INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
BACHARELADO EM ENGENHARIA MECATRÔNICA

LEONARDO DOS SANTOS PACIFICO

USO DE PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO OU SEM USO DE CODIFICAÇÃO PARA O CONTROLE DE UMIDADE DE VASO DE PLANTAS

FLORIANÓPOLIS, DEZEMBRO DE 2022.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
BACHARELADO EM ENGENHARIA MECATRÔNICA

LEONARDO DOS SANTOS PACIFICO

USO DE PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO OU SEM USO DE CODIFICAÇÃO PARA O CONTROLE DE UMIDADE DE VASO DE PLANTAS

Trabalho de Conclusão de Curso Submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Bacharel em Engenharia Mecatrônica.

Professor Orientador: Dr. Eng. Roberto Alexandre Dias

FLORIANÓPOLIS, DEZEMBRO DE 2022.

Ficha de identificação da obra elaborada pelo autor.

Pacifico, Leonardo dos Santos
USO DE PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO OU SEM USO DE CODIFICAÇÃO PARA O CONTROLE DE UMIDADE DE VASO DE PLANTAS / Leonardo dos Santos Pacifico; orientação de Roberto Alexandre Dias. - Florianópolis, SC, 2022. 78 pg.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Campus Florianópolis. Bacharelado em Engenharia Mecatrônica. Departamento Acadêmico de Metal Mecânica.
Inclui Referências.

1. Plataforma de desenvolvimento de software. 2. Controle de Umidade. 3. Bubble.io. 4. NodeRED.
I. Dias, Roberto Alexandre. II. Instituto Federal de Santa Catarina. Departamento Acadêmico de Metal Mecânica.
III. USO DE PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO OU SEM USO DE CODIFICAÇÃO PARA O CONTROLE DE UMIDADE DE VASO DE PLANTAS .

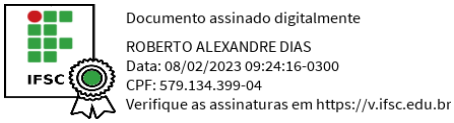
USO DE PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO OU SEM USO DE CODIFICAÇÃO PARA O CONTROLE DE UMIDADE DE VASO DE PLANTAS

Leonardo dos Santos Pacifico

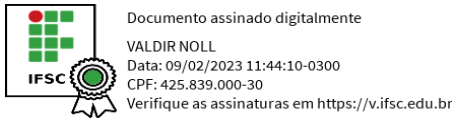
Este trabalho foi julgado adequado para obtenção do Título de Bacharel em Engenharia Mecatrônica e aprovado na sua forma final pela banca examinadora do Curso de Bacharelado em Engenharia Mecatrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 19 de dezembro de 2022.

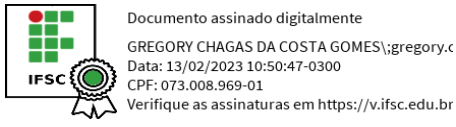
Banca Examinadora:



Roberto Alexandre Dias, Dr.Eng. (Orientador)



Valdir Noll, Dr. Eng.



Gregory Chagas da Costa Gomes. M. e Eng.

AGRADECIMENTOS

Primeiramente a Deus, por me fortalecer e abençoar durante todos os momentos da minha vida acadêmica.

Ao professor orientador, Dr. Eng. Roberto Alexandre Dias, pelo acompanhamento exemplar durante a orientação e formação no período da graduação.

Aos demais professores e colegas que estiveram comigo ao longo da minha formação, pelo conhecimento adquirido juntos e pela amizade construída.

Aos meus amigos de turma do IFSC, em que uns concluíram o curso e outros não, mas foi imprescindível o apoio de cada um, que tornaram muitos momentos difíceis mais leves. Em especial ao João Gabriel Miranda Lessa, “Joãozinho”, Luiz Felipe Scharf da Silva, “Scharf”, Sahmuel Melo Wiese Thiesen, “Samuca”, e Leonardo Gonçalves Rosa, “Léozinho”.

À minha noiva, Lidia Pacifico de Souza, por todo o amor, incentivo e dedicação ao longo da minha formação.

Aos meus pais, Eliane e José Luiz, pois devo a eles todo meu agradecimento e a honra de ter eles em minha vida. O apoio deles foi imprescindível durante toda minha jornada.

Na adversidade, uns desistem, enquanto outros batem recordes.

Airton Senna

RESUMO

No presente trabalho é descrito controle de umidade de vaso de plantas com *softwares* desenvolvidos por meio de plataformas com baixo uso ou sem uso de codificação e com *hardware* reaproveitado de projeto com características semelhantes, sendo que todos eles foram disponibilizados pelo orientador deste trabalho. O controle descrito é feito por sensores que verificam o nível de umidade do solo, nível de água no reservatório, acionamento de uma bomba que realiza a irrigação quando necessária, além de possibilitar comunicação com o usuário, via interface gráfica. O desdobramento deste trabalho consiste basicamente na integração entre *hardware* e *softwares* por meio dos conhecimentos adquiridos durante a graduação de Engenharia Mecatrônica. Assim, foi desenvolvida a interface gráfica com o uso de plataforma sem codificação Bubble.io e a plataforma com baixo uso codificação NodeRED para realizar a comunicação entre a interface e o *hardware*. Foi necessário adequar o *hardware* para a realização deste trabalho, introduzindo uma ESP8266 NodeMCU a fim de permitir a comunicação via *WiFi* necessária para integração entre *softwares* e *hardware*, os quais resultaram na montagem de um protótipo funcional.

Palavras-chave: Plataforma de desenvolvimento de *software.* Controle de umidade. Bubble.io. NodeRED.

ABSTRACT

In the present work, the development of a humidity control for potted plants is described, with software developed through platforms with low or no use of coding and hardware reused from a project with similar characteristics, made available by the advisor, which has sensors to verify the soil moisture level, water level in the reservoir, activation of a water pump that performs irrigation when necessary, in addition to providing a means of communication with the user, via a graphical interface. The developments of this work basically consist of the integration between hardware and software that will be done through the knowledge acquired during the graduation of Mechatronics Engineering. In this context, the graphical interface was developed using the platform without coding Bubble.io and the platform with low coding NodeRED to carry out the communication between the interface and the hardware. It was necessary to adapt the hardware for the application of this work, introducing an ESP8266 NodeMCU, in order to allow the communication through the WiFi necessary for the integration between software and hardware, which resulted in the assembly of a functional prototype.

Key words: Software development platform, Humidity control, Bubble.io, NodeRED.

LISTA DE FIGURAS

Figura 1 - Modelo de publish/subscribe.....21
Figura 2 - ESP 8266 NodeMCU e seus pinos 23
Figura 3 - Arduino Nano e seus pinos.....24
Figura 4 - Sensor de umidade.....25
Figura 5 - Fluxograma da proposta de projeto.....28
Figura 6 - Interface de desenvolvimento Bubble.io.....31
Figura 7 - Interface de criação do fluxo de trabalho 32
Figura 8 - Interface de criação do banco de dados…….....32
Figura 9 - Informações para a configuração da chamada de API…….....34
Figura 10 – Interface do aplicativo web.....35
Figura 11 - Mudança da cor de fundo da forma geométrica, quando selecionado ON/OFF...36
Figura 12 - Entrada de dados a serem enviados via chamada de API.....36
Figura 13 - Exibição dos dados recebidos via chamada de API.....37
Figura 14 - Registro de comandos.....37
Figura 15 - Início dos *workflows* de “estado da bomba”.....38
Figura 16 - Interface Node-red 40
Figura 17 - Tipo de *nodes* do Node-RED 41
Figura 18 - Propriedades payload e topic 43
Figura 19 - Estrutura de requisição HTTP request.....44
Figura 20 - Editor do *node* de HTTP request.....45
Figura 21 - Edição do *node* route.....46
Figura 22 - Estrutura do node *switch.*.....47
Figura 23 - Informações necessárias para o node *switch.*.....48
Figura 24 - “Function nodes” utilizados na API do projeto.....49
Figura 25 - Trecho código de “function node” executada na mensagem transportada.....50
Figura 26 - Rotas dos “function nodes” para o *node* MQTT.....52
Figura 27 - Informações contidas no node mqtt.....52
Figura 28 - Node mqtt command e conversor de linguagem.....53
Figura 29 - Propriedades do *node mqtt command*.....54
Figura 30 - Rotas *node* http *response*.....55
Figura 31 - Esquema de integração dos componentes do *hardware*.....56
Figura 32 - Esquemático elétrico do *hardware*.....57
Figura 33 - *Hardware*.....58
Figura 34 - Teste de sinal serial.....60

Figura 35 - Ponteira do osciloscópio na saída do redutor de tensão…….....61
Figura 36 - Sinal serial observado no osciloscópio……..…….....61

LISTA DE SIGLAS

API	Interface de Programação de Aplicação
HTTP	HyperTextTransferProtocol
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
TI	Tecnologia da Informação
URL	Uniform Resource Locator
V	Volt

SUMÁRIO

1 INTRODUÇÃO 13
1.1 JUSTIFICATIVA 14
2 OBJETIVOS 16
2.1 OBJETIVO PRINCIPAL 16
2.2 OBJETIVOS ESPECÍFICOS 16
3 FUNDAMENTAÇÃO TEÓRICA 17
3.1 PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE SEM O USO DE CODIFICAÇÃO 17
3.2 PLATAFORMAS DE DESENVOLVIMENTO DE SOFTWARE COM BAIXO USO DE CODIFICAÇÃO 18
3.3 API 19
3.4 MQTT BROKER 20
3.5 PROTOCOLO HTTP 21
3.6 NODEMCU ESP8266 22
3.7 ARDUINO NANO 23
3.8 SENSOR DE UMIDADE DO SOLO 24
3.9 COMUNICAÇÃO SERIAL 25
4 DESENVOLVIMENTO DA INTEGRAÇÃO DOS SOFTWARES COM O HARDWARE 27
4.1 APRESENTAÇÃO DO PROJETO 27
4.1.1 Bubble.io 29
4.1.1.1 Funcionamento do Bubble.io 30
4.1.1.2 Conector de API do Bubble.io 33
4.1.2 Chamadas de API 33
4.1.3 Criação de Interface 34
4.1.4 Criação do Fluxo do Trabalho (workflow) 37
4.1.5 Node-RED 39
4.2 DESENVOLVIMENTO DE API UTILIZANDO A PLATAFORMA NODE-RED 37
4.3 HARDWARE 55
4.4 TESTES DE COMUNICAÇÃO SERIAL 58
4.5 COMUNICAÇÃO ENTRE O NodeMCU ESP 8266 E A API 62
5 CONSIDERAÇÕES FINAIS 63
REFERÊNCIAS 65
APÊNDICE A – PROGRAMA UTILIZADO NO ARDUINO 68
APÊNDICE B – PROGRAMA UTILIZADO NA ESP 8266 72

1. INTRODUÇÃO

Novas soluções tecnológicas são necessárias atualmente devido às mudanças que acontecem com muita frequência e isso pode ser observado no avanço das organizações que auxiliam empresas por meio de aplicativos, tais como os para vendas de produtos e serviços. Por necessitarem de trocas de informações, acesso a bancos de dados ou realizar a integração entre sistemas, diversas empresas, devido a alta demanda e competitividade, seguem na busca contínua por reduzir o período de desenvolvimento dessas tecnologias, a fim de otimizar a produtividade e manter a qualidade de seus produtos (RECCHI, 2022). Com isso surgiu no mercado plataformas que visam facilitar o desenvolvimento de *softwares*.

No começo do ano de 2020 a grave crise sanitária da Covid-19, fez com que grande parte dos processos de produção fossem estagnados, pois as pessoas eram orientadas a não saírem de casa, a menos que fosse realmente necessário. Tudo isso fez com que muitos setores procurassem alternativas tecnológicas para continuar vendendo e atendendo à população, uma vez que boa parte dela estava mais tempo em casa trabalhando, entretendo-se e consumindo. Esse novo modo de viver impulsionou ainda mais o consumo de aplicativos, que tornam mais fáceis as atividades diárias, sem precisar sair de casa (FARIAS, 2022). Por outro lado, muitos que perderam seus empregos e tinham conhecimento para tanto viram como alternativa trabalhar por conta própria no ramo do comércio eletrônico, desenvolvendo *softwares* que permitissem interações entre pessoas, serviços e entretenimento. Tudo isso foi possível graças a essas plataformas de desenvolvimento (REPULLO, 2022).

As plataformas de desenvolvimento de *software* que serão utilizadas nesse trabalho permitem que usuários sem habilidades e técnicas de codificação criem *softwares*, pois essas plataformas utilizam elementos de programação visuais, que podem ser arrastados e organizados de maneira intuitiva pelo usuário, por meio de interface amigável. Elas permitem criar *softwares* rapidamente, com menor custo, mais flexibilidade e de acordo com a necessidade do usuário.

De acordo com o Gartner, 2021, 41% dos funcionários que não são da área de TI (Tecnologia da Informação) estavam envolvidos na personalização e

construção de *softwares*. No mesmo relatório prevê-se que até o final de 2025, metade de todos os novos clientes venham de fora da TI. Isso significa que a demanda por soluções digitais está em constante crescimento e, consequentemente, é cada vez mais fácil acessá-las, graças a essas plataformas de desenvolvimento (GARTNER, Inc, 2021).

Para compreender um pouco mais dessas plataformas de desenvolvimento de *software* como novas soluções tecnológicas, este trabalho é fundamentado em estudo de caso apresentado pelo professor orientador deste trabalho, que consiste em projeto de automatização de vaso para plantas. Dessa forma, descreve-se a integração de interface gráfica interativa, por meio de comunicação via internet, a *hardware* desenvolvido em projeto anterior, com os sensores necessários, uma vez que foi constatada semelhança em parte da aplicação. Como os sensores do *hardware* estavam atrelados a um arduino que não possui conexão com a internet previamente disponível, foi necessário adaptar uma ESP8266 NodeMCU conectado via serial com o arduino, para que fosse possível aplicar o *hardware* a esse trabalho. Essa foi a solução mais rápida e com o menor custo, tendo sido apresentada pelo professor orientador, uma vez que o *hardware* foi disponibilizado por ele.

1.1. JUSTIFICATIVA

É praticamente impossível imaginar nos dias de hoje alguma área que não tenha sofrido algum tipo de apoio tecnológico. Com isso, pode-se observar o crescimento das IoTs que possibilitam a integração de *hardwares* e *softwares*, isso significa que há a possibilidade de utilizar os mais variados padrões de dispositivos e, a partir dos dados coletados, transformar a dinâmica de tomada de decisão. Junto a isso, por meio de protocolos de comunicação via internet, é possível disponibilizar na rede esse dados para que possam ser utilizados em algum tipo de aplicação (PINTO, 2020).

No que se refere ao apoio da tecnologia em diversas áreas, é de grande relevância as tecnologias que estão presentes no cotidiano das pessoas e que facilitam as atividades, otimizando-as. Um exemplo disso é que atualmente muitas pessoas devido ao trabalho ou afazeres passam a maior parte do seu dia fora de

casa, além de, eventualmente, deixarem suas casas por longos períodos devido a viagens e férias, por exemplo. Por esse motivo, muitas plantas cultivadas em casa sofrem os efeitos da falta de água (MEDEIROS, 2018).

Uma alternativa para esse problema é a implementação de sistemas de irrigação de vasos de plantas existentes no mercado. No entanto, a maior parte dos sistemas de irrigação disponíveis utiliza temporizadores para ativar o sistema, como o modelo apresentado por Zazueta (ZAZUETA et al. , 1994), ou sistemas de controle mecânicos, gotejamento, o que dificulta o controle mais adequado da quantidade de água para cada tipo de planta, devido à falta de controle da umidade do solo.

O desenvolvimento de um sistema de controle de umidade de vaso de plantas permite controlar com mais precisão a umidade do solo necessária para cada tipo de planta. Esse sistema é útil, uma vez que o uso dessa tecnologia pode vir a ser de baixo custo e de fácil implementação, o que facilita esse tipo de tarefa para as pessoas que não têm muito tempo para cuidar de suas plantas ou simplesmente são adeptas a tecnologias. Desta forma, o presente trabalho procura fornecer uma alternativa para essa aplicação, o que permite também inserir os conhecimentos adquiridos ao longo do curso de Engenharia Mecatrônica, no que diz respeito à integração de *software* e *hardware*, com ênfase na utilização de plataformas de desenvolvimento de *software* com baixo uso ou sem uso de codificação.

2. OBJETIVOS

2.1. OBJETIVO PRINCIPAL

Desenvolver uma aplicação, que consiste no uso de plataformas de desenvolvimento de *software* com baixo ou sem uso de codificação para realizar o controle de umidade em vaso de plantas.

2.2. OBJETIVOS ESPECÍFICOS

- Desenvolver a programação do *front-end* utilizando plataforma de desenvolvimento de *softwar*e o uso sem codificação;
- Desenvolver a programação do *back-end* utilizando plataforma de desenvolvimento de software com baixo uso de codificação;
- Adequar o *hardware* para a aplicação neste projeto e realizar a integração com os *softwares*;
- Desenvolver testes de funcionalidade.

3. FUNDAMENTAÇÃO TEÓRICA

3.1. PLATAFORMAS DE DESENVOLVIMENTO DE *SOFTWARE* SEM O USO DE CODIFICAÇÃO

As plataformas de desenvolvimento de *software* sem o uso de codificação, também conhecidas como plataformas *no-code* (sem código), trazem consigo a ideia de não necessitar de profissionais desenvolvedores altamente qualificados. Sua forma de trabalho consiste basicamente em “arrastar e soltar”, bem como toda a parte de parametrização é de forma mais amigável, como também os formulários, a criação de regras e ações e a customização dos *layouts* (TABNEWS, 2022).

O surgimento dessas plataformas trouxe à tona um termo chamado de *citizen developer* (desenvolvedor cidadão), que consistem em usuários não técnicos na área de TI, mas conhecedores das necessidades e regras dos negócios. Eles assumem este papel devido a algum tipo de necessidade e se propõem a conhecer e estudar um pouco a respeito dessa forma de programação, o que permite-lhes criar, testar e implementar aplicações que realizam um determinado fluxo de trabalho e auxiliem a sua produtividade (BACELAR, 2022).

Entre as vantagens do uso desses tipos de plataformas está o fato de que elas permitem que as empresas tenham mais flexibilidade para criarem suas próprias aplicações móveis para os utilizadores. Economia de tempo, pois no estágio de desenvolvimento a modelagem óptica cria informações visualizadas do aplicativo e suas etapas. O custo é reduzido devido ao fato de que não há necessidade de programadores de *software* especializados, uma vez que qualquer indivíduo pode adquirir a expertise necessária para o desenvolvimento do *software* desejado (TABNEWS, 2022).

Vale a pena ressaltar que esse tipo de programação “sem código” não significa que realmente que não exista nenhum tipo de código na programação, na verdade as codificações mais complexas, que não são acessadas, já foram inseridas nos “blocos” pré-programados, feitos pelos criadores da plataforma, o que fica disponível é apenas a parte mais intuitiva (SANTOS, 2021).

Hoje há uma gama de plataformas de desenvolvimento de *software* sem o uso de codificação, que são utilizadas para diversas aplicações, cada uma delas com as suas peculiaridades. Para a elaboração desse projeto o professor orientador sugeriu que fosse utilizada a plataforma *no-code* Bubble.io, pois possui um domínio maior sobre ela e isso auxiliaria no desenvolvimento do projeto. Mesmo assim, foram analisadas outras opções de plataformas de desenvolvimento de *software* sem o uso de codificação, que fossem capazes de suprir as necessidades, atendendo o requisito de acesso gratuito tendo boas referências e bibliografias disponíveis.

Com uma diversidade de plataformas de desenvolvimento de *software* sem o uso de codificação, algumas das principais concorrentes da Bubble.io são as plataformas Adalo, Webflow, Appypie, Softr, Sharetribe, Glide (JURKÉNAS, 2022). No entanto, a sugestão de utilizar a plataforma Bubble.io feita pelo professor orientador atende muito bem as necessidades deste trabalho. Ela dispõe de uma versão de desenvolvimento de *software* gratuita, há cursos de desenvolvimento disponíveis sobre ela, apresenta método de construção interativo, prototipando de forma intuitiva e testando visualmente, além de permitir ótimas integrações.

3.2. PLATAFORMAS DE DESENVOLVIMENTO DE *SOFTWARE* COM BAIXO USO DE CODIFICAÇÃO

As plataformas de desenvolvimento de *software* com baixo uso de codificação, também conhecidas como plataformas *low-code* (baixo código), permitem que o usuário desenvolva aplicações de forma simples, rápida e com um baixo custo. Além disso, essas plataformas possuem uma baixa necessidade de manutenção, uma vez que essas ferramentas são previamente testadas e ambientadas antes que o desenvolvedor consiga operar dentro delas. Ainda nesse contexto, as plataformas *low-code* também possuem característica positiva de permitir maior controle sobre as permissões de acessos e implementação de recursos, facilitando a adequação dessas implementações às leis de proteção de dados (MARUTI TECHLABS, 2018).

