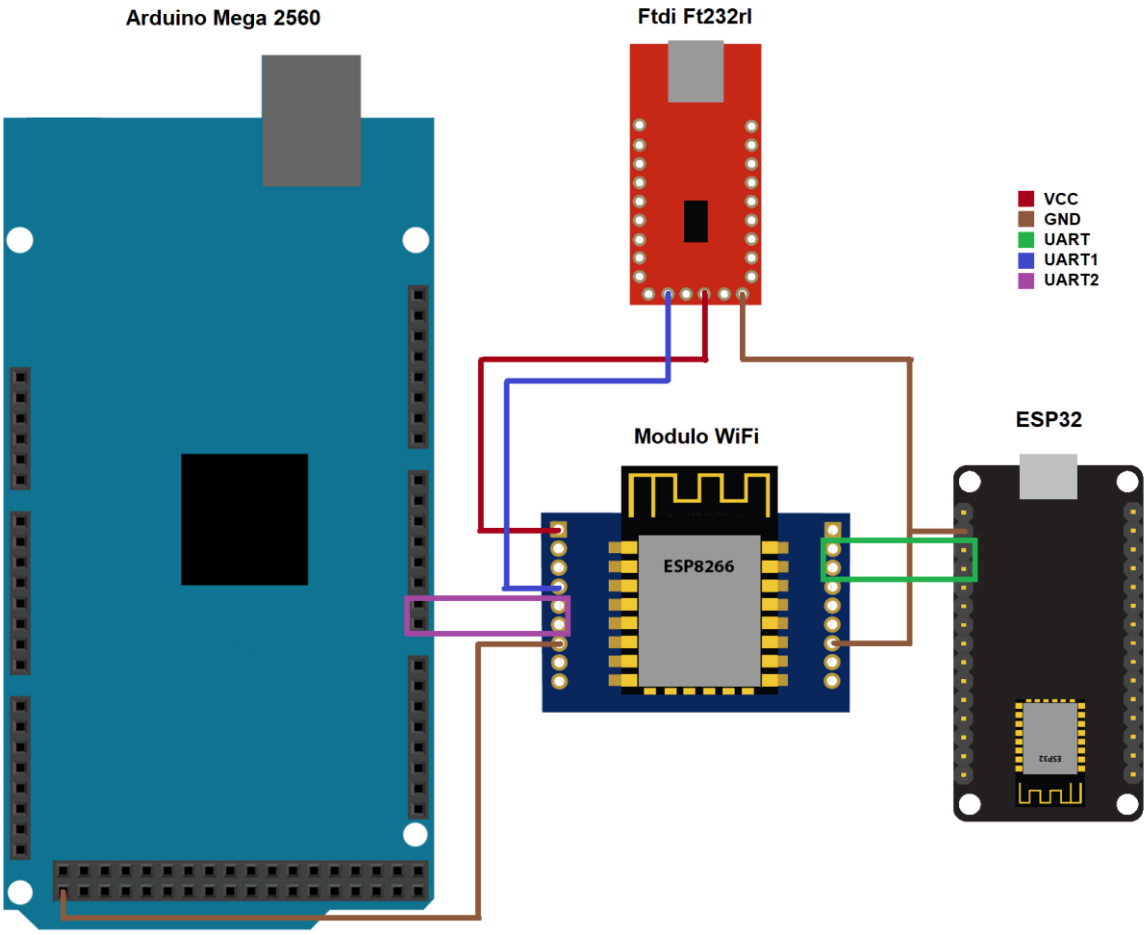
A comunicação pelo protocolo HTTP acontece diretamente entre o módulo e um servidor web, a partir de requisições HTTP. Dessa forma, foi utilizado o *host* "ptsv2.com", que é um servidor HTTPS grátis, disponível para prototipação utilizando requisições do tipo *post* e get. Nele foi criado um espaço chamado de "/testutfprtcc", exclusivamente para o presente trabalho, utilizando autenticação de usuário e senha. A interface principal do servidor pode ser visualidade na Figura 14.

Figura 16 - Servidor HTTP

Fonte: Autoria Própria (2022)

Por último, para a comunicação utilizando o protocolo UART, o dispositivo foi conectado diretamente a um módulo NodeMCU ESP32, utilizando um *baud rate* de 115200 símbolos por segundo, que para o caso do UART, significa 115200 *bits* por segundo. Também foram realizados testes utilizando uma placa Arduino Mega, a partir da conversão do nível lógico. Foi desenvolvido um código em linguagem C++ especialmente para a realização dos testes em ambas as plataformas supracitadas. Para a programação e o monitoramento do módulo, foi utilizado o conversor USB-TTL Ftdi Ft232rl. O setup montado, que foi utilizado para a realização dos testes, é apresentado na Figura 17.

Figura 17 - Esquema Realizado para Testes em Serial

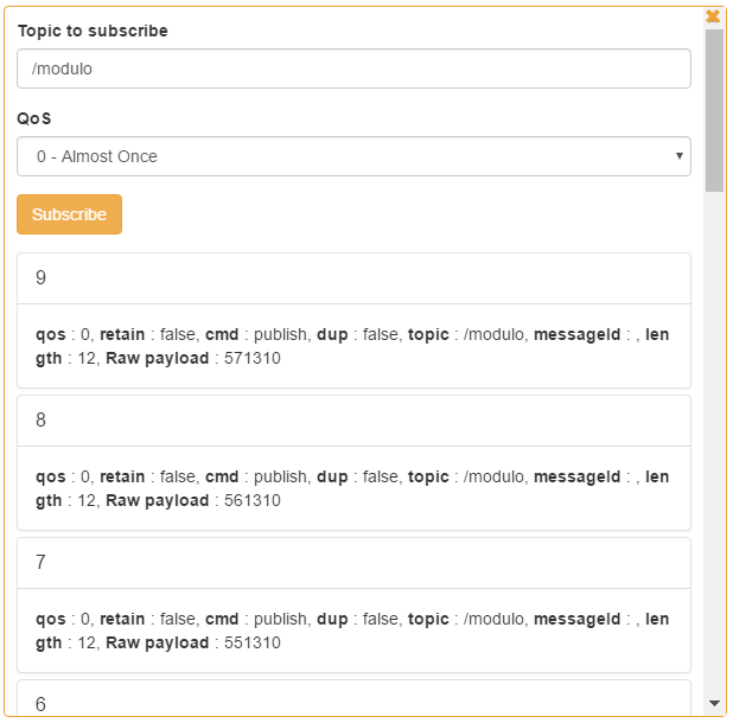


Fonte: Autoria Própria (2022)

Finalmente, já com o dispositivo conectado à internet e com os parâmetros para os 3 protocolos de comunicação totalmente configurados e prontos para uso, é possível realizar os devidos testes para atestar o transporte de dados dos mesmos. Sendo assim, foram definidas 3 séries de testes, representando os principais cenários de utilização do dispositivo, a fim de conseguir contornar todas as possíveis utilizações.

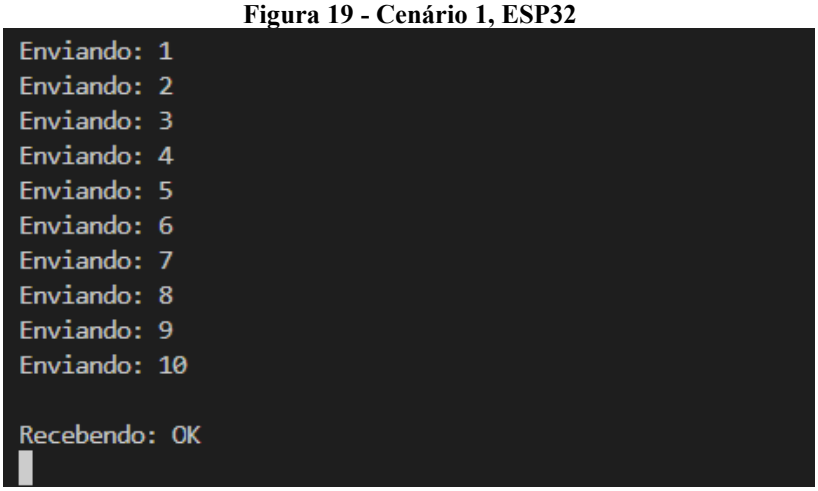
O primeiro cenário corresponde ao módulo trabalhando com os protocolos MQTT e UART, o que representa o funcionamento clássico de um dispositivo IOT. Dessa forma, o exercício realizado foi de simular o envio de dados para um servidor, representado na forma de uma série de 1 a 10, e, em contrapartida, o servidor (MQTTBox) enviando “OK” para sinalizar o recebimento dos dados. A Figura 15 apresenta os dados que chegaram ao MQTTBox e a Figura 16 o sinal de confirmação chegando ao microcontrolador externo.

