UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

COLEGIADO DO CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Minini Dardengo

Integrando o Node-RED no ambiente de experimentação OTALab

Vitória, ES

2023

Gabriel Minini Dardengo

Integrando o Node-RED no ambiente de experimentação OTALab

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Colegiado do Curso de Ciência da Computação

Orientador: Prof. Dr. Vinícius Fernandes Soares Mota

Vitória, ES

2023

Gabriel Minini Dardengo

Integrando o Node-RED no ambiente de experimentação OTALab

Monografia apresentada ao Curso de Ciência da Computação do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

Trabalho aprovado. Vitória, ES, 3 de fevereiro de 2023:

Vinícius Fernandes Soares Mota
Orientador

Rodrigo Laiola Guimarães
UFES

José Gonçalves Pereira Filho
UFES

Vitória, ES
2023

Agradecimentos

Agradeço primeiramente aos meus pais, por me incentivar a fazer o curso de Ciências da Computação, e prestarem apoio nos momentos mais difíceis durante a minha graduação. Por isso, sou muitíssimo grato por ajudarem a realizar o meu sonho.

Ainda, presto agradecimento ao meu professor e orientador Dsc. Vinícius Fernandes Soares Mota, que foi bastante compreensivo diante as situações difíceis, e objetivo quanto ao compartilhamento de toda a experiência necessária para a realização deste trabalho. Sempre lembrarei de seu apoio e ensinamentos.

Por fim, agradeço aos amigos que fiz durante minha vida na universidade, que me ajudaram a vencer os obstáculos da graduação e, devido a isso, serão do meu círculo de amizade para a vida toda.

Resumo

A Internet das Coisas (IoT) é uma tecnologia que está se tornando cada vez mais comum, mas há um desafio relacionado à necessidade de atualizar e escalar dispositivos de IoT remotamente, seja através de uma conexão com a internet ou por uma conexão Wi-Fi local sem acesso à internet. É crucial que um dispositivo possa receber atualizações críticas, como correções de vulnerabilidades, *Over-the-air* (OTA), sem precisar estar conectado a um cabo USB. Para abordar esse problema, foi criado o OTALab, um projeto de pesquisa científica realizado na UFES, que sugere a criação de um *testbed* simples, rápido e de fácil configuração para estudos de aplicações de IoT. O OTALab é uma ferramenta para criar e implementar ambientes para o estudo de protocolos e aplicações IoT em dispositivos com microcontroladores, permitindo o upload remoto de códigos. Ele foi projetado para funcionar como uma API REST, oferecendo várias funções para o acesso à ferramenta, e conteinerizado em um *backend* chamado de OTALabBackEnd. O projeto OTALab tem como foco o desenvolvimento do OTALabBackEnd, mas a interface de *frontend*, chamada de OTALabFrontEnd, ainda não foi implementada. Para resolver isso, esse trabalho propõe integrar o Node-RED ao ambiente de experimentação OTALab, para adicionar uma nova camada ao projeto, que funcione como uma camada de interpretação dos dados da API e como uma camada de *frontend*. Para isso, foram utilizadas algumas paletas de nós fornecidas pela biblioteca do Node-RED sendo criados dois contêineres Node-RED para separar os perfis de usuários. Finalmente, foram realizados alguns testes de desempenho da aplicação.

Palavras-chave: OTALab; Node-RED; Internet das Coisas(IoT); API; backend; frontend; WI-Fi; OTALabBackEnd; OTALabFrontEnd.

Lista de ilustrações

Figura 1 – ESP8266 NodeMcu . . . 18
Figura 2 – ESP32 DevKit V1 . . . 18
Figura 3 – Visão geral da arquitetura do OTALab Cussuol e Mota (2022). . . . 21
Figura 4 – Schema do OTALabDB Cussuol e Mota (2022). . . . 23
Figura 5 – Exemplo de um design básico em FBP, Morrison (1994). . . . 26
Figura 6 – Interface de edição do Node-RED, Hagino e O'Lary (2021). . . . 27
Figura 7 – Visão geral da arquitetura proposta para implantação do Node-RED no OTALab . . . 31
Figura 8 – Telas de login para a *dashboard-ui* na parte superior e *flow-editor* na parte inferior . . . 32
Figura 9 – Tela inicial no *dashboard-ui* do Administrador . . . 33
Figura 10 – Todos os *subflows* criados para o perfil Administrador . . . 36
Figura 11 – Exemplo de 3 (três) *flows*, do *dispositivo-controller*, do *configuração-controler*, e do *conexao-controller* criados para o perfil Administrador . 36
Figura 12 – Tela inicial no *dashboard-ui* do Experimentador . . . 37
Figura 13 – Tela *dashboard-ui* do Experimentador ilustrando o funcionamento do botão Serviços. . . . 38
Figura 14 – Tela do *flwo-editor* com todas as funções usadas pelo perfil Experimentador 40
Figura 15 – Visão geral do esquema montado para os testes. . . . 43
Figura 16 – Gráfico referente a Tabela 1. . . . 44
Figura 17 – Gráfico referente a Tabela 2. . . . 45
Figura 18 – Gráfico referente a Tabela 3. . . . 46
Figura 19 – Gráfico referente a Tabela 4. . . . 47
Figura 20 – Gráfico referente a Tabela 5. . . . 48
Figura 21 – Gráfico referente a Tabela 6. . . . 49
Figura 22 – Gráfico referente a Tabela 7. . . . 50
Figura 23 – Gráfico referente a Tabela 8. . . . 51
Figura 24 – Gráfico referente a Tabela 9. . . . 52
Figura 25 – Erro NO ANSWER ao tentar executar o *script* Python spota.py . . . 54
Figura 26 – *Logs* do *dmesg* onde o *driver brltty* para Braile "sequestra"a entrada ttyUSB0 . . . 54
Figura 27 – Lista de *endpoints* disponibilizados pela API do OTALabBackEnd . . . 59

Lista de tabelas

Tabela 1 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com DOIS ESPs Ativos. Usando um *Borker* Mosquitto Local. 44
Tabela 2 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com DOIS ESPs Ativos. Usando um *Borker* público HiveMQ "broker.mqtt-dashboard.com". 45
Tabela 3 – Tabela de tempo gasto em segundos, para Acender e Apagar o LED dos ESPs, com TRÊS ESPs Ativos. Usando um *Borker* Mosquitto Local. 46
Tabela 4 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com TRÊS ESPs Ativos. Usando um *Borker* público HiveMQ "broker.mqtt-dashboard.com". 46
Tabela 5 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com QUATRO ESPs Ativos. Usando um *Borker* Mosquitto Local. 48
Tabela 6 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com QUATRO ESPs Ativos. Usando um *Borker* publico HiveMQ "broker.mqtt-dashboard.com". 48
Tabela 7 – Tempo em segundos para fazer *upload* do código do Telegram usando o OTALab e o Arduino IDE. 50
Tabela 8 – Tempo em segundos para fazer *upload* do código de Temperatura usando o OTALab e o Arduino IDE. 50
Tabela 9 – Tempo em segundos para cadastrar um dispositivo no OTALab. 51

Lista de Códigos

2.1 Templete Arduino a ser usado. 24
3.1 *Template* usado para o botão de atualizar Dispositivos. 41
1 Código da aplicação para acender e apagar LED do ESP pelo Telegram. . . 60
2 Código da aplicação de monitoramento de temperatura com sensor BMP280. 61

Lista de abreviaturas e siglas

API *Application programming interface*. 13, 28, 29

CLI *Command Line Interface*. 21

CPU *Central Processing Unit*. 19

FBP *Flow base programing*. 12, 24–26

GUI *Grafical user interface*. 13

HTTP *Hypertext Transfer Protocol*. 9, 13, 23

I/O *Input/Output*. 19

IoT *Internet of things*. 12, 16, 20, 21

MIT *Massachusetts Institute of Technology*. 16

MQTT *Message Queue Telemetry Transport*. 23, 26, 34

OTA *Over the air*. 12, 13, 17, 21, 53

PAAS *Platform-as-a-Service*. 20

PoC *Prove of Concept*. 26

SSID *Service set Identifier*. 12, 22, 34

UFES Universidade Federal do Espírito Santo. 12, 21

URL *Uniform Resource Locator*. 13

USB *Universal Serial Bus*. 17

WEB *World Wide Web*. 13

Wi-Fi *Wireless Field*. 12, 17

Glossário

backEnd se relaciona com o que está por trás das aplicações desenvolvidas na programação. 20

firmware é um conjunto de instruções operacionais que são programadas diretamente no hardware. 12, 21, 29

frontEnd é uma interface gráfica de uma software ou site. 20, 29

hardware é todo componente físico, interno ou externo de um equipamento. 12, 25, 28

host é um computador ou outro dispositivo conectado a uma rede de computadores. 13, 19

http request é uma solicitação HTTP feita por um cliente, para um host. 13

Internet of Things em português Internet das coisas. 16

kernel parte central de um sistema operacional. 19, 20

namespaces é um conjunto de sinais (nomes) que são usados para identificar e se referir a objetos de vários tipos. 20

software é um programa de computador. 12, 25, 27

testbeds Ambiente de experimentação. 13, 20

updates Atualizações. 17

upload Envio de arquivos pelo computador. 21, 29

Sumário

Lista de ilustrações 5
Lista de tabelas 6
Lista de Códigos 7
Lista de abreviaturas e siglas 8
Glossário 9

1 INTRODUÇÃO 12
1.1 Motivação e Justificativa 13
1.2 Objetivos 14
1.2.1 Objetivo Geral 14
1.2.2 Objetivo Específicos 14
1.3 Método de Desenvolvimento do Trabalho 14
1.4 Contribuições 15
1.5 Organização da Monografia 15

2 REFERENCIAL TEÓRICO E TECNOLOGIAS UTILIZADAS 16
2.1 Internet das coisas (IoT) 16
2.1.1 Paradigma Over-the-Air(OTA) 17
2.1.2 Dispositivos: ESP8266 e ESP32 17
2.2 Docker Contêiner 19
2.2.1 Visão geral 19
2.3 Ambiente de experimentação 20
2.3.1 Visão geral 20
2.3.2 Ambiente OTALab 21
2.4 Programação em fluxo 24
2.4.1 Visão geral 25
2.4.2 Node-red 26

3 ARQUITETURA PROPOSTA 29
3.1 Perfil Administrador 32
3.1.1 Flow-editor do Administrador 35
3.2 Perfil Experimentador 37
3.2.1 Flow-editor do Experimentador 39

4 ESTUDO DE CASOS 42
4.1 Metodologia 42
4.2 Resultados 43

5 CONCLUSÃO 53
5.1 Considerações Finais 53
5.2 Limitações 54
5.3 Trabalhos Futuros 55

REFERÊNCIAS 56

APÊNDICES 58

1 Introdução

A Internet das Coisas, advinda do termo em inglês, *Internet of things* (IoT) está se tornando cada vez mais comum, principalmente com o surgimento das *Smart Homes*(Casas Inteligentes) e da automação. De acordo com Atzori, Iera e Morabito (2010), o conceito central do IoT é a presença de uma ampla variedade de objetos e dispositivos, como sensores, tags, microcontroladores e telefones celulares, que podem se comunicar e cooperar entre si para alcançar objetivos comuns.

No entanto, uma dificuldade enfrentada pela IoT é a necessidade de atualizar remotamente esses dispositivos através de uma conexão pela Internet ou de uma conexão local de Wi-Fi sem acesso à Internet. Esse problema pode surgir quando um dispositivo está em uma localização de difícil acesso para ser conectado a um cabo USB para atualização. Portanto, é crucial que os dispositivos tenham a habilidade de receber atualizações críticas *Over the air* (OTA) para resolver vulnerabilidades.

OTA permite que dispositivos conectados à Internet baixem aplicativos e configurações pela rede, "pelo ar", sem precisar de fios. Ou seja, é uma forma de atualizar o *software* ou *firmware* de um *hardware* sem a necessidade de utilizar cabos USB.

Outra característica importante de IoT é a utilização de dispositivos de baixo custo. Neste trabalho, dois dispositivos ESP foram usados para testes: o ESP8266 e o ESP32. Além de serem acessíveis e baratos, esses dispositivos são amplamente utilizados.

Em vista disso, e com a crescente popularidade da Internet das Coisas (IoT), surgiram projetos inovadores e ferramentas úteis para o estudo e a implementação de tecnologias IoT. Um deles é o projeto OTALab, um projeto de iniciação científica pela Universidade Federal do Espírito Santo (UFES). De acordo com Cussuol et al. (2022), o OTALab é uma ferramenta que permite a criação e implementação de ambientes de teste para protocolos e aplicações IoT em dispositivos com microcontroladores.

O OTALab oferece dois perfis de usuário configuráveis. O Administrador pode adicionar ou remover dispositivos IoT, serviços e funcionalidades, configurar as definições da rede, incluindo SSID e senha Wi-Fi, e adicionar novos dispositivos na rede. A primeira configuração e cadastro de um dispositivo é necessário conectá-lo via USB. O Experimentador, por sua vez, pode visualizar uma lista de dispositivos e serviços ativos na rede e escolher um dispositivo para carregar um novo código (CUSSUOL et al., 2022).

Outra ferramenta popular no cenário de IoT é o Node-RED, uma plataforma baseada na web que aplica o conceito de programação em fluxo (FBP, na sigla em inglês) para tornar a programação mais acessível. Desenvolvido originalmente pela IBM

e atualmente mantido pela OpenJS Foundation, o Node-RED apresenta uma interface gráfica de usuário (GUI, na sigla em inglês) que permite a adição de componentes através de uma interface de arrastar e soltar (*drag-and-drop*) blocos que representam componentes de um sistema maior.

O Node-RED é uma plataforma de conexão de dispositivos, plataformas de software e serviços web, onde blocos adicionais podem ser inseridos para representar funções que processam e transformam dados durante a transmissão. O OTALab (atualizado à data deste texto) foi projetado para funcionar como uma API, conhecida como OTALabBackEnd, fornecendo várias funções de acesso. Para fazer a implantação e uso do Node-RED, é necessário utilizar nós da biblioteca do Node-RED para realizar uma solicitação HTTP (*http request*, em inglês) das funções oferecidas pela API do OTALab.

A comunicação entre o cliente e o servidor é realizada através do protocolo HTTP, seguindo um conjunto de regras definidas. A solicitação HTTP, também conhecida como *http request*, é uma requisição realizada pelo cliente ao *host* em um servidor, visando acessar um recurso específico. O cliente usa componentes da *Uniform Resource Locator* (URL) para efetuar a solicitação, que inclui as informações necessárias para acessar o recurso.

O Node-RED é uma ferramenta desenvolvida especialmente para aplicações IoT e se baseia no conceito de FBP, onde o processo é assíncrono e cada nó é um processo separado. Além disso, possui um *flow-editor* baseado em interface WEB de fácil utilização. Por estas razões, o Node-RED foi escolhido como a interface entre o usuário e o servidor para o projeto OTALab.

Com o uso do módulo *node-red-dahsboard*, que oferece uma ampla gama de nós para a criação rápida de painéis de dados em tempo real, é possível criar uma interface de usuário gráfica (GUI) melhor e mais intuitiva tanto para o usuário quanto para o desenvolvedor, além de aprimorar a manutenção do projeto OTALab.

1.1 Motivação e Justificativa

A Internet das coisas (IoT) está rapidamente se tornando um setor de alta demanda, oferecendo conveniência, eficiência e automação tanto para as tarefas cotidianas quanto para as tarefas industriais. Com uma ampla variedade de dispositivos e sensores disponíveis a baixo custo, o cenário para IoT é extremamente positivo.

No entanto, a atualização remota desses dispositivos e sensores ainda representa uma dificuldade significativa. É aqui que entra o projeto OTALab, cujo objetivo é criar ambientes de teste, conhecidos como "*testbeds*", para atualização de *software*, *firmware* e *upload* de código de forma remota através do paradigma "*Over the air* (OTA)".

O projeto OTALab tem se concentrado no desenvolvimento do OTALabBackEnd,

deixando o OTALabFrontEnd ainda sem implementação. Dessa forma, este trabalho visa adicionar uma nova camada ao projeto usando Node-RED, que atuará como uma camada de interpretação de dados da API e também como uma camada de *frontend*, utilizando uma biblioteca de nós do Node-RED chamada *node-red-dashboard*.

Em suma, o objetivo é aprimorar as funcionalidades e interfaces do projeto OTALab, explorando as capacidades do Node-RED, uma ferramenta desenvolvida especificamente para aplicações IoT, projetada para ser uma interface gráfica de fácil utilização.

1.2 Objetivos

1.2.1 Objetivo Geral

Integrar a programação em fluxo com a ferramenta Node-RED no ambiente de experimentação OTALab.

1.2.2 Objetivo Específicos

- Fundamentar o uso de programação em fluxo com a ferramenta Node-RED, para o projeto OTALab.
- Implementar o uso do Node-RED no projeto OTALab.
- Analisar as melhorias feitas com a implantação do Node-RED no projeto OTALab.
- Análise de desemprenho da ferramenta.
- Facilitar a manutenção da aplicação com a utilização da ferramenta Node-RED.

1.3 Método de Desenvolvimento do Trabalho

O projeto será dividido em quatro partes principais. A primeira parte consistirá em uma análise aprofundada da tecnologia Node-RED e o motivo de sua integração ao projeto OTALab ser tão importante.

A segunda parte será dedicada ao desenvolvimento de uma instância do Node-RED conteinerizada específica para o perfil Experimentador do OTALab, incluindo uma interface intuitiva através do *dashboard-ui*.

A terceira parte será dedicada ao desenvolvimento de uma instância do Node-RED conteinerizada para o perfil Administrador do OTALab, também com uma interface através do *dashboard-ui*.

Por último, haverá uma avaliação de desempenho da aplicação como um todo.

1.4 Contribuições

Este trabalho contribui para a ampliação do projeto OTALab, visto que pelo Node-RED foi criado uma camada entre a API REST do OTALabBackEnd e o usuário. Além disso, também criou-se uma separação em contêiner diferentes para cada tipo de usuário — o Experimentador e o Administrador.

Com o Node-RED foi possível criar uma camada poderosa entre o usuário e o desenvolvedor, utilizando o nó *function* para criar códigos em JavaScript — responsável em analisar e processar os dados da API. Além disso, foram criadas interfaces gráficas usando *node-red-dashboard*, que facilitam muito a experiência do usuário.

O projeto com o *compose* para os contêineres e o arquivo JSON com os *flows* utilizados podem ser acessados pelo link do projeto no GitHub. Há também um video para mostrar todos os módulos e funções criadas para o Administrador, que pode ser acessado pelo seguinte link.

Ao final do trabalho espera-se que o uso de Node-RED no projeto OTALab contribua para o seu desenvolvimento, com novas funcionalidades. Além, de facilitar o acesso para a API, e se tornar uma forma mais simples de dar manutenção para a interface do projeto.

