UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia
Câmpus de Sorocaba

MATHEUS HIPÓLITO PROENÇA

Arquitetura IoT para Aplicação em Smart Campus

Sorocaba

2022

MATHEUS HIPÓLITO PROENÇA

ARQUITETURA IOT PARA APLICAÇÃO EM SMART CAMPUS

Trabalho de Conclusão de Curso apresentado ao Instituto de Ciência e Tecnologia de Sorocaba, Universidade Estadual Paulista (UNESP), como parte dos requisitos para obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Orientador:
Prof. Dr. Eduardo Paciência Godoy

Sorocaba
2022

P964a

Proença, Matheus Hipólito
Arquitetura IoT para Aplicação em Smart Campus / Matheus Hipólito Proença. -- Sorocaba, 2022
72 p. : il., fotos

Trabalho de conclusão de curso (Bacharelado - Engenharia de Controle e Automação) - Universidade Estadual Paulista (Unesp), Instituto de Ciência e Tecnologia, Sorocaba
Orientador: Prof. Dr. Eduardo Paciência Godoy

1. Internet das coisas. 2. Automação. 3. Tecnologia da informação.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Ins Ciência e Tecnologia, Sorocaba. Dados fornecidos peloautor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia
Câmpus de Sorocaba

ARQUITETURA IOT PARA APLICAÇÃO EM SMART CAMPUS

MATHEUS HIPÓLITO PROENÇA

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO GRAU DE **BACHAREL EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

Prof. Dr. Mauricio Becerra Vargas Marques

Coordenador

BANCA EXAMINADORA:

Prof. Dr. EDUARDO PACIENCIA GODOY

Orientador/UNESP-Campus de Sorocaba

Prof. Dr. FERNANDO PINHABEL MARAFÃO

UNESP-Campus de Sorocaba

Prof. Me. LUCAS NUNES MONTEIRO

UNESP-Campus de Sorocaba

Abril de 2022

especialmente à minha mãe, que me inspirou e me incentivou aos estudos desde criança.

AGRADECIMENTOS

Primeiramente, agradeço a Deus por iluminar meu caminho nessa jornada e me dar forças nos momentos de maiores desafios. Agradeço à minha família, minha irmã Natália, minha mãe Isabel que me acompanharam desde o início da minha trajetória e me apoiaram sempre. Agradeço à minha namorada Raissa, por todo o apoio e suporte nos momentos de maiores desafios e me incentivar em mais uma conquista. Agradeço a meu amigo Selkon, pelo suporte e empréstimo de equipamentos para realização do projeto. Finalmente, meus sinceros agradecimentos ao professor Eduardo Paciencia pelo grande aprendizado que pude adquirir por meio deste trabalho, agradeço pelas explicações, orientações e atenção prestada em diversos momentos. Agradeço também pelo empréstimo da Raspberry Pi e das placas ESP32 para realização do trabalho de forma remota.

“Sonhos determinam o que você quer. Ação determina o que você conquista.”

Aldo Novak

PROENÇA, M. H. **Arquitetura IoT para Aplicação em Smart Campus**. Trabalho de Graduação (Engenharia de Controle e Automação) – Instituto de Ciência e Tecnologia de Sorocaba, UNESP - Universidade Estadual Paulista, Sorocaba, 2022.

RESUMO

Com o rápido avanço da tecnologia *wireless* nos últimos anos, a Internet das Coisas (*Internet of Things* - IoT) tem se tornado cada vez mais relevante. A IoT está relacionada à conexão de diversos dispositivos e troca de dados entre si por meio da Internet. Nesse contexto, enquadra-se o conceito de *smart space*, isto é, um local inteligente capaz de perceber mudanças no ambiente, por meio do constante fluxo de dados entre os dispositivos nele presentes, e realizar determinadas ações relacionadas a tais mudanças. As cidades inteligentes (*smart cities*) são exemplos de *smart spaces*. Nessas cidades, semáforos trocam de estado de acordo com a presença ou ausência de carros e usuários conectados a Internet são informados quanto a desvios em percursos, devido a acidentes ocorridos. Outro exemplo de *smart space* são os *smart campus*. *Smart Campus* é um ambiente acadêmico que utiliza a IoT para facilitar a vida de pessoas que convivem diariamente nesse espaço, desenvolver o sistema didático entre professores e alunos, e auxiliar no gerenciamento do campus de forma mais eficiente e sustentável. Apesar das diversas vantagens do *Smart Campus*, sua implementação é de alto custo. Dessa forma, faz-se necessário o desenvolvimento de uma implementação de menor custo. Neste trabalho, foi implementada uma arquitetura IoT de baixo custo, utilizando-se de tecnologias *open source* (acesso aberto), que servirá de suporte para o desenvolvimento do *Smart Campus* na UNESP Sorocaba. A arquitetura IoT é composta por cinco componentes: dispositivo, rede, integração, armazenamento de dados e interação. No desenvolvimento deste trabalho, foram utilizadas placas ESP32 WiFi e ESP32 LoRa como dispositivos, os protocolos de comunicação MQTT e LoRaWAN como redes, integração via Node-RED, armazenamento de dados por meio do InfluxDB e a interação é realizada no Grafana. Constatou-se que é possível construir uma arquitetura IoT para aplicação em *Smart Campus* por meio de dispositivos de baixo custo e *softwares open source*.

Palavras-chave: *Internet of Things*. MQTT. LoRaWAN. ESP32. Automação.

PROENÇA, M. H. **IoT Architecture for Application in Smart Campus**. Graduation Work (Control and Automation Engineering) – Institute of Science and Technology of Sorocaba, UNESP – Universidade Estadual Paulista, Sorocaba, 2022.

ABSTRACT

With the rapid advancement of wireless technology in recent years, the Internet of Things (IoT) has become increasingly relevant. IoT is related to connecting different devices and exchanging data with each other through the Internet. In this context, the changes of the intelligent space are framed, that is, an intelligent place capable of perceiving changes in the environment, through the concept of constant data flow between the devices present in it, and performing certain actions related to such flows. Smart cities are examples of smart spaces. In these cities traffic lights change state according to the presence or absence of cars and users, the Internet is diverted as to deviations in routes, due to incidents that have occurred. Another example of smart space is smart campuses. The Smart Campus is to facilitate the management of the environment that the Smart Campus is to facilitate the lives of the people who live in this space daily, to develop a didactic system between and the students, and to assist the campus in a more efficient and sustainable way. Despite the many advantages of Smart Campus, its implementation is expensive. Thus, it is necessary to develop a lower cost implementation. In this work, a low-cost IoT architecture was developed, using open source technologies (access), which will serve as an open support for the development of the Smart Campus at UNESP Sorocaba. The IoT architecture is composed of five components: device, network, integration, data storage and interaction. In the development of this work, ESP32 WiFi and ESP32 LoRa boards were used as devices, the MQTT and LoRaWAN communication protocols as networks, integration via Node-RED, data storage through InfluxDB and the interaction is performed in Grafana. It was found that it is possible to build an IoT architecture for application in Smart Campus through low-cost devices and open source software.

Keywords: Internet of Things. MQTT. LoRaWAN. ESP32. Automation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

MQTT	-	MQ Telemetry Transport
IoT	-	Internet of Things
M2M	-	Machine To Machine
TTN	-	The Things Network
HTTP	-	Hypertext Transfer Protocol
HTML	-	Hypertext Mark-up Language
TLS	-	Transport Layer Security
SSL	-	Secure Sockets Layer
TCP	-	Transmission Control Protocol
IP	-	Internet Protocol
RFID	-	Radio-Frequency Identification
LPWAN	-	Low Power Wide Area Network
CA	-	Certification Authority
IDE	-	Integrated Development Environment
BLE	-	Bluetooth Low Energy
SOC	-	System On Chip
TB	-	Terabyte
PWM	-	Pulse Width Modulation
API	-	Application Programming Interface
SSID	-	Service Set Identifier

SUMÁRIO

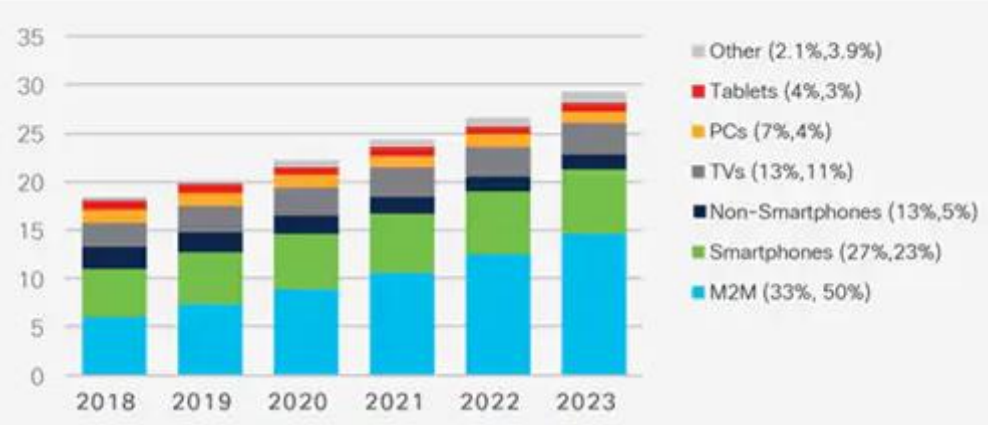
1 INTRODUÇÃO

1.1 Justificativa

Com o avanço de tecnologias *wireless*, como o Wi-Fi, por exemplo, aplicações IoT (*Internet of Things*) têm-se tornado cada vez mais relevantes.

Segundo Relatório Anual da CISCO (2018-2023), estima-se que haverão 29,3 bilhões de dispositivos conectados a Internet até 2023, dentre os quais mais da metade (14,7 bilhões) corresponderão a aplicações IoT, também chamadas M2M (*Machine To Machine*) (Figura 1).

Figura 1 – Crescimento de dispositivos conectados a Internet por categoria (em bilhões).



Fonte: Adaptado de Cisco Annual Internet Report (2020).

Atualmente, é possível encontrar aplicações IoT em diversos setores, tais como: agroindústria, indústria, transportes e segurança.

Na agroindústria, já é possível monitorar a saúde e os movimentos do gado em tempo real, a fim de se garantir alta qualidade da carne (MADAKAM, 2015; TALAVERA, 2017). Já no setor industrial, encontram-se aplicações relacionadas a controle e gerenciamento de estoque, bem como análises da produção em tempo real (CHENG, 2018). No setor de transportes, tem-se destacado o rastreamento de pacotes, enquanto que, no setor de segurança, vem se destacando a vigilância por vídeo (CISCO, 2020).

Dentro desse contexto, existem os ambientes inteligentes (do inglês *smart spaces*), em que estão presentes sensores, dispositivos e aplicativos interconectados via Internet que agem dinamicamente, tomando decisões de acordo com as condições analisadas. Dessa forma, é possível que o usuário conectado a esse ambiente receba alertas de condições não planejadas e monitore dados em tempo real (GILMAN et al., 2020).

Aplica-se a esse modelo de ambiente: as cidades inteligentes (*smart cities*) e as casas inteligentes (*smart homes*). Nas cidades inteligentes, semáforos mudam seu estado de acordo com o tráfego, além de serem emitidos alertas relacionados à poluição e eventuais acidentes aos usuários conectados (RIO, 2018). Já nas casas inteligentes, o alarme de um *smartphone* pode servir de gatilho para acionar uma cafeteira elétrica e uma tostadeira, adiantando, assim, o café da manhã (EVANS, 2011).

Enquadram-se nesse modelo de ambiente os chamados *smart campus*. O propósito do *smart campus* é construir, por meio da integração de tecnologias IoT, um ambiente sustentável, seguro e que auxilie a gestão administrativa, bem como o desenvolvimento acadêmico (XIONG, 2016).

Na UNICAMP, o projeto de *smart campus* foi iniciado em 2016. São exemplos de projetos já implatados: o *Smart Lock* e o *Smart Parking*. Por meio do projeto *Smart Lock* (Figura 2), foi desenvolvida uma fechadura eletromecânica capaz de travar e destravar portas por meio de etiqueta RFID (*Radio-frequency identification*) e QR Code (CLÉTO, 2021). Já no projeto *Smart Parking* (Figura 3), é realizado um monitoramento em tempo real do número de vagas livres no estacionamento da UNICAMP (BAGGIO, GONZALEZ, BORIN; 2020).

Figura 2 – Projeto *Smart Lock*.

Fonte: *Smart Campus* Unicamp, 2021.

Figura 3 – Projeto *Smart Parking*.

Fonte: (BAGGIO, GONZALEZ, BORIN; 2020, p.13).

Apesar das diversas vantagens, a implementação de um *smart campus* geralmente requer altos investimentos (SARI *et al*., 2017), o que pode ser uma barreira principalmente em Universidades Públicas. Nesse contexto, este trabalho visa o desenvolvimento de uma arquitetura IoT de baixo custo, por meio da utilização de tecnologias *open source* (acesso aberto), que servirá de suporte para o desenvolvimento do *Smart Campus* na UNESP Sorocaba.

1.2 Objetivos Gerais

Projetar e desenvolver uma arquitetura IoT de baixo custo, flexível, escalável e segura, que servirá de referência para a implementação do *Smart Campus* na UNESP Sorocaba.

1.3 Objetivos Específicos

Criar um *firmware* padronizado em *hardware* de baixo custo (ESP32) para conectividades MQTT e LoRaWAN.

Criar uma integração via Node-RED capaz de receber, organizar e escrever os dados em banco local (InfluxDB) automaticamente.

1.4 Organização do Trabalho

O presente trabalho está organizado em 6 capítulos. No primeiro capítulo, são apresentados conceitos iniciais sobre IoT, sua relevância no cenário atual, bem como os objetivos deste projeto.

No segundo capítulo, são apresentados conceitos teóricos chaves para o entendimento do desenvolvimento do trabalho.

Já no terceiro capítulo, é apresentada a arquitetura IoT proposta no trabalho, bem como explicações breves e especificações dos componentes utilizados.

No quarto capítulo, são apresentados os desenvolvimentos do *backend* e do *frontend* da arquitetura.

No quinto capítulo, são apresentados os resultados obtidos na implementação da arquitetura bem como testes para validação do sistema.

No sexto capítulo, são apresentadas as conclusões relativas ao desenvolvimento do projeto.

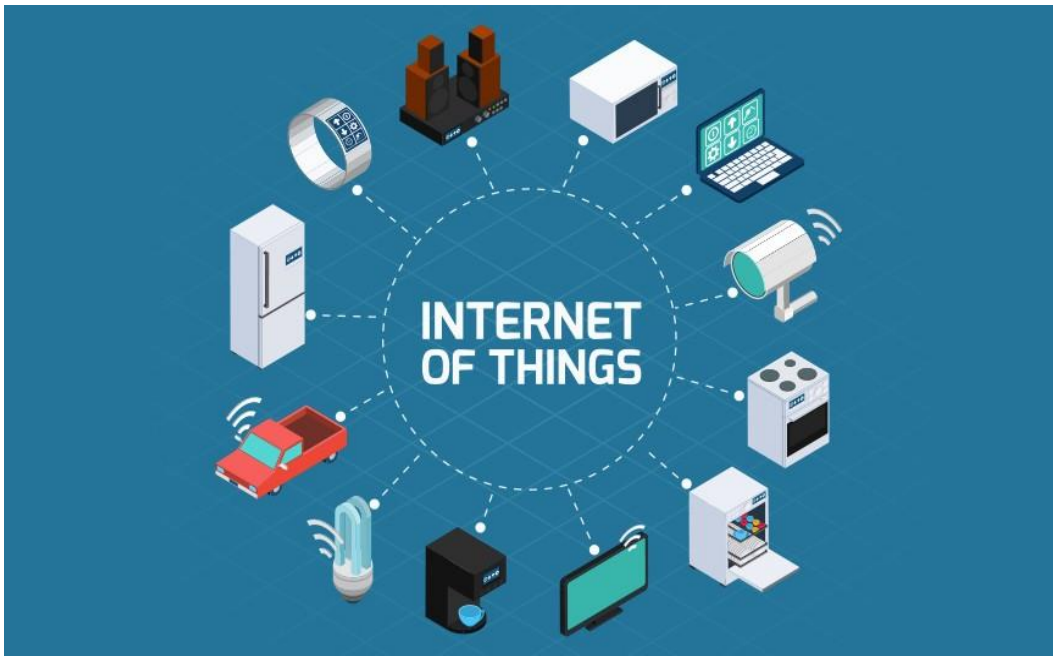
2 REFERENCIAL TEÓRICO

2.1 Internet das Coisas (IoT)

Segundo Madakam *et al*. (2015), Internet das Coisas ou simplesmente IoT (do inglês *Internet of Things*) pode ser definida como: “Uma rede aberta e abrangente de objetos inteligentes que têm a capacidade de se auto-organizar, compartilhar informações, dados e recursos, reagindo e agindo diante de situações e mudanças no ambiente”. Dessa forma, diversos dispositivos conectados a Internet são capazes de interagir entre si e com usuários de acordo com variações ocorridas no ambiente em que estão inseridos, sendo possível realizar monitoramentos, ajustes e análises de dados em tempo real.

O nome *Internet of Things* é composto pela junção de duas partes: *Internet* e *Things. Internet* se refere ao sistema global de redes de computadores interconectados por meio do padrão *Internet Protocol suite (TCP/IP)*, já *Things* pode se referir a dispositivos eletrônicos, tais como cafeteiras, televisões, geladeiras, ar condicionados, máquinas industriais; a objetos vestíveis, como, por exemplo, relógios e roupas e, ainda, pode se referir a pessoas (EVANS, 2011). Dessa forma, Internet das Coisas (*Internet of Things*) se refere a diversos dispositivos eletrônicos e usuários interconectados por meio da Internet como ilustrado na Figura 4.

Figura 4 – Dispositivos eletrônicos interconectados por meio da Internet.



Fonte: Seropédica Online (2021).

A Internet das Coisas vem sendo aplicada em diversos setores da sociedade, dentre os quais se destacam os seguintes: transporte, logística, saúde e ambientes inteligentes:

- **Transporte e logística:** por meio de etiquetas RFID (*Radio-frequency identification*) é possível monitorar em tempo real grande parte dos elos de uma cadeia de suprimentos (*supply chain*), desde matéria-prima, transporte, estoque, produto final até a distribuição aos pontos de venda. Além disso, por meio de sensores, é possível monitorar importantes variáveis durante o transporte de alimentos perecíveis, tais como temperatura e umidade, garantindo a qualidade dos produtos (KAUR;SINGH, 2016).
- **Saúde:** tecnologias IoT oferecem identificações de pacientes, que ajudam a evitar incidentes prejudiciais aos mesmos, como o oferecimento de um medicamento em hora errada, por exemplo. Por meio de etiquetas RFID, é possível automatizar a coleta de dados dos pacientes, reduzindo o tempo com preenchimento de formulários (KAUR;SINGH, 2016).
- **Ambientes inteligentes:** casas inteligentes propiciam um ambiente mais confortável e sustentável. O aquecimento pode ser adaptado de acordo com as preferências do indivíduo, as luzes se adaptam de acordo com o horário do dia, sistemas de alarmes garantem maior segurança e aparelhos eletrônicos são desligados automaticamente, caso não estejam sendo utilizados. Já em plantas industriais, etiquetas RFID permitem a transmissão de dados de um produto a robôs. Por meio do processamento dos dados recebidos, o robô saberá qual procedimento deverá ser executado (KAUR;SINGH, 2016).

2.1.1 *Smart Campus*

Segundo Xiong (2016), "*Smart Campus* é uma forma avançada de informatização universitária, também uma ampliação e melhoria do campus digital." Portanto, a infraestrutura do Campus Digital é desenvolvida por meio da implementação de tecnologias da informação, tais como Internet das Coisas, *Big Data*, armazenamento em nuvem, rede sem fio e internet móvel, evoluindo, assim, para um *Smart Campus* (NIE, 2013). No Quadro 1 a seguir, são destacadas as características do Campus Digital e do *Smart Campus*:

Quadro 1 – Características do Campus Digital e *Smart Campus*.

	Campus Digital	***Smart Campus***
Ambiente Técnico	• Rede Local • Internet	• IoT • Armazenamento em nuvem • Rede sem fio • Internet Móvel
Aplicação	• Biblioteca digital • Educação a Distância • Ensino digital	• Biblioteca inteligente • Ensino com realidade aumentada • *Smart Classroom*
Sistema de gestão	• Sistema isolado	• Sistema de compartilhamento • Sistema Inteligente

Fonte: Adaptado (NIE, 2013).

No cenário atual, o *Smart Campus* é uma tendência nas universidades. Por meio de sua implementação, é possível gerenciar os recursos de forma mais sustentável e eficiente (*smart management*) (Xiong, 2016). Além disso, educadores acreditam que novos métodos de ensino (*smart learning*), como aulas com realidade aumentada, em que alunos podem interagir com objetos 3D, objetos microscópicos e até com orgãos humanos virtuais, aumentam a interatividade dos alunos com o conteúdo, de forma que fortalecem o processo de aprendizagem (CHAN, 2018).

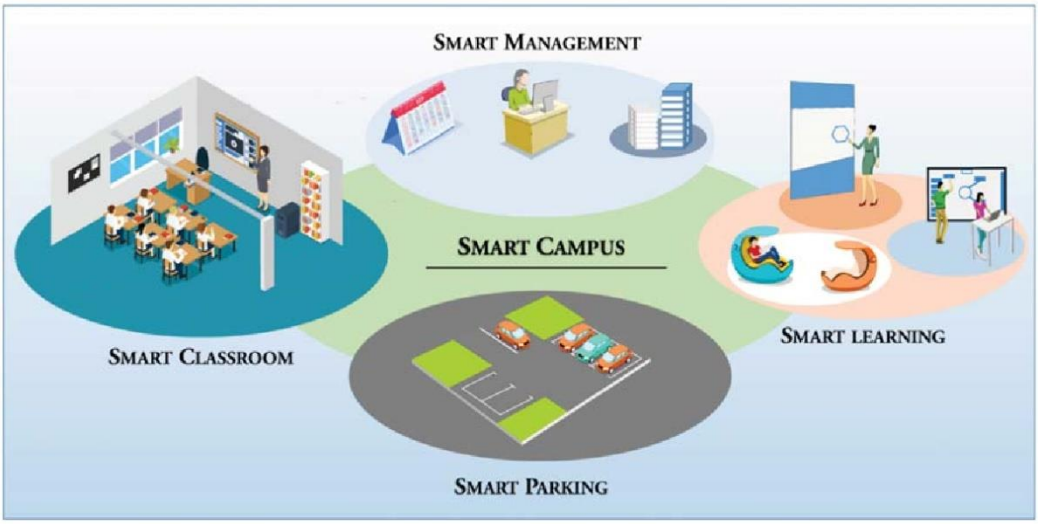
São alguns exemplos de aplicações práticas do *Smart Campus* (Figura 5):

- ***Smart classroom*:** nestas salas, há o controle da luz e do ar condicionado de forma autônoma de acordo com a presença ou ausência de pessoas no interior da

sala. É possível, ainda, verificar quais salas estão ocupadas e quais estão vazias (SARI, 2017).

