INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

FELIPE D'AVILA MESQUITA

PLATAFORMA DE CÓDIGO ABERTO PARA DESENVOLVIMENTO DE APLICAÇÕES IOT

FLORIANÓPOLIS, 2022.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL-MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA MECATRÔNICA

FELIPE D'AVILA MESQUITA

PLATAFORMA DE CÓDIGO ABERTO PARA DESENVOLVIMENTO DE APLICAÇÕES IOT

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

Orientador:
Prof. Francisco Édson Nogueira de Mélo, M. Eng.

FLORIANÓPOLIS, 2022.

FICHA DE IDENTIFICAÇÃO

Ficha de identificação da obra elaborada pelo autor.

Mesquita, Felipe
PLATAFORMA DE CÓDIGO ABERTO PARA DESENVOLVIMENTO DE APLICAÇÕES IOT / Felipe Mesquita; orientação de Francisco Édson Nogueira de Mélo. – Florianópolis, SC, 2022.
60 p.

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

1. INTERNET DAS COISAS. 2. CODIGO ABERTO. 3. SERVIDOR. I. Nogueira de Mélo, Francisco Édson. II. Instituto Federal de Santa Catarina. III. PLATAFORMA DE CÓDIGO ABERTO PARA DESENVOLVIMENTO DE APLICAÇÕES IOT.

PLATAFORMA DE CÓDIGO ABERTO PARA DESENVOLVIMENTO DE APLICAÇÕES IOT

FELIPE D'AVILA MESQUITA

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

Florianópolis, 22 de julho, 2022.

Banca Examinadora:

IFSC

Documento assinado digitalmente
FRANCISCO EDSON NOGUEIRA DE MELO
Data: 16/08/2022 19:44:57-0300
CPF: 261.107.173-04
Verifique as assinaturas em https://v.ifsc.edu.br

Francisco Édson Nogueira de Mélo, M. Eng.

gov.br

Documento assinado digitalmente
DELCINO PICININ JUNIOR
Data: 17/08/2022 09:03:26-0300
Verifique em https://verificador.iti.br

Delcino Picinin Júnior, Dr. Eng.

GOV.BR

Documento assinado digitalmente
FRANCISCO RAFAEL MOREIRA DA MOTA
Data: 17/08/2022 09:11:59-0300
CPF: 012.003.153-16
Verifique as assinaturas em https://v.ufsc.br

Francisco Rafael Moreira da Mota, Dr. Eng.

Everson Osvanir da Silva, M. Tec.

INSTITUTO FEDERAL
SANTA CATARINA

MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS

DECLARAÇÃO DE FINALIZAÇÃO DE TRABALHO DE CURSO

Declaro que o estudante **Felipe D'Ávila Mesquita**, matrícula n° **131003360-9**, do Curso de Engenharia Mecatrônica, defendeu o trabalho intitulado ***PLATAFORMA DE CÓDIGO ABERTO PARA APLICAÇÕES IOT***, o qual está apto a fazer parte do banco de dados da Biblioteca Hercílio Luz do Instituto Federal de Santa Catarina, Campus Florianópolis.

Florianópolis, 27 de julho de 2022.

IFSC
Documento assinado digitalmente
FRANCISCO EDSON NOGUEIRA DE MELO
Data: 16/08/2022 19:44:29-0300
CPF: 261.107.173-04
Verifique as assinaturas em https://v.ifsc.edu.br

Prof. Orientador do TCC: Francisco Édson Nogueira de Mélo

Dedicado à minha família.

AGRADECIMENTOS

Agradeço aos meus pais e minhas irmãs por todo o apoio nesta etapa.

Agradeço também a todo o corpo docente do IFSC por tudo que apreendi durante o curso, em especial ao professor Édson, orientador deste trabalho, que desde o primeiro semestre do curso é um incrível mentor e amigo.

Pelo apoio e interesse no projeto, agradeço aos meus colegas Maria Eduarda Silvano, Matheus Silva, Otávio Padilha e Jorge Rzatki.

“Simplicidade é um pré-requisito para a confiabilidade”
Edsger W. Dijkstra

RESUMO

O desenvolvimento de projetos eletrônicos com recursos de Internet das Coisas (do inglês, *Internet of Things* - IoT) se beneficia de grandes avanços em ferramentas e plataformas de código livre. Muitos produtos comerciais, industriais e soluções didáticas baseiam-se em *hardware* livre para seus componentes conectados, e usam de programas de código aberto para programar estes sistemas. Além do *hardware* e *software* embarcados, estes produtos conectados precisam de um servidor para armazenar dados compartilhados entre o sistema eletrônico e aplicativos *web*. As plataformas de servidor IoT existentes tendem a ser proprietárias, e, mesmo que fáceis de usar, possuem limitações nos recursos disponibilizados gratuitamente. Este trabalho apresenta o desenvolvimento de uma plataforma/serviço livre e de código aberto para sistemas IoT. O produto desenvolvido está instalado em um servidor, podendo ser usado de maneira gratuita e fácil. Por ser livre e de código aberto, pode ser instalado em qualquer máquina, bem como ter seu código modificado. O trabalho também apresenta exemplos de uso do produto desenvolvido.

Palavras-chave: Internet das Coisas, Servidor, código aberto, *Ruby on Rails*.

ABSTRACT

The development of electronic projects with IoT (Internet of Things) relies on advances of the open-source platforms and tooling that became freely available. Many commercial products, industrial and educational solutions are based on open-source hardware for their internet connected components and use open-source frameworks to program their systems. In addition to the development of the embedded hardware and software, these connected products need a server to store data and relay the connection between the electronics and a web application. For this function, the available solutions are usually closed source, and even if easy to use, they have limited resources available for free and their paid plans are ever changing and sometimes discontinued. This work documents the development of an open-source server platform that offers an easy-to-use free version and allows anyone to get a copy of its code to host a server on their own computers and modify the source freely. This work also includes example use cases of the developed platform.

Keywords: Internet of Things, Server-side, Open-Source, Ruby on Rails.