1.5 Organização da Monografia

Além desta introdução, este modelo de monografia é composto por outros quatro capítulos:

- Capítulo 2 – apresenta os aspectos relativos ao conteúdo teórico relevante para o trabalho;
- Capítulo 3 – apresenta a arquitetura proposta e o que foi desenvolvido;
- Capítulo 4 – apresenta uma análise de casos de uso do ambiente e um estudo dos resultados obtidos;
- Capítulo 5 – apresenta as considerações finais, as limitações, e possíveis trabalhos futuros do projeto.

2 Referencial Teórico e Tecnologias Utilizadas

2.1 Internet das coisas (IoT)

Embora a *Internet of things* (IoT) seja um termo bastante usado, o seu conceito é bem antigo. Um dos primeiros exemplos de IoT remete ao início da década de 1980, representada pela máquina da Coca-Cola. Onde os programadores locais se conectam pela Internet ao aparelho refrigerado e verificava se tinha alguma bebida disponível e se estava gelada antes de ir até à máquina para compra-la (FOOTE, 2022).

Entretanto, a frase Internet of Things foi formalizada em 1999, por Kevin Ashton, cientista da computação e diretor-executivo do MIT de Auto-ID, na revista Forbes (Schoenberger (2002)), Ashton relata "Precisamos de uma Internet para as coisas, uma forma padronizada para os computadores entenderem o mundo real". O artigo publicado pela revista Forbes foi intitulado "*The Internet of things*", e foi o primeiro uso documentado de forma literal do termo.

A IoT é conhecida por ser uma rede que se conecta a qualquer tipo de objeto. A Internet evoluiu para além de uma rede de computadores, abrangendo, assim, a todo tipo de objetos ou coisas, do carro a sua casa, do seu smartphone aos brinquedos, e de câmeras a equipamentos de medicina, dentre outros. Tudo isso, conectado e se comunicando entre si em uma rede inteligente, visando se ter um controle de monitoramento e segurança em tempo real (PATEL et al., 2016).

Pelo ponto de vista técnico, a IoT é resultado de uma mistura de várias tecnologias que, juntas, ajudam a diminuir a distância entre o mundo virtual e físico. Dentre elas, estão: comunicação, endereçamento, identificação, detecção, atuação, processamento de informação, localização e interface de usuário.

Nem toda aplicação precisa de todas as funcionalidades citadas, até porque acaba saindo caro, e um dos principais pilares de IoT é ter um baixo custo. Por isso, cada aplicação usa certa gama de funcionalidades diferentes e conforme a sua necessidade (MATTERN; FLOERKEMEIER, 2010).

Muitos anos se passaram desde o artigo na revista Forbes, ocasionando, por conseguinte, em diversas alterações no mundo todo, principalmente no que se refere à Internet. Assim, resta a pergunta: Em 2022 ainda é possível definir IoT da mesma forma? A IoT é comumente associada a uma gama bem variada de conceitos, tecnologias, e soluções. Os autores em Patel et al. (2016) chegaram à conclusão de que a melhor forma de descrever IoT é: "*IoT é uma estrutura conceitual que se aproveita da disponibilidade de dispositivos heterogêneos e soluções interconectadas, bem como objetos físicos que fornecem uma base de*

informações compartilhadas em escala global para apoiar o desenvolvimento de aplicações que envolvem o mesmo nível virtual tanto de pessoas quanto de representações de objetos". Porém, a IoT é uma tecnologia que está sempre evoluindo. Essa é uma definição que se faz presente, entretanto pode mudar no futuro.

2.1.1 Paradigma Over-the-Air(OTA)

O conceito de OTA nasceu da necessidade de precisar atualizar o *software* de certos dispositivos de forma remota. Existem vários tipos de soluções OTA para esse problema, de proprietárias a software livre. Escolher qual tipo de solução usar depende muito da necessidade da aplicação, logo deve-se considerar o custo, a largura de banda, o armazenamento, a manutenção do serviço, entre outras especificações (SALVADOR, 2019).

Receber *updates* remotamente e sem fio é essencial para resolver vulnerabilidades. Em razão disso que OTA *updates* age de forma imediata, mantendo a implementação robusta e assegurando a proteção dos dados.

Para ter uma solução IoT resiliente, robusta e confiável, é imprescindível manter a aplicação sempre atualizada, reagir rapidamente às vulnerabilidades que aparecerem nos seus dispositivos IoT e consertar os problemas ante que ele se espelhe para o resto da rede.

Existe 2 formas de fazer *updates* em seu dispositivo IoT, sendo por um cabo USB ou sem fio OTA. Conectar cada dispositivo implantado em um ambiente por um cabo USB para efetuar a atualização não é uma tarefa viável, pois o custo seria tão alto que coloca-se em risco perder segurança e recursos (El Jaouhari; BOUVET, 2022).

2.1.2 Dispositivos: ESP8266 e ESP32

Os dispositivos usados para testes neste projeto são os ESP32 e ESP8266. Estes, foram escolhidos por serem dispositivos de baixo custo, muito utilizados em implementações tanto residenciais quanto empresariais, sem contar que são de fácil acesso. Além disso, o fato de terem Wi-Fi embutido os tornam essenciais para o funcionamento do projeto.

Algumas especificações técnicas ESP8266:

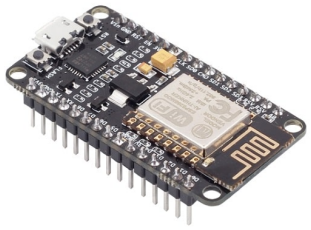

Figura 1 – ESP8266 NodeMcu

Fonte: <https://www.filipeflop.com/produto/modulo-wifi-esp8266-nodemcu-esp-12>

- Wireless padrão 802.11 b/g/n;
- Antena embutida;
- Conector micro-usb;
- Modos de operação: STA/AP/STA+AP;
- Suporta 5 conexões TCP/IP;
- Tensão de operação: 4,5 á 9V;
- Taxa de transferência: 110-460800bps.

Algumas especificações técnicas ESP32:

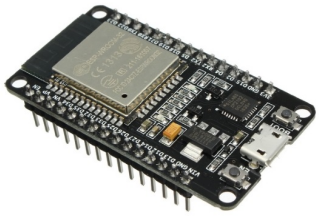

Figura 2 – ESP32 DevKit V1

Fonte: <https://www.filipeflop.com/produto/modulo-wifi-esp32-bluetooth>

- Wireless padrão 802.11 b/g/n;
- Antena embutida;
- Conector micro-usb;
- Conexão Wifi 2.4Ghz (máximo de 150 Mbps);
- Modos de operação: STA/AP/STA+AP;
- Tensão de operação: 4,5 á 9V;

- Bluetooth BLE 4.2–;
- ROM: 448 KBytes;
- RAM: 520 Kbytes.

2.2 Docker Contêiner

É imprescindível abordar acerca da virtualização para explicar como surgiu o conceito de contêiner. Simplificando, na computação, máquinas virtuais são as virtualizações ou emulações de um sistema de computador. Originalmente, em 1974 Popek e Goldberg (1974) definiram como *"uma cópia eficiente e isolada de uma máquina de computador real"*.

O termo contêiner não tem somente uma única origem. Assim, alguns exemplos de uso relevantes iniciais são advindos de Banga, Druschel e Mogul (1999), Lottiaux e Morin (2001), Morin et al. (2002), e em Price e Tucker (2004). Inicialmente, as literaturas se referiam confusamente a contêineres como um tipo de virtualização, ou até mesmo os chamava de máquinas virtuais. À medida que os contêineres se tornaram mais populares, a confusão mudou para as máquinas virtuais, chamadas de contêineres. Porém, na prática, a diferença entre contêineres e maquinas virtuais é mais complexa do que um simples 0 ou 1, onde as técnicas comuns a um podem ser aplicadas de forma eficaz ao outro (RANDAL, 2020).

A origem tanto das máquinas virtuais quanto dos contêineres podem ser atribuídas a uma mudança fundamental da arquitetura de hardware e software no final dos anos 1950. Na época, começou a ser introduzido o conceito de multiprogramação, que envolvia os conceitos de execução em paralelo de múltiplos programas na mesma máquina, além de multiprocessamento básico na forma de processadores de I/O dedicados e várias CPU (RANDAL, 2020).

2.2.1 Visão geral

Os contêineres têm uma longa história na computação. Diferentemente da virtualização, onde uma ou mais maquinas rodam virtualmente em um hardware físico por meio de uma camada de intermediação, os contêineres executam ao nível de usuário no *kernel* do próprio sistema operacional(SO). Como resultado, a virtualização de contêiner é geralmente chamada de virtualização no nível do SO. A tecnologia de contêiner permite que várias instâncias isoladas do espaço do usuário sejam executadas em um único *host* (TURNBULL, 2014).

Os contêineres são geralmente considerados uma tecnologia enxuta, pois possuem menor sobrecarga de processos. Ao contrário das tecnologias tradicionais de virtualização, elas não exigem uma camada de emulação para serem executadas e, em vez disso, usam a interface normal de chamada de sistema do sistema operacional. Isso reduz a sobre-

carga necessária para executar contêineres e pode permitir que uma maior densidade de contêineres seja executada em um *host* (TURNBULL, 2014).

Docker é um projeto de código aberto(*open-source*) que automatiza a implantação de aplicativos em contêineres. Ele foi escrito pela equipe da Docker, Inc (anteriormente dotCloud Inc — um dos primeiros participantes do mercado *Platform-as-a-Service* (PAAS)) e lançado por eles sob a licença Apache 2.0.

No caso do Docker, ter recursos modernos do *kernel* Linux, como grupos de controle e *namespaces* significa que os contêineres podem ter isolamento forte, suas próprias pilhas de rede e armazenamento, bem como a capacidade de gerenciamento de recursos para permitir a coexistência amigável de vários contêineres em um mesmo *host* (TURNBULL, 2014).

Neste trabalho, Docker contêiner é utilizado para fazer a separação de aplicações como a API *backEnd* do OTALab, o banco de dados MySQL e as aplicações de *frontEnd* em Node-RED.

2.3 Ambiente de experimentação

Um ambiente de experimentação, também conhecido como uma testbeds, é basicamente um ambiente onde você pode conduzir testes de forma mais rigorosa, transparente e reaplicáveis, de teorias científicas à ferramentas computacionais e novas tecnologias.

No contexto de IoT, usar uma *testbed* para analisar as novas tecnologias e algoritmos tornou-se essencial. Tal como alguns ambientes de experimentação já foram publicados e utilizados, entre eles estão o MoteLab [Werner-Allen, Swieskowski e Welsh (2005)], o INDRYA 2 [Appavoo et al. (2019)] e o FIT IoT-LAB [Adjih et al. (2015)]. Todos esses, são *testbeds* voltados à área de IoT, mas que demandam de certa complexidade e curva de aprendizado maior.

2.3.1 Visão geral

O OTALab propõe a criação de um *testbed* de forma simples, rápida e com pouca configuração para estudos de aplicações IoT. Nele, temos dois perfis de configuração, o Administrador, e o Experimentador. Na versão mais recente, o OTA-Lab expõe uma API REST, que oferece a ambos usuários as funcionalidades designadas a cada um deles. O primeiro consegue configurar a rede e adicionar ou remover, serviços, funcionalidades e dispositivos IoT do sistema. Já o segundo, este tem acesso a lista de dispositivos e seus respectivos serviços acoplados e faz *upload* de códigos para tais dispositivos. O administrador e o experimentador devem interagir com o *testbed* via API REST (CUSSUOL; MOTA, 2022).

No OTALab, a interação com o sistema pode ser feita por *Command Line Interface* (CLI) ou por API. No projeto, também é aplicado o paradigma *Over the air* (OTA) para poderem receber uma atualização de *firmware* via Wi-Fi, ou seja, sem fio (CUSSUOL et al., 2022).

O OTALab é um projeto de iniciação científica da UFES, utilizado como ferramenta de ensino em matérias relacionadas à IoT.

2.3.2 Ambiente OTALab

O OTALab possui dois perfis para a sua utilização: o Administrador, que pode adicionar remover, tanto serviços como dispositivos, onde cada dispositivo pode ter um conjunto de sensores atrelados a ele; o Experimentador, que consegue visualizar os dispositivos ativos na sua rede, bem como as suas funcionalidades, além de fazer *upload* de códigos para qualquer um desses dispositivos, necessitando apenas a biblioteca disponibilizada pelo OTALab seja utilizada. Que será responsável por fazer os processos de compilação e atualização de *firmware*, além de manter a conectividade (CUSSUOL et al., 2022).

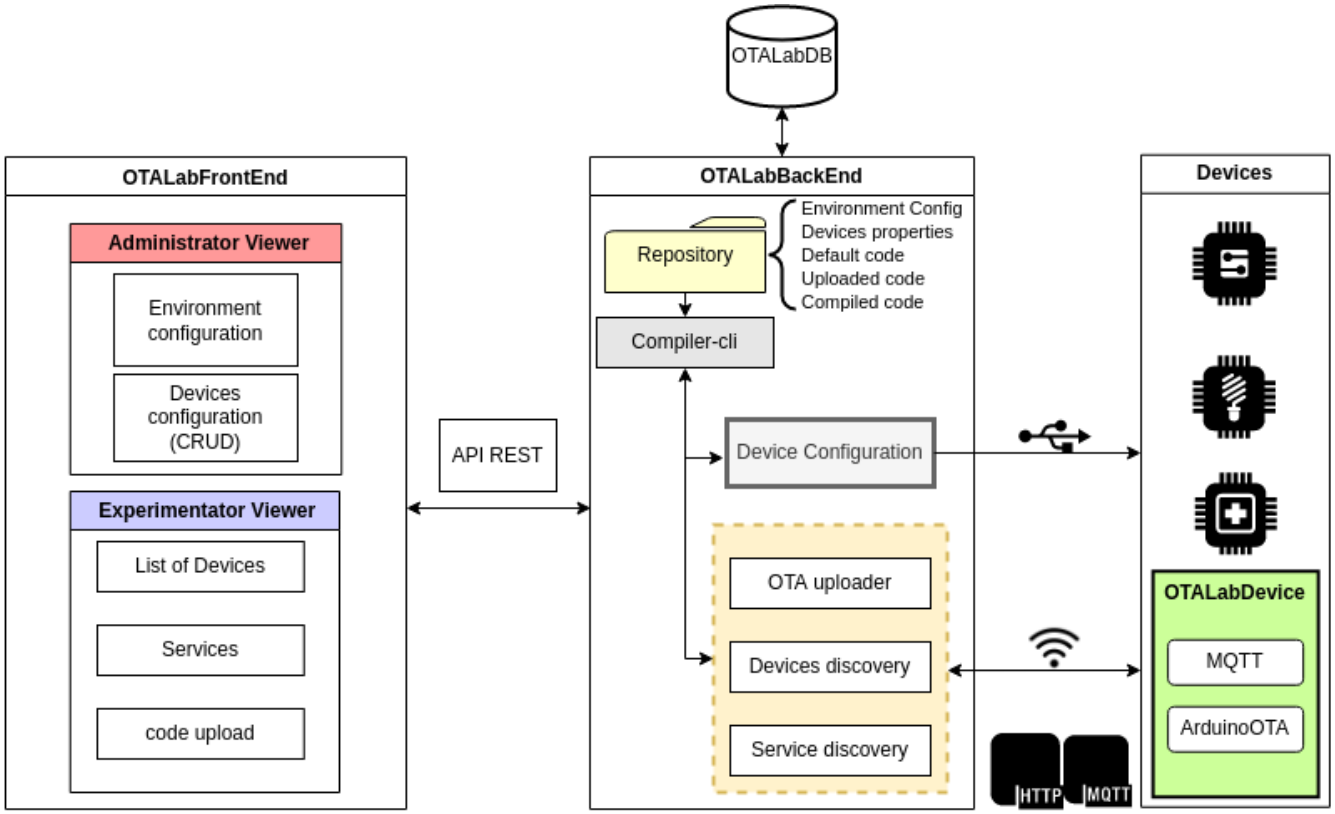

Figura 3 – Visão geral da arquitetura do OTALab Cussuol e Mota (2022).

A arquitetura do OTALab pode ser vista na Figura 3. Nela se torna possível observar como é feita a interação entre os componentes. Onde módulo OTALabFrontEnd, ainda a ser implementado, deve oferecer ao usuário uma forma simples, prática e de fácil aprendizado para interagir com o sistema.

No OTALabFrontEnd é onde esse trabalho propõe a implantação do Node-RED. Já no módulo OTALabBackEnd é onde todos os recursos são implementados, sendo ele o responsável pela configuração dos dispositivos, descobertas de dispositivos, serviços, e controla o *upload* de códigos para dispositivos. Por meio de uma API REST o OTALabBackEnd oferece ao *frontend* acesso a esses recursos (CUSSUOL; MOTA, 2022).

O OTALabFrontEnd apresenta visões distintas para o Administrador e o Experimentador do sistema. Na interface do Administrador, o usuário define as configurações da rede, bem como o SSID e senha do Wi-Fi, além de configurar e adicionar novos dispositivos na rede, no qual, na primeira configuração, o dispositivo precisa estar conectado via USB. Já na interface do Experimentador, é possível visualizar uma lista com os dispositivos e serviços ativos na sua rede, e escolher um dispositivo para fazer *upload* de um novo código (CUSSUOL; MOTA, 2022).

O OTALabBackEnd, desenvolvido em Java, se responsabiliza pelo gerenciamento e orquestração de todos os dispositivos que constituem o *testbed*. Ele se conecta com um banco de dados MySQL, o OTALabDB, encarregado pelo armazenamento de todos os dados gerados pelos usuários.

Ademais, o OTALabBackEnd armazena códigos enviados, compiladores específicos para os microcontroladores suportados, suporte para envio de *firmware* via cabo. Além disso, se encarrega pela atualização de *firmware* via OTA, bem como a descoberta de dispositivos e seus respectivos serviços.

Pensando também em projetos futuros e na escalabilidade do OTALab, houve a conteinerização do projeto. Desse modo temos então um contêiner para OTALabBackEnd, onde rodará todas as aplicações de *backend*. Um contêiner para o OTALabDB, com banco de dados MySql, que atuará como um banco de configuração, servindo para armazenar todos os dados gerados pelos usuários e informações simples do *backend*, como os dispositivos cadastrados. A Figura 4 ilustra o esquema do OTALabDB criado para a aplicação.

