DESENVOLVIMENTO DE UMA PLATAFORMA INFORMATIZADA PARA PROJETOS IOT COM SUPORTE AOS PROTOCOLOS HTTP E MQTT COM INTERFACE HUMANA E INTERFACE MÁQUINA

Matheus Willian Sprotte, Pablo Dutra Da Silva, Alexandre Altair de Melo
Instituto Federal de Santa Catarina
Câmpus Jaraguá do Sul – Rau – Curso de Bacharelado em Engenharia Elétrica
e-mail: msprotte@outlook.com, pablo.silva@ifsc.edu.br, alexandremelo.br@gmail.com
Trabalho de Conclusão de Curso – 01/07/2024

***Resumo* – O desenvolvimento de um sistema dedicado para um projeto IoT traz diversas vantagens, como: integração direta com o sistema embarcado, liberdade de custos, autonomia para escolha de protocolo e independência para implementação em redes locais. Porém desenvolver um sistema próprio, e além disso a API do mesmo, requer conhecimentos de áreas complementares a formação da engenharia, como, por exemplo, informática. Dessa forma, este trabalho tem como objetivo desenvolver uma plataforma informatizada para projetos IoT com interface de fácil uso para automatizar a integração de projetos IoT a banco de dados, interface humana (*dashboard* com gráficos e *cards* moldáveis para cada projeto), interface máquina e suporte a múltiplos protocolos. Para alcançar esse objetivo foi empregado o desenvolvimento de uma API REST e de uma biblioteca de integração para microcontroladores. A plataforma resultante foi submetida a testes de carga e performance que atestaram a sua robustez, e indicaram que ela tem e uma boa performance quando comparado com outros projetos similares. Nos testes de carga com milhares de requisições não houve nenhuma perda de dado e obteve-se um tempo de resposta médio de 21 ms. Além disso a interface humana na plataforma foi submetida a heurística de Nilsen a qual atestou sua qualidade. Por fim, essa plataforma foi implementada no hardware de baixo custo Raspberry Pi 3 e teve suas documentações de uso e implementação publicadas.**

***Palavras-chave* – Internet das coisas, servidor web, API REST, sistemas embarcados.**

DEVELOPMENT OF A COMPUTERIZED PLATFORM FOR IOT PROJECTS WITH SUPPORT FOR HTTP AND MQTT PROTOCOLS WITH HUMAN INTERFACE AND MACHINE INTERFACE

***Abstract* – Developing a dedicated system for an IoT project brings several advantages, such as direct integration with the embedded system, freedom of costs, autonomy to choose the protocol, and independence for implementation in local networks. However, developing a dedicated system, in addition to its API, requires knowledge of areas that complement engineering training, such as computer science. Thus, this work aims to develop a computerized platform for IoT projects with an easy-to-use interface to automate the integration of IoT projects with a database, a human interface (dashboard with graphics and cards that can be molded for each project), a machine interface, and support for multiple protocols. To achieve this goal, a REST API and an integration library for microcontrollers were developed. The resulting platform was subjected to load and performance tests that attested to its robustness and indicated that it performed well when compared to other similar projects. In the load tests with thousands of requests, there was no loss of data and an average response time of 21 ms was obtained. Furthermore, the human interface on the platform was subjected to the Nilsen heuristic, which attested to its quality. Finally, this platform was implemented on the low-cost Raspberry Pi 3 hardware and its usage and implementation documentation was published.**

***Keywords* – Internet of things, web server, REST API, embedded systems**

I. INTRODUÇÃO

O conceito de Internet das coisas, ou IoT (*Internet of Things*), surgiu em 1999 por Kevin Ashton, pesquisador britânico do MIT (Massachusetts Institute of Technology) e se referia inicialmente a objetos conectados com identificadores de radiofrequência, ou RFID (*Radio Frequency Identification*) [1]. Entretanto esse termo ganhou maior amplitude e significância em 2005 com a ascensão dos WSNs (*wireless sensor networks*) e da *Web* 2.0 e em 2012 com a evolução da computação móvel e dos sistemas operacionais, ou seja: esse termo passou a abranger de forma mais geral os dispositivos capazes de sensorear o ambiente (bem como realizar *feedbacks*), tratar os dados e transmiti-los em uma rede integrada [2].

Esse avanço tenológico se deu em conjunto com a evolução e barateamento das estruturas de rede e dos microprocessadores, o que proporcionou o crescimento do IoT nas mais diversas áreas, como: saúde, automobilística, logística, automação

industrial, manutenção preditiva, entre outros [3]. Além disso, o barateamento dos microprocessadores possibilitou o surgimento de um nicho de computadores de baixo custo denominados computadores de placa única, ou SBC (*single board computers*). Dentre eles, destacam-se o Raspberry Pi 3/4, BeagleBoard, Toradex e Intel Edison. Esses dispositivos são compactos, baratos e robustos de forma que são comumente aplicados em projetos de automação, IoT, ou como um substituto miniaturizado de computadores [4] [5].

O desenvolvimento de projetos IoT, no geral, faz uso de microcontroladores com conectividade *wireless* ou Bluetooth (como por exemplo ESP32, RP2040 e Realtek RTL8710) e servidores com sistemas responsáveis por armazenar dados e viabilizar a troca de informações entre os dispositivos IoT ou aplicativos e serviços, formando assim uma rede integrada [1]. Esses servidores podem ser providos por serviços na internet, como por exemplo: Blynk, WEGnology e Microsoft Azure ou podem ser desenvolvidos e implementados como plataforma própria e exclusiva, por meio de programação, na rede local dos dispositivos IoT.

Por exemplo, o projeto apresentado em [6] desenvolveu um sistema IoT próprio para sensores de medição de qualidade do ar, ou IAQ (*indoor air quality*), em laboratórios denominado iAQ Plus. Esse sistema coleta dados de dispositivos através de conexão Wi-Fi e os armazena em um banco de dados MySQL, se comunicando via procolo HTTP (*Hypertext Transfer Protocol*). Inclusive o sistema conta com uma interface *web* para visualizar os dados coletados, porém não oferece funcionalidades para análise e processamento desses dados. Já em [7], foi desenvolvido outro sistema orientado para IAQ porém capaz de monitorar não só a qualidade do ar, mas também, temperatura e umidade. Adicionalmente, faz o uso de protocolo MQTT (*Message Queuing Telemetry Transport*) e de um Raspberry Pi como servidor local.

De toda forma, ao devolver um sistema IoT próprio, como os mencionados, garante-se que projeto seja independente e livre dos custos ou diretrizes dos serviços da internet, porém desenvolver um servidor e sistema IoT, demanda conhecimentos tanto no campo da informática como da engenharia, o que pode se tornar um desafio para o andamento do projeto.

Essa é uma problemática vigente inclusive no cenário de desenvolvimento de projetos de pesquisa na área de IoT no IFSC Campus Rau, onde há a carência de uma plataforma própria e escalável para comportar vários projetos sem necessidade de programação avançada (a próxima seção detalhará de forma mais aprofundada os desenvolvimentos locais).

A partir do texto exposto, este trabalho visa a construção de uma plataforma informatizada para projetos IoT com interface de fácil uso para automatizar a integração de projetos IoT a banco de dados, interface humana (*dashboard* com gráficos e *cards* moldáveis para cada projeto), interface máquina e suporte a múltiplos protocolos, e a implementação dessa plataforma em servidor na rede local do laboratório de pesquisa do IFSC Campus Rau usando o *hardware* de baixo custo Raspberry Pi 3. Pra isso propõe-se a elaboração de testes de validação a respeito dos protocolos MQTT e HTTP REST, tanto em nível de carga de trabalho no servidor quanto facilidade de uso e integração; desenvolvimento de plataforma *web* de fácil uso para criação de *dashboards* e microcódigos em projetos de IoT, juntamente com servidor e banco de dados implementados em Raspberry Pi 3; realização de testes funcionais e de mesa com uma rede de sensores fictícia e iniciar a migração dos projetos IoT do sistema legado para essa nova plataforma, bem como documentações a respeito do uso dela na rede do laboratório e sua implantação em outras redes.

Com isso, pretende-se que esse trabalho beneficie não só o ecossistema de projetos de pesquisa do IFSC Campus Rau, mas também a toda a comunidade de desenvolvimento IoT de código aberto ao propiciar um sistema confiável, robusto, escalável e passível de ser implementado no hardware de baixo custo Raspberry Pi 3.

II. REVISÃO BIBLIOGRÁFICA

Os projetos IoT podem valer-se tanto de sistemas de terceiros ou de um desenvolvimento próprio pra o projeto em si e os dois formatos possuem suas vantagens e empecilhos. A seguir, esses sistemas são mapeados, detalhados e é apresentado o estado da arte de desenvolvimentos de projetos independentes.

A. Pesquisa de mercado

Através de pesquisa de mercado, foram mapeadas as principais plataformas de desenvolvimento IoT com objetivos comerciais (todas são pagas ou *fremium*):

- Google Cloud IoT Platform: Plataforma IoT da Google que oferece recursos de inteligência artificial, aprendizado de máquina e gerenciamento de múltiplos dispositivos [8]. Para seu uso é necessário a contratação de serviços de parceiros de tecnologia e de implementação, que possuem custos variados. Os preços desses parceiros não são divulgados abertamente.
- Plataforma de serviços da Web da Amazon para IoT: A plataforma oferece recursos de integração de dispositivos e criação de *dashboards*, além disso, permite integração facilitada com outros serviços da Amazon [9]. É gratuita para testes, porém após o período de teste possuí custos que aumentam conforme o volume de uso (quantidade de dispositivos conectados, quantidade de dados gerados para cada dispositivo, dados consultados por mês, etc.). Por exemplo: de acordo com o simulador de preços da própria empresa [10] o serviço AWS IoT Analytics custaria R$ 957,64 mensais (em conversão direta) para o seguinte volume de dados mensais definidos arbitrariamente: 90 dispositivos IoT conectados, 90 MB de dados gerados por cada dispositivo, 100 pipelines de dados, 1000 MB por de dados consultados por mês e 1000 consultas.
- ThingWorx: O ThingWorx é uma plataforma IoT específica para a indústria e oferece um ambiente

segmentado em produtos para cada aplicação: fabricação, prestação de serviços, engenheira, manutenção preditiva e gerenciamento industrial [11]. Os preços desses produtos em específico não são abertamente divulgados. Porém preço para o *driver* relacionado a IoT "MQTT Client"foi divulgado em R$2.976,7 anuais (em conversão direta) [12].

- Plataforma Microsoft Azure IoT: A plataforma Azure IoT oferece uma variedade de serviços e ferramentas para desenvolvimento e gerenciamento de soluções IoT e é voltada para integração com outros serviços da Microsoft, como Azure Machine Learning e Power BI [13]. Possuí um sistema de cobrança similar aos da Plataforma de serviços da Web da Amazon para IoT, mencionada anteriormente.
- Oracle IoT Intelligent Applications: A Oracle oferece uma plataforma para desenvolvimento e gerenciamento de soluções IoT orientadas para logística, fábrica e gerenciamento industrial. Além disso ela possuí integração com bancos de dados Oracle. Os preços dessa plataforma não são abertamente divulgados [14].
- IBM Watson IoT: A IBM Watson IoT busca oferecer uma solução focada em gerenciamento de dispositivos inteligentes com interface simples e objetiva, e exclusivamente com uso do protocolo MQTT. Além disso, oferece suporte a integração com outras soluções IBM. Essa ferramenta é paga, porém seu preço não é abertamente divulgado.
- IoT WEGnology®: Plataforma de desenvolvimento IoT *Low-Code* baseada em *Cloud Computing* e protocolo MQTT, possui integração com outros produtos da empresa [15]. Essa ferramenta possuí demonstrações gratuitas, porém seu uso regular (mais de 90 dias) é pago. Os preços não são abertamente divulgados.
- Blynk: Sistema IoT *low-Code*, focado em baixo custo, conta com biblioteca de integração para vários dispositivos e faz uso de protocolo MQTT. Além disso, tem suporte a aplicativo *mobile* configurável [16]. Essa ferramenta possuí versão gratuita com limites de dispositivos e de histórico de dados (permite apenas dois dispositivos e mantém dados salvos por apenas uma semana). Na versão mais completa, que permite mais de 100 dispositivos e mantém histórico de dados por seis messes, a ferramenta tem o preço de R$ 509,85 mensais (em conversão direta).
- Arduino Clound IoT: PLataforma IoT desenvolvida pela própria Arduino, focada em facilidade de uso, controle de aparelhos e possui integrações com demais produtos da marca, bem como diversas bibliotecas prontas [17]. Possuí verão gratuita com as seguintes limitações: suporta apenas dois dispositivos conectados e armazena os dados por um único dia. No seu pacote mais completo, permite até 100 dispositivos conectados e mantém o histórico de dados por um ano e custa R$ 302,95 mensais (em conversão direta).