LISTA DE FIGURAS

Figura 1 – Interface para seleção de pinos virtuais no aplicativo *Blynk* 18
Figura 2 – Aviso de abandono da versão original da plataforma..... 18
Figura 3 – Diagrama de alto nível da plataforma *RainMaker* 20
Figura 4 – Módulos e componentes prontos oferecidos pelo serviço de computação em nuvem AWS..... 22
Figura 5 – Pilha de tecnologias de um servidor *Rails*..... 24
Figura 6 – Representação simplificada de uma leitura e escrita no protocolo HTTP 25
Figura 7 – Fluxo de conexão e comunicação entre dois dispositivos (*Client* A) e (*Client* B) pelo protocolo MQTT através de um Broker 26
Figura 8 – Requisição de página *web* pelo site com indicação da interação entre os componentes da arquitetura *Model-View-Controller* 28
Figura 9 – Visualização do esquema do banco de dados 32
Figura 10 – Exemplo de estações meteorológicas..... 33
Figura 11 – Página para criação de uma rede (funcionalidade 1)..... 36
Figura 12 – Página que lista as redes cadastradas (funcionalidade 2) 36
Figura 13 – Página de edição de uma rede existente (funcionalidade 3)..... 36
Figura 14 – Página com a lista de registros de uma rede (funcionalidade 5) e exclusão (funcionalidade 4) 37
Figura 15 – Servidor do aplicativo *web* idcparatodos.com.br para a rede Controle Tanque..... 46
Figura 16 – Descrição dos pinos Módulo WIFI ESP 32 *NodeMcu* 46
Figura 17 – Parte do código que mostra o uso da biblioteca “IDC.h” 47
Figura 18 – Parte do código que mostra o uso do método idc.escreverRegistro()..... 47
Figura 19 – Blocos de conexões feito no LabVIEW para funcionamento da interface 48
Figura 20 – Interface de visualização dos dados do protótipo 49
Figura 21 – Protótipo montado em uma matriz de contato..... 49
Figura 22 – Servidor do aplicativo *web* idcparatodos.com.br para a rede Balança IOT..... 50
Figura 23 – protótipo do sistema de medição de uma célula de carga 51
Figura 24 – ciclo principal do instrumento virtual..... 51
Figura 25 – Instrumento virtual da aplicação célula de carga..... 52
Figura 26 – Lista de massas 52
Figura 27 – Gráfico das leituras efetuadas..... 53

LISTA DE TABELAS

Tabela 1 – Tabela comparativa entre as plataformas comerciais analisadas 21
Tabela 2 – Tabela comparativa entre os protocolos MQTT e HTTP 27
Tabela 3 – Complexidade das alternativas para separação de variáveis.....30
Tabela 4 – Tabela "redes" do banco de dados.....34
Tabela 5 – Tabela "registros" do banco de dados.....34
Tabela 6 – Rotas do site 38
Tabela 7 – Rotas da API 39
Tabela 8 – Exemplo de requisições e suas respostas 40

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	*Amazon Web Services*
CSS	*Cascading Style Sheets*
DAE	Departamento Acadêmico de Eletrotécnica
ESP-IDF	*Espressif IoT Development Framework*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDC	Internet das Coisas
IDE	*Integrated Development Environment*
IFSC	Instituto Federal de Santa Catarina
IoT	*Internet of Things*
MQTT	*Message Queuing Telemetry Transport*
OSS	*Open Source Software*
SQL	*Standard Query Language*
URL	*Uniform Resource Locator*
VI	*Virtual Instrument*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 Contextualização 13
1.1.1 Projetos e produtos com recursos IoT 13
1.1.2 Avanços e democratização do *hardware* e *software* 14
1.1.3 Soluções para os componentes de servidor 14
1.2 Justificativa 15
1.3 Definição do Problema 15
1.4 Objetivo Geral 15
1.5 Objetivos Específicos 16

2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 Plataformas IoT Comerciais 17
2.1.1 Plataforma *Blynk* 17
2.1.2 Plataforma *RainMaker* 19
2.1.3 Comparativo entre as plataformas IoT comerciais 20
2.2 Serviços em nuvem e hospedagem própria 21
2.3 Linguagens de programação de servidor 23
2.4 Requisitos para um sistema de servidor IoT 24
2.4.1 Protocolos MQTT e HTTP 24
2.4.2 Servidor *Web* 27
2.5 Programação do microcontrolador 28

3 DESENVOLVIMENTO 29
3.1 Levantamento de funções 29
3.2 Modelo de agrupamento de variáveis 29
3.3 Implementação do banco de dados 31
3.3.1 Exemplo de banco de dados 33
3.4 Interface do aplicativo *web* 34
3.5 Implementação da versão pública 37
3.6 Rotas do site 38
3.7 Rotas da API 38
3.8 Lógica do controlador 39
3.9 Exemplo de requisições na API 39
3.10 Biblioteca Arduino para ESP32 e ESP8266 40
3.10.1 Definição das funções da biblioteca 40
3.10.2 Implementação da biblioteca 41

4 RESULTADOS 45
4.1 Aplicação em protótipo de planta de controle de nível 45
4.2 Aplicação didática com célula de carga 50
4.3 Uso em aplicações didáticas 53
4.4 Uso em aplicações gerais 54

5 CONSIDERAÇÕES FINAIS 55
5.1 Sugestões para trabalhos futuros 55

REFERÊNCIAS 58

1 INTRODUÇÃO

Segundo IDC (2022, p. 1), “em mercados emergentes, como o Brasil, implantações de IoT que visam reduzir custos e aumentar a eficiência operacional têm mais força frente a temas de inovação.”. Nesse sentido, temas envolvendo Internet das Coisas (IDC) (do inglês, *Internet of Things* - IoT), têm ganhado bastante destaque no mercado tecnológico do país. Além disso, IDC (2022) projeta para esse ano um crescimento de 17,6% em relação ao ano anterior, equivalente a US$ 1,6 bilhão, composto por soluções e serviços relacionados a IoT no Brasil.

Com a perspectiva de crescente interesse do mercado, é interessante que sejam avaliadas as soluções disponíveis para a criação de produtos e serviços no seguimento de internet das coisas, bem como as oportunidades de desenvolvimento de novas ferramentas.

1.1 Contextualização

O termo IoT “refere-se à rede coletiva de dispositivos conectados e à tecnologia que facilita a comunicação entre os dispositivos e a nuvem, bem como entre os próprios dispositivos.” AWS (2022a, p. 1). Ainda, a AWS (2022a) afirma que em um típico sistema de IoT possui três componentes: dispositivos inteligentes, aplicação de IoT (conjunto de serviços e softwares) e uma interface gráfica do usuário. Neste sentido, um sistema com recursos de IoT depende de ferramentas para seu desenvolvimento, *hardware* eletrônico embarcado e um servidor para integrar esse *hardware* com a interface em, por exemplo, um *site web* ou um aplicativo móvel.

1.1.1 Projetos e produtos com recursos IoT

A inclusão de recursos de IoT em projetos eletrônicos possibilita o monitoramento e controle remoto de dispositivos de maneira remota, através da internet. Estes recursos, aqui definidos como capacidade do sistema de consultar e registrar informações sobre o processo para um servidor externo, são importantes em linhas industriais para o acompanhamento em tempo real de sensores que podem apontar a necessidade de manutenção de um equipamento, como um sensor de

vibração em um motor detectando uma irregularidade e, também, o ajuste de parâmetros de operação, como uma das constantes de um controlador Proporcional Integral Derivativo (PID) em um forno. Estes recursos também adicionam valor a produtos comerciais, possibilitando, por exemplo, controle remoto de lâmpadas, portões e eletrodomésticos em geral.