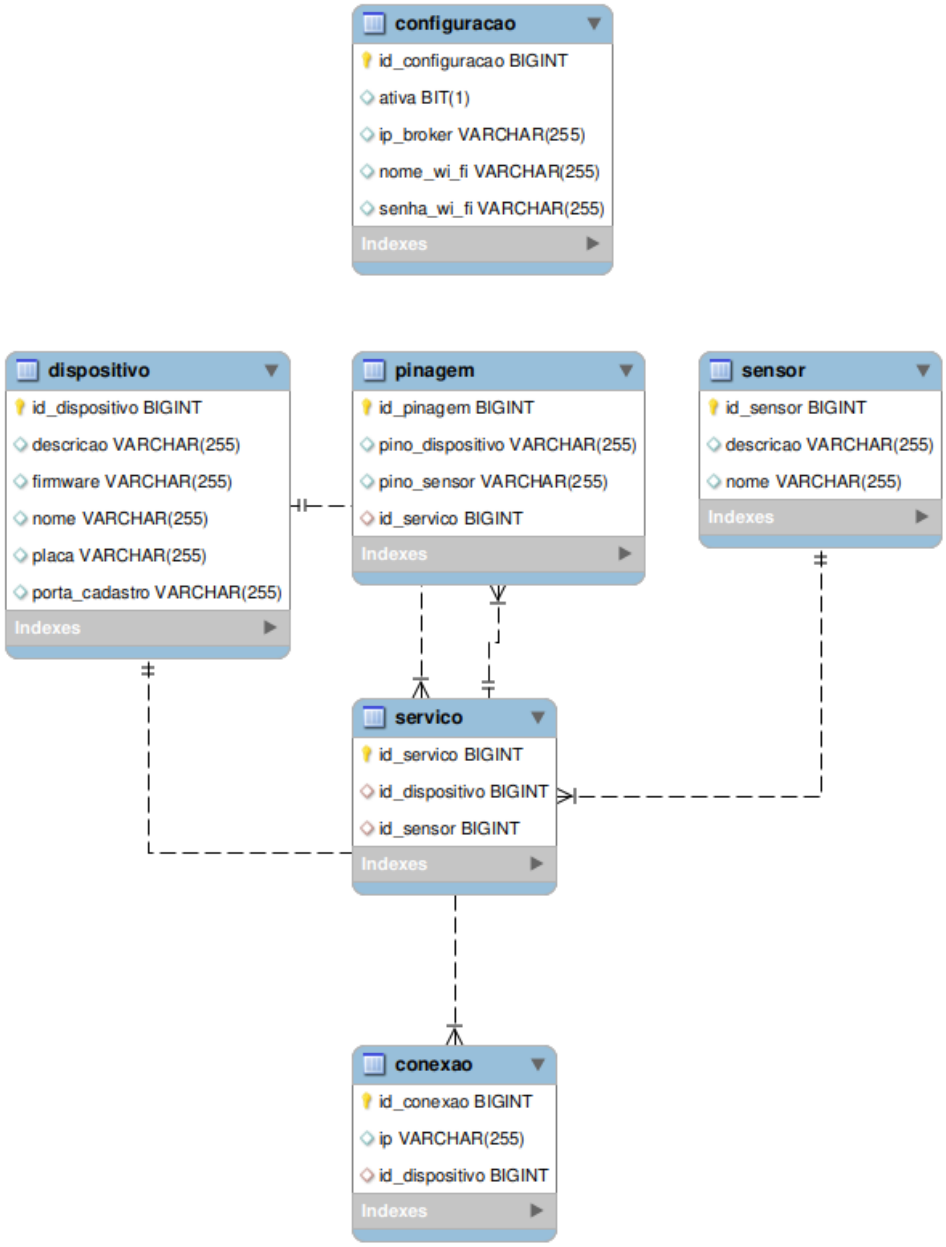

Figura 4 – Schema do OTALabDB Cussuol e Mota (2022).

Uma parte essencial do OTALab é a sua biblioteca, que deve ser obrigatoriamente incluída em todos os códigos Arduino que forem utilizados. Na Listagem 2.1 abaixo, temos o *template* que deverá ser utilizado. Essa biblioteca foi desenvolvida para ser utilizada com o Arduino IDE e, sua principal função é abstrair as configurações de rede sem fio o e uso dos protocolos de comunicação HTTP e MQTT.

O protocolo HTTP é usado como um servidor que permite a consulta de informações, como uma listagem de serviços. Já o MQTT é utilizado para descobrir os dispositivos conectados na rede, e ao conectar um dispositivo na rede, ele se registra a um tópico MQTT. Ao receber uma mensagem do tópico OTALab Core, o dispositivo responde com o seu nome e IP (CUSSUOL et al., 2022).

Listagem 2.1 – Templete Arduino a ser usado.

```
#include <OTALabDevice.h>

String id = "";
OTALabDevice* device = new OTALabDevice();

void setup(){
    Serial.begin(115200);
    device->setup(id);
}

void loop(){
    device->handle();
}
```

Além da própria biblioteca que foi desenvolvida, a <OTALabDevice.h>, é necessário o uso de algumas outras bibliotecas desenvolvidas pela comunidade, como a PubSubClient[1], usada para implementar o protocolo MQTT, a ArduinoOTA[2], responsavel pelas atualizações sem fio *over-the-air*, e as ESP32WiFI[3] e ESP8266WiFi[4], usadas para estabelecer conexão Wifi com os dispositivos.

2.4 Programação em fluxo

O primeiro conceito de programação em fluxo surgiu no início dos anos 1970, quando o J. Paul Morrison se juntou a um grupo na IBM para ajudar a desenvolver e criar o *Bank of Montreal's 'Mech'*, um sistema de banco online, inovador, que funcionasse 24 horas durante 7 dias da semana. Nesse processo, J. Paul Morrison inventou a *Flow base programing* (FBP), que fez parte desse sistema. Sua primeira implementação ganhou o nome de AMPS(Advanced Modular Processing System), programada na linguagem *Assembly S/360*. O AMPS rodava em um único computador da IBM. Parte desse projeto ficou em funcionamento e ativo até o ano de 2013, quase 40 anos depois de sua ativação.

Um artigo descrevendo os conceitos usados foi publicado pela IBM em 1978, no *IBM Technical Disclosure Bulletin* com o nome de *Data Responsive Modular, Interleaved Task Programming System* Morrison (1971). Logo após, em uma parceria da IBM Canada e IBM Japão, uma segunda implementação foi desenvolvida com o nome de *Data Flow Development Manager* (DFDM). No final dos anos 1980 ganhou o nome de *Data Flow Programming Manager*.

O conceito era comumente chamado dentro da IBM de *Data Flow*, o nome *Flow base programing* foi adotado quando J. Paul Morrison publicou o livro de mesmo nome em 1994, que consolidou o termo FBP. Uma segunda edição do mesmo livro foi publicada em

[1] https://github.com/knolleary/pubsubclient
[2] https://github.com/jandrassy/ArduinoOTA
[3] https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi
[4] https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi

2010. Ao longo dos anos, várias empresas desenvolveram e implementaram ferramentas baseadas em conceitos de FBP, como o Node-RED.

2.4.1 Visão geral

Flow base programing (FBP) é um paradigma de programação criado por J. Paul Morrison nos anos 1970, que usa a metáfora da "fábrica de processamentos de dados"para projetar e construir aplicações. Nela, a FBP define aplicações como uma rede de processos "caixa preta"(o qual é um sistema fechado de complexidade potencialmente alta, em que a sua estrutura interna é desconhecida ou não é considerada em sua análise), que se comunica por meio de blocos de dados, chamados de Pacotes de Informação, que viajam por conexões pré-definidas (metaforicamente seriam as correias transportadoras), onde as conexões são definidas externamente do processo. Nesses processos, a caixa preta pode se reconectar infinitamente e formar aplicações diferentes sem precisar ser alterada internamente. Devido a isso, a FBP é orientada a componentes (MORRISON, 1994).

Para entender como surgiu a FBP é importante entender o contexto da época, onde as linguagens e *softwares* eram baseados no modelo de computador de *von Neumann*. Esse modelo tradicional utilizado, era produtivo e funcionou muito bem nas últimas décadas. Projetado em torno de um único contador de instruções, que percorre sequencialmente o código decidindo o que fazer a cada passo, os códigos podem ser tratados como dados e como comandos.

Esse modelo funcionou bem durante muitos anos, mas tinha seus problemas. Em Morrison (1994), o autor descreve o seguinte cenário: "Imagine que você tenha uma aplicação grande e complexa rodando em uma fábrica, e você descobre que precisa fazer algumas mudanças complexas e que precisa ser rápido. Você consulta os programadores e eles te avisam que tais mudanças provavelmente levariam meses, mas que iriam olhar. Uma reunião é chamada e todas as pessoas envolvidas, não apenas programadores, mas também pessoas de operações. A lógica principal do programa é colocada em evidência, e os programadores fazem um passo a passo da estrutura do programa com o grupo. Durante essa discussão é decidido que 2 módulos precisam ser adicionados ao programa e alguns precisam mudar de lugar, e o tempo estimado para resolver isso é de uma semana!".

Toda essa demora acaba gerando um grande custo e como o autor cita no livro, esse foi um cenário real que aconteceu com ele, o que levou J.P. Morrison a desenvolver a *Flow base programing* (FBP). Um novo paradigma substitui o modelo de *von Neumann* como uma ponte entre o hardware e o software.

A FBP é caracterizada por processos assíncronos e simultâneos no *background*, pacotes de dados com tempo de vida definido, portas nomeadas, conexões com *buffer* limitado e definições das conexões externas aos componentes. Isso tudo melhora o tempo de

desenvolvimento, manutenção, reutilização, prototipagem rápida, melhora de desempenho, entre outros pontos de todo o processo. Além disso, a FBP tira proveito da existência de múltiplos núcleos de processamento (MORRISON, 1994).

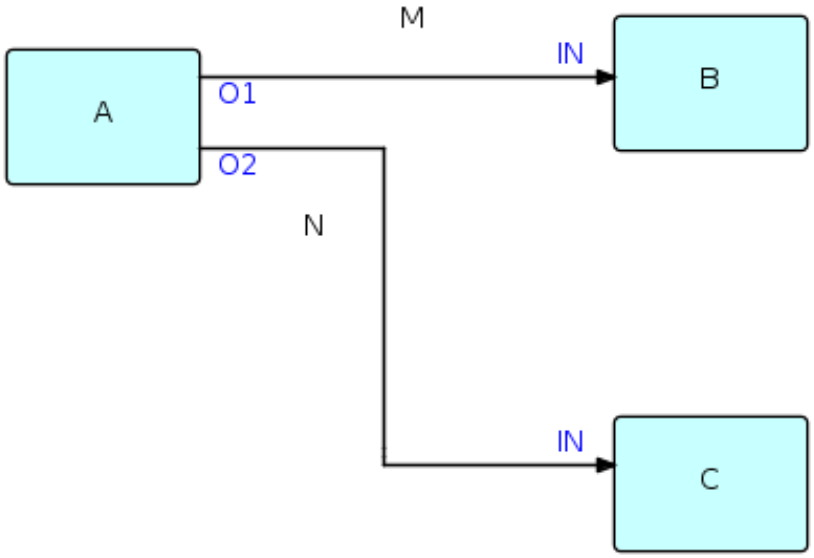

Figura 5 – Exemplo de um design básico em FBP, Morrison (1994).

Na Figura 5, temos um exemplo de design bem básico de um fluxo em FBP. Com a criação do *Flow base programing* vários sistemas se basearam em FPB, como o NoFlo, Flowhub e Node-RED [Hagino e O'Lary (2021)], que foi a ferramenta escolhida para ser usada neste projeto.

2.4.2 Node-red

Node-RED é uma ferramenta de desenvolvimento baseada em FBP para programas mais visuais. Seu Desenvolvimento deu-se pelo time de Tecnologias e Serviços Emergentes da IBM, composto pelos membros Nick-O'Leary e Dave Conway-Jones, no início de 2013. Inicialmente era apenas um *Prove of Concept* (PoC) para ajudar a visualizar e entender o mapeamento de tópicos MQTT. O Node-RED se tornou um software livre em setembro de 2013 e se mantém assim até o momento. Enquanto o Node-RED está sobre a administração da OpenJS Foundation.

Vale ressalta que a FBP descreve o comportamento da aplicação como uma "caixa preta", que no caso do Node-RED é chamada de *node* ou nós, em português. O Processamento é definido em cada nó, que recebe os dados, a faz o processamento usando esses dados e passa o resultado para outro nó. A rede faz o papel de permitir que haja um fluxo de dados entre os nós (HAGINO; O'LARY, 2021).

O Node-RED não é somente uma ferramenta de programação, mas também uma

plataforma de execução. O Node-RED funciona como uma aplicação web feita em Node.js, que utiliza o *runtime* do próprio Node.js. Para fazer as aplicações para IoT, os serviços web, entre outros é necessário usar o *flow-editor*, que funciona como uma página web no seu navegador, em que é possível escolher o nó que queira usar e arrastá-lo para o seu *workspace*. Após isso, dá para efetuar as conexões dos nós por "fios"e com isso, temos uma aplicação que pode ser rodada clicando em *Deploy*.

Na Figura 5, temos uma imagem que ilustra essa interface (HAGINO; O'LARY, 2021).

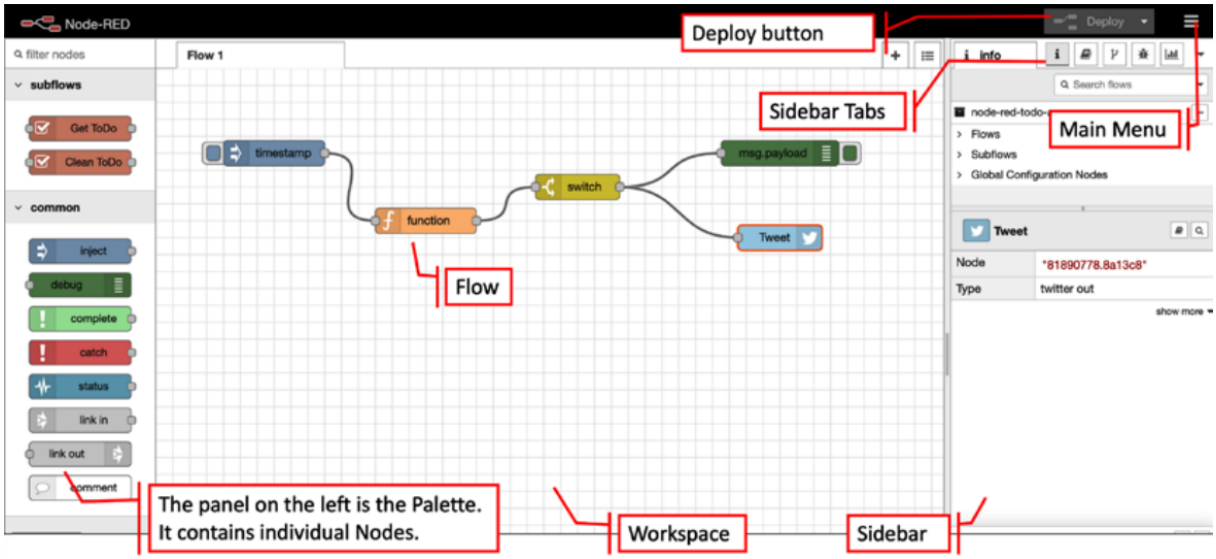

Figura 6 – Interface de edição do Node-RED, Hagino e O'Lary (2021).

O Node-RED é uma ferramenta de programação em fluxo, adequada para fazer controle de dados de aplicações web e IoT. O ambiente de desenvolvimento e execução sendo todo feito direto no navegador usando Node.js faz com que o desenvolvimento de aplicações sejam bem triviais. Com o uso de Node-RED, torna-se possível evitar o uso de programação mais complexa, principalmente para quem não tem conhecimento acerca de programação (HAGINO; O'LARY, 2021).

Além disso, tem alguns outros pontos importantes a favor do uso da ferramenta, são eles: simplificação. Com o Node-RED, temos um nível de complexidade muito baixo e uma interface bem simples, intuitiva e fácil de usar. Eficiência. Isso dá-se por haver uma interface que faz toda a parte de execução e sincronização com as bibliotecas. Assim, dá para se concentrar mais no desenvolvimento. Alta qualidade. Cada nó já foi testado, garantido assim sua qualidade.

Desse modo, o erro humano é eliminado justamente por não haver preocupação com o que tem dentro do nó. Código aberto. O Node-RED é um software livre, permitindo uma aproximação maior com a comunidade, ajudando na criação de uma plataforma capaz de publicar nós desenvolvidos individualmente para todos usarem.Ainda, faz-se mencionar

o último ponto: possuir uma vasta biblioteca com mais de 2000 (dois mil) nós disponíveis (HAGINO; O'LARY, 2021).

O Node-RED é um ambiente virtual que combina dispositivos hardware como os ESP32 e ESP8266, com API's e outros serviços onlines. Por isso, é uma ferramenta poderosa para construir serviços de IoT. Originalmente, o conceito de Node-RED foi desenvolvido para aplicações em IoT.

Assim, temos então 3 (três) pontos importantes: primeiro, como pode ser executado em *edge devices* (dispositivos da "borda"), como o Raspberry Pi, ela é ideal para manipulação de dados na camada do dispositivo; segundo, como pode ser executado na nuvem, usando serviços como a AWS e IBM Cloud, é fácil se conectar com uma forma de armazenamento e *middleware* de análises. Por último, como os protocolos HTTP e MQTT são partes do Node-RED, fica fácil para trocar dados com *edge devices* e um servidor de processamento na nuvem (HAGINO; O'LARY, 2021).

3 Arquitetura Proposta

Como citado na seção 2.3.2, o projeto OTALab foi desenvolvido como uma API em Java, o OTALabBackEnd. Nele, são armazenados os códigos enviados, os compiladores específicos para cada microcontrolador suportado, o suporte para envio de *firmware* via cabo, a descoberta de dispositivos e seus respectivos serviços e, por fim, o *upload* de código via OTA. Ainda nesse capítulo, a Figura 3 ilustra a separação do OTALabBackEnd e do OTALabFrontEnd. No *frontEnd* é mostrado a separação entre o perfil de Administrador e Experimentador.

O projeto OTALab teve como foco o desenvolvimento do OTALabBackEnd. Além disso, o OTALabFrontEnd ainda não foi implementado, sugerindo apenas a criação de dois perfis de configuração, o Administrador e o Experimentador. No Administrador, o usuário tem acesso a todas as funções disponibilizadas pela API. Já o Experimentador tem um acesso limitado a API.

Entretanto, no de desenvolvimento do projeto OTALab foi realizado apenas um *backend* na forma de uma API REST chamado de OTALabBackEnd. Com isso o OTALab na versão atual não possui um *frontend*, deixando assim essa parte em aberto. Sendo assim, este trabalho implementa o *frontend*, chamado de OTALabFrontEnd, usando a ferramenta Node-RED, que servirá também como uma camada de separação entre os perfis de configuração e possuirá uma interface gráfica para cada perfil. Para entender o porquê da escolha quanto a ferramenta Node-RED, criou-se uma lista com os principais motivos:

1. O núcleo do Node-RED está o Node.js, um *runtime* do JavaScript que possui um dos maiores ecossistemas de componentes de código aberto. Em detrimento disso, é maduro e possui uma comunidade muito ativa e com muitos usuários. Além disso, JavaScript é muito fácil de aprender, pois não exige nenhuma configuração. Com tudo embutido em seu navegador web, basta começar a escrever o código e ver os resultados imediatamente em seu navegador.