B. Desenvolvimentos cientfficos/educacionais

Além dos trabalhos já citados [6] e [7] na seção de Introdução, a seguir são apresentados outros desenvolvimentos acadêmicos de sistemas IoT e por fim é detalhado o cenário de desenvolvimento no IFSC Campus Rau.

Em [18] foi proposto um sistema IoT para fazer a telemetria de água subterrânea em cidades inteligentes. Esse sistema usa de requisições HTTP POST para transmitir os dados em formato JSON (*JavaScript Object Notation*) e os armazena em um banco de dados MongoDB (não relacional). Além disso, ele permite compartilhar esses dados por meio de uma API REST.

O sistema IoT desenvolvido em [19] tem essa mesma arquitetura (protocolo HTTP e dados estruturados em JSON) porém aplicado ao monitoramento de rodovias usando sensores móveis. Além disso, permite a exportação de dados em arquivos CSV (*comma-separated values*).

Uma plataforma de desenvolvimento IoT de código aberto largamente usada é a NodeRED, um desenvolvimento da IBM [20]. Essa ferramenta é gratuita, porém sua utilização exige aprendizado em programação por blocos, diversos níveis de parametrização e possuí restrições de escalabilidade.

A variedade de protocolos e arquitetura para o desenvolvimento de um sistema IoT é discutida detalhadamente em [21], onde concluiu-se que os protocolos MQTT e HTTP são os preferíveis para desenvolvimento de projetos IoT, sendo o MQTT recomendável para aplicações que demandam baixa latência e consumo de energia (por parte do microcontrolador, já que a mensagem MQTT exige menos *bytes* para ser enviada) e o HTTP mais adequado para aplicações que envolvam maior processamento em nuvem (tende a demandar menos carga de trabalho dos servidores, já que não exige serviços adicionais para integração com *broker* MQTT). Além disso, os autores destacam como desafio a dificuldade de interoperabilidade entre esses protocolos.

Nesse sentido, em [22] é desenvolvido um sistema IoT capaz de se comunicar via múltiplos protocolos (tanto MQTT quanto HTTP) com suporte a cinco projetos distintos, mas o mesmo carece de recursos de usabilidade, regras de uso, fácil implementação em múltiplos ambientes de desenvolvimento, bem como não suporta integrações automáticas com microcontroladores (geração de microcódigo).

No cenário de desenvolvimento de projetos IoT no IFSC Campus Rau, em [23], foi apresentado o desenvolvimento de um sistema IoT próprio para aplicação em telemetria de motores elétricos com interface *web* e os seguintes recursos: gravação das leituras dos sensores em banco de dados (MySQL), exibição desses dados em gráficos e *cards* em tempo real e geração de relatórios. Em 2021, o projeto de pesquisa cujo os resultados são apresentados em [24] usava um sistema IoT para realizar monitoramento energético, porém esse sistema era provido por um serviço da nuvem que deixou de funcionar, pois os provedores mudaram suas políticas e passaram a monetizar o uso da ferramenta.

A solução encontrada foi adaptar o sistema IoT desenvolvido

no projeto citado anteriormente ([23]) para comportar esse projeto realcioand a monitoramento energético também. Em 2022, esse sistema foi adaptado novamente para comportar o projeto que é apresentado em [25], voltado para telemetria e controle de posição de uma placa fotovoltaica.

Apesar de robusto e funcional esse sistema que comportou os três projetos de pesquisa citados não é escalável, pois demanda alguém com conhecimentos específicos de programação (PHP, JavaScript e MySQL) para realizar a inserção de novos projetos. Além disso, a ferramenta desenvolvida faz um uso limitado do protocolo HTTP não permitindo a troca de informação de dispositivos IoT entre si, nem mesmo possibilitando integração com outros sistemas ou aplicativos, o que inviabiliza o desenvolvimento de redes mais complexas.

III. A ARQUITETURA BÁSICA DE UM SISTEMA IOT

IoT se refere de forma ampla a interconexão de objetos cotidianos ou industriais à internet e sistemas de monitoramento e controle. Para tal, pode-se fazer uso de sensores e atuadores dos mais diversos tipos, microcontroladores, rede *wireless* e sistemas envolvendo banco de dados, servidores e interfaces gráficas. Nesse contexto surge a indústria 4.0, termo que se refere a quarta revolução industrial vigente, a qual tem como uma de suas principais características o uso de sistemas IoT para monitorar e controlar de forma mais eficiente o funcionamento de máquinas e objetos industriais, visando à redução de custos de produção e manutenção [26]. Um monitoramento constante proporciona melhoria da eficiência operacional, menos vulnerabilidades e anomalias que segundo [27] podem ser previstas com o uso de sensores que monitoram grandezas como: temperatura, pressão, tensão, corrente, entre outros. Dessa forma, o conceito mais geral de IoT ramificou-se para o conceito de internet das coisas para a indústria, ou IIoT (*Industrial Internet of Things*).

De acordo com [2], o IIoT caracteriza-se por uma interação M2M (*machine to machine*) com foco em supervisionamento/monitoração, controle em malha fechada e controle de intertravamento. De forma que o IoT é voltado para M2U (*machine to user*), ou seja, focado em ser acessível, ter uma interface amigável em menos volume de dados. Essa diferenciação não impacta na arquitetura geral de um sistema IoT, mas sim em camadas específicas dela.

A arquitetura de um sistema IoT, de acordo com a União Internacional de Comunicação, ou ITU, (*International Telecommunication Union*) consiste em cinco camadas principais: sensoriamento, acesso, rede, ponte de acesso e interface de aplicação. Em, [28] é apresentado um modelo com apenas 3 camadas: sensoriamento, rede e serviço. Um diagrama que representa esse modelo é representado na Fig. 1

Na Fig. 1 a camada dos sensores e atuadores é representada na base da pilha, a camada de rede é representada na camada intermediária (”IoT GATEWAYS”) e a cada de serviços propriamente é representada no tipo da pilha (”INTERNET”).

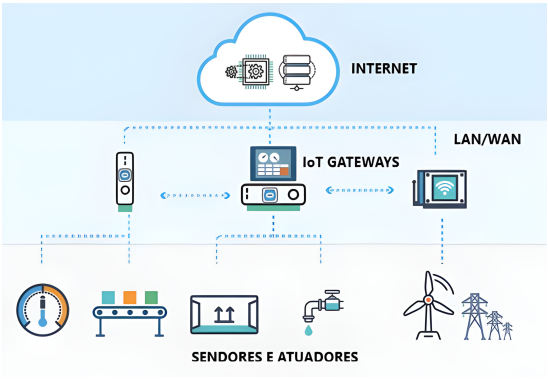


Fig. 1. Diagrama da arquitetura de um sistema IoT em 3 camadas. Adaptado de [29].

Já [3] propõe a divisão e um sistema IoT em 4 camadas, pela perspectiva de suas funcionalidades oferecidas individualmente: sensoriamento, rede, serviço e interface. No decorrer deste texto, será considerada essa subdivisão devido a sua boa distinção.

A. Sensoriamento

A camada de sensoriamento (representada no primeiro nível da pilha da Fig. 1), de acordo com [3] refere-se propriamente aos sensores que coletam as grandezas a serem monitoradas, a dispositivos Bluetooth e RFID capazes de coletar informações de proximidade e autenticação entre seus correspondentes e a sensores WSNs. Os sensores, tanto analógicos quanto digitais, necessitam de microcontroladores para receber suas leituras e comunicar-se entre si e com a rede *wireless*, formando assim uma WSNs. Os sensores analógicos, muitas vezes, necessitam de circuitos de condicionamento de sinais específicos para adequação da sua resposta aos níveis de tensão requeridos para integração com os microcontroladores.

Já os microcontroladores, segundo [1], são dispositivos que incluem processador, memória RAM e armazenamento em um único *chip*. Eles se comunicam com os sensores usando de bibliotecas específicas para cada sensor ou através de leituras dos sinais analógicos provenientes dos sensores. Além disso, podem ter funções extras que variam de acordo com suas propostas de uso, como exemplo pode-se citar recursos de conectividade (*wireless*, Bluetooth etc) presentes no microcontrolador Espressif ESP32.

B. Rede

Na perspectiva da Camada de Rede, a conexão entre a camada de sensoriamento e a camada de serviço pode ser feita por meio do protocolo mais tradicional, o HTTP, ou protocolos mais específicos para o fluxo de dados sensoreados como o MQTT. As Fig. 2 e Fig. 3 ilustram o funcionamento desses dois protocolos.

O protocolo HTTP baseia-se em uma estrutura básica de requisições e respostas [1], onde o cliente envia informações através da requisição e o servidor retorna uma resposta corresponde às informações de entrada.

Já o protocolo MQTT funciona da seguinte forma: O servidor

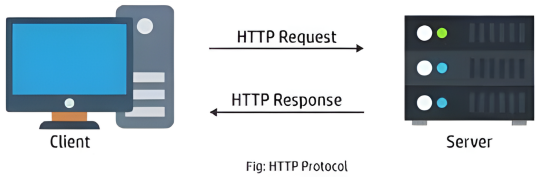


Fig. 2. Diagrama representativo do protocolo HTTP. Retirado de [30].

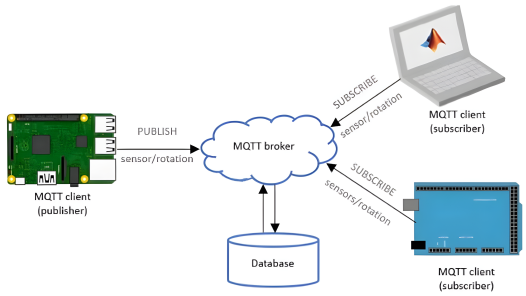


Fig. 3. Diagrama representativo do protocolo MQTT. Retirado de [31].

(*broker*) estabelece dois tipos de conexões entre os dispositivos, integradas a partir de um tópico: *subcriber* e *publisher*. O primeiro serve para realizar o envio de dados a partir de sensores ou outros dispositivos para um tópico específico, e o segundo trata da leitura desses dados a partir de outros dispositivos conectados ao *broker*. Além disso, o protocolo MQTT é baseado em diferentes níveis de QoS (*quality of service*), onde níveis mais baixos representam conexões mais rápidas e níveis mais altos conexões mais seguras (confiáveis). Por exemplo: para um QoS nível 0 os dados são apenas enviados ao *broker*, mas para um QoS nível 1 os dados são enviados ao *broker* e então o *broker* os envia novamente pra o *publisher* para assegurar que o dado foi recebido corretamente. As respostas para requisições HTTP podem ser customizadas por parte do servidor para simular esses níveis de forma não nativa.

C. Serviço

A Interface de Serviço engloba um conjunto de programas e protocolos capazes de troca e armazenamento de informações, gerenciamento de dados, mecanismos de busca e comunicação podendo valer-se de banco de dados, programas *web* e servidores [22]. Esses programas podem ser desenvolvidos nas mais diversas linguagens, como C++, Java, PHP etc. e o banco de dados pode ser tanto de natureza não relacional (Elastic, Mongo, Redis, etc) quanto relacional (MySQL, Oracle, SQL Server etc). O primeiro tipo se vale de estruturas diversas (documentos, grafos etc) para armazenar dados, isso é denominada em uma abordagem computacional como NoSQL (*Not Only SQL*). Já o segundo tipo é baseado no modelo relacional (dados estruturados em tabelas).

Como resultado dessa integração entre os programas e bancos de dados tem-se diversas arquiteturas de serviços e APIs (*Application Programming Interface*), sendo as mais comuns para desenvolvimento IoT as arquiteturas SOAP (*Simple Object Access Protocol*) e REST (*representational state transfer*) [21]. A primeira faz uso de XML para trocar informações e é mais focada para realizar operações. Já a arquitetura REST pode usar de diversos formatos para trocar informações, principalmente JSON e é orientada para acessar dados e não operações propriamente. De qualquer forma, a última é mais aceita pelos navegadores de internet, mais fácil de integrar com outros sistemas e mais rápida [32]. Nessa arquitetura, de acordo com [33] sempre haverá um verbo para requisição do cliente para o servidor, um *header* (cabeçalho da requisição, que transmite dados sobre elas), um *path* (caminho na rede opara acessar os dados) e os dados do corpo da requisição propriamente.

D. Interface

Por fim, interface refere-se tanto para interação humana como para a de troca de informações com outros dispositivos IoT. Para o primeiro caso é comum o uso de *dashboards* interativos onde se mostram dados, gráficos e informações relevantes ao monitoramento e a tomada de decisões por parte do usuário. Já o segundo caso é correspondente ao IIoT, onde outras máquinas ou dispositivos acessam os dados via *endpoints* ou protocolos específicos, visando automatização de processos, controle e supervisório [2].