Figura 18 - Cenário 1, MQTTBox



Fonte: Autoria Própria (2022)

Figura 19 - Cenário 1, ESP32

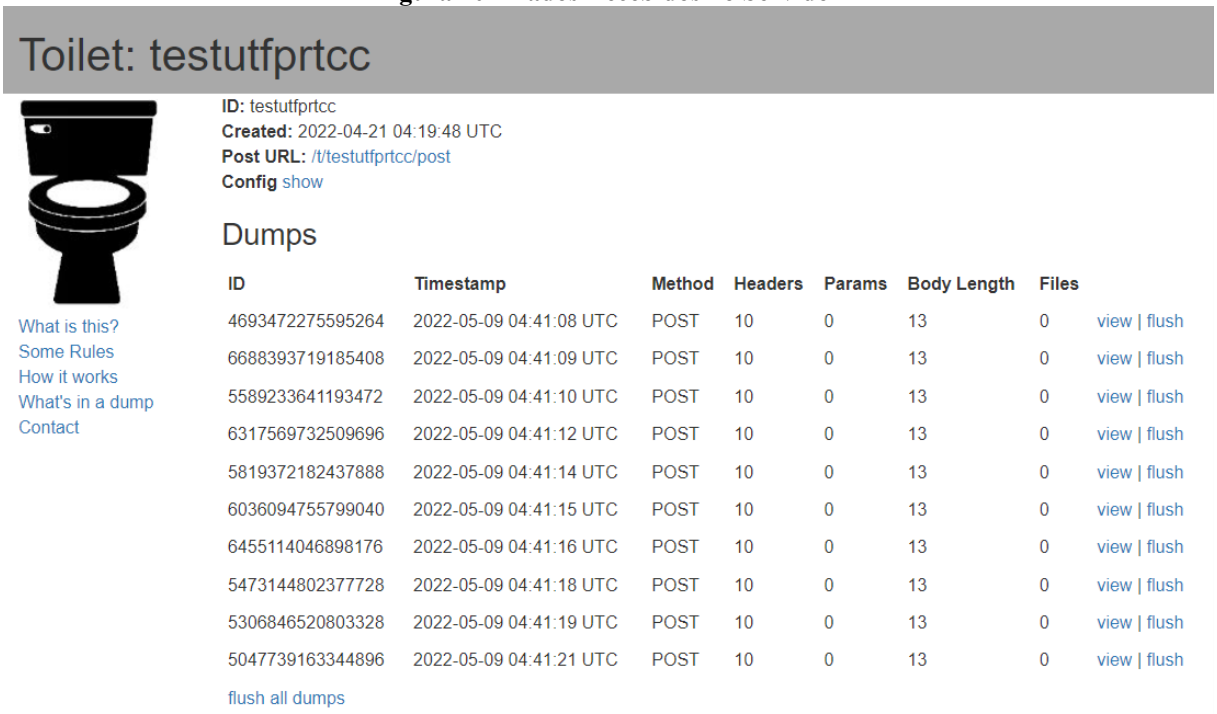


Fonte: Autoria Própria (2022)

O segundo cenário é o da utilização dos protocolos HTTP e UART, simulando como seria para enviar os dados diretamente a uma aplicação web, ou para serem armazenados em algum servidor. Uma pequena diferença dos demais casos está no fato de que para enviar dados a um servidor HTTP é necessário realizar uma requisição ao mesmo, e cada requisição, caso bem sucedida, apresenta um retorno, existindo a possibilidade de o servidor ou site em questão enviar dados diretamente ao módulo.

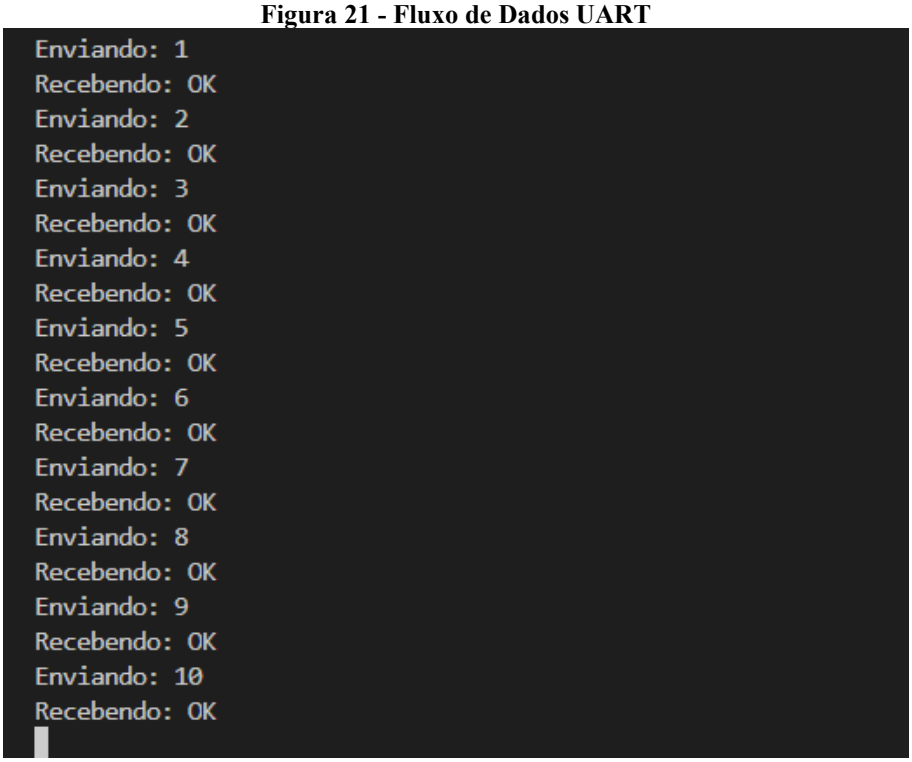
Nesse sentido, os dados recebidos do servidor, após a realização de um envio, são enviados de volta ao remetente anterior, elemento final para a composição do caso em questão. Para esse senário, o dispositivo continua enviando uma série de 1 a 10 (utilizando protocolo UART), entretanto, agora é enviado um sinal de “OK”, após o envio de cada dado. As Figuras 18 e 19 representam os dados que foram recebidos pelo servidor e pelo microcontrolador, respectivamente.

Figura 20 - Dados Recebidos no Servidor



Fonte: Autoria Própria (2022)

Figura 21 - Fluxo de Dados UART



Fonte: Autoria Própria (2022)

O último cenário, é para o caso de o usuário querer realizar o transporte de dados entre os protocolos MQTT e HTTP. Esse não representa o uso mais adequado para o dispositivo, pois existem maneiras mais práticas de fazer essa comunicação; de qualquer modo, caso o usuário queira usar o dispositivo dessa maneira, é totalmente possível. Nesse exercício, é enviado apenas uma checagem, onde é enviado pelo MQTT a mensagem “Status” e sendo retornado um sinal de “Disponível”; isso pode ser verificado nas Figuras 20, sendo apresentados à esquerda os dados enviados pelo MQTTBox, e, à direita, os dados que retornam do módulo, recebidos do servidor.

Figura 22 - Comunicação MQTT-HTTP

Fonte: Autoria Própria (2022)

Por fim, ainda existe a possibilidade de realizar a comunicação entre duas portas UART. Á primeira vista pode parecer desnecessário, visto que se trata do mesmo protocolo, mas, nesse caso, o usuário pode estar utilizando o módulo para conectar dois dispositivos incompatíveis, com taxas de comunicação diferentes, ou níveis lógicos distintos. Não foi mencionado anteriormente, mas esse é o único cenário, dentre os demais, em que é possível realizar as trocas de informação sem a necessidade de o módulo estar conectado à internet.

6 CONCLUSÃO

Foi produzido neste trabalho de conclusão de curso (TCC) um módulo Wi-Fi microcontrolado, utilizando o microcontrolador ESP-12S, capaz de realizar e viabilizar a troca e dados entre dispositivos e meios que utilizem os protocolos MQTT, HTTP e/ou UART.

Acredita-se que a principal funcionalidade do dispositivo, onde está sua principal importância, é o fato dele possibilitar a conversão de uma enorme variedade de equipamentos elétricos e eletrônicos simples, em produtos IOT, transformando, por exemplo, um mero interruptor eletrônico num interruptor inteligente, com a possibilidade de ser utilizado à distância e também de indicar o estado da iluminação local.

Também existem aplicações para outras áreas, como as de sensoriamento e monitoramento. É possível acoplar o módulo via UART a um microcontrolador RL78/G10 acoplado a um sensor de temperatura, para realizar à distância a medição da temperatura de algum ambiente desejado (RENESAS, 2018).

Além de todas essas funcionalidades, o dispositivo é bastante aplicável para a área da educação. Ele pode ser aplicado diretamente em sala de aula, possibilitando que quaisquer pessoas possam criar seus próprios projetos IOT, sem se preocupar com especificações de hardware e protocolos.

Uma boa melhoria a ser implantada ao projeto é a de adequá-lo a um paradigma de programação orientado a objetos. Isso torna o software embarcado mais adaptável, pois, na prática, significa que será possível adaptar o mesmo firmware para outros dispositivos, como o ESP32; isso apenas efetuando a modificação dos parâmetros do programa, e sem modificar a estrutura.

Aliado a isso, podem ser feitas muitas melhorias na página de configuração do dispositivo, podendo deixar ela bem mais completa e, ao mesmo tempo, fazer com que a sua interação com o usuário seja facilitada, tornando fatores como a velocidade de carregamento e disposição dos elementos visuais otimizados.

Por último, estuda-se a possibilidade de aplicar o código diretamente no chip ESP-07, ou algum outro. Dessa forma, elimina-se o sistema interno do chip como intermediário, e torna-se possível “conversar” diretamente com o microcontrolador, deixando a resposta dele bem melhor, diminuindo o consumo de memória e reduzindo o tempo de resposta do dispositivo.

REFERÊNCIAS

ALECRIM, Emerson. **O que é Wi-Fi?, conceito e versões**. [*S. l.*]: Info Wester, 2021. Disponível em: https://www.infowester.com/wifi.php. Acesso em: 16 jun. 2022.