- ***Smart parking***: é possível saber previamente quais vagas estão disponíveis para estacionar ou se o estacionamento já está lotado (SARI, 2017).
- **Acesso remoto a testes de laboratório:** por meio de etiquetas RFID é possível acessar remotamente resultados de experimentos (NIE, 2013).
- **Bibliotecas inteligentes:** é possível monitorar quantas pessoas frequentaram a biblioteca em um dia e gerar gráficos estatísticos; emitir alertas caso um livro esteja alocado em local incorreto na plateleira e também monitorar a localidade em tempo real de um livro (CHAN, 2018).

Figura 5 – Exemplos de aplicações práticas do *Smart Campus*.



Fonte: Semantic Scholar (2019).

2.2 Protocolo de comunicação MQTT

MQTT (*MQ TelemetryTransport*) é um protocolo de transporte de mensagens de publicação (*publish*) e assinatura (*subscribe*). Caracteriza-se por ser um protocolo binário, leve, aberto e fácil de implementar no lado do cliente. Tais características possibilitam desenvolvimentos de projetos que envolvam dispositivos com recursos limitados e em que a largura de banda da rede é limitada, sendo, portanto, ideal no contexto da IoT (HIVEMQ, 2015).

Inventado por Andy Stanford-Clark (IBM) e ArlenNipper (Arcom, agora Cirrus Link) em 1999, surgiu da necessidade de um protocolo que contemplasse perda mínima de bateria e

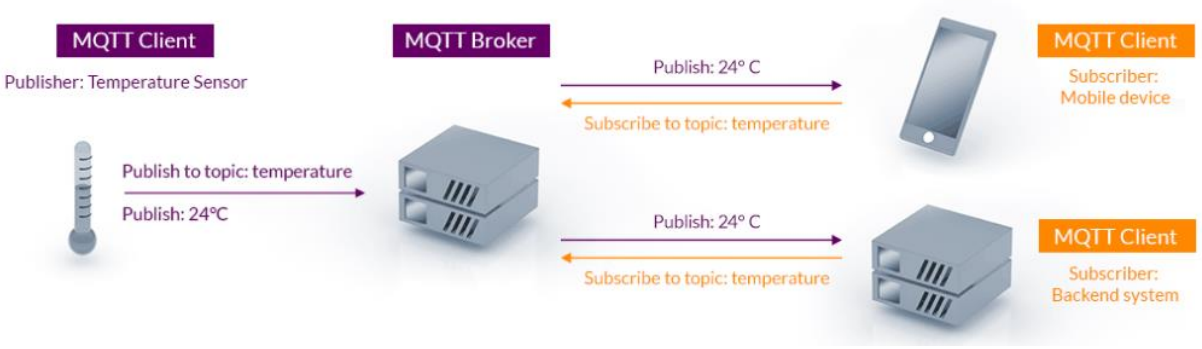
largura de banda mínima para realizar conexões com oleodutos via satélite (HIVEMQ, 2015).

Em 2014, o protocolo se tornou um padrão oficializado pela OASIS, uma organização aberta sem fins lucrativos que trabalha no desenvolvimento e adoção de padrões abertos para segurança cibernética, IoT, computação em nuvem, entre outras áreas (OASIS, 2020).

Como mencionado, o protocolo MQTT utiliza o modelo *publish/ subscribe*, também chamado de pub/sub, para transmissão de mensagens. Este modelo dissocia a transmissão direta entre quem envia mensagens (*Publisher*) e quem recebe (*Subscriber*), de forma que a conexão entre *Publisher* e *Subscriber* é realizada por meio do *Broker* (Figura 6) (HIVEMQ, 2015).

O *Broker* é responsável por receber todas as mensagens do *Publisher*, filtrá-las e transmiti-las corretamente para os *Subcribers* de acordo com os tópicos em que estes se increveram (HIVEMQ, 2015).

Figura 6 – Arquitetura *Publish/Subcribe* do MQTT.



Fonte: MQTT.org (2022).

Importante ressaltar três aspectos consequentes da dissociação de mensagens no protocolo MQTT:

- **Espaço:** há uma dissociação espacial entre *Publisher* e *Subcriber*, de forma que para enviar ou receber mensagens é necessário apenas conhecer o *hostname* / IP e a porta do *Broker* MQTT (HIVEMQ, 2015).
- **Tempo:** há uma dissociação temporal, pois, apesar da maioria das conexões ocorrerem em tempo real, é possível armazenar mensagens para clientes *offline* e entregá-las posteriormente ao se restaurar a conexão (HIVEMQ, 2015).
- **Sincronização:** o envio e recebimento de mensagens ocorrem de forma

assíncrona. Dessa forma, não é necessário aguardar uma mensagem ser recebida para enviar novas mensagens (HIVEMQ, 2015).

2.3 Protocolo de comunicação LoRaWAN

LoRaWAN é um protocolo de código aberto que define a arquitetura de rede e os parâmetros de comunicação a serem utilizados por tecnologia de rádio frequência, denominada LoRa (ALMEIDA;BORIN, 2019). LoRaWAN é categorizado como uma LPWAN (*Low Power Wide Area Network*), isto é, uma arquitetura de rede de baixo consumo de energia e longo alcance. Foi desenvolvida visando otimizar aspectos relativos à vida útil de bateria, alcance, capacidade de rede e custo (LORA, 2015).

O protocolo LoRaWAN define detalhes relativos a funcionamento, segurança, ajuste de potência (para maximização da duração das baterias dos módulos) e qualidade de serviço. Quanto à arquitetura de rede, pode ser dividida em 4 partes: módulos (*end nodes*), *gateway*, servidores de rede (*network server*) e servidores de aplicações (*application server*) (ALMEIDA;BORIN, 2019). A arquitetura de rede LoRaWAN é ilustrada na Figura 7 a seguir.

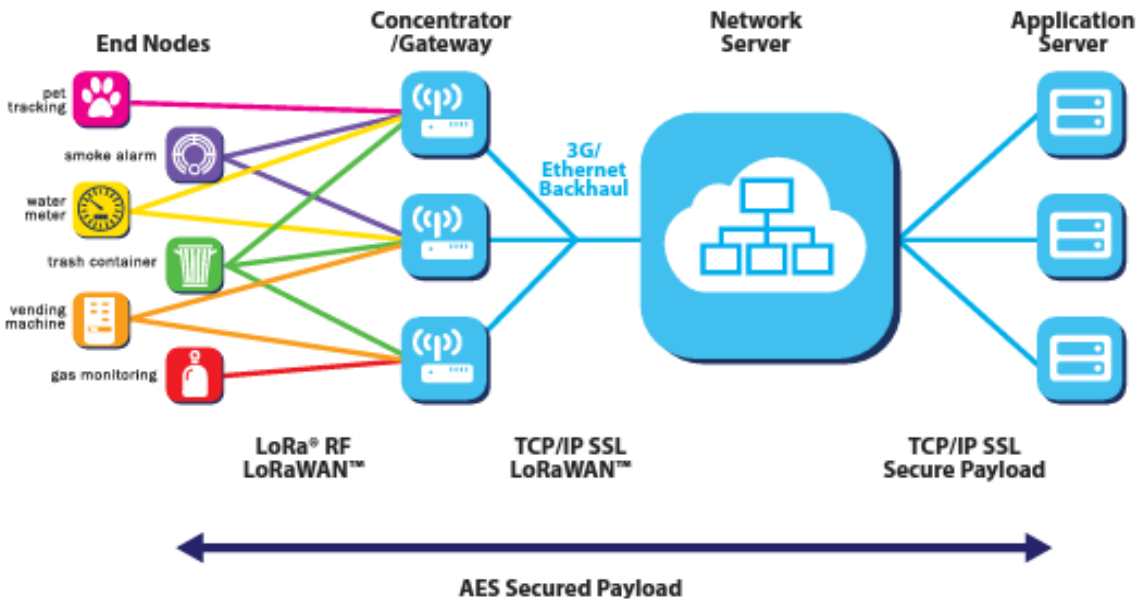
1. **Módulos (*end nodes*):** são elementos responsáveis por captar informações via sensores ou acionar dispositivos via atuadores. Dentro do protocolo LoRaWAN, são divididos em 3 classes:
    - **Classe A:** composta por sensores que operam por meio de baterias em consumo reduzido. A comunicação é bidirecional e a recepção dos pacotes de dados vindos do *gateway* são possíveis apenas após envio de informações pelo módulo.
    - **Classe B:** composta por atuadores que operam por meio de bateria. Neste caso, a recepção dos pacotes de dados vindos do *gateway* ocorre de forma agendada (recepção agendada), isto é, são trocadas informações entre módulo e *gateway* avisando se o módulo já está pronto para receber o pacote de dados.
    - **Classe C:** composta por dispositivos com alto consumo de energia que operam, geralmente, ligados a uma rede de energia elétrica. Realizam comunicação bidirecional e estão sempre disponíveis para receber dados vindo do *gateway*.

2. ***Gateway:*** são responsáveis por receber os dados vindos dos módulos e enviá-los aos servidores de rede. Possuem acesso à Internet geralmente via Wi-Fi / Ethernet ou via 3G/4G em locais mais remotos. Possuem ampla área de cobertura, sendo que um único gateway pode chegar a cobrir cidades e até países inteiros.

3. **Servidores de rede:** são responsáveis pelo gerenciamento de dados vindo do *gateway* e envio de tais dados para os servidores de aplicação. Dentre suas funcionalidades, destacam-se a eliminação de pacotes duplicados, ajustes das taxas de dados e gerenciamento de tempo entre comunicações e do consumo de energia.

4. **Servidores de aplicações:** são responsáveis pelo recebimento dos pacotes de dados dos servidores de rede e execução de ações específicas de acordo com a informação recebida.

Figura 7 – Arquitetura de rede LoRaWAN.



Fonte: (LORA ALLIANCE, 2015, p. 7).

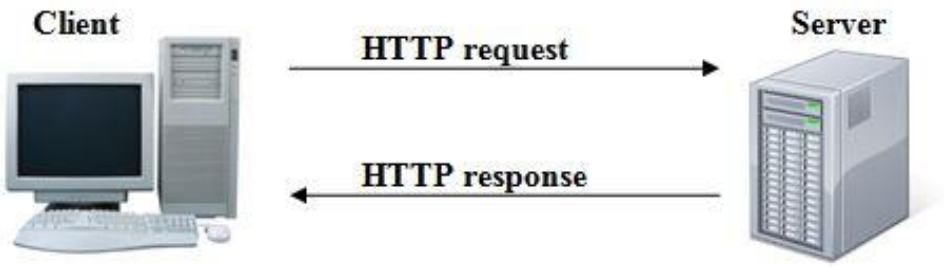
2.4 Protocolo HTTP

Hypertext Transfer Protocol (HTTP) é um protocolo de nível de aplicação para sistemas de informação hipermídia, distribuídos e colaborativos. Está em uso na *World Wide Web* desde 1990, sendo sua primeira versão denominada HTTP/0.9, caracterizada por um protocolo simples de transferência de dados brutos na Internet (FIELDING *et al*., 1999).

A comunicação é realizada entre cliente (*Client*) e servidor (*Server*) e ocorre 3 etapas: conexão, solicitação (*request*) e resposta (*response*), como ilustrado na Figura 8 (TIM BERNERS-LEE, 1996).

Inicialmente, o cliente faz uma conexão TCP/IP utilizando o nome de domínio ou IP e a porta do endereço. O servidor aceita a conexão. É enviada, então, uma solicitação “GET”, que tem como resposta uma mensagem em HTML (*Hypertext Mark-up Language*) pelo servidor. O processo se encerra com o fechamento da conexão pelo servidor (TIM BERNERS-LEE, 1996).

Figura 8 – Etapas do Protocolo HTTP.



Fonte: Programa En Línea (2022).

2.5 Protocolo de segurança TLS

TLS (*Transport Layer Security*) é um protocolo de segurança sucessor do antigo SSL (*Secure Sockets Layer*). É utilizado para garantir integridade e privacidade dos dados transmitidos entre duas aplicações, evitando-se espionagem, adulteração e falsificação de mensagens (DIERKS;RESCORLA, 2008).

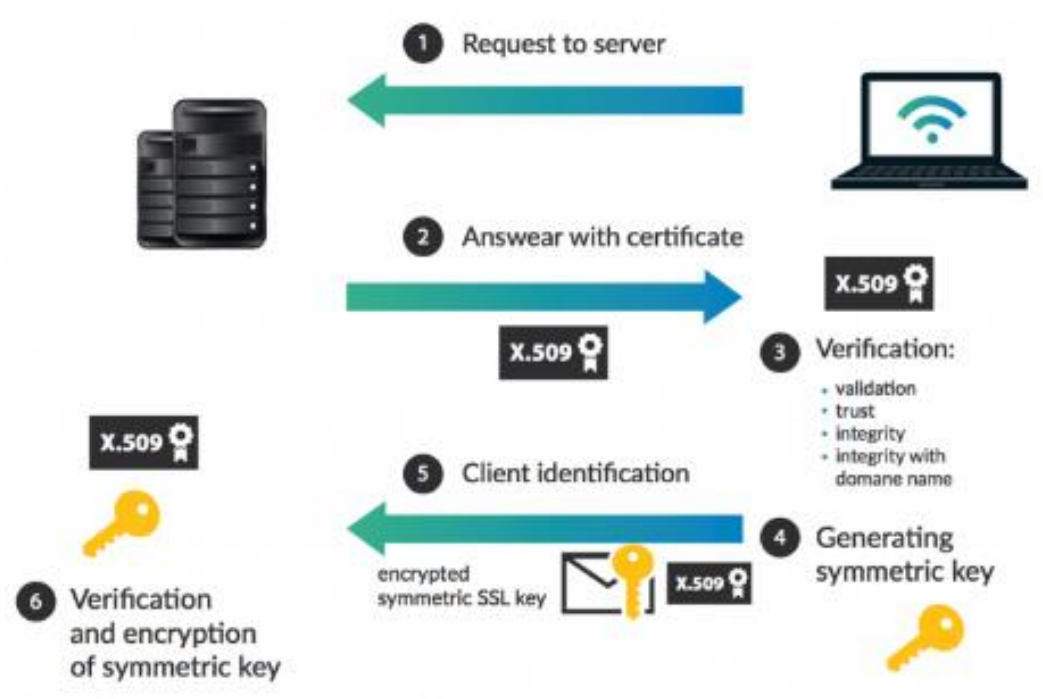
Como conexões entre servidor e cliente podem ser realizadas utilizando-se ou não do TLS, o cliente deve sinalizar ao servidor o uso do protocolo. Uma das formas de tal sinalização é por meio da utilização de portas diferentes (DIERKS;RESCORLA, 2008).

Após a concordância pelo uso do protocolo por cliente e servidor, inicia-se o processo de *handshake* (Figura 9). Nesse processo, o servidor envia ao cliente informações específicas de identificação, tais como nome do servidor, autoridade de certificação (CA, do inglês *Certification Authority*) e a chave pública do servidor, por meio de um certificado digital. O cliente, então, realiza a verificação e validação das informações (DIERKS;RESCORLA, 2008).

Em seguida, o cliente envia ao servidor suas informações de identificação, também por certificado digital e uma chave de sessão simétrica, que será utilizada para criptografar e descriptografar mensagens transmitidas e recebidas (DIERKS;RESCORLA, 2008).

Após a verificação das informações do cliente pelo servidor e geração da mesma chave de sessão simétrica, o processo de *handshake* é finalizado, estabelecendo-se uma conexão segura (DIERKS;RESCORLA, 2008).

Figura 9 – *Handshake* realizado entre cliente e servidor.



Fonte: CryptoID (2014).

3 ARQUITETURA IOT

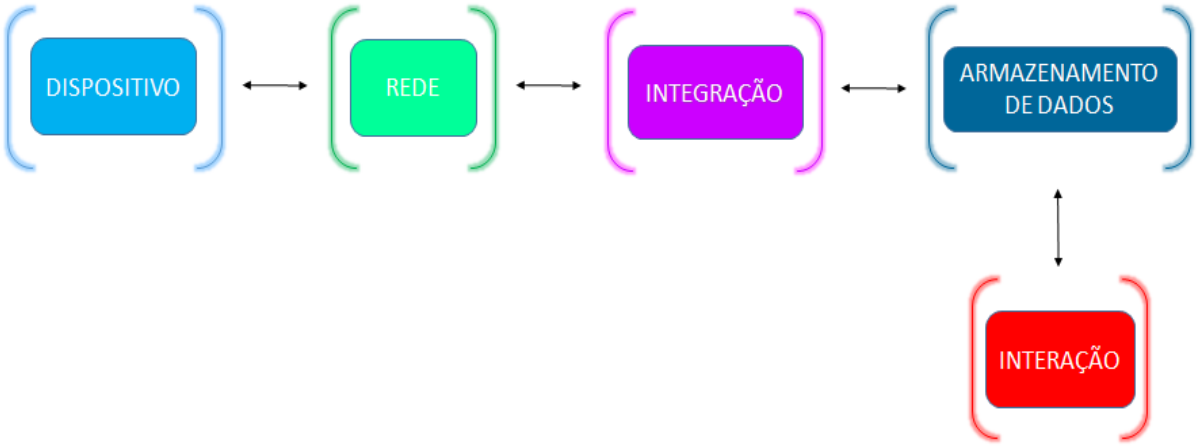
O presente capítulo apresenta a arquitetura IoT proposta neste trabalho, bem como os componentes necessários para sua construção, além da metodologia empregada.

3.1 Proposta do trabalho

Esse trabalho apresenta o desenvolvimento de uma arquitetura IoT de baixo custo, construída a partir de tecnologias *open source*, flexível, escalável e segura, que servirá de referência para a implementação do *Smart Campus* na UNESP Sorocaba.

A arquitetura IoT proposta (Figura 10) é formada por 5 componentes: dispositivo, rede, integração, armazenamento de dados e interação.

Figura 10 – Arquitetura IoT proposta.



Fonte: Autoria própria.

O dispositivo é responsável por coletar dados e enviá-los à integração por meio da rede. Na integração, os dados recebidos são adequados para a escrita no banco de dados. Os dados são, então, escritos e armazenados no banco e acessados pela interação. Na interação, ocorre o monitoramento dos dados por meio de gráficos.

Neste trabalho, foram utilizados como dispositivos os módulos NodeMCU ESP32 DEVKIT V1 (Wi-Fi), em que foi desenvolvido *firmware* para conectividade MQTT, e ESP32 LoRa V2, em que foi desenvolvido *firmware* para conectividade LoRaWAN. Dessa forma, as redes utilizadas para a transmissão e recepção de dados foram: MQTT e LoRaWAN.

A integração MQTT foi realizada por meio do Mosquitto *Broker* e pelo Node-RED. O Mosquitto *Broker* é responsável por receber os dados publicados pela ESP32 Wi-Fi (*Publisher*) em tópico específico e enviar para os respectivos *Subscribers*, isto é, quem está inscrito no mesmo tópico. Neste caso, o Node-RED é um *Subscriber*. Dessa forma, os dados são enviados ao Node-RED e, então, são apropriados para a escrita no banco de dados.

Já a integração LoRaWAN, foi realizada por meio do TTN (*The Things Network*) e pelo Node-RED. No TTN, os dados enviados pela ESP32 LoRa são recebidos e armazenados na nuvem temporariamente. Por meio de um *node* de integração MQTT disponível no Node-RED, é possível acessar os dados do TTN. Assim, os dados são enviados ao Node-RED e adequados para a escrita no banco de dados.

Em ambas integrações, foi utilizado o banco de dados local InfluxDB para a escrita e armazenamento dos dados.

Por fim, a interação ocorre no Grafana, em que os dados podem ser visualizados em uma visão geral no *dashboard* principal e em detalhes nos *dashboards* das aplicações.

A segurança da arquitetura foi desenvolvida por meio de configurações usuário/senha e protocolo TLS no Mosquitto para conexões MQTT. Nas conexões LoRaWAN, é realizada criptografia dos dados utilizando o algoritmo *Advanced Encryption Standard* (AES). Este processo ocorre automaticamente ao enviar dados pelo protocolo LoRaWAN.

Apesar da definição da arquitetura IoT para a realização deste trabalho, os componentes são flexíveis, possibilitando diversas formas da arquitetura ser apresentada, desde que as padronizações sejam seguidas.

3.2 Componentes da Arquitetura

A seguir são apresentados os componentes necessários para o desenvolvimento da arquitetura IoT proposta neste trabalho.

3.2.1 Dispositivos

Criado pela Espressif Systems, o ESP32 é um sistema em um chip (SOC) de baixo custo e baixo consumo de energia, ideal para aplicações IoT. Possui recursos como Wi-Fi e *Bluetooth*, um microprocessador Tensilica Xtensa LX6 *dual-core* ou *single-core*, antena integrada RF tipo *balun*, amplificador de potência, receptor de baixo ruído amplificado, filtros e módulos de gerenciamento de energia (ESP32NET, 2017).