1.1.2 Avanços e democratização do *hardware* e *software*

O desenvolvimento e ensino de recursos de internet das coisas tornou-se mais acessível nos últimos anos com a popularização de novos controladores de baixo custo que já integram modems e antenas *wifi* e *bluetooth*, além de facilidades como integração com o ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) de código aberto *Open Source Software* (OSS) Arduino. A linha ESP da empresa chinesa *Espressif Systems* é um exemplo muito popular para entusiastas e utilizado em muitos produtos comerciais de produção em série, devido ao seu baixo custo e boa documentação. A *Espressif* oferece também como código aberto a *Espressif IoT Development Framework* (ESP-IDF) com recursos para projetos mais avançados, como sistema operacional em tempo real e leitura e escrita de arquivos à memória interna do chip.

1.1.3 Soluções para os componentes de servidor

Um sistema com recursos IoT necessita, além do processador, de um modem e de um *software* embarcado, de um componente em servidor que receba, armazene e disponibilize as variáveis e eventos para o sistema embarcado e interface com o usuário, tipicamente através de uma página *web* ou aplicativo móvel. Neste espaço também ocorreram avanços: plataformas simplificadas, como o *RainMaker* (da própria *Espressif*) e *Blynk, que* facilitam a experimentação, a prototipagem e possibilitam o desenvolvimento de um produto IoT sem a necessidade de desenvolver/codificar um serviço, bem como de instalar esse serviço em um servidor.

Estas soluções são interessantes em muitos casos, porém apresentam deficiências, como limites nas capacidades da oferta gratuita e impossibilidade de se modificar ou hospedar localmente o código de servidor. O custo também é um

problema para casos de grande volume de informações transmitidas nestas plataformas, afetando estudantes e países com câmbio desfavorável com o dólar americano, utilizado como forma de pagamento. Esses problemas de custo e inflexibilidade podem ser resolvidos se o componente de servidor for de código aberto, permitindo modificações e execução local e sendo distribuído gratuitamente.

Estes serviços são essenciais para aplicações IoT que devam ser acessíveis pela internet em sites ou aplicativos móveis, mantendo um banco de dados comum entre os dispositivos IoT e os aplicativos de interface.

1.2 Justificativa

Devido à carência de uma plataforma de baixo **c**usto ou gratuita e flexível em questão de personalização, este trabalho integra tecnologias de múltiplas áreas de estudo da Engenharia Mecatrônica para desenvolver uma plataforma que busca tornar mais acessível o componente de servidor de sistemas conectados para fins de aprendizado ou uso comercial, através da disponibilização das tecnologias resultantes em código livre.

1.3 Definição do Problema

Observa-se um espaço para uma plataforma de servidor para IoT de código aberto, com as facilidades de experimentação rápida das versões comerciais de código fechado. Esta plataforma também deve ser gratuita para qualquer desenvolvedor poder obter uma cópia do código fonte e executá-lo em qualquer computador.

1.4 Objetivo Geral

Desenvolver e validar uma plataforma fácil de usar e código aberto para servidores de aplicações IoT.

1.5 Objetivos Específicos

Os objetivos específicos abrangem os seguintes itens:

a) Avaliar sistemas de servidor e computação em nuvem para modelar uma solução acessível em termos de expansão por outros desenvolvedores e custo de operação do servidor;

b) Avaliar casos de uso de plataformas IoT comerciais para mapear funções simplificadas de comunicação entre o sistema embarcado e o servidor;

c) Desenvolver um programa de servidor, implementando as funções necessárias para a comunicação com o sistema embarcado e o armazenamento dos registros em um banco de dados;

d) Desenvolver uma interface através de aplicativo *web* para configuração e visualização dos dados;

e) Desenvolver uma biblioteca para a plataforma ESP que exemplifique a comunicação com o servidor;

f) Validar esta plataforma e biblioteca em aplicações práticas.

2 FUNDAMENTAÇÃO TEÓRICA

O desenvolvimento de um sistema de servidor requer a análise de diversas escolhas entre um crescente e cada vez mais complexo ecossistema de ofertas em serviços de computação em nuvem, sistemas operacionais de servidor, modelos de organização de código, tipos de bancos de dados, *frameworks* de desenvolvimento de interface gráfica e muitos outros componentes. Os desafios específicos da criação de uma plataforma de código aberto para dispositivos com recursos IoT orientam essas escolhas na busca por soluções que ofereçam boa performance, simplicidade no código e não envolvam módulos proprietários que impossibilitariam que todo o código necessário ao funcionamento da plataforma fosse disponibilizado livremente para qualquer uso.

2.1 Plataformas IoT Comerciais

As soluções existentes de fácil utilização para projetos IoT implementam recursos comuns de leitura e escrita de variáveis pelos sistemas eletrônicos embarcados e outros dispositivos conectados ao seu servidor e apresentação destas variáveis aos usuários finais por um aplicativo *web*.

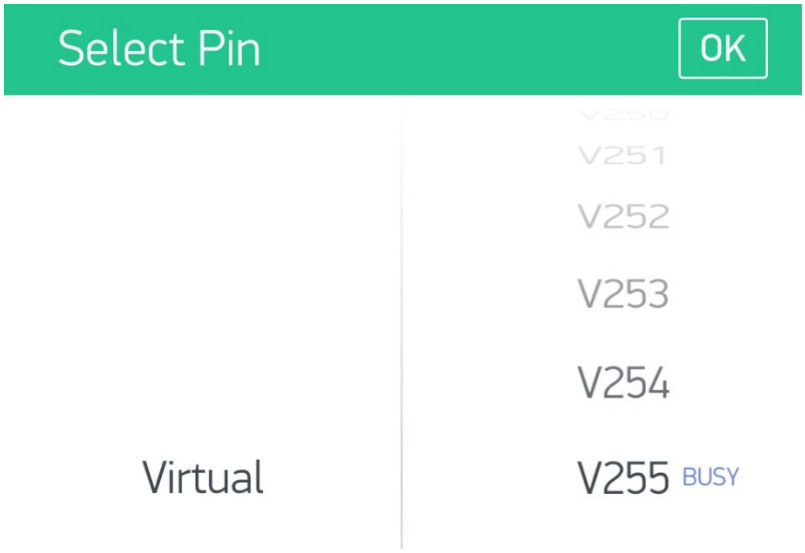
2.1.1 Plataforma *Blynk*

A plataforma *Blynk* oferece soluções bem completas com integração a ferramentas para projeto de interface gráfica. Segundo a documentação de Blynk (2022, tradução nossa), “o Blynk é um conjunto completo de *software* necessário para prototipar, implantar e gerenciar remotamente dispositivos eletrônicos conectados em qualquer escala: de projetos pessoais de IoT a milhões de produtos comerciais conectados”.