BADESCU, Viorel; LAZAROIU, George C; BARELLI, Linda. **Power engineering: advances and challenges**. 1st ed. Boca Raton: CRC Press, 2018. E-book. Disponível em: https://www.taylorfrancis.com/books/edit/10.1201/9780429453717/power-engineering-viorel-badescu-george-cristian-lazaroiu-linda-barelli?refId=f7e81c65-b39a-45cb-9dbe-64b08d62ff44&context=ubx. Acesso em: 16 jun. 2022.

BARTNITSKY, Jan. **HTTP vs MQTT performance tests**. [*S. l.*]: Flespi, 2018. Disponível em: https://flespi.com/blog/http-vs-mqtt-performance-tests. Acesso em: 16 jun. 2022.

BERCHELE, Giuliane; CASSETTARI, Eder; FARIA, Giselma; SILVA, Diana; SILVA, Izabella. **A inteligência artificial na tomada de decisão do projeto e desenvolvimento do produto na indústria automobilística**. Maceió, AL: ENEGEP, 2018. Disponível em: http://www.abepro.org.br/biblIoTeca/TN_STP_262_506_36539.pdf. Acesso em: 16 jun. 2022.

BORBA, Gustavo. **Microcontroladores: noções gerais**. Curitiba, PR: UTFPR, 2017. Disponível em: http://paginapessoal.utfpr.edu.br/gustavobborba/material/files/mc_nocoesGerais.pdf. Acesso em: 16 jun. 2022.

CAMPBELL, Scott. **Basics of UART communication**. [*S. l.*]: Circuit Basics, 2016. Disponível em: https://www.circuitbasics.com/basics-uart-communication/. Acesso em: 16 jun. 2022.

CARMO, Sidney; COLOMBO, Luciane; FAVOTO, Thais. **A evolução da sociedade de consumo**. Umuarama, PR: Akrópólis, 2008. Disponível em: http://hilaineyaccoub.com.br/wp-content/uploads/2018/01/Luciane-Colombo-Thais-Brandt-Sidney-Carmo-A-evolu%C3%A7%C3%A3o-da-sociedade-de-consumo.pdf . Acesso em: 16 jun. 2022.

ESPRESSIF SYSTEMS. **ESP8266: technical reference**. Version 1.7. Shanghai, China: Espressif IoT Team, Espressif Systems, 2020a. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf. Acesso em: 16 jun. 2022.

ESPRESSIF SYSTEMS. **ESP8266EX: datasheet**. Version 6.6. Shanghai, China: Espressif IoT Team, Espressif Systems, 2020b. Disponível em: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. Acesso em: 16 jun. 2022.

ESPRESSIF SYSTEMS. **About Espressif. Shanghai**, China: Espressif Systems, 2021. Disponível em: https://www.espressif.com/en/company/about-us/who-we-are. Acesso em: 16 jun. 2022.

HALFACREE, Gareth. **The official raspberry pi beginner's guide: how to use your new computer**. [*S. l.*]: Raspberry Pi Press, 2020. E-book. Disponível em: https://magazines-attachments.raspberrypi.org/books/full_pdfs/000/000/038/original/BeginnersGuide-4thEd-Eng_v2.pdf. Acesso em: 16 jun. 2022.

HUAWEI. **What are the differences between enterprise Wi-Fi and home Wi-Fi?**. [*S. l.*]: Huawei, 2018. Disponível em: https://e.huawei.com/br/eblog/enterprise-networking/wifi6/What-the-difference-between-corporate-Wi-Fi-and-home-Wi-Fi. Acesso em: 16 jun. 2022.

IEEE 802.11. Rio de Janeiro, RJ: GTA UFRJ, 2016. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2016-1/16_1_2/saullo/trabalho_redes1/Referencias/IEEE%20802.11.pdf. Acesso em: 16 jun. 2022.

KODALI, Ravi. **MQTT based home automation system using ESP8266**. Warangal, India: Department of Elecronics and Communication Engineering, National Institute Of Technology, 2016. Disponível em: https://www.researchgate.net/publication/316448543_MQTT_based_ home_automation_system_using_ESP8266. Acesso em: 16 jun. 2022.

KOLBAN, Neil. **Kolban's book on ESP8266**. Texas, USA: Leanpub, 2016. E-book. Disponível em: https://leanpub.com/ESP8266_ESP32. Acesso em: 16 jun. 2022.

LATHI, Bhagawandas; DING, Zhi. **Modern Digital and Analog Communication Systems**. New York, USA: Oxford, 2010. E-book. Disponível em: https://edisciplinas.usp.br/pluginfile.php/5251120/mod_resource/content/1/B.%20P.%20Lathi%2C%20Zhi%20Ding%20-%20Modern%20Digital%20and%20Analog%20Communication%20Systems-Oxford%20University%20Press%20%282009%29.pdf. Acesso em: 16 jun. 2022.

MODULES. in: **ESP8266 community wiki**. [*S. l.*]: ESP8266 Community Wiki, 2021. Disponível em: https://www.esp8266.com/wiki/doku.php?id=esp8266-module-family. Acesso em: 16 jun. 2022.

NIELSEN, Henrik. et al**. Network performance effects of HTTP/l. 1, CSS 1, and PNG**. [*S. l.*]: World Wide Web Consortium; [*S. l.*]: Digital Equipment Corporation, 1997. Disponível em: https://dl.acm.org/doi/abs/10.1145/263105.263157. Acesso em: 16 jun. 2022.

OKI, Nobuo; MANTOVANI, Suely. **Microcontroladores - PIC**, TEEE I - Projeto de Robôs Móveis. Ilha Solteira, SP: Departamento de Engenharia Elétrica, Faculdade de Engenharia de Ilha Solteira, 2013. Disponível em: https://www.feis.unesp.br/Home/departamentos/engenhariaeletrica/microcontroladores-_pic-1.pdf. Acesso em: 16 jun. 2022.

OLIVEIRA, Ricardo. **Uso do microcontrolador ESP8266 para automação residencial**. Rio de Janeiro, RJ: Universidade Federal do Rio de Janeiro, 2017. Disponível em: http://repositorio.poli.ufrj.br/monografias/monopoli10019583.pdf. Acesso em: 16 jun. 2022.

OLIVEIRA, Sérgio. **Internet das coisas com ESP8266, Arduino e Raspberry** PI. 1.ed. São Paulo, SP: Novatec Editora, 2017. E-book. Disponível em: https://novatec.com.br/livros/IoT-com-esp8266-arduino-raspberry-2ed/. Acesso em: 16 jun. 2022.

PENIDO, Édilus; TRINDADE, Ronaldo. **Microcontroladores**. Ouro Preto, MG: Rede e-Tec Brasil, Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, 2013. E-book. Disponível em: https://www.passeidireto.com/arquivo/66280471/microcontroladores. Acesso em: 16 jun. 2022.

RENESAS. **RL78/G10: temperature sensor control using UART communication**. [*S. l.*]: Renesas, 2018. Disponível em: https://www.renesas.com/br/en/document/apn/rl78g10-

temperature-sensor-control-using-uart-communication-rev100?language=en. Acesso em: 16 jun. 2022.

SECURE W2. **Simplifying WPA2-enterprise and 802.1x**. [*S. l.*]: Secure W2, 2021. Disponível em: https://www.securew2.com/solutions/wpa2-enterprise-and-802-1x-simplified #ent2. Acesso em: 16 jun. 2022.

SEROZHENKO, Marina. **MQTT vs. HTTP, which one is the best for IoT?**. [*S. l.*]: Medium, 2017. Disponível em: https://medium.com/mqtt-buddy/mqtt-vs-http-which-one-is-the-best-for-IoT-c868169b3105. Acesso em: 16 jun. 2022.

STREIF, Rudolf. **Microcontroller versus System-on-Chip in embedded system designs**. [*S. l.*]: Ibeeto, 2019. Disponível em: https://www.ibeeto.com/wordpress/wp-content/uploads/2019/09/tc-mcsoc.pdf. Acesso em: 16 jun. 2022.