O ESP32 DEVKIT V1 (Figura 11) utilizado possui as seguintes especificações:

- *Dual Core*
- Wi-Fi de 2.4 GHz a 150 Mbits/s
- *Bluetooth* BLE (*Bluetooth Low Energy*)
- Arquitetura de 32 bits
- Frequência de *clock* máxima de 240 MHz
- RAM de 512 kB
- 30 pinos
- Periféricos:
  ADC (*Analog to Digital Converter*), DAC (*Digital to Analog Converter*), UART (*Universal Asynchronous Receiver/Transmitter*), PWM (*Pulse Width Modulation*)

Por sua vez, o ESP32 LoRa V2 (Figura 12) possui as seguintes especificações:

- Fabricante: Heltec
- Microcontrolador ESP32
- *Dual Core*
- Wi-Fi de até 150 Mbits/s
- *Bluetooth* BLE (*Bluetooth Low Energy*)
- *Display* OLED
- Conector JST para bateria Li-Ion/Li-Po

Possui integrado o *chip* de rádio frequência LoRa SX1276, que utiliza a frequência de 915 MHz.

Figura 11 – ESP32 DEVKIT V1.

Fonte: Saravati (2022).

Figura 12 – ESP32 LoRa V2.

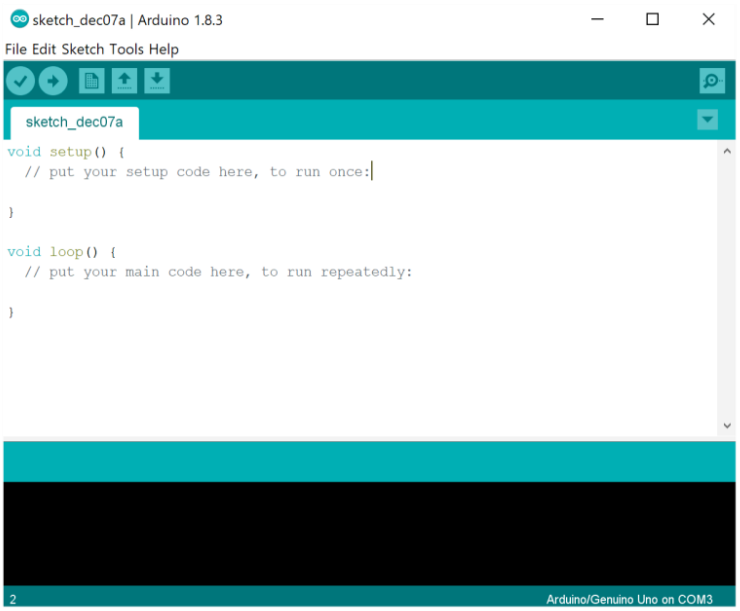
Fonte: Usinainfo (2022).

3.2.1.1 Firmware

Os *firmwares* para conectividade MQTT e LoraWAN foram desenvolvidos no Arduino IDE (Figura 13).

Através do Arduino IDE é possível controlar, por meio de código, ações a serem realizadas pela placa em que está configurado. Por ser um *software* simples e de acesso aberto, tem sido utilizado em diversos projetos relacionados a IoT (ARDUINO, 2018).

Figura 13 – Arduino IDE inicializado.



Fonte: Microsoft (2022).

3.2.2 Rede

Como mencionado nos itens 2.2 e 2.3, MQTT é um protocolo de comunicação que utiliza o modelo *publish/subscribe* para transmissão e recepção de mensagens. Já o LoRaWAN é um protocolo de código aberto que define a arquitetura de rede e os parâmetros de comunicação a serem utilizados por tecnologia de rádio frequência, denominada LoRa.

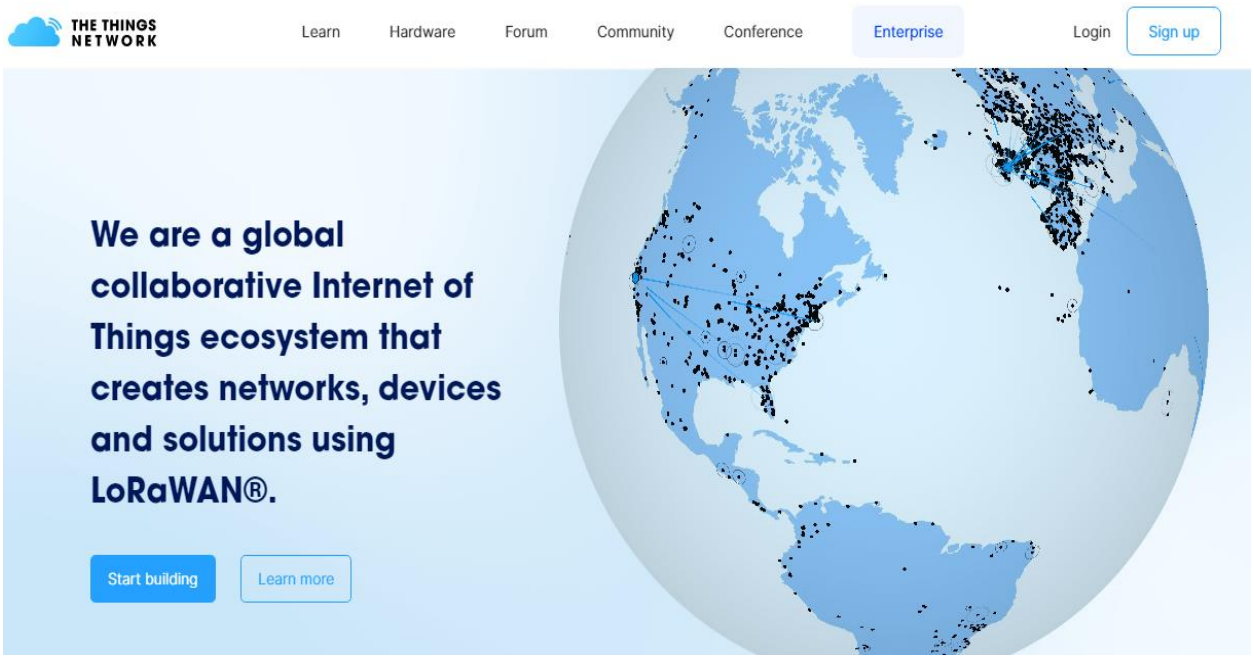
3.2.3 Integração

3.2.3.1 TTN (*The Things Network*)

TTN (*The Things Network*) é uma infraestrutura descentralizada de código aberto voltada para aplicações IoT utilizando LoRaWAN (Figura 14) Por meio do TTN, é possível realizar testes de novas aplicações, integrações entre dispositivos e aprender sobre o protocolo em si (SELIMOVIĆ, 2021).

O TTN disponibiliza gratuitamente o *The Things Stack Community Edition*, um servidor de rede LoRaWAN de nível empresarial, em que é possível construir e gerenciar redes LoRaWAN na nuvem (OLAYINKA, 2022).

Figura 14 – Site oficial do TTN.



Fonte: thethingsnetwork.org (2022).

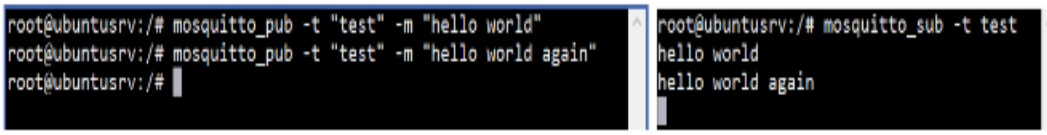
3.2.3.2 Mosquitto

Mosquitto é um *software* livre que implementa o protocolo MQTT nas versões 5.0, 3.1.1 e 3.1. Por fornecer um método leve de execução de mensagens, é adequado para aplicações IoT (MOSQUITTO, 2022).

Por meio do Mosquitto é possível implementar clientes MQTT: *Publisher*s (publicadores de mensagens) e *Subscribers* (receptores de mensagens); e o *Broker* MQTT, cuja função é receber as mensagens dos *Publishers* e direcioná-las aos *Subscribers* de acordo com os tópicos em que foram inscritos.

O Mosquitto fornece, ainda, uma biblioteca em C para implementar clientes MQTT e os comandos de linha “mosquitto_pub”, em que o cliente MQTT publica mensagens, e “mosquitto_sub”, em que o cliente MQTT recebe as mensagens (Figura 15) (MOSQUITTO, 2022).

Figura 15 – Exemplo de publicação e recepção de mensagens com Mosquitto.



Fonte: Spylinuxtips (2020).

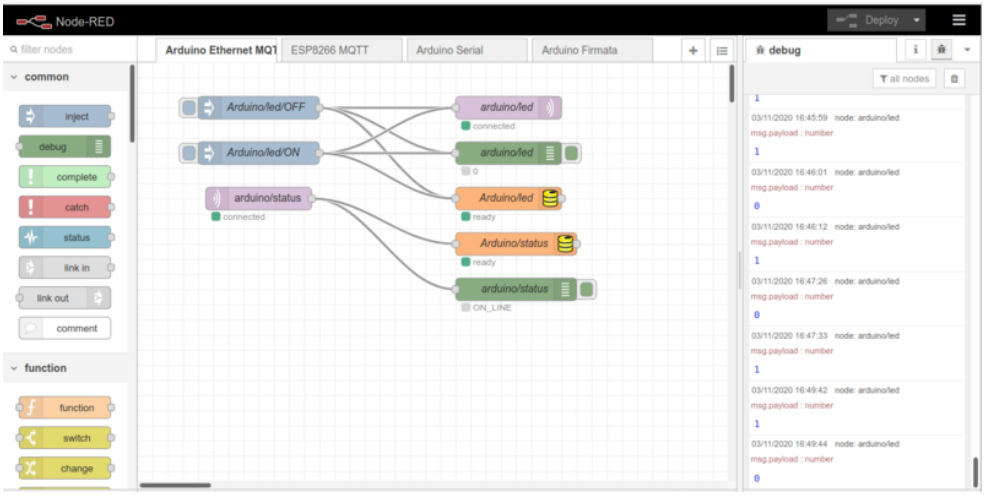
3.2.3.3 Node-RED

Desenvolvida pela IBM, Node-RED é uma ferramenta de programação baseada em fluxo utilizada em aplicações de IoT para conectar dispositivos, APIs (*Application Programming Interface*) e serviços *online* (OPENJSFOUNDATION, 2022).

O Node-RED fornece um editor de fluxo baseado em navegador *web* (Figura 16), em que é possível criar funções *JavaScript*. Seu tempo de execução é construído em Node.js e os fluxos são armazenados em JSON, possibilitando fácil compartilhamento entre usuários.

Possui diversas integrações por meio de seus nós, tais como protocolo MQTT e o banco de dados InfluxDB, utilizados neste trabalho. Além disso, possibilita configurações de segurança, como o protocolo TLS.

Figura 16 – Exemplo de aplicação no Node-RED.



Fonte: IFPR (2022).

3.2.4 Armazenamento de dados

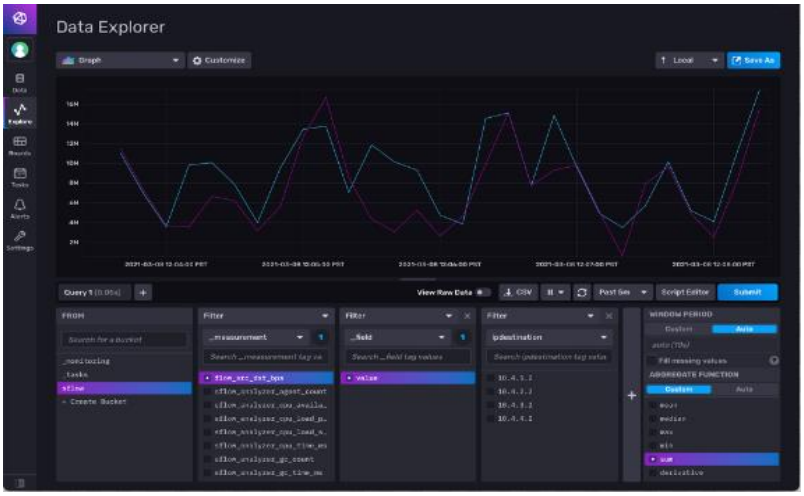
3.2.4.1 InfluxDB

Desenvolvido pela InfluxData, o InfluxDB é um banco de dados de série temporal de código aberto, que permite armazenar, analisar e monitorar dados em tempo real (Figura 17). Também é possível criar alertas em casos de anomalia (INFLUXDATA, 2021).

Possui diversas integrações, dentre as quais podem-se destacar: Node-RED, Grafana, Google Data Studio e Google BigTable.

A linguagem de programação utilizada para consultar e analisar dados é chamada Flux.

Figura 17 – Monitoramento de dados em tempo real no InfluxDB.



Fonte: Sflow (2021).

3.2.5 Interação

3.2.5.1 Grafana

Grafana é uma aplicação *web* que permite analisar, monitorar e visualizar dados por meio de gráficos. É compatível com diversos bancos de dados e sistemas operacionais. Além disso, é possível configurar alertas para que os usuários recebam notificações relevantes relacionados aos dados supervisionados (GRAFANA, 2021).

Por meio dessa plataforma, o usuário é capaz de criar, além de gráficos, *dashboards* dinâmicos e realizar um monitoramento dos dados em tempo real como ilustrado na Figura 18 (GRAFANA, 2021).

Justamente por esse monitoramento de dados em tempo real, tem sido bastante utilizado em aplicações de Internet das Coisas.

Figura 18 – Exemplo de aplicação no Grafana.

Fonte: WFG Solutions (2019).

3.3 Implementação da Arquitetura

Inicialmente o projeto foi desenvolvido em Raspberry Pi 4, porém, devido a maior capacidade de memória e processamento, foi realizada a migração para um Mini PC Intel NUC localizado na UNESP Sorocaba (Figura 19). No Mini PC, os *softwares* Mosquitto, Node-RED e InfluxDB rodam localmente. Já o Grafana pode ser acessado remotamente via protocolo HTTP.

A seguir são apresentadas especificações do Mini PC:

- Sistema Operacional: Windows 11 (64 bits) e Windows 10 (64 bits)
- Processador: Intel Core i5
- Armazenamento interno: 1TB
- Tamanho máximo de memória: 64 GB
- *Quadcore*
- Conexão sem fio: Intel® Wi-Fi 6 AX201
- *Bluetooth* integrado

Figura 19 – Fotografia do Mini PC Intel NUC.

Fonte: Acervo pessoal.

4 DESENVOLVIMENTO DA ARQUITETURA

O desenvolvimento da arquitetura é dividido em duas partes: *backend* e *frontend*. No *backend*, são desenvolvidos os *firmwares*, as integrações das redes, a escrita e o armazenamento dos dados. Já no *frontend*, é desenvolvida a interação dos dados por meio de *dashboards* e gráficos.

4.1 Desenvolvimento do *Backend*

4.1.1 Desenvolvimento do *Firmware*

Inicialmente, foram desenvolvidos dois códigos de envio de dados via MQTT no Arduino IDE, considerando aspectos de segurança. No primeiro código, utilizou-se a segurança por usuário e senha. Já no segundo código, além de usuário e senha, foi configurado o protocolo TLS.

Para o código com segurança de usuário e senha, utilizou-se a porta 1883 do MQTT, pois esta é a porta *default* do MQTT utilizada para conexões simples, em que há menor nível de segurança. Foram configurados os valores de Wi-Fi (*ssid* e *password*), endereço do *Broker* MQTT (*mqttServer*) e definidos os valores de usuário e senha para realizar a conexão MQTT (Figura 20).

Figura 20 – Definição da porta, usuário e senha do MQTT.

```
const char* ssid = "seuSSID";
const char* password =  "suaSENHA";
const char* mqttServer = "IP.DO.BROKER.MQTT"; // IP DO MINI PC
const int mqttPort = 1883;
const char* mqttUser = "usuarioMQTT";
const char* mqttPassword = "senhaMQTT";
```

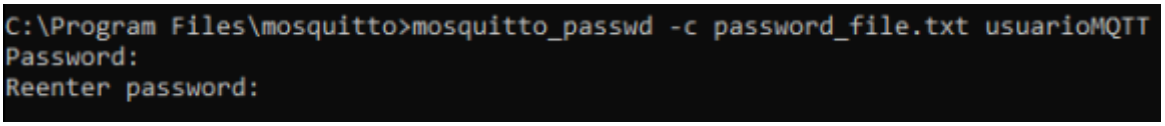
Fonte: Autoria própria.

Com os valores de usuário e senha definidos, foi realizada a configuração no Mosquitto. Para criar um usuário e senha, basta acessar a pasta “mosquitto” pelo *prompt* de comando (rodando como administrador) e digitar o comando:

- mosquitto_passwd –c password_file.txt (nome do usuário)

E, em seguida, inserir e confirmar a senha (Figura 21).

Figura 21 – Comando para criar usuário e senha no Mosquitto.



Fonte: Autoria própria.

Em seguida, foram realizadas as configurações no arquivo do Mosquitto (mosquitto.conf), para habilitar a utilização de usuário e senha. Para tanto, foram realizadas as seguintes mudanças:

- per_listener_settings **true**
- allow_anonymous **false**
- password_file **C:\Program Files\mosquitto\password_file.txt**

Após reinicialização do Mosquitto, já foi possível realizar as conexões MQTT com usuário e senha definidos.

Realizadas as configurações de segurança, foram definidas as variáveis a serem enviadas pela ESP32 Wi-Fi. Foram padronizadas 9 variáveis:

- **ID:** ID do dispositivo;
- **Valor:** valor da medição;
- **Aplicação:** número ou nome da aplicação;
- **Local:** código do local
- **Tipo:** tipo ou fonte do dispositivo
- **Variável:** tipo da variável;
- **Unidade:** unidade de medida utilizada;
- **Rede:** rede utilizada na comunicação;
- **Professor:** nome do professor responsável.

Por meio da função “random”, foi simulada a aquisição de dados de um sensor de

umidade relativa do ar na estação meteorológica da UNESP Sorocaba (Figura 22). Na definição das variáveis, foi utilizada a extensão de arquivo JSON.

Realizada a aquisição de dados e a definição das variáveis, foi realizada a compactação das informações em um pacote de dados por meio da função “serializeJson”. O pacote de dados é enviado ao *Broker* via protocolo MQTT e publicado no seguinte tópico:

- ESP32ClientII/PFC_Matheus/Paciencia

O tópico segue o seguinte padrão:

- Identificador do dispositivo/Aplicação/Nome do Professor

Figura 22 – Aquisição de dados, definição de variáveis e envio do pacote de dados via protocolo MQTT.

```
StaticJsonDocument<300> doc;

float X;

X = (random (0,1000))/10.00;

doc["ID"] = "S1_U";
doc["Valor"] = X;
doc["Aplicacao"] = "PFC_Matheus";
doc["Local"] = "Estacao_Meteorologica";
doc["Tipo"] = "Sensor";
doc["Variavel"] = "Umidade_do_ar";
doc["Unidade"] = "g/m^3";
doc["Rede"] = "MQTT";
doc["Professor"] = "Paciencia";

char JSONmessageDoc[300];
serializeJson(doc,JSONmessageDoc);
Serial.println("Sending message to MQTT topic..");
Serial.println(JSONmessageDoc);

if (client.publish("ESP32ClientII/PFC_Matheus/Paciencia", JSONmessageDoc) == true) {
  Serial.println("Success sending message");
} else {
  Serial.println("Error sending message");
}

client.loop();
Serial.println("-------------");
```

Fonte: Autoria própria.

No segundo código criado no Arduino IDE, além da segurança por usuário e senha, foi acrescentada a configuração do protocolo TLS. Ao utilizar esse protocolo, a porta MQTT utilizada deve ser alterada para 8883 (Figura 23).

Figura 23 – Definição da porta segura do MQTT.

```
const char* ssid = "seuSSID";
const char* password =  "suaSENHA";
const char* mqttServer = "IP.DO.BROKER.MQTT"; // IP DO MINI PC
const int mqttPort = 8883;
const char* mqttUser = "usuarioMQTT";
const char* mqttPassword = "senhaMQTT";
```

Fonte: Autoria própria.

Em seguida, é acrescentado ao código o certificado criptografado (ca.crt) criado previamente por meio do protocolo TLS (Figura 24). Parte do certificado foi ocultada para preservação da segurança da arquitetura do trabalho.

Figura 24 – Certificado criptografado (ca.crt).

```
const char* local_root_ca = \

"-----BEGIN CERTIFICATE-----\n" \
"MIID5zCCAs+gAwIBAgIJANb5gvvtyKe+MA0GCSqGSIb3DQEBCwUAMIGJMQswCQYD\n" \
"VQQGEwJCUjESMBAGA1UECAwJU2FvIFBhdWxvMREwDwYDVQQHDAhTb3JvY2FiYTEO\n" \
"MAwGA1UECgwFVU5FU1AxDTALBgNVBAsMBE1DVFMxEzARBgNVBAMMC1RMU19jbGll\n" \
"bnQxHzAdBgkqhkiG9w0BCQEWEHNtYXJ0QGNhbXB1cy5jb20wHhcNMjIwMjA2MTg0\n" \
"OTI4WhcNMzIwMjA0MTg0OTI4WjCBiTELMAkGA1UEBhMCQlIxEjAQBgNVBAgMCVNh\n" \
.
.
.
.
.
.
.
.
.
"Iu6oDeeENUBhXW+e6odBkuo2TaIonytZQKm12+E8v368qLQvnLn/cHFTGA==\n" \
"-----END CERTIFICATE-----\n";
```

Fonte: Autoria própria.

Utilizando-se a função “random”, foi simulada a aquisição de dados de um sensor de temperatura de um painel solar (Figura 25). Neste caso, a rede é denominada “MQTTS”, pois a conexão é segura por TLS. O pacote de dados é publicado no tópico:

- ESPClientI/PFC_Matheus/Paciencia

Figura 25 – Aquisição de dados, definição de variáveis e envio do pacote de dados via protocolo MQTT seguro por TLS.

```
StaticJsonDocument<300> doc;

float X;
X = (random (0,1000))/10.00;
Serial.println (X);


doc["ID"] = "S1_T";
doc["Valor"] = X;
doc["Aplicacao"] = "PFC_Matheus";
doc["Local"] = "GASI";
doc["Tipo"] = "Sensor";
doc["Variavel"] = "Temperatura";
doc["Unidade"] = "Celsius";
doc["Rede"] = "MQTTS";
doc["Professor"] = "Paciencia";

char JSONmessageDoc[300];
serializeJson(doc,JSONmessageDoc);
Serial.println("Sending message to MQTT topic..");
Serial.println(JSONmessageDoc);

if (client.publish("ESP32ClientI/PFC_Matheus/Paciencia", JSONmessageDoc) == true) {
  Serial.println("Success sending message");
} else {
  Serial.println("Error sending message");
}

client.loop();
Serial.println("-------------");
```

Fonte: Autoria própria.