As plataformas de desenvolvimento de *software* com baixo uso de codificação têm se tornado uma tendência em todo o mundo e estão ganhando cada vez mais espaço nas empresas. Com o avanço tecnológico elas são cada vez mais

evidentes e exigentes, pois permitem a criação de aplicativos com pouca programação de códigos, o que impacta na diminuição da dependência de programadores e profissionais técnicos, que são atualmente muito disputados e caros no mercado. Há quem compare o desenvolvimento de *softwares* por meio de plataformas *low-code* com montagens utilizando peças de um Lego, no qual o que é criado acontece de acordo com o manejo e encaixe de cada peça a fim de atingir aquilo que se deseja formar (PACETE, 2022).

Assim como nas plataformas *no-code*, o desenvolvimento consiste em “arrastar” e “alocar” os “blocos”. Em razão da parte principal do código já estar pronta, os usuários só precisam realizar configurações visuais ou fazer alguns ajustes necessários. Com isso, há redução no tempo total de desenvolvimento e maior velocidade na disponibilização do aplicativo (OUTSYSTEMS, 2019; RICHARDSON et al., 2016). O importante é que o resultado funcione e tudo se encaixe perfeitamente, se transformando em uma aplicação funcional. Segundo pesquisas realizadas pela Forrester Research, empresa que realiza estudos de mercado sobre o impacto existente e potencial da tecnologia, plataformas em *low-code* irão representar 75% do total de *softwares* desenvolvidos em 2024 (BONETTI, 2022).

Assim como na escolha da plataforma de desenvolvimento de software sem uso de codificação, o professor orientador deste trabalho sugeriu que fosse utilizada a plataforma de desenvolvimento de *software* com baixo uso de codificação Node-RED, pois havia um exemplo muito útil disponibilizado por COPE, 2020, o que traria agilidade e versatilidade ao programar, além de flexibilizar a criação da aplicação. Nesse caso, a plataforma de desenvolvimento de *software* com baixo uso de codificação Node-RED será utilizada para desenvolver a parte *back-end* do projeto, o qual consiste em uma API (Interface de Programação de Aplicação) que permite a integração entre aplicativos e *softwares*.

3.3. API

As APIs são um conjunto de rotinas e padrões de programação que têm como função conectar dois sistemas diferentes, sejam eles aplicativos ou *softwares.*

Elas possuem um conjunto de rotas, em que cada rota permite publicar um tópico no *broker*. É uma forma de simplificar o trabalho dos programadores, sem precisar criar novos códigos para atribuir as funções desejadas (De CODAR, 2022). As APIs têm um papel fundamental na programação distribuída, pois permite que qualquer cliente, com conhecimento sobre sua documentação, possa consumir sem ter que criar novamente a mesma lógica simplesmente aproveitando a já existente, evitando trabalhos redundantes e podendo aproveitar de serviços já consolidados e inovadores (OLIVEIRA, 2018).

A utilização das APIs tem se espalhado nos *plugins*, que complementam a funcionalidade de determinado programa. Os desenvolvedores de programa criam uma API específica e fornecem a outros criadores, que desenvolvem *plugins* para aumentar o potencial do programa (CANALTECH, 2014).

Neste trabalho, a API irá receber e enviar dados do tipo JSON e seu desenvolvimento foi realizado por meio da plataforma de desenvolvimento de *software* com baixo uso de codificação Node-RED.

3.4. MQTT BROKER

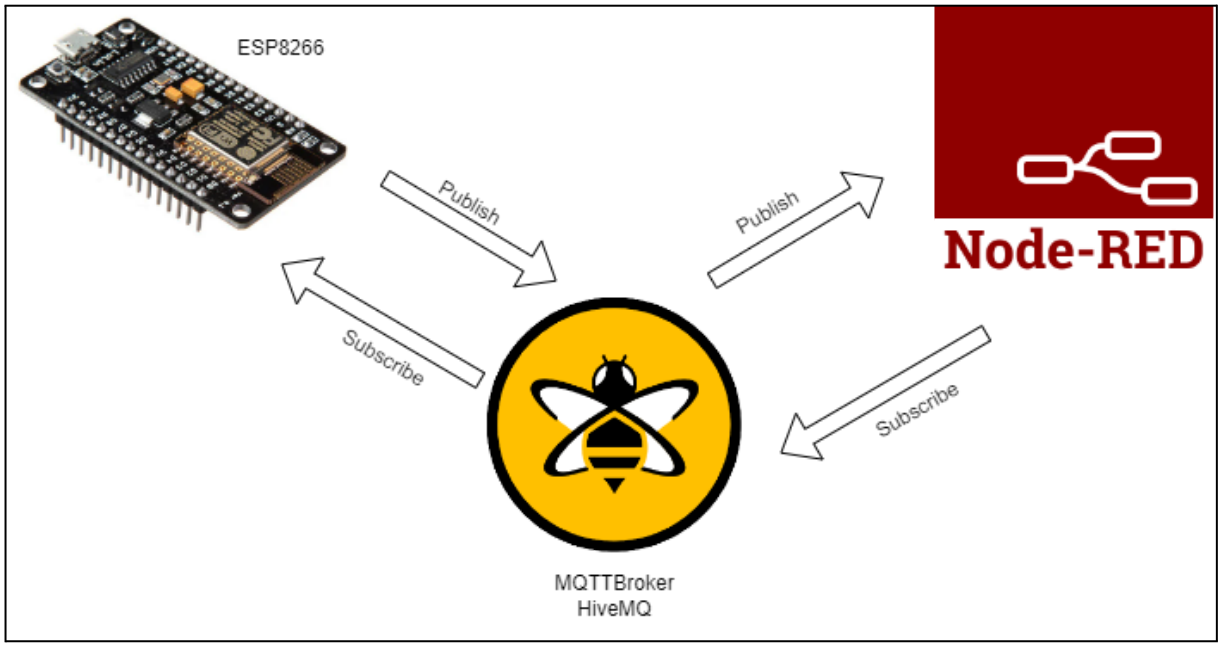
O MQTT é um protocolo que possui como características principais o baixo consumo de rede, banda e demais recursos, muito utilizado em comunicação de dispositivos IoT (*Internet of Things*). Baseia-se no formato Ciente/Servidor(*broker*) por meio do modelo de publicação/assinatura (*Publish/Subscribe)*, no qual os clientes podem realizar postagens ou receber informações enquanto o servidor administra esse envio e recebimento. Sendo assim, em um MQTT (*Message Queuing Telemetry Transport*) haverá o publicador, que será responsável por publicar as mensagens (cujo conteúdo é conhecido como *payload*) em um determinado tópico em que o assinante irá inscrever-se para poder acessar as mensagens (FREITAS, 2017).

O *broker* é uma espécie intermediário responsável por formar uma ponte de comunicação entre o *Publisher* e o *Subscriber*, que fará o recebimento, enfileiramento e envio de mensagens. O esquema funciona da seguinte forma: o cliente conecta-se ao *broker*, possibilitando-o assinar qualquer tópico (endereço para o qual os dados das mensagens serão enviados). Ele então publica uma mensagem

no tópico desejado, a qual será enviada ao servidor para o tópico no qual o conteúdo deve ser publicado. O *broker*, por sua vez, faz o encaminhamento da mensagem a todos os clientes que assinaram o referido tópico. As mensagens organizadas por tópicos permitem que se especifique quais clientes podem interagir com determinadas mensagens (LOCATELLI, 2020).

Para a escolha do MQTT Broker existem algumas opções disponíveis no mercado e, para esse trabalho em específico, o MQTT Broker selecionado para realizar a comunicação foi o HiveMQ, por ser uma plataforma de mensagens baseada em cliente projetada para a movimentação rápida, eficiente e confiável de dados de e para dispositivos IoT conectados.

Figura 1 - Modelo de publish/subscribe.



Fonte: Próprio autor.

3.5. PROTOCOLO HTTP

O protocolo HTTP pode ser considerado a base de funcionamento da internet, que consiste em requisições e respostas entre cliente e servidor. Um fluxo típico do HTTP envolve uma máquina cliente que faz uma solicitação a um servidor, o qua,l por sua vez, envia uma mensagem de resposta. Uma mensagem enviada pelo cliente é chamada de requisição (*request*), enquanto a mensagem enviada pelo servidor é chamada de resposta (*response*) (MELO, 2021).

Ao realizar uma requisição para o servidor, é necessário que na mensagem esteja definido qual método será utilizado para o envio dela. Os métodos HTTP são responsáveis por indicar qual a ação a ser realizada no servidor. O protocolo HTTP conta com nove métodos, mas os quatro mais utilizados são apresentados a seguir (OLIVEIRA, 2018):

- O método GET requisita um determinado recurso e deve ser usado apenas para recuperar dados, e não para modificar, sendo considerado como método seguro devido a essa sua característica;
- O método POST envia dados a serem processados no servidor e é utilizado para criar novas informações;
- O método PUT atualiza informações existentes;
- O método DELETE é o responsável pela exclusão de informações.

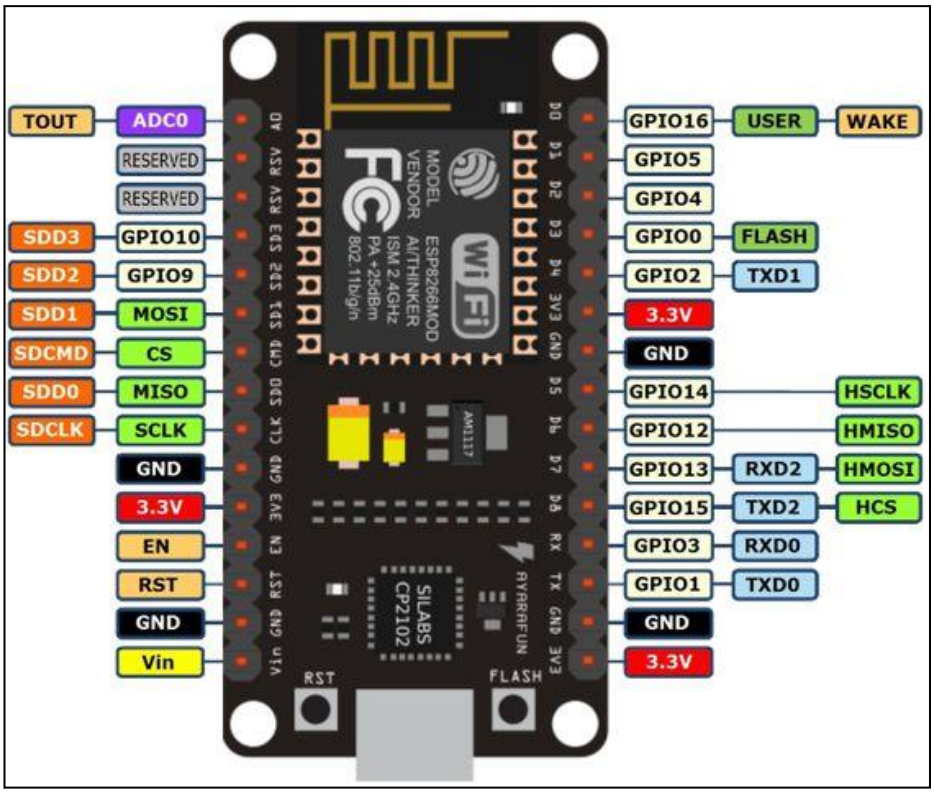
3.6. NODEMCU ESP8266

O ESP8266 é um microcontrolador que chegou ao mercado com um grande poder de processamento, possui sistema de comunicação WiFi permitindo que microcontroladores se conectem à internet sem a necessidade de cabeamento. Algumas das suas aplicações estão no controle de sensores, na robótica e na comunicação. Uma das suas grandes vantagens está diretamente ligada ao seu baixo custo quando comparado a outros microcontroladores (MOURA JUNIOR, 2020).

Para este projeto foi utilizado uma plataforma chamada NodeMCU integrada ao ESP8266, uma vez que o *hardware* disponibilizado pelo professor orientador não dispõe de comunicação via WiFi, a solução mais rápida e de baixo custo foi o NodeMCU ESP8266 ao *hardware*, além de dispor de uma porta micro USB, que facilita a alimentação e a programação.

O NodeMCU é uma plataforma que combinada com o microcontrolador ESP8266 facilita o desenvolvimento de aplicações de IoT. A programação do NodeMCU pode ser feita utilizando o programa Arduino IDE e a comunicação serial via cabo micro-usb para transferir a programação. Conforme apresentado na Figura 2, o NodeMCU possui pinos de comunicação serial Rx/Tx, além de 11 pinos de I/O e conversor analógico/digital (MOURA JUNIOR,2020).

Figura 2 - NodeMCU ESP8266 e seus pinos.

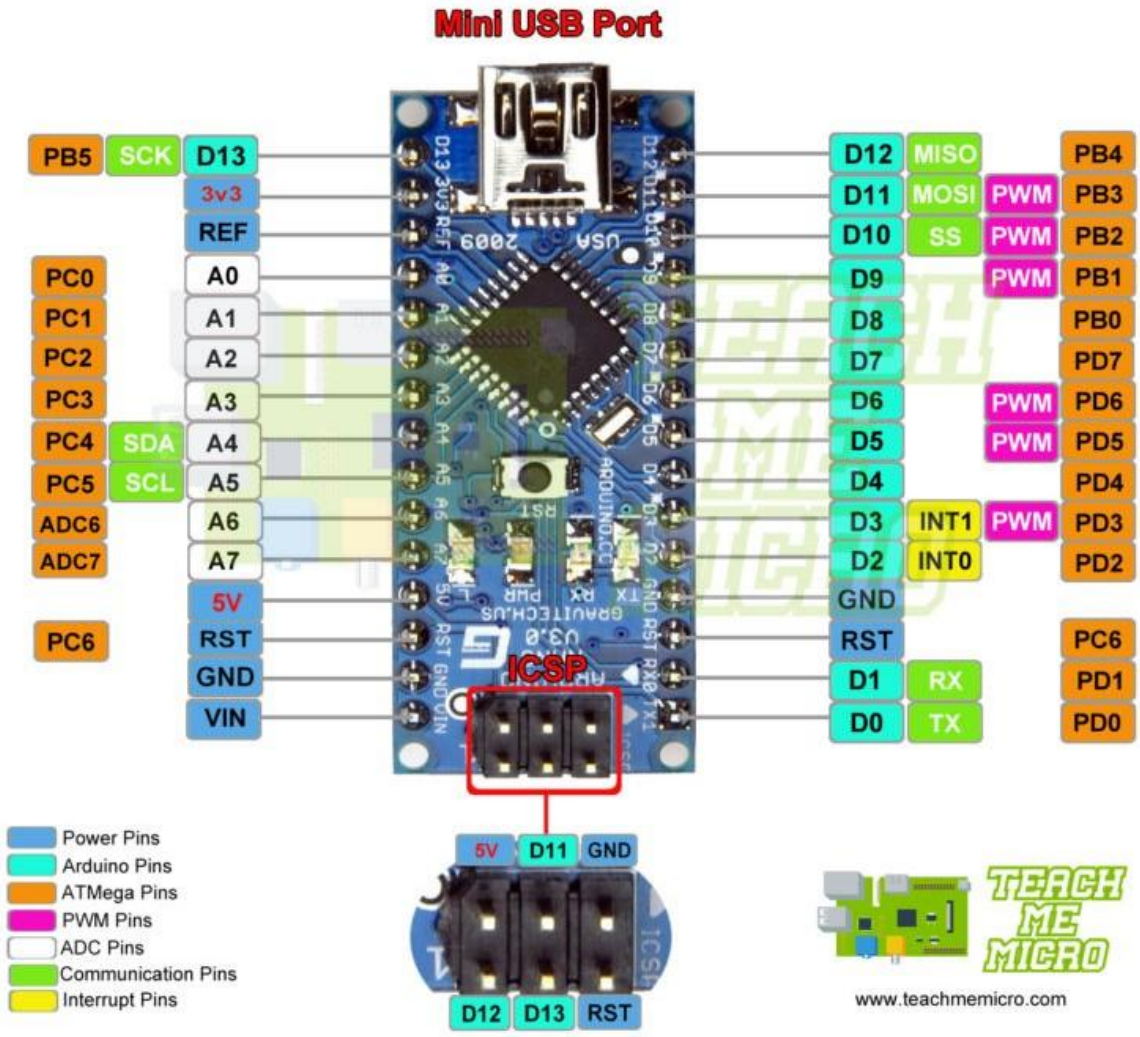


Fonte: (LOJA ESP8266, 2022).

3.7. ARDUINO NANO

O Arduino Nano (Figura 3) é a melhor escolha para projetos de restrição de dimensões físicas. O Nano (com processador ATmega328) tem uma mini-USB integrada, um formato compacto para uso em placas testes. A alimentação para a placa é fornecida por USB ou por dois pinos separados: o pino 30 pode receber uma tensão desregulada entre 6V e 20V, ou o pino 27 pode receber uma tensão regulada de 5,5V. A placa seleciona a tensão que for mais alta. Conta com pinos de comunicação serial Rx/Tx. O tamanho reduzido da placa faz com que seja ideal para projetos com espaço limitado (NOVATEC, 2013).

Figura 3 - Arduino Nano e seus pinos.



Fonte: (TEACH ME MICRO, 2022).

3.8. SENSOR DE UMIDADE DO SOLO

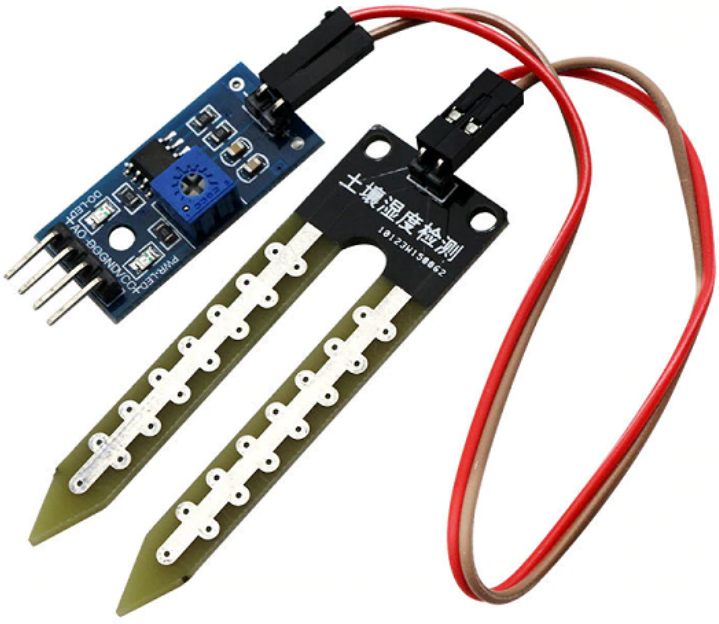
O sensor de umidade do solo apresentado na Figura 4 é ideal para fazer a leitura da umidade do solo onde estiver instalado e informar ao microcontrolador como um Arduino. Sendo assim, é necessário programar, por exemplo, o Arduino para utilizar a informação recebida do sensor e realizar alguma função, como, por exemplo, ativar o relé de acionamento de uma bomba d'água para irrigação. Quando o solo está seco a saída do sensor fica em estado alto / nível lógico 1, e quando o solo está úmido a saída do sensor fica em estado baixo / nível lógico 0 (OLIVEIRA, 2021).

O limite entre seco e úmido pode ser ajustado por meio do potenciômetro

presente no sensor que fará o ajuste do ponto exato em que a saída digital D0 alternará seu estado / nível lógico.

Este sensor utiliza duas pontas de prova para passar a corrente pelo solo e sua leitura é baseada na resistência elétrica resultante. Quanto mais água no solo, mais baixa é a resistência dele e mais fácil fica a condução entre as pontas de prova. Quando o solo está seco, a condutividade é baixa, logo a resistência é alta.

Figura 4 - Sensor de umidade.



Fonte: (Master Walker, 2021).

3.9. COMUNICAÇÃO SERIAL

A comunicação em série é o processo de enviar dados de *bit* em *bit,* sequencialmente. É um protocolo de comunicação muito utilizado por diversos dispositivos para instrumentação e também para aquisição de dados (FABRETTI, 2016).

O conceito de comunicação serial consiste em utilizar um canal de comunicação ou barramento de computador para enviar e receber *bytes* de

informação, um *bit* de cada vez, tornando-a mais simples e muito utilizada em distâncias maiores.

A velocidade deste tipo de comunicação, conhecida como *baud rate*, é dada em *bits* por segundo e varia de sistema para sistema (sabendo que alguns sistemas, como Arduinos, de acordo com suas configurações, podem aceitar diferentes velocidades) (PEREIRA, 2019).

Este projeto a comunicação serial será responsável por realizar a troca de informações entre o arduino e a placa NodeMCU ESP8266, além de permitir que as informações enviadas ou recebidas sejam apresentadas no monitor serial do Arduino IDE.