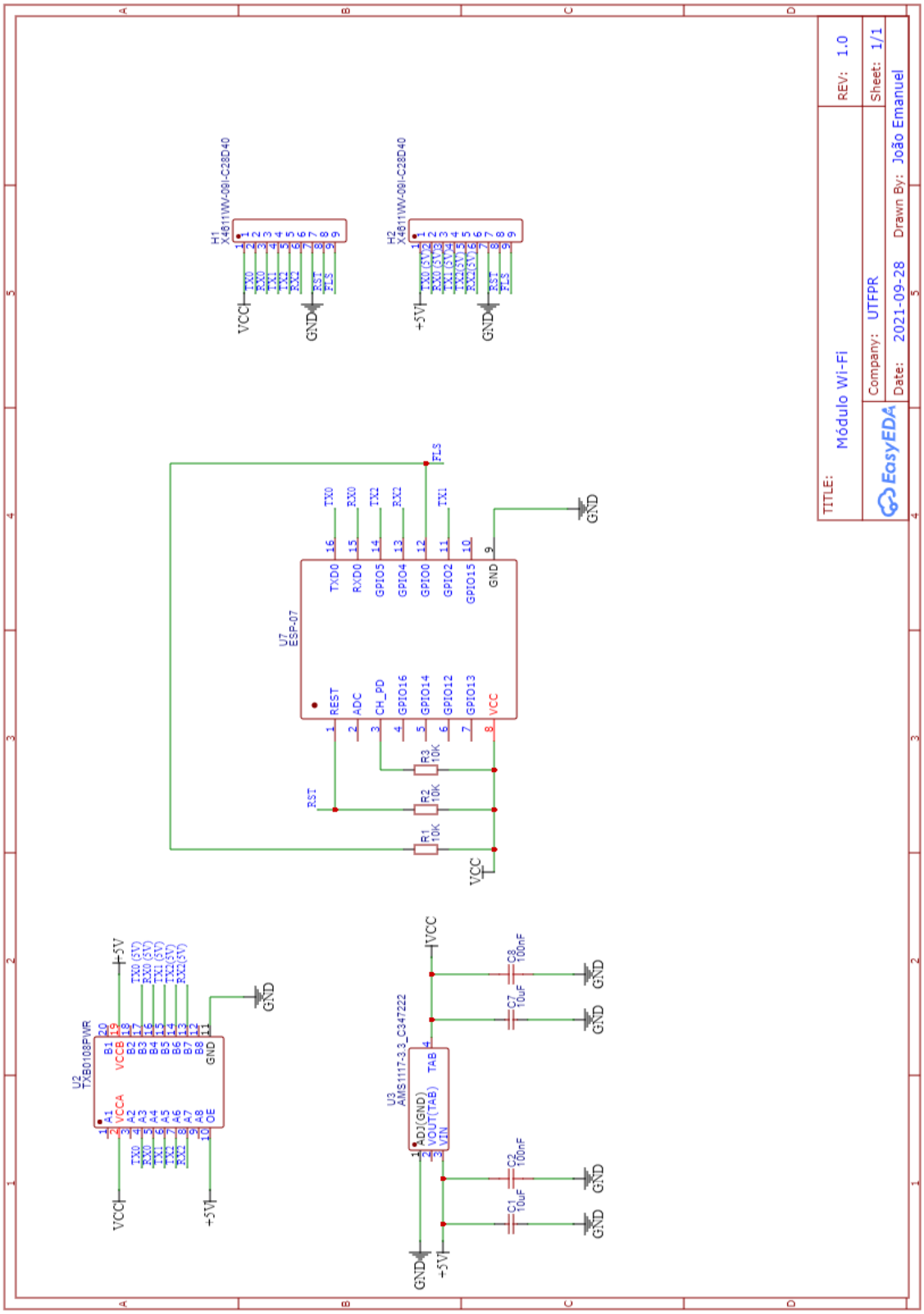
SOUZA, Marcelo. **Domótica de baixo custo usando princípios de IoT**. Natal, RN: Universidade Federal do Rio Grande Do Norte, Instituto Metrópole Digital, 2016. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/22029/1/MarceloVarelaDeSouza_DISSERT.pdf . Acesso em: 16 jun. 2022.

UART. in: **Transmissão e recepção serial assíncronas**. Versão 2014. São Paulo, SP: PCS USP, 2014. Disponível em http://www2.pcs.usp.br/~labdig/pdffiles_2014/UART.pdf. Acesso em: 16 jun. 2022.

WI-FI ALLIANCE. Security. [*S. l.*]: Wi-Fi Alliance, 2021. Disponível em: https://www.wi-fi.org/discover-wi-fi/security. Acesso em: 16 jun. 2022.

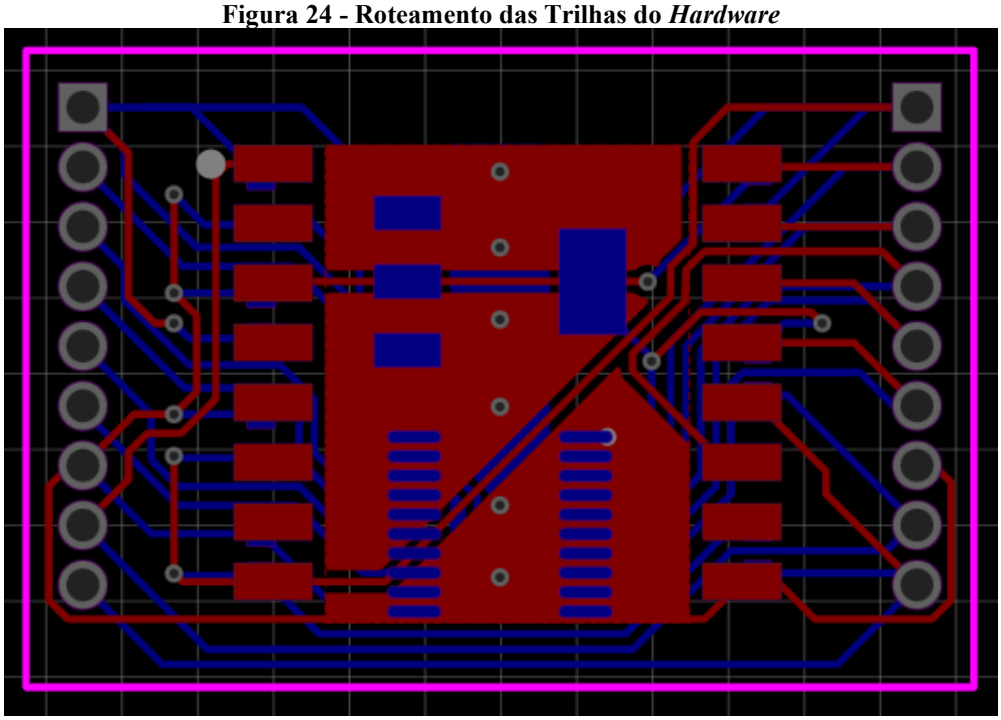
APÊNDICE A - DIAGRAMAS DE *HARDWARE*

Figura 23 - Diagrama Esquemático do *Hardware*



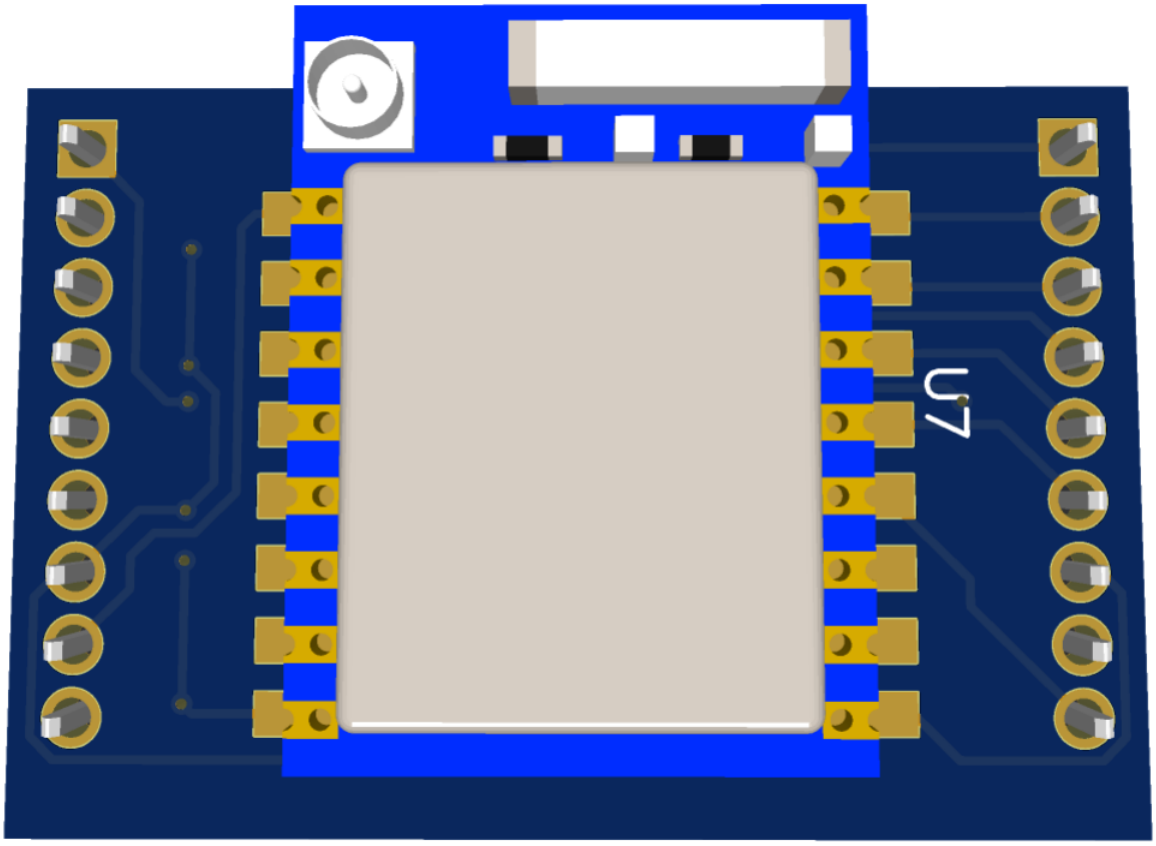
Fonte: Autoria Própria (2022)

Figura 24 - Roteamento das Trilhas do *Hardware*



Fonte: Autoria Própria (2022)

Figura 25 - Representação Tridimensional do *Hardware*



Fonte: Autoria Própria (2022)