Para a realização da conexão LoRaWAN, foi disponibilizado um código padrão pelo professor orientador. Este código foi editado a fim de ser simulada uma aquisição de dados de um sensor de potência ativa de um painel solar por meio do protocolo LoRaWAN. Para tanto, foi utilizada a função “random”.

Em seguida, foram definidas as variáveis de acordo com o padrão previamente

determinado de 9 variáveis (ID, Valor, Aplicação, Local, Tipo, Variável, Unidade, Rede e Professor) (Figura 26).

Figura 26 – Simulação da aquisição de dados e definição das variáveis.

```
// Preparação dos dados para envio (payload)

float X;
X = (random (0,1000))/10.00;

doc["ID"] = "S1_P";
doc["Valor"] = X;
doc["Aplicacao"] = "PFC_Matheus";
doc["Local"] = "GASI";
doc["Tipo"] = "Sensor";
doc["Variavel"] = "Potencia_Ativa";
doc["Unidade"] = "Watts";
doc["Rede"] = "LoraWan";
doc["Professor"] = "Paciencia";
```

Fonte: Autoria própria.

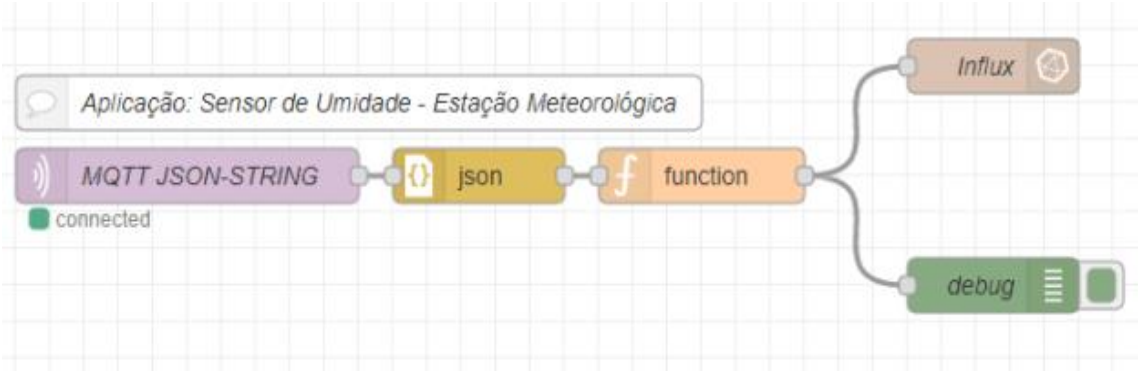
4.1.2 Integração MQTT

O Mosquitto *Broker* instalado no Mini PC, recebe os pacotes de dados e os direciona aos *Subscribers* que estiverem inscritos nos seguintes tópicos no Node-RED:

- ESP32ClientII/PFC_Matheus/Paciencia
- ESPClientI/PFC_Matheus/Paciencia

Para o sensor de umidade relativa do ar foi construído o seguinte *flow* (Figura 27) no Node-RED, responsável por receber as mensagens publicadas no tópico (ESP32ClientII/ PFC_Matheus/Paciencia) e enviá-las ao banco de dados local (InfluxDB):

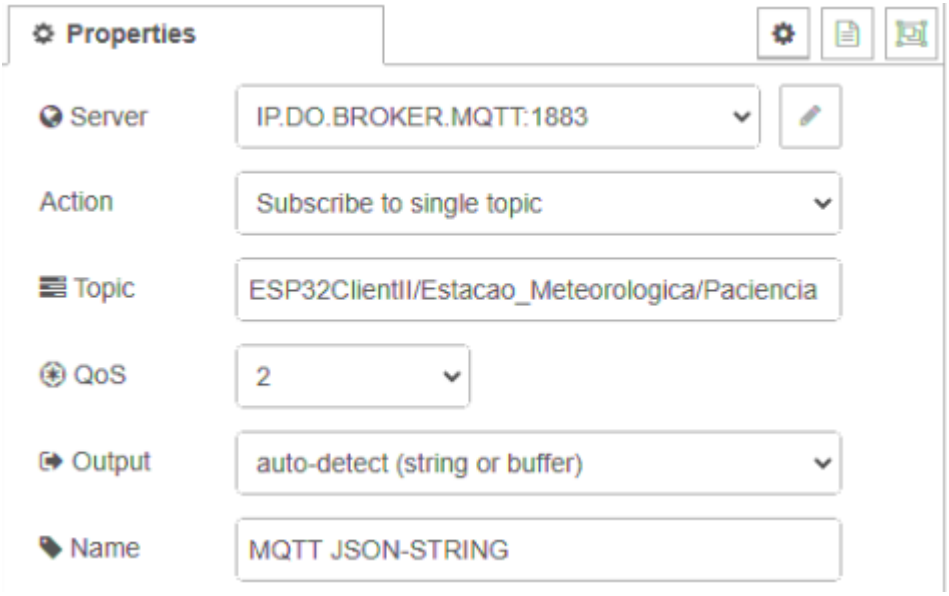
Figura 27 – *Flow* construído no Node-RED.



Fonte: Autoria própria.

Inicialmente, foi configurado o *node* "MQTT JSON-STRING" (*mqtt in*), conforme mostrado nas Figuras 28 a 30.

Figura 28 – Configuração geral do *node* MQTT.



Fonte: Autoria própria.

Na configuração geral (Figura 28), foram editados os seguintes campos:

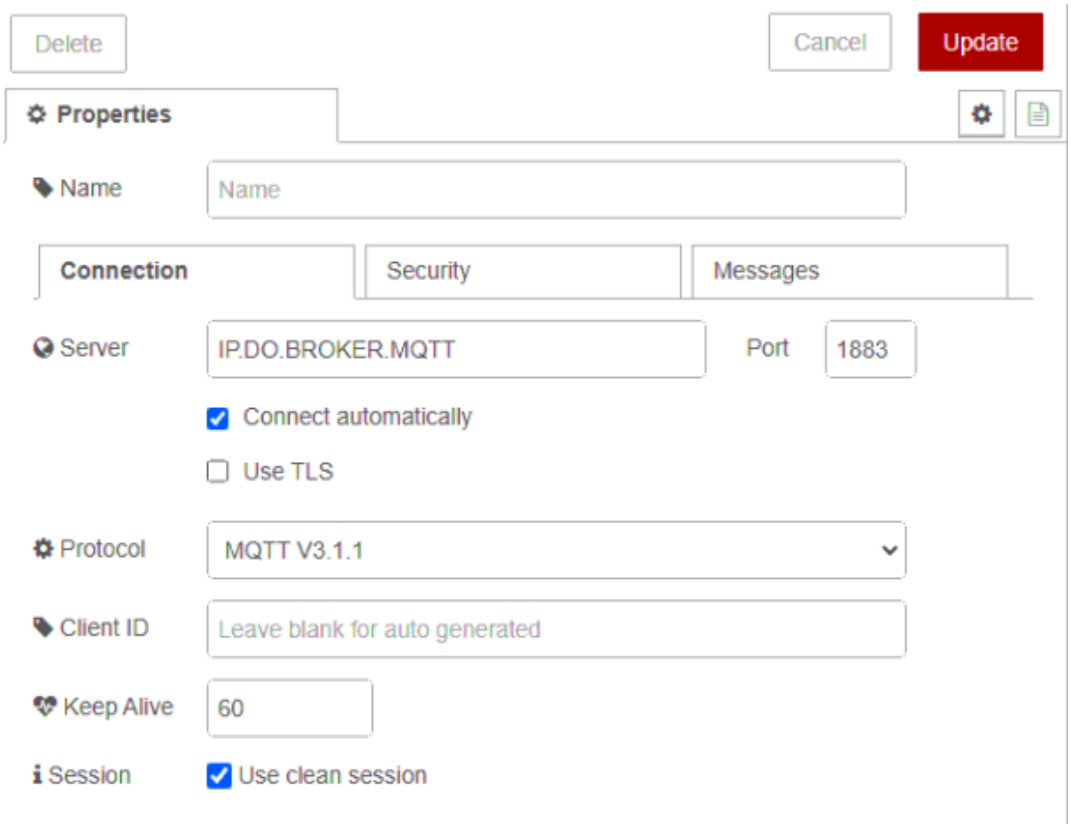
- ***Topic:*** nome do tópico em que o *node* será inscrito;
- ***Name:*** nome do *node.*

Para configuração do *Broker* (*Server*), basta clicar no lápis à direita para abrir a tela de propriedades (Figura 29).

Para a conexão MQTT, foram editados os seguintes campos:

- ***Server:*** IP do Mini PC;
- ***Port:*** porta *default* do MQTT (1883);
- ***Protocol:*** protocolo utilizado - MQTT V3.1.1.

Figura 29 – Configuração do *Broker* do *node* MQTT.



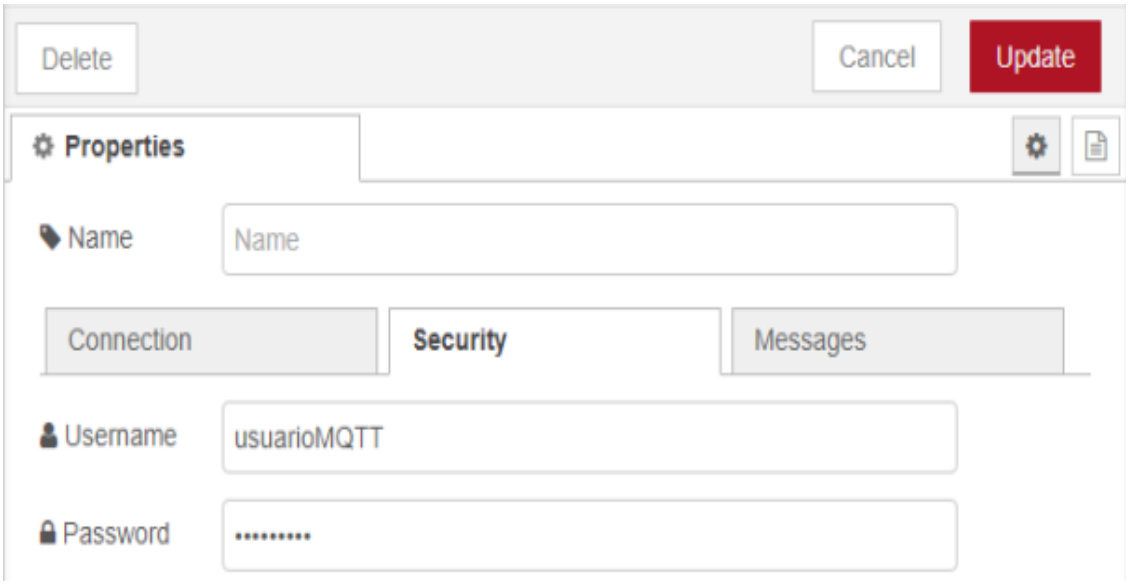
Fonte: Autoria própria.

Foi configurada a segurança por usuário e senha na aba “Security” (Figura 30). Foram editados os seguintes campos:

- ***Username:*** nome do usuário configurado no Mosquitto;
- ***Password:*** senha configurada no Mosquitto.

Para finalizar e atualizar as configurações do *node* MQTT, basta clicar em “Update” e, em seguida, “Done”. O símbolo verde escrito “connected” sinaliza que a conexão foi estabelecida.

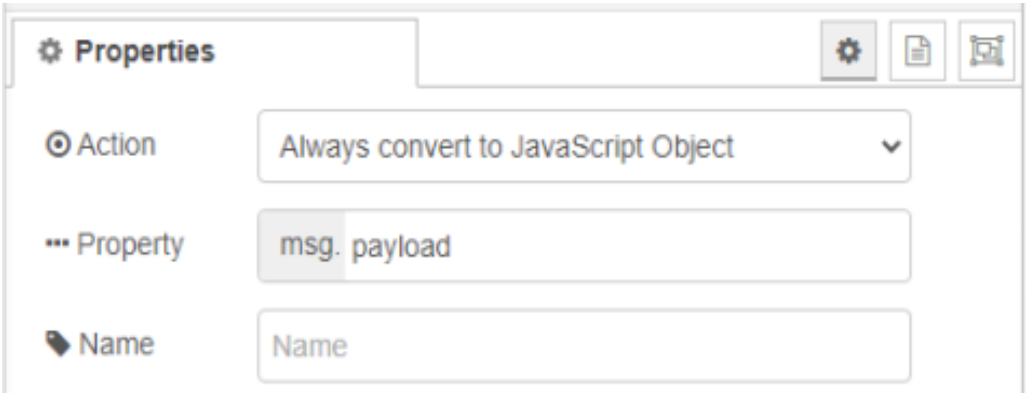
Figura 30 – Configuração de usuário e senha do *node* MQTT.



Fonte: Autoria própria.

O próximo *node* do *flow* é o *node* "JSON". Este *node* é responsável por converter as mensagens recebidas no *node* MQTT do formato *JSON-String* para *JavaScript-Object*. Dessa forma, faz-se possível a manipulação dos valores contidos no pacote de dados por meio do node "FUNCTION", que será detalhado a seguir. Sendo assim, foi configurado no campo "Action", a opção "Always convert to JavaScript Object" (Figura 31).

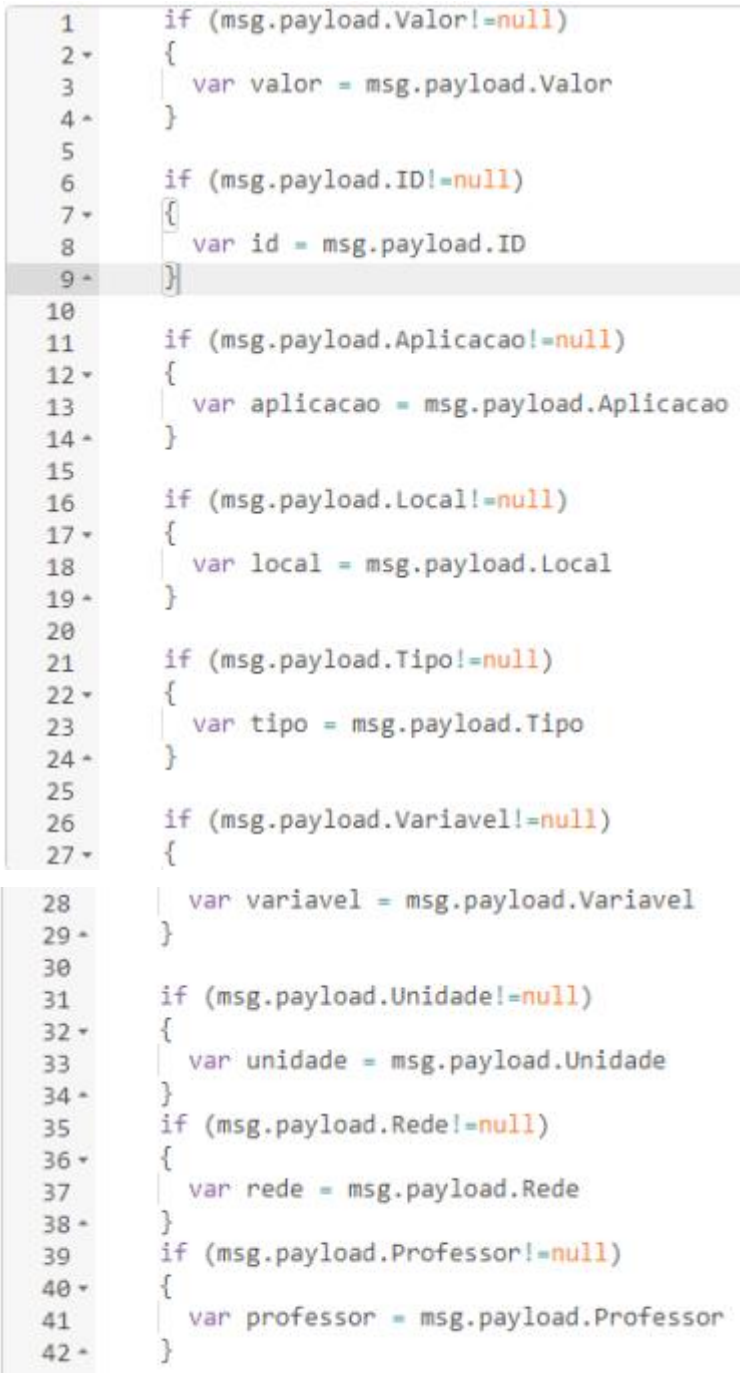
Figura 31 – Configuração do *node* JSON.



Fonte: Autoria própria.

Em seguida, no *node* "FUNCTION", foi construído um código em *JavaScript* capaz de filtrar as 9 variáveis presentes no pacote dados (msg.payload) e alocá-las em variáveis locais correspondentes (Figura 32).

Figura 32 – Código construído no node FUNCTION.



```
if (msg.payload.Valor!=null)
{
  var valor = msg.payload.Valor
}

if (msg.payload.ID!=null)
{
  var id = msg.payload.ID
}

if (msg.payload.Aplicacao!=null)
{
  var aplicacao = msg.payload.Aplicacao
}

if (msg.payload.Local!=null)
{
  var local = msg.payload.Local
}

if (msg.payload.Tipo!=null)
{
  var tipo = msg.payload.Tipo
}

if (msg.payload.Variavel!=null)
{
  var variavel = msg.payload.Variavel
}

if (msg.payload.Unidade!=null)
{
  var unidade = msg.payload.Unidade
}
if (msg.payload.Rede!=null)
{
  var rede = msg.payload.Rede
}
if (msg.payload.Professor!=null)
{
  var professor = msg.payload.Professor
}
```

Fonte: Autoria própria.

Para que haja a escrita correta no InfluxDB, é acrescentado um código em que são feitas distinções entre variáveis de valor, chamadas “Field Value”, e variáveis de *strings*, chamadas “TagValue” (Figura 33).

Figura 33 – Código para escrita dos dados no InfluxDB.

```
45    msg.payload = [{
46
47        Valor:valor
48
49  },
50
51
52  {
53
54        ID: id,
55        Aplicacao: aplicacao,
56        Local: local,
57        Tipo: tipo,
58        Variavel: variavel,
59        Unidade: unidade,
60        Rede: rede,
61        Professor: professor
62
63
64  }];
```

Fonte: Autoria própria.

Como se nota na Figura 33, “Fields Value” ficam na parte superior do código, enquanto “Tags Value” ficam na parte inferior.

Dessa forma, os dados serão escritos em 9 colunas no InfluxDB:

- Valor;
- ID;
- Aplicação;
- Local;
- Tipo;
- Variavel
- Unidade;
- Rede;
- Professor.

Por fim, ao final do *flow*, há os *nodes* “DEBUG” e “INFLUX” (influxdb out). O *node* “DEBUG” é utilizado para visualizar a chegada dos dados vindos da ESP32 (Figura 34). Já o *node* “INFLUX” é utilizado para realizar a escrita dos dados no InfluxDB.

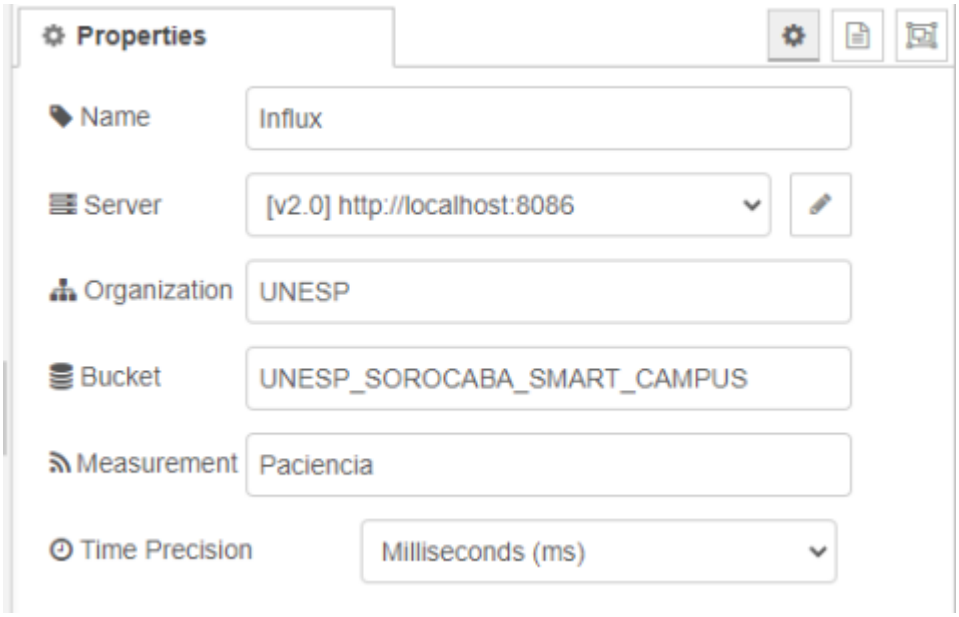
Figura 34 – Monitoramento da chegada de dados da ESP32 no Node-RED.

Fonte: Autoria própria.

O *node* “INFLUX” foi configurado conforme Figuras 35 e 36. Na configuração geral (Figura 35) foram editados os seguintes campos:

- ***Name:*** nome do *node;*
- ***Organization:*** organização responsável (deve ser a mesma escolhida no InfluxDB);
- ***Bucket:*** nome da base de dados criada no InfluxDB;
- ***Measurement:*** nome do banco de dados que será criado no InfluxDB pelo Node-RED (corresponde ao nome do Professor).

Figura 35 – Configuração geral do *node* INFLUX.