2. O Node-RED, utiliza JSON(JavaScript Object Notation) para descrever seus metadados. Ainda, o JSON é muito mais simples e fácil de entender que o XML. Sem contar que o JSON também é mais rápido, visto que foi projetado especificamente para intercâmbio de dados. A codificação JSON é concisa, requerendo menos bytes para transferências. As traduções e análises de arquivos JSON são menos complexos, requerendo menos tempo de processamento e sobrecarga de memória. O XML é mais lento porque foi projetado para muito mais do que apenas intercâmbio de dados.

3. No Node-RED, a programação é baseada em fluxo como mostrado na seção 2.4.

A FBP é uma forma diferente de pensar programação, que divide o problema em dados, em processos que operam nesses dados e na rede que conecta tais processos. Estes, processos são agrupados em um fluxo para cumprir um objetivo. De forma semelhante, os fluxos podem ser logicamente agrupados para atingirem objetivos de ordem superior e assim por diante, dados entram e dados saem. Com isso, a FBP se revela bastante apropriada para a programação visual, permitindo assim desenhar códigos.

4. O Node-RED, foi projetado para estar na borda da rede, pois é ali que estão sendo criados os novos dados, como o de GPS, de câmeras, de sensores, de e Smartfones, e até de carros. A IBM no desenvolvimento do projeto do Node-RED definiu como seu DNA uma maneira de visualizar o envio e o recebimento de mensagens MQTT para dispositivos de hardware, bem como seus fluxos de dados a aplicativos.

5. O Node-RED, não é só protótipo, com o ele é simplesmente possível adicionar e remover uma coleção de novas funções e funcionalidades, ou um aplicativo inteiro do navegador ao banco de dados. Além disso, o Node-RED atenderá perfeitamente o usuário, isto, desde o protótipo à produção. Do mais, o Node-RED possui uma poderosa comunidade ativa que mantém uma biblioteca muito ampla de funções e fluxos prontos para serem usados.

6. O Node-RED permite a criação de funcionalidade ao conectar fluxos de dados entre nós utilizando um navegador. Ganhou enorme popularidade no espaço da IoT, modelando bits de funcionalidade de aplicativos entre dispositivos de IoT, como sensores, câmeras e roteadores sem fio. Contudo, o Node-RED não é somente para aplicações IOT, visto que existem milhares de nós prontos, que vão de leitura de dados do Twitter à análise de sentimentos desses dados, de leitura de dados de um formulário da internet, à interação com bot do Telegram usando inteligencia artificial como o IBM Watson. Tornando, assim, o Node-RED uma ferramenta extremamente flexível.

Com a intenção de fazer a integração do Node-RED no OTALab, será utilizado um arquivo de configuração do próprio Node-RED para adicionar login e senhas para cada perfil, bem como 3 (três) novas paletas de nós, que estão disponíveis na biblioteca do Node-RED, sendo: O *node-red-dashboard*[1], responsável por grande parte da *dashboard-ui*. Este, é um conjunto de nós que podem ser usados para criar rapidamente uma interface gráfica com um painel de dados ao vivo. Todavia, o *node-red-dashboard* não possui nativamente um nó para fazer *upload* de arquivos pelo *dahsboard*. Será necessário também o uso do *node-red-contrib-ui-upload*[2], que soluciona esse problema. Ademais será utilizado o *node-*

[1] <https://flows.nodered.org/node/node-red-dashboard>
[2] <https://flows.nodered.org/node/node-red-contrib-ui-upload>

red-node-ui-table[3] que facilita a criação de tabelas para a visualização das informações no *dashboard-ui*.

Desse modo, para completar o projeto OTALab e iniciar o desenvolvimento do OTALabFrontEnd, é proposto então um novo desenho para complementar a arquitetura do projeto atual, que pode ser visto na Figura 7.

Nessa arquitetura proposta, temos a adição de dois contêineres com Node-RED instalado em cada um deles, um para o Administrador e um para o Experimentador. Tendo assim uma separação completa entre os dois perfis de usuários.

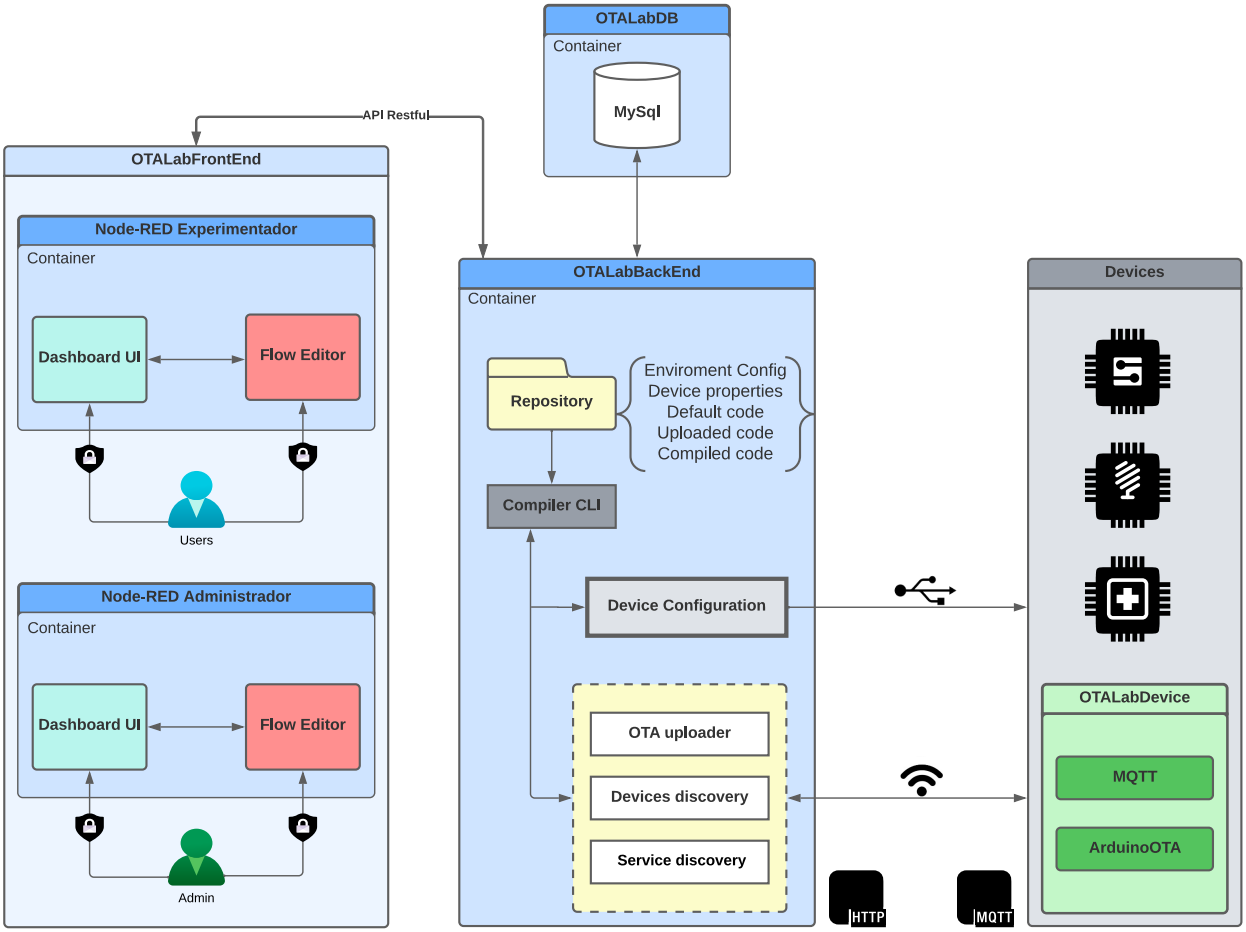

Figura 7 – Visão geral da arquitetura proposta para implantação do Node-RED no OTA-Lab

A separação dos dois ambientes faz-se necessária para que o Experimentador não tenha acesso às funções do Administrado. Ainda outra separação é feita entre o ambiente do *dashboard-ui* e do *flow-editor* para que um usuário comum não tenha acesso ao editor de *flows* e altere a interface gerada pelos nós da *dashboard*. Para isso, o Node-RED permite, pela configuração do arquivo *setings.js*, adicionar usuários e senha para cada um dos ambientes, tanto o *flow-editor*, que se rodando localmente, pode ser acessado por

[3] <https://flows.nodered.org/node/node-red-node-ui-table>

http://127.0.0.1:1880, quanto o *dashboard-ui*, que nesse mesmo contexto é acessado por *http://127.0.0.1:1880/ui*.

Assim, é possível observar na Figura 8, melhorando consideravelmente a sua segurança e alguns acessos indevidos são limitados. Para fazer isso, basta procurar no arquivo *setings.js* pela variável *adminAuth*, para adicionar ou alterar um usuário e senha no *flow-editor* e por *httpNodeAuth* para realizar o mesmo procedimento para interface da *dashboard-ui*. As senhas são armazenadas no arquivo em um formato de *hash*. Para converter uma senha normal para *hash*, basta digitar no terminal *"node-red admin hash-pw"* e depois digitar sua senha. Esse comando irá retornar uma *hash* correspondente a sua senha.

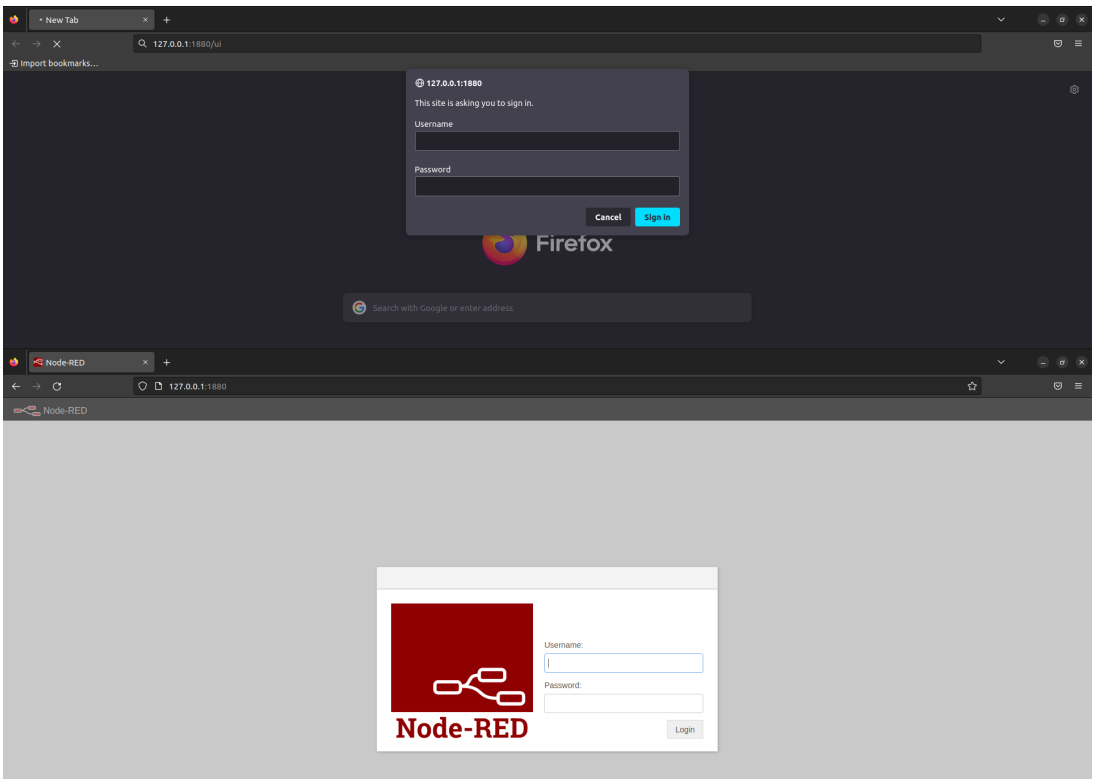

Figura 8 – Telas de login para a *dashboard-ui* na parte superior e *flow-editor* na parte inferior

3.1 Perfil Administrador

O perfil Administrador possui acesso a todas as funções disponibilizadas pela API. Na Figura 9 é possível observar a interface gerada usando a *dashboard-ui* do Node-RED, onde o Administrador é dividido em 7 módulos, e cada módulo contendo um conjunto de funções. Já no Apêndice, na Figura 27, podemos visualizar todos os *endpoints* disponibilizados pela API do OTALab.

Figura 9 – Tela inicial no *dashboard-ui* do Administrador

Cada um dos 7 mudulos tem suas respectivas funções, sendo:

1. **Configurações de Dispositivos:**
   - **Cadastrar Dispositivo** — local onde é feito o cadastro de um novo dispositivo. Para isso é necessário digitar um nome, fazer uma breve descrição do dispositivo, escolher a porta USB na qual o dispositivo está conectado, ttyUSB0, por exemplo, e ao final digitar o nome da placa a ser usada. Ao utilizar, por exemplo, a *esp8266:esp8266:nodemcu*, se torna importante considerar que a placa usada seja de uma já previamente instalada para funcionar. A lista de placas instaladas pode ser visualizada no módulo de configuração de recursos.
   - **Listar Dispositivos cadastrados** — essa função apresenta uma tabela com todos os dispositivos cadastrados e várias informações sobre eles, tais como, o ID, o *firmware*, os sensores acoplados, o IP caso esteja conectado, entre outras informações.
   - **Listar Dispositivo por ID** — apresenta todas as informações de um dispositivo específico, dado um ID.
   - **Deletar Dispositivo** — deleta um dispositivo da base de dados, dado um ID.
2. **Configurações de Recursos:**
   - **Instalar Biblioteca** — essa função possibilita a instalação de novas bibliotecas para Arduino, como a *PubSubClient*, uma biblioteca usada para conexões MQTT.
   - **Listar Placas instaladas** — apresenta uma lista de todas as placas previamente instaladas.

- **Listar Portas USB** — exibe uma lista com todas as portas USBs em uso no momento.
- **Listar Bibliotecas instaladas** — apresenta uma lista com todas as bibliotecas instaladas.
- **Deletar Biblioteca** — deleta uma biblioteca instalada no OTALabBackEnd, dado um ID.

3. **Configurações de Rede:**

   - **Listar Redes cadastradas** — essa função apresenta uma tabela com todas as redes cadastradas, e as informações sobre essas redes, SSID ou nome da rede, IP do *Broker*, que é responsável pela conexão MQTT e o seu *status*, que identifica qual rede está ativa no momento.
   - **Listar rede por ID** — exibe todas as informações de uma rede dado um ID.
   - **Cadastrar dados de rede** — onde é feito o cadastro de uma nova rede. Para isso, é necessário digitar o nome e a senha da rede Wi-Fi, e em seguida, um IP do *broker* MQTT que deseja usar, sendo ele publico ou privado.
   - **Marcar rede como Ativa** — como é permitido ter várias redes cadastradas simultaneamente, marcar como ativo é a função que você seleciona, dado um ID qual rede irá usar.
   - **Deletar rede cadastrada** — basicamente, deleta uma rede cadastrada dado um ID.

4. **Conexão e Upload:**

   - **Atualizar lista de conexões** — essa função envia um sinal MQTT, dado uma rede, para descobrir quais dispositivos estão vivos nessa rede, e espera uma resposta por um determinado tempo.
   - **Listar todas as conexões** — exibe uma lista com o IP e um ID de conexão de todas as conexões ativas no momento.
   - **Listar conexão por ID** — apresenta o IP de uma conexão dado o ID.
   - **Upload** — essa é a função onde será feito o *upload* do código usando OTA para um determinado dispositivo. Para isso, é necessário indicar qual o ID da conexão e selecionar, qual arquivo deseja fazer o *upload*.

5. **Configurações de Sensores:**

   - **Cadastrar Sensores** — cadastra um sensor na base de dados, com um nome e uma breve descrição.

- **Listar Sensores cadastrados** — exibe uma lista com todos os sensores cadastrados e algumas informações, como o nome, descrição, e ID do serviço, o link entre o dispositivo e o sensor.
- **Listar Sensor por ID** — apresenta todas as informações de um sensor dado o seu ID.
- **Deletar Sensor** — deleta um sensor da base de dados, dado um ID.

6. **Configurações de Serviços:**

   - **Cadastrar Serviços** — para associar um dispositivo a um sensor é necessário criar um serviço. Para isso, basta digitar o ID do sensor e o ID do dispositivo.
   - **Listar Serviços cadastrados** — exibe uma lista com todos os serviços cadastrados e quais pinagens estão relacionadas entre o dispositivo e o sensor.
   - **Listar Serviço por ID** — apresenta as pinagens equivalentes do sensor e dispositivos de uma dado serviço.
   - **Deletar Serviço** — deleta um link entre um dispositivo e um sensor.

7. **Configurações de Pinagem:**

   - **Cadastrar Pinagem** — além de associar um sensor a um diapositivo é necessário listar as pinagens entre eles. Para isso, basta digitar o pin do dispositivo e do sensor, e o ID do serviço.
   - **Listar Pinagens cadastradas** — exibe todas as pinagens cadastradas.
   - **Listar Pinagem por ID** — presenta as informações de uma pinagem, dado o seu ID.
   - **Deletar Pinagem** — deleta uma pinagem da base de dados dado um ID.

3.1.1 Flow-editor do Administrador

O *flow-editor* é o local em que fica o núcleo de desenvolvimento do Node-RED. No caso do perfil do Administrador, criou-se um *flow* para cada um dos módulos e um *subflow* para cada uma das funções.

A Figura 10 ilustra todos os *subflows* criados para o Administrador.

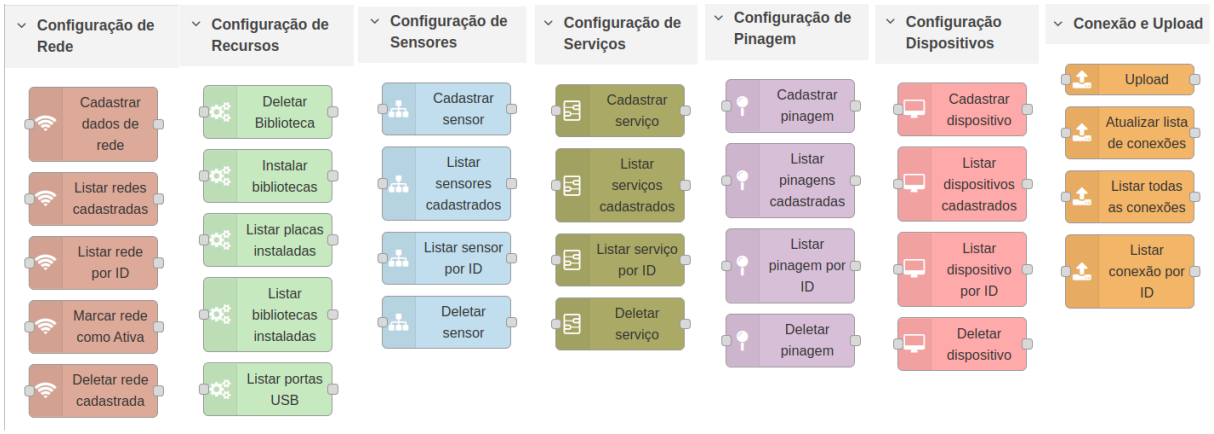

Figura 10 – Todos os *subflows* criados para o perfil Administrador

Cada um desses *subflows* é responsável por fazer o HTTP *request* para a API do OTALabBackEnd, e dependendo da resposta é feito uma manipulação com o resultado para exibir de forma assertiva as informações para o usuário no *dashboard-ui*.