APÊNDICE B - CÓDIGO FONTE

```
/*
This license allows you to remix, adapt and build upon the work, even for commercial purposes, as long as you credit the author(s) and license your new creations under identical terms. Content created by third parties, cited and referenced in this work is not covered by the license.
*/
#include <Arduino.h>
#include "ESP8266WiFi.h"
#include "ESP8266WebServer.h"
#include <PubSubClient.h>
#include "FS.h"
#include <ESP8266HTTPClient.h>
#include <SoftwareSerial.h>
#include <WiFiClient.h>
#include <ElegantOTA.h>

//Prototypes
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Network ------------------------------------------------------------------------------------------------------------------------------

void wifiConnect();

void wifiReconnect();

void configurationServerSetup();

void configurationServerLoop();

void handleIndexFile();

void handleWifiFile();

void handleMqttFile();

void handleHttpFile();

void handleUartFile();

void handleUart2File();

void handleSettingsFile();

void handleStyleFile();

void handleScriptFile();

void handleJqueryFile();

void handleSmartLogoFile();
```

```
void handleSandwichFile();
void handleSandwichNewFile();
void handleLogoMqttFile();
void handleLogoHttpFile();
void handleLogoUartFile();
void handleLogoUart2File();
void handleLogoMqttNewFile();
void handleLogoHttpNewFile();
void handleLogoUartNewFile();
void handleLogoUart2NewFile();
void handleArrowRightFile();
void handleArrowDownFile();
void restoreSystem();
void restartSystem();
void saveIndex();
void saveWifi();
void saveMqtt();
void saveHttp();
void saveUart();
void saveUart2();
void saveSettings();
void sendFeedback(String uri);
// Memory ------------------------------------------------------------------------------------------
-------------------------------------
void memoryBegin();
```

```
void factureSettings();

bool getMQTT_Flag();

void setMQTT_Flag(bool MQTTFlag);

bool getHTTP_Flag();

void setHTTP_Flag(bool HTTPFlag);

bool getUART_Flag();

void setUART_Flag(bool UARTFlag);

bool getUART2_Flag();

void setUART2_Flag(bool UART2Flag);

char* getAP_SSID();

void setAP_SSID(const char *apSsid);

char* getAP_PASS();

void setAP_PASS(const char *apPass);

char* getLocal_IP();

void setLocal_IP(const char *local_IP);

char* getGate_Way();

void setGate_Way(const char *gateway);

char* getNet_Mask();

void setNet_Mask(const char *netmask);

char* getWiFi_SSID();

void setWiFi_SSID(const char *wifiSSID);

char* getWiFi_Pass();

void setWiFi_Pass(const char *wifiPass);

char* getWiFi_Id();

void setWiFi_Id(const char *wifiId);
```

```
char* getWiFi_User();
void setWiFi_User(const char *wifiSecretId);
char* getMQTT_Server();
void setMQTT_Server(const char *mqttServer);
char* getMQTT_IP();
void setMQTT_IP(const char *mqttIP);
char* getMQTT_Id();
void setMQTT_Id(const char *mqttId);
char* getMQTT_User();
void setMQTT_User(const char *mqttUser);
char* getMQTT_Port();
void setMQTT_Port(const char *mqttPort);
char* getMQTT_Pass();
void setMQTT_Pass(const char *mqttPass);
char* getMQTT_PubTopic();
void setMQTT_PubTopic(const char *mqttWillTopic);
char* getMQTT_SubTopic();
void setMQTT_SubTopic(const char *mqttWillTopic);
char* getMQTT_WillTopic();
void setMQTT_WillTopic(const char *mqttWillTopic);
char* getMQTT_WillMessage();
void setMQTT_WillMessage(const char *mqttWillMessage);
bool getHTTP_SFlag();
void setHTTP_SFlag(bool httpSFlag);
char* getHTTP_HostServer();
```

```
void setHTTP_HostServer(const char *httpHostServer);

char* getHTTP_PostServer();

void setHTTP_PostServer(const char *httpPostServer);

char* getHTTP_RequestServer();

void setHTTP_RequestServer(const char *httpRequestServer);

char* getHTTP_User();

void setHTTP_User(const char *httpUser);

char* getHTTP_Pass();

void setHTTP_Pass(const char *httpPass);

char* getHTTP_Port();

void setHTTP_Port(const char *httpPort);

char* getHTTP_Freq();

void setHTTP_Freq(const char *httpFreq);

char* getUART_BaudRate();

void setUART_BaudRate(const char *uartBaudrate);

char* getUART2_BaudRate();

void setUART2_BaudRate(const char *uartWaittime);

void memorySave();

bool isWritten(char *c);

bool isWritten(const char *c);

bool isWritten(String c);

void debug();

// Converter ----------------------------------------------------------------------------------------
---------------------------------------

void MQTT_Callback(char* topic, byte* payload, unsigned int length);

void readMQTT();
```

```
void readHTTP();

void readUART();

void writeMQTT(String s);

void writeHTTP(String s);

void writeUART(String s);

void writeUART2(String s);

void readUART2();

void vectorizer(String dataS, uint8_t **Bytes);

uint8_t binConverter(char *dataC, bool flagArrange);

void dataConnect();

void MQTT_Connect();

void dataScan();

//                                                                                                                                                                Main
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void setup()
{
  Serial1.begin(115200);
  memoryBegin();
  wifiConnect();
  dataConnect();
  configurationServerSetup();
}

void loop()
{
  wifiReconnect();
  configurationServerLoop();
  dataScan();
}

//                                                                Functions                                                                Body
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////

// Network ---------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------
```

```
// Network Parameters

ESP8266WebServer server;

// Connect to the WiFi Network and create the Access Point
void wifiConnect() {
  unsigned long timePass;

  Serial1.print("Connecting to the WiFi: ");
  Serial1.print("SSID: ");
  Serial1.println(getWiFi_SSID());
  Serial1.print("Password:");
  Serial1.println(getWiFi_Pass());

  if (isWritten(getWiFi_SSID())) {
    Serial1.println("Domestic network.");

    WiFi.mode(WIFI_STA);
    WiFi.begin(getWiFi_SSID(), getWiFi_Pass());

    WiFi.printDiag(Serial1);

    timePass = millis();
    timePass += 5000;
    while ((WiFi.status() != WL_CONNECTED) && (millis() < timePass))
    {
      Serial1.print(".");
    }
    if (WiFi.status() == WL_CONNECTED) {
      Serial1.println("\nConnected to the WiFi network.");
      Serial1.print("IP address: ");
      Serial1.println(WiFi.localIP());
      WiFi.printDiag(Serial1);
    }
    else {
      Serial1.println("\nError: Unable to connect to WiFi network.");
    }
  }

  else Serial1.println("\nError: No Wi-Fi network defined.");

  IPAddress local_IP;
  IPAddress gateway;
  IPAddress netmask;
  (bool)local_IP.fromString(getLocal_IP());
  (bool)gateway.fromString(getGate_Way());
  (bool)netmask.fromString(getNet_Mask());

  WiFi.softAPConfig(local_IP, gateway, netmask);
```

```
  WiFi.softAP(getAP_SSID(), getAP_PASS());

  Serial1.println("\nAP Actived.");
  Serial1.print("SSID: ");
  Serial1.println(getAP_SSID());
  Serial1.print("PASS: ");
  Serial1.println(getAP_PASS());
  Serial1.print("Local IP: ");
  Serial1.println(local_IP);
  Serial1.print("Gateway: ");
  Serial1.println(gateway);
  Serial1.print("Netmask: ");
  Serial1.println(netmask);
}

// Reconnect to the WiFi Network in case lost connection
void wifiReconnect(){
  if (WiFi.status() != WL_CONNECTED && isWritten(getWiFi_SSID())){
    unsigned long timePass;

    WiFi.reconnect();

    timePass = millis();
    timePass += 5000;
    while ((WiFi.status() != WL_CONNECTED) && (millis() < timePass))
    {
      Serial1.print(".");
    }
    if (WiFi.status() == WL_CONNECTED) {
      Serial1.println("\nConnected to the WiFi network.");
      Serial1.print("IP address: ");
      Serial1.println(WiFi.localIP());
      WiFi.printDiag(Serial1);
    }
    else {
      Serial1.println("\nError: Unable to connect to WiFi network.");
    }
  }
}

// Handle web server requisitions
void configurationServerLoop(){
  server.handleClient();
}

// Initialize the web server
void configurationServerSetup(){

  server.on("/", handleIndexFile);
  server.on("/home", handleIndexFile);
```

```
  server.on("/wifi", handleWifiFile);
  server.on("/mqtt", handleMqttFile);
  server.on("/http", handleHttpFile);
  server.on("/uart", handleUartFile);
  server.on("/uart2", handleUart2File);
  server.on("/settings", handleSettingsFile);
  server.on("/css", handleStyleFile);
  server.on("/js", handleScriptFile);
  server.on("/jquery", handleJqueryFile);