Tanto a camada de serviço quanto a de interface são dependentes de hospedagem em servidor para que estejam disponíveis na rede, sendo que o servidor pode ser tanto um computador de mesa ou computador portátil, quanto uma máquina desenvolvida especialmente para esse fim (com sistema operacional específico para essa aplicação, como o Windows Server, Ubuntu Server ou Solaris). Inclusive, SBCs de baixo custo como o Raspberry Pi 3, podem embarcar Ubuntu Server e cumprir o papel de servidor local em diversas aplicações [5].

IV. DESENVOLVIMENTO DE UMA PLATAFORMA IOT ESCALÁVEL

O serviço de um sistema IoT pode ser desenvolvido via diversas linguagens de programação e fazer uso de uma grande variedade de banco de dados. Para o desenvolvimento da plataforma de projetos IoT proposta nessa pesquisa, optou-se por desenvolver um serviço no formato de API REST programado em Java 21, com interface *web* programada em JavaScript fazendo uso do *framework* Bootstrap.

Essas escolhas justificam-se pelos seguintes motivos: a arquitetura API REST é uma arquitetura de desenvolvimento de *softwares* específica para serviços, é altamente escalável e de fácil integração tanto com interfaces *web* quanto outros sistemas. Optou-se por Java 21 devido a esta linguagem ter suporte a dois importantes *frameworks*: Spring Boot e Eclipse Paho MQTT. O primeiro possibilita facilitar a integração com banco de dados através de JPA (*Java Persistence API*), permite a geração de *enpoints* e possuí servidor *web* integrado (Tomcat). Já o Eclipse Paho MQTT é o *framework* que viabiliza a criação tópicos na rede MQTT e a atuação como *subscriber* ou *publisher* na rede. Para interface *web* da plataforma optou-se no lado cliente

pelo uso das tecnologias HTML, JavaScript e o *framework* Bootstrap. Este último se baseia em vários conceitos de IHC (Interface Humano Computador), como, por exemplo, adaptar as informações para diferentes tamanhos de resolução de tela. O desenvolvimento da plataforma e essas tecnologias empregadas serão detalhadas nas próximas subseções a seguir.

A. A estrutura da API e banco de dados

O funcionamento da API é baseado da relação de três entidades de dados principais, as quais complementam-se entre si dentro de construtores e laços lógicos para conferir a plataforma escalabilidade tanto em nível de múltiplos projetos quanto múltiplos dados e múltiplos dispositivos. Da forma como está arquitetado, não há um limite lógico para nenhuma dessas instâncias, porém há limitações inerentes às condições de *hardware* e redes que serão discutidas posteriormente.

A Fig. 4 ilustra o modelo ER (relacionamento de entidade) para as entidades de dados mencionadas:

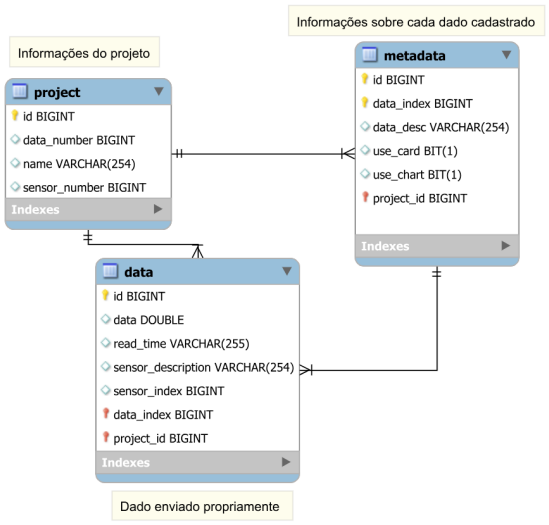


Fig. 4. Modelo ER do banco de dados

As três entidades, representadas são:

- **Projeto (project)**: Entidade que guarda as informações do projeto. Cada projeto pode ser relacionar com infinitas entidades metadados e infinitas entidades dados. Mas essas duas últimas só podem pertencer à um único projeto. O projeto é composto por: identificador (id), nome o projeto (name), número de dados (data_number) e número de dispositivos integrados (sensor_number).
- **Metadado (metadata)**: Entidade que reúne as informações a respeito de cada dado. Os metadados podem ser associados a infinitos dados, mas cada dado só pode ter um metadado. Um metadado é composto por: identificador (id), projeto de referência (project_id), índice do dado (data_index), nome da grandeza (data_desc), e tipo de exibição - numérica (use_card), gráfica (use_chart) ou ambas.
- **Dado (data)**: Unidade de dado coletado propriamente. Possui um identificador (id), armazena cada dado coletado em uma variável *double* (data), guarda informações textuais em uma variável do tipo *string* (sensor_description), está associado a seu metadado específico (data_index), a um projeto específico (project_id), contém o instante em que o dado foi recebido (read_time), o índice do dispositivo que a enviou (sensor_index) e um valor de descrição adicional para o dado (sensor_description).

Essa lógica pode ser abstraída para conjuntos numéricos em um diagrama de Venn. A Fig. 5 ilustra esse diagrama.

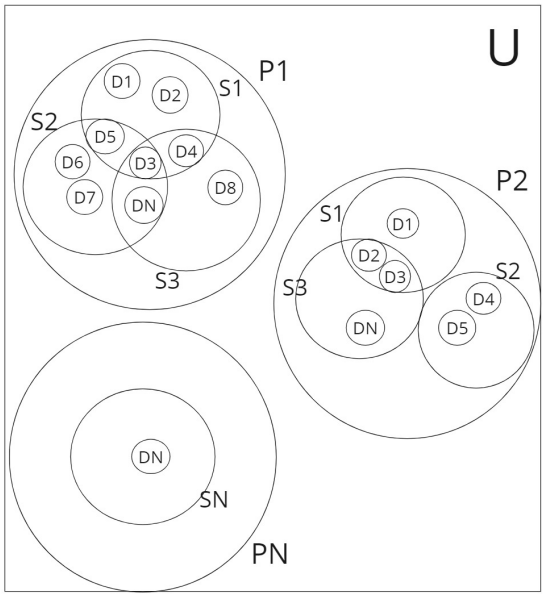


Fig. 5. Diagrama de Veen para a estrutura de dados do projeto.

No diagrama de Veen exposto na Fig. 5, U representa o conjunto de todos os dados enviados ao sistema, P1, P2, PN representam os dados correspondentes dos projetos de 1 a N (quantidade total de projetos). S1, S2 e SN representam os dispositivos conectados em cada projeto, e os conjuntos D1, D2 ... DN representam os conjuntos de entidades Dado dento de cada projeto, sendo que esses podem ou não ser intersecção de entidades correspondentes a dispositivos específicos. Por exemplo: O conjunto de entidades Dado D3 do P1 é composto pela união dos três dispositivos (S1, S3 e S3) dentro do projeto, ou seja: esses três dispositivos podem enviar essa entidade Dado. Já o projeto P2, possui o conjunto de entidade Dado proveniente do dispositivo S2 (conjuntos D4 e D5) que não intercede com os demais conjuntos de entidade Dado dos outros dispositivos. Ou seja, S2 poderia representar um sensor em específico que só envia os dados correspondentes as entidades Dado D4 e D5, que não são enviados pelos demais sensores (S1 e S3) dentro do projeto.

Dessa forma, as entidades Projeto, Dispositivo e Dado

são inter-relacionadas na API para entregar as informações solicitadas, que pode ser tanto por meio de um serviço *web* de telas (interface humana) como um *endpoint* na rede (interface máquina). O desenvolvimento dessas duas interfaces é detalhado posteriormente nessa seção. A aquisição dos dados por meio dos dois protocolos suportados é detalhada na subseção a seguir.

B. Protocolo interno para suporte MQTT e HTTP

A lógica de construtores para envio de dados se baseia em envios de entidades Dado (cada uma transporta um valor coletado para ser salvo) agrupadas em pacotes (listas de dados) ou individualmente para a API. No caso do protocolo HTTP é enviado um pacote de entidades Dado estruturado em *JSON*, diretamente para um *endpoint* específico da API que consome listas de entidade Dados, via requisição HTTP POST. A Fig. 6 ilustra esse fluxo.

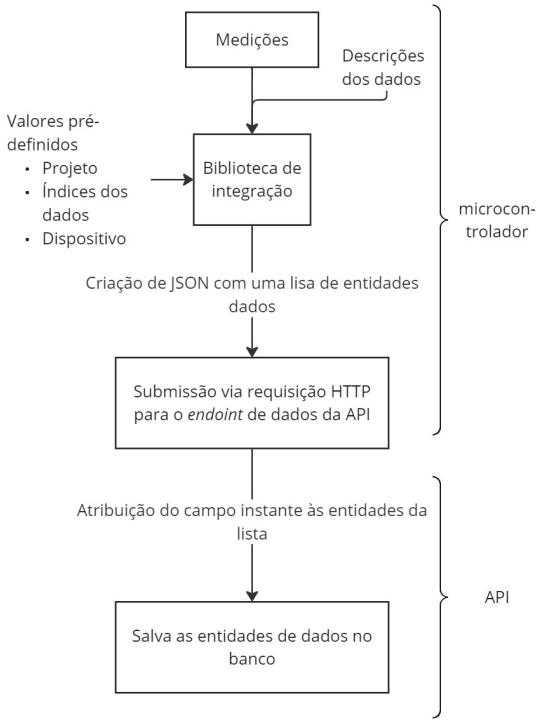


Fig. 6. Fluxo de envio de dados via procolo HTTP.

A biblioteca de integração do microcontrolador recebe a leitura da medição, associa a descrição definida pelo usuário no código embarcado e então cria uma *string* JSON que representa a entidade Dato, com todos os seus atributos. Essa *string* é concatenada com as demais entidades Dado para formar uma lista de Dado, é adicionada ao corpo da requisição e então submetida para a API.

A Fig. 7 ilustra o fluxo de envio de dados via protocolo MQTT.

No caso do procolo MQTT, a biblioteca de integração concatena um pacote de entidades Dados em uma *string* e o envia via mensagem MQTT para o tópico MQTT criado pela

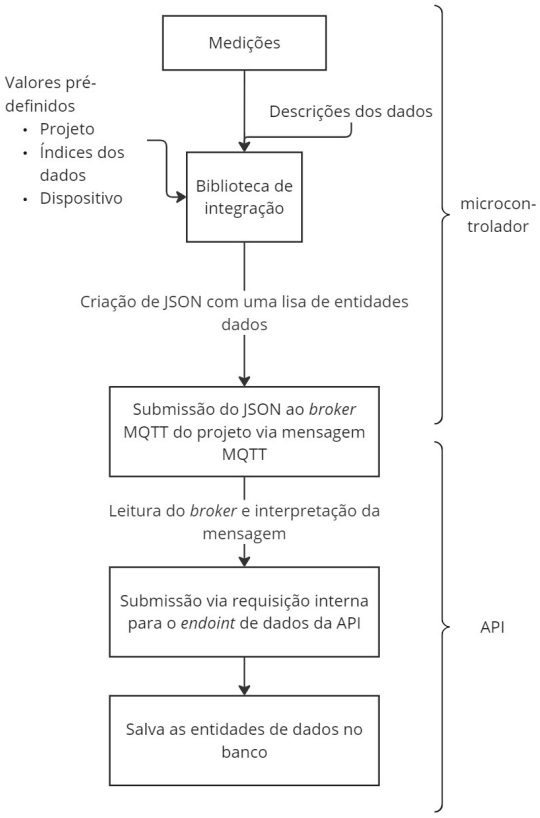


Fig. 7. Fluxo de envio de dados via procolo MQTT.

API, a qual possui um método para interpretar essa mensagem, transformar em uma lista de entidades Dados em *JSON* e encaminhar para o fluxo do *endpoint* que recebe listas de dados mencionado anteriormente.

Uma vez que cada entidade Dado empacotada em uma lista ou não, via qualquer um dos dois protocolos, carrega consigo identificadores de projeto, dispositivo e índice de dados a API consegue endereçar corretamente requisições simultâneas para múltiplos projetos, dispositivos, protocolos e dados. Trechos de código comas funções de salvamento dos dados nesses dois protocolos são exibidos no Apêndice I.

C. Interface visual e Interface máquina

Além de permitir o salvamento de dados via múltiplos protocolos, a API possui uma interface *web* embarcada (interface humana) e uma interface de *enpoints* HTTP estruturada sob uma lógica de caminhos que podem ser acessados por outros serviços na rede (interface máquina).