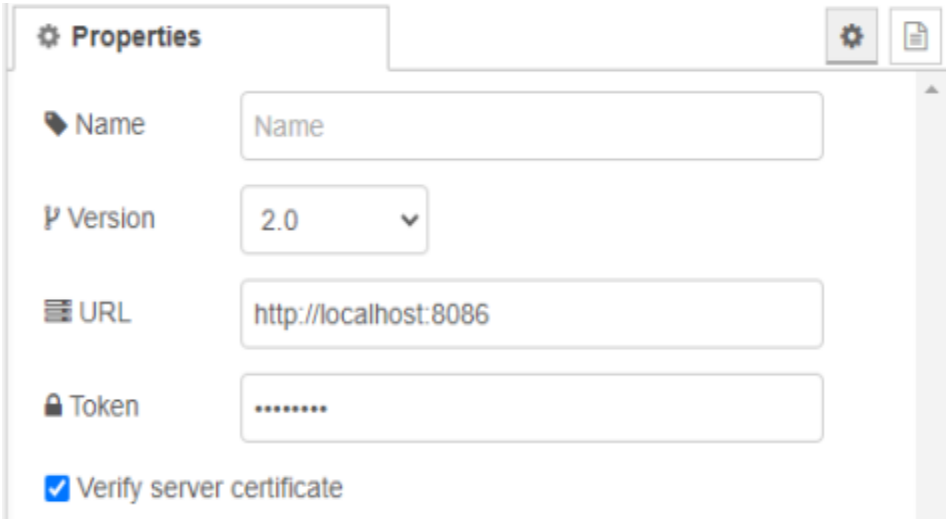
Fonte: Autoria própria.

Para a configuração do Servidor (*Server*) (Figura 36), basta clicar no lápis à direita. Foram, então, editados os seguintes campos:

- *Version:* versão utilizada do InfluxDB – 2.0;
- *Token:* a chave *API Token* gerada através do InfluxDB.

Importante ressaltar que por *default* o InfluxDB roda localmente através da porta 8086.

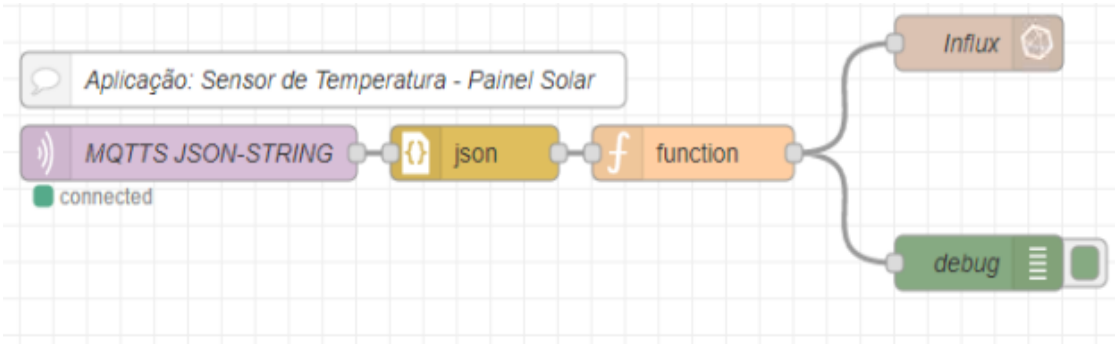
Figura 36 – Configuração do servidor do *node* INFLUX.



Fonte: Autoria própria.

Para o sensor de temperatura, foi criado o *flow* apresentado na Figura 37. Realizou-se a configuração do *node* “MQTTS JSON-STRING” (mqtt in), conforme Figura 38.

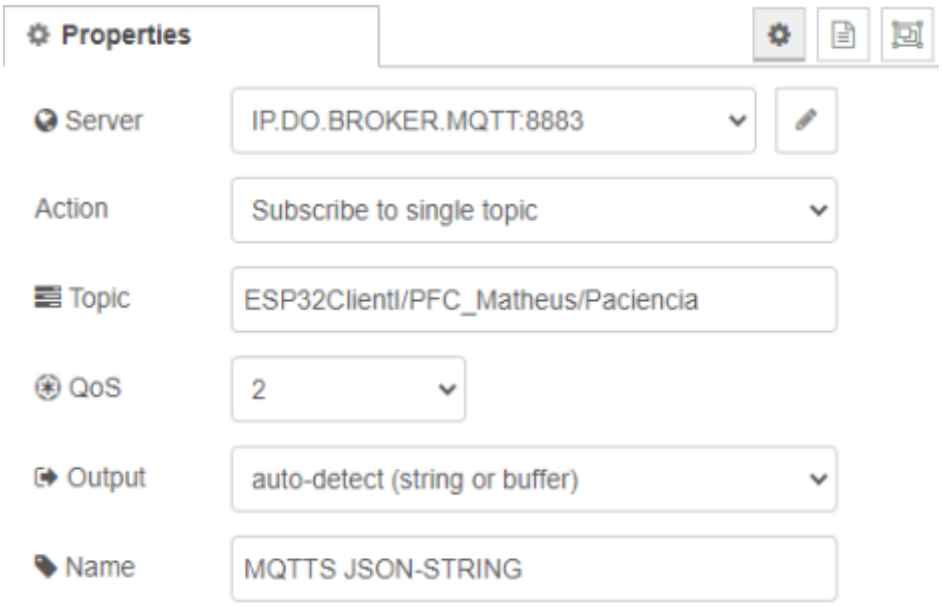
Figura 37 – *Flow* do sensor de temperatura construído no Node-RED.



Fonte: Autoria própria.

No campo “Topic”, foi acrescentado o nome do tópico em que o *node* seria inscrito. Para editar o campo “Server”, é necessário clicar no lápis à direita.

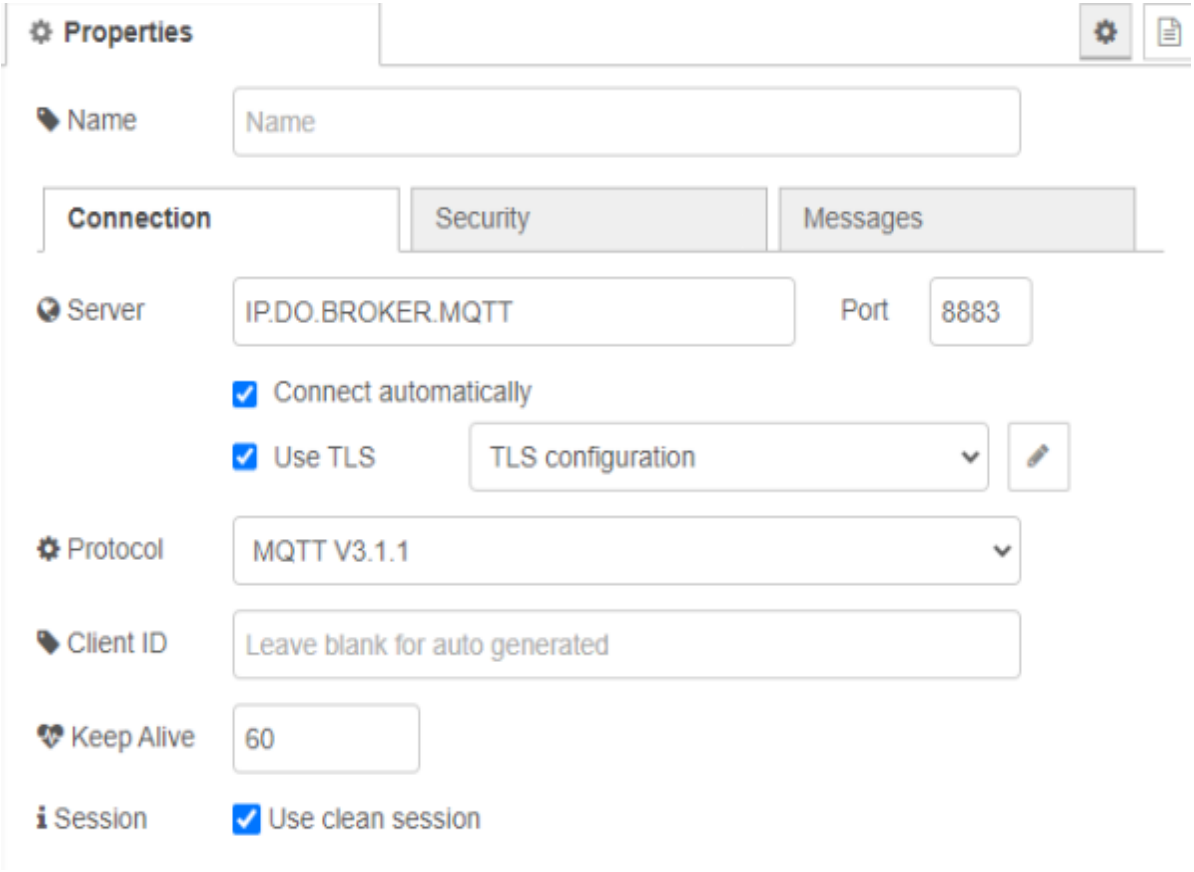
Figura 38 – Configuração do *node* “MQTTS JSON-STRING”.



Fonte: Autoria própria.

Na configuração do “Server” (Figura 39), deve-se colocar o IP do Mini PC e a porta 8883. Além disso, deve-se clicar na opção “Use TLS” para habilitar o uso do protocolo. Clicando no lápis novamente, são realizadas as configurações específicas do TLS (Figura 40).

Figura 39 – Configuração do campo “Server”.



Fonte: Autoria própria.

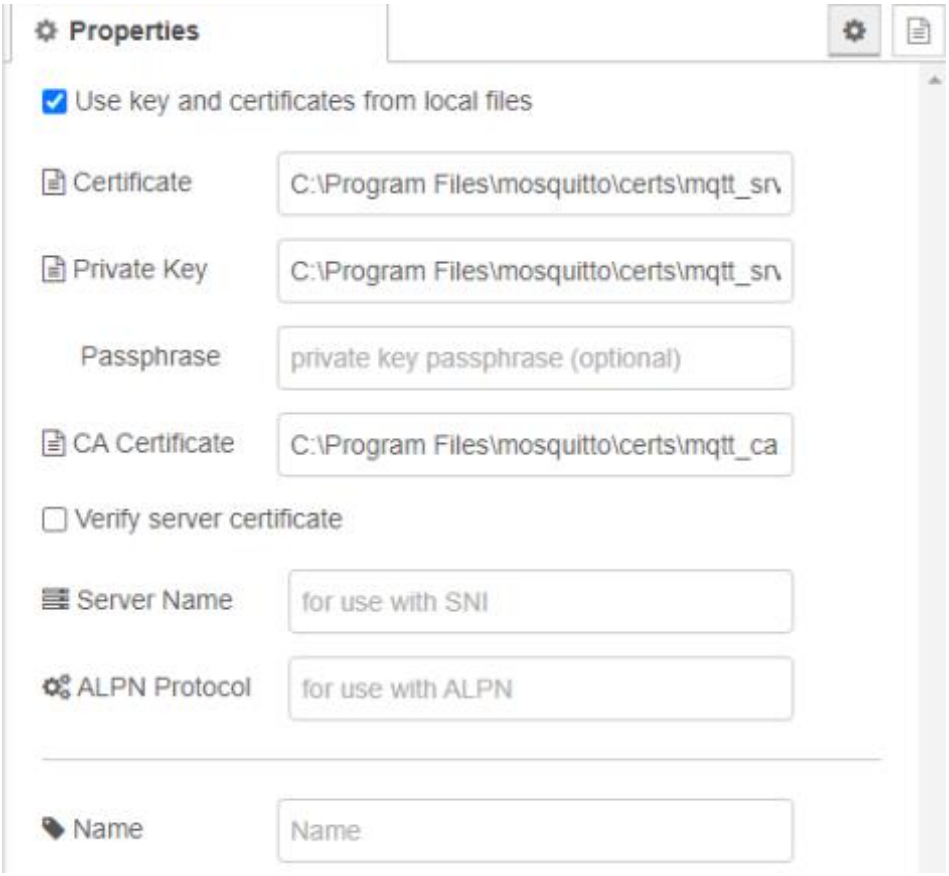
Na configuração do TLS (Figura 40), são utilizados os certificados e a chave privada criados a partir do protocolo e armazenados em arquivos locais. Os campos são preenchidos de acordo com o caminho em que tais arquivos se encontram no Mini PC:

- ***Certificate:*** caminho do certificado do servidor (mqtt_srv.crt);
- ***Private key:*** caminho da chave privada do servidor (mqtt_srv.key);
- ***CA Certificate:*** caminho do certificado CA (mqtt_ca.crt).

Os *nodes* posteriores do *flow* do sensor de temperatura possuem as mesmas configurações do *flow* do sensor de umidade relativa do ar descritos anteriormente.

Os dados são enviados à base de dados criada no InfluxDB e armazenados localmente.

Figura 40 – Configuração TLS.



Fonte: Autoria própria.

4.1.3 Integração LoRaWAN

Inicialmente, a ESP32 LoRa envia o pacote de dados a um *gateway* LoRa mais próximo via protocolo LoRaWAN. O *gateway*, por sua vez, envia o pacote de dados recebido ao servidor de rede (TTN). Os dados são, então, temporariamente, armazenados na nuvem.

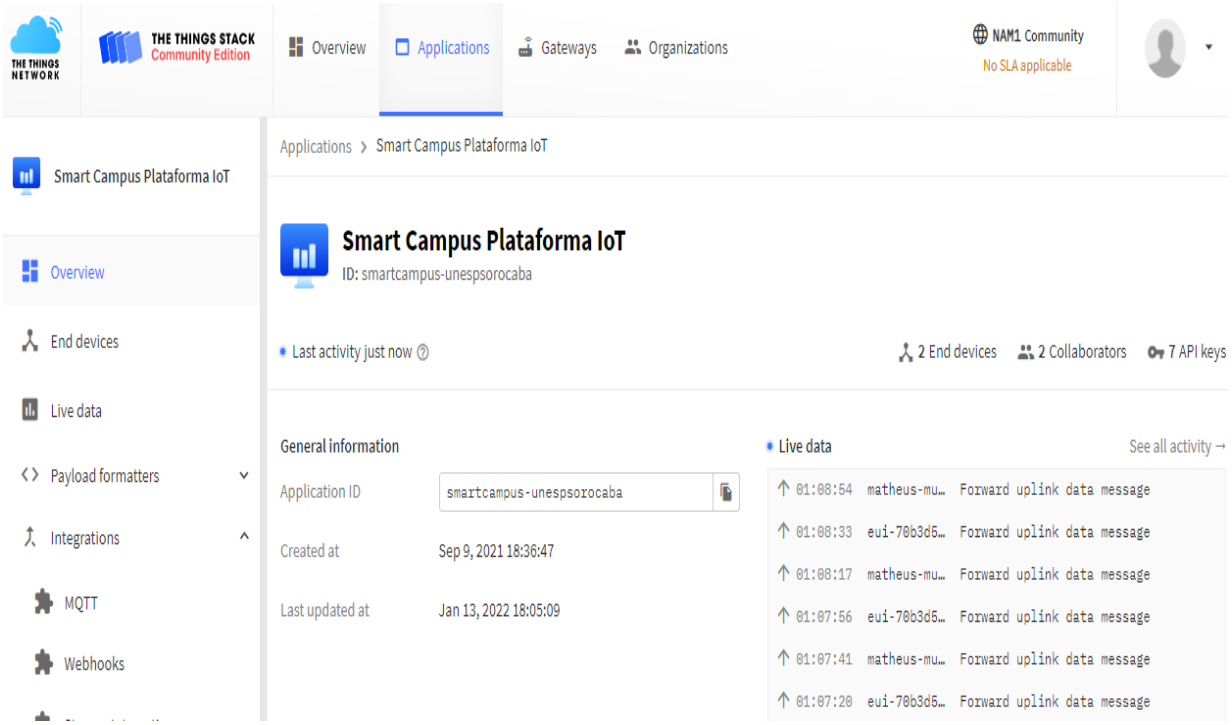
Por meio do TTN, é possível realizar a integração via MQTT com o Node-RED. Para tanto, basta acessar a aba “Applications” e, em seguida, em “Integrations” à esquerda da tela e, então, no ícone MQTT (Figura 41).

Na configuração MQTT (Figura 42), são mostrados os campos que serão utilizados no Node-RED:

- ***Public address:*** servidor MQTT utilizando porta *default* 1883;
- ***Username:*** usuário TTN;
- ***Password:*** chave API gerada (deve-se clicar em “Generate new API key”).

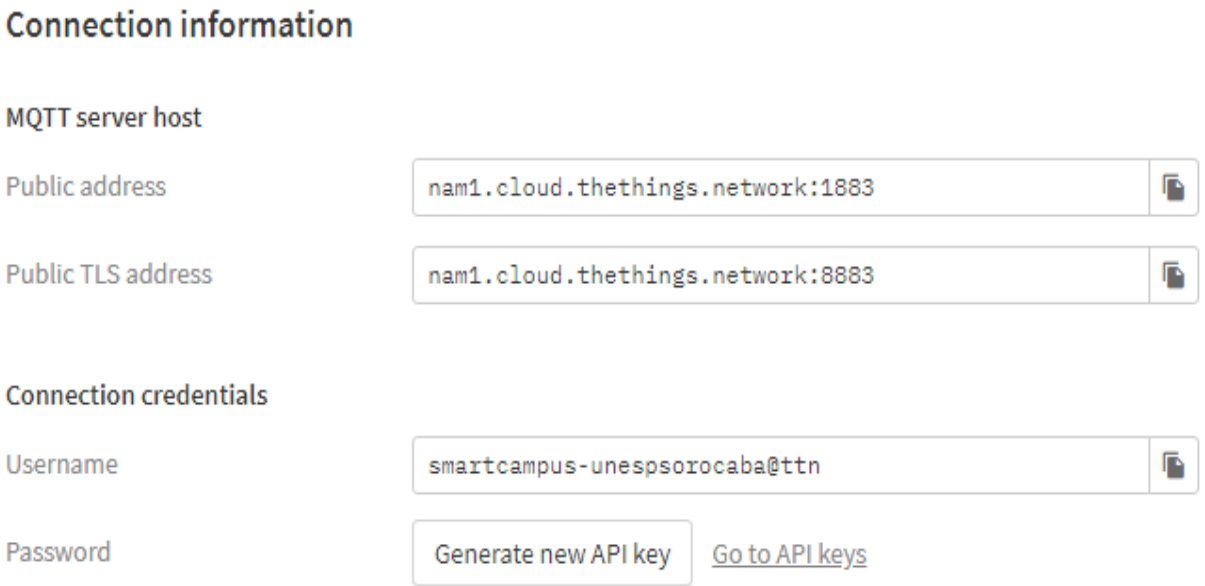
Para realizar a integração com o TTN e, em seguida, realizar a escrita dos dados no InfluxDB, foi construído o *flow* mostrado na Figura 43 no Node-RED.

Figura 41 – Integração do TTN com o Node-RED via MQTT.



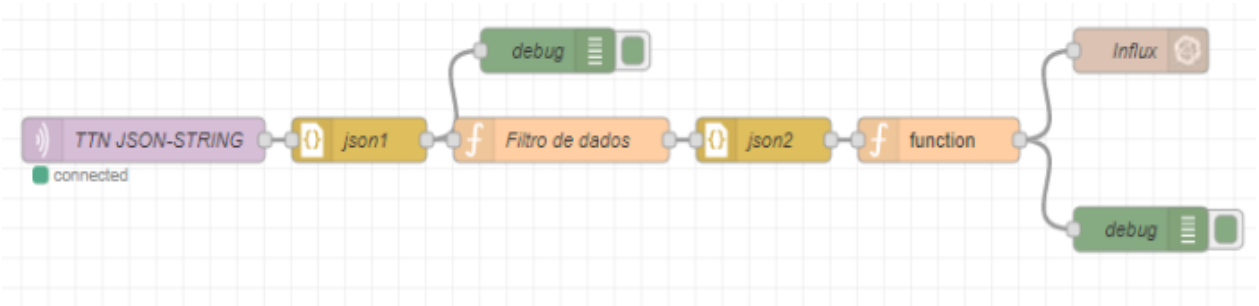
Fonte: Autoria própria.

Figura 42 – Configuração da conexão MQTT no TTN.



Fonte: Autoria própria.

Figura 43 – *Flow* de integração com o TTN construído no Node-RED.

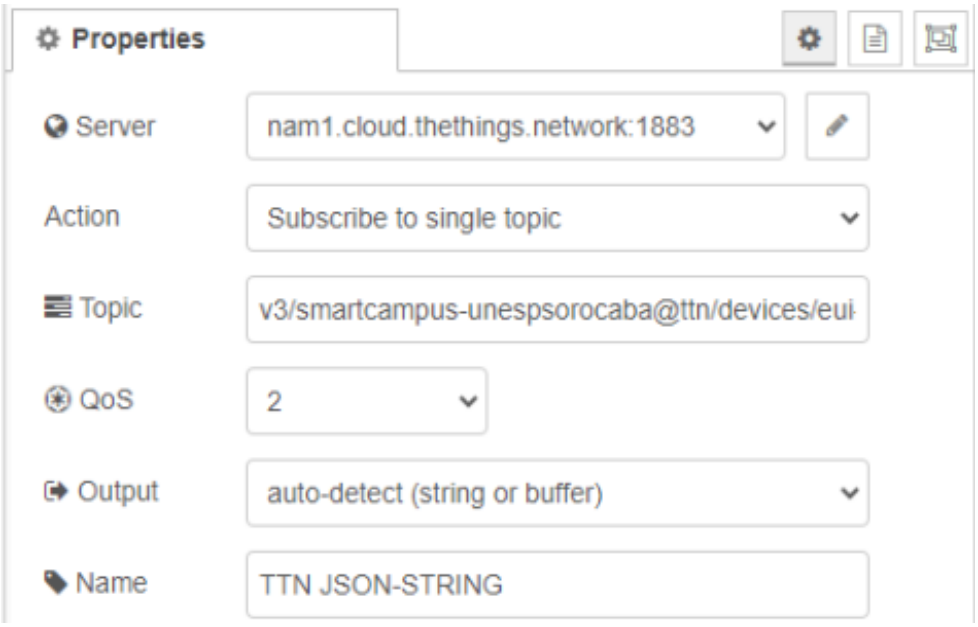


Fonte: Autoria própria.