  server.on("/logo/smart", handleSmartLogoFile);
  server.on("/img/sandwich", handleSandwichFile);
  server.on("/img/sandwich/novo", handleSandwichNewFile);
  server.on("/logo/mqtt", handleLogoMqttFile);
  server.on("/logo/http", handleLogoHttpFile);
  server.on("/logo/uart", handleLogoUartFile);
  server.on("/logo/uart2", handleLogoUart2File);
  server.on("/logo/mqtt/novo", handleLogoMqttNewFile);
  server.on("/logo/http/novo", handleLogoHttpNewFile);
  server.on("/logo/uart/novo", handleLogoUartNewFile);
  server.on("/logo/uart2/novo", handleLogoUart2NewFile);
  server.on("/flecha/direita", handleArrowRightFile);
  server.on("/flecha/baixo", handleArrowDownFile);
  server.on("/reset", restoreSystem);
  server.on("/restart", restartSystem);
  server.on("/submit/index", HTTP_POST, saveIndex);
  server.on("/submit/wifi", HTTP_POST, saveWifi);
  server.on("/submit/mqtt", HTTP_POST, saveMqtt);
  server.on("/submit/http", HTTP_POST, saveHttp);
  server.on("/submit/uart", HTTP_POST, saveUart);
  server.on("/submit/uart2", HTTP_POST, saveUart2);
  server.on("/submit/settings", HTTP_POST, saveSettings);

  ElegantOTA.begin(&server,"Smart Collect","senha");
  server.begin();
}

// The next 23 functions construct the web pages
void handleIndexFile()
{
  File file = SPIFFS.open("/index.html","r");
  server.streamFile(file, "text/html");
  file.close();
}

void handleWifiFile()
{
  File file = SPIFFS.open("/pagina-01.html","r");
  server.streamFile(file, "text/html");
  file.close();
}
```

```
void handleMqttFile()
{
 File file = SPIFFS.open("/pagina-02.html","r");
 server.streamFile(file, "text/html");
 file.close();
}

void handleHttpFile()
{
 File file = SPIFFS.open("/pagina-03.html","r");
 server.streamFile(file, "text/html");
 file.close();
}

void handleUartFile()
{
 File file = SPIFFS.open("/pagina-04.html","r");
 server.streamFile(file, "text/html");
 file.close();
}

void handleUart2File()
{
 File file = SPIFFS.open("/pagina-05.html","r");
 server.streamFile(file, "text/html");
 file.close();
}

void handleSettingsFile()
{
 File file = SPIFFS.open("/pagina-06.html","r");
 server.streamFile(file, "text/html");
 file.close();
}

void handleStyleFile()
{
 File file = SPIFFS.open("/main.css","r");
 server.streamFile(file, "text/css");
 file.close();
}

void handleScriptFile()
{
 File file = SPIFFS.open("/main.js","r");
 server.streamFile(file, "text/javascript");
 file.close();
}
```

```
void handleJqueryFile()
{
 File file = SPIFFS.open("/jquery-3.6.0.min.js","r");
 server.streamFile(file, "text/javascript");
 file.close();
}

void handleSmartLogoFile()
{
 File file = SPIFFS.open("/logo-smart.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleSandwichFile()
{
 File file = SPIFFS.open("/sandwich-menu-novo.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleSandwichNewFile()
{
 File file = SPIFFS.open("/sandwich-menu-novo.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleLogoMqttFile()
{
 File file = SPIFFS.open("/mqtt-icon.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleLogoHttpFile()
{
 File file = SPIFFS.open("/http-icon.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleLogoUartFile()
{
 File file = SPIFFS.open("/uart-icon.png","r");
 server.streamFile(file, "image/png");
 file.close();
}

void handleLogoUart2File()
```

```
{
  File file = SPIFFS.open("/uart2-icon.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleLogoMqttNewFile()
{
  File file = SPIFFS.open("/mqtt-icon-ver.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleLogoHttpNewFile()
{
  File file = SPIFFS.open("/http-icon-ver.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleLogoUartNewFile()
{
  File file = SPIFFS.open("/uart-icon-ver.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleLogoUart2NewFile()
{
  File file = SPIFFS.open("/uart2-icon-ver.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleArrowRightFile()
{
  File file = SPIFFS.open("/arrow-icon.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

void handleArrowDownFile()
{
  File file = SPIFFS.open("/arrow-down-icon.png","r");
  server.streamFile(file, "image/png");
  file.close();
}

// Restore the system to factory configurations
void restoreSystem()
```

```
{
  File file = SPIFFS.open("/pagina-07.html","r");
  server.streamFile(file, "text/html");
  file.close();

  factureSettings();
  Serial1.println("Restarting System...");
  delay(2000);
  memorySave();
  delay(50);
  ESP.restart();
}

// Restart the system
void restartSystem()
{
  File file = SPIFFS.open("/pagina-07.html","r");
  server.streamFile(file, "text/html");
  file.close();

  Serial1.println("Restarting System...");
  delay(2000);
  memorySave();
  delay(50);
  ESP.restart();
}

// The next 7 functions save the informations aquired from the web pages
void saveIndex()
{
  setMQTT_Flag(false);
  setHTTP_Flag(false);
  setUART_Flag(false);
  setUART2_Flag(false);


  if(server.arg("protocol_MQTT") == "1"){
    setMQTT_Flag(true);
  }

  if(server.arg("protocol_HTTP") == "1"){
    setHTTP_Flag(true);
  }

  if(server.arg("protocol_UART") == "1"){
    setUART_Flag(true);
  }

  if(server.arg("protocol_UART2") == "1"){
    setUART2_Flag(true);
```

```
  }

  sendFeedback("/home");
}

void saveWifi()
{
  setWiFi_SSID(server.arg("wifiSSID").c_str());
  setWiFi_Pass(server.arg("mqttPass").c_str());
  setWiFi_Id(server.arg("wifiId").c_str());
  setWiFi_User(server.arg("wifiUser").c_str());

  sendFeedback("/wifi");
}

void saveMqtt()
{
  setMQTT_Server(server.arg("mqttServer").c_str());
  setMQTT_Id(server.arg("mqttId").c_str());
  setMQTT_User(server.arg("mqttUser").c_str());
  setMQTT_Pass(server.arg("mqttPass").c_str());
  setMQTT_PubTopic(server.arg("mqttPubTopic").c_str());
  setMQTT_SubTopic(server.arg("mqttSubTopic").c_str());
  setMQTT_IP(server.arg("mqttIP").c_str());
  setMQTT_Port(server.arg("mqttPort").c_str());
  setMQTT_WillTopic(server.arg("mqttWillTopic").c_str());
  setMQTT_WillMessage(server.arg("mqttWillMessage").c_str());

  sendFeedback("/mqtt");
}

void saveHttp()
{
  setHTTP_HostServer(server.arg("httpHostServer").c_str());
  setHTTP_PostServer(server.arg("httpPostServer").c_str());
  setHTTP_RequestServer(server.arg("httpRequestServer").c_str());
  String data = server.arg("hidden_Switch");
  if (data[0] == '1') {
    setHTTP_SFlag(true);
  }
  if (data[0] == '2') {
    setHTTP_SFlag(false);
  }
  setHTTP_User(server.arg("httpUser").c_str());
  setHTTP_Pass(server.arg("httpPass").c_str());
  setHTTP_Freq(server.arg("httpFreq").c_str());
  setHTTP_Port(server.arg("httpPort").c_str());

  sendFeedback("/http");
}
```

```
void saveUart()
{
  setUART_BaudRate(server.arg("uartBaudrate").c_str());

  sendFeedback("/uart");
}

void saveUart2()
{
  setUART2_BaudRate(server.arg("uart2Baudrate").c_str());

  sendFeedback("/uart2");
}

void saveSettings()
{
  setAP_SSID(server.arg("apSsid").c_str());
  setAP_PASS(server.arg("apPass").c_str());
  setLocal_IP(server.arg("localIP").c_str());
  setGate_Way(server.arg("gateway").c_str());
  setNet_Mask(server.arg("netmask").c_str());

  sendFeedback("/settings");
}

// redirects the browser back to the previous web page after saving the data
void sendFeedback(String uri){
  String data = "<html><head><meta http-equiv=";
  data += "";
  data += "refresh";
  data += "";
  data += "content=";
  data += "";
  data += "0; URL=";
  data += uri;
  data += "";
  data += "/><script>function myFunction(){alert(";
  data += "";
  data += "Dados Salvos!";
  data += "";
  data += ");}</script></head><body onload=";
  data += "";
  data += "myFunction()";
  data += "";
  data += "></body></html>";

  server.send(200, "text/html", data);
}
```

```
// Memorý Functions ------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------

// Memorý Parameters

typedef struct {
  //Connection Parameters
  bool MQTTFlag;
  bool HTTPFlag;
  bool UARTFlag;
  bool UART2Flag;

  //Access Point Parameters
  char apSsid           [20] = "";
  char apPass           [20] = "";

  //Network Parameters
  char local_IP         [20] = "";
  char gateway          [20] = "";
  char netmask          [20] = "";

  //Station Parameters
  char wifiSSID         [20] = "";
  char wifiPass         [20] = "";
  char wifiId           [20] = "";
  char wifiUser         [20] = "";

  //MQTT Parameters
  char mqttServer       [20] = "";
  char mqttIP           [20] = "";
  char mqttId           [20] = "";
  char mqttUser         [20] = "";
  char mqttPass         [20] = "";