O modelo de compartilhamento de variáveis entre os dispositivos e o servidor é limitado a 256 variáveis por dispositivo, uma vez que estas variáveis podem apenas ser atribuídas a “pinos virtuais”, em uma lista pré-definida que vai de V0 até V255, conforme a interface para seleção de pinos virtuais no aplicativo *Blynk* mostrado na Figura 1. Para protótipos e projetos simples, essa limitação não deve causar

grandes problemas, mas em casos que o projeto envolva vários dispositivos que precisem comunicar variáveis entre si, a impossibilidade de nomear uma variável deixa de ser uma boa escolha.

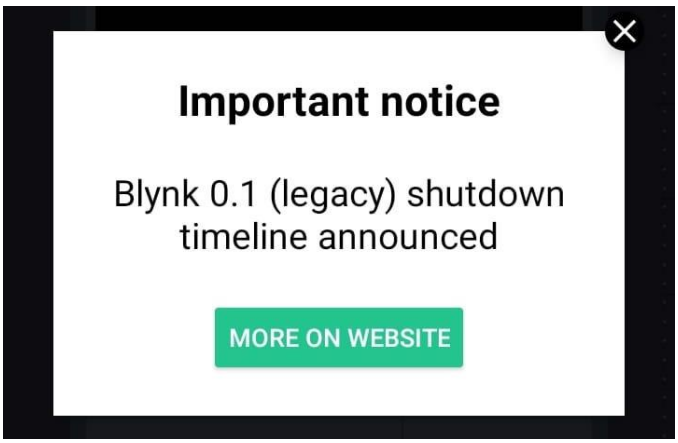
Figura 1 – Interface para seleção de pinos virtuais no aplicativo *Blynk*



Fonte: Blynk (2022).

A plataforma *Blynk* contava com um sistema de servidor de código aberto, porém essa oferta foi removida. Atualmente existem opções apenas entre uso do limitado plano gratuito e o plano pago que a empresa oferece. Além da interrupção da oferta do sistema de código aberto para o servidor, as ferramentas de interface gráfica foram abandonadas, conforme aviso de abandono da versão original da plataforma mostrado na Figura 2, em favor de um novo sistema de projeto de interface gráfica, também com um plano gratuito limitado e ofertas pagas com custos elevados.

Figura 2 – Aviso de abandono da versão original da plataforma



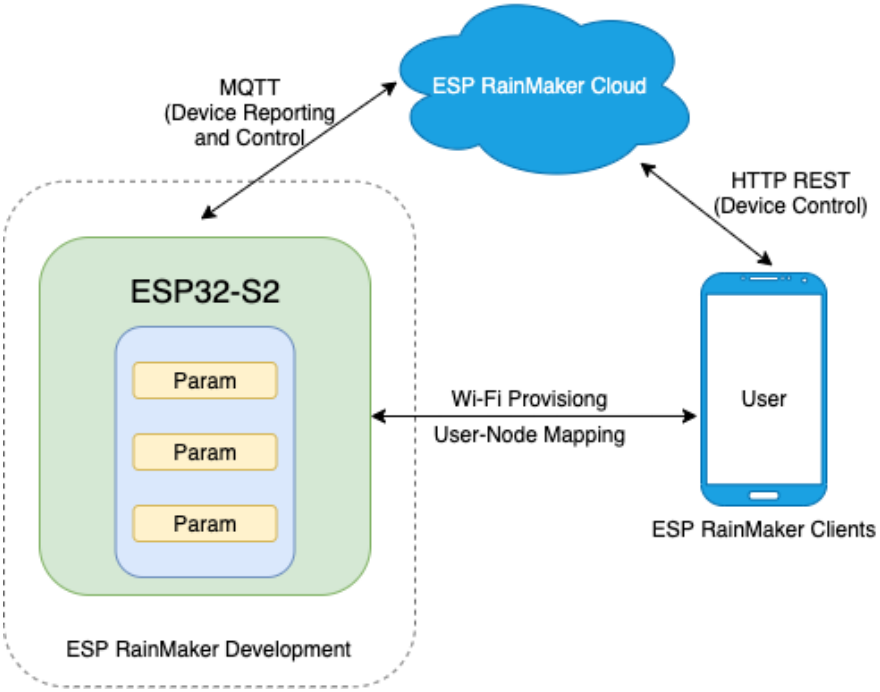
Fonte: Blynk (2022).

Uma análise da plataforma *Blynk* mostra que ela é interessante para usuários iniciantes, por sua interface organizada e facilidade de experimentação. Também pode servir para usuários mais avançados como ferramenta de prototipagem ou em projetos com grande disponibilidade de recursos, que podem contratar os planos mais caros e então remover os limites impostos aos planos mais acessíveis e ao gratuito. Porém, devido às frequentes mudanças na estratégia de precificação da plataforma, acredita-se que ela não serve como fundamento sólido para projetos comerciais com recursos financeiros menores. O cenário de uso em sala de aula também poderia ser mais bem atendido por uma plataforma que possibilitasse que os alunos alcançassem projetos mais complexos sem precisarem contratar planos de alto custo.

2.1.2 Plataforma *RainMaker*

A oferta da chinesa *Espressif* para servidor oferece uma solução muito completa, com escolhas bem fundamentadas para protocolos de comunicação. Conforme a Figura 3, em que o servidor é indicado pelo balão “ESP *RainMaker Cloud*”, os dispositivos se comunicam com o servidor pelo protocolo *Message Queuing Telemetry Transport* (MQTT), segundo ESP *RainMaiker* (2022), oferecendo alta velocidade e tamanho reduzido (em bytes) dos pacotes que precisam ser transmitidos pela rede. O servidor também implementa o protocolo *Hypertext Transfer Protocol* (HTTP) para poder comunicar-se com aplicativos *web* e móveis nos dispositivos dos usuários finais. A seção 2.4.1 deste trabalho aborda questões práticas sobre a implementação destes dois protocolos.

Figura 3 – Diagrama de alto nível da plataforma *RainMaker*



Fonte: Espressif (2022).

Esta solução da *Espressif* é proprietária, apesar de oferecer como código aberto todas as ferramentas de desenvolvimento embarcado para os seus módulos ESP32 e ESP8266 e os aplicativos para celular que se comunicam com o servidor *RainMaker*, o componente de servidor é completamente fechado, deixando seus usuários sem recurso algum quanto a mudanças nas capacidades ou preço que a empresa venha a fazer. Aponta-se um problema também no preço, que ainda não é divulgado publicamente, a empresa indica apenas que essa informação será disponibilizada futuramente.