Cada comunicação serial possui um protocolo de comunicação para ser utilizado, no caso deste projeto o protocolo utilizado é o UART (Universal Asynchronous Receiver/Transmitter) do tipo RS232, que permite conectar o computador e seus dispositivos periféricos e realizar a troca de dados seriais entre eles à medida que obtém a tensão para o caminho utilizado para a troca de dados entre os dispositivos (CACPNRJ, 2020).

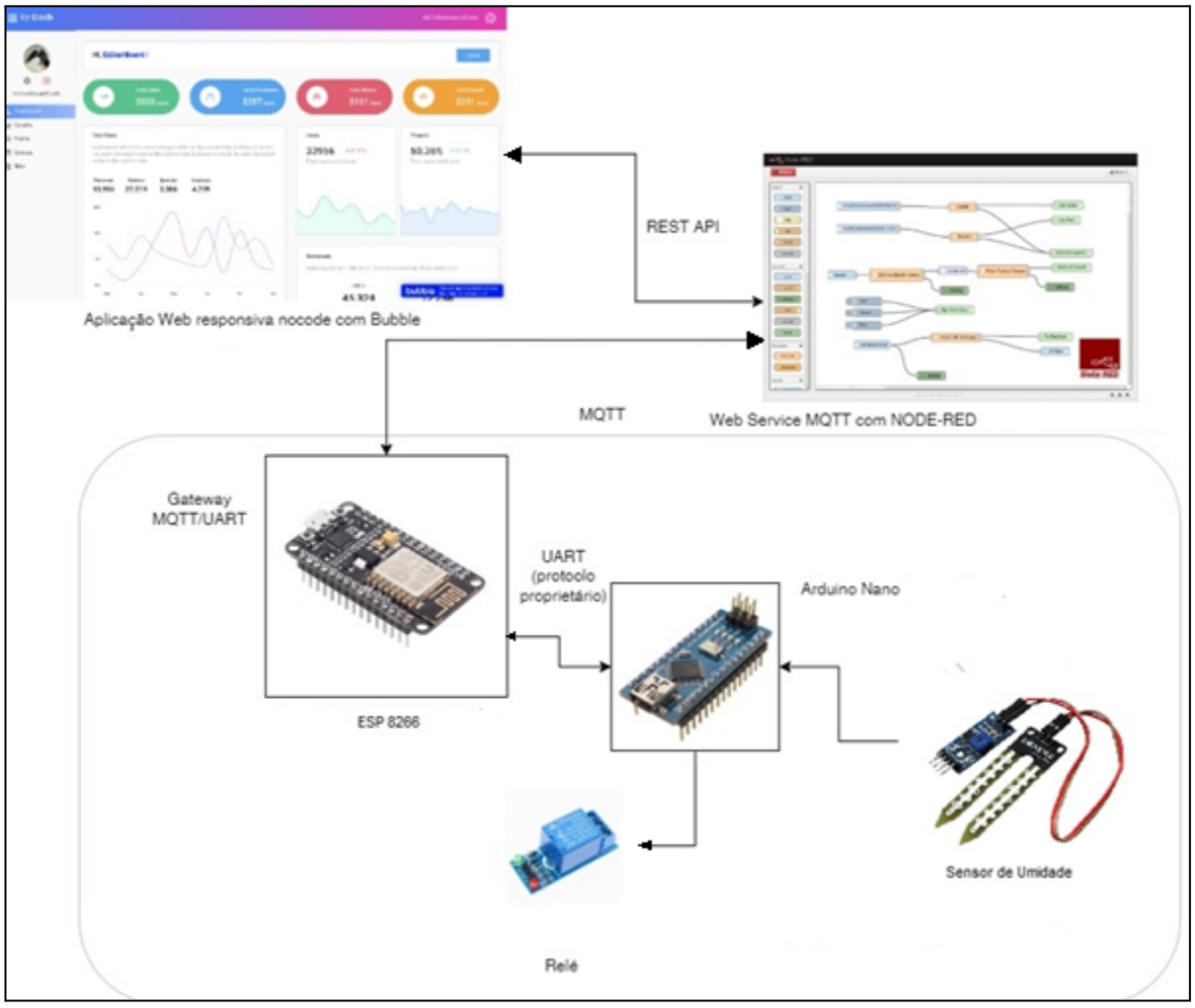
4. DESENVOLVIMENTO DA INTEGRAÇÃO DOS *SOFTWARES* COM O *HARDWARE*

4.1. APRESENTAÇÃO DO PROJETO

Todo projeto possui uma primeira fase de elaboração, em que são apresentados os requisitos do projeto, possibilidades para o desenvolvimento, as metodologias a serem seguidas e qual o resultado esperado. No caso deste trabalho, as especificações foram estabelecidas pelo professor orientador.

Este projeto consiste em desenvolver *softwares* com a finalidade de realizar o controle de umidade de vaso de plantas utilizando plataformas de desenvolvimento de *software* com baixo ou sem uso de codificação para desenvolver o *backend* e o *frontend,* que serão integrados a um *hardware* reaproveitado de uma outra aplicação acadêmica semelhante, tendo sido cedido pelo professor orientador deste projeto. A Figura 5 apresenta um fluxograma contendo os componentes existentes no *hardware*, os *softwares* e as conexões e comunicações existentes.

Figura 5 - Fluxograma da proposta de projeto.



Fonte: Próprio autor.

De acordo com o solicitado, o *hardware* cedido dispõe de um Arduino Nano que comanda os sensores de umidade que verificam a umidade do solo e o nível do reservatório de água, assim se a umidade do solo detectada for baixa, um relé é acionado, ligando uma bomba d'água para realizar irrigação do solo. Caso o nível de água do reservatório esteja baixo, o sistema não funciona, pois a bomba d'água não é acionada a seco, o que causaria a queima do sistema. Como a aplicação do antigo projeto não exigia comunicação *wireless* e para este projeto isso se faz extremamente necessário, para uma solução com menor custo e com menos tempo de implementação, adicionou-se o microcontrolador NodeMCU ESP8266 que realiza a comunicação serial via protocolo RS232, o qual tem como função básica

servir de antena para conexão Wi-Fi. Dessa forma, caso esse projeto fosse desenvolvido apenas com o microcontrolador NodeMCU ESP8266, seria necessário criar um novo *hardware*, o que demandaria prazo maior que o disponível.

Para este projeto o *frontend* consiste em uma interface gráfica para o usuário, feita via plataforma de desenvolvimento de *software* sem o uso de codificação Bubble.io. O *backend* faz a integração da eletrônica de controle com a interface gráfica (aplicativo web), para isso utiliza uma API concebida utilizando a plataforma de desenvolvimento de *software* com baixo uso de codificação Node-RED, que será responsável por receber as requisições e enviar as respostas conforme foi solicitado.

4.1.1. Bubble.io

Fontes do próprio site da Bubble.io indicam que seu surgimento ocorreu quando Josh, um dos dois sócios de uma empresa que atuava no ramo da tecnologia, precisava desfazer a sociedade, mas não queria deixar seu parceiro na mão, já que este não tinha domínio desse ramo. Dessa forma, teve início o desenvolvimento de uma plataforma que fosse capaz de auxiliar qualquer pessoa que não fosse do ramo tecnológico a iniciar uma *startup*. Em 2012, Josh já havia iniciado as programações do que viria a ser a Bubble, quando encontrou Emmanuel Straschnov, com formação em ciência e tecnologia, que veio a ser co-fundador desse novo projeto. Ambos tinham grande convicção do potencial gigantesco que a plataforma teria, pois permitiria criar aplicativos Web, totalmente funcional, para usuários sem conhecimento de programação, já que haveria uma simplificação de toda a parte de codificação, que é a mais complexa (CHEN, 2020).

Ao longo do tempo, a plataforma foi sendo aprimorada e foram introduzidos vários conceitos, que surgiram de acordo com a demanda. Considerada uma das primeiras formas de linguagem de “programação visual”, tem como princípio a diferença entre codificação e programação. A programação está relacionada com a capacidade de raciocinar logicamente no fluxo de trabalho e na estrutura de um aplicativo, por exemplo. Já a codificação, por sua vez, consiste no domínio de regras e linguagem de computador, a fim de aplicar a lógica (CHEN, 2020).

Optar pela utilização desta plataforma de desenvolvimento de *software* sem o uso de codificação neste projeto foi uma sugestão do professor orientador, uma vez que ele tem o conhecimento para auxiliar ao longo do desenvolvimento do projeto. Além disso, a Bubble.io destaca-se bem em diferentes aspectos como a agilidade na construção do aplicativo, principalmente nas fases iniciais, pois permite aplicar qualquer ideia de forma rápida, validando-a e a disponibilizando para uso, pois não há necessidade de entrar em contato com o desenvolvedor para realizar qualquer tipo de modificação, o que pode ser feito pelo próprio usuário (WELL, 2022).

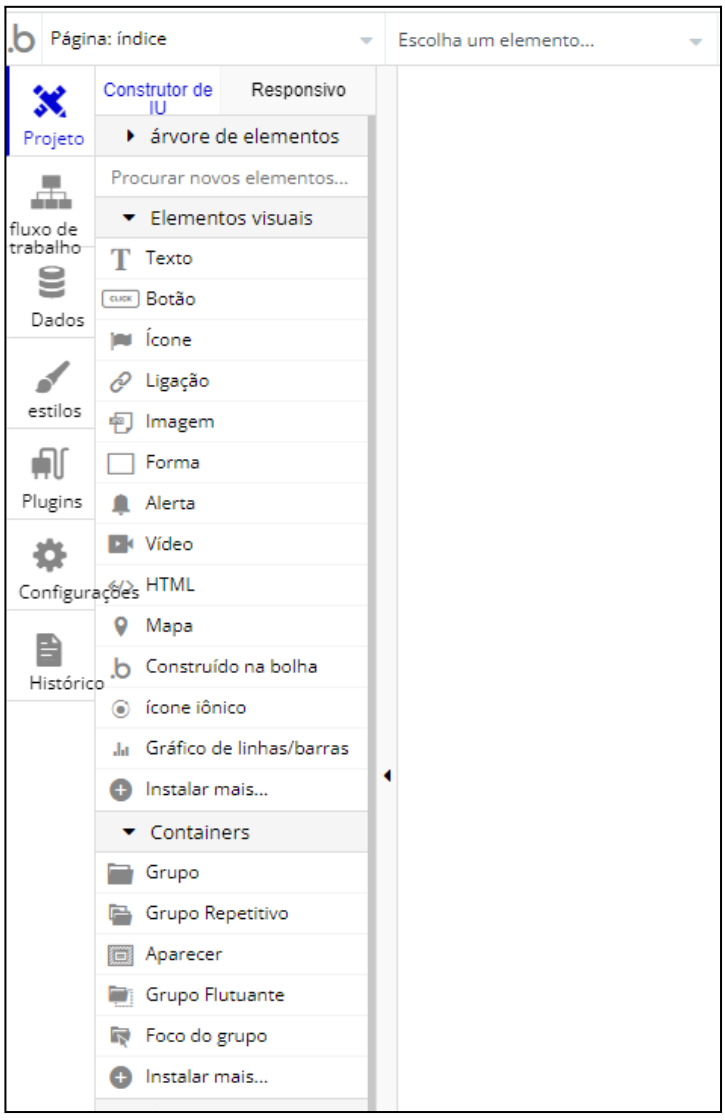
Essa plataforma conta com recursos muito poderosos, que são capazes de tornar a aplicação exclusiva, em que se pode criar páginas, adicionar funcionalidades de preenchimento automático, personalizar planos de fundo com imagens ou vídeos, inserir textos e atualizá-los, sem a necessidade de qualquer tipo de programação (WELL, 2022).

4.1.1.1. Funcionamento do Bubble.io

Como a Bubble.io foi a plataforma de desenvolvimento de *software* sem o uso de codificação selecionada para criar *frontend* de um *software*, ela dispõe de versão gratuita com diversas funcionalidades que atendem aos requisitos do projeto. Para dar início ao desenvolvimento, é preciso criar uma conta no *site* da plataforma, pois tudo irá rodar no servidor web.

Após o cadastro, pode-se ter os primeiros contatos com a plataforma, iniciando com a criação de uma identificação para o aplicativo, para que, de fato, possa começar sua criação. A Figura 6 mostra que a interface possui ícones e abas de personalização intuitivos, que trazem praticidade no acesso aos elementos necessários para a criação da aplicação, já que o método de desenvolvimento desta plataforma consiste em “arrastar” e “soltar” os ícones e inserir em cada um deles uma ação ou função desejada.

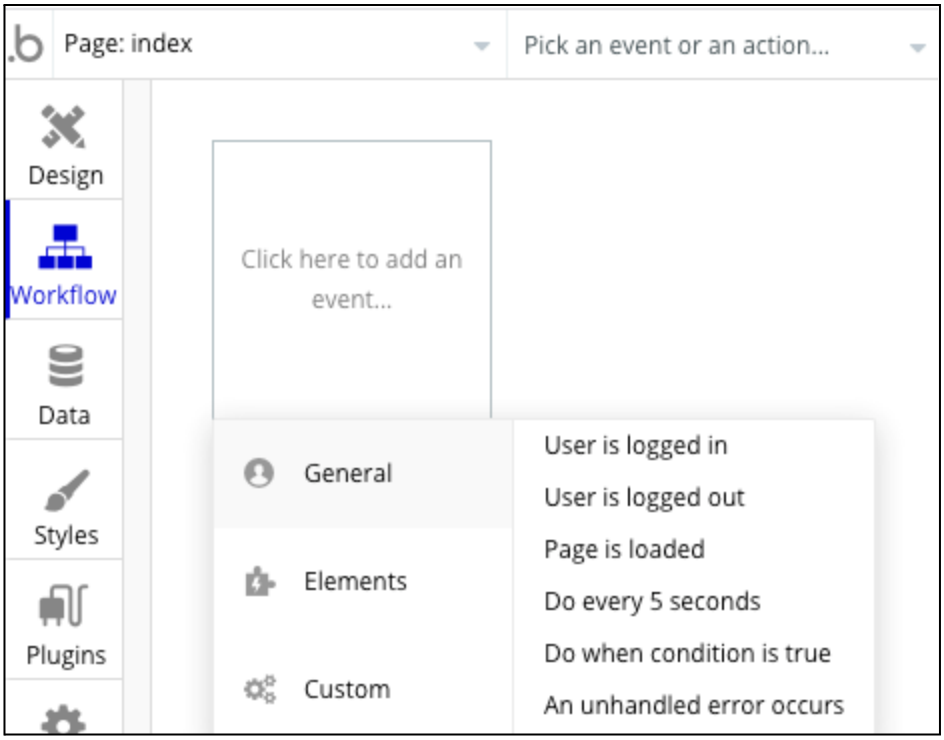
Figura 6 - Interface de desenvolvimento Bubble.io



Fonte: Próprio autor.

Durante o desenvolvimento do software é possível observar no seu funcionamento que as alterações realizadas são inseridas automaticamente na área de criação, podendo observar-se as mudanças realizadas instantaneamente. Ao longo da criação do aplicativo pode-se inserir os fluxos de trabalho (*workflows*), representado na Figura 7, que consistem em ações que ocorrerem, caso um botão for pressionado, algum dado for enviado para cadastro, houver mudanças de cores no *layout*, etc.

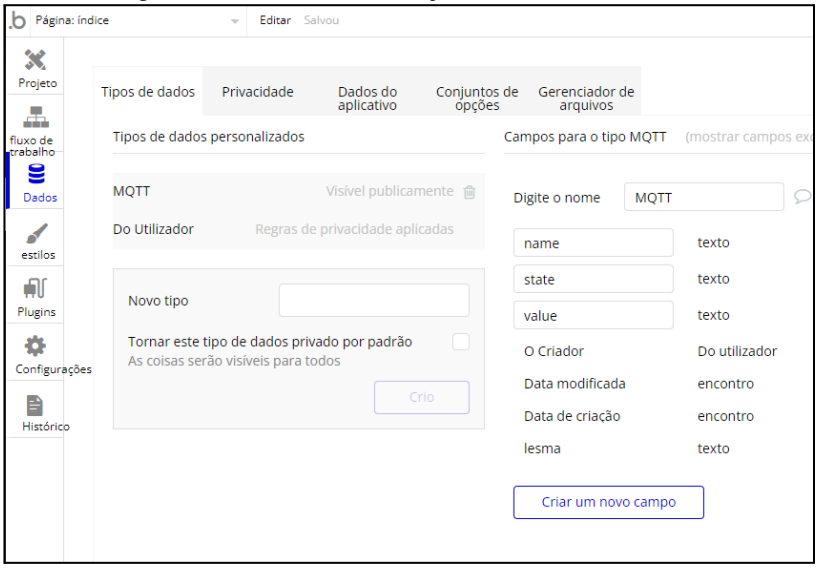
Figura 7 - Interface de criação do fluxo de trabalho.



Fonte: Próprio autor.

Na versão gratuita, a plataforma disponibiliza um banco de dados de aproximadamente 0,5 GB, servindo principalmente para aplicativos ainda em fase teste, no entanto bem completo e personalizável (FERREIRA, 2022). A Figura 8 mostra como é a interface de criação do banco de dados.

Figura 8 - Interface de criação do banco de dados



Fonte: Próprio autor.

Antes de iniciar a criação do *software*, é necessário para este projeto fazer a instalação do *plug-in* de Conector de API que permitirá conectar o *frontend* a uma API externa, tornando o projeto personalizado e interativo.

4.1.1.2. Conector de API do Bubble.io

Com os conceitos de API apresentados na seção 3.3, torna-se possível compreender a função de *API Connector* (conector de API), disponibilizado pela plataforma Bubble.io, que permite conectar a aplicação desenvolvida com praticamente qualquer API externo, via protocolo HTTP de comunicação, assim *API Connector* espera um corpo JSON formatado corretamente como resposta. É possível fornecer cabeçalhos, parâmetros e um corpo de chamada, sendo que cada chamada pode ser configurada para ser usada como ação em seu aplicativo ou como fonte de dados.

As chamadas de API criadas podem ser usadas como ações ou consumidoras de dados. Elas geralmente necessitam de alguma autenticação para que o serviço identifique quem está realizando a chamada. As chamadas de API estão adicionadas na guia API do Editor de plug-ins do Bubble.io, no qual será definido como as chamadas serão autenticadas e quais delas serão usadas como fontes de dados ou como ações.

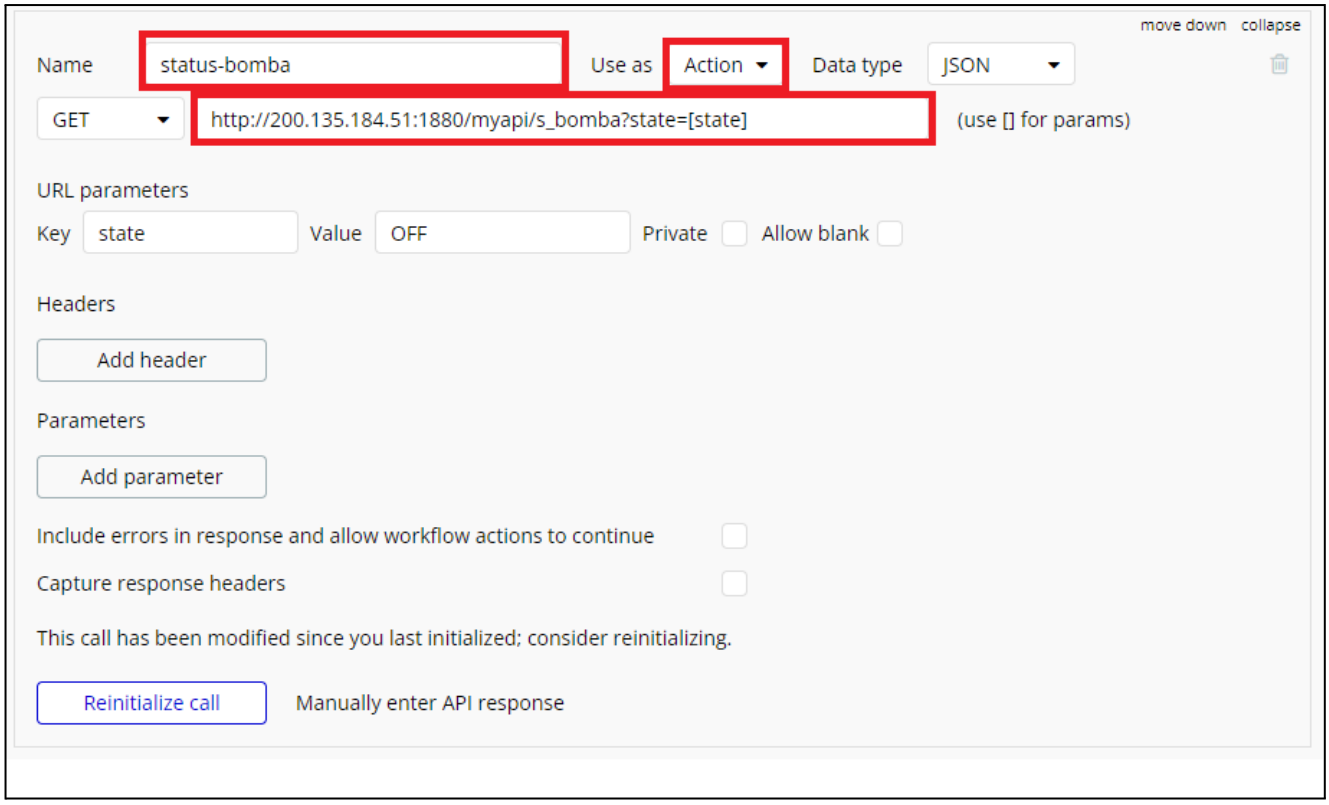
4.1.2. Chamadas de API

As chamadas são realizadas toda vez que o usuário faz alguma interação com a interface do aplicativo web, clicando em algum botão ou inserindo algum dado. Além disso, a interface pode conter campos que mostram informações externas recebidas, a exemplo de aplicações em que os dados são atualizados constantemente, fazendo constantes atualizações nos tópicos da API e, cada vez que isso acontece, automaticamente há uma chamada para atualizar a informação na interface.