A interface visual possui as seguintes telas:

- Tela Inicial
- Criar Projeto
- *Dashboard*

A Tela Inicial dá acesso a tela de Criar Projeto e a tela de *Dashboard*. A primeira consiste em um painel de

criação de projetos, onde o usuário entra com as informações do novo projeto, como nome do projeto IoT, quantidade de dados, grandezas de cada dado, microcontroladores conectados e protocolo etc. Uma vez que o usuário entra com todas as informações necessárias e clica no botão criar projeto, a tela submete para a API uma entidade Projeto (formada pelos dados fornecidos) e uma lista de entidades Metadados correspondente a parametrização dos dados definida pelo usuário.

Como resposta, são criadas no banco de dados da plataforma a entidade do Projeto, as entidades de Metadados, é gerado o microcódigo do sistema embarcado para o microcontrolador escolhido e aberta uma nova guia com o mesmo. Além disso, o *dashboard* para o projeto é criado e passa a estar disponível para ser acessado. A Fig. 8 ilustra esse fluxo.

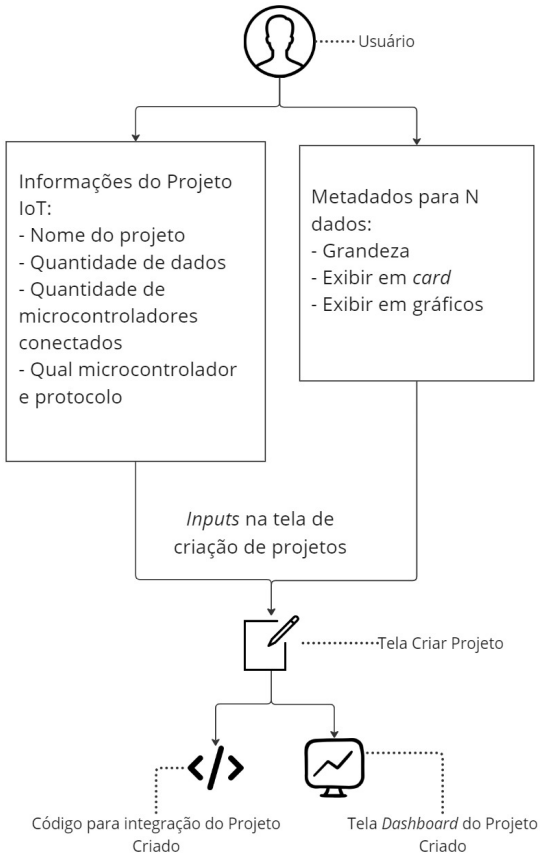


Fig. 8. Fluxo do uso da interface visual.

Na Fig. 8 é representado o fluxo onde o usuário entra com os dados do projeto e metadados, como resposta tem o microcódigo gerado e o *dashboard* disponível. A partir desse ponto já estão formadas as entidades de metadados e a do projeto IoT e a API já está pronta para receber os dados. No Apêndice I, são exibidos trechos de código que retornam os dados para serem exibidos na interface.

Já a interface máquina foi estruturada sob uma lógica de *endpoints* para aquisição dos dados que respeita o caminho geral apresentado na Fig. 9.

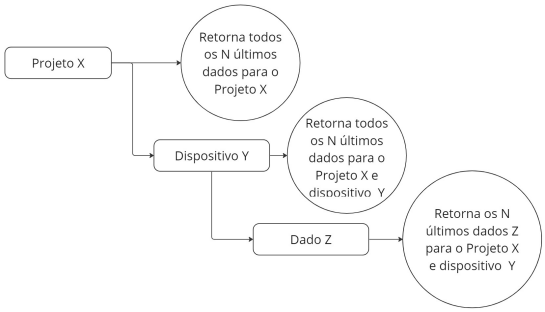


Fig. 9. Exemplo de fluxo completo da interface máquina para a leitura de dados.

O caminho apresentado na Fig. 9 retornaria para o cliente todos os últimos N dados Z enviados pelo dispositivo Y para o projeto X. Além disso, a API também responde a caminhos intermediários, como por exemplo o exposto na Fig. 10.

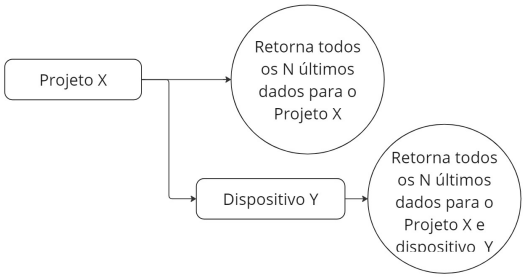


Fig. 10. Fluxo intermediário da interface máquina.

Esse caminho retorna todos últimos N dados, independentemente de seu índice, enviados pelo dispositivo Y ao projeto X.

Ou seja, a interface máquina permite: Obter todos os dados para um projeto, sem distinção de dispositivos de envio ou dado; obter os dados para um projeto enviados por um dispositivo em específico; ou então obter os dados para um projeto, dispositivo e índice de dado especifico.

V. IMPLEMENTAÇÃO EM RASPBERRY PI 3

A plataforma desenvolvida conforme descrito na seção anterior foi embarcado no *hardware* de baixo custo Raspberry Pi 3, o qual atua como servidor local da aplicação. O sistema operacional escolhido para usar no Raspberry Pi 3 foi o Ubuntu Server 23.10, devido a esse sistema ser otimizado para uso em servidores (não possuí interface visual e consome poucos recursos de *hardware*). Além disso, para que a plataforma funcionasse foram instalados os seguintes recursos: Java 21, MySQL Server e Mosquitto MQTT Broker [34]. Essas ferramentas são denominadas *stack* (pilha) da platafroma e são fundamentais para seu correto funcionamento.

A Fig. 11 ilustra o fluxo completo de envio de dados considerando os elementos de hardware e seus serviços:

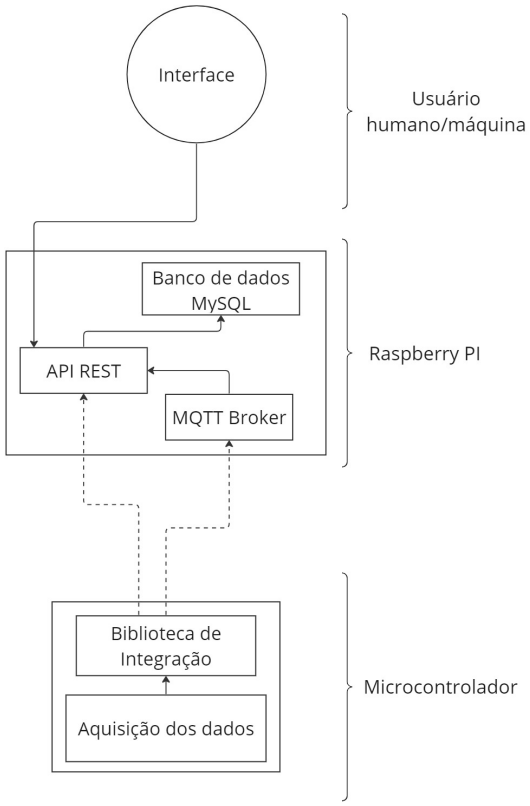


Fig. 11. *Stack* completa embarcada no Raspberry e suas integrações.

A. Metodologia de Testes de Performance

Para testar tanto a performance da plataforma em si, quando a capacidade do *hardware* em sustentar a API e sua *stack*, foram realizados testes de carga de salvamento de dados - simulando microcontroladores enviando dados (requisições do tipo HTTP POST e envio de *messages* via MQTT), sendo que a todo momento o consumo de recursos de *hardware* do Raspberry Pi 3 foi monitorado através da ferramenta Glances (ferramenta do Ubuntu Server para visualizar processos e consumo dos recursos do sistema em tempo real) [35].

Como sugere a lógica de estrutura e salvamento dos dados apresentada na seção anterior, na perspectiva da API os dados sempre chegarão em listas de entidades dados via requisição, para o mesmo *endpoint*. Ou seja, nesse fluxo são irrelevantes quais dispositivos e projetos de destino dessas requisições, de forma que o único fator real de impacto é a carga bruta delas.

Nesse sentido, para testar o envio de dados via MQTT foi desenvolvido um *script shell*, executado a partir de um computador pessoal com Windows (na mesma rede que o Raspberry), que se conecta ao *broker* MQTT da API e executa um *loop* de envios de dados para a plataforma com 1000 iterações. Ou seja, o fluxo normal que um microcontrolador faria é respeitado, porém o *script* potencializa um ciclo de envio de dados sequenciais controlado e de grande volume. Já para testar as requisições do tipo HTTP foi utilizada a mesma estrutura de *script* porém fazendo requisições HTTP POST via curl (ferramenta do *prompt* de comando do Windows que atua como *client* de requisições). Dessa forma, esse *script* conecta-se com a API e realiza várias requisições em *loop* via HTTP POST. Esses *scrips* de testes são exibidos no Apêndice II.

Os testes foram executados com o Raspberry Pi 3 sustentando a *stack* completa da plataforma, executando de uma a quinze iterações simultâneas de envios de dados e os resultados obtidos (como tempo das requisições, perda de dados ou estresse do *hardware*) são apresentados na seção a seguir.

VI. RESULTADOS E DISCUSSÕES

Nessa seção serão apresentados os resultados com a plataforma e sua *stack* implementada e funcional no *hardware* proposto. Na primeira subseção são apresentados e discutidos os resultados dos testes de performance, conforme descritos na seção anterior. Na segunda subseção foi feita uma avaliação da interface humana por meio da Heurística de Nielsen. Por fim, foi documentada a migração de um projeto IoT real para a plataforma.

A. Avalização da performance da plataforma embarcada em Raspberry Pi 3

O Raspberry Pi 3 usado nos testes possuí o *hardware* detalhado na tabela I.

TABELA I
Hardware do Raspberry Pi 3

Característica	Descrição
Modelo	Pi 3 Model B rev 1.2
CPU	4 x 1.2GHz Broadcom BCM2837 64-bit
RAM	1GB RAM
ROM	Micro SD 16GB SanDisk Ultra 80MB/s
Modem (na placa)	BCM43438 wireless LAN and (BLE)

Os testes, descritos conforme a metodologia da seção anterior, foram realizados tanto para requisições via protocolo MQTT quanto HTTP. Ou seja: o *script* que dispara 1000 requisições sequenciais foi executado de forma singular e até quinze vezes simultâneas, conforme o eixo horizontal do gráfico das Fig. 12, Fig. 14 e Fig. 13. A Fig. 12 representa a curva do tempo de resposta de salvamento dos dados para os protocolos HTTP e MQTT.

A curva da Fig. 12 mostra que o tempo de resposta é maior nos menores números de execuções paralelas e se estabiliza em quantidades maiores de requisições paralelas (acima de 5 x 1000) . Isso sugere que a própria metodologia de testes nivelou por baixo inicialmente (devido aos próprios limites do tempo de requisição do cliente), passando a atingir propriamente o limite da API a partir de 5 execuções simultâneas do *script*. De qualquer forma, o tempo médio de requisições nesse estado (nivelando pelos limites da API) foi de 28 ms para o protocolo HTTP e de 21 ms para o protocolo MQTT.

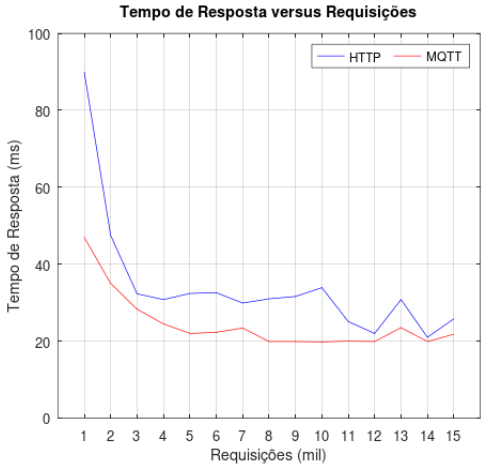


Fig. 12. Tempo de resposta das requisições pra salvar dados em ambos os protocolos, quanto menor melhor.

A título de comparação, em [22] foi desenvolvida e testada uma API REST MQTT onde para testes de carga foi considerando de 1 a 125 dispositivos virtuais conectados, cada um enviando uma mensagem MQTT por segundo. Como resultado obteve-se tempos de resposta médio de 315 ms para salvamento de dados (o melhor tempo foi da ordem de 125 ms). Ou seja, esses resultados sugerem que a arquitetura desse sistema é menos eficiente do que a proposta nessa pesquisa. Talvez isso se justifique pela estrutura que [22] arquitetou seu sistema: cada projeto IoT nele tem um tópico MQTT especifico. Já nessa pesquisa a API usa apenas um único tópico MQTT para todos os projetos e então interpreta suas mensagens para saber de qual projeto correspondem os dados enviados. Além disso, o sistema de [22] foi embarcado em um dispositivo Windows enquanto a plataforma proposta nessa pesquisa executa toda a sua *stack* em Ubuntu Server (sistema mais propício para aplicações em servidor).