2.1.3 Comparativo entre as plataformas IoT comerciais

Conforme a análise feita das plataformas *Blynk* e *RainMaker* nas seções 2.1.1 e 2.1.2, destaca-se na Tabela 1 os principais pontos de comparação discutidos entre esses plataformas comerciais.

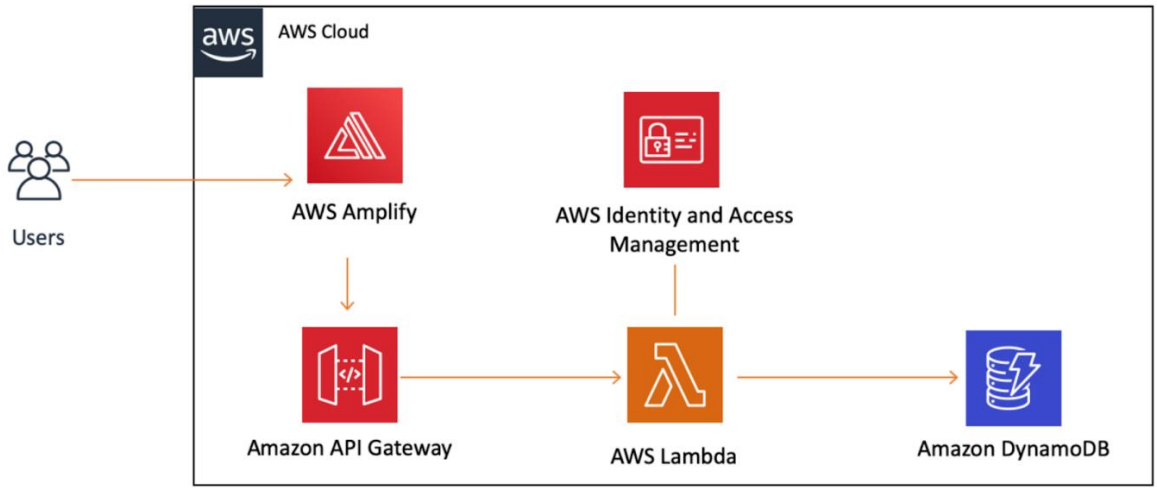
Tabela 1 – Tabela comparativa entre as plataformas comerciais analisadas

	Blynk	**RainMaker**
Modelo de variáveis	Conceito de pinos virtuais numerados de 0 até 255	Parâmetros nomeados
Protocolo	HTTP	MQTT entre o servidor e o sistema embarcado, HTTP entre o servidor e demais clientes
Custo	Inicialmente cobranças dentro do app, atualmente planos de assinatura mensal, cobrados em USD	Não divulgado, ainda em período de teste gratuito.
Servidor código aberto	Descontinuado	Não

2.2 Serviços em nuvem e hospedagem própria

Existem muitas formas de desenvolver e operacionalizar servidores *web*. Uma distinção inicial pode ser feita no uso de plataformas mais prontas de computação em nuvem, como os serviços do *Amazon Web Services (AWS)* e o *Google Cloud Platform*. Este primeiro tipo de solução traz vantagens para grandes empresas principalmente pela sua capacidade de ampliar automaticamente os recursos computacionais utilizados conforme a demanda de seus clientes varia. Analisando-se a estrutura de um sistema simples com *Application Programming Interface* (API) e banco de dados, conforme AWS (2022b), são encontrados diversos módulos exclusivos à “nuvem” da *Amazon*, conforme a Figura 4.

Figura 4 – Módulos e componentes prontos oferecidos pelo serviço de computação em nuvem AWS



Fonte: Amazon AWS (2022b).

Estes módulos são código fechado e podem ser usados apenas em computadores dos servidores da empresa de computação em nuvem, sendo necessário um contrato com a empresa para seu uso.

Um segundo método faz uso de sistemas operacionais de servidor, entre os quais, de acordo com W3Techs (2022), o *Ubuntu* é o mais utilizado. Desenvolvem-se os programas de servidor para este sistema, que depois pode ser executado em um computador comum ou em um servidor comercial contratado que apenas recebe o *software* pronto e o executa em uma máquina virtual com o mesmo sistema. Esta forma de desenvolvimento possibilita que os componentes criados sejam distribuídos e executados sem as ferramentas dos serviços prontos de computação em nuvem, que podem ter custos proibitivos para vários projetos ou situações. Considerando os objetivos de criar um sistema acessível de código aberto, este projeto será desenvolvido para o sistema operacional de código aberto *Ubuntu*.

O termo “hospedagem própria” é comumente empregado para referir-se a configurações em que o desenvolvedor controla todo o sistema do servidor, como no caso descrito anteriormente de um sistema *Linux*. O termo não implica na manutenção de servidores físicos, ou seja, considera-se hospedagem própria uma situação em que o desenvolvedor paga para um serviço de servidor hospedar seu sistema *Linux* em uma máquina virtual, a distinção está no uso dos componentes ou módulos que não poderiam ser usados em outro vendedor de serviço de servidor ou em um computador do próprio desenvolvedor.

2.3 Linguagens de programação de servidor

O desafio da programação de servidores encontra-se em integrar diversas tecnologias: banco de dados, servidor *web* (componente que efetivamente recebe, processa e responde às requisições *web* HTTP) e funções do sistema operacional como leitura e escrita de arquivos e comando de outros processos. Segundo Schutt e Balci (2016), os desenvolvedores de programas de servidor buscam plataformas para tornar seu desenvolvimento mais integrado, testável, fácil de manter e configurável. Os autores apresentam Java, .NET, *Ruby*, PHP, *JavaScript* e Python como principais linguagens para programação de servidores, e Java EE, .NET, *Ruby on Rails*, *Zend*, *Node.js* e Django como suas principais plataformas. Cada solução é comparada pelos autores e são percebidas muitas similaridades em funções e performance.

Em *Ruby on Rails*, segundo Rails (2022), seus usuários podem aprender apenas o necessário para começar a desenvolver com a *framework* e então subir de nível [em seu aprendizado] conforme desenvolvem. Rails (2022) também argumenta que oferece proteções sólidas a ataques e tentativas de invasão do servidor.

O foco de *Ruby on Rails* em se manter acessível para iniciantes é considerado importante no contexto deste projeto, uma vez que se espera oferecer uma ferramenta para projetistas interessados em recursos de internet das coisas que podem não ter ainda se aprofundado no estudo de programação de servidores.