Conforme apresentado na Figura 9, para configurar as chamadas de API é necessário inserir algumas informações como nome para API, método de conexão, no caso deste projeto todas as chamadas utilizaram o método do tipo GET, o

endereço URL da API com os parâmetros utilizados para cada chamada. As chamadas de API podem ser utilizadas como *action* (ação), caso haja qualquer interação no aplicativo, que consista em enviar uma mudança nos tópicos, ou *data* (dado), que realiza a chamada toda vez que o tópico do broker for modificado, para que a atualização com a nova informação enviada ao tópico do broker seja feita no campo informado na interface do aplicativo.

Figura 9 - Informações para a configuração da chamada de API.



Fonte: Próprio autor.

4.1.3. Criação da interface

A interface gráfica do aplicativo é desenvolvida na aba Design da plataforma Bubble.io, na qual estão disponíveis os elementos visuais e interativos como, por exemplo, os botões, caixas de texto, imagens de fundo, as entradas de dados, formas geométricas, entre outros tantos itens que não foram necessários, durante a criação. A Figura 10 apresenta o resultado final do desenvolvimento.

Figura 10 - Interface do aplicativo web.

Fonte: Próprio autor.

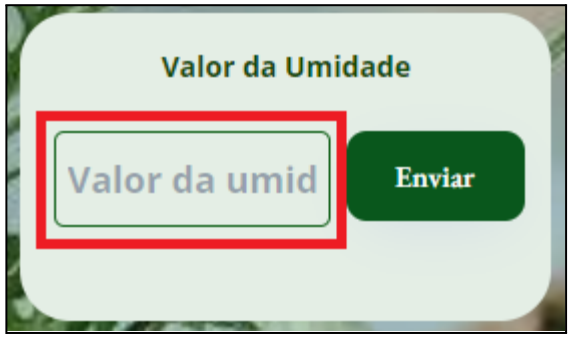
Cada elemento inserido precisa ser configurado e condicionado de acordo com a sua função no aplicativo, como os botões de opção ON e OFF mostrados na Figura yy, por exemplo, em que é possível observar uma interação quando o botão varia entre os dois estados, alterando a cor de fundo da forma geométrica de cinza para verde quando o botão é mudado de OFF para ON e vice-versa.

Figura 11 - Mudança da cor de fundo da forma geométrica, quando selecionado ON/OFF.

Fonte: Próprio autor.

Neste aplicativo existe um ícone para inserir os dados a serem enviados para publicar no tópico do *broker*, conforme mostrado em destaque na Figura 12.

Figura 12 - Entrada de dados a serem enviados via chamada de API.



Fonte: Próprio autor.

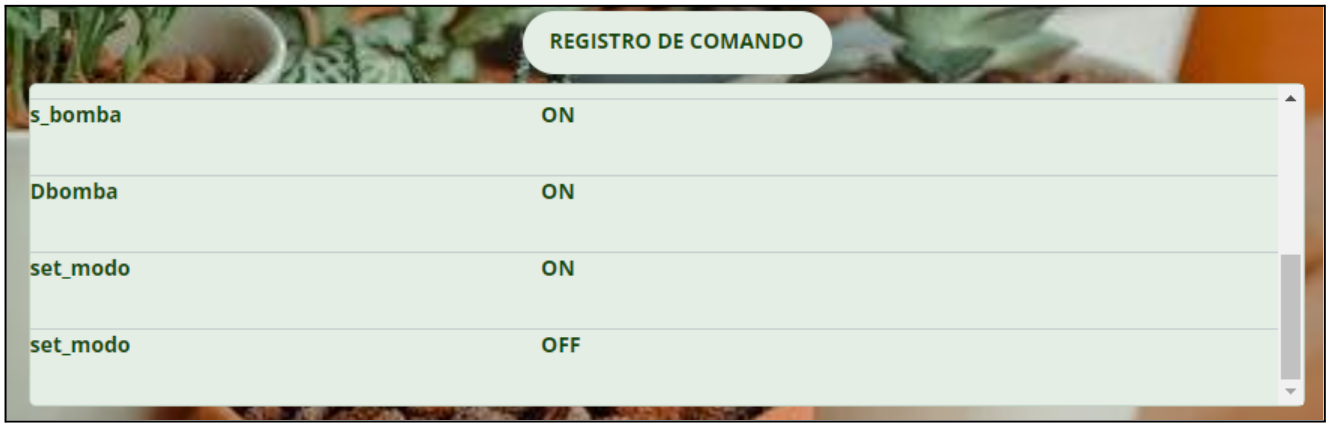
No entanto, também existem outros dois ícones responsáveis por mostrar os dados recebidos via chamada de API de alguma atualização nos tópicos do broker que possuem a informação desejada. Os ícones responsáveis por mostrar as informações recebidas estão presentes em destaque na Figura 13.

Figura 13 - Exibição dos dados recebidos via chamada de API.

Fonte: Próprio autor.

A parte inferior do aplicativo, apresentado na Figura 14, possui um registro de comando, no qual as atualizações recebidas ou enviadas serão disponibilizadas na interface para que se possa ter um registro dos comandos que forem realizados no aplicativo.

Figura 14 - Registro de comandos.



Fonte: Próprio autor.

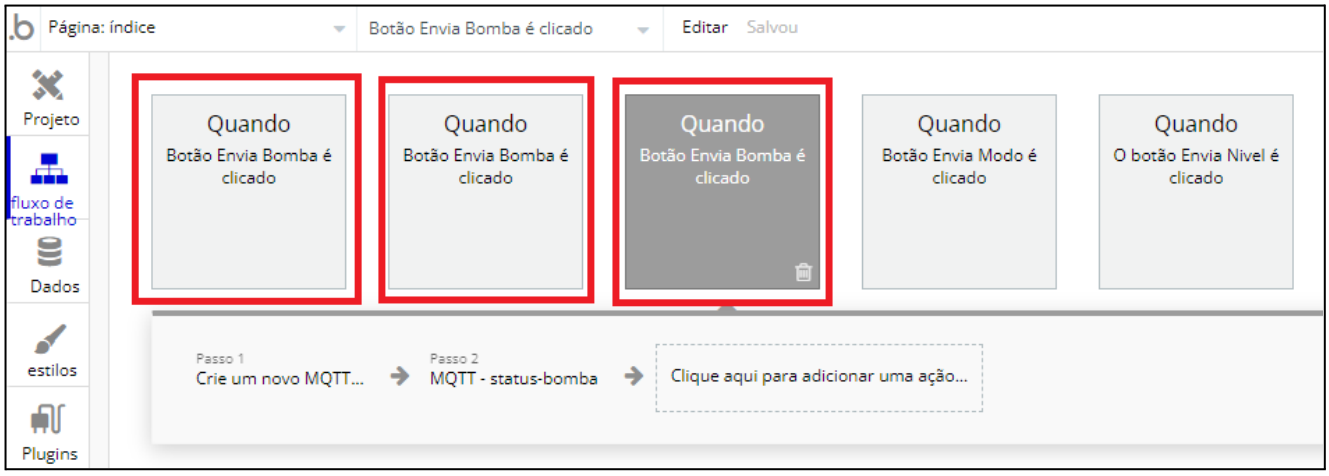
4.1.4. Criação do fluxo de trabalho (*workflow)*

No fluxo de trabalho desse aplicativo encontram-se todas as condições criadas para determinar o que acontece com as informações ou ações realizadas na interface. Neste aplicativo existem três botões “Enviar”, configurados para que, quando clicados, sejam verificadas as condições selecionadas para botões ON/OFF dos itens “estado da bomba” e “modo automático” e o valor determinado para o nível de umidade. Isso faz com que seja realizada uma chamada de API toda vez que cada um desses botões “Enviar” for pressionado, de modo a enviar a informação

inserida, a fim de que ela seja publicada no tópico do *broker* correspondente.

O botão “Enviar”, referente ao item “estado da bomba", está como início de três fluxos de trabalho, conforme destacado na Figura 15, os quais serão realizados de acordo com a condição selecionada no botão ON/OFF. Para esse projeto, cada fluxo de trabalho será responsável por solicitar uma chamada de API. Isso ocorre para que a informação enviada seja publicada em três tópicos diferentes do *broker*. Para realizar a publicação em cada um dos tópicos é necessário realizar chamadas de API em fluxos de trabalho diferentes, pois elas não podem estar vinculadas e dependentes entre si.

Figura 15 - Início dos *workflows* de “estado da bomba”.



Fonte: Próprio autor.

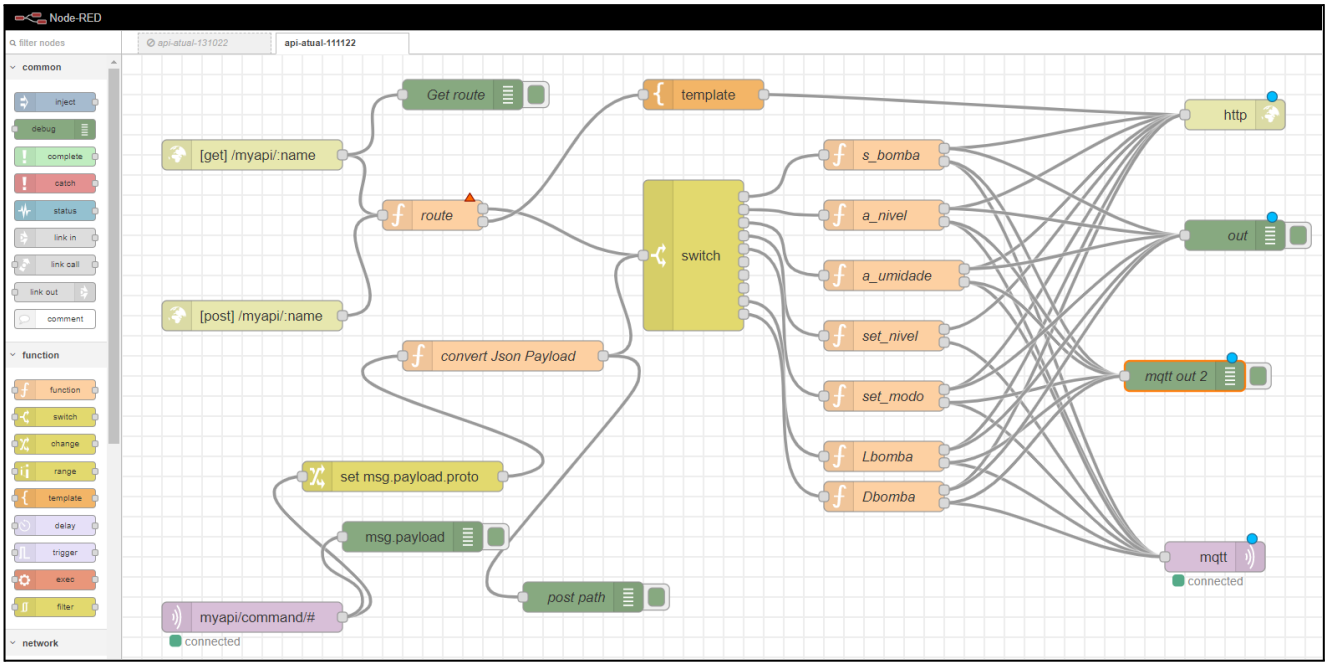
Já os outros dois botões “Enviar”, um refere-se ao modo de funcionamento e está relacionado ao estado do botão ON/OFF correspondente, definindo “ON” para modo automático e “OFF” para modo manual. Ao pressionar o botão “Enviar” do modo de funcionamento é avaliado o estado do botão ON/OFF correspondente, para então iniciar o respectivo fluxo de trabalho, realizando a chamada de API e então a publicação pode ser feita no respectivo tópico do broker. Por fim, o último botão “Enviar”, ao ser pressionado faz com que por meio do seu *workflow* seja solicitada uma chamada de API a fim de publicar, no tópico do broker referente ao nível umidade, o valor inserido na interface do aplicativo.

4.1.5. Node-RED

O Node-RED é uma plataforma de desenvolvimento de *software* com baixo uso de codificação, sendo muito aplicada para a criação de aplicativos e serviços de IoT. Essa plataforma foi desenvolvida por Nick O'Leary e Dave Conway-Jones e seu início se deu devido a necessidade de criar protótipos de diversas aplicações de IoT. Os desenvolvedores deram início a Node-RED com o intuito de utilizá-lo para as suas próprias aplicações e projetos. No entanto, buscavam formas de deixar o processo ainda mais simples e que permitisse conectar sistemas e sensores. Já no final de 2013, foi lançado como um projeto de código aberto, permitindo que alguns usuários se tornassem adeptos a essa tecnologia. Atualmente já existe uma comunidade fiel que auxilia no desenvolvimento constante da plataforma, criando bibliotecas e trabalhando com os códigos da própria plataforma (LEA, 2016).

A plataforma baseia-se em blocos de códigos, organizados e interligados pelo usuário, que podem enviar, processar e receber diversas informações, além de comunicar-se com alguns tipos de protocolos. Na Figura 16, pode-se observar apenas como fica a disposição dos elementos interface de programação, percebe-se que cria espécie de fluxograma em que os blocos de códigos (*nodes* – “nós”) são conectados entre si por meio das rotas (r*outes*), de acordo com o fluxo da informação. Cada um desses nodes pode ser uma entrada (“input-nodes”), uma função de processamento (“function-nodes”) ou uma saída (“output-nodes”).

Figura 16 - Interface Node-red



Fonte: Próprio autor.

O Node-RED é acessado via navegador web, no qual é possível encontrar o editor, inserir os nodes e conectá-los. Após conectar esses nodes é possível executar a aplicação (ROCHA, 2018).

Estas são suas características principais:

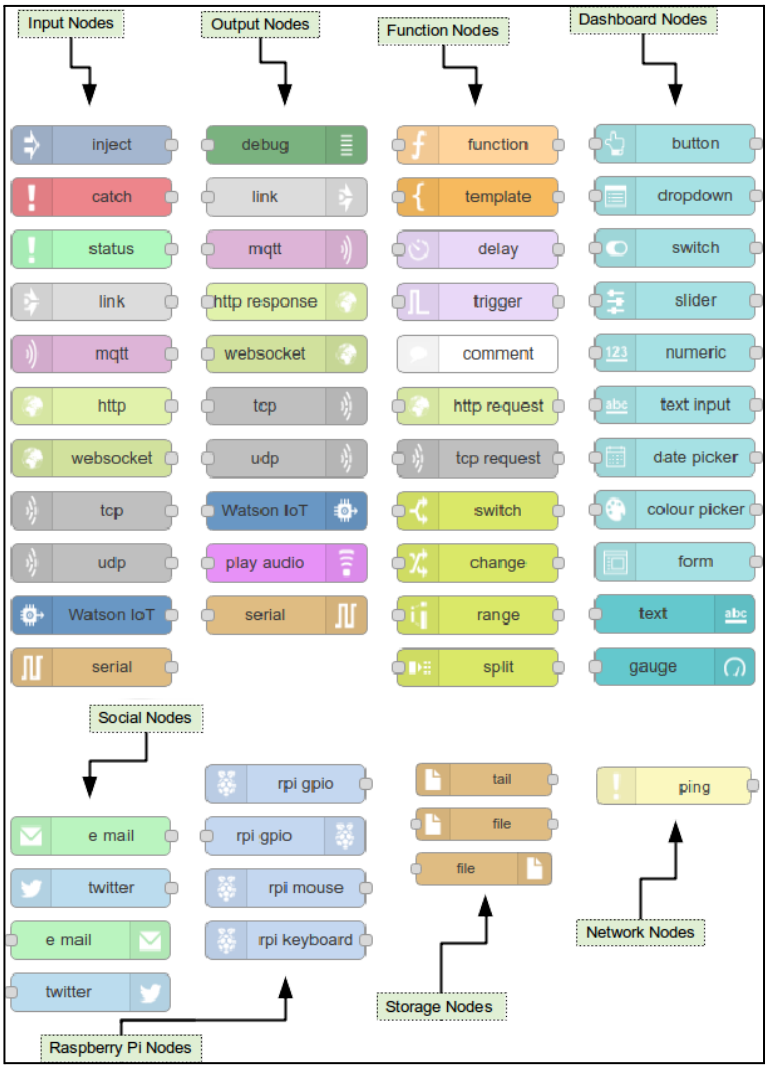
- Editor de fluxos no navegador – Basta um navegador web para começar a conectar o nodes e criar fluxos.
- Construído em Node.js – O Node-RED é ideal para rodar em *hardwares* de baixo custo e também na nuvem.
- Fácil compartilhamento – Os fluxos criados em Node-RED são armazenados usando JSON e podem ser importados e exportados com facilidade permitindo o compartilhamento com outras pessoas.

Assim que o Node-RED já estiver em funcionamento, pode-se acessar o editor de qualquer outra máquina, com outro navegador, desde que o IP de quem está rodando o Node-Red seja conhecido.

A plataforma de desenvolvimento de *software* com baixo uso de codificação Node-RED servirá para implementar e adaptar as necessidades deste

projeto, por exemplo API (COPE, 2020) apresentado pelo professor orientador. Para compreender o funcionamento é preciso avaliar qual a funcionalidade de cada um dos nodes, como mostra a Figura 17 é possível observar os tipos de *nodes* existentes na instalação padrão do Node-RED. A seguir estão descritos os principais *nodes* utilizados na criação da aplicação desse projeto, a API Rest.

Figura 17 - Tipo de *nodes* do Node-RED



Fonte: (ROCHA, 2018).

Os "input-nodes" são responsáveis por dar início a um fluxo e normalmente esses nodes são conectados a evento externo como uma requisição HTTP, uma mensagem MQTT, a chegada de uma informação na porta serial e até mesmo a um botão. Possuem apenas um ponto de conexão no lado direito, o conector de saída, não contendo ponto de conexão de entrada.

Os "output-nodes" fazem o envio das informações para a aplicação. Podem ser facilmente identificados, pois não contêm o conector de saída do lado direito. Eles permitem que dados sejam enviados para um MQTT broker.

Nos "function-nodes" ocorre o processamento das informações recebidas, já nesses *nodes* há conexões de entrada e saída. Ao receber uma mensagem na entrada, ele processa a informação (executa uma função) e com isso é capaz de modificá-la e enviá-la por sua conexão de saída.

As informações que seguem por meio das rotas estabelecidas entre os *nodes*, é um objeto JavaScript denominado msg. Sendo possível adicionar, modificar e remover algumas propriedades. Na Figura 18 pode-se obser como as informações do msg são apresentadas, dentre elas, as descritas abaixo possuem destaque:

- payload – Considera-se a propriedade mais importante, através dela as informações são transferidas de um *nodes* para outro. Lidas pelo *node* que recebe as informações, podendo ou não modificá-las. O Payload pode vir a ser de um dos seguintes tipos: Object, Date, Boolean, String ou Number. (ROCHA,2018)
- topic – Essa propriedade é do tipo String e pode ser usada para orientar o que está acontecendo com objeto. Útil quando se deseja saber qual tópico a mensagem foi publicada. (ROCHA,2018)

Figura 18 - Propriedades payload e topic

Fonte: Próprio autor.