Já o uso de memória RAM pela API é exibida na Fig. 13.

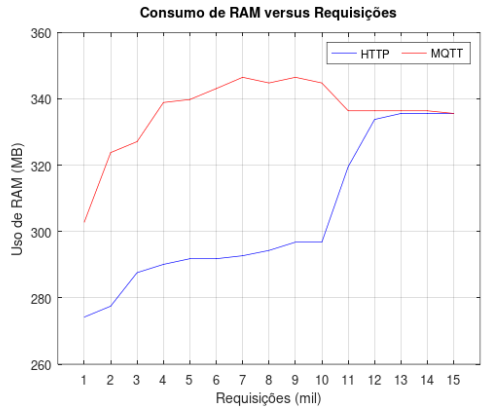


Fig. 13. Uso de RAM pela API.

Na Fig. 13 é notável que o uso do protocolo MQTT acarreta um maior consumo de memória RAM por parte da API, o que é esperado já que esse protocolo apresentou melhor performance de tempo de resposta, ou seja, fez a API trabalhar mais. Além disso, o consumo se estabiliza ao longo dos ciclos em decorrência dos mecanismos de gerenciamento de memória da máquina virtual Java e o seu processo de ciclos chamado *garbage collector*, que procura objetos que não estão mais sendo utilizados, mas que ainda ocupam a memória do dispositivo computacional. Isso é uma hipótese para justificar o aumento abrupto no consumo de memória para os testes com HTTP, pois o gerenciador de memória pode ter executado a limpeza momentos antes de necessitar novamente dos recursos. O consumo de CPU também foi monitorado e é apresentado na Fig. 14.

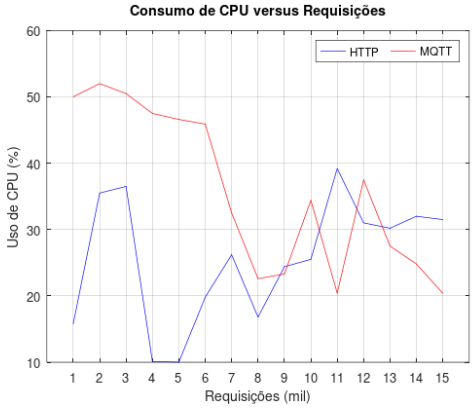


Fig. 14. Uso de CPU pela API.

Na curva da Fig. 14 percebe-se que o consumo de CPU foi maior nos primeiros ciclos de testes e depois reduziu-se e passou a ter comportamento arbitrário. Isso se deve a natureza entrópica da própria distribuição de recursos do sistema operacional, a qual leve em consideração fatores variados, como temperatura da CPU, processos e rotinas em segundo plano etc. Por exemplo, a temperatura inicial da CPU para ambos os testes foi na faixa de 40° (de acordo com a leitura do Glances), porém toda vez que ela atingia cerca de 65° o consumo era reduzido pelo sistema para evitar sobreaquecimento.

De qualquer forma, a tabela II traz um resumo com os principais dados obtidos para cada protocolo.

TABELA II
Resumo dos testes para cada protocolo

Teste	HTTP	MQTT
Tempo de resposta (ms)	28	21
Memória RAM usada (MB)	303,59	335,95
CPU usada (%)	25,63	35,73

Durante todos os testes apresentados constatou-se que não houve perda de dados nem quedas da API ou dos serviços de sua *stack*.

B. Análise da IHC (Interface Humano Computador)

A seguir a interface humana (*web*) desenvolvida para a plataforma, abreviada como IHC, será analisada sob a metodologia Heurísticas de Nielsen [36], elaborada por Jakob Nielsen em 1994, a qual prescreve que para a interface ser usável e de boa qualidade, a mesma deve respeitar as seguintes regas de UX (*user experience*): Visibilidade do status do sistema; compatibilidade entre o sistema e o mundo real; controle e liberdade para a pessoa usuária; consistência e padronização; prevenção de erros; reconhecimento em vez de memorização; eficiência e flexibilidade de uso; estética e design minimalista; auto auxilio ao usuário para reconhecer, diagnosticar e recuperar-se de erros; ajuda e documentação. O Apêndice III apresenta diversas capturas de tela da interface da plataforma e todas as regras da heurística são exploradas abaixo no contexto da interface elaborada:

- **Visibilidade do *status* do sistema:** em todas as situações em que a tela faz requisições lentas é exibida uma animação de *loading* (carregamento), de forma que o usuário esteja ciente de que a plataforma não travou e seu *status* atual é: aguardando. Além disso, são exibidas mensagens claras ao finalizar processos (como criar projetos ou deletá-los). Como melhoria futura, poderia-se implementar uma barra de progresso animada.
- **Compatibilidade entre o sistema e o mundo real:** A interface sempre exibe informações de forma direta e em linguagem comum para o público alvo (projetista de IoT ou usuário de um projeto). Além disso, os *feedbacks* ao realizar ações são consistentes, como animações que indicam que as ações foram reconhecidas. A interface também respeita padrões de design comuns em outros produtos ou sistemas. Por exemplo, a posição do logotipo no canto superior esquerdo da tela, a estrutura de um menu de navegação e a aparência dos botões de conformação (*checklist*).
- **Controle e liberdade para a pessoa usuária:** A interface possui botões para retornar, entrar em projetos e navegar para a tela inicial. Além disso, o usuário tem liberdade absoluta para criar seu *dasbhoard* com quantos *cards* de dados e dados exibidos em gráficos quiser. Como melhoria futura, poderia-se incrementar mais formas de visualização dos dados na tela, como *gauges* e mais formatos de gráficos.
- **Prevenção de erros:** A própria interface possuí restrições e impõe testes as entradas dos usuários a fim de prevenir ações que dispariam erros na API, por exemplo: não é permito criar projetos IoT sem nome nem definir como nula a quantidade de dados a ser exibida.
- **Reconhecimento em vez de memorização:** O *dashboard* é estruturado de forma lógica e intuitiva, agrupando os *cards* na tela seguindo uma ordem coerente e em seguida exibindo o gráfico expandido. Além disso, a interface faz uso de ícones significativos na tela inicial (para as ações de criar, gerenciar e acessar *dashboard*) e rótulos claros em todas as suas telas.
- **Eficiência e flexibilidade de uso:** É possível acessar um *dashbaord* de projeto com apenas dois cliques e voltar para a tela inicial com apenas um, o que garante um uso eficiente. Graças a adoção do *framework* visual Bootstrap, a interface da plataforma se adapta a diferentes dispositivos, como *smartphones*, tablets e computadores. Além disso, acompanha o perfil de cor do usuário ativando o modo escuro se o dispositivo de acesso assim estiver.
- **Estética e design minimalista:** Os elementos são dispostos na tela de forma centralizada e sem informações desnecessárias, e são utilizados de ícones e hierarquia visual clara. Além disso, a interface possuí responsividade para diferentes padrões de tela conforme já mencionado.
- **Auto auxilio ao usuário para reconhecer, diagnosticar e recuperar-se de erros:** Caso o usuário tente acessar a janela de gerenciamento com um login incorreto, a interface dispara uma mensagem relatando esse erro. Além disso, caso o navegador do usuário bloqueie a guia de microcódigo gerado, a plataforma possibilita gerá-lo novamente.
- **Ajuda e documentação:** A plataforma conta com documentação própria completa, e possuí atalho para seu acesso através dela mesmo.

Essa análise atesta que a interface *web* desenvolvida no projeto está de acordo com todas as eurísticas de Nilsen, ou seja, como resultado tem-se uma IHC intuitiva e amigável. Entretanto essa interface é passível de melhoras, como: criação de mais formatos de visualizações para os dados e desenvolvimento de animações mais elaboradas, como barras de progresso por exemplo.

C. Uso em Projetos IoT

A Plataforma IoT desenvolvida implementada no *hardaware* Raspberry Pi com sua *stack* para Ubuntu Server conforme descrito, e testada na rede interna do laboratório de pesquisa do IFSC Campus Rau. A seguir, foi criado um projeto IoT de testes para validar o funcionamento nesse ambiente. Nesse projeto foi implementada a geração aleatória de dados em dois microcontroladores: um ESP8266 e um ESP32, ambos testados com os protocolos HTTP e MQTT. Verificou-se que o projeto IoT de testes criado na plataforma funcionou corretamente, salvando e exibindo dados sem perdas.

Em seguida, o projeto IoT real desenvolvido em [25], que estava com sua estrutura e microcontrolador disponíveis no laboratório de pesquisa do ISFC Rau, foi migrado para esta nova plataforma. Esse projeto IoT tem as seguintes características: usa de dois sensores analógicos de luminosidade, sensores digitais para captar temperatura, umidade e realiza a medição de tensão e corrente para mensurar a potência e energia geradas. Para tal, faz uso do microcontrolador ESP8266. Do ponto de vista de sistema, esse projeto usava o sistema IoT proveniente de

readaptações que comportava mais dois projetos IoT distintos, conforme já descrito na seção de Revisão Bibliográfica.

A titulo de comparação, na Tabela III é exposta uma comparação entre os recursos presentes no sistema antigo referido e esta nova plataforma (todos os recursos listados foram descritos detalhadamente nas seções anteriores).

TABELA III
Comparação entre o Sistema Antigo e a Nova Plataforma

Característica	Antigo	Novo
Interface guiada	Não	Sim
Geração de microcódigo	Não	Sim
Gráficos e *cards*	Sim	Sim
Geração de relatórios	Sim	Sim
Interface responsiva	Não	Sim
Protocolo HTTP	Sim	Sim
Protocolo MQTT	Não	Sim

A Tabela III indica que ao migrar do sistema antigo para a nova plataforma, obtém-se como benefício um microcódigo próprio, uma interface mais responsiva e suporte ao protocolo MQTT. Para criar esse projeto dentro da plataforma seguiu-se o fluxo representado na Fig. 8 e detalhado no passo a passo a seguir.

1. Acesso a tela de criação de projetos através da interface *web*.
2. Entrada com os parâmetros do Projeto: Nome (”Solar Tracker”), quantidade de dados (seis), número de microcontroladores conectados (um) e protocolo (MQTT).
3. A seguir, inseriu-se os metadados para os dados: nome de cada grandeza (”Luminosidade média [lux]”, ”Temperatura (ºC)”, ”Potência [W]”, ”Energia Gerada [Wh]”, ”Tensão [V]”, ”Corrente [A]”), e formato de exibição (*cards* e gráficos para todos os dados).
4. Por fim, a plataforma gerou o *dashboard* e o microcódigo para o ESP8266. Esse código foi complementado para suportar e leitura dos sensores e então transferido para o microcontrolador.

Foi testado o funcionamento de sua integração e constatou-se que a plataforma funcionou como o esperado, sem perda de dados e sincronizando os dados coletados em tempo real. Figuras da interface contendo *cards*, gráficos e a visão de relatórios para esse ensaio são exibidas no Apêndice IV.

VII. CONCLUSÃO

Dado os resultados obtidos pode-se concluir que o objetivo de construir uma plataforma informatizada para projetos IoT com interface de fácil uso para automatizar a integração de projetos IoT a banco de dados, interface humana (*dashboard* com gráficos e *cards* moldáveis para cada projeto), interface máquina e suporte a múltiplos protocolos foi realizado com sucesso. Essa plataforma foi devidamente implementada na rede local do laboratório de pesquisa do IFSC Campus Rau e foi elaborada e publicada sua documentação [37]. Dessa forma, a plataforma desenvolvida poderá não só ser utilizado por demais discentes no ambiente do laboratório e em outros locais, mas também pode receber novos desenvolvimentos e projetos futuros. Além disso, os resultados apontam que o uso de protocolo MQTT proporcionou um melhor tempo de resposta, porém consumiu mais recursos do servidor de forma oposta ao uso do protocolo HTTP (foi mais lento porém exigiu menos recursos do servidor). Esses resultados condizem com as conclusões de [21]: o protocolo MQTT é mais performático porém o HTTP é mais leve na parte do servidor. O fato de o MQTT consumir mais recursos da API se deve a esta necessitar ferramentas específicas para receber e interpretar as mensagens MQTT (como mencionado na seção IV), enquanto o procolo HTTP faz um fluxo mais direto. Esse resultado sugere que o protocolo ideal depende do objetivo do projeto em específico: se o foco é respostas rápidas o procolo MQTT é melhor, porém se isso não for crucial, mas sim o volume de dados, usar o protocolo HTTP é mais vantajoso por economizar mais recursos do servidor. Possivelmente, no lado do microcontrolador o uso de MQTT acarretará em menos consumo de recursos, conforme sugerem [21] e [38]. Porém esses testes não foram realizados nessa pesquisa.