Inicialmente, foi configurado o *node* “TTN JSON STRING” (*mqtt in*) (Figura 44). O *node* foi inscrito no seguinte tópico:

- v3/smartcampus-unespsorocaba@ttn/devices/eui-70b3d57ed00474d9/up

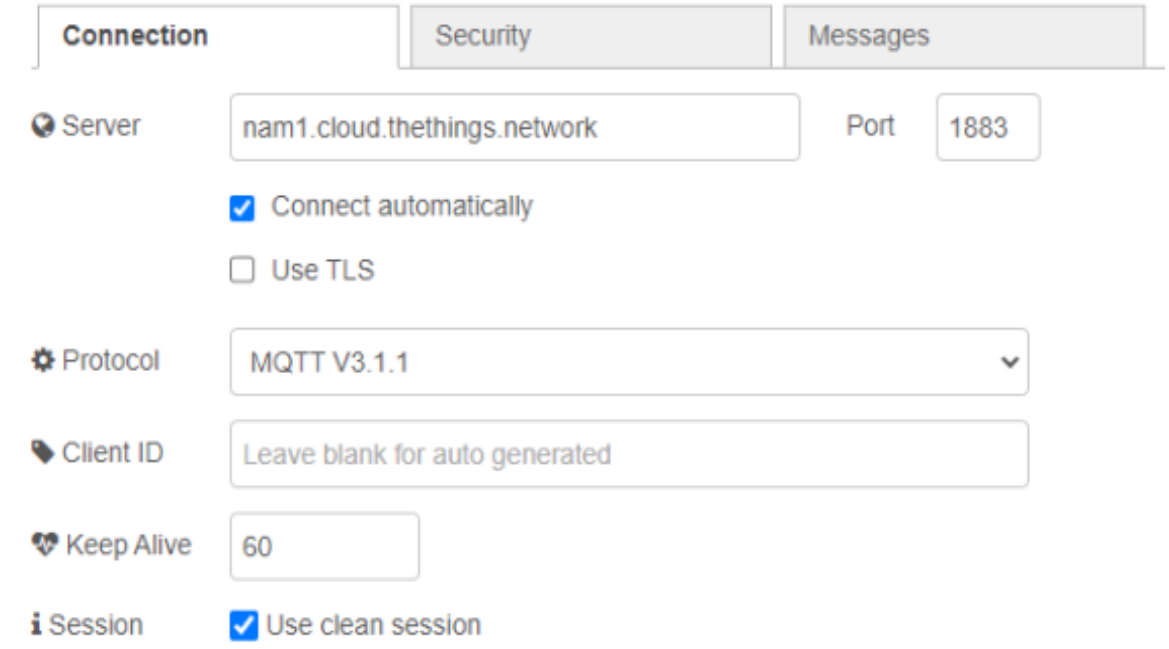
Figura 44 – Configuração do *node* “TTN JSON STRING”.



Fonte: Autoria própria.

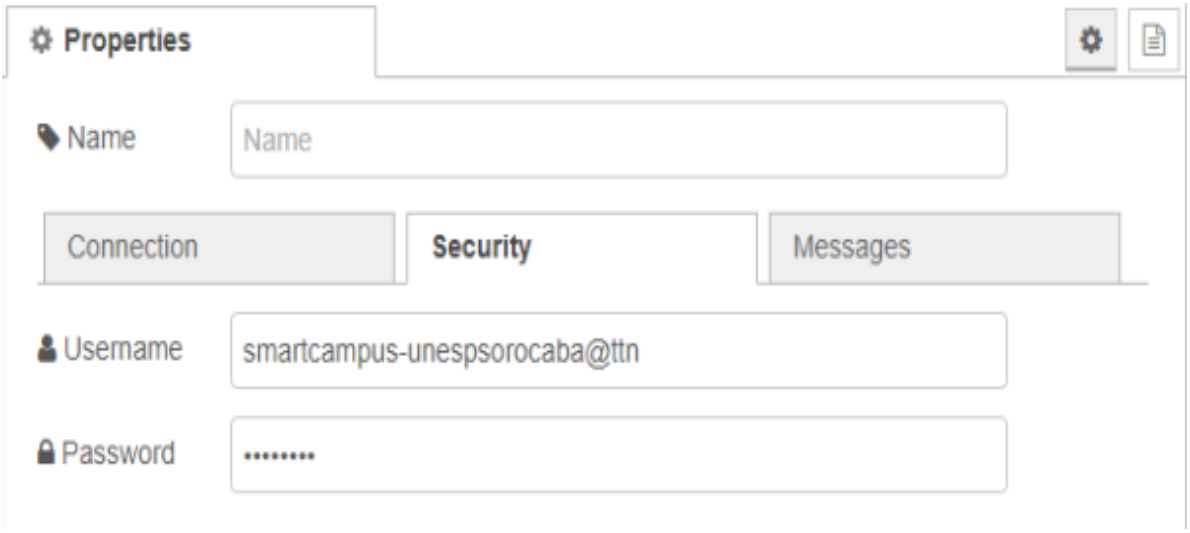
Em seguida, foi configurado o “Server”, clicando-se no lápis à direita, conforme mostrado nas Figuras 45 e 46. Os dados do servidor MQTT são inseridos em “Server” e “Port” na aba “Connection”. Na aba “Security”, deve-se colocar o usuário TTN no campo “Username” e a chave API gerada no campo “Password”.

Figura 45 – Configuração da conexão MQTT.



Fonte: Autoria própria.

Figura 46 – Configuração de autenticação.



Fonte: Autoria própria.

O próximo *node* do *flow* é o "json1". Este *node* é responsável pela conversão da mensagem recebida do tipo *String* para *JavaScript Object*. Em seguida, o *node* "Filtro de dados" é uma função que filtra os dados recebidos do TTN, a fim de se obter apenas os dados principais, relacionados às variáveis padronizadas (ID, Valor, Aplicação, Local, Tipo, Variável, Unidade, Rede e Professor). O código utilizado é mostrado na Figura 47.

Figura 47– Código responsável por filtrar mensagens recebidas do TTN.

```
1  var data = msg.payload.uplink_message.decoded_payload.data
2
3      msg.payload = data
4
5  return msg;
```

Fonte: Autoria própria.

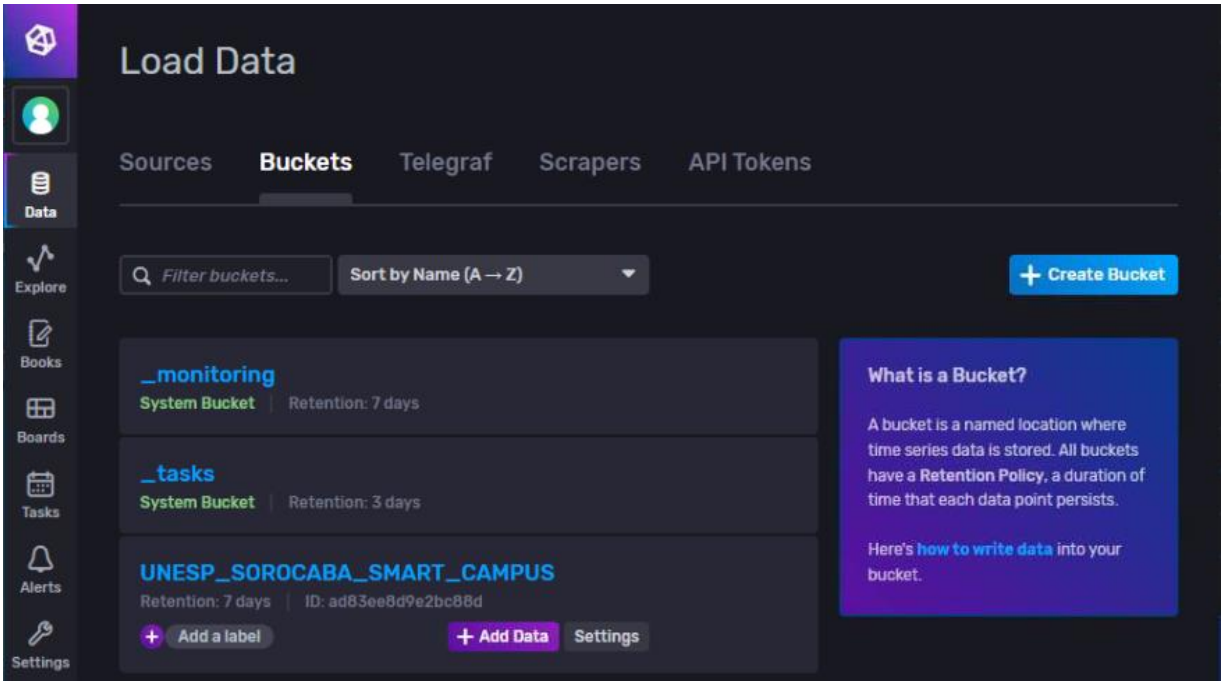
Após a filtragem dos dados principais, é necessário realizar novamente a conversão da mensagem do tipo *String* para *JavaScript Object*. Tal conversão é realizada pelo node "json2".

Por fim, os *nodes* "function" e "Influx" possuem a mesma configuração apresentada anteriormente na aplicação do sensor de umidade relativa do ar.

4.1.4 Escrita e Armazenamento dos Dados

No InfluxDB 2.0, a base de dados é denominada *Bucket*. Para criar a base de dados, basta clicar em "Data" no lado esquerdo da tela do InfluxDB, escolher a aba "Buckets" e, então, no botão azul "Create Bucket" situado à direita, conforme mostrado na Figura 48.

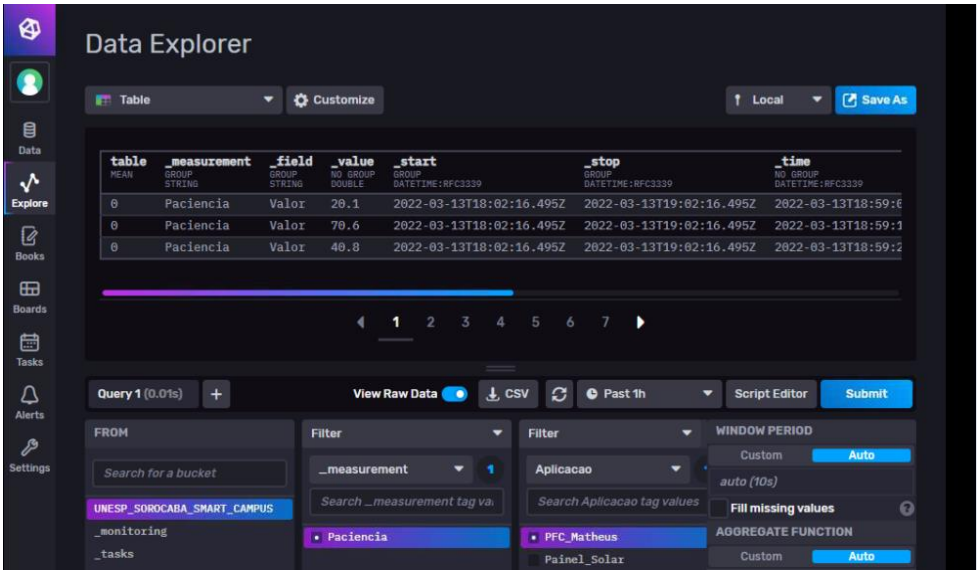
Figura 48 – Criação do *Bucket* no InfluxDB.



Fonte: Autoria própria.

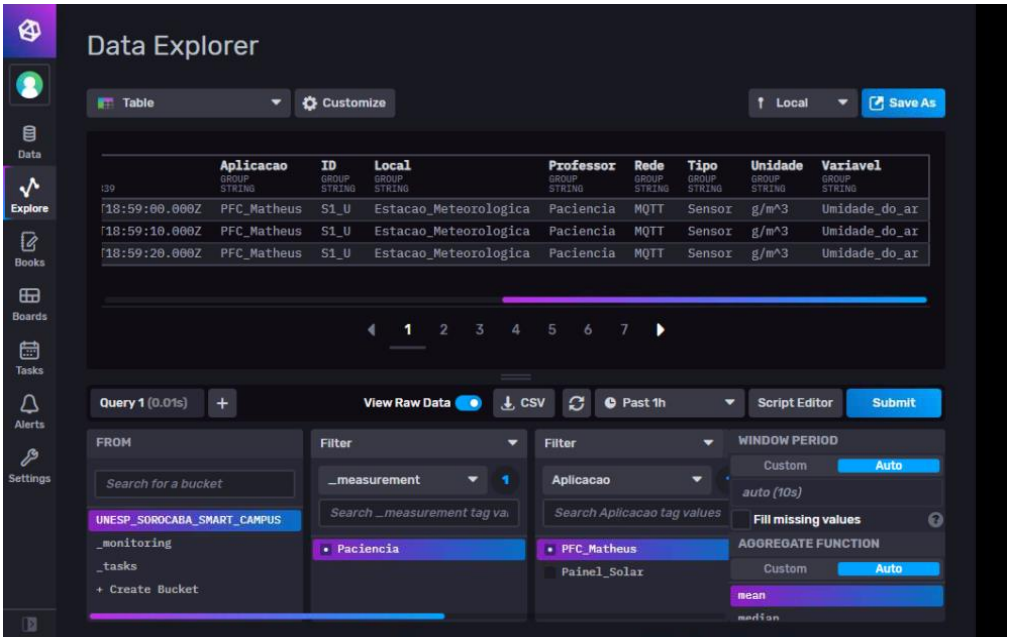
A base de dados criada foi denominada “UNESP_SOROCABA_SMART_CAMPUS”. Acessando a base de dados, é possível encontrar o *measurement* (banco de dados) criado no Node-RED. Neste caso, o *measurement* criado foi “Paciencia”. Importante ressaltar que o nome do *measurement* deve ser o mesmo do professor responsável pela aplicação. Para acessar os dados, basta clicar no botão azul “Submit” à direita. Dessa forma, pode-se ter a visualização dos dados armazenados localmente, como demonstrado nas Figuras 49 a 52.

Figura 49 – Visualização dos valores simulados para o sensor de umidade do ar.



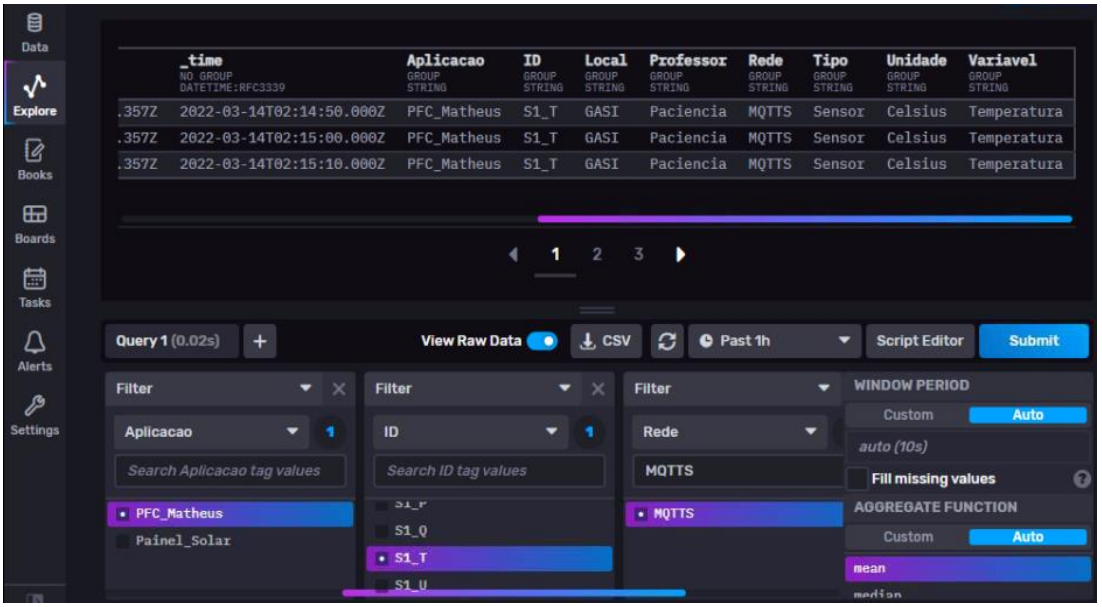
Fonte: Autoria própria.

Figura 50 – Visualização das variáveis relacionadas ao sensor de umidade do ar.



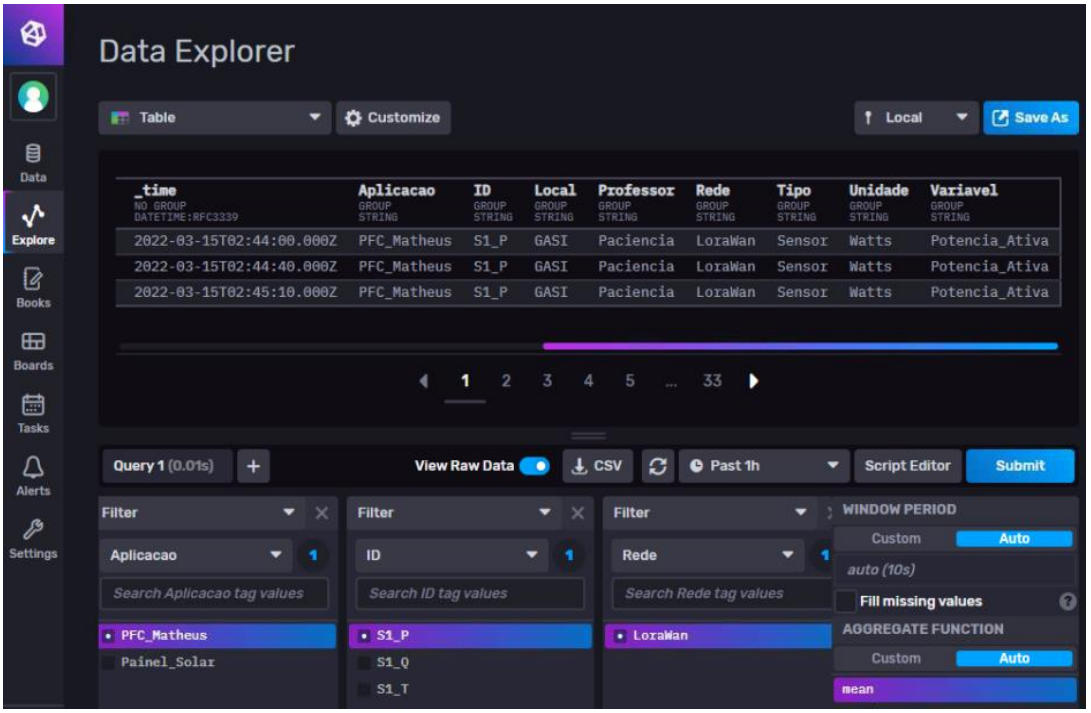
Fonte: Autoria própria.

Figura 51 –Visualização das variáveis relacionadas ao sensor de temperatura.



Fonte: Autoria própria.

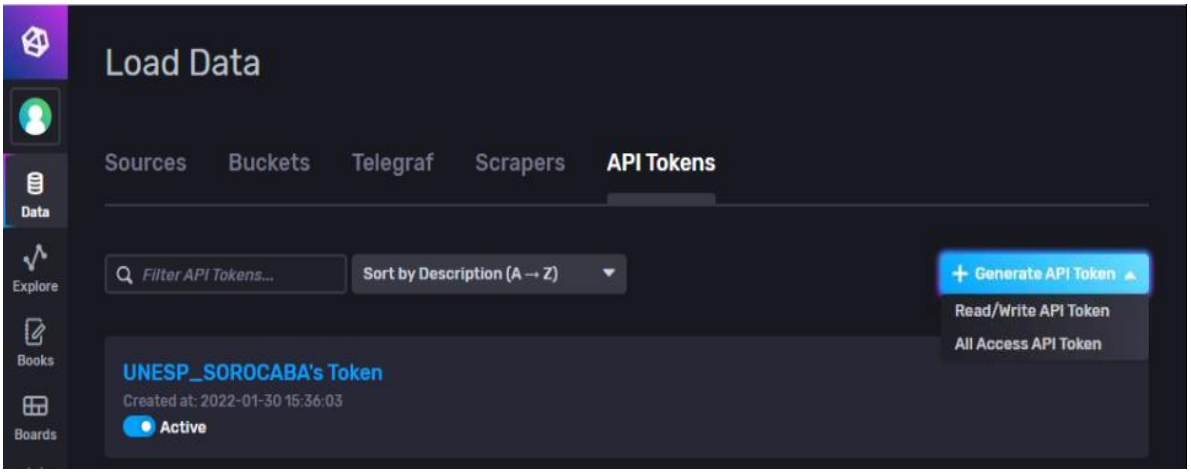
Figura 52 – Visualização das variáveis relacionadas ao sensor de potência ativa.



Fonte: Autoria própria.

Para criar a chave *API Token* utilizada no Node-RED, basta clicar em “Data” e, em seguida, no botão azul “Generate API Token” à direita, conforme Figura 53. A opção escolhida deve ser “All Access API Token”. Por fim, basta nomear a chave e salvar.

Figura 53 – Criação da chave *API Token*.



Fonte: Autoria própria.