4.2. DESENVOLVIMENTO DE API UTILIZANDO A PLATAFORMA NODE-RED

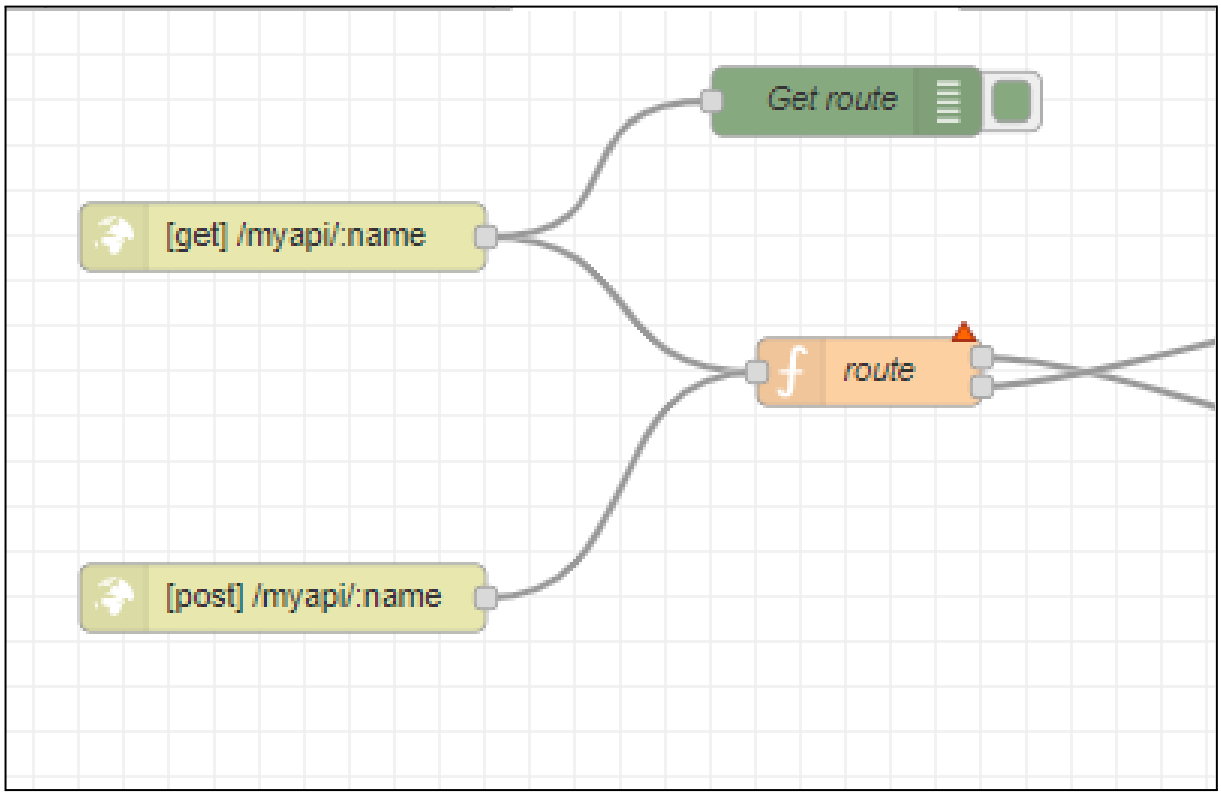
Após o professor orientador apresentar qual seria o exemplo de API que serve como base para desenvolver a API deste projeto, iniciou-se uma avaliação para verificar o funcionamento da “API base” e como poderiam ser feitas as adaptações para que os requisitos pudessem ser atingidos, como testar o funcionamento das conexões tanto via protocolo HTTP, quanto MQTT, bem como se as informações estariam seguindo as rotas corretas e seguindo de um *node* para o outro.

Para descrever melhor o funcionamento desta API, a seguir será explicado em partes a função de cada *node* e o caminho que a informação precisa percorrer entre eles para chegar ao local desejado.

Na Figura 19 é possível observar que há um trecho conectando os *nodes* que realizam a requisição de dados via protocolo HTTP. Nesta etapa o aplicativo

web faz suas solicitações via HTTP request, enviando a informação para verificar o endereçamento (*route*).

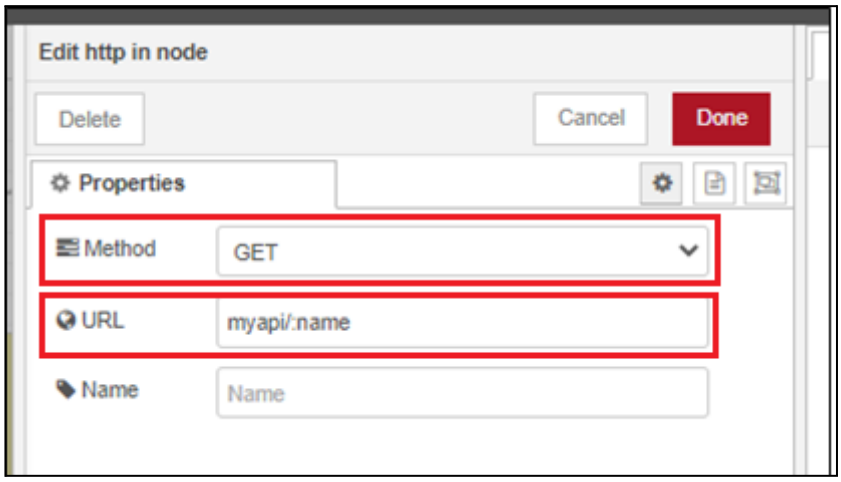
Figura 19 - Estrutura de requisição HTTP request.



Fonte: Próprio autor.

Cada um dos *nodes* precisa conter as informações correspondentes ou códigos necessários. A Figura 20 mostra a informações contidas no *node* de HTTP request, em que é possível inserir o método utilizado, nesse caso o GET e o endereço URL desejado, sendo que, para este projeto, foi utilizado o mesmo da “API base” que é: myapi/:name.

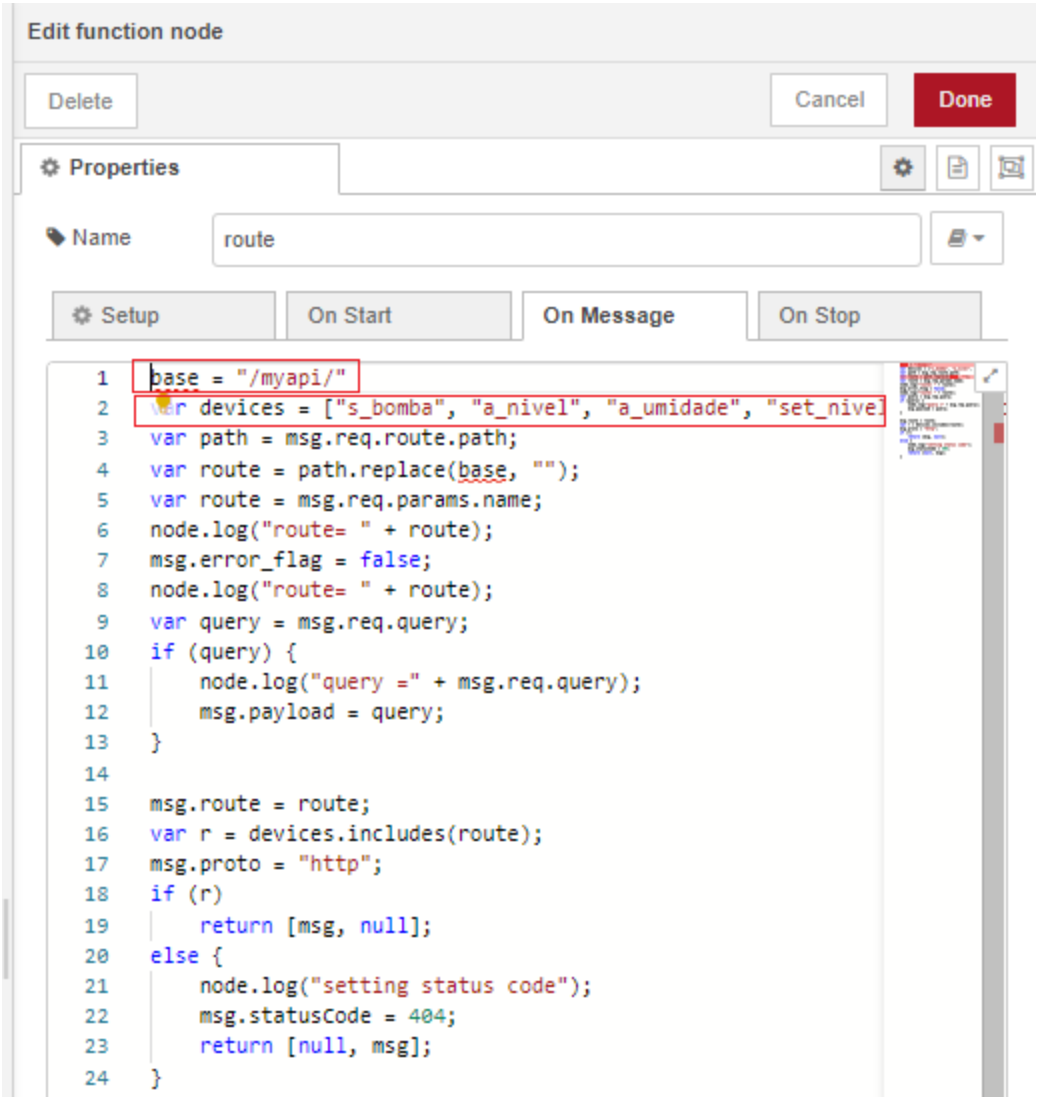
Figura 20 - Editor do *node* de HTTP request.



Fonte: Próprio autor.

A função route começa a realizar o endereçamento da informação antes de publicá-la, inserindo a base do endereço, no caso “/myapi/” e as opções de tópico para publicação, levando em consideração que neste trabalho as rotas possuem o mesmo nome dos tópicos do *broker*. A Figura 21 permite observar o código com os tópicos disponíveis, s_bomba, a_umidade, a_nível e outros que também precisam estar definidos na chamada da API para que cada publicação seja enviada com o tópico correto do broker.

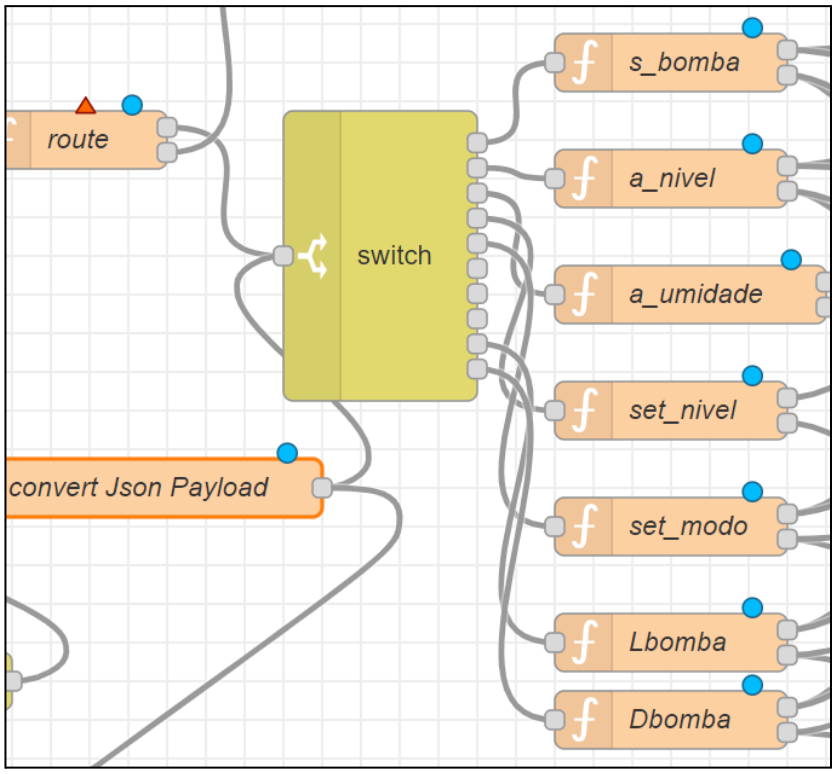
Figura 21 - Edição do *node* route.



Fonte: Próprio autor.

Seguindo com a identificação dos *nodes* e inserindo as informações necessárias para desenvolvimento da API, a próxima etapa é colocar os dados necessários no *node switch,* apresentada na Figura 22, que possui entrada de dados e número de saídas de acordo com o número de tópicos.

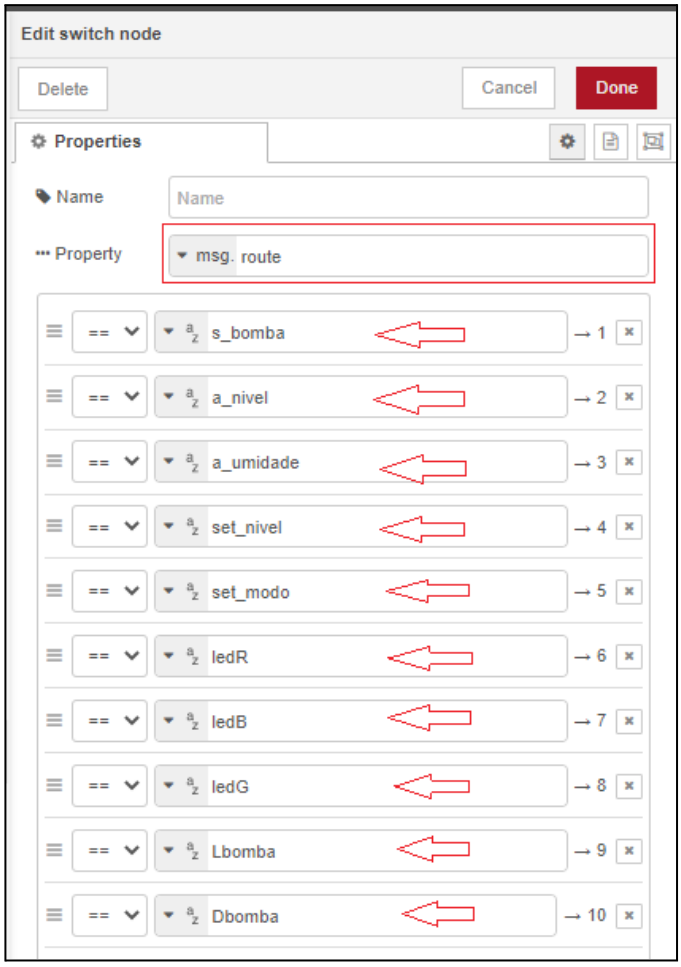
Figura 22 - Estrutura do node *switch*.



Fonte: Próprio autor.

Dessa forma, ele funciona como um identificador de endereço, enviado pelo *node route* para saber qual tópico está sendo requisitado de fato. Assim, ele precisa conter as rotas necessárias da API para que possa encaminhar a informação corretamente, conforme mostra a Figura 23.

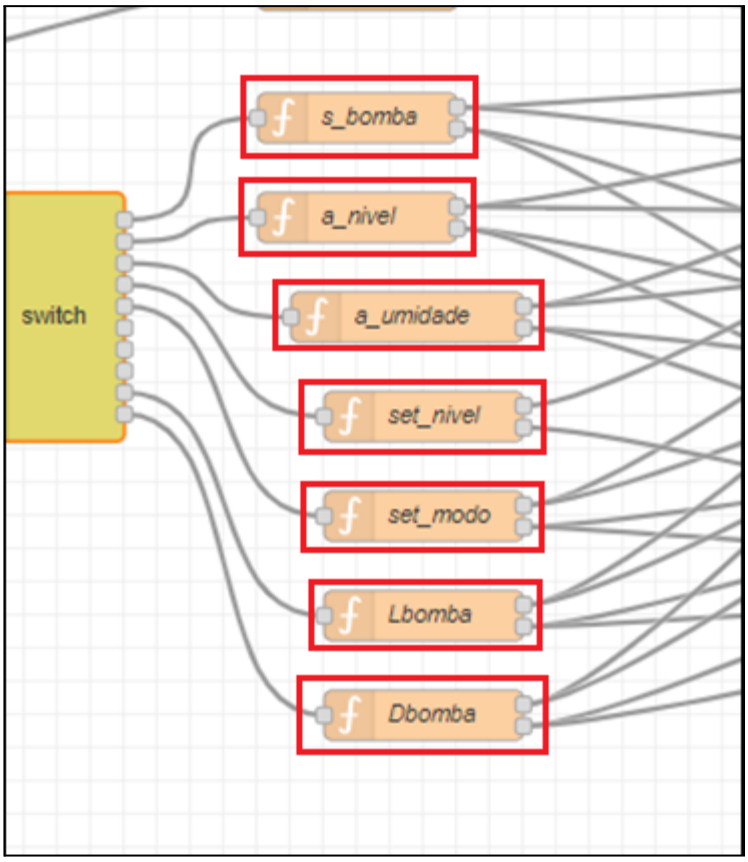
Figura 23 - Informações necessárias para o node *switch.*



Fonte: Próprio autor.

Nesta etapa, o endereço dos tópicos do *broker* já estão definidos na programação de cada um dos “function nodes”, os quais receberam o mesmo nome dos tópicos do *broker*, *estão* conectados às saídas do node switch, conforme a Figura 24.

Figura 24 - “Function nodes” utilizados na API do projeto.



Fonte: Próprio autor.

Além disso, um breve código no formato JSON é executado nas mensagens que passam por ele, esse código contém as condições necessárias para verificar e validar se o conteúdo da mensagem que está sendo transportada em cada uma das rotas e se está de acordo com o que foi solicitado. Isso serve para que a mensagem seja publicada ou requerida do tópico do broker, a mensagem contendo todas as informações necessárias conforme for requisitado.

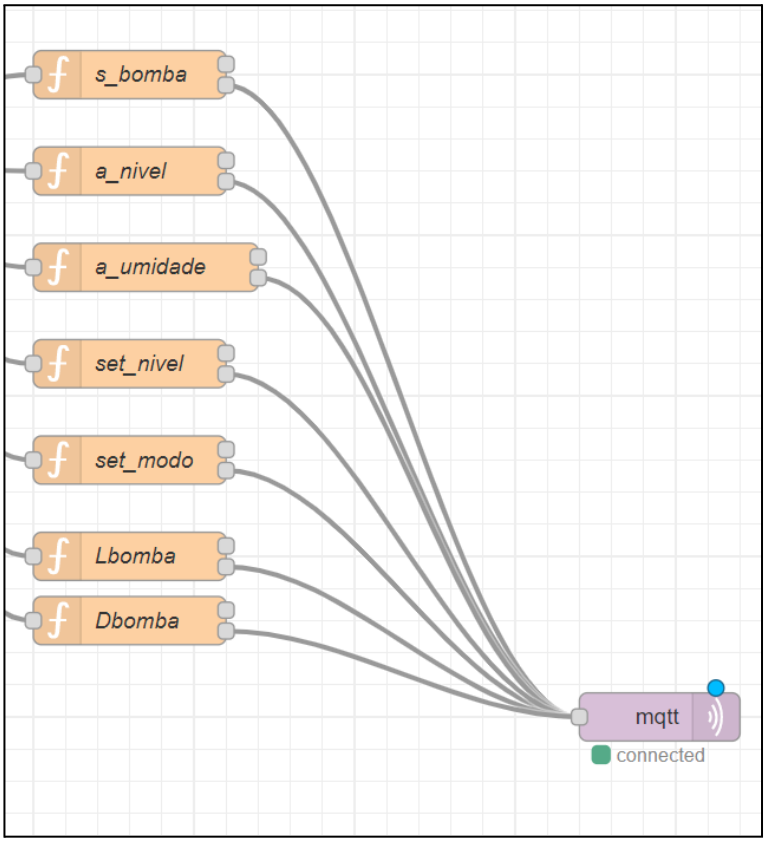
A Figura 25 apresenta um trecho do código JSON contido em um dos “function nodes” da API, executado assim que a mensagem passa por ele. Esse código foi utilizado da “API base”, com apenas algumas adequações, como a alteração na origem da mensagem, conforme destacado na figura a seguir, além de alguns trechos que foram comentados, pois não havia necessidade para a aplicação na API deste projeto.

Figura 25 - Trecho código de “function node” executada na mensagem transportada.

Fonte: Próprio autor.

O *node mqtt* mostrado na Figura 26, é responsável por disponibilizar tudo o que for alterado para o servidor (MQTT Broker HiveMQ), possibilitando que os mesmo disponibilize as informações para os clientes cadastrados no servidor. Na figura a seguir é possível observar que o *node mqtt* possui apenas uma entrada e cada um dos “function nodes” possui uma rota, que se conecta a ele, a fim de que qualquer nova requisição seja encaminhada ao servidor.