Na Figura 11, temos um exemplo de como ficou o desenho no *flwo-editor* para 3 (três) *flows*, o *configração-controler*, o *dispositivo-controller*, e o *conexao-controller*. Um video foi feito com a finalidade de mostrar todos os módulos e funções criadas para o Administrador, que pode ser acessado pelo seguinte link.

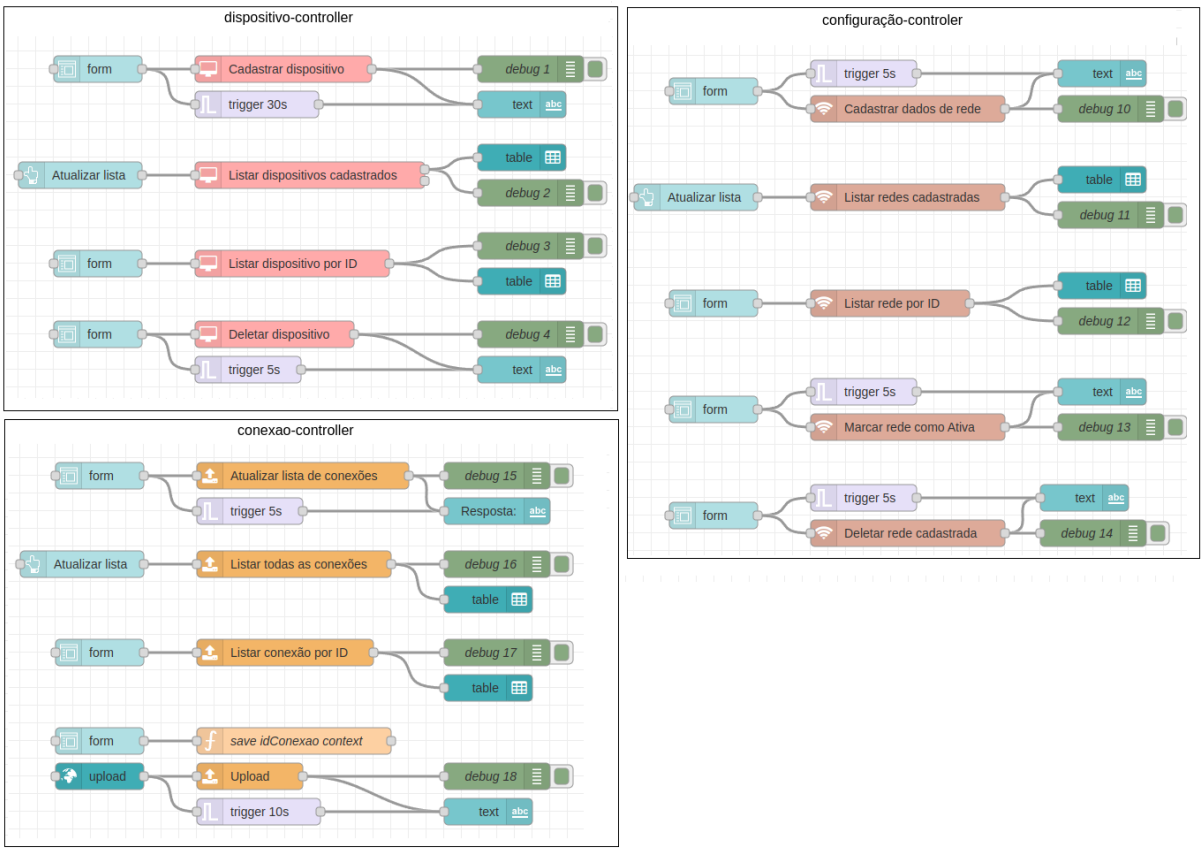

Figura 11 – Exemplo de 3 (três) *flows*, do *dispositivo-controller*, do *configração-controler*, e do *conexao-controller* criados para o perfil Administrador

3.2 Perfil Experimentador

O perfil Experimentador possui acesso limitado a API do OTALabBackEnd. O motivo para isso é o fato do Experimentador ser um perfil de usuário pouco experiente e novato em relação à manipulação e programação de dispositivos como os ESP e Arduinos. Então, com essa interface será possível fazer *uploads* e testar o código remotamente sem ter acesso a um dispositivo de fato.

Um exemplo para esse caso seria: em uma disciplina de IOT será necessário fazer um programa para piscar o led de um ESP ou fazer uma captura das informações de temperatura com um sensor, com o professor disponibilizando os dispositivos no laboratório. Liberando o aceso via OTALab, o aluno poderá fazer o *upload* e testar seu código remotamente sem estar no laboratório ou com acesso aos dispositivos.

Assim, utilizando o módulo *dashboar-ui* do Node-RED foi desenvolvido uma interface para o Experimentador, que pode ser vista na Figura 12.

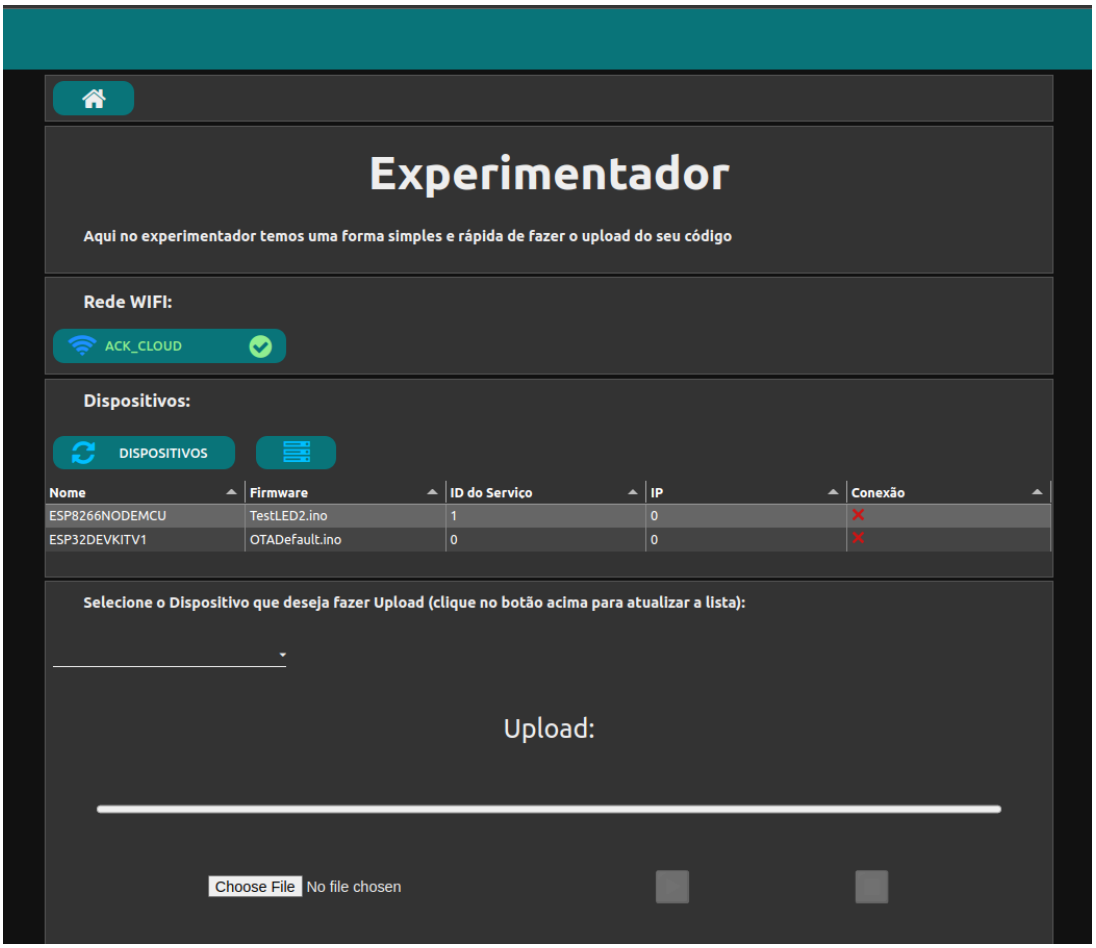

Figura 12 – Tela inicial no *dashboard-ui* do Experimentador

Com essa interface é possível observar, inicialmente, qual rede Wi-Fi está ativa, e qual é atualizada automaticamente a cada 30 segundos. Mas, ao clicar na rede, também é

possível atualizá-la.

Também é possível visualizar uma tabela com todos os dispositivos cadastrados e algumas informações importantes como, o nome do dispositivo, o *firmware*, o ID do serviço (que se diferir de 0 indica que aquele dispositivo tem um sensor acoplado), o IP do dispositivo e se sua conexão está ativa ou não, ou seja, se está ligado na mesma rede.

Ao clicar no botão *DISPOSITIVOS* ocorrerá a atualização dos dispositivos cadastrados e das conexões, isto, baseado na rede ativa mostrada acima. Para atualizar as conexões, um sinal MQTT é enviado à rede com o intuito de descobrir quais dispositivos estão vivos nessa rede. A resposta é esperada por um determinado tempo.

Ao lado do botão *DISPOSITIVOS*, temos o botão de serviços. Na Figura 13 é possível ver o seu funcionamento. Ao clicar no botão serviços, um novo bloco aparece na tela mostrando, assim, uma lista de todos os serviços cadastrados e quais pinagens estão relacionadas entre os dispositivo e os sensores.

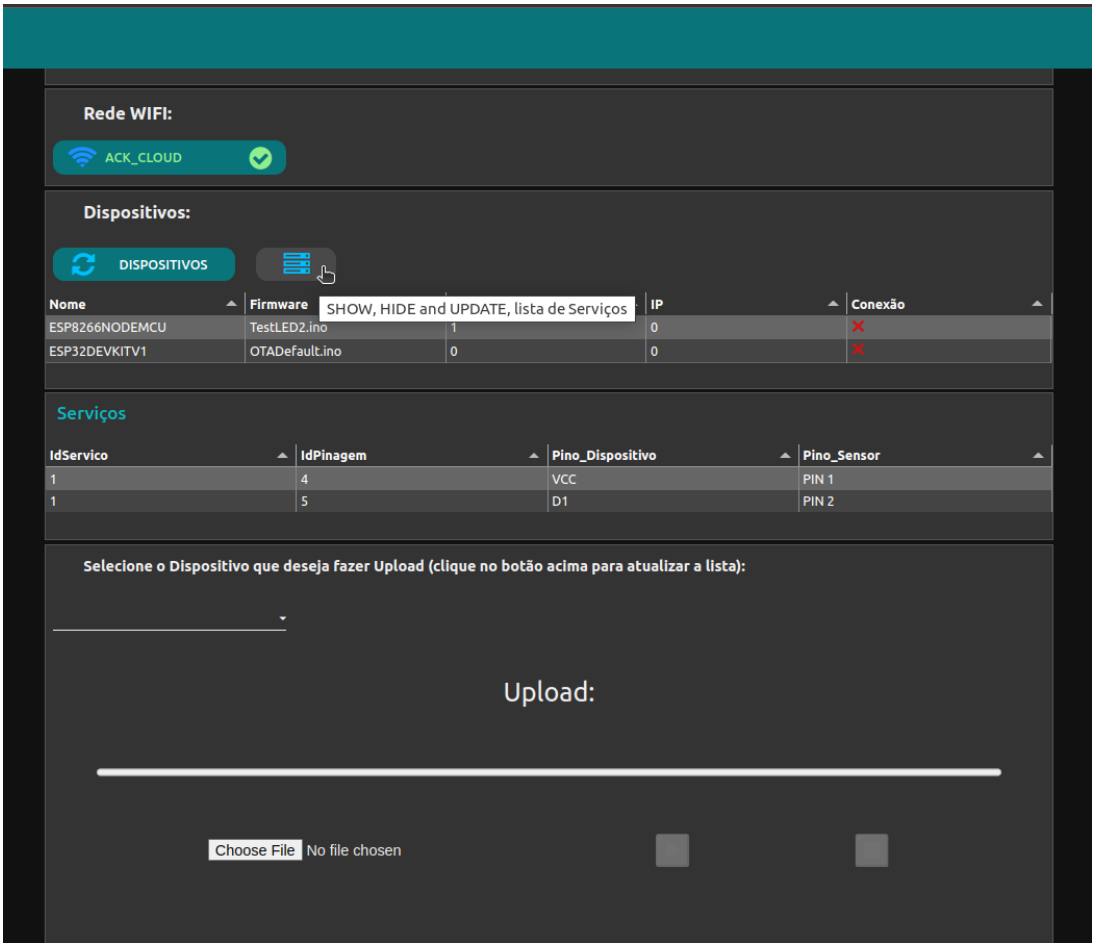

Figura 13 – Tela *dashboard-ui* do Experimentador ilustrando o funcionamento do botão Serviços.

Ao finalizar, é possível selecionar qual dispositivo deseja fazer o *upload* do seu código por meio de um *Dropdown* que lista apenas os dispositivos ativos. Após selecionar

o dispositivo, basta optar pelo arquivo com o seu código e fazer o *upload*. Se o processo ocorrer normalmente, o seu novo *firmware* vai rodar no dispositivo. Ademais, atualizando a lista de dispositivos se tornar possível observar a mudança na coluna do *firmware*.

3.2.1 Flow-editor do Experimentador

O *flow-editor* do experimentador é bem mais simples que o do Administrador, pois ele não tem acesso a todas as funções da API.

A Figura 14 ilustra como foi montado o *flow* do Experimentador. Nela, podemos observar que são utilizados apenas 5 (cinco) *subflows*, sendo:

- **Lista Redes**;
- **Atualizar lista de Conexões**;
- **Lista Dispositivos**;
- **Upload**;
- **Lista Serviços**.

Para demostrar qual conexão está ativa no momento e atualizar após um determinado tempo, é realizado um *request* para a função Lista Redes. Em seguida é filtrado apenas a rede marcada como ativa e um nó de *template* é criado em HTML e CSS para mostrar ao usuário essa informação de uma forma mais visualmente agradável.

Um nó de *template* foi customizado para gerar o botão de atualizar os dispositivos. Na listagem 3.1, podemos ver o seu código, que utiliza HTML, CSS e Javascript para fazer a animação de *loding* por um determinado tempo até atualizar a lista assim que o botão é clicado. Da linha 2 à linha 55, temos o código CSS onde foi criado uma classe *button* e uma classe *button–loading*. Nessa classe é utilizada uma regra de CSS chamada *@keyframes* para fazer sua animação. Das linhas 56 a 67, temos o código JavaScript, que identifica o clique no botão e troca o perfil de *buttton* para o *button–loding* por um determinado tempo. Por fim, da linha 70 a 74, temos o código HTML simples, que usa as *tags* <*md-button*> para o botão, <*i*> para o ícone de *refesh*, e <*spam*> para o texto do botão.

Ao clicar nesse botão, à primeira instância vai ser acionado a função de "Atualizar lista de Conexões". Por meio dela que uma mensagem MQTT é enviada para rede, e por um determinado tempo espera uma reposta dos dispositivos que estão ativos. Após o fim do tempo de espera da função anterior, a função "Lista Dispositivos"é acionada, mostrando assim uma tabela com os dispositivos cadastrados e as seguintes informações, como, nome, *Firmware*, ID do Serviço, IP e Conexão, que ilustra qual conexão está ativa.

Em seguida, é feito um filtro para criar um *dropdown* com os dispositivos, com suas conexões ativas. Para efetuar o *upload* é usado o nó da *node-red-contrib-ui-upload*, chamado de *"upload"*, e o *subflow*, chamado de *"Upload"* para fazer o envio do código para o dispositivo selecionado. Um nó *template* também foi customizado para gerar uma animação de *loading* enquanto o *upload* é feito. Ao final, é feito um botão customizado, que funciona como um *toggle*, que esconde e mostra a lista de serviços quando é clicado.

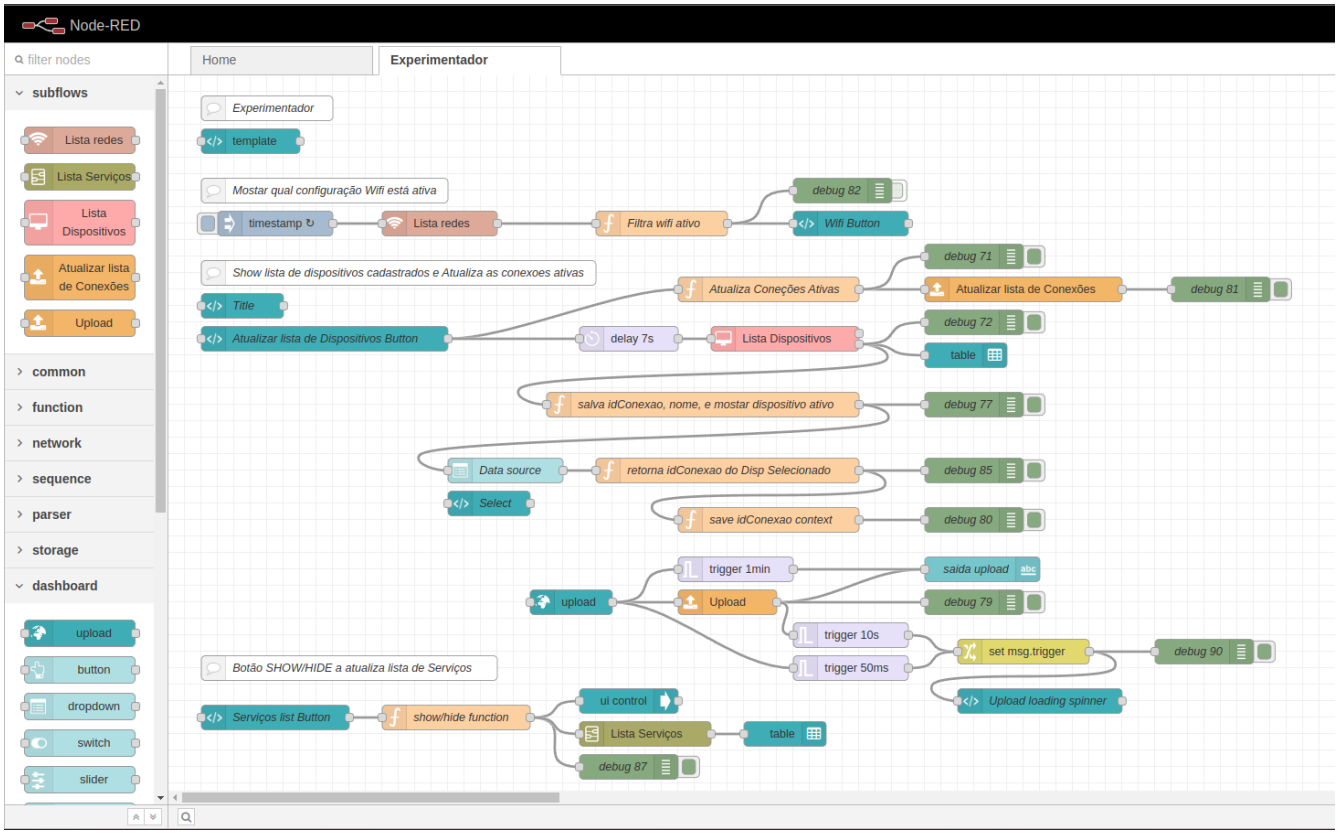

Figura 14 – Tela do *flwo-editor* com todas as funções usadas pelo perfil Experimentador

A *node-red-dashboard* disponibiliza uma forma fácil de criar uma interface gráfica para o Node-RED na web. São utilizados *Angular*, *Angular Material* e *JQuery*, na qual todos estão disponíveis para utilizar. Todos os nós disponibilizados pela *node-red-dashboard* são bem simples e fáceis de usar, entretanto, quando é necessário fazer algo mais customizado, igual o que foi realizado nas duas animações para o Experimentador, temos a necessidade de usar o nó *template*. Esse nó pode ser muito simples em alguns casos, no entanto, pode ficar bem complexo dependendo da situação. As animações para o Experimentador, por exemplo, se trata de um caso complexo, pois para utilizar um *script* em Javascript usando o nó *template* para acessar um objeto *msg* do navegador, é preciso usar a função *function(scope)*[4], que pode ser visto na Listagem 3.1, presente na linha 57.