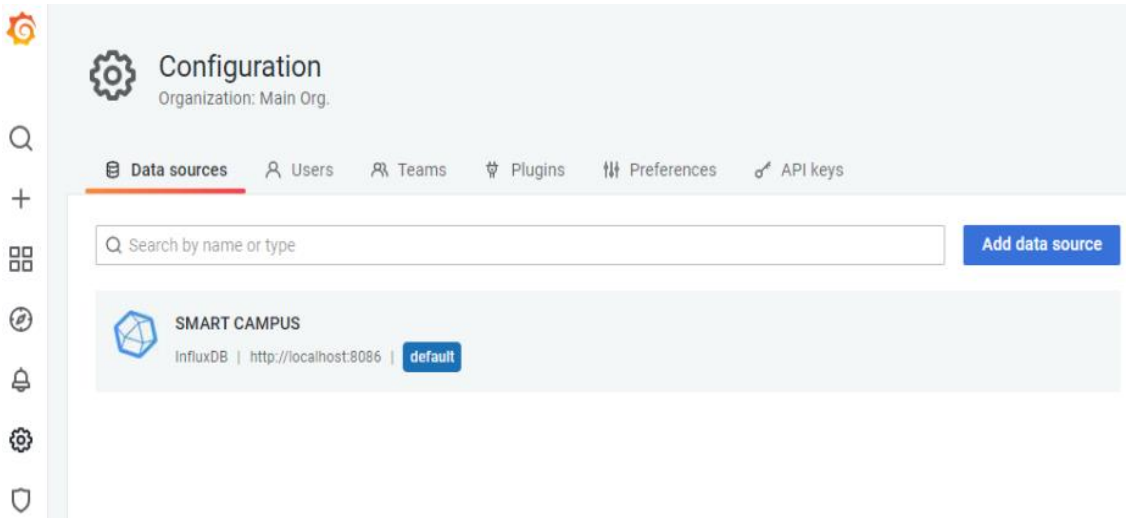
4.2 Desenvolvimento do *Frontend*

4.2.1 Desenvolvimento da Interação

Por fim, os dados armazenados no InfluxDB podem ser acessados pelo Grafana, sendo possível obter uma visualização gráfica de tais dados. O Grafana pode ser acessado remotamente por meio do protocolo HTTP, utilizando-se a porta 3000.

Inicialmente, deve ser feita a integração com o InfluxDB. Para tanto, basta clicar no ícone “Configuration” à esquerda da tela, em “Data sources” e, então, no botão azul “Add data source” à direita, conforme Figura 54. Em seguida, deve-se escolher InfluxDB.

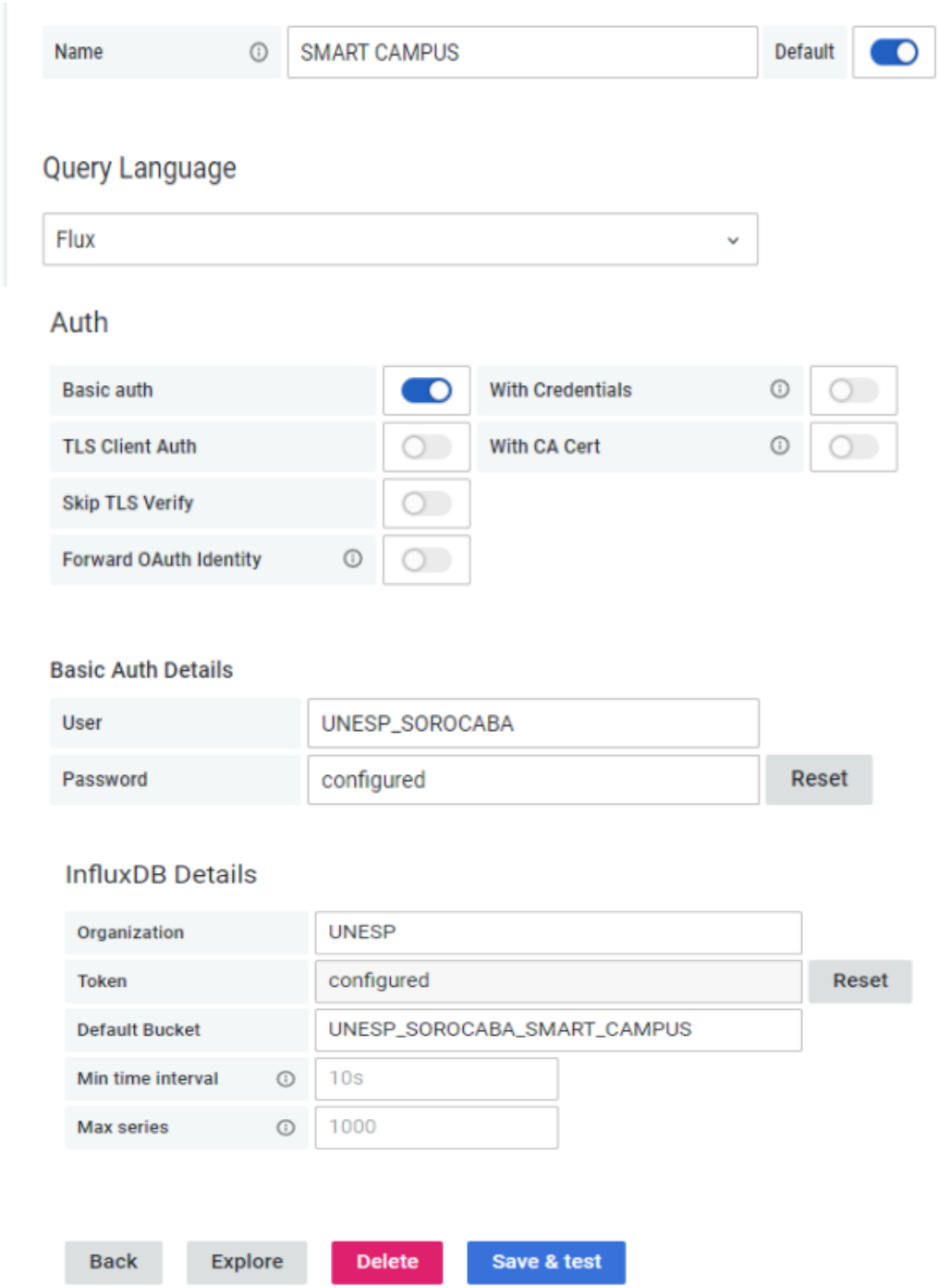
Figura 54 – Passo inicial para integração entre InfluxDB e Grafana.



Fonte: Autoria própria.

São editados os campos, conforme Figura 55. Deve-se escolher a linguagem Flux, pois é a linguagem utilizada a partir do InfluxDB 2.0. No item “Basic Auth Details”, deve-se colocar o usuário e senha do InfluxDB. Por fim, no item “InfluxDB Details”, deve-se colocar o nome da organização, a chave *API Token* e o *Bucket* criados no InfluxDB. Para finalizar, basta clicar em “Save & Test”.

Figura 55 – Configurações para integração entre InfluxDB e Grafana.



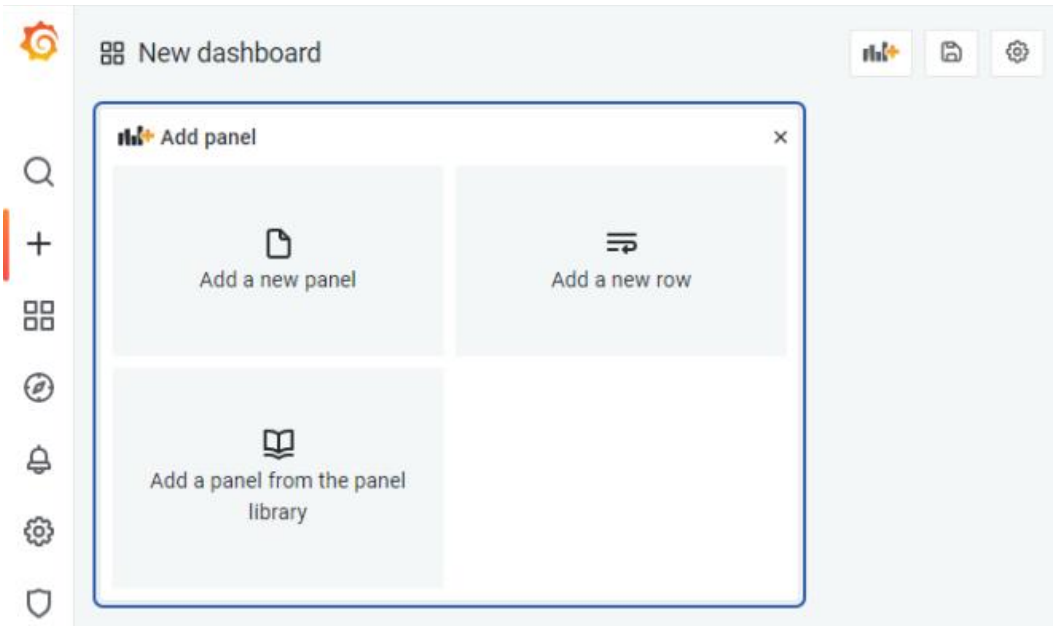
Fonte: Autoria própria.

Realizadas as configurações de integração, foram criados 3 *dashboards*:

- ***Dashboard* Principal:** há uma visão geral das aplicações presentes no *Smart Campus*;
- **Estação Meteorológica:** onde é realizado o monitoramento de aplicações da Estação Meteorológica. Neste caso, pode-se monitorar o sensor de umidade simulado;
- **Painel Solar:** onde é realizado o monitoramento de aplicações relacionadas a um painel solar. Neste caso, são monitorados sensores de temperatura, potência ativa e potência reativa simulados, que serão abordados mais adiante.

Para criar um *dashboard*, basta clicar no ícone "+" à esquerda da tela do Grafana e, então, em "Add a new panel", conforme mostrado na Figura 56.

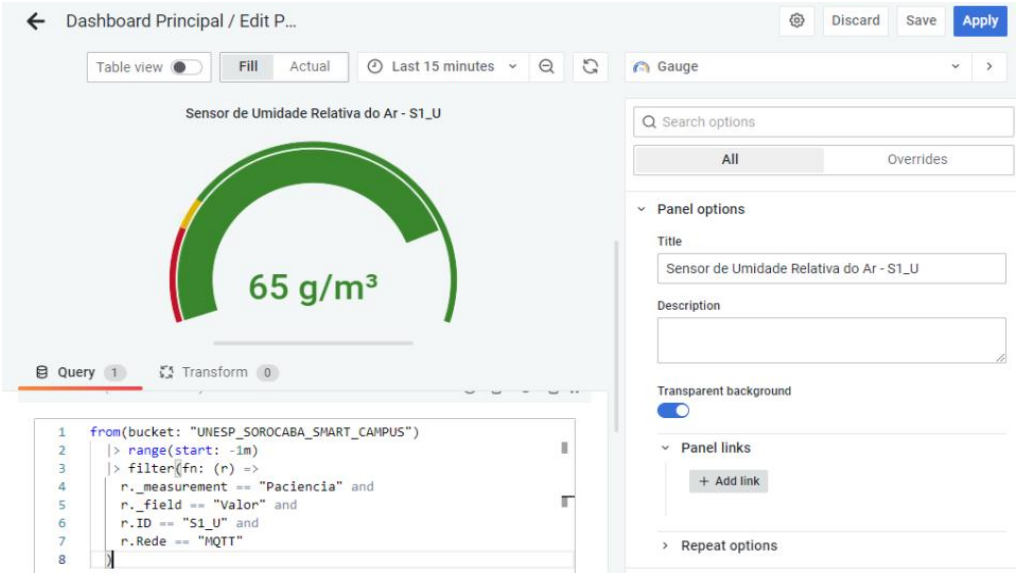
Figura 56 – Criação de um *dashboard* no Grafana.



Fonte: Autoria própria.

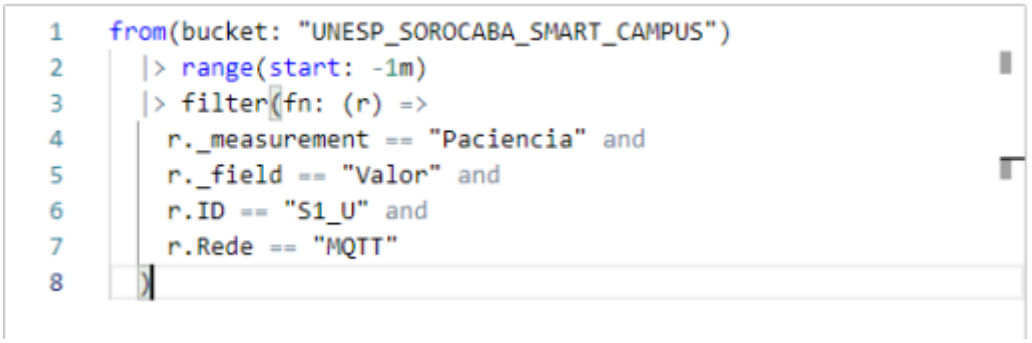
No *Dashboard* Principal, foram utilizados gráficos "Gauge" para a visualização dos dados. Sua configuração é realizada da seguinte forma: no canto inferior esquerdo, é utilizada a linguagem Flux para definição dos parâmetros que serão mostrados no gráfico, enquanto à direita da tela são realizadas configurações gerais, conforme apresentado nas Figuras 57 a 61.

Figura 57 – Configuração do gráfico “Gauge” no Grafana.



Fonte: Autoria própria.

Figura 58 – Configuração da linguagem Flux.

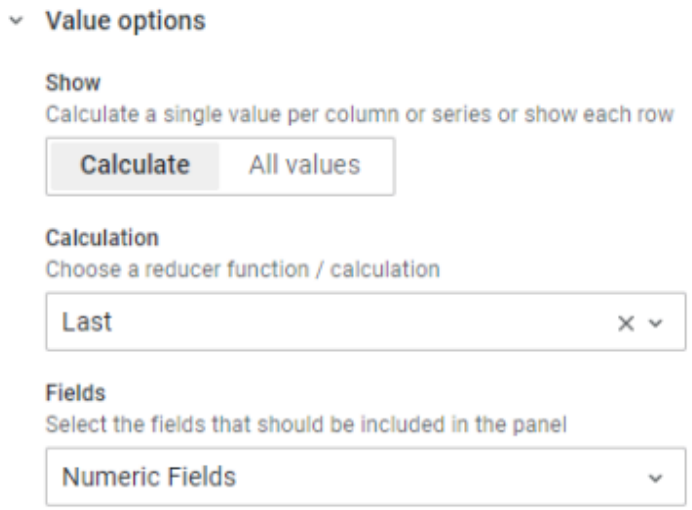


Fonte: Autoria própria.

Na configuração da linguagem Flux, é determinada uma amostragem de dados do último minuto (-1m) e filtrados os dados referentes ao *measurement* “Paciencia”, valor de medição, ID e Rede da aplicação.

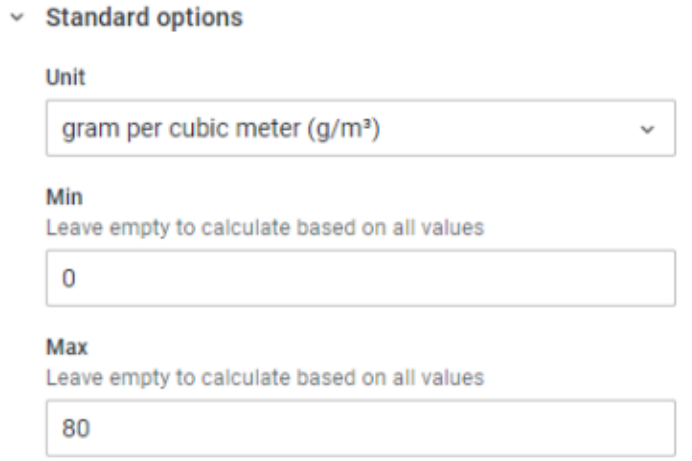
Já nas configurações gerais, no item “Value options” (Figura 59) são escolhidas as opções “Last” e “Numeric Fields”, para que sejam mostrados apenas os últimos valores númericos. No item “Standard options” (Figura 60), é selecionada a unidade de medida, bem como os valores mínimo e máximo do gráfico *Gauge*. Por fim, no item “Threshold” (Figura 61), são escolhidas as cores correspondentes aos valores obtidos. Neste caso, vermelho representa baixo índice de umidade relativa do ar, amarelo representa valor moderado e verde indica valor aceitável.

Figura 59 – Configuração do item “Value options”.



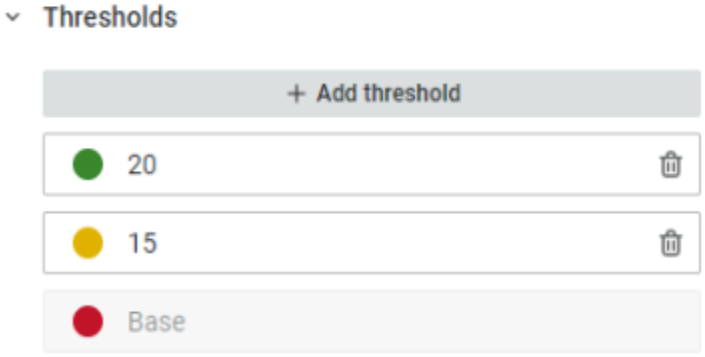
Fonte: Autoria própria.

Figura 60 – Configuração do item “Standard options”.



Fonte: Autoria própria.

Figura 61 – Configuração do item “Thresholds”.



Fonte: Autoria própria.

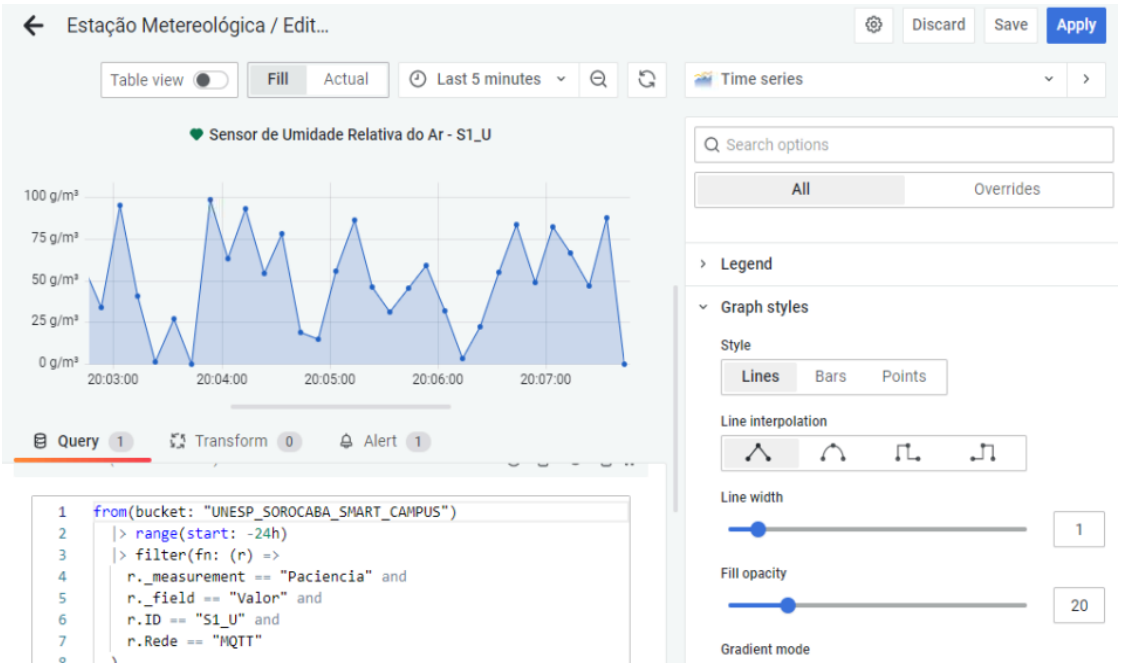
Nos *dashboards* “Estação Meteorológica” e “Painel Solar”, foram utilizados gráficos “Time series”. Sua configuração é mostrada na Figura 62.

O tempo de amostragem configurado foi de 1 dia (-24h). No item “Graphstyles”, foram selecionadas as seguintes opções:

- ***Style:*** Lines;
- ***Line interpolation:*** Linear;
- ***Line width:*** 1;
- ***Fill opacity:*** 20;
- ***Line style:*** solid;
- ***Point size:*** 5.

Por fim, no item “Standard options”, foi selecionada a unidade de medida. Neste caso, “gram per cubic meter (g/m^3)”.

Figura 62 – Configuração do gráfico “Time series” no Grafana.



Fonte: Autoria própria.

5 RESULTADOS

Realizados os desenvolvimentos do *backend* e do *frontend* da Arquitetura IoT, foram realizados testes para verificação e validação do seu correto funcionamento.

5.1 Verificação do Funcionamento da Arquitetura

Devido a pandemia da COVID-19, o trabalho foi integralmente realizado de forma remota. Como não foi possível implementar uma aplicação real na arquitetura, foram realizadas simulações de situações reais do *smart campus* da UNESP Sorocaba.

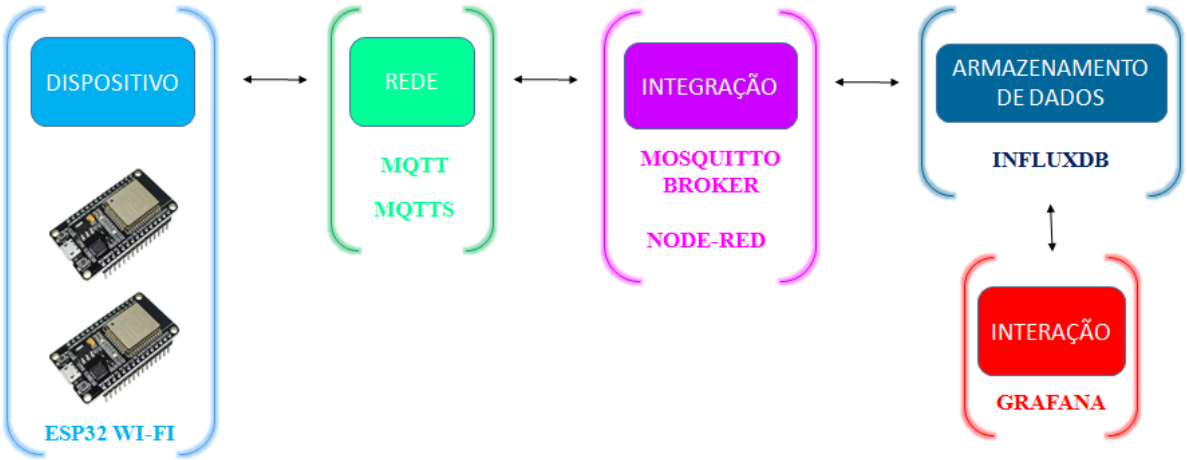
Para o protocolo MQTT, inicialmente, foi simulado um sensor de umidade relativa do ar localizado na estação meteorológica da UNESP. O código da simulação foi compilado em uma placa ESP32 Wi-Fi e a segurança utilizada foi a de usuário/senha.