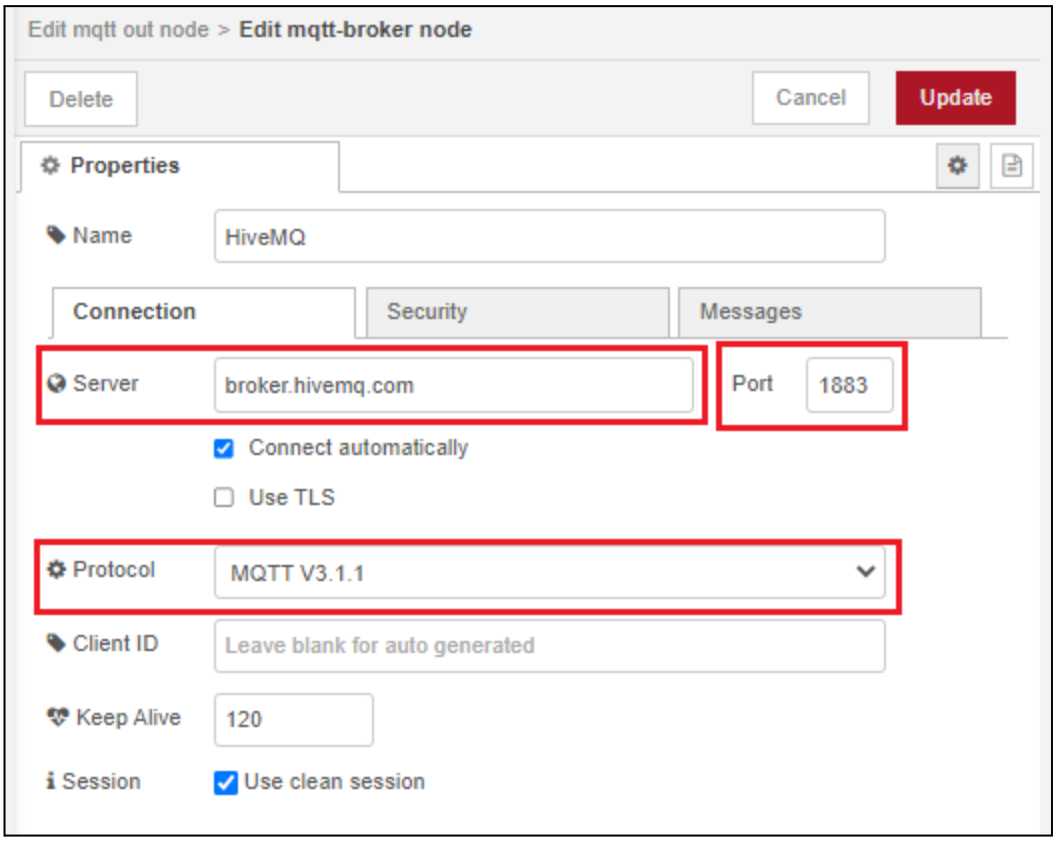
Figura 26 - Rotas dos “function nodes” para o *node* MQTT.



Fonte: Próprio autor.

A Figura 27 destaca as configurações das propriedades do *node mqtt*, que consiste em inserir o servidor (*broker*) denominado de “broker.hivemq.com”, determinar a porta de funcionamento, neste caso 1883 e o tipo de protocolo MQTT utilizado.

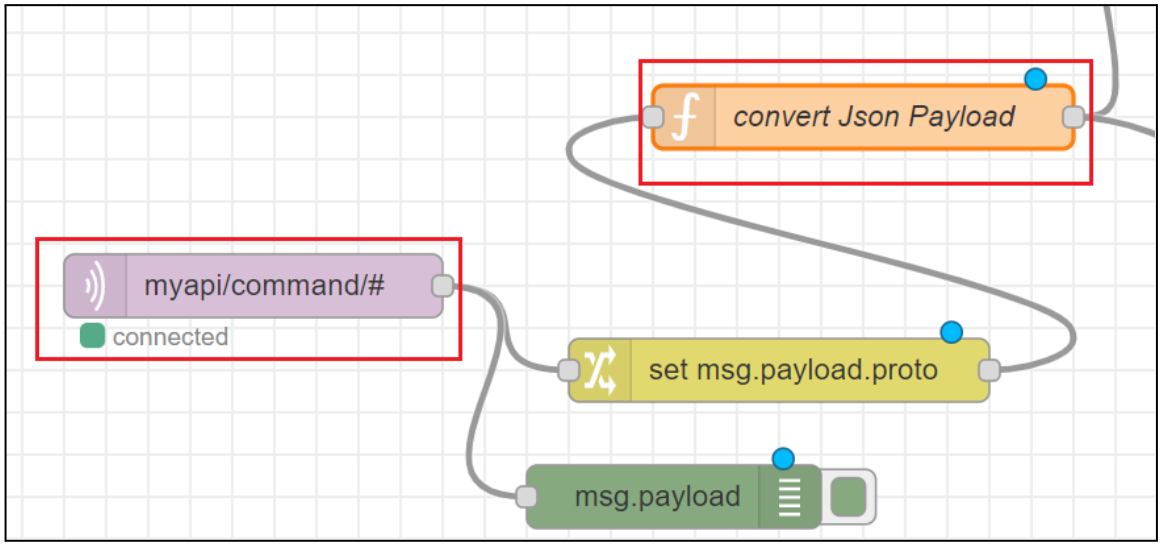
Figura 27 - Informações contidas no node mqtt.



Fonte: Próprio autor.

Esta API também permite que sejam subscritas as informações recebidas via microcontrolador NodeMCU ESP8266, nos tópicos do *broker*. Para isso, é necessário inserir um novo node, chamado de *node mqtt command*, que é capaz de receber os comandos, decifrá-los, convertê-los para a linguagem JSON e publicá-los nos tópicos do *broker*. Esse *node* possui apenas uma saída e a entrada é via WiFi, exemplificado na Figura 28.

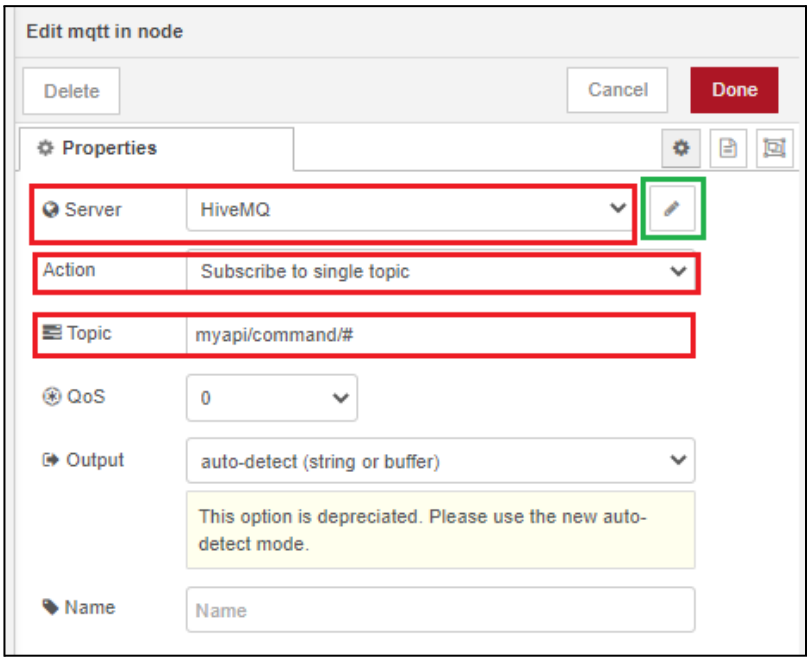
Figura 28 - Node mqtt command e conversor de linguagem.



Fonte: Próprio autor.

Para que as mensagens sejam subscritas nos tópicos do brooker, é necessário inserir algumas informações nas propriedades do *node mqtt command.* Na Figura 29, o que está destacado na cor vermelho são informações implementadas para definir o servidor e a descrição do tópico, o tipo de ação que esse *node* irá fazer, no caso subscrever uma mensagem, que seguirá uma rota de conversão de linguagem e endereçamento para então publicar no tópico do *broker.* Já o ícone destacado na cor verde contém as mesmas propriedades do *node mqtt* que foram citadas anteriormente.

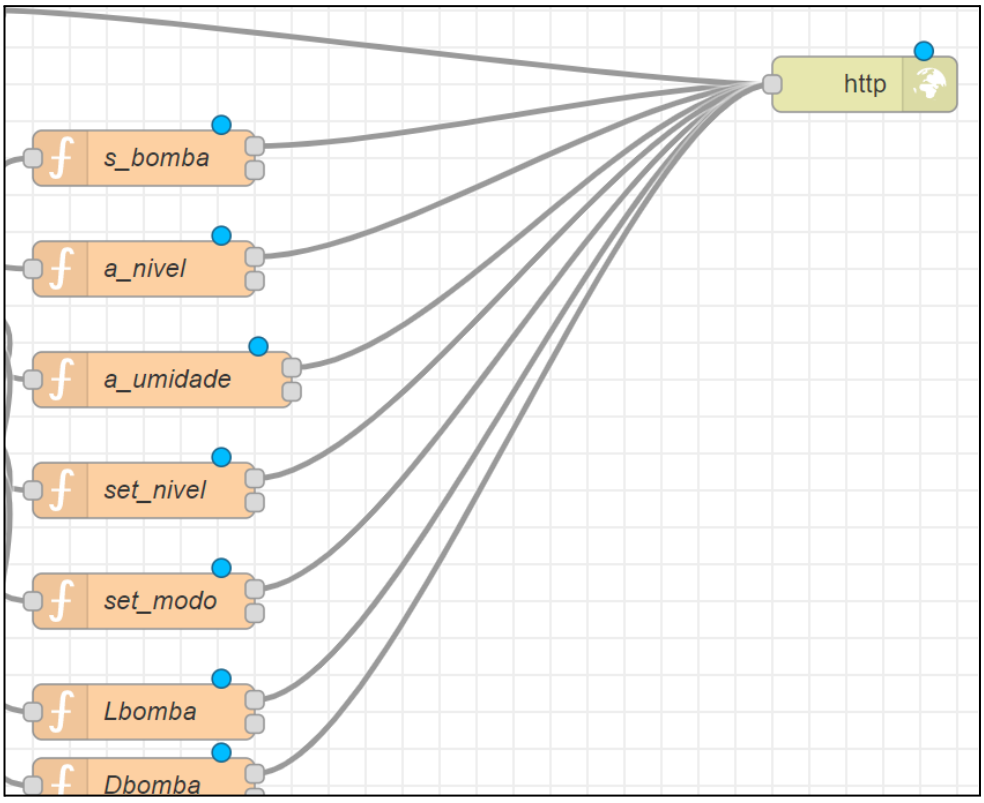
Figura 29 - Propriedades do *node mqtt command*.



Fonte: Próprio autor.

Por fim, faz-se necessário ter um *node* para o http *response* (resposta), que está conectado diretamente ao “function nodes” para verificar se há alguma atualização nos tópicos do *broker*, derivados de publicação via MQTT. Isso permite que as alterações também sejam atualizadas diretamente no aplicativo web. Na Figura 30 é possível observar que o *node* http *response* possui apenas uma entrada, que está conectada a todos os “function nodes” e a saída é via web.

Figura 30 - Rotas *node* http *response*.



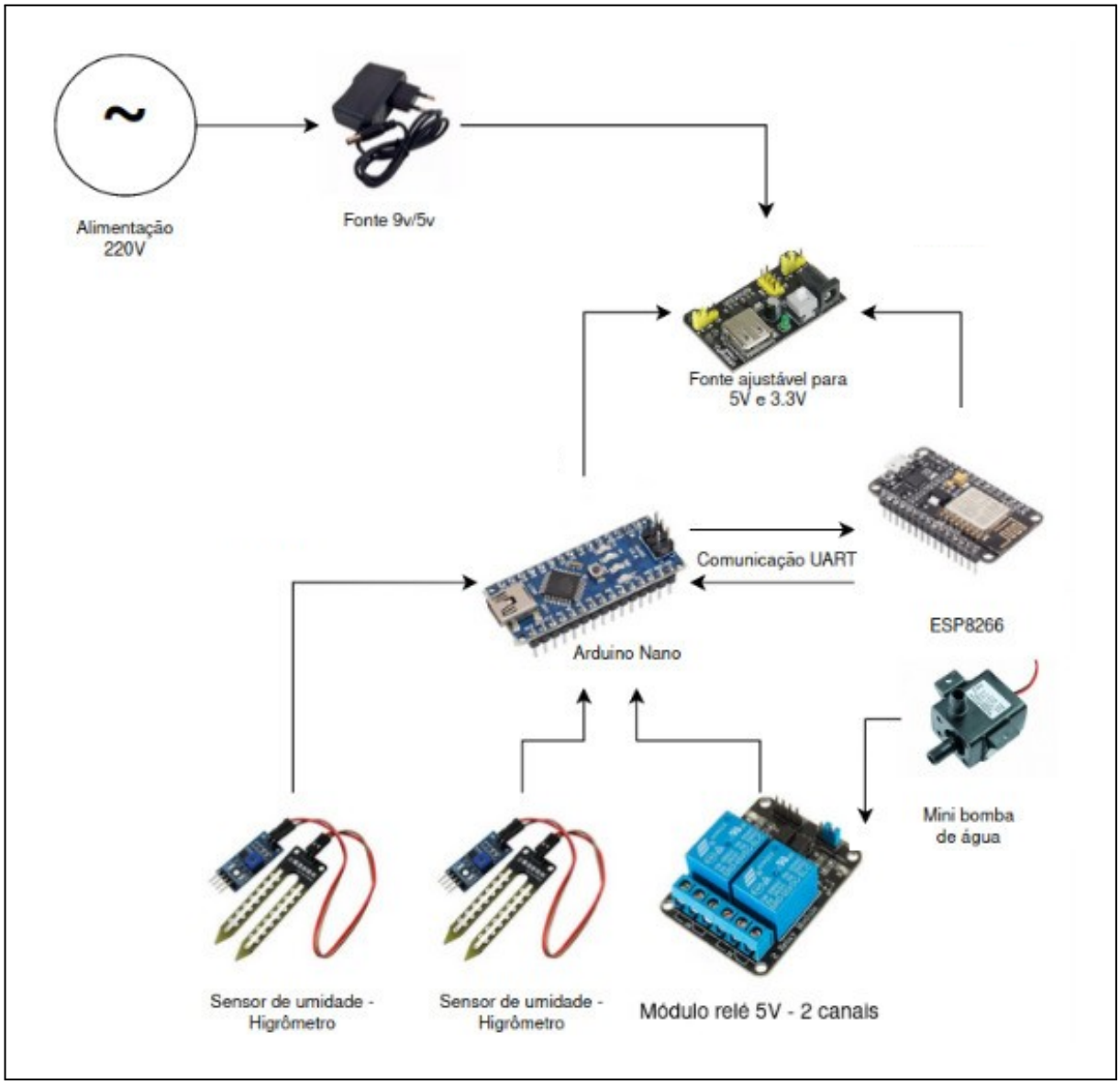
Fonte: Próprio autor.

4.3. *HARDWARE*

Como já mencionado anteriormente, a proposta apresentada pelo professor orientador foi reaproveitar *hardware* de fins didáticos desenvolvido pelo professor do Instituto Federal de Santa Catarina, Valdir Noll, que foi utilizado em projeto com características similares a este. No entanto, este projeto apresenta a peculiaridade de necessitar de *hardware* com conexão Wi-Fi que possibilite a integração com os *softwares* desenvolvidos. A estrutura inicial do *hardware* apresentava placa de circuito contendo um arduino nano, fonte 9V, dois sensores de umidade e um relé para acionar uma bomba d'água. Para adequar o *hardware* às conformidades deste projeto, a solução que demandou menos tempo de desenvolvimento e menor custo foi a inserção de microcontrolador NodeMCU ESP8266 ligado em série com o Arduino Nano, conforme já mencionado anteriormente na seção 3.6. Esta solução se mostrou ideal porque os sensores e o relé já estariam conectados com a placa de circuito, fazendo com que, ao

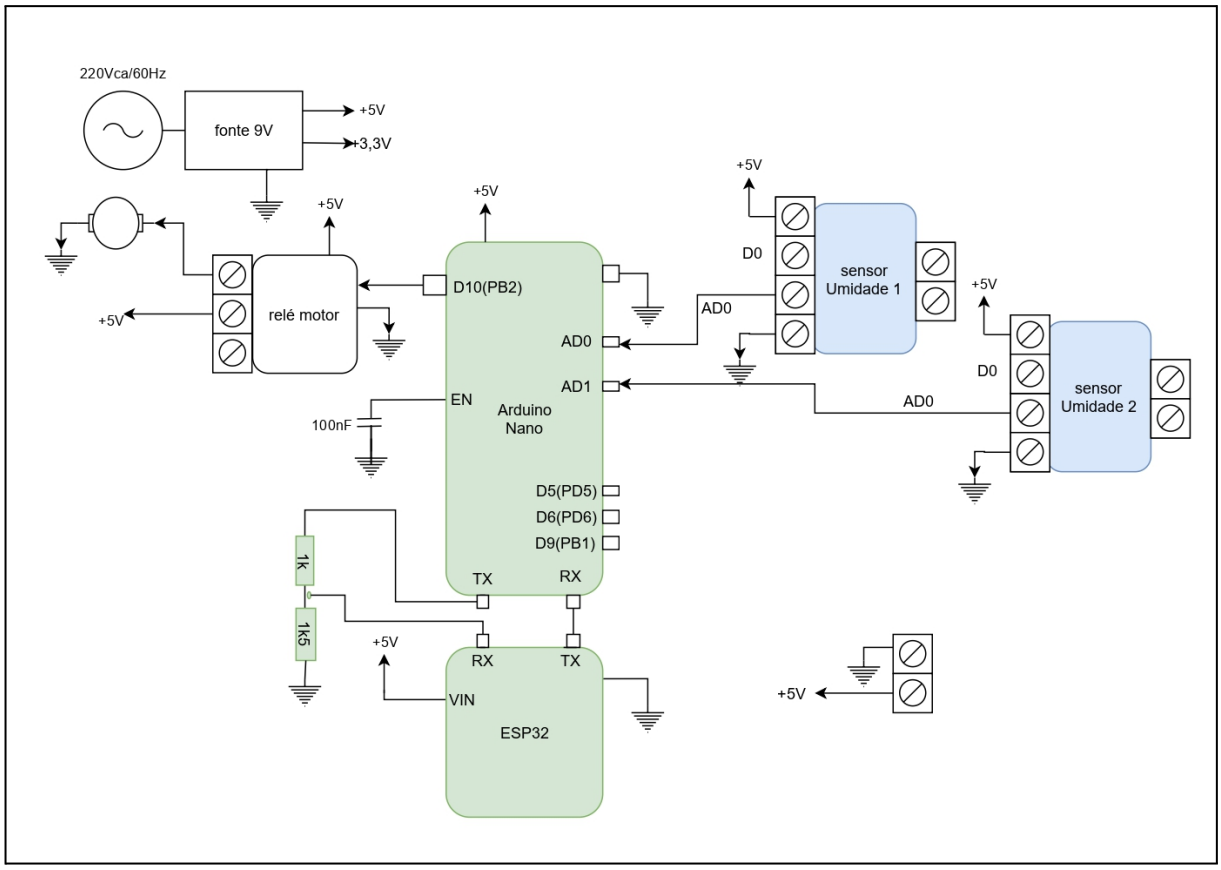
implementar o microcontrolador NodeMCU ESP8266, esse funcione como uma espécie de antena Wi-Fi. Sendo assim, o *hardware* final apresenta a integração dos componentes conforme a Figura 31, esquema elétrico de acordo com a Figura 32 e a montagem final como demonstrado na Figura 33.

Figura 31 - Esquema de integração dos componentes do *hardware*.