Um conjunto de ferramentas estabelecidas é usado por padrão em projetos *Rails*, facilitado parte do complexo processo de determinação das tecnologias a serem utilizadas em um servidor. De forma resumida, conforme ilustrado na Figura 5, a *framework Ruby on Rails* utiliza a linguagem de programação *Ruby*, que tem um “foco em simplicidade e produtividade”, de acordo com Ruby (2022). O banco de dados baseado na linguagem *Standard Query Language* (SQL) usado por padrão é o SQLite, segundo SQLite (2022) é o mais utilizado do mundo e oferecido no domínio público para qualquer aplicação.

Conforme apresentado na seção 2.2, este projeto utilizará o sistema operacional *Ubuntu* para seu desenvolvimento e implementação da versão pública, porém todo o código aberto do servidor desenvolvido será executável de diversos outros sistemas, como *Windows*, *Mac* e outras distribuições de *Linux*.

Figura 5 – Pilha de tecnologias de um servidor *Rails*

Fonte: Elaboração própria (2022).

2.4 Requisitos para um sistema de servidor IoT

A aplicação específica de Internet das Coisas necessita de um servidor acessível por microcontroladores limitados, sendo importante que o servidor responda requisições *web* sem exigir protocolos específicos modernos como HTTP2 e envie respostas simples de serem processadas pelo microcontrolador, contendo exatamente o conteúdo requisitado por ele.

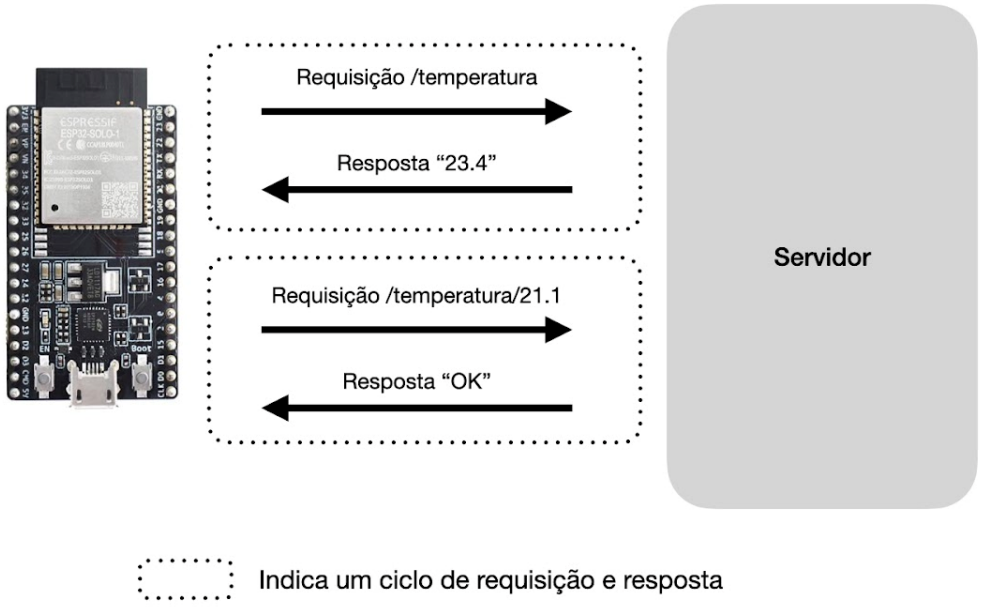
2.4.1 Protocolos MQTT e HTTP

Os principais protocolos de conexão que podem ser utilizados na comunicação entre o servidor e o microcontrolador são o MQTT e o HTTP.

O protocolo HTTP é utilizado por todos os navegadores da internet e é o padrão para comunicação entre sistemas *web* através de APIs. Seu ciclo de requisição e resposta implementa códigos padronizados para operações bem-sucedidas e erros. No contexto deste projeto, o microcontrolador iniciaria requisições HTTP ao servidor

para ler ou escrever dados e receberia como resposta na leitura o dado em questão e na escrita se a operação foi executada sem erros. A Figura 6 mostra uma representação simplificada de uma leitura e escrita no protocolo HTTP.

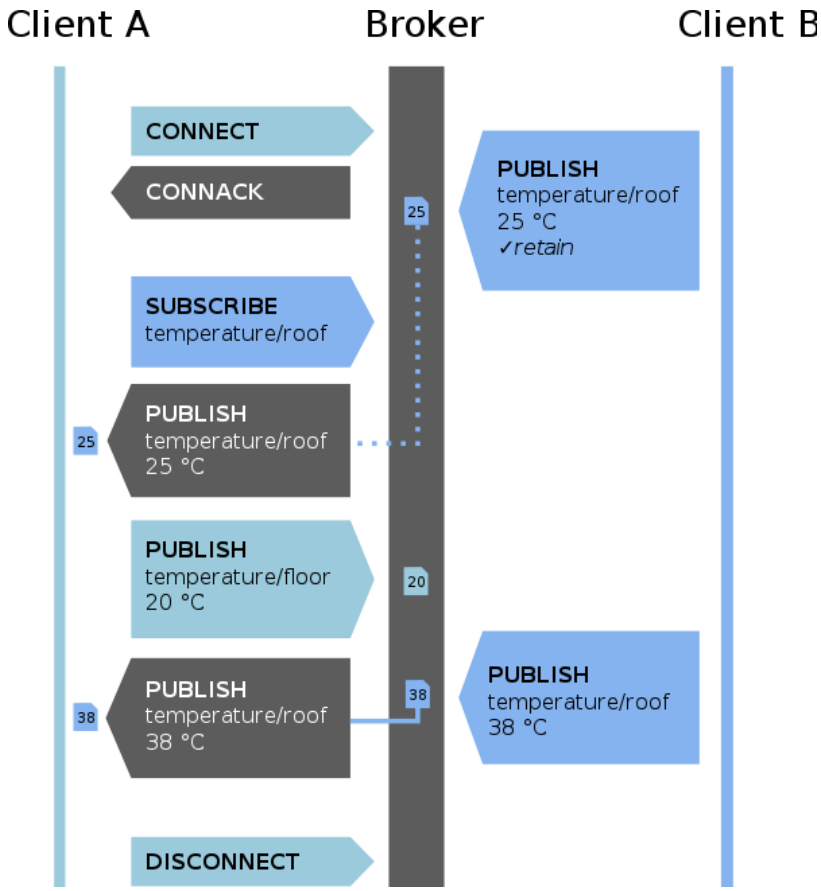
Figura 6 – Representação simplificada de uma leitura e escrita no protocolo HTTP



Fonte: Elaboração própria (2022).