Um caminho de continuidade deste trabalho seria mensurar os impactos de performance (e consumo energético, nesse caso) de cada protocolo no *hardware* do microcontrolador comunicando com essa API para então comparar com os impactos no lado do servidor expostos nessa pesquisa. Poderia-se também monitorar o consumo da rede e da temperatura no *hardware* do servidor para compará-los a nível de largura de banda e obter uma análise mais avançada do estresse no hardware (inclusive, realizando múltiplas rodadas de testes). Em relação a melhorias de interface e usabilidade, poderia-se citar: nova forma de governança de uso da plataforma, com gerenciamento mais avançado de usuários, desenvolvimento de novas formas de visualização de dados além de gráficos e *cards* etc. Por fim, a interface máquina da plataforma pode ser amplamente explorada em projetos relacionados a controle a automação, de forma que o uso da API proposta nessa pesquisa não ficaria limitada apenas para aplicações de telemetria.

A maior dificuldade na execução dessa pesquisa foi em encontrar metodologias de testes da plataforma que atestassem seu funcionamento e gerassem resultados sólidos tanto a nível de testes de uso quanto performance.

REFERÊNCIAS

[1] A. McEwen, H. Cassimally, *Designing the Internet of Things*, 1a ed., John Wiley & Sons, New Jersey, EUA, 2013.

[2] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, M. Gidlund, *Industrial Internet of Things: Challenges,*

Opportunities, and Directions, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, 2018.

[3] S. Li, L. D. Xu, S. Zhao, *The internet of things: a survey*, Springer Science+Business Media Magazine, 2014.

[4] N. S. Yamanoor, S. Yamanoor, *High Quality, Low Cost Education with the Raspberry Pi*, IEEE Global Humanitarian Technology Conference (GHTC), 2017.

[5] "Curso Raspberry PI Básico para IoT", https://estude.ifrs.edu.br/cursos/raspberry-pi-basico-para-iot/, acessado em: 20 de julho de 2024.

[6] G. Marques, R. Pitarma, *An Internet of Things-Based Environmental Quality Management System to Supervise the Indoor Laboratory Conditions*, Applied Sciences Journal, 2018.

[7] M. Benammar, A. Abdaoui, S. H. Ahmad, F. Touati, A. Kadri, *A Modular IoT Platform for Real-Time Indoor Air Quality Monitoring*, Sensors Journal, 2018.

[8] "Google Cloud IoT Platform: Plataforma IoT da Google", https://cloud.google.com/iot-core?hl=pt-br, acessado em: 09 de maio de 2024.

[9] "AWS IoT", https://aws.amazon.com/pt/iot/, acessado em: 09 de maio de 2024.

[10] "Calculadora de custos AWS", https://calculator.aws//addService, acessado em: 09 de maio de 2024.

[11] "ThingWorx IIoT Platform", https://www.ptc.com/en/products/thingworx/, acessado em: 09 de maio de 2024.

[12] "MQTT Client", https://www.ptc.com/en/store/kepware/drivers/mqtt-client, acessado em: 09 de maio de 2024.

[13] "Azure IoT", https://azure.microsoft.com/en-us/solutions/iot/solution-overview, acessado em: 09 de maio de 2024.

[14] "Internet of Things (IoT)", https://www.oracle.com/internet-of-things/, acessado em: 09 de maio de 2024.

[15] "WEGnology", https://www.weg.net/institutional/BR/pt/digital-solutions/solutions/wegnology, acessado em: 09 de maio de 2024.

[16] "Blink", https://blynk.io/blynk-iot-low-code-software-platform, acessado em: 10 de maio de 2024.

[17] "Arduinon Cloud", https://cloud.arduino.cc/, acessado em: 10 de maio de 2024.

[18] M. Senožetnik, Z. Herga, T. Šubic, L. Bradeško, K. Kenda, K. Klemen, P. Pergar, D. Mladenić, *IoT Middleware for Water Management*, Proceedings Journal, 2018.

[19] F. Oliveira, D. G. Costa, L. Lima, I. Silva, *iBikeSafe: A Multi-Parameter System for Monitoring, Evaluation and Visualization of Cycling Paths in Smart Cities Targeted at Cycling Adverse Conditions*, Smart Cities Journal, 2021.

[20] "About", https://nodered.org/about/, acessado em: 06 de junho de 2024.

[21] J. Dizdarević, F. Carpio, A. Jukan, X. Masip-Bruin, *A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration*, ACM Computing Surveys, 2019.

[22] Y. Y. F. Panduman, N. Funabiki, P. Puspitaningayu, M. Kuribayashi, S. Sukaridhoto, W.-C. Kao, *Design and Implementation of SEMAR IoT Server Platform with Applications*, Sensors Journal, 2022.

[23] L. Vieira, P. D. da Silva, *Medidor IOT de deformação de bandagens de rotores de máquinas elétricas de grande porte usando extensômetros*, IFSC, 2022.

[24] P. D. da Silva, Y. M. Scheuer, *Medidor de Energia Inteligente: Aplicação em um Gradador de Potência*, SICT-Sul, 2020.

[25] M. W. Sprotte, G. H. Tavares, V. Alexandre, M. A. Salvador, *Sistema IoT de Controle de Solar Tracker, com Monitoramento de Energia, Luminosidade e Temperatura*, SEPETEC 2023, 2022.

[26] A. B. M. e Jardel Vieira, *Desenvolvimento de um Sistema de Telemetria utilizando conceitos de IOT*, XXXIV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2016.

[27] J. R. E. Leite, P. S. Martins, E. L. Ursini, *A INTERNET das COISAS (IoT): Tecnologias e Aplicações*, Brazilian Technology Symposium, 2017.

[28] L. Atzori, A. Iera, G. Morabito, *The internet of things: A survey*, Computer networks, vol. 54, no. 15, pp. 2787–2805, 2010, 2010.

[29] "What is IoT Gateway?", https://engineerscommunity.com/t/what-is-iot-gateway/6861, acessado em: 10 de junho de 2024.

[30] "Introduction to HTTP", https://www.w3schools.in/http/intro, acessado em: 10 de maio de 2024.

[31] "Introduction to MQTT", https://vasanza.blogspot.com/2021 de julho de esp32-mqtt-introduccion.html, acessado em: 10 de maio de 2024.

[32] "SOAP vs. REST: Differences in Performance, APIs, and More", https://dzone.com/articles/differences-in-performance-apis-amp-more, acessado em: 10 de junho de 2024.

[33] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, UNIVERSITY OF CALIFORNIA, 2000.

[34] "Eclipse Mosquitto™", https://mosquitto.org/, acessado em: 20 de julho de 2024.

[35] "Glances", https://nicolargo.github.io/glances/, acessado em: 20 de julho de 2024.

[36] "10 Usability Heuristics for User Interface Design", https://www.nngroup.com/articles/ten-usability-heurstics/, acessado em: 02 de junho de 2024.

[37] "sensorsync", https://github.com/mwsprotte/sensorsync-docs, acessado em: 06 de junho de 2024.

[38] J. Joshi, V. Rajapriya, S. R. Rahul, P. Kumar, S. Polepally, R. Samineni, D. G. K. Tej, *Performance Enhancement and IoT Based Monitoring for Smart Home*, NIIT University, Rajasthan, India, 2017.

APÊNDICE I – TRECHOS DE CÓDIGO

A. *Função de salvamento de dados via protocolo HTTP*

```
/** Recebendo uma lista de dados para salvar no banco*/
@PostMapping(value = "/saveList", produces = MediaType.APPLICATION_JSON_VALUE,
consumes = MediaType.APPLICATION_JSON_VALUE)
public boolean saveList(@RequestBody List<Data> dataList) {
    try {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("HH:mm:ss
dd/MM/yyyy");
        for (Data data : dataList) {
            data.setReadTime(dtf.format(LocalDateTime.now()));
            dataServices.save(data);
        }
        return true;
    } catch (Exception e) {
        return false;
    }
}
```

B. *Função de salvamento de dados via protocolo MQTT*

```
public void messageArrived(String topic, MqttMessage message) throws IOException
{
                    ObjectMapper mapper = new ObjectMapper();
                    List<Data> dataList = new ArrayList<>();
                    try {
                        dataList = mapper.readValue(new String(message.getPay-
load()).replace("'", "\""), mapper.getTypeFactory().constructCollection-
Type(List.class, Data.class));
//                          logger.info("Requisicao MQTT recebida");
                        MQTTPost(dataList);
                    } catch (JsonProcessingException e) {
                        logger.info("Não foi possivel interpretar a requisicao
MQTT! Erro: " + e);
                    }
                }
```

C. *Função que retorna todos os dados de determinado projeto para gerar os cards da interface web*

```
/**RETORNA TODOS OS CARDS PARA UM PROJETO DE ACORDO COM O DISPOSITIVO ESCO-
LHIDO*/
 @GetMapping(value = "/cards/{projectID}/device/{device}", produces = Media-
Type.APPLICATION_JSON_VALUE)
 public List<CardView> findDataCardsByDevice(@PathVariable(value = "projectID")
Long id, @PathVariable(value = "device") Long device) {
     List<CardView> cards = new ArrayList<>();
     for (Long j = 0L; j < (projectService.findById(id).getDataNumber()); j++) {
         var entity = dataServices.findForCard(id, device, j);
         CardView dataToAdd = new CardView();
         dataToAdd.setData(entity.getData());
         dataToAdd.setReadTime(entity.getReadTime());
         dataToAdd.setSensorDescription(entity.getSensorDescription());
         dataToAdd.setSensorIndex(entity.getSensorIndex());
         dataToAdd.setDataDesc(metadataService.findByProjectIdAndDataIndex(id,
j).getDataDesc());
         cards.add(dataToAdd);
     }
     return cards;
 }
```

D. *Função que retorna todos os dados de determinado projeto e dispositivo para gerar os gráficos de acordo com o range definido*

```
    /**RETORNA OS DADOS PARA O GRÁFICO DE UM PROJETO DE ACORDO COM O DISPOSITIVO
ESCOLHIDO*/
    @GetMapping(value = "/charts/{projectID}/device/{device}/length/{length}",
produces = MediaType.APPLICATION_JSON_VALUE)
    public List<List<String>> findDataCharts(@PathVariable(value = "projectID")
Long id, @PathVariable(value = "device") Long device, @PathVariable(value =
"length") Long length) {
        List<List<String>> dataForCharts = new ArrayList<>();

//        Iterando para carregar os títulos das curvas
        List<String> labels = new ArrayList<>();
        var entityDesc = metadataService.findByProjectId(id);
        for (int i = 0; i < entityDesc.size(); i++) {
            if (entityDesc.get(i).isUseChart()) {
                labels.add(entityDesc.get(i).getDataDesc());
            }
        }

        dataForCharts.add(labels);

//      Iterando para carregar o eixo x
        List<String> labelsX = new ArrayList<>();
        var entityX = dataServices.findForChart(id, device, 0L, length);
        for (int i = 0; i < entityX.size(); i++) {
            labelsX.add(entityX.get(i).getReadTime());
        }
        Collections.reverse(labelsX);
        dataForCharts.add(labelsX);

//        Iterando para carregar os conjuntos de dados para cada grandeza
        int maxIndex = (int) projectService.findById(id).getDataNumber();
        for (int i = 0; i < maxIndex; i++) {
            List<String> labelsY = new ArrayList<>();
            var entityY = dataServices.findForChart(id, device, Long.valueOf(i),
length);
            for (int j = 0; j < entityY.size(); j++) {
                labelsY.add(String.valueOf(entityY.get(j).getData()));
            }
            Collections.reverse(labelsY);
            dataForCharts.add(labelsY.stream().toList());
        }
        return dataForCharts;
    }
```