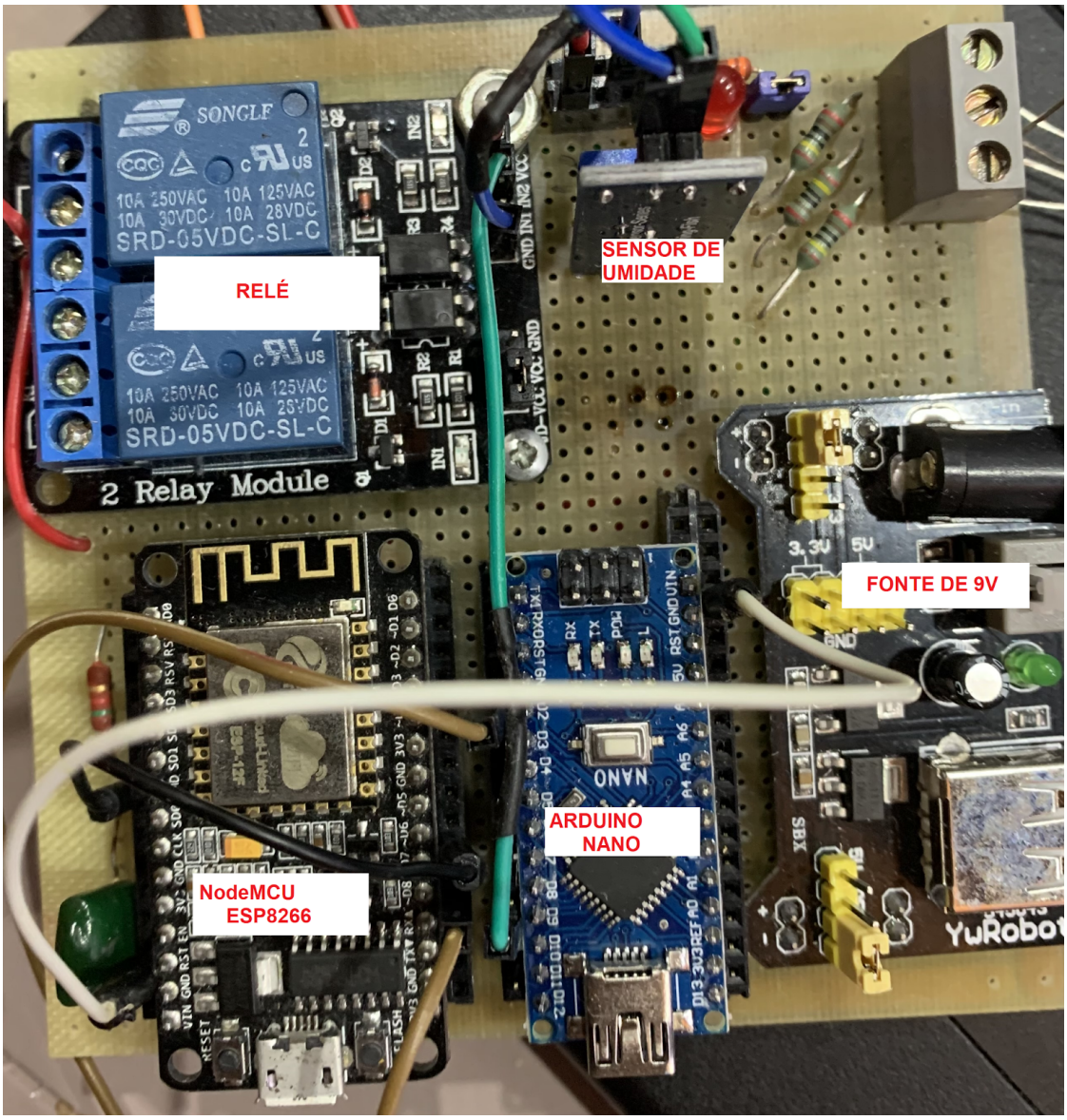
Fonte: Próprio autor.

Figura 32 - Esquemático elétrico do *hardware*.



Fonte: (NOLL, 2022).

Figura 33 - *Hardware*.



Fonte: Próprio autor.

4.4. TESTES DE COMUNICAÇÃO SERIAL

Durante o desenvolvimento do projeto alguns testes foram realizados, um deles foi em relação à comunicação serial entre a ESP 8266, que será conectada à API para fins de recebimento e envio de comandos ou dados, e o outro com o Arduino Nano, responsável por enviar as informações coletadas dos sensores e receber comandos de acionamento da bomba d'água, por exemplo, tudo isso ocorre via serial. Foram criados dois programas na plataforma Arduino IDE, uma para NodeMCU ESP8266 e outra para o Arduino Nano, em que ambos apresentassem

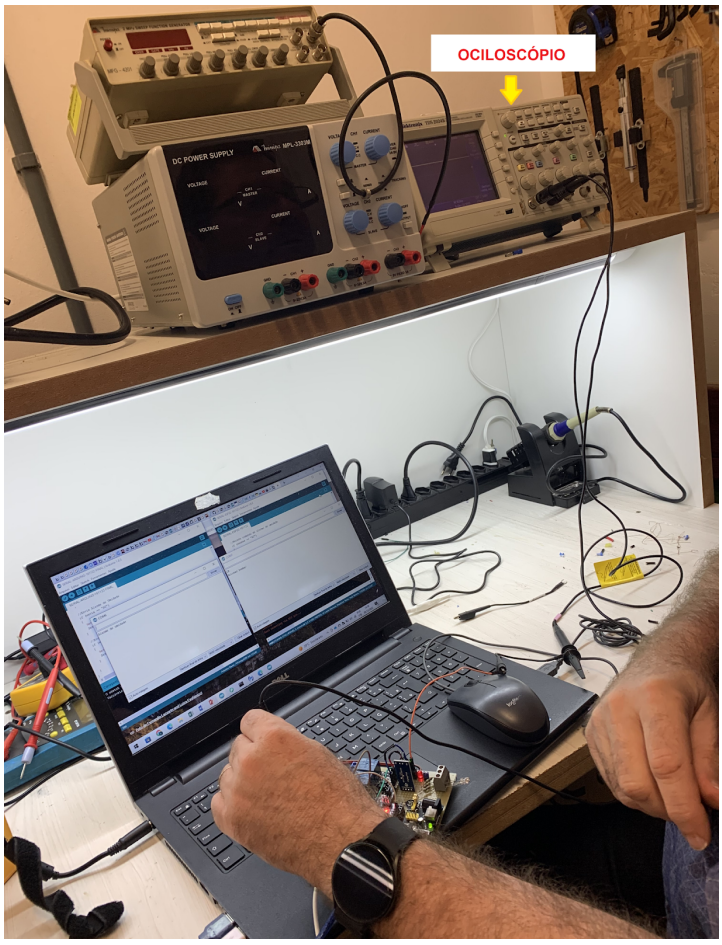
nas estruturas de seus códigos os comandos para apresentar no monitor serial as informações trocadas.

No entanto, houve alguns impasses ao longo dos testes, o primeiro deles foi a necessidade de criar no código arduino uma serial emulada com a biblioteca *SoftwareSerial.h*, ou seja, escolher outros pinos para comunicação serial com a ESP, que não fosse o Rx/Tx da placa, uma vez que elas já estariam sendo utilizados para comunicar-se via usb com o computador e mostrar as informações no monitor serial. Sendo assim, os pinos D2 e D3 do arduino tornaram-se Rx/Tx respectivamente.

O próximo obstáculo exigiu um pouco mais de dedicação para chegar a solução, dessa vez foi com a serial da ESP 8266. Ao olhar suas pinagens é possível observar que existe além dos pinos de Rx/Tx, os quais também estavam sendo utilizados para se comunicar via usb com o computador e também apresentar as informações no monitor de serial, há os pinos Rx2/Tx2, que aparentemente pareciam ser utilizados exclusivamente para comunicação serial. Nesse caso, apenas conectar os fios vindos da serial emulada do arduino com Rx2/Tx2 da ESP garantiria o funcionando, mas não funcionou corretamente.

Como a NodeMCU ESP8266 e o Arduino Nano trabalham com tensões diferentes, 3,3 V e 5V respectivamente, é necessário colocar resistores para fazer essa redução de tensão, para que se possa conectar o Tx do Arduino ao Rx do NodeMCU. Essa breve explicação sobre a diferença nas tensões desses componentes serve para compreender a realização do teste de sinal serial, feito utilizando osciloscópio, conforme mostra a Figura 34.

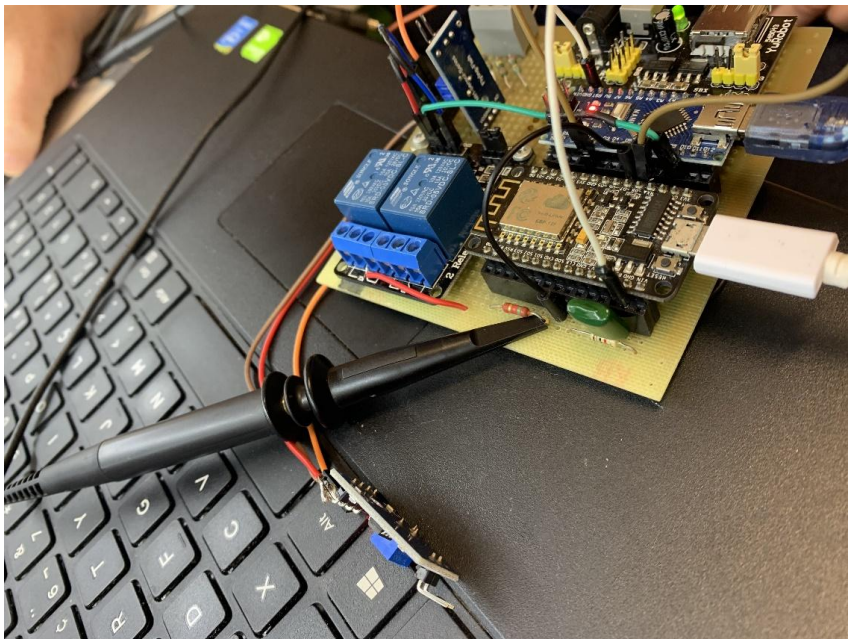
Figura 34 - Teste de sinal serial.



Fonte: Próprio autor.

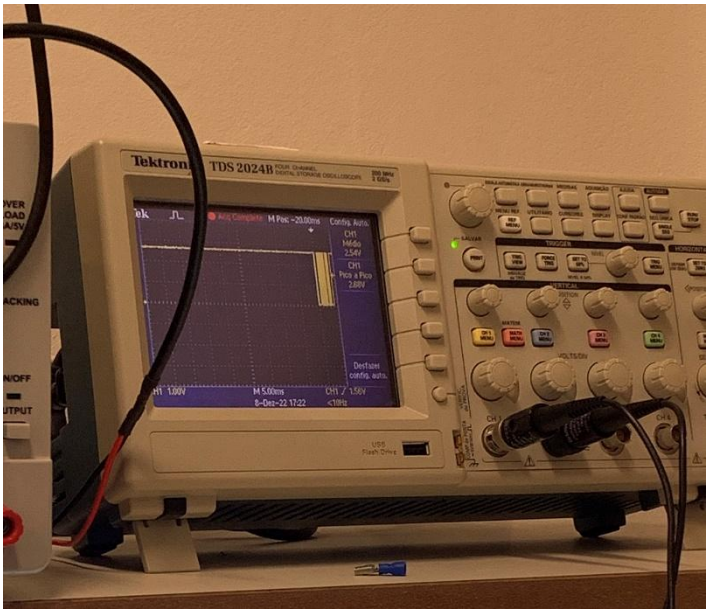
Ao colocar a ponteira do osciloscópio na saída do redutor de tensão, como pode-se observar na Figura 35, é possível constatar que o arduino está enviando o pulso serial, observado Figura 36, conclui-se que o pulso chega ao outro lado do divisor de tensão.

Figura 35 - Ponteira do osciloscópio na saída do redutor de tensão.



Fonte: Próprio autor.

Figura 36 - Sinal serial observado no osciloscópio.



Fonte: Próprio autor.

Como o pulso do sinal serial está presente, isso dá indícios da possível existência de algum erro na escolha da pinagem do microcontrolador NodeMCU ESP8266. De fato, tanto o NodeMCU como o Arduino Nano possuem apenas conjunto de pinos um Rx/Tx disponível, que neste caso estão sendo utilizados para comunicação com o computador.

Para contornar a falta das portas seriais chegou-se a conclusão que tanto do Arduino quanto o NodeMCU necessitam que seja criada uma serial emulada, podendo utilizar os pinos que correspondem ao Rx2/Tx2, que também precisam estar declarados no programa do NodeMCU, bem como no Arduino.

Neste caso, foi escolhido os pinos correspondentes na placa do NodeMCU e são eles: D7: Rx e D8:Tx. Já na programação no Arduino IDE a declaração desses pinos é um pouco diferente, ficando Rx: 13 (presente na descrição GPIO13 da ESP) e Tx: 15 (também da descrição GPIO15).

Esta comunicação serial existente entre o NodeMCU ESP8266 e o Arduino tem como função enviar os seguintes comandos do tipo *string*, inseridos nos códigos de programação das placas: <MA> (Modo Automático), <MM> (Modo Manual), <LB> (Ligar Bomba), <NU> (Nível de Umidade), <AU> (Alerta de Umidade), <UO> (Umidade OK), <NO> (Nível de Umidade OK), <BL> (Bomba Ligada), <BD> (Bomba Desligada).

4.5. COMUNICAÇÃO ENTRE O NodeMCU ESP8266 E A API

Os primeiros passos para iniciar essa conexão consiste em instalar no programa do NodeMCU ESP8266 no Arduino IDE as bibliotecas ESP8266WiFi.h, que permitem criar uma rede na própria NodeMCU e a PubSubClient.h, responsável pelas requisições de publicar e subscrever via MQTT. Em seguida, é preciso inserir as configurações do MQTT, como o endereço do servidor (Broker MQTT) e a porta utilizada. Além disso, é necessário declarar no programa os tópicos com seus respectivos endereços.

Toda vez que o Arduino detecta variação nos sensores ou faz o alguma variação no estado da bomba d’água, por exemplo, o mesmo envia, via comunicação serial para a placa do NodeMCU, a informação atualizada, que, por sua vez, realiza as conversões de linguagem necessárias para que então possa publicar (*publish*) os novos dados nos tópicos do *broker*.

No entanto, para realizar uma subscrição (*subscribe*) no NodeMCU, é preciso que em seu programa possua uma função denominada Callback, a qual “monitora” atualizações nos tópicos do *broker,* em caso afirmativo, essa função encontra o tópico atualizado, coleta a nova informação e a transforma numa string.

5. CONSIDERAÇÕES FINAIS

O trabalho em questão refere-se ao uso de plataformas de desenvolvimento de *software* com baixo ou sem uso de codificação para controle de umidade de vaso de plantas.

Inicialmente, realizou-se alguns estudos e testes a acerca do funcionamento das plataformas de desenvolvimento de *software* com baixo ou sem uso de codificação, principalmente no que diz respeito aos exemplos apresentados pelo professor orientador para compreender melhor os fluxos de trabalho, o que cada elemento faz e o que poderia ser extraído como base para esse projeto, uma vez que essas tecnologias ainda são de certa forma recentes e não existem tantas aplicações evidenciadas.

Na sequência, foi desenvolvida uma API com base em um exemplo apresentado pelo professor orientador. Essa API foi desenvolvida por meio da plataforma de desenvolvimento de *software* com baixo uso de codificação (Node-RED), cujo objetivo principal é servir de intermediário para troca de informação entre o *hardware* e o aplicativo. Já o aplicativo web foi criado em uma plataforma de desenvolvimento de *software* sem o uso de codificação (Bubble.io), tendo como principal função criar uma interface gráfica para que se possa interagir diretamente com o sistema.

Para tornar o *hardware* cedido pelo professor orientador útil para este projeto, inseriu-se microcontrolador NodeMCU ESP8266 ao *hardware* reaproveitado, conforme foi mencionado ao longo deste trabalho, pois isso serviu de base para realizar os testes de integração e fazer simulações, uma vez que agora o *hardware* conta com comunicação via Wi-Fi.

O projeto pôde ser validado, pois foram executadas as publicações e subscrições nos tópicos do *broker*, bem como as requisições via protocolo HTTP e MQTT na API, permitindo a interação entre o aplicativo web e o *hardware*. Os sensores realizaram as medições dos níveis umidade, enviando as informações para o Arduino, o qual as transmitiu via comunicação serial para o microcontrolador NodeMCU ESP8266 para que fosse acessada a API e fossem publicadas nos tópicos do *broker* as informações atualizadas.

Por ser um projeto que utiliza recursos bem atuais, como as plataformas de desenvolvimento de *software* e aplicações de IoT, conclui-se que ele pode servir de embasamento para outros projetos de automação, pois são inúmeras as possibilidades para implementar esses conceitos. Além disso, neste trabalho foram introduzidos alguns conceitos sobre plataformas de desenvolvimento de *software* com baixo ou sem o uso de codificação, integrado a uma aplicação IoT. Propõe-se, para um futuro trabalho, que seja desenvolvido *hardware* com mais componentes, que realize mais funções, com a criação de uma API mais completa, com mais *nodes* e rotas, além do desenvolvimento de um aplicativo mais completo a fim de explicar ainda mais as funcionalidades das plataformas, o que permitiria realizar, por exemplo, o monitoramento de um processo industrial.

REFERÊNCIAS

BACELAR, C. **O que significa desenvolvedor cidadão?** Disponível em: <https://www.pipefy.com/pt-br/blog/desenvolvedor-cidadao-citizen-developer/>. Acesso em: 28 out. 2022.

BODETTI, R. **Tiago Farias:** plataformas *low-code* devem representar 75% dos *softwares* até 2024 . Disponível em: <https://exame.com/bussola/tiago-farias-plataformas-low-code-devem-representar-75-dos-softwares-ate-2024/>. Acesso em: 29 out. 2022.

CANALTECH. O que é API? Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 6 nov. 2022.

CHEN, V. **Sobre o blog bolha-bolha**. Disponível em: <https://bubble.io/blog/about-bubble/>. Acesso em: 5 nov. 2022.

COPE, S. **Usando APIs MQTT para IOT** - guia para iniciantes. Disponível em: <http://www.steves-internet-guide.com/mqtt-apis-iot-guide/>. Acesso em: 3 nov. 2022.

DE CODAR, E. H. **Para que serve uma Application Programming Interface (API)?** Disponível em: <https://horadecodar.com.br/2022/11/07/para-que-serve-uma-application-programming-interface-api/>. Acesso em: 15 nov. 2022.

CACPNRJ, P. **Protocolo de comunicação serial RS232:** Noções básicas, funcionamento e especificações . Disponível em: <https://capsistema.com.br/index.php/2020/12/18/protocolo-de-comunicacao-serial-rs232-nocoes-basicas-funcionamento-e-especificacoes/>. Acesso em: 15 nov. 2022.

EVANS, M.; NOBLE, J; HOCHENBAUM, J. **Arduino em Ação**. São Paulo: Novatec, 2013.

FARIAS, T. **Plataformas low-code devem representar 75% dos softwares até 2024**. Disponível em: <https://inforchannel.com.br/2022/11/10/plataformas-low-code-devem-representar-75-dos-softwares-ate-2024/>. Acesso em: 4 nov. 2022.

FREITAS, Artur Sousa. **Estudo e implementação do protocolo de comunicação MQTT aplicado a um sistema de automação predial**. Universidade Federal de Campina Grande/PB, 2017.

GARTNER. Forecasts Worldwide Low-Code Development Technologies Market to Grow 23% in 2021. Disponível em: <https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021>. Acesso em: 5 nov. 2022.

HIVEMQ. Interprise MQTT broker. Disponível em: <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe>. Acesso em: 5 nov. 2022.

JURKĖNAS, R. **Top 6 Bubble competitors you should try right now**. Code or No Code, 11 fev. 2022. Disponível em: <https://codeornocode.com/comparisons/top-bubble-competitors-you-should-try/>. Acesso em: 15 nov. 2022

LEA, R. **Node-RED**: Lecture 1 – A brief introduction to Node-RED – Node RED Programming Guide. Disponível em: <http://noderedguide.com/nr-lecture-1/>. Acesso em: 23 nov. 2022.

LOCATELLI, C. **Introdução ao MQTT**. 14 jan. 2020. Disponível em: <https://curtocircuito.com.br/blog/introducao-ao-mqtt>. Acesso em: 14 nov. 2022

MELO, D. **O que é HTTP?** Disponível em: <https://tecnoblog.net/responde/o-que-e-http/>. Acesso em: 9 nov. 2022.

MEDEIROS, Pedro Henrique Silva. **Sistema de irrigação automatizado para plantas caseiras**. Universidade Federal de Ouro Preto/MG, 2018.

MOURA JUNIOR, Anderson David de. **Automação residencial de baixo custo**. Faculdade de Tecnologia e Ciências Sociais Aplicadas (FATECS), Brasília/DF, 2020.

NODE-RED: Lecture 1 – A brief introduction to Node-RED – Node RED Programming Guide. Disponível em: <http://noderedguide.com/nr-lecture-1/> Acesso em: 28 out. 2022.

NOVATEC. Arduino em Ação. Disponível em: <https://s3.novatec.com.br/capitulos/capitulo-9788575223734.pdf>. Acesso em: 14 nov. 2022.

OLIVEIRA, A. G. G. de. **Construção de Aplicações Distribuídas Utilizando-se de APIs REST**. Universidade do Estado do Rio Grande do Norte (UERN), 2018.

OLIVEIRA, E. **Como usar com Arduino** - Sensor (Medidor) de Umidade do Solo (Higrômetro). Blog Masterwalker, Shopmasterwalker Electronic Shop, 24 out. 2018. Disponível em: <https://blogmasterwalkershop.com.br/arduino/como-usar-com-arduino-sensor-medidor-de-umidade-do-solo-higrometro>. Acesso em: 11 nov. 2022.

PEREIRA, Daniel. **Desenvolvimento de plataforma para controle do laser.** YLPN-1-1x120-50-M. Instituto Federal de Educação e Tecnologia de Santa Catarina, 2019.

PINTO, G. **Por que é tão importante falar de Hardware na era dos Softwares?** Disponível em: <https://v2com.com/2020/03/03/hardware/>. Acesso em: 2 nov. 2022.

RECCHI, R. **Desmistificando o low-code e suas tendências para 2022**. Disponível em: <https://abes.com.br/desmistificando-o-low-code-e-suas-tendencias-para-2022/>. Acesso em: 9 nov. 2022.

REPULLO, R. **Low Code não é modinha.** É emprego certo. Disponível em: <https://www.convergenciadigital.com.br/Carreira-.-Artigos/Low-Code-nao-e-modinha.-E-emprego-certo-59735.html>. Acesso em: 9 nov. 2022.