O protocolo MQTT usa um modelo Publicador-Subscritor, em que é estabelecido um canal de comunicação com um intermediário (*broker*) e as mensagens subsequentes são enviadas nesse canal (BANKS *et al*, 2019). Este protocolo é otimizado para alcançar o menor tamanho possível das mensagens enviadas, procurando atender aplicações em que a disponibilidade de energia, banda de comunicação e processamento são extremamente restritos. A Figura 7 mostra o fluxo de conexão e comunicação entre dois dispositivos (*Client* A) e (*Client* B) pelo protocolo MQTT através de um *Broker*.

Figura 7 – Fluxo de conexão e comunicação entre dois dispositivos (*Client* A) e (*Client* B) pelo protocolo MQTT através de um Broker



Fonte: Eugster (2018).

Avaliando estes protocolos e considerando o resultado de análises técnicas como a de Wang (2018), o MQTT apresenta vantagens para aplicações alimentadas por pequenas baterias e consegue transmitir uma mesma mensagem em menos bytes enviados pela rede. O HTTP dispensa a necessidade de um *broker* e possibilita eliminar a complexidade de um protocolo adicional, uma vez que ele obrigatoriamente será utilizado para responder as requisições dos navegadores *web* que acessarem o site do projeto. Também, estima-se que a maior parte dos projetos desenvolvidos com o objeto deste trabalho terão conexão com a internet por *wifi* (e não GSM ou 4G), onde a redução do tamanho das mensagens enviadas oferecida pelo MQTT pode ser considerada irrelevante. A Tabela 2 apresenta um resumo comparativo entre os protocolos MQTT e HTTP.

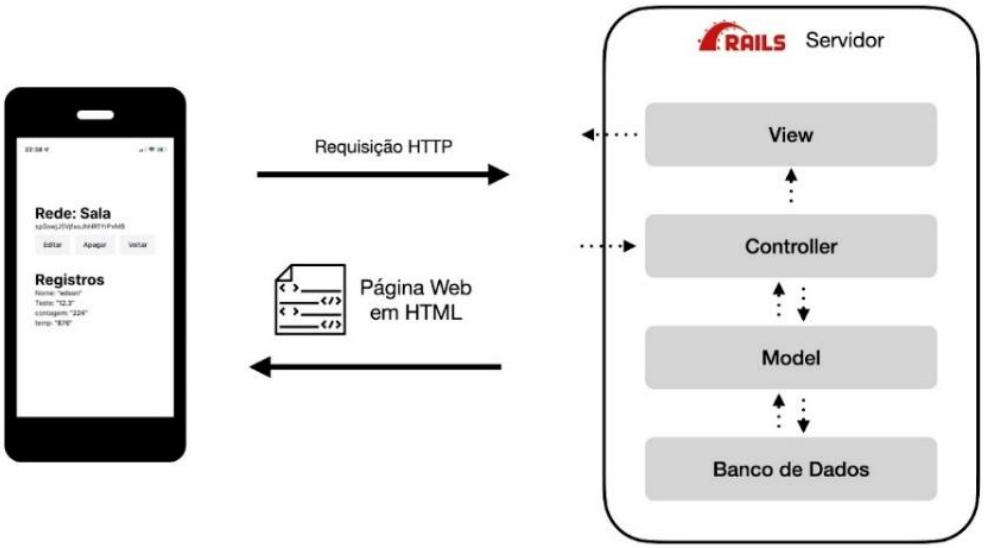
Tabela 2 – Tabela comparativa entre os protocolos MQTT e HTTP

	MQTT	**HTTP**
Consumo de energia	Baixo	Alto
Tamanho adicional transferido a cada mensagem	Muito pequeno	Grande
Arquitetura	Publicador-Subscritor	Requisição-Resposta
Necessidade de intermediador	Sim, broker	Não
Compatibilidade com navegadores web	Não	Sim

2.4.2 Servidor *Web*

O servidor proposto neste projeto deve, além de enviar e receber dados de sistemas eletrônicos embarcados, oferecer uma interface *web* que permita a visualização destes dados e configuração de outros parâmetros. Este componente deve receber requisições do navegador, acessar os registros necessários no banco de dados e organizar estes dados numa página para o navegador exibir. No *framework Ruby on Rails* é utilizada a arquitetura *Model-View-Controller* (em português, Arquitetura Modelo-Visão-Controle - MVC), que orienta a organização da programação do servidor *web* em responsabilidades distintas: o *Controller* recebe a requisição do cliente (um navegador em um celular ou computador) e chama funções do *Model*, que por sua vez interage com o banco de dados e retorna ao *Controller* as informações recuperadas ou escritas. O *Controller* então decide qual *View* (página) deve ser exibida ao cliente e passa a ela os dados que obteve do Model. A *View* então tem a responsabilidade de organizar estas informações de forma legível em um leiaute para ser enviado ao cliente. A Figura 8 mostra a requisição de página *web* pelo site com indicação da interação entre os componentes da arquitetura *Model-View-Controller*.

Figura 8 – Requisição de página *web* pelo site com indicação da interação entre os componentes da arquitetura *Model-View-Controller*



Fonte: Elaboração própria (2022).

2.5 Programação do microcontrolador

A IDE do Arduino é um excelente ponto de entrada para iniciantes em programação de sistemas embarcados, além de popular ferramenta para prototipagem rápida. Considera-se importante para uma plataforma que busca facilidade de uso oferecer uma biblioteca simplificada compatível com a IDE do Arduino, removendo obstáculos à experimentação inicial com a plataforma desenvolvida neste trabalho. Para usuários experientes, espera-se que a biblioteca compatível com a IDE do Arduino seja uma referência fácil e primeiro exemplo do funcionamento do sistema.

3 DESENVOLVIMENTO

O desenvolvimento da plataforma IoT foi estruturado em um processo de definição dos requisitos funcionais de uma primeira versão desta plataforma, seguido de análises sobre aplicação dos conceitos básicos de leitura e escrita de variáveis entre dispositivos e servidores buscando estruturar esses recursos de forma que seu funcionamento e código sejam de fácil compreensão. Foi então projetada uma interface *web* para a plataforma e configurado um servidor para disponibilizar uma versão publica da plataforma em idcparatdodos.com.br. Foi também desenvolvida uma biblioteca compatível com o ambiente de desenvolvimento Arduino.

3.1 Levantamento de funções

A plataforma desenvolvida neste projeto deve conter funcionalidade para possibilitar a leitura e a escrita de variáveis em um banco de dados no servidor (API) por dispositivos IoT e buscar o máximo de compatibilidade com quaisquer dispositivos conectados que implementem comunicação HTTP, como instrumentos virtuais (do inglês, *Virtual Instrument* – VI) do *LabVIEW* e *sketches* do *Processing*. Essas variáveis devem também ser exibidas em um *website* (*web app*) que ofereça formas de apagar/criar grupos de variáveis para diferentes projetos ou usuários. Com o objetivo de oferecer todo o código de forma livre, pode-se identificar uma terceira interface oferecida no projeto como o próprio código que será escrito, que deve ser construído para possibilitar fácil extensão e modificação.