[4] <https://flows.nodered.org/flow/2f1aaf0635f9bf23207152682323240a>

Listagem 3.1 – *Template* usado para o botão de atualizar Dispositivos.

```
<head>
<style>
    .button {
        position: relative;
        padding: 8px 16px;
        background: #097479;
        border: none;
        outline: none;
        border-radius: 12px;
        cursor: pointer;
    }

    .button:active {
        background: #007a63;
    }
    .button__text {
        transition: all 0.2s;
        position: absolute;
        top: 1px;
        left: 70px;
        color: #ffffff;
    }
    .button--loading .fa_customBT{
        visibility: hidden;
        opacity: 0;
    }
    .button--loading::after {
        content: "";
        position: absolute;
        width: 16px;
        height: 16px;
        top: 5px;
        left: 20px;
        border: 4px solid transparent;
        border-top-color: #ffffff;
        border-radius: 50%;
        animation: button-loading-spinner 1s ease infinite;
    }
    .fa_customBT{
        position: absolute;
        top: 5px;
        left: 20px;
        color: #00BFFF;
        transition: all 0.2s;
    }
    @keyframes button-loading-spinner {
        from {
            transform: rotate(0turn);
        }

        to {
            transform: rotate(1turn);
        }
    }
</style>
<script>
    (function(scope) {
        scope.toggleLoding = function(){
            const botaoNovo = document.querySelector(".button");
            botaoNovo.classList.toggle("button--loading");
            scope.send({payload:"att"});
            setTimeout(function() {
                botaoNovo.classList.toggle("button--loading");
            }, 8000);
        }
    })(scope);
</script>
</head>
<body>
    <md-button class="button" title="Atualizar lista de Dispositivos" ng-click="
        toggleLoding()">
        <i class="fa fa-refresh fa_customBT fa-2x" ></i>
        <spam class="button__text"> Dispositivos</spam>
    </md-button
</body>
```

4 Estudo de Casos

4.1 Metodologia

O projeto tem como objetivo integrar o Node-RED ao ambiente de experimentação OTALab, utilizando-o como camada intermediária entre a API e o usuário e gerando também um *frontend* para a aplicação. Embora fosse desejável realizar testes de usabilidade com usuários reais para avaliar o *frontend*, infelizmente, não foi possível devido à falta de tempo e recursos humanos. Assim, foram realizados testes de desempenho da ferramenta.

Durante a semana do CT de portas abertas[1], o projeto OTALab com a implementação Node-RED esteve em exibição no Laboratório de Redes e Multimídia (LPRM) durante os dias 22 e 23 de novembro de 2022, como parte da mostra de profissões da UFES. Os códigos dos Apêndices 1 e 2 foram remotamente carregados para as demonstrações. O código do Apêndice 1 permitia acender e apagar o LED de um ESP8266 por meio de comandos enviados a um bot no Telegram, enquanto o código do Apêndice 2 monitorava e exibia em gráficos em tempo real a temperatura, altitude e pressão com o uso do sensor BMP280 e um ESP8266.

Inicialmente, o código do Telegram foi carregado em 5 ESP8266, conectados a um broker público da HiveMQ em "http://broker.mqtt-dashboard.com/". Durante a demonstração, foi observado que havia certa demora na propagação das mensagens para todos os ESP8266 quando estavam conectados simultaneamente. Alguns testes foram realizados para verificar essa questão, e os resultados serão apresentados na seção subsequente.

Para avaliar o desempenho do sistema, foi configurado um cenário de teste descrito na Figura 15. O sistema incluía um computador equipado com o *broker* Mosquitto local, o Node-RED, o Arduino IDE e o OTALab. Todos esses dispositivos estavam conectados à internet via Wi-Fi a um roteador/modem, e um smartphone com o aplicativo do Telegram também estava na mesma rede Wi-Fi. Além disso, todos os dispositivos ESP estavam conectados à mesma rede Wi-Fi.

Foram realizados três tipos de testes diferentes. O primeiro teste foi projetado para avaliar a diferença no tempo de resposta para acender e apagar um LED por meio de um comando pelo Telegram, utilizando um *broker* público e local. Neste caso, o Mosquitto atuou como o broker local e o Node-RED foi usado para receber a mensagem enviada pelo usuário no smartphone através do Telegram. A mensagem então foi encaminhada via MQTT para o *broker* público HiveMQ ou para o *broker* local Mosquitto. O tempo de resposta

[1] <https://informatica.ufes.br/pt-br/conteudo/ct-portas-abertas>

estava relacionado com a velocidade da internet e dependia também da disponibilidade e tráfego de acesso ao *broker* público HiveMQ. Estes testes foram realizados usando uma conexão de internet VIVO em fibra óptica com 200Mbs de velocidade.

O segundo teste foi realizado para avaliar o tempo necessário para fazer o *upload* de um código para o ESP usando o OTALab e a Arduino IDE, com o ESP conectado ao computador via USB. Por fim, o terceiro teste foi o tempo necessário para registrar um dispositivo no OTALab, também com o ESP conectado ao computador via USB.

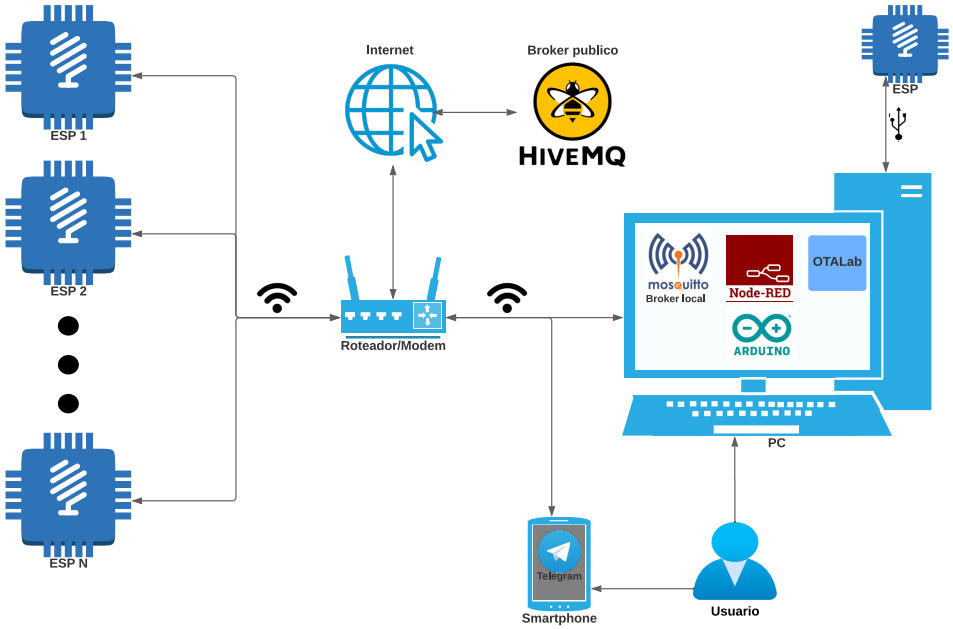

Figura 15 – Visão geral do esquema montado para os testes.

4.2 Resultados

Nesta seção serão apresentados os resultados obtidos e a análise de cada resultado. O tempo mostrado nas tabelas deu-se conforme o período que demorou para acender/apagar o LED do primeiro, do segundo, do terceiro ESP, e assim por diante. Isso mostra que o tempo do ESP1 representa o tempo para acender o primeiro LED, o do ESP2 é o tempo para acender o segundo LED e o ESP3 é o tempo para acender o terceiro e ultimo, caso o teste tenha apenas 3(três) ESP. Em vista disso, o tempo do último LED é sempre o tempo para que todos os ESP recebam a mensagem MQTT.

Assim, nas Tabelas 1 e 2 a seguir, torna-se possível observar os tempos em segundos para acender e apagar o LED do ESP após enviar a mensagem pelo *bot* do Telegram. Nesse caso, foram utilizados apenas dois ESP. A fim de verificar a diferença, foram efetuados

testes usando um *broker* local na Tabela 1 e um *broker* público na Tabela 2.

Assim é possível diferenciar, a média de tempo nos dois casos, com o *broker* local sendo bem mais rápido, quase a metade do tempo, que o público. Os dois gráficos ilustrando os valores das Tabelas 1 e 2 também podem ser visualizados na Figura 16 e na Figura 17, respectivamente.

Tabela 1 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com DOIS ESPs Ativos. Usando um *Borker* Mosquitto Local.

	Acender LED			Apagar LED	
	ESP 1	**ESP 2**		**ESP 1**	**ESP 2**
Teste 1	2.07	7.08		5.02	10.16
Teste 2	4.02	9.22		3.05	8.25
Teste 3	11.06	16.31		1.5	6.3
Teste 4	1.54	6.41		2.02	7.34
Teste 5	2.8	8.1		3.55	8.3
	Média:	**9.42**		**Média:**	**8.07**

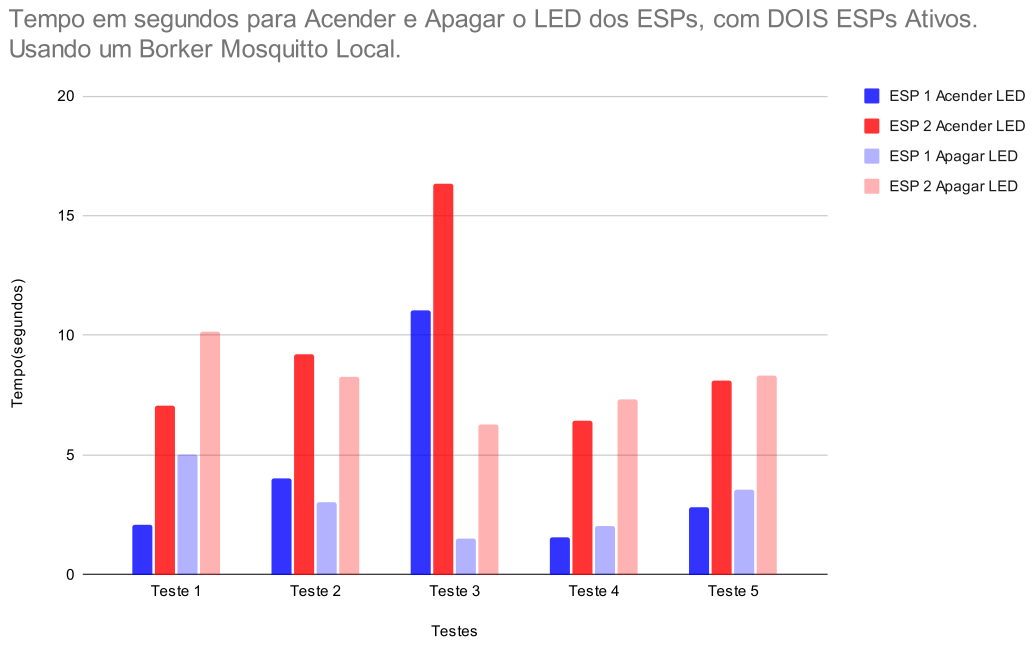

Figura 16 – Gráfico referente a Tabela 1.

Tabela 2 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com DOIS ESPs Ativos. Usando um *Borker* público HiveMQ "broker.mqtt-dashboard.com".

	Acender LED			Apagar LED	
	ESP 1	**ESP 2**		**ESP 1**	**ESP 2**
Teste 1	21.01	27.06		1.73	10.61
Teste 2	1.39	17.62		2.53	28.78
Teste 3	12.81	42.67		9.41	18.2
Teste 4	3.19	9.95		20.58	26.66
Teste 5	2.97	16.01		7.43	13.38
	Média:	**17.66**		**Média:**	**22.02**

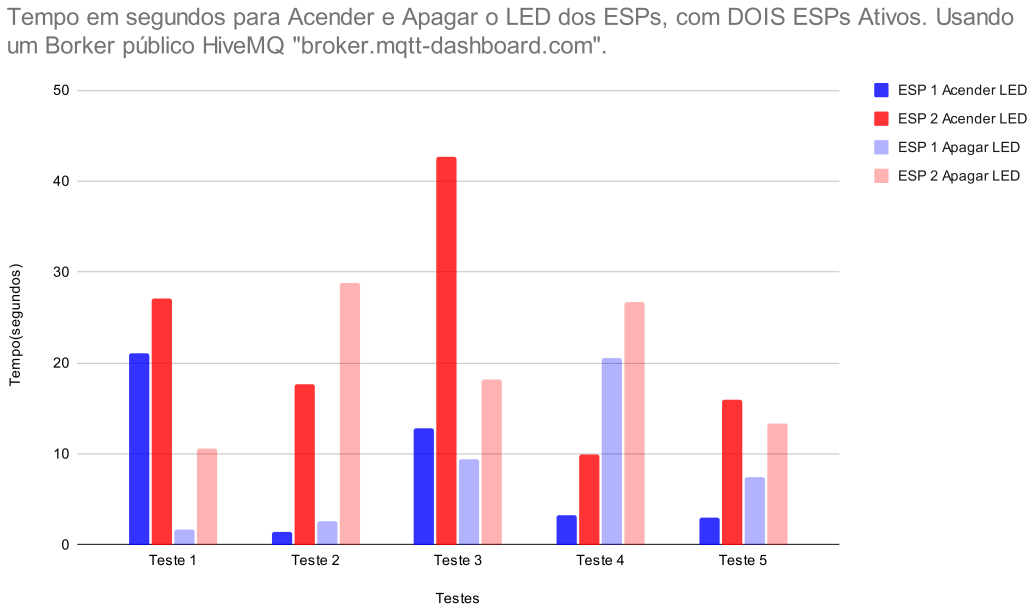

Figura 17 – Gráfico referente a Tabela 2.

Nas Tabelas 3 e 4 a seguir, observa-se a mesma medição de tempo e de resposta ao enviar a mensagem para o *bot* no Telegram, contudo, é adicionado mais um ESP, ficando assim com 3 (três) ESP no total. Aqui, é possível observar uma diferença ainda maior entre o *broker* público e local. Ainda, podemos observar também um que com o *broker* local e a diferença de tempo entre usar 3 ESPs ou 2 ESPs é pequena, tendo uma variação de apenas 4 (quatro) segundos. Já com *broker* público, onde é necessário usar a internet, a variação é muito maior, podendo demorar até 130 segundos para acender o LED dos 3 ESP.

Desse modo, é difícil mensurar, mas podemos inferir também que o tráfego na rede influencia no tempo quando é usado o *broker* público, pois este necessita da internet para fazer o encaminhamento das mensagens. Os dois gráficos ilustrando os valores das Tabelas 3 e 4 também podem ser visualizados na Figura 18 e na Figura 19, respectivamente.

Tabela 3 – Tabela de tempo gasto em segundos, para Acender e Apagar o LED dos ESPs, com TRÊS ESPs Ativos. Usando um *Borker* Mosquitto Local.

	Acender LED				Apagar LED		
	ESP 1	**ESP 2**	**ESP 3**		**ESP 1**	**ESP 2**	**ESP 3**
Teste 1	0.60	3.10	13.65		0.80	5.20	10.30
Teste 2	0.48	3.28	8.40		0.60	3.16	15.96
Teste 3	0.55	2.58	7.70		1.57	16.33	21.40
Teste 4	0.50	4.93	10.27		0.65	2.54	7.70
Teste 5	0.53	4.15	8.13		0.93	4.31	9.56
		Média:	**9.63**			**Média:**	**12.98**

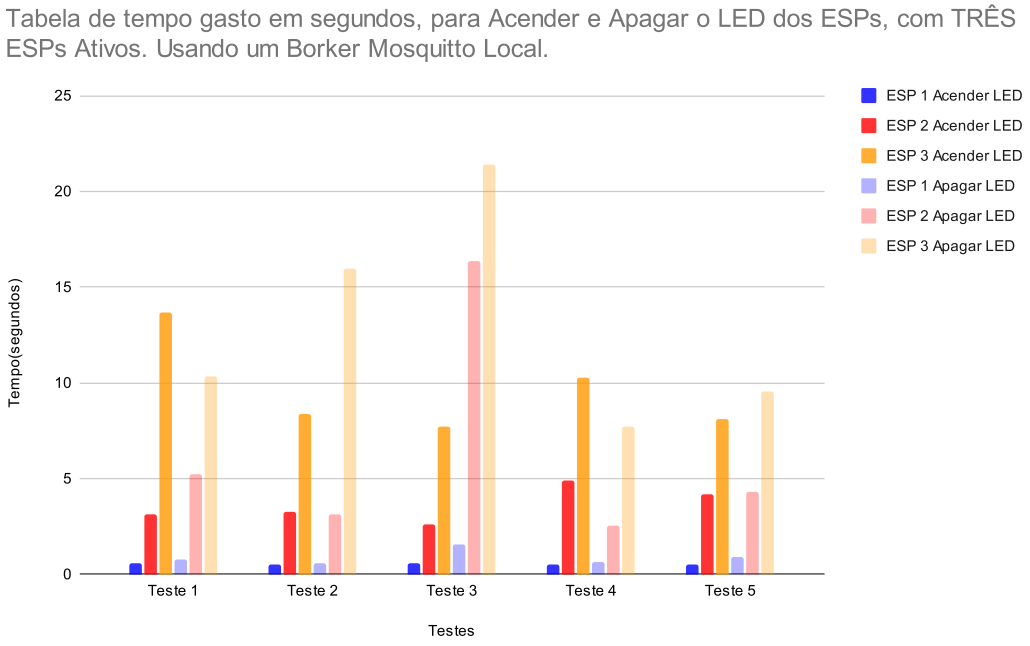

Figura 18 – Gráfico referente a Tabela 3.