ROCHA, Marcelo Marques da. **Desenvolvimento open-source para internet das coisas** (arquiteturas para interfaces web e móvel) — Universidade Federal Fluminense, 2018

SANTOS, P. C. **Programação vs No-code**: O que nunca te disse. Disponível em: <https://blogdaengenharia.com/secoes/colunistas-blog-da-engenharia/programacao-vs-plataformas-no-code-o-que-voce-nunca-soube/>. Acesso em: 23 out. 2022.

SOUZA, K. de. **No-code: o que é?** Disponível em: <https://blog.zeev.it/o-que-e-no-code/#:~:text=No%2Dcode%20traduzindo%20para%20o>. Acesso em: 15 nov./2022.

TABNEWS. As plataformas no-code e low-code já se consagraram como o novo “boom” da tecnologia? Disponível em: <https://www.tabnews.com.br/leoandrade/as-plataformas-no-code-e-low-code-ja-se-consagraram-como-o-novo-boom-da-tecnologia> . Acesso em: 23 out. 2022.

TABNEWS. Revolução no-code: como a programação sem código pode auxiliar diferentes tipos de negócios? Disponível em: <https://www.tabnews.com.br/leoandrade/revolucao-no-code-como-a-programacao-sem-codigo-pode-auxiliar-diferentes-tipos-de-negocios>. Acesso em: 23 out. 2022.

TECHLABS. Is Low-Code No-Code the Future of Software Development?. Disponível em: <https://medium.com/geekculture/is-low-code-no-code-the-future-of-software-development-7b4ea6b64130> . Acesso em: 23 out. 2022.

THE No-Code Manifesto. Disponível em: <https://bubble.io/blog/no-code-manifesto/>. Acesso em: 12 nov. 2022.

WHAT is Low-Code/No-Code? Pros and Cons + Examples. Disponível em: <https://stratoflow.com/what-is-low-code-no-code/>. Acesso em: 13 nov. 2022.

WELL. Conheça o Bubble.io, uma das principais ferramentas no-code do mercado. Disponível em: <https://leoandrade.net/conheca-o-bubble-io-uma-das-principais-ferramentas-no-code-do-mercado/>. Acesso em: 10 nov. 2022.

ZAZUETA, F. S., SMAJSTRLA, A.G., & CLARK, G.A. **Irrigation system controllers**. University of Florida.USA, 1994.

APÊNDICE A – PROGRAMA UTILIZADO NO ARDUINO

```
/**
  Trabalho de Conclusão de Curso - Engenharia Mecatrônica.
  Novembro de 2022.
  Leonardo dos Santos Pacifico

  Código desenvolvido para um Arduino Uno com o intuito de enviar e receber
  dados de um ESP32.

  Pinos RX e TX utilizados:
  ESP32:
    - Rx: 13
    - Tx: 15
  Arduino:
    - Rx: 2
    - Tx: 3
*/

//Biblioteca
//Como o Arduino não possui uma segunda Serial nativa,
//é necessário utilizar uma biblioteca para "emular" uma segunda serial
#include <SoftwareSerial.h>

//Pinos Rx e Tx do Arduino
const byte rxPin = 2; //Rx
const byte txPin = 3; //Tx

SoftwareSerial SerialEmulado(rxPin, txPin); //Cria objeto "SerialEmulado" a partir da
biblioteca para emular a segunda serial

#define pino_umidade A0
#define pino_nivel A1


int valor_umidade;
int valor_nivel;
const int pino_bomba = 10;


void setup() {
  Serial.begin(9600); //Serial nativa do Arduino
  SerialEmulado.begin(9600); //Serial "emulada" do ESP32
  pinMode(pino_umidade, INPUT);
  pinMode(pino_nivel, INPUT);
  pinMode(pino_bomba, OUTPUT);

}
```

```
//Declaração de Variáveis:

String recebe; //String que recebe dados do ESP32 após convertida para receber apenas os
                2 primeiros caracteres
String envia; //String que envia para o ESP32 após convertida para ler apenas os 2
                primeiros caracteres
String comando; //String que estará se comunicando com o que estiver enviando na serial
                antes de ser convertida
String comando_recebido;


/**
Função para converter dados de nível e umidade para apenas números:

Como em casos de umidade ou nível o ESP32 e o Arduino irão enviar/receber Strings do
tipo "XXYYY", onde XX são letras e YYY são números (de 0-100),
é necessário tratar esse tipo de String para que busquem apenas os números.

Função substring(): irá retornar parte da string original. O primeiro índice começa em 0
(zero), então contará a partir (e inclusive) do terceiro caracter.
Como a String recebida será XXYYY, no final irá retornar apenas o valor YYY, convertido
para número inteiro com a função "toInt()";
*/



void loop() {

        //Le o valor do pino A0 do sensor
      valor_umidade = analogRead(pino_umidade);

         //Le o valor do pino A1 do sensor
      valor_nivel = analogRead(pino_nivel);


      //Nivel OK Reservatório
        if (valor_nivel > 0 && valor_nivel < 400){

           //Envia status da bomba de água ligada
           Serial.println(" Status: NIVEL OK");
           SerialEmulado.println("NO"); //Envia na Serial do ESP8266
           //Serial.println("NIVEL OK");
           delay(3000);
```

```
if(recebe == "MA"){

  Serial.println("Modo Automatico");



      //Solo umido
      if (valor_umidade > 0 && valor_umidade < 400){

          //Envia status da bomba de água desligada
          Serial.println(" Status: Solo umido");
          Serial.println("Status da bomba: Desligada"); //Envia na Serial do Arduino
          SerialEmulado.println("BD"); //Envia na Serial do ESP32
          SerialEmulado.println("UO");
          digitalWrite(pino_bomba, 0); /* Desliga */
          delay(2000);

        return(0);
      }


      //Solo com umidade OK
      if (valor_umidade > 400 && valor_umidade < 800){

          Serial.println(" UMIDADE SOLO OK");
          SerialEmulado.println("UO");
          digitalWrite(pino_bomba, 0); /* Desliga */
          delay(2000);

          return(0);

       }


       //Solo seco
       if (valor_umidade > 800 && valor_umidade < 1024){

           //Envia status da bomba de água ligada
           Serial.println(" Status: Solo seco");
           Serial.println("Status da bomba: Ligada"); //Envia na Serial do Arduino
           SerialEmulado.println("BL"); //Envia na Serial do ESP32

            //Envia Alarme de Umidade
            Serial.println("Alarme de umidade"); //Envia na Serial do Arduino
            SerialEmulado.println("AU"); //Envia na Serial do ESP32
            digitalWrite(pino_bomba, 1); /* liga */
            delay(4000);

            return(0);

         }


    }
```

```
        if (recebe == "MM"){

              Serial.println("MODO MANUAL");

               if (recebe == "LB"){

                  digitalWrite(pino_bomba, 1); /* liga */
                  Serial.println(" Status: BOMBA LIGADA");
                  SerialEmulado.println("BL"); //Envia na Serial do ESP32

                }

                if (recebe == "DB"){

                   digitalWrite(pino_bomba, 0); /* liga */
                   Serial.println(" Status: BOMBA DESLIGADA");
                   SerialEmulado.println("BD"); //Envia na Serial do ESP32

                 }

            }


     }



        //Nivel Baixo Reservatório
        if (valor_nivel > 800 && valor_nivel < 1024){

           //Envia status nivel baixo
           Serial.println(" Status: NIVEL BAIXO");
           SerialEmulado.println("AN"); //Envia na Serial do ESP8266
           digitalWrite(pino_bomba, 0);
           delay(5000);
        }


 if (SerialEmulado.available() > 0) //Se a serial "emulada" do ESP32 estiver disponível
  {
    comando_recebido = SerialEmulado.readStringUntil('\n'); //Irá ler até o final da linha o que
estiver na Serial do ESP
    recebe = comando_recebido.substring(0,2); //Irá ler a substring do primeiro caracter
(índice zero) até, exclusive, o terceiro (índice dois)
    Serial.println(recebe); //Irá publicar na serial do Arduino o que o ESP32 está enviando


  }
```

APENDICE B – PROGRAMA UTILIZADO NA ESP 8266

```
/**
  Trabalho de Conclusão de Curso - Engenharia Mecatrônica.
  Novembro de 2022.
  Leonardo dos Santos Pacifico

  Código desenvolvido para uma ESP32 com o intuito de enviar e receber
  dados de um Arduino e publicar em broker MQTT.

  Pinos RX e TX utilizados:
  ESP32:
    - Rx: 13
    - Tx: 15
  Arduino:
    - Rx: 2
    - Tx: 3
*/

//Bibliotecas
#include <ESP8266WiFi.h> //Biblioteca conexão WIFI
#include <PubSubClient.h> //Biblioteca MQTT

#include <SoftwareSerial.h>

//Pinos Rx e Tx do Arduino
const byte rxPin = 13; //Rx
const byte txPin = 15; //Tx

SoftwareSerial SerialEmulado(rxPin, txPin); //Cria objeto "SerialEmulado" a partir da
biblioteca para emular a segunda serial


//Parâmetros Wifi
#define ssid "Casa123_1" //SSID Rede wifi
#define password "satellite1" //Senha wifi

//Configurações MQTT
const char *mqtt_server = "broker.hivemq.com"; //Broker MQTT
const char *mqtt_user = "admin";
const char *mqtt_password = "nersdc319b";
const int mqtt_port = 1883; //Porta MQTT

//Tópicos MQTT
const char *s_bomba = "myapi/response/s_bomba"; //Status da bomba de água
const char *a_nivel = "myapi/command/a_nivel"; //Alarme nível de água
const char *a_umidade = "myapi/command/a_umidade"; //Alarme umidade
const char *set_nivel = "myapi/response/set_nivel"; //Nível desejado de água
const char *set_modo = "myapi/response/set_modo"; //Modo Automático (ON) ou
Manual/Semi Automático (OFF)
const char *Lbomba = "myapi/response/Lbomba"; //Comando para ligar a bomba
const char *Dbomba = "myapi/response/Dbomba"; //Comando para desligar a bomba
```

```
//Declaração de Variáveis
String ligaMQTT = "{\"state\": \"ON\"}"; //String que será enviada ao broker MQTT
String desligaMQTT = "{\"state\": \"OFF\"}"; //String que será enviada ao broker MQTT

int nivel_umidade;

WiFiClient espClient; //Objeto da rede Wifi
PubSubClient client(espClient); //Instanciar o MQTT com o objeto de Wifi criado
anteriormente

//Função de Callback MQTT
  void callback(char* topic, byte* payload, unsigned int length){

  //armazena msg recebida em uma sring
  payload[length] = '\0';
  String strMSG = String((char*)payload);
  String strTopic = String(topic);

  Serial.print("Mensagem chegou do tópico: ");
  Serial.println(topic);
  Serial.print("Mensagem:");
  Serial.print(strMSG);
  Serial.println();
  Serial.println("-----------------------");


    if (strTopic.substring(15, 23) == "set_modo") {


       if (strMSG.substring(10,12) == "ON") {

          Serial.print("Modo Automatico");
          SerialEmulado.println("MA"); //Envia na Serial do Arduino

        }

       if (strMSG.substring(10,13) == "OFF") {

           Serial.print("Modo Manual");
           SerialEmulado.println("MM"); //Envia na Serial do Arduino
       }

     }

      if (strTopic.substring(15, 21) == "Lbomba"){

          if (strMSG.substring(10,12) == "ON"){

                Serial.println("Acionar bomba!"); //Envia na Serial da ESP32
                SerialEmulado.println("LB"); //Envia na Serial do Arduino

          }
       }
```

```
      if (strTopic.substring(15, 21) == "Dbomba"){

          if (strMSG.substring(10,12) == "ON") {

             Serial.println("Desligar bomba!"); //Envia na Serial da ESP32
             SerialEmulado.println("DB"); //Envia na Serial do Arduino

           }

      }

 }
void setup() {
 Serial.begin(9600); //Serial padrão ESP32
 SerialEmulado.begin(9600); //Serial Arduino (A ESP32 possui pode utilizar até 3 Seriais de
comunicação)

 //Conexão wifi
 WiFi.begin(ssid, password); //Conectar a rede wifi
 while (WiFi.status() != WL_CONNECTED) //Caso não esteja conectado na rede Wifi
 {
  delay(500);
  Serial.println("Conectando ao wifi...");
 }
 Serial.println("Conectado a rede wifi!");

 //Conexão MQTT
 client.setServer(mqtt_server, mqtt_port); //Conectar ao broker MQTT
 client.setCallback(callback); //Utilização da função de Callback do MQTT


 while (!client.connected()) //Enquanto não estiver conectado ao broker MQTT
 {
  String client_id = "ESP8266_TCC";
  client_id += String(WiFi.macAddress());
  Serial.printf("Client %s se conecta ao broker.\n", client_id.c_str());

  //Se for conectado com sucesso
  if (client.connect(client_id.c_str(), mqtt_user, mqtt_password))
  {
   Serial.println("Conectado ao broker!");
  }
  else //Se houver falha ao se conectar ao broker MQTT
  {
   Serial.print("Falha ao conectar com estado ");
   Serial.println(client.state());
   delay(1000);
  }
 }
```

```
  client.subscribe(s_bomba);
  client.subscribe(set_nivel);
  client.subscribe(set_modo);
  client.subscribe(Lbomba);
  client.subscribe(Dbomba);

}

//Variáveis para comunicação entre ESP32 e Arduino
String envia; //String que envia para o Arduino após convertida para ler apenas os 2 primeiros caracteres
String recebe; //String que recebe dados do Arduino após convertida para receber apenas os 2 primeiros caracteres

String comando; //String que estará se comunicando com o que estiver enviando na serial antes de ser convertida
String comando_recebe; //String que estará se comunicando com que estiver recebendo da serial vindo do Arduino antes de ser convertida

//String comando_bubble;
//String envia_bubble;

/**
Função para converter dados de nível e umidade para apenas números:

Como em casos de umidade ou nível o ESP32 e o Arduino irão enviar/receber Strings do tipo "XXYYY", onde XX são letras e YYY são números (de 0-100),
é necessário tratar esse tipo de String para que busquem apenas os números.

Função substring(): irá retornar parte da string original. O primeiro índice começa em 0 (zero), então contará a partir (e inclusive) do terceiro caracter.
Como a String recebida será XXYYY, no final irá retornar apenas o valor YYY, convertido para número inteiro com a função "toInt()";
*/

int nivel_envia()
{
  String agua = comando.substring(2);
  int valorU = agua.toInt();
  return valorU;

}

void loop() {

  client.loop(); //Função para que o processo de envio e recebimento no MQTT esteja sempre funcionando
```

```
  //Comandos enviados para o Arduino

  if (Serial.available() > 0) //Se a serial do ESP32 estiver disponível
  {
    comando = Serial.readStringUntil('\n'); //Irá ler até o final da linha
    envia = comando.substring(0,2); //Irá ler a substring do primeiro caracter (índice zero) até,
exclusive, o terceiro (índice dois)
    //comando_bubble = client.readStringUntil('\n');
    //envia_bubble = comando_bubble.substring(0,2);

    //Enviar comando de acionar a bomba
    if (envia == "LB")
    {
      Serial.println(envia); //Envia na Serial da ESP32
      Serial.println("Acionar bomba!"); //Envia na Serial da ESP32
      SerialEmulado.println("Acionar bomba!"); //Envia na Serial do Arduino
      SerialEmulado.println(envia); //Envia na Serial do Arduino
      client.publish(Lbomba, String(ligaMQTT).c_str()); //Publicará no tópico "Lbomba" a string
"ligaMQTT"
      client.publish(Dbomba, String(desligaMQTT).c_str()); //Publicará no tópico "Dbomba" a
string "desligaMQTT"
    }

    //Enviar comando de desaacionar a bomba
    if (envia == "DB")
    {
      Serial.println(envia); //Envia na Serial da ESP32
      Serial.println("Desligar bomba!"); //Envia na Serial da ESP32
      SerialEmulado.println("Desligar bomba!"); //Envia na Serial do Arduino
      SerialEmulado.println(envia); //Envia na Serial do Arduino
      client.publish(Dbomba, String(ligaMQTT).c_str()); //Publicará no tópico "Dbomba" a string
"ligaMQTT"
      client.publish(Lbomba, String(desligaMQTT).c_str()); //Publicará no tópico "Lbomba" a
string "desligaMQTT"
    }

    //Enviar comando para ativar Modo Automático
    if (envia == "MA")
    {
      Serial.println("Modo automatico"); //Envia na Serial da ESP32
      Serial.println(envia); //Envia na Serial da ESP32
      SerialEmulado.println("Modo automatico"); //Envia na Serial do Arduino
      SerialEmulado.println(envia); //Envia na Serial do Arduino
      client.publish(set_modo, String(ligaMQTT).c_str()); //Publicará no tópico "set_modo" a
string "ligaMQTT"
    }
```

```
 //Enviar comando para ativar Modo Manual/Semi Automático
    if (envia == "MM")
    {
      Serial.println("Modo manual"); //Envia na Serial da ESP32
      Serial.println(envia); //Envia na Serial da ESP32
      SerialEmulado.println("Modo manual"); //Envia na Serial do Arduino
      SerialEmulado.println(envia); //Envia na Serial do Arduino
      client.publish(set_modo, String(desligaMQTT).c_str()); //Publicará no tópico "set_modo"
a string "desligaMQTT"
    }

    //Envia o nível desejado de umidade
    if (envia == "NU")
    {
      nivel_umidade = nivel_envia(); //Utiliza a função "nivel_envia()" para realizar a conversão
para que o Arduino receba apenas um número
      Serial.print("Nivel de umidade: "); //Envia na Serial da ESP32
      Serial.println(nivel_umidade); //Envia na Serial da ESP32
      SerialEmulado.print("Nivel de umidade: "); //Envia na Serial do Arduino
      SerialEmulado.println(nivel_umidade); //Envia na Serial do Arduino
      String novoUmidade = String(nivel_umidade).c_str(); //Converte o nível enviado da
variável nivel_umidade (valor inteiro) para uma String
      String nivelMQTT = "{\"nivel\": " + novoUmidade + '}'; //Tratamento da String
"novoUmidade" para receber no broker MQTT
      client.publish(set_nivel, String(nivelMQTT).c_str()); //Publicará no tópico "set_nivel" o
nível de umidade com a String "nivelMQTT"
    }

  }

  //Dados recebidos do Arduino
  if (SerialEmulado.available() > 0) //Se a Serial do Arduino estiver disponível
  {

    comando_recebe = SerialEmulado.readStringUntil('\n'); //Irá ler até o final da linha o que
estiver na Serial do Arduino
    recebe = comando_recebe.substring(0,2); //Irá ler a substring do primeiro caracter (índice
zero) até, exclusive, o terceiro (índice dois)

    //Recebe comando de alarme de umidade
    if (recebe == "AU")
    {
      client.publish(a_umidade, String(ligaMQTT).c_str()); //Publica no tópico "a_umidade" o
alarme de umidade baixa a string "ligaMQTT"
      Serial.println("Enviado status ALARME UMIDADE ao MQTT"); //Escreve na Serial do
ESP32
    }

    //Recebe comando de umidade OK
    if (recebe == "UO")
    {
      client.publish(a_umidade, String(desligaMQTT).c_str()); //Publica no tópico "a_umidade"
o alarme de umidade baixa a string "ligaMQTT"
      Serial.println("Enviado status UMIDADE OK ao MQTT"); //Escreve na Serial do ESP32
```

```
    }

    //Recebe comando de alarme de nível
    if (recebe == "AN")
    {
      client.publish(a_nivel, String(ligaMQTT).c_str()); //Publica no tópico "a_nivel" o alarme de
nivel baixo de água a string "ligaMQTT"
      Serial.println("Enviado status ALARME NIVEL ao MQTT"); //Escreve na Serial do ESP32
    }

    //Recebe comando de nível OK
    if (recebe == "NO")
    {
      client.publish(a_nivel, String(desligaMQTT).c_str()); //Publica no tópico "a_nivel" o
alarme de nivel baixo de água a string "ligaMQTT"
      Serial.println("Enviado status NIVEL OK ao MQTT"); //Escreve na Serial do ESP32
    }

    //Recebe status da bomba de água ligada
    if (recebe == "BL")
    {
      client.publish(s_bomba, String(ligaMQTT).c_str()); //Publica no tópico "s_bomba" o status
da bomba de água ligada com a string "ligaMQTT"
      Serial.println("Enviado o status da bomba LIGADO ao MQTT"); //Escreve na Serial do
ESP32
    }

    //Recebe status da bomba de água desligada
    if (recebe == "BD")
    {
      client.publish(s_bomba, String(desligaMQTT).c_str()); //Publica no tópico "s_bomba" o
status da bomba de água desligada com a string "desligaMQTT"
      Serial.println("Enviado o status da bomba DESLIGADO ao MQTT"); //Escreve na Serial
do ESP32
    }

  }

}
```