APÊNDICE II – SCRIPTS DE TESTES

A. *Script* de teste HTTP

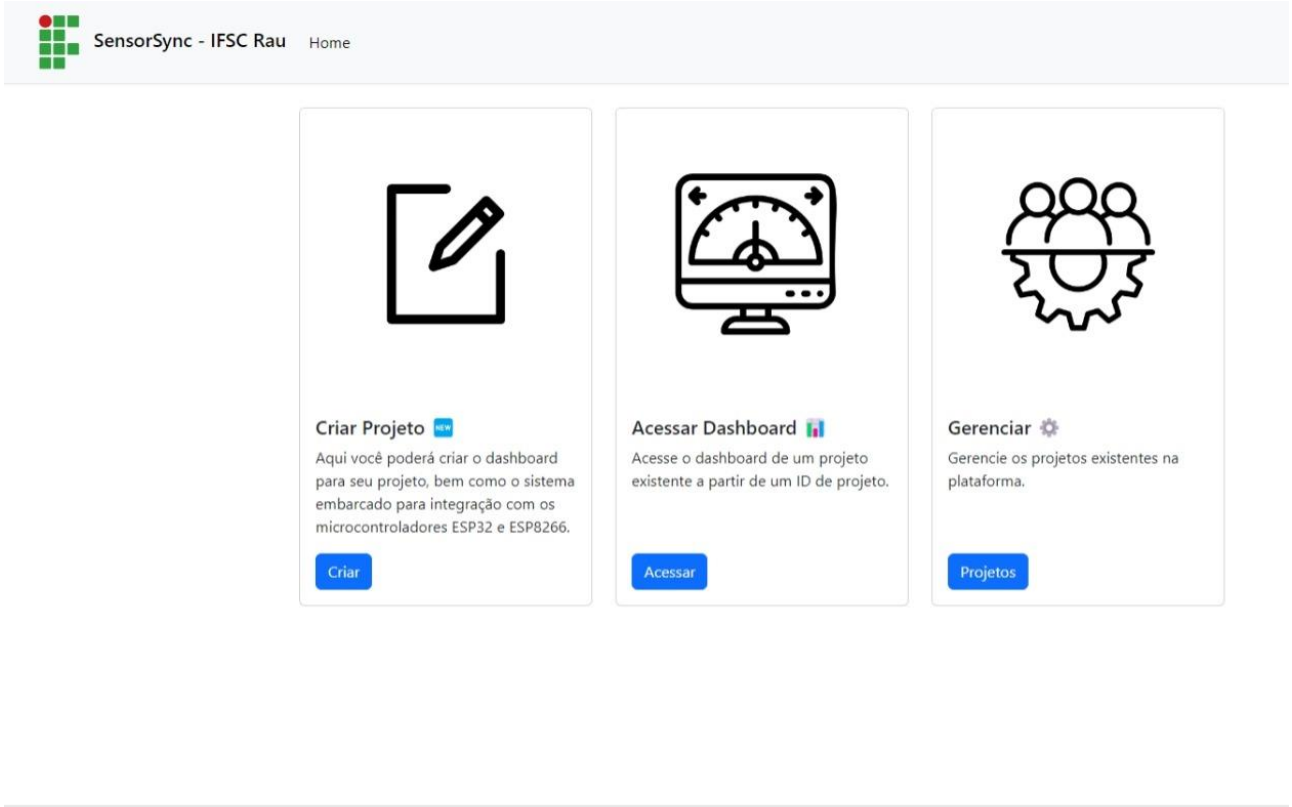
```
@ECHO OFF
SETLOCAL ENABLEDELAYEDEXPANSION

SET "MIN=1"
SET "MAX=1000"

FOR /L %%n IN (!MIN!,1,!MAX!) DO (
    SET /A "randomNum=!RANDOM! %% (!MAX! - !MIN! + 1) + !MIN!"
    ECHO Dado enviado na iteracao %%n: !randomNum!

    curl -X POST http://192.168.100.100:8080/data/saveList -H "Content-type:appli-
cation/json" -d "[{\"sensorDescription\": \"Teste 1 HTTP\",\"projectID\": 4,\"sen-
sorIndex\": 0,\"dataIndex\": 0,\"data\": "!randomNum!"}]"
)

ENDLOCAL
```

B. *Script* de teste MQTT

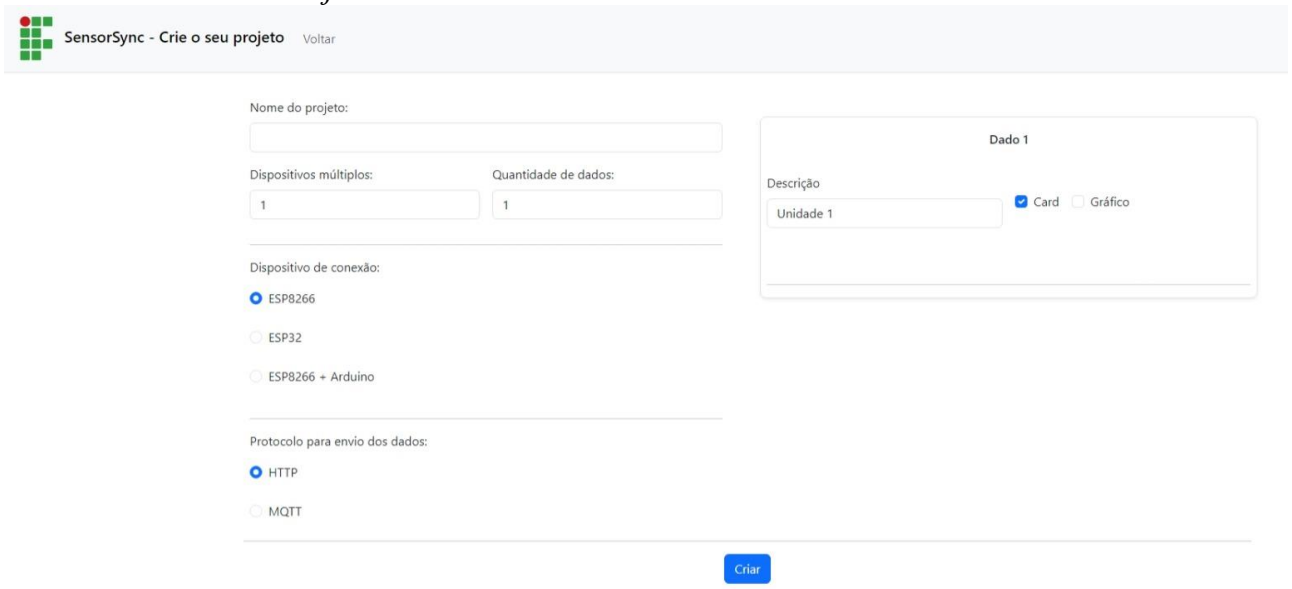
```
@ECHO OFF
SETLOCAL ENABLEDELAYEDEXPANSION

SET "MIN=1"
SET "MAX=1000"

FOR /L %%n IN (!MIN!,1,!MAX!) DO (
    SET /A "randomNum=!RANDOM! %% (!MAX! - !MIN! + 1) + !MIN!"
    ECHO Dado enviado na iteracao %%n: !randomNum!

    mosquitto_pub -h 192.168.100.100 -t sensorsync -m "[{'sensorDescription':'Teste
1 MQTT','projectID': 4,'sensorIndex': 0,'dataIndex': 0,'data':"!randomNum!"}]"
)

ENDLOCAL
```

APÊNCIDE III – CAPTURAS DE TELA DA INTERFACE

A. *Tela inicial do sistema*



B. *Tela de Criar Projeto do Sistema*



C. *Dashboard de dados para um projeto fictício (dois dados e um dispositivo)*

D. *Dashboard de dados para um projeto fictício (dois dados e um dispositivo) – em modo escuro*

E. Dashboard de dados para um projeto fictício (dois dados e um dispositivo) – Adaptado para tela de smartphone

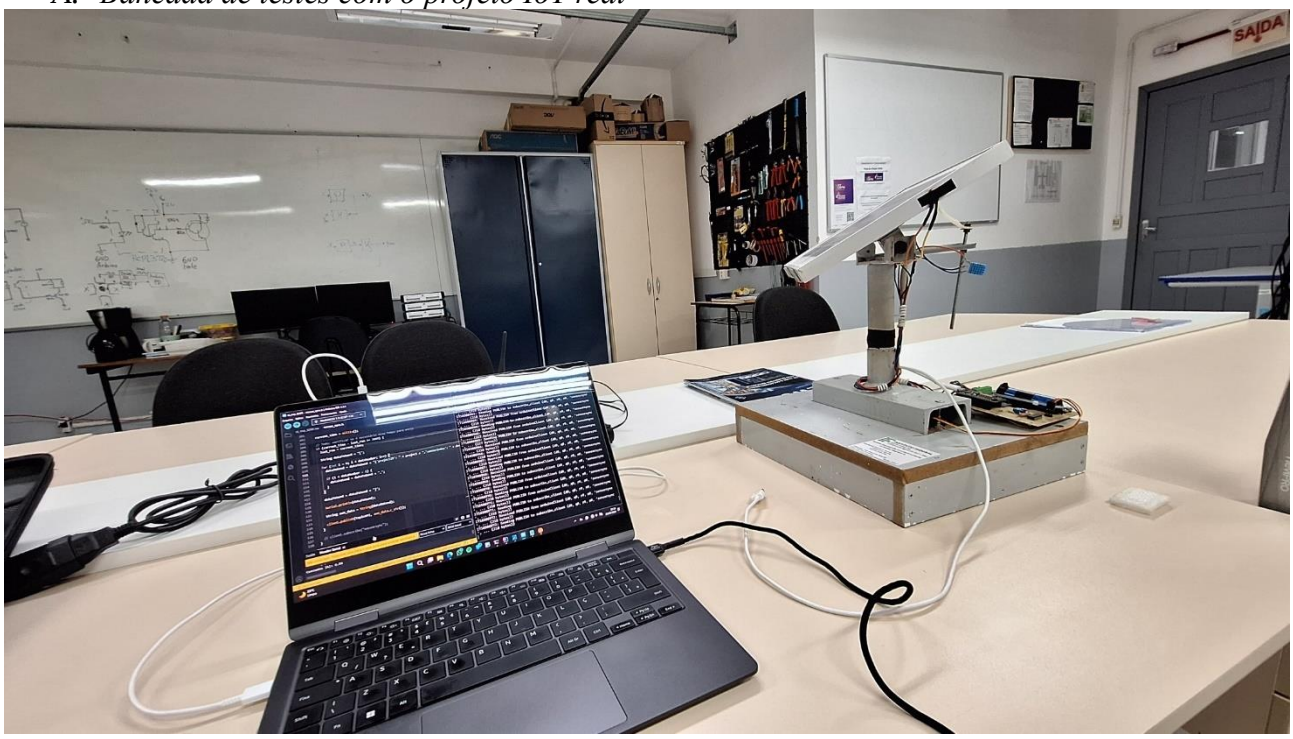
F. Microcódigo gerado para um projeto fictício

Código do Sistema Embarcado

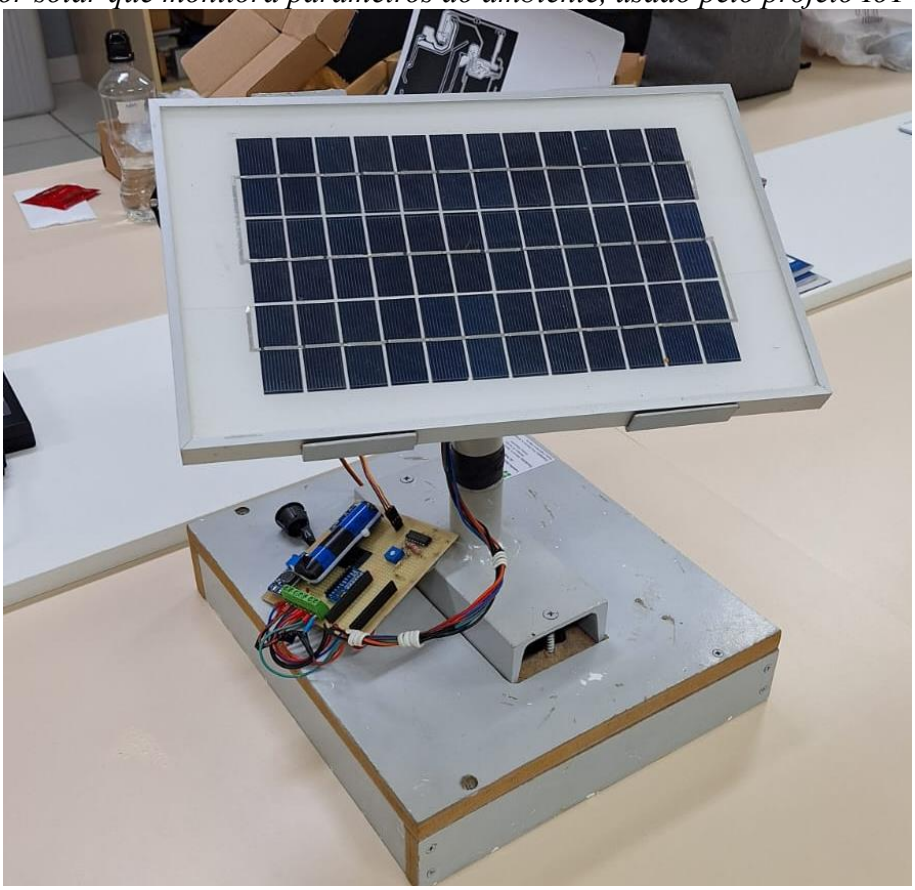
```
// CÓDIGO ESP8266 - HTTP | TEMPLATE GERADO VIA API SENSORSYNC

#include "Arduino.h"
#include "ESP8266WiFi.h"
#include "ESP8266HTTPClient.h"
#include "sensor_sync.h"

const char *WIFI_SSID = "Sistema_IoT"; //ATUALIZAR PARA AS CREDENCIAS DA SUA REDE CASO USE FORA DO LABORATÓRIO
const char *WIFI_PASSWORD = "entrarentrar" //ATUALIZAR PARA AS CREDENCIAS DA SUA REDE CASO USE FORA DO LABORATÓRIO

#define project 42
#define device 0  //ATUALIZAR PARA O ÍNDICE DO DISPOSITIVO CASO EXISTA MAIS DE UM

char *URL = "http://10.0.0.103:8080/datapackage"; //ATUALIZAR PARA O IP DO SEU SERVIDOR CASO UTILIZE FORA DO LABORATÓRIO
String desc[3];
float data[3];

void setup() {
  setup_wifi(WIFI_SSID, WIFI_PASSWORD);

  // ************************************************************************************************
  // ESPAÇO DESTINADO PARA INICIALIZAÇÃO DOS SENSORES



  // ************************************************************************************************
}

void loop() {

  // ************************************************************************************************
  // ESPAÇO DESTINADO PARA AQUISIÇÃO DOS DADOS

desc[0] = "Teste 0";
data[0] = random(0, 99);
desc[1] = "Teste 1";
data[1] = random(0, 99);
desc[2] = "Teste 2";
data[2] = random(0, 99);

  // ************************************************************************************************

  send_HTTP_data(project, device, desc, data, URL);
}
```