Tabela 4 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com TRÊS ESPs Ativos. Usando um *Borker* público HiveMQ "broker.mqtt-dashboard.com".

	Acender LED				Apagar LED		
	ESP 1	**ESP 2**	**ESP 3**		**ESP 1**	**ESP 2**	**ESP 3**
Teste 1	61.77	105.13	130.71		20.96	27.22	75.07
Teste 2	41.93	48.45	68.06		37.95	51.66	64.09
Teste 3	31.65	78.65	116.94		16.15	22.17	55.18
Teste 4	2.86	15.16	23.14		3.34	23.57	94.99
Teste 5	23.63	30.15	49.08		11.08	19.88	59.38
		Média:	**77.59**			**Média:**	**69.74**

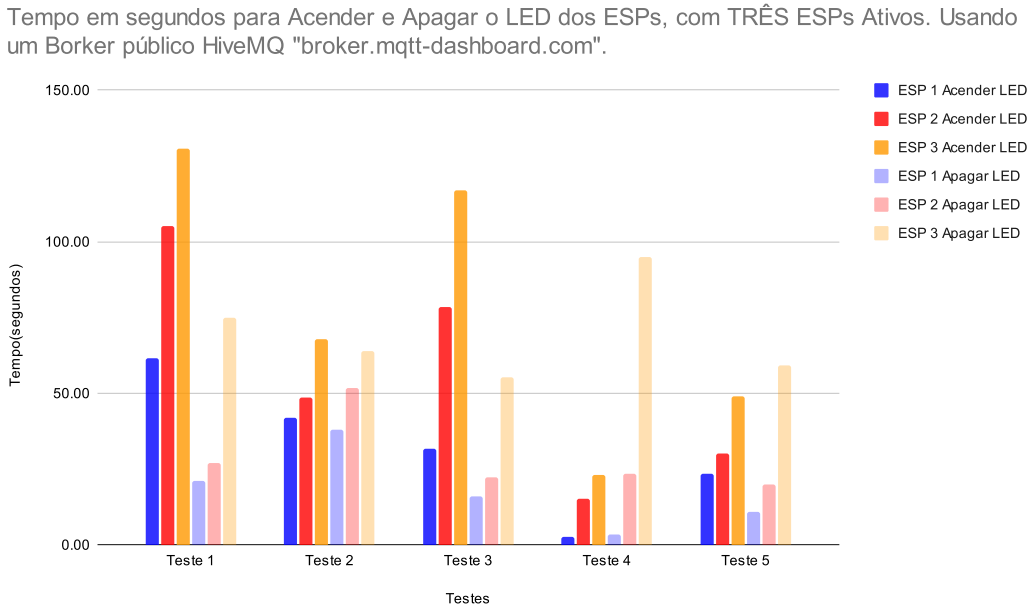

Figura 19 – Gráfico referente a Tabela 4.

Nas Tabelas 5 e 6 a seguir, é observado novamente a mesma medição de tempo, todavia, desta vez é adicionado mais um ESP, totalizando assim 4 ESP conectados simultaneamente. Assim, é possível observar que, com 4 ESPs há uma mudança muito grande de tempo, isto, tanto para o *broker* público quanto para o local. Este, variou em até 180 segundos com relação ao teste com apena 3 ESP. Com o público, variou em até 230 segundos com relação ao teste com 3 ESP.

Embora tenha ocorrido um aumento considerável do tempo, o *broker* local é ainda muito mais rápido que o público, com uma diferença de mais de 100 segundos entres eles, com o *broker* local podendo demorar até no máximo 302 segundos para todos os ESP receberem a mensagem MQTT para apagar o LED, e com o *broker* público podendo demorar até no máximo 546 segundos para acender o LED de todos os 4 ESP.

Vale destacar também que, com apenas um ESP conectado o tempo para acender e apagar o LED era muito pequeno, menos de 1 segundo. Isso foi tanto para o *broker* local quanto para o público. Os dois gráficos ilustrando os valores das Tabelas 5 e 6 também podem ser visualizados na Figura 20 e na Figura 21, respectivamente.

Tabela 5 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com QUATRO ESPs Ativos. Usando um *Borker* Mosquitto Local.

	Acender LED					Apagar LED			
	ESP 1	**ESP 2**	**ESP 3**	**ESP 4**		**ESP 1**	**ESP 2**	**ESP 3**	**ESP 4**
Teste 1	20.56	53.81	60.02	162.89		39.97	50.83	60.14	98.91
Teste 2	129.26	230.37	251.90	261.33		127.47	132.48	206.23	302.16
Teste 3	79.40	85.10	114.41	126.22		5.96	9.41	161.48	165.52
Teste 4	104.49	147.56	203.12	291.65		7.35	73.84	90.18	216.91
Teste 5	59.48	64.78	131.07	146.8		17.64	24.1	34.25	55.22
			Média:	**197.77**				**Média:**	**167.74**

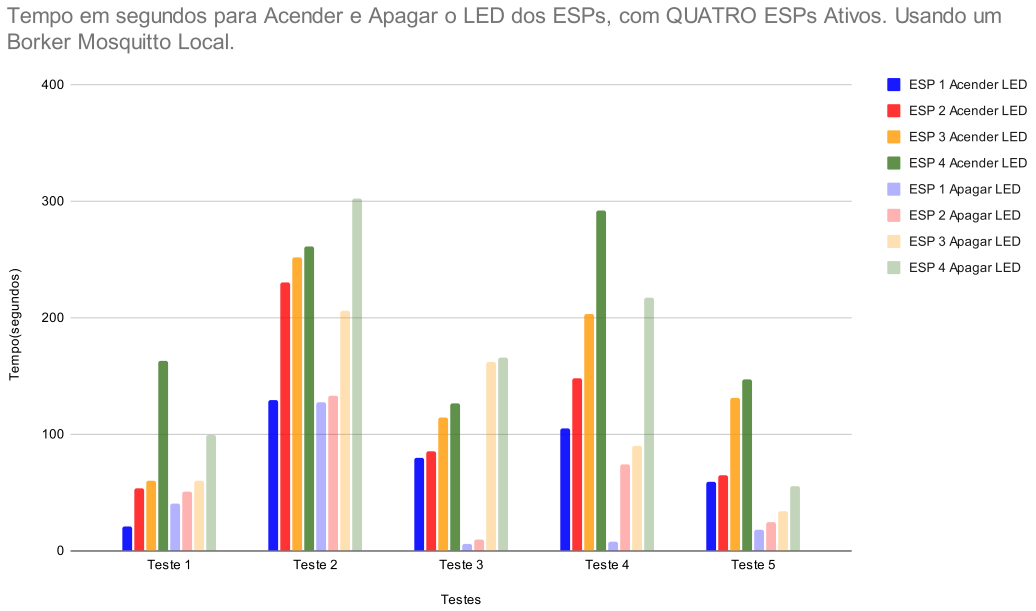

Figura 20 – Gráfico referente a Tabela 5.

Tabela 6 – Tempo em segundos para Acender e Apagar o LED dos ESPs, com QUATRO ESPs Ativos. Usando um *Borker* publico HiveMQ "broker.mqtt-dashboard.com".

	Acender LED					Apagar LED			
	ESP 1	**ESP 2**	**ESP 3**	**ESP 4**		**ESP 1**	**ESP 2**	**ESP 3**	**ESP 4**
Teste 1	28.88	34.85	241.06	546.32		24.87	40.16	145.21	208.07
Teste 2	59.3	71.92	198.83	231.4		167.22	193.12	332.06	373.26
Teste 3	40.66	111.54	281.45	288.84		36.01	81.57	146.92	307.22
Teste 4	25.03	34.05	159.13	187.53		2.53	47.87	67.26	182.69
Teste 5	29.16	35.76	94.39	287.53		34.14	54.08	132+16	158.97
			Média:	**308.32**				**Média:**	**246.04**

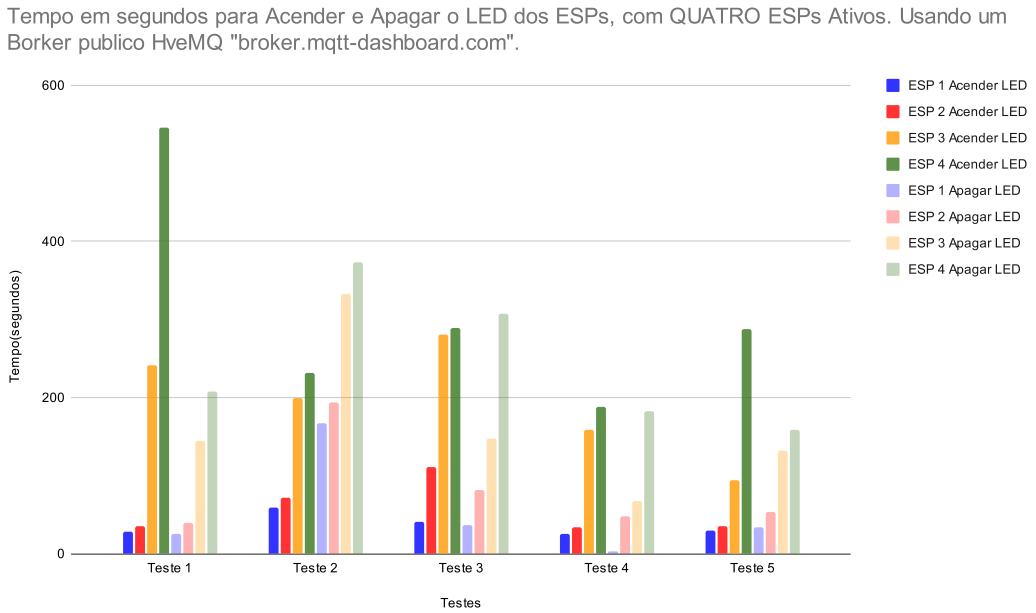

Figura 21 – Gráfico referente a Tabela 6.

Outra métrica foi analisada, no caso, o tempo de *upload* do OTALab, visando comparar o tempo para fazer *upload* de um código usando a ferramenta do OTALab pela interface do *dashboard-ui* criada com o Node-RED e o tempo para fazer *upload* usando a Arduino IDE.

Na Tabela 7 a seguir, é possível observar o tempo gasto para fazer o *upload* do código do Telegram que pode ser visto no apêndice 1 e na Tabela 8 para o código de Temperatura do Apêndice 2. Em ambas as tabelas é efetuado a comparação entre o tempo para *upload* via OTALab e usando a interface Arduino IDE.

Desse modo, se torna possível observar que o OTALab é mais lento para fazer *upload* dos dois códigos utilizados, entretanto esse resultado já era esperado e não gera grande desvantagem. Com ele, é apresentado variação entre os dois de no máximo 20 segundos, pois com o OTALab é possível fazer o *upload* remotamente e com Arduino IDE é necessário o ESP estar conectado no computador via um cabo USB. Os dois gráficos ilustrando os valores das Tabelas 7 e 8 podem ser visualizados na Figura 22 e na Figura 23, respectivamente.

Tabela 7 – Tempo em segundos para fazer *upload* do código do Telegram usando o OTALab e o Arduino IDE.

	Upload código Telegram	
	Arduino IDE	**OTALab**
Teste 1	11.45	22.08
Teste 2	10.90	23.11
Teste 3	10.87	33.58
Teste 4	11.05	32.69
Teste 5	11.32	44.67
Média	**11.12**	**31.23**

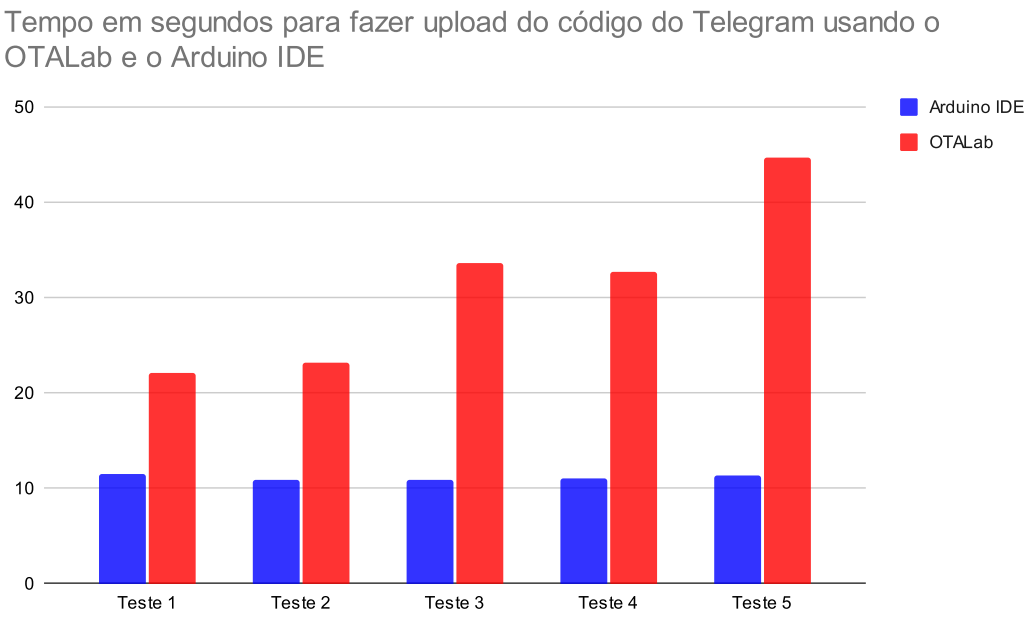

Figura 22 – Gráfico referente a Tabela 7.

Tabela 8 – Tempo em segundos para fazer *upload* do código de Temperatura usando o OTALab e o Arduino IDE.

	Upload código Temperatura	
	Arduino IDE	**OTALab**
Teste 1	12.83	26.67
Teste 2	12.91	27.11
Teste 3	13.00	25.97
Teste 4	12.62	24.27
Teste 5	12.79	25.87
Média	**12.83**	**25.98**

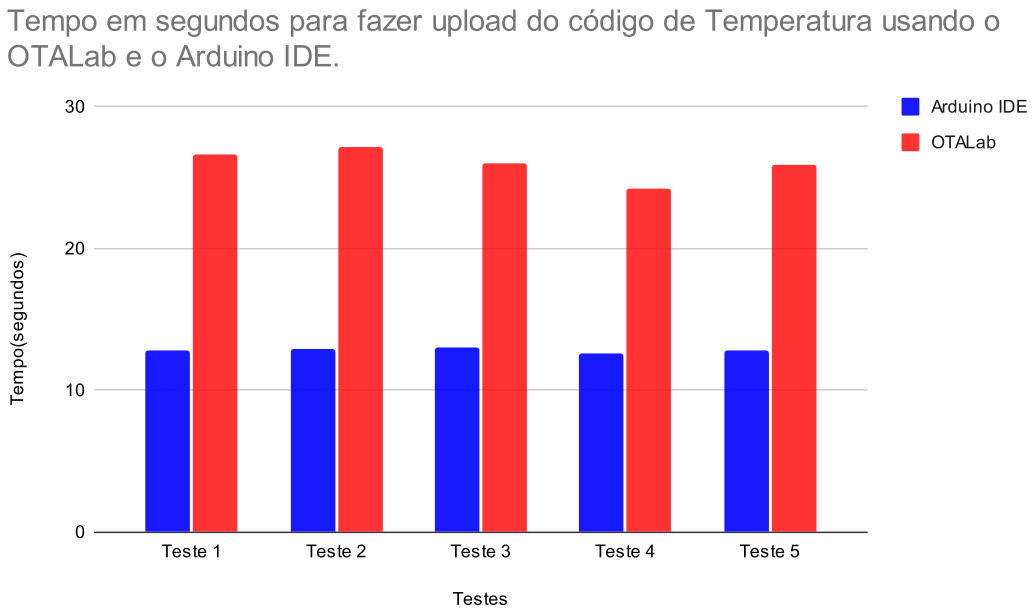

Figura 23 – Gráfico referente a Tabela 8.

Por fim, para utilizar todas as funcionalidades do OTALab é necessário que todos os ESPs estejam cadastrados na ferramenta. Levando isso em consideração, a última métrica foi uma análise de tempo para fazer o cadastro e o resultado pode ser visto na Tabela 9.

Esse cadastro é necessário ser realizado inicialmente, pois é ele quem inicia o dispositivo, fazendo *upload* do OTATemplate da Listagem 2.1 e armazenando todas as suas informações no banco MySQL para ser possível fazer o *upload* remotamente do código que futuramente desejar.

Assim, se torna possível observar na Tabela 9 que há certa oscilação no tempo, mas a média ficou em torno de 20 segundos, sendo um tempo baixo ao considerar o benefício que futuramente pode ser obtido ao fazer *uploads* remotamente para esse dispositivo. Um gráfico ilustrando os valores da Tabela 9 pode ser visualizado na Figura 24.

Tabela 9 – Tempo em segundos para cadastrar um dispositivo no OTALab.

	Cadastrar Dispositivo
Teste 1	26.17
Teste 2	12.69
Teste 3	40.87
Teste 4	22.93
Teste 5	12.67
Teste 6	12.65
Média	**21.33**

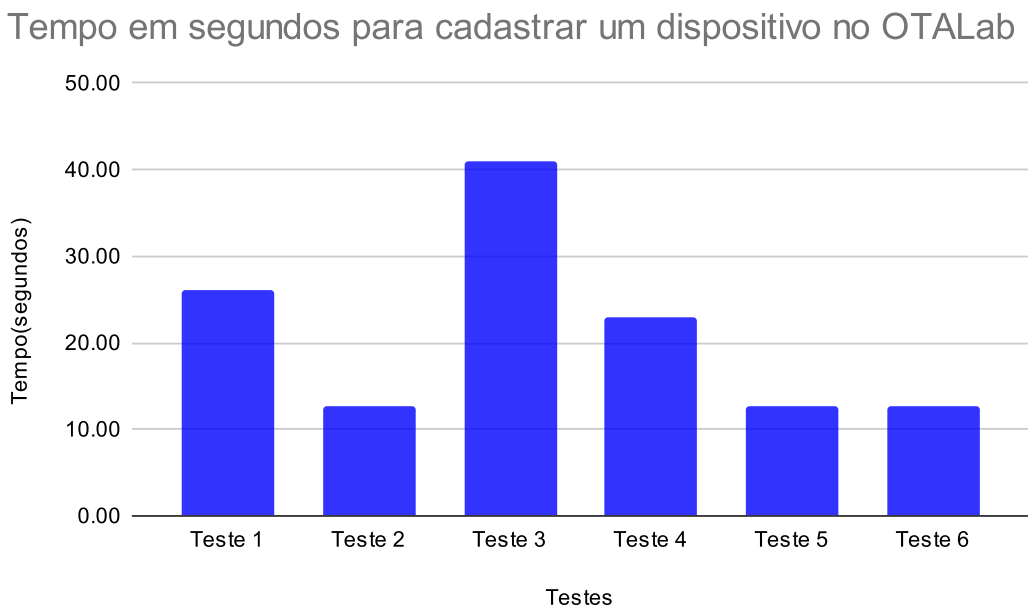

Figura 24 – Gráfico referente a Tabela 9.