  char mqttPort         [20] = "";
  char mqttPubTopic     [20] = "";
  char mqttSubTopic     [20] = "";
  char mqttWillTopic    [20] = "";
  char mqttWillMessage  [20] = "";

  //HTTP Parameters
  bool httpSFlag;
  char httpHostServer   [100] = "";
  char httpPostServer   [100] = "";
  char httpRequestServer [100] = "";
  char httpUser         [20] = "";
  char httpPass         [20] = "";
  char httpPort         [20] = "";
  char httpFreq         [20] = "";

  //UART Parameters
```

```
  char uartBaudrate       [20] = "";

  //UART2 Parameters
  char uart2Baudrate      [20] = "";
} memory;

memory MemoryR;
memory MemoryW;

// Get the configuration parameters
void memoryBegin()
{
  bool s = SPIFFS.begin();

  if (s) {
    Serial1.println("Memory initialized correctly.");

    File myFile = SPIFFS.open("/database.txt", "r");

    if(myFile.size() > 0){
      Serial1.println("Previous settings restored.");
      myFile.read((byte *)&MemoryR, sizeof(MemoryR));
      MemoryW = MemoryR;
      myFile.close();
    }

    else {
      Serial1.println("Error: Missing previous settings.");
      Serial1.println("Initializing with factory settings.");
      myFile.close();
      factureSettings();
      MemoryR = MemoryW;
      File myFile = SPIFFS.open("/database.txt", "w");
      myFile.write((byte *)&MemoryW, sizeof(MemoryW));
      myFile.close();
    }
  }

  else {
    Serial1.println("Error: Unable to initialize memory.");
    exit( 0 );
  }
}

// Set the configuration parameters to facture default
void factureSettings()
{
  MemoryW.MQTTFlag = false;
  MemoryW.HTTPFlag = false;
  MemoryW.UARTFlag = false;
```

```
  MemoryW.UART2Flag = false;

  strncpy(MemoryW.apSsid         ,"SmartCollectDevice" ,20);  //é necessário colocar os ultimos dígitos do mac https://rntlab.com/question/how-can-i-get-Serial1-number-of-esp32-cam-hardware/ ESP.getEfuseMac();
  strncpy(MemoryW.apPass         ,""                   ,20);

  strncpy(MemoryW.local_IP       ,"192.168.11.1" ,20);
  strncpy(MemoryW.gateway        ,"192.168.11.1" ,20);
  strncpy(MemoryW.netmask        ,"255.255.255.0",20);

  strncpy(MemoryW.wifiSSID       ,"",20);
  strncpy(MemoryW.wifiPass       ,"",20);
  strncpy(MemoryW.wifiId         ,"",20);
  strncpy(MemoryW.wifiUser       ,"",20);

  strncpy(MemoryW.mqttServer     ,"test.mosquitto.org",20);
  strncpy(MemoryW.mqttIP         ,"192.168.1.1",20);
  strncpy(MemoryW.mqttId         ,"ESP",20);
  strncpy(MemoryW.mqttUser       ,"JJESP",20);
  strncpy(MemoryW.mqttPass       ,"1234",20);
  strncpy(MemoryW.mqttPort       ,"1880",20);
  strncpy(MemoryW.mqttPubTopic   ,"/modulo",20);
  strncpy(MemoryW.mqttSubTopic   ,"/box",20);
  strncpy(MemoryW.mqttWillTopic  ,"status",20);
  strncpy(MemoryW.mqttWillMessage  ,"Eu vou cair!!!!!!",20);

  MemoryW.httpSFlag == true;
  strncpy(MemoryW.httpHostServer   ,"ptsv2.com",100);
  strncpy(MemoryW.httpPostServer   ,"/t/testutfprtcc/post",100);
  strncpy(MemoryW.httpRequestServer,"/t/testutfprtcc/get",100);
  strncpy(MemoryW.httpUser       ,"",20);
  strncpy(MemoryW.httpPass       ,"",20);
  strncpy(MemoryW.httpPort       ,"80",20);
  strncpy(MemoryW.httpFreq       ,"0",20);

  strncpy(MemoryW.uartBaudrate   ,"115200",20);

  strncpy(MemoryW.uart2Baudrate  ,"115200",20);
}

// The next 66 functions get parametes from storage/reset the parameters stored
bool getMQTT_Flag(){
  return MemoryR.MQTTFlag;
}

void setMQTT_Flag(bool MQTTFlag){
  MemoryW.MQTTFlag = MQTTFlag;
}
```

```
bool getHTTP_Flag(){
  return MemoryR.HTTPFlag;
}

void setHTTP_Flag(bool HTTPFlag){
  MemoryW.HTTPFlag = HTTPFlag;
}

bool getUART_Flag(){
  return MemoryR.UARTFlag;
}

void setUART_Flag(bool UARTFlag){
  MemoryW.UARTFlag = UARTFlag;
}

bool getUART2_Flag(){
  return MemoryR.UART2Flag;
}

void setUART2_Flag(bool UART2Flag){
  MemoryW.UART2Flag = UART2Flag;
}

char* getAP_SSID(){
  return MemoryR.apSsid;
}

void setAP_SSID(const char *apSsid){
  strncpy(MemoryW.apSsid,apSsid,20);
}

char* getAP_PASS(){
  return MemoryR.apPass;
}

void setAP_PASS(const char *apPass){
  strncpy(MemoryW.apPass,apPass,20);
}

char* getLocal_IP(){
  return MemoryR.local_IP;
}

void setLocal_IP(const char *local_IP){
  strncpy(MemoryW.local_IP,local_IP,20);
}

char* getGate_Way(){
  return MemoryR.gateway;
```

```
}

void setGate_Way(const char *gateway){
  strncpy(MemoryW.gateway,gateway,20);
}

char* getNet_Mask(){
  return MemoryR.netmask;
}

void setNet_Mask(const char *netmask){
  strncpy(MemoryW.netmask,netmask,20);
}

char* getWiFi_SSID(){
  return MemoryR.wifiSSID;
}

void setWiFi_SSID(const char *wifiSSID){
  strncpy(MemoryW.wifiSSID,wifiSSID,20);
}

char* getWiFi_Pass(){
  return MemoryR.wifiPass;
}

void setWiFi_Pass(const char *wifiPass){
  strncpy(MemoryW.wifiPass,wifiPass,20);
}

char* getWiFi_Id(){
  return MemoryR.wifiId;
}

void setWiFi_Id(const char *wifiId){
  strncpy(MemoryW.wifiId,wifiId,20);
}

char* getWiFi_User(){
  return MemoryR.wifiUser;
}

void setWiFi_User(const char *wifiUser){
  strncpy(MemoryW.wifiUser,wifiUser,20);
}

char* getMQTT_Server(){
  return MemoryR.mqttServer;
}
```

```
void setMQTT_Server(const char *mqttServer){
  strncpy(MemoryW.mqttServer,mqttServer,20);
}

char* getMQTT_IP(){
  return MemoryR.mqttIP;
}

void setMQTT_IP(const char *mqttIP){
  strncpy(MemoryW.mqttIP,mqttIP,20);
}

char* getMQTT_Id(){
  return MemoryR.mqttId;
}

void setMQTT_Id(const char *mqttId){
  strncpy(MemoryW.mqttId,mqttId,20);
}

char* getMQTT_User(){
  return MemoryR.mqttUser;
}

void setMQTT_User(const char *mqttUser){
  strncpy(MemoryW.mqttUser,mqttUser,20);
}

char* getMQTT_Pass(){
  return MemoryR.mqttPass;
}

void setMQTT_Pass(const char *mqttPass){
  strncpy(MemoryW.mqttPass,mqttPass,20);
}

char* getMQTT_Port(){
  return MemoryR.mqttPort;
}

void setMQTT_Port(const char *mqttPort){
  strncpy(MemoryW.mqttPort,mqttPort,20);
}

char* getMQTT_PubTopic(){
  return MemoryR.mqttPubTopic;
}

void setMQTT_PubTopic(const char *mqttPubTopic){
  strncpy(MemoryW.mqttPubTopic,mqttPubTopic,20);
```

```
}

char* getMQTT_SubTopic(){
  return MemoryR.mqttSubTopic;
}

void setMQTT_SubTopic(const char *mqttSubTopic){
  strncpy(MemoryW.mqttSubTopic,mqttSubTopic,20);
}

char* getMQTT_WillTopic(){
  return MemoryR.mqttWillTopic;
}

void setMQTT_WillTopic(const char *mqttWillTopic){
  strncpy(MemoryW.mqttWillTopic,mqttWillTopic,20);
}

char* getMQTT_WillMessage(){
  return MemoryR.mqttWillMessage;
}

void setMQTT_WillMessage(const char *mqttWillMessage){
  strncpy(MemoryW.mqttWillMessage,mqttWillMessage,20);
}

bool getHTTP_SFlag(){
  return MemoryR.httpSFlag;
}

void setHTTP_SFlag(bool httpSFlag){
  MemoryW.httpSFlag = httpSFlag;
}

char* getHTTP_HostServer(){
return MemoryR.httpHostServer;
}

void setHTTP_HostServer(const char *httpHostServer){
  strncpy(MemoryW.httpHostServer,httpHostServer,100);
}

char* getHTTP_PostServer(){
return MemoryR.httpPostServer;
}

void setHTTP_PostServer(const char *httpPostServer){
  strncpy(MemoryW.httpPostServer,httpPostServer,100);
}
```

```
char* getHTTP_RequestServer(){
return MemoryR.httpRequestServer;
}

void setHTTP_RequestServer(const char *httpRequestServer){
  strncpy(MemoryW.httpRequestServer,httpRequestServer,100);
}

char* getHTTP_User(){
return MemoryR.httpUser;
}

void setHTTP_User(const char *httpUser){
  strncpy(MemoryW.httpUser,httpUser,20);
}

char* getHTTP_Pass(){
return MemoryR.httpPass;
}

void setHTTP_Pass(const char *httpPass){
  strncpy(MemoryW.httpPass,httpPass,20);
}

char* getHTTP_Port(){
return MemoryR.httpPort;
}

void setHTTP_Port(const char *httpPort){
  strncpy(MemoryW.httpPort,httpPort,20);
}

char* getHTTP_Freq(){
return MemoryR.httpFreq;
}

void setHTTP_Freq(const char *httpFreq){
  strncpy(MemoryW.httpFreq,httpFreq,20);
}

char* getUART_BaudRate(){
return MemoryR.uartBaudrate;
}

void setUART_BaudRate(const char *uartBaudrate){
  strncpy(MemoryW.uartBaudrate,uartBaudrate,20);
}

char* getUART2_BaudRate(){
return MemoryR.uart2Baudrate;
```

```
}

void setUART2_BaudRate(const char *uart2Baudrate){
  strncpy(MemoryW.uart2Baudrate,uart2Baudrate,20);
}

// Save the new parameteres in the flash memory
void memorySave()
{
  File myFile = SPIFFS.open("/database.txt", "w");
  myFile.write((byte *)&MemoryW, sizeof(MemoryW));
  myFile.close();
}

// The next 3 functions check it the given pointer contains data
bool isWritten(char *data){
  if ((data != NULL) && (data[0] == '\0')) {
    return false;
  }
  return true;
}

bool isWritten(const char *data){