APÊNDICE IV – FOTOS E CAPTURAS DE TELA DO SISTEMA APLICADO EM UM PROJETO IOT REAL

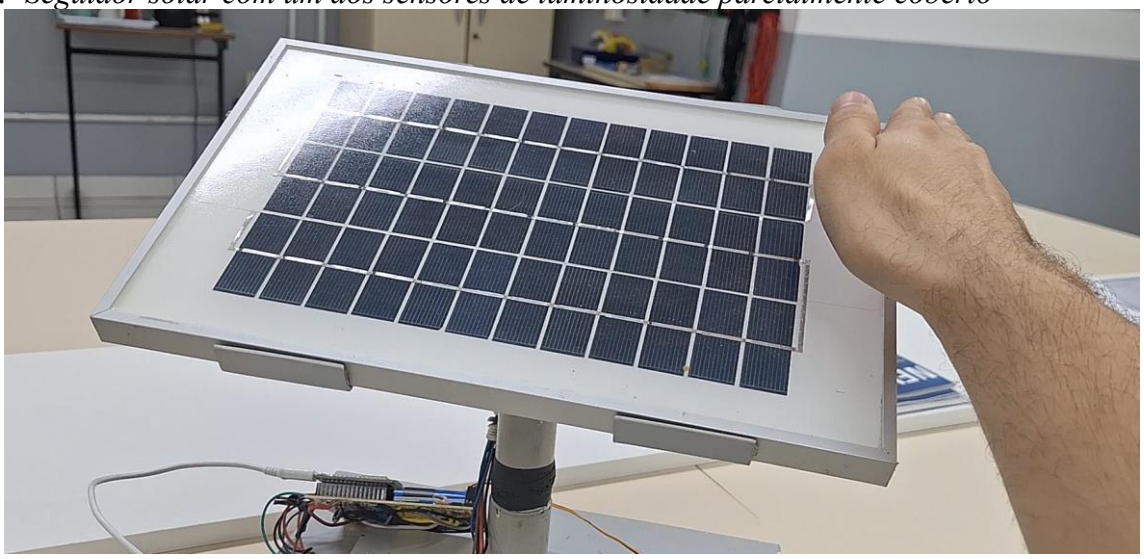
A. *Bancada de testes com o projeto IoT real*



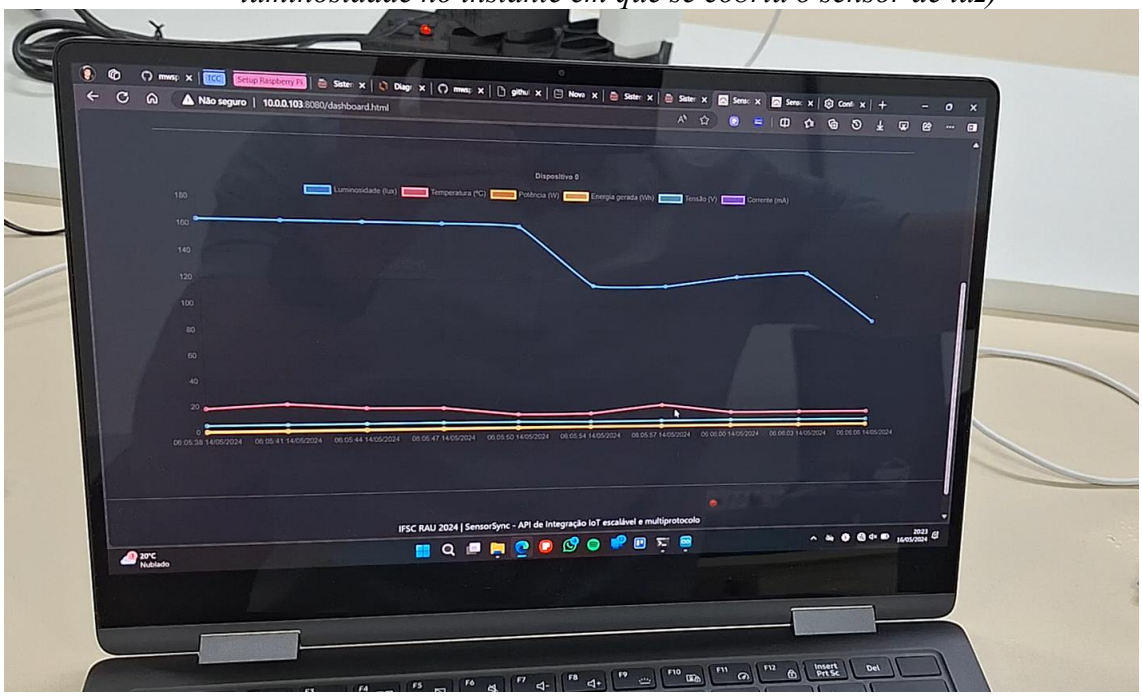
B. *Seguidor solar que monitora parâmetros do ambiente, usado pelo projeto IoT migrado*



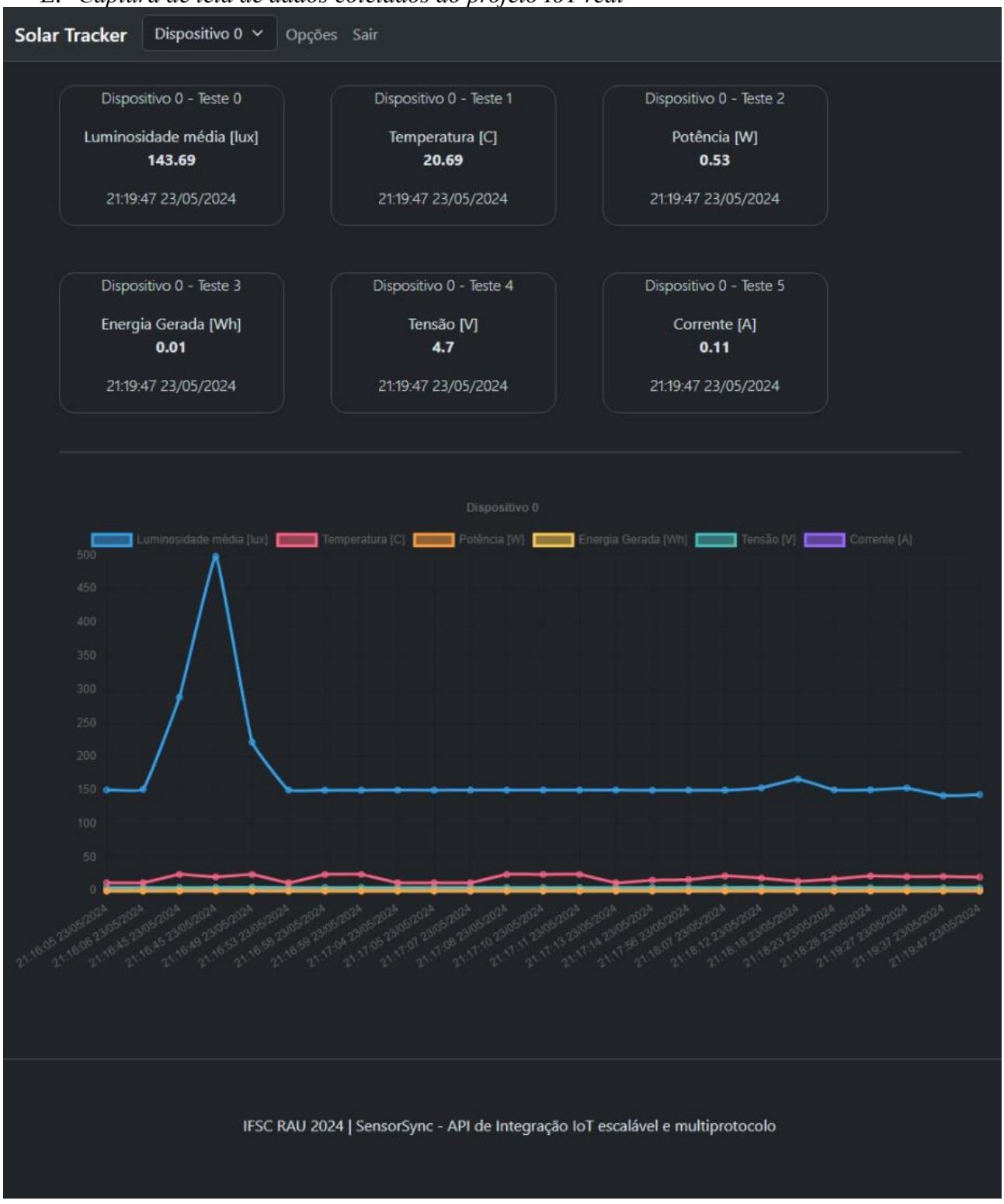
C. *Seguidor solar com um dos sensores de luminosidade parcialmente coberto*



D. *Dashboard do projeto, atualizando os dados em tempo real (queda na curva de luminosidade no instante em que se cobriu o sensor de luz)*



E. Captura de tela de dados coletados do projeto IoT real



F. Tela de relatório emitida pelo sistema, para o projeto IoT real migrado

Relatório de dados - Dispositivo 0

Instante	Luminosidade média [lux]	Temperatura [C]	Potência [W]	Energia Gerada [Wh]	Tensão [V]	Corrente [A]
21:19:47 23/05/2024	143,69	20,69	0,53	0,01	4,7	0,11
21:19:37 23/05/2024	142,15	21,7	0,53	0,01	4,73	0,11
21:19:27 23/05/2024	153,43	21,39	0,54	0,01	4,79	0,11
21:18:28 23/05/2024	150,75	22,43	0,57	0,01	4,92	0,12
21:18:23 23/05/2024	150,73	17,88	0,57	0,01	4,92	0,12
21:18:18 23/05/2024	166,62	14,57	0,6	0,01	5,0	0,12
21:18:13 23/05/2024	153,65	19,12	0,6	0,01	5,06	0,12
21:18:07 23/05/2024	150,16	22,43	0,59	0,01	5,04	0,12
21:17:56 23/05/2024	150,24	16,85	0,6	0,01	5,05	0,12
21:17:14 23/05/2024	150,16	16,02	0,59	0,01	5,01	0,12
21:17:13 23/05/2024	150,37	12,3	0,59	0,01	5,0	0,12
21:17:11 23/05/2024	150,4	24,7	0,59	0,01	4,99	0,12
21:17:10 23/05/2024	150,35	24,7	0,59	0,01	5,0	0,12
21:17:08 23/05/2024	150,38	24,7	0,59	0,01	4,99	0,12
21:17:07 23/05/2024	150,3	12,3	0,59	0,01	5,0	0,12
21:17:05 23/05/2024	150,27	12,3	0,59	0,01	4,99	0,12
21:17:04 23/05/2024	150,29	12,3	0,58	0,01	4,98	0,12
21:16:59 23/05/2024	150,26	24,7	0,59	0,01	5,0	0,12
21:16:58 23/05/2024	150,27	24,6	0,59	0,01	4,98	0,12
21:16:54 23/05/2024	150,48	12,3	0,59	0,01	5,01	0,12
21:16:49 23/05/2024	221,62	24,6	0,67	0,02	5,24	0,13