Em seguida, foi simulado um sensor de temperatura de um painel solar. O código da simulação foi compilado em uma placa ESP32 Wi-Fi. Neste caso, além da segurança por usuário/senha, também foi utilizada a segurança pelo protocolo TLS. Ao se utilizar o protocolo TLS, o protocolo MQTT é denominado MQTTS.

A integração dos dados simulados ocorre no Mosquitto *Broker* e, em seguida, no Node-RED. A escrita e armazenamento dos dados ocorre no InfluxDB e a interação em forma gráfica ocorre no Grafana.

A arquitetura IoT referente à simulação utilizando-se o protocolo MQTT é mostrada na Figura 63 a seguir:

Figura 63 – Arquitetura IoT referente à conectividade MQTT.



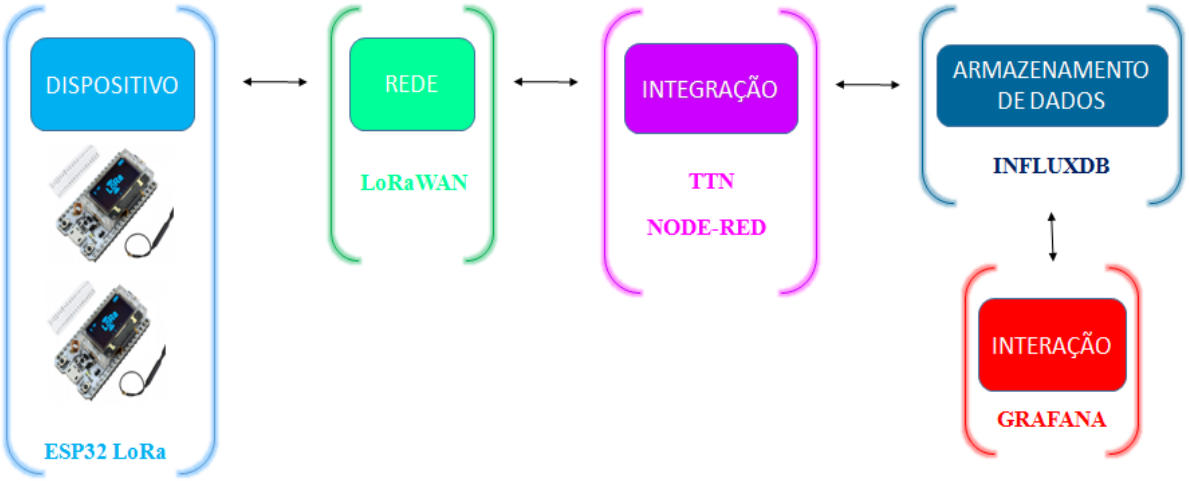
Fonte: Autoria própria.

Para o protocolo LoRaWAN, foram simulados: um sensor de potência ativa, um sensor de potência reativa e um sensor de temperatura, todos referentes a um painel solar. Para tanto, foram utilizadas duas placas ESP32 LoRa como dispositivos. Em uma das placas, foi compilado o código referente ao sensor de potência ativa, enquanto na outra placa foi compilado o código referente aos sensores de potência reativa e de temperatura.

A integração dos dados simulados ocorre no TTN e, em seguida, no Node-RED. A escrita e armazenamento dos dados ocorre no InfluxDB e a interação em forma gráfica ocorre no Grafana.

A arquitetura IoT referente à simulação utilizando-se o protocolo LoRaWAN é mostrada na Figura 64 a seguir:

Figura 64 – Arquitetura IoT referente à conectividade LoRaWAN.



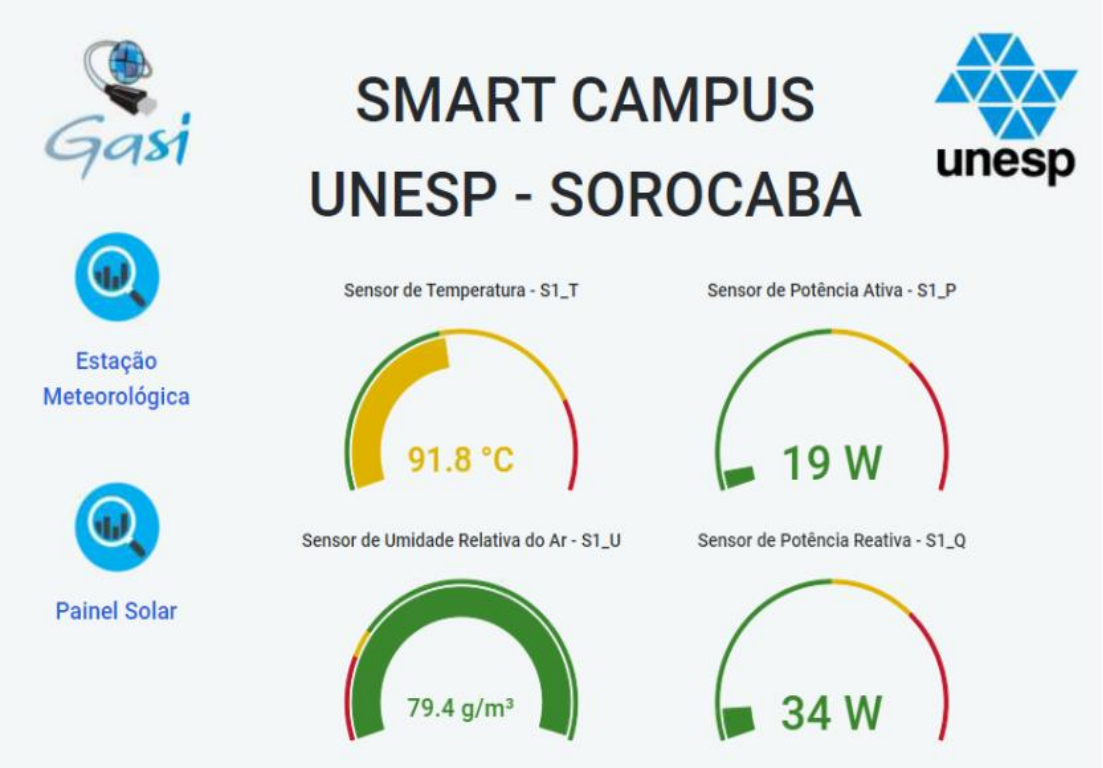
Fonte: Autoria própria.

Foi, então, realizado um monitoramento em tempo real no Grafana dos dados enviados pelas 4 placas. Os dados são atualizados a cada 5s no *dashboard* principal, como mostrado nas Figuras 65 e 66.

Clicando-se no botão azul à esquerda do *dashboard*, foi possível monitorar o histórico dos dados enviados referente a uma aplicação específica. Foi escolhida a aplicação do painel solar, conforme mostrado na Figura 67. Dentre as diversas opções de tempo, foi selecionado o histórico dos últimos 15 minutos.

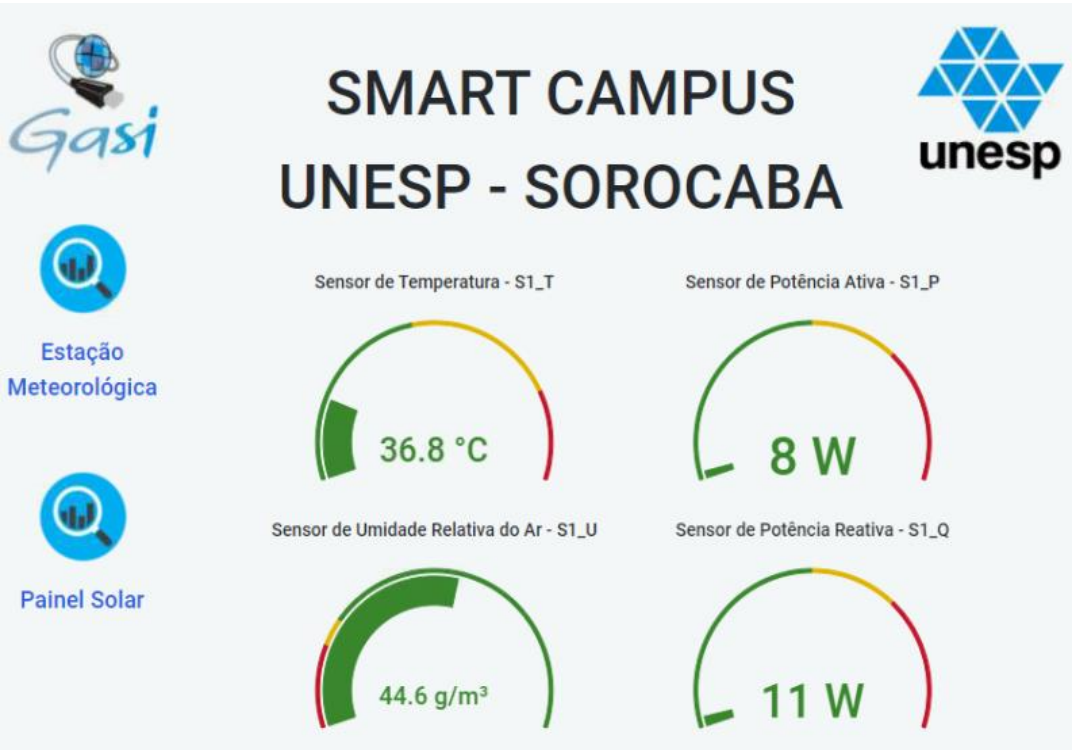
O desenvolvimento do *dashboard* principal no Grafana está em sua etapa inicial. Como continuidade do trabalho, pode-se citar o desenvolvimento de novos botões de interatividade com o usuário, bem como a visualização de dados por meio de outros tipos de gráficos.

Figura 65 – Monitoramento em tempo real dos dados enviados pelas placas.



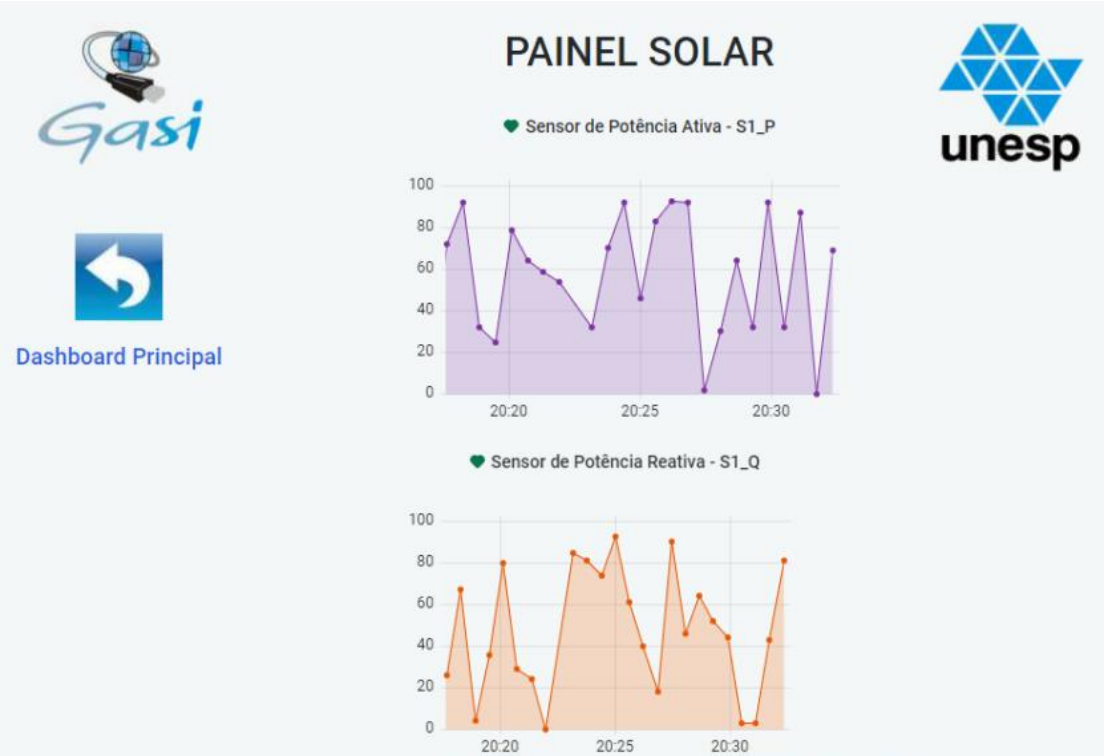
Fonte: Autoria própria.

Figura 66 – Atualização dos dados após 5s.



Fonte: Autoria própria.

Figura 67 – Histórico de dados dos sensores do painel solar.



Fonte: Autoria própria.

Por meio do link a seguir, é possível acompanhar a dinâmica da arquitetura, bem como a chegada dos dados em cada componente da arquitetura: integração, armazenamento de dados e interação.

Vídeo do funcionamento da arquitetura

https://youtu.be/QM-pxwMsbe4

6 CONCLUSÃO

Tendo em vista o rápido avanço tecnológico dos últimos anos, é importante observar tendências e oportunidades relacionadas às novas tecnologias. Nesse contexto, a Internet das Coisas (IoT) tem-se mostrado relevante em diversos campos, tais como: industrial, agroindustrial, doméstico e acadêmico. No meio acadêmico, o chamado *smart campus* é uma tendência nas universidades. Por meio do *smart campus*, é possível gerenciar os recursos da universidade de forma mais sustentável e eficiente, além de aprimorar o aprendizado dos alunos, utilizando-se das novas tecnologias no processo de aprendizagem.

Apesar da implementação de um *smart campus* geralmente requerer altos investimentos, constatou-se que é possível construir uma arquitetura IoT aplicável a *smart campus* utilizando-se dispositivos de baixo custo e *softwares open source*, viabilizando sua implementação em universidades públicas.

Dentre os desafios enfrentados nas etapas do trabalho, pode-se citar o desenvolvimento do programa de obtenção de dados no Arduino IDE, em que foi necessário amplo estudo relacionado aos protocolos MQTT e TLS e da extensão de arquivo JSON.

Além disso, foi utilizada uma conexão Wi-Fi doméstica e, assim, em certos momentos houve instabilidade nas conexões MQTT. Dessa forma, é importante que haja roteadores estáveis e que estejam localizados próximos à aplicação.

Outro grande desafio enfrentado foi o desenvolvimento do projeto em 2 sistemas operacionais distintos. Inicialmente, o projeto foi desenvolvido em uma Raspberry Pi 4, utilizando-se o sistema Raspbian OS. Em seguida, o desenvolvimento foi concluído no Mini PC, que utiliza o sistema Windows. Apesar das dificuldades relacionadas aos comandos dos dois sistemas, a importação de dados em formato JSON do Node-RED e do Grafana facilitaram o processo de transição.

Apesar dos desafios enfrentados, o objetivo principal desse trabalho de desenvolver uma arquitetura IoT de baixo custo, que servirá como base para o desenvolvimento do *smart campus* na UNESP Sorocaba, foi concluído.

Como trabalhos futuros, pode-se citar a implementação de aplicações reais na arquitetura IoT, o desenvolvimento do envio de dados por meio de outros protocolos, como ZigBee e o desenvolvimento de alertas no Grafana, permitindo, assim, o envio de notificações relacionadas aos *dashboards* via Telegram, Discord e Microsoft Teams.

7 REFERÊNCIAS

ALMEIDA, Lahis G. de; BORIN, Juliana F.. **Plataforma Inteligente de Coleta de Resíduos Baseda em Internet das Coisas e LPWAN**. 2019. 15 f. Dissertação (Mestrado) - Curso de Computação, Unicamp, Campinas, 2020. Disponível em: chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/viewer.html?pdfurl=https%3A%2F%2Fsmartcampus.prefeitura.unicamp.br%2Fpub%2Fartigos_relatorios%2FLahis-Plataforma_Inteligente_de_Coleta_de_Residuos_baseda_em_Internet_das_Coisas_e_LPWAN.pdf&clen=4047783&chunk=true. Acesso em: 6 mar. 2022.

ARDUINO. **What is Arduino?** 2018. Disponível em: https://www.arduino.cc/en/Guide/Introduction. Acessoem: 08 mar. 2022.

BERNERS-LEE, Tim. **The Original HTTP as defined in 1991**. 1996. Disponível em: https://www.w3.org/Protocols/HTTP/AsImplemented.html. Acesso em: 5 mar. 2022.

CHAN, Hubert C. Y.; CHAN, Linus. Smart Library and Smart Campus. **Journal Of Service Science And Management.** [S.L.], p. 543-564. nov. 2018. Disponível em: https://www.scirp.org/journal/paperinformation.aspx?paperid=88829. Acessoem: 3 mar. 2022.

CHENG, Jiangfeng*et al*. Industrial IoT in 5G environment towards smart manufacturing. **Elsevier.** Beijing, p. 10-19. jan. 2018. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S2452414X18300049. Acesso em: 27 fev. 2022.

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. 2020. Disponível em: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. Acesso em: 27 fev. 2022.

CLÉTO, Jhonatan. **Projeto de Smart Lock utilizando MicroPython e o Microcontrolador ESP32**. 2021. Disponível em: https://smartcampus.prefeitura.unicamp.br/pub/artigos_relatorios/Jhonatan_Cleto_Smart_Lock_utilizando_a_ESP32.pdf. Acesso em: 24 mar. 2022.

DIERKS, T.; RESCORLA, E.. **The Transport Layer Security (TLS) Protocol Version 1.2**. 2008. Disponível em: https://datatracker.ietf.org/doc/html/rfc5246.html. Acesso em: 6 mar. 2022.

ESP32NET. **The Internet Of Things with ESP32**. 2017. Disponível em: http://esp32.net/. Acesso em: 08 mar. 2022.

EVANS, Dave. **A Internet das Coisas**: como a próxima evolução da internet está mudando tudo. Como a próxima evolução da Internet está mudando tudo. 2011. Disponível em: https://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf. Acesso em: 1 mar. 2022.

FIELDING, R. *et al*. **Hypertext Transfer Protocol -- HTTP/1.1**. 1998. Disponível em: https://datatracker.ietf.org/doc/html/rfc2616. Acessoem: 5 mar. 2022.

GILMAN, Ekaterina *et al*. Internet of Things for Smart Spaces: A University Campus Case Study. **Sensors.** Oulu, p. 1-28. 2 jul. 2020. Disponível em: https://www.mdpi.com/1424-8220/20/13/3716#cite. Acessoem: 27 fev. 2022.

GRAFANA. **The analytics platform for all your metrics**. Disponível em: https://grafana.com/grafana/. Acesso em: 08 mar. 2022.

INFLUXDATA. **InfluxDB is a time series platform**. 2021. Disponível em: https://www.influxdata.com/. Acessoem: 08 mar. 2022.

J.V.BAGGIO; L.F.GONZALEZ; J.F.BORIN. **Smart Parking**: a smart solution using deep learning. A smart solution using Deep Learning. 2020. Disponívelem: https://smartcampus.prefeitura.unicamp.br/pub/artigos_relatorios/PFG_Joao_Victor_Estacionamento_Inteligente.pdf. Acessoem: 23 mar. 2022.

KAUR, Sapandeep; SINGH, Ikvinderpal. A Survey Report on Internet of Things Applications. **International Journal Of Computer Science Trends And Technology.** Amritsar, p. 330-335. abr. 2016.

LORA ALLIANCEŽ (org.). **What is LoRaWAN®**. 2022. Disponível em: https://lora-alliance.org/resource_hub/what-is-lorawan/. Acessoem: 6 mar. 2022.

MADAKAM, Somayya*et al*. Internet of Things (IoT): A Literature Review. **Journal Of Computer And Communications.** India, p. 164-173. maio 2015. Disponível em: http://www.scirp.org/journal/jcc. Acessoem: 27 fev. 2022.

MOSQUITTO (org.). **Eclipse Mosquitto™**: an open source mqtt broker. An open source MQTT broker. 2022. Disponívelem: https://mosquitto.org/. Acessoem: 10 mar. 2022.
NIE, Xiao. Constructing Smart Campus Based on the Cloud Computing Platform and the Internet of Things. **Atlantis Press.** Paris, França, p. 1576-1578. mar. 2013. Disponível em: https://www.atlantis-press.com/proceedings/iccsee-13/4826. Acesso em: 01 mar. 2022

OASIS OPEN (org.). **About Us**. Disponível em: https://www.oasis-open.org/org/. Acessoem: 3 mar. 2022.

OLAYINKA, Ben. **What Is The Things Stack?** 2022. Disponível em: https://www.thethingsindustries.com/docs/getting-started/what-is-tts/. Acessoem: 10 mar. 2022.

OPENJSFOUNDATION (org.). **Node-RED**: low-code programming for event-driven applications. Low-code programming for event-driven applications. 2022. Disponívelem: https://nodered.org/. Acesso em: 10 mar. 2022.

RIO, Larissa Souto del *et al*. **Proposta de ambientes inteligentes IoT sob a ótica da eficiência energética**. 2018. Disponível em: https://www.researchgate.net/publication/328768934_Proposta_de_ambientes_inteligentes_IoT_sob_a_otica_da_eficiencia_energetica. Acesso em: 27 fev. 2022.

SARI, Marti Widya*et al*. Study of Smart Campus Development Using Internet of Things Technology. **Iop Publishing.** Orlando, Flórida, p. 1-4. jun. 2017. Disponível em: https://iopscience.iop.org/article/10.1088/1757-899X/190/1/012032/meta. Acesso em: 01 mar. 2022

SELIMOVIC, Nejra. **The Things Network**. 2021. Disponível em: https://www.thethingsindustries.com/docs/getting-started/ttn/. Acessoem: 10 mar. 2022.

TALAVERA, Jesús Martín *et al*. Review of IoT applications in agro-industrial and environmental fields. **Elsevier.** Cali, p. 283-297. set. 2017. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0168169917304155. Acessoem: 27 fev. 2022.

THE HIVEMQ TEAM (org.). **Introducing the MQTT Protocol - MQTT Essentials: Part 1**. 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/. Acessoem: 3 mar. 2022.

XIONG, Liu. A Study on Smart Campus Model in the Era of Big Data. **Atlantis Press.** Wuhan, China, p. 919-922. jun. 2016. Disponível em: https://www.atlantis-press.com/proceedings/icemeet-16/25869251. Acesso em: 27 fev. 2022.