  if ((data != NULL) && (data[0] == '\0')) {
    return false;
  }
  return true;
}

bool isWritten(String data){
  if ((data != NULL) && (data[0] == '\0')) {
    return false;
  }
  return true;
}

// Converter Functions ----------------------------------------------------------------------------
----------------------------------------------------------------------------------------

//MQTT Parameters
WiFiClient espClient;
PubSubClient client(espClient);
IPAddress mqtt_IP;
bool x = mqtt_IP.fromString(getMQTT_IP());
const char* id = getMQTT_Id();
const char* user = getMQTT_User();
const char* pass = getMQTT_Pass();
const char* willTopic = getMQTT_WillTopic();
const char* pubTopic = getMQTT_PubTopic();
const char* subTopic = getMQTT_SubTopic();
```

```
uint8_t willQos = 1;
boolean willRetain = true;
const char* willMessage = getMQTT_WillMessage();
boolean cleanSession = false;

//HTTP Parameters
WiFiClient httpClient;
HTTPClient http;
char *tst;
float frequency;
double period;
unsigned long past_time = 0;

//UART Parameters
long baud_rate = atol(getUART_BaudRate());

// UART2 Parameters
SoftwareSerial Serial2;
long baud_rate2 = atol(getUART2_BaudRate());

// Function called once MQTT receives data
void MQTT_Callback(char* topic, byte* payload, unsigned int length) {
  String s = "";
  for (size_t i = 0; i < length; i++) {
    s += (char)payload[i];
  }

  writeHTTP(s);
  writeUART(s);
  writeUART2(s);
}

// Reads MQTT inputs
void readMQTT(){
  if(getMQTT_Flag()){
    if (!client.connected()) {
      MQTT_Connect();
    }
    else{
      client.loop();
    }
  }
}

// Reads HTTP inputs
void readHTTP(){
  if(getHTTP_Flag()){
    if (micros() - past_time > period) {
      String serverHost = String(getHTTP_HostServer());
      String serverURI = String(getHTTP_RequestServer());
```

```
      bool flagHTTPS = getHTTP_SFlag();

      http.setURL(serverURI);
      int httpResponseCode = http.GET();
      Serial1.println(httpResponseCode);
      Serial1.printf("%s\n", http.errorToString(httpResponseCode).c_str());

      if (httpResponseCode <= 0) {
        Serial1.println("Error: Connection lost. Reconecting...");
        http.end();
        uint16_t port = 80;
        httpResponseCode = http.begin(httpClient, serverHost, port, serverURI, flagHTTPS);
      }

      if (httpResponseCode>0) {
        String s = http.getString();
        if(isWritten(s)){
          writeMQTT(s);
          writeUART(s);
          writeUART2(s);
        }
      }

      else {
        Serial1.println("Error: Impossible do reconect");
      }

      past_time = micros();
    }
  }
}

// Reads UART inputs
void readUART(){
  if(getUART_Flag()){
    String s;

    while (Serial.available())
    {
      char c = Serial.read(); // Receive a single character from the software serial port
      s += c;
    }
    writeMQTT(s);
    writeHTTP(s);
    writeUART2(s);
  }
}

// Reads UART2 inputs
void readUART2(){
```

```
  if(getUART2_Flag()){
    String s;

    while (Serial2.available())
    {
      char c = Serial2.read(); // Receive a single character from the software serial port
      s += c;
    }
    writeMQTT(s);
    writeHTTP(s);
    writeUART(s);
  }
}

// Sends MQTT outputs
void writeMQTT(String s){
  if(getMQTT_Flag()){
    client.publish(pubTopic, s.c_str());
  }
}

// Sends HTTP outputs
void writeHTTP(String s){
  if(getHTTP_Flag()){
    String serverHost = String(getHTTP_HostServer());
    String serverURI = String(getHTTP_PostServer());
    bool flagHTTPS = getHTTP_SFlag();

    http.setURL(serverURI);
    int httpResponseCode = http.POST("Hello, World!");
    Serial1.println(httpResponseCode);

    if (httpResponseCode <= 0) {
      Serial1.println("Error: Connection lost. Reconecting...");
      Serial1.printf("[HTTP] POST... failed, error: %s\n",
http.errorToString(httpResponseCode).c_str());
      http.end();
      uint16_t port = 80;
      httpResponseCode = http.begin(httpClient, serverHost, port, serverURI, flagHTTPS);
    }

    if (httpResponseCode>0) {
      String payload = http.getString();
      if(isWritten(payload)){
        writeMQTT(payload);
        writeUART(payload);
        writeUART2(payload);
      }
    }
```

```
    else {
      Serial1.println("Error: Impossible do reconect");
    }
  }
}

// Sends UART outputs
void writeUART(String s){
  if(getUART_Flag()){
    uint8_t u[s.length()];
    s.toCharArray((char*)u,s.length());

    Serial1.write(u,s.length());
  }
}

// Sends UART2 outputs
void writeUART2(String s){

  if(getUART2_Flag()){
    uint8_t u[s.length()];
    s.toCharArray((char*)u,s.length());

    Serial2.write(u,s.length());
  }
}

// Initialize the given protocols
void dataConnect(){
  if(getMQTT_Flag()){
    client.setServer(getMQTT_Server(), 1883);
    client.setCallback(MQTT_Callback);
  }
  if(getHTTP_Flag()){
    String serverHost = String(getHTTP_HostServer());
    String serverURI = String(getHTTP_PostServer());
    const char* HttpUser = getHTTP_User();
    const char* HttpPass = getHTTP_Pass();
    bool flagHttpUser;
    flagHttpUser = isWritten(HttpUser);
    bool flagHTTPS = getHTTP_SFlag();
    http.setReuse(true);
    uint16_t port = atoi(getHTTP_Port());
    http.begin(httpClient, serverHost, port, "/", flagHTTPS);
    if(flagHttpUser) {
      http.setAuthorization(HttpUser, HttpPass);
    }
    http.addHeader("Content-Type", "text/plain");

    tst = getHTTP_Freq();
```

```
    frequency = atof(tst);
    period = ((1/frequency)*1000000);
   }
   if(getUART_Flag()){
    Serial.begin(baud_rate);
   }
   if(getUART2_Flag()){
    Serial2.begin(9600,SWSERIAL_8N1,5,4);
    if (!Serial2) { // If the object did not initialize, then its configuration is invalid
     Serial1.println("Error: Unable to initialize UART2");
    }
   }
   if(getMQTT_Flag() == false && getMQTT_Flag() == false && getMQTT_Flag() == false
&& getMQTT_Flag() == false) {
    Serial1.println("Error: No protocol difined.");
   }
}

// Connect to the MQTT Network
void MQTT_Connect() {
  int connectFlag = 4;

  bool flagMqttId;
  bool flagMqttUser;
  bool flagMqttPass;
  bool flagMqttWillTopic;
  bool flagMqttWillMessage;
  flagMqttId = isWritten(id);
  flagMqttUser = isWritten(user);
  flagMqttPass = isWritten(pass);
  flagMqttWillTopic = isWritten(willTopic);
  flagMqttWillMessage = isWritten(willMessage);

  if (flagMqttWillTopic && flagMqttWillMessage){
    if (flagMqttId && flagMqttUser && flagMqttPass) {
      connectFlag = 3;
    }
    else if (flagMqttId) {
      connectFlag = 2;
    }
    else {
      connectFlag = 4;
    }
  }

  else if (flagMqttId && flagMqttUser && flagMqttPass) {
    connectFlag = 1;
  }

  else if (flagMqttId) {
```

```
  connectFlag = 0;
 }

 else {
  connectFlag = 4;
 }

 switch (connectFlag)
 {
  case /* id */ 0:
   if (client.connect(id)) {          //executando conexão
    Serial1.println("Connected to MQTT broker.");
    client.subscribe(subTopic);
   }
   else {
    Serial1.println("Error: Unable to connected to MQTT broker.");
    Serial1.print("RC = ");
    Serial1.println(client.state());
   }
   break;

  case /* id+usuário+senha */ 1:
   if (client.connect(id, user, pass)) {          //executando conexão
    Serial1.println("Connected to MQTT broker.");
    client.subscribe(subTopic);
   }
   else {
    Serial1.println("Error: Unable to connected to MQTT broker.");
    Serial1.print("RC = ");
    Serial1.println(client.state());
   }
   break;

  case /* id+will */ 2:
   if (client.connect(id, willTopic, willQos, willRetain, willMessage)) {          //executando
conexão
    Serial1.println("Connected to MQTT broker.");
    client.subscribe(subTopic);
   }
   else {
    Serial1.println("Error: Unable to connected to MQTT broker.");
    Serial1.print("RC = ");
    Serial1.println(client.state());
   }
   break;

  case /* id+usuário+senha+will */ 3:
   if (client.connect(id, user, pass, willTopic, willQos, willRetain, willMessage)) {
    Serial1.println("Connected to MQTT broker.");
    client.subscribe(subTopic);
```

```
      }
      else {
        Serial1.println("Error: Unable to connected to MQTT broker.");
        Serial1.print("RC = ");
        Serial1.println(client.state());
      }
      break;

    case /* Erro */ 4:
      Serial1.println("Error: MQTT parameters set incorrectly");
      break;

    default:
      break;
  }
}

// Verifies if there is incoming data
void dataScan(){
  readMQTT();

  readHTTP();

  readUART();

  readUART2();
}
```