5 Conclusão

5.1 Considerações Finais

Este trabalho apresenta a integração do Node-RED no ambiente de experimentação OTALab. Para tanto, foi proposto um novo design de arquitetura que amplia as funcionalidades anteriores. Devido ao crescimento das tecnologias e a proliferação de dispositivos de Internet das Coisas, tornou-se crucial poder fazer *upload* remotamente.

Por esse motivo, foi desenvolvido o projeto OTALab, que funciona como uma API REST construída em Java e que utiliza contêineres Docker para armazenar e executar todo o seu *backend*. As informações e dados são armazenados em um banco MySQL, que também está instalado em outro contêiner. Para usar o OTALab, basta fazer uma requisição a API REST, acessando algum dos *endpoints* disponíveis.

A integração do Node-RED trouxe uma contribuição significativa ao projeto OTALab. Neste caso, o Node-RED funciona como uma camada intermediária entre a API e o usuário, fornecendo também uma interface gráfica (*frontend*) para a aplicação.

A opção pelo Node-RED foi baseada na sua construção em Node.js, um ecossistema de código aberto que conta com uma comunidade extremamente ativa. Além disso, o Node-RED apresenta metadados em formato JSON e uma programação baseada em fluxo, tornando o desenvolvimento de aplicações mais visual e intuitivo. A facilidade de comunicação com os protocolos MQTT e HTTP para fazer requisições à API do OTALab é outro fator importante. O Node-RED possui também uma vasta biblioteca de fluxos e nós disponíveis, que podem ser utilizados não somente em projetos de IoT, mas também para se comunicar com um Bot no Telegram, como já foi mostrado nos testes. O desenvolvimento de aplicações e conexão de fluxos de dados podem ser realizados diretamente no navegador *web*.

Os resultados dos testes apontaram que quanto mais dispositivos ESP forem conectados à rede simultaneamente, mais demorado será o processo de propagação de mensagens para todos eles. Além disso, foi comparado o tempo de propagação usando um *broker* público (HiveMQ) e um *broker* local (Mosquitto), e constatou-se que o *broker* local é consideravelmente mais rápido em comparação com o público. Isso ocorre, pois o tempo de propagação também é influenciado pelo tráfego de rede e pela velocidade da internet, que, em geral, é mais lenta quando se usa um *broker* público. Finalmente, é importante destacar que o processo de *upload* pelo OTALab é mais lento do que pelo Arduino IDE, mas isso era esperado, uma vez que o *upload* pelo OTALab é realizado via *Over the air* (OTA).

5.2 Limitações

Uma das dificuldades encontradas foi com o *script* Python utilizado o spota.py[1]. Esse *script* é um código disponível no *github*, que faz o *upload* vi OTA, porém, em alguns casos, sobretudo com dispositivos ESP32, ocorre um erro de NO ANSWER, que pode ser visto na Figura 25.

Como esse código não foi escrito por ninguém envolvido no projeto OTALab, fica muito difícil corrigir algum *bug* ou adicionar novas funcionalidades.

Esse *script* também contribui para gerar uma das grandes limitações do projeto, que se refere à quantidade de dispositivos suportados.

```
Erro ao enviar o código submetido via OTA:
23:15:27 [DEBUG]: Options: {'esp_ip': '192.168.15.90', 'host_ip': '0.0.0.0', 'esp_port': 8266, 'host_port': 20000, 'auth': '', 'image': 'Blink/Blink.ino.bin',
'spiffs': False, 'debug': True, 'progress': False}
23:15:27 [INFO]: Starting on 0.0.0.0:20000
23:15:27 [INFO]: Upload size: 752224
23:15:27 [INFO]: Sending invitation to: 192.168.15.90
23:15:37 [ERROR]: No Answer
```

Figura 25 – Erro NO ANSWER ao tentar executar o *script* Python spota.py

Outro problema encontrado foi com BRLTTY, que se diz respeito a um processo em segundo plano (daemon), que é responsável por fornecer acesso ao console Linux/Unix (quando em modo texto) para uma pessoa com deficiência visual, usando um *display* braile atualizável. Ele aciona a exibição em braile e fornece a funcionalidade completa de revisão de tela. Alguns recursos de fala também estão disponíveis. Todavia, quando é utilizado o sistema Linux Mint Cinnamon 21, a função *brltty* sequestrava a entrada ttyUSB0, que estava conectada a um ESP

Assim, fazia com que a porta ttyUSB0 não fosse listada em */dev*, impossibilitando efetuar o cadastro do dispositivo no OTALab ou fazer qualquer *upload* para aquele ESP.

Na Figura 26, esse processo pode ser observado no *log* do *dmesg*. Assim, para solucionar o problema foi necessário realizar *sudo apt remove brltty* promovendo a desinstalação do *brltty*.

```
[ 9151.983703] cp210x 1-2:1.0: cp210x converter detected
[ 9151.985659] usb 1-2: cp210x converter now attached to ttyUSB0
[ 9153.682634] usb 1-2: usbfs: interface 0 claimed by cp210x while 'brltty' sets config #1
[ 9153.683638] cp210x ttyUSB0: cp210x converter now disconnected from ttyUSB0
[ 9153.683694] cp210x 1-2:1.0: device disconnected
```

Figura 26 – *Logs* do *dmesg* onde o *driver brltty* para Braile "sequestra"a entrada ttyUSB0

Outra dificuldade fora encontrada para gerar um *script* Javascrip no nó *template* do *dashboard-ui*. Isso aconteceu porque para acessar um objeto *msg* do navegador é preciso

[1] <https://github.com/esp8266/Arduino/blob/master/tools/espota.py>

usar a função *function(scope)*[2], que pode ser visualizada na Listagem 3.1, presente na linha 57. Isso pode ser conferido na animação de *loading*, no botão de atualizar dispositivos e ao realizar o *upload*.

5.3 Trabalhos Futuros

Após a realização deste trabalho, espera-se que a ferramenta seja utilizada em ambientes de estudos e que pesquisadores e entusiastas de Internet das Coisas possam criar novos *testbeds* a partir do projeto exposto.

Uma possibilidade de trabalho futuro está atrelada a criar um código customizado que substitua o *script* Python(spota.py) utilizado para fazer o *upload* via OTA. Assim, poderá tornar a ferramenta mais customizada e facilitará a resolução de problemas, como o *NO ANSWER* citado acima, que acontece em alguns casos com os dispositivos ESP32.

Ainda, tornará mais simples a implementação de novas funcionalidades e compatibilidades com outros dispositivos. Em conjunto com isso, outro trabalho a ser desenvolvido é a ampliação de dispositivos suportados pelo OTALab.

Por fim, outro trabalho futuro é a realização de teste de usabilidade e interface de usuário, com alunos de matérias relacionadas a IoT.

[2] <https://flows.nodered.org/flow/2f1aaf0635f9bf23207152682323240a>

Referências

ADJIH, C. et al. Fit iot-lab: A large scale open experimental iot testbed. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2015. p. 459–464. Citado na página 20.

APPAVOO, P. et al. Indriya2: A heterogeneous wireless sensor network (wsn) testbed. In: *Testbeds and Research Infrastructures for the Development of Networks and Communities*. [S.l.]: Springer International Publishing, 2019. p. 3–19. Citado na página 20.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 2010. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128610001568>. Citado na página 12.

BANGA, G.; DRUSCHEL, P.; MOGUL, J. C. Resource containers: A new facility for resource management in server systems. In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. USA: USENIX Association, 1999. (OSDI '99), p. 45–58. ISBN 1880446391. Citado na página 19.

CUSSUOL, E. B.; MOTA, V. F. S. Relatório do Projeto Disciplinar de IoT - OTALab: um ambiente de experimentação remota de protocolos e aplicações em Internet das Coisas. 2022. Citado 5 vezes nas páginas 5, 20, 21, 22 e 23.

CUSSUOL, E. B. et al. OTALab: Implantando um ambiente de experimentação remota de protocolos e aplicações em Internet das Coisas . SRBC, 2022. Citado 3 vezes nas páginas 12, 21 e 23.

El Jaouhari, S.; BOUVET, E. Secure firmware over-the-air updates for iot: Survey, challenges, and discussions. *Internet of Things*, v. 18, p. 100508, 2022. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2542660522000142>. Citado na página 17.

FOOTE, K. D. A brief history of the internet of things. dataversity, 2022. Citado na página 16.

HAGINO, T.; O'LARY, N. *Pratical Node-RED Programing*. [S.l.]: Packt, 2021. Citado 4 vezes nas páginas 5, 26, 27 e 28.

LOTTIAUX, R.; MORIN, C. Containers: a sound basis for a true single system image. In: *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. [S.l.: s.n.], 2001. p. 66–73. Citado na página 19.

MATTERN, F.; FLOERKEMEIER, C. From the internet of computers to the internet of things. In: _____. *From Active Data Management to Event-Based Systems and More: Papers in Honor of Alejandro Buchmann on the Occasion of His 60th Birthda*. [S.l.]: Springer Berlin Heidelberg, 2010. p. 242–259. Citado na página 16.

MORIN, C. et al. Towards an efficient single system image cluster operating system. In: *Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings.* [S.l.: s.n.], 2002. p. 370–377. Citado na página 19.

MORRISON, J. Data responsive modular interleaved task programming system. IBM Technical Disclosure Bulletin, v. 13, n. 8, 1971. Citado na página 24.

MORRISON, J. P. *Flow Base Programing: A New Approach to Application Development*. [S.l.]: Van Nostrand Reinhold, 1994. Citado 3 vezes nas páginas 5, 25 e 26.

PATEL, K. et al. Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application and future challenges. IJESC, 2016. Citado na página 16.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 17, n. 7, p. 412–421, jul 1974. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/361011.361073>. Citado na página 19.

PRICE, D.; TUCKER, A. Solaris zones: Operating system support for consolidating commercial workloads. In: *Proceedings of the 18th USENIX Conference on System Administration*. USA: USENIX Association, 2004. (LISA '04), p. 241–254. Citado na página 19.

RANDAL, A. The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 1, feb 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3365199>. Citado na página 19.

SALVADOR, O. Over-the-air (ota) update — solution for remote firmware updates. dataversity, 2019. Citado na página 17.

SCHOENBERGER, C. The internet of things. Forbes Magazine, 2002. Citado na página 16.

TURNBULL, J. *The Docker Book*. James Turnbull, 2014. Disponível em: <https://dockerbook.com/>. Citado 2 vezes nas páginas 19 e 20.

WERNER-ALLEN, G.; SWIESKOWSKI, P.; WELSH, M. Motelab: a wireless sensor network testbed. In: *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* [S.l.: s.n.], 2005. p. 483–488. Citado na página 20.

Apêndices

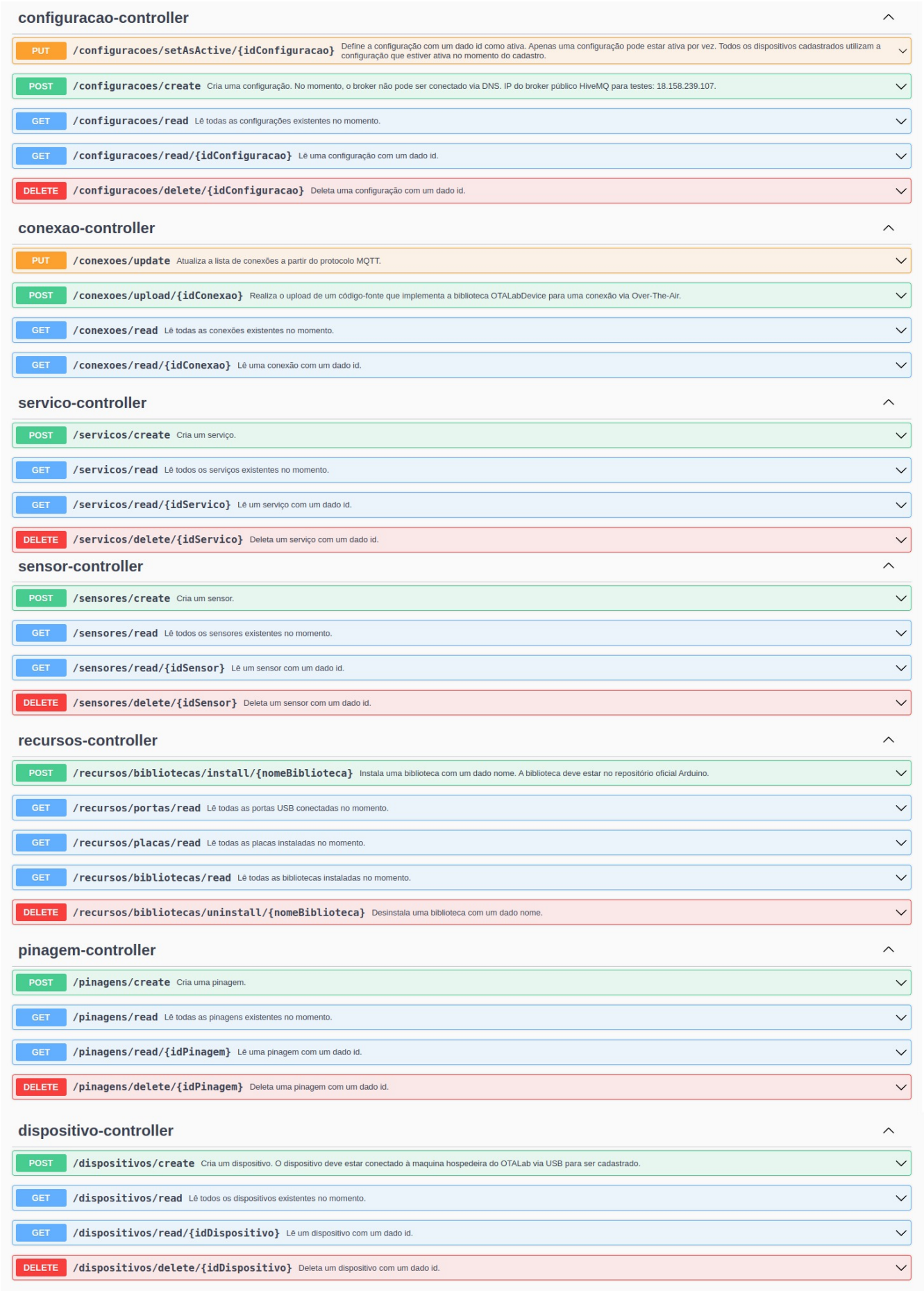

Figura 27 – Lista de *endpoints* disponibilizados pela API do OTALabBackEnd

Listagem 1 – Código da aplicação para acender e apagar LED do ESP pelo Telegram.

```
#include <OTALabDevice.h>

String id = "15";
OTALabDevice* device = new OTALabDevice();
const char* mqtt_server = "broker.mqtt-dashboard.com";
const int lamp = 2;
WiFiClient espClient;
PubSubClient client(espClient);

void callback(String topic, byte* message, unsigned int length) {
  Serial.print("Chegou uma mensagem no tópico: ");
  Serial.print(topic);
  Serial.print(". Mensagem: ");
  String messageTemp;

  for (int i = 0; i < length; i++) {
    messageTemp += (char)message[i];
  }
  Serial.println(messageTemp);
  if(topic=="tcc/iot/mqtt/led"){
      if(messageTemp == "acende" || messageTemp == "Acende"|| messageTemp == "Acender"||
          messageTemp == "acender"){
        Serial.print("Acendendo LED");
        digitalWrite(lamp, LOW);
      }
      else if(messageTemp == "apaga" || messageTemp == "Apaga" || messageTemp == "Apagar"
          || messageTemp == "apagar"){
        Serial.print("Apagando LED");
        digitalWrite(lamp, HIGH);
      }
  }
  Serial.println();
}

void reconnect1() {
  while (!client.connected()) {
    Serial.print("Tentando conexão MQTT...");
    if (client.connect("EspClientIot")) {
      Serial.println("connectado");
      client.subscribe("tcc/iot/mqtt/led");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  pinMode(lamp, OUTPUT);
  digitalWrite(lamp, LOW);
  Serial.begin(115200);
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
  device->setup(id);
}

void loop() {
  if (!client.connected()) {
    reconnect1();
    delay(5000);
  }
  if(!client.loop())
    client.connect("EspClientIot");
  device->handle();
}
```

Listagem 2 – Código da aplicação de monitoramento de temperatura com sensor BMP280.

```
#include <OTALabDevice.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_BMP280.h>
#define BMP_SCK  (13)
#define BMP_MISO (12)
#define BMP_MOSI (11)
#define BMP_CS   (10)
String id = "15";
OTALabDevice* device = new OTALabDevice();
Adafruit_BMP280 bmp;
char temp[10];
char pres[10];
char alti[10];
const char* mqtt_server = "broker.mqtt-dashboard.com";
const int lamp = 2;
WiFiClient espClient;
PubSubClient client(espClient);

void reconnect2() {
  while (!client.connected()) {
    Serial.print("Tentando conexão MQTT...");
    if (client.connect("EspClientIot")) {
      Serial.println("connectado");
      client.subscribe("tcc/iot/mqtt/led");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void setup() {
  device->setup(id);
  pinMode(lamp, OUTPUT);
  digitalWrite(lamp, LOW);
  Serial.begin(115200);
  Serial.println(F("BMP280 test"));
  client.setServer(mqtt_server, 1883);
  if (!bmp.begin(0x76)) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
    while (1);
  }
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL, Adafruit_BMP280::SAMPLING_X2,
      Adafruit_BMP280::SAMPLING_X16, Adafruit_BMP280::FILTER_X16, Adafruit_BMP280::
      STANDBY_MS_500);
}

void loop() {
    device->handle();
    Serial.print(F("Temperature = "));
    Serial.print(bmp.readTemperature()); Serial.println(" *C");
    Serial.print(F("Pressure = "));
    Serial.print(bmp.readPressure()); Serial.println(" Pa");
    Serial.print(F("Approx altitude = "));
    Serial.print(bmp.readAltitude(1013.25)); Serial.println(" m");
    dtostrf(bmp.readTemperature(),6,2,temp);
    dtostrf(bmp.readPressure(),6,2,pres);
    dtostrf(bmp.readAltitude(1013.25),6,2,alti);
    if (!client.connected()) {
      reconnect2();
      delay(5000);
    }
    if(!client.loop())
      client.connect("EspClientIot");
    client.publish("tcc/iot/mqtt/led",temp);
    delay(1000);
    client.publish("tcc/iot/mqtt/led",pres);
    delay(1000);
    client.publish("tcc/iot/mqtt/led",alti);
    Serial.println();
    delay(10000);
}
```