3.2 Modelo de agrupamento de variáveis

A plataforma deve oferecer acesso às variáveis de cada usuário ou projeto, além da configuração destes agrupamentos de variáveis, por exemplo, um usuário pode desejar uma variável chamada temperatura no contexto de um sistema de controle de um forno IoT e no contexto de uma estação meteorológica. Para separação destes contextos mantendo a simplicidade de referir-se a variáveis por um nome e não um id numérico, existem algumas possibilidades a serem consideradas:

- Um sistema de login de usuários com e-mail e senha;
- Um sistema de *token* por projeto;
- Uma combinação de ambos, com usuários que são donos de projetos que possuem *tokens*.

Essas alternativas devem ser analisadas conforme seu impacto na complexidade e familiaridade nas três interfaces oferecidas no projeto: API, *Web App* e o próprio código. Na Tabela 3 é mostrada a complexidade das alternativas para separação de variáveis.

Tabela 3 – Complexidade das alternativas para separação de variáveis

	Logins/Usuários	***Tokens***	**Ambos**
Complexidade na sua implementação no código de servidor	Média	Baixa	Alta
Possibilita um mesmo usuário facilmente gerenciar múltiplos projetos	Não (cada usuário poderia ter apenas uma variável com um dado nome)	Sim, variáveis seriam separadas por *token*	Sim, variáveis seriam separadas por *token*
Complexidade do uso da API	Alta, autenticação por usuário e senha envolvem um fluxo mais complexo	Baixa, deve-se apenas incluir o *token* nas chamadas à API	Baixa, deve-se apenas incluir o *token* nas chamadas à API
Complexidade devido ao armazenamento de dados de usuários	Requer endereço de e-mail ou login com alguma outra informação identificável	Não requer nenhuma informação do usuário	Requer endereço de e-mail ou login com alguma outra informação identificável
Necessidade de algoritmo seguro de geração de números aleatórios	Não	Sim	Não

Fonte: Elaboração própria (2022).

Conforme a Tabela 3, percebe-se que o modelo que utiliza apenas *tokens* para separar grupos de variáveis oferece menor complexidade adicional, uma vez que a plataforma objeto deste trabalho não busca oferecer interações complexas entre usuários ou oferecer planos pagos por assinatura. Estas considerações certamente seriam diferentes para um produto comercial, e podem ser implementadas por desenvolvedores que busquem tal resultado a partir do código oferecido por este projeto.

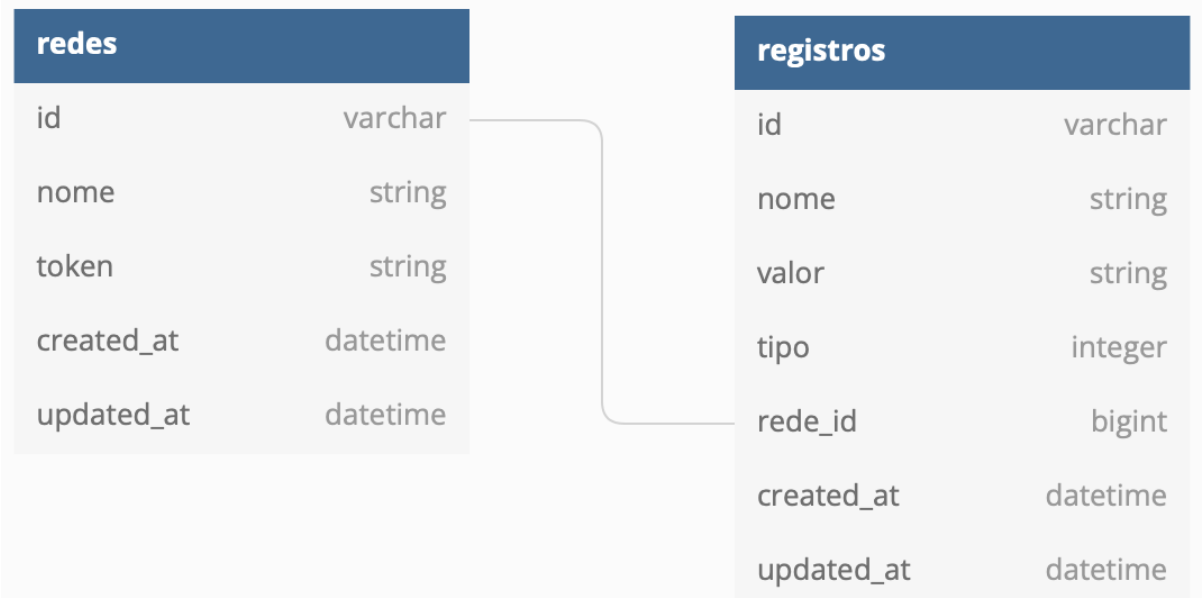
3.3 Implementação do banco de dados

Decidindo-se pelo agrupamento de variáveis por *tokens*, pode-se modelar a estrutura das tabelas - chamada esquema - do banco de dados relacional que será utilizado. Será necessário armazenar informações sobre dois conceitos: *tokens* e variáveis. Para evitar confusão com conceitos similares, propõe-se que neste projeto faça-se uso de nomes diferentes. Variável é um termo amplamente utilizado na área de programação, tornando-se uma escolha ruim para definir o caso específico dos valores salvos no servidor da plataforma desenvolvida neste projeto. O termo “registro” oferece distinção entre estes valores compartilhados com os sistemas embarcados e as demais variáveis comuns dos programas. O termo *token*, mesmo não sendo tão amplamente utilizado, representa apenas parte do modelo do banco de dados que o contém: devemos armazenar o nome que o usuário escolher e parâmetros para ordenar listas como data de criação e atualização. Portando, o termo “rede” será utilizado para definir este modelo que contém o id interno para performance do banco de dados, o nome do grupo de variáveis que o usuário escolheu, o *token* gerado para que se possa acessar estas variáveis e as datas de criação e atualização de quaisquer destes parâmetros.

Definem-se dois modelos necessários ao funcionamento da plataforma: redes e registros. Uma rede pode possuir qualquer número de registros, com a limitação de que os nomes de cada registro sejam únicos. Os usuários da plataforma podem então criar qualquer número de redes conforme sua necessidade de separar os registros necessários para seus projetos.

Em um banco de dados relacional, cada modelo será traduzido a uma tabela, com as dependências entre modelos indicadas em colunas que armazenam o id de um item em outra tabela. Uma vez que uma rede pode possuir vários registros, porém um registro deve pertencer sempre a uma e somente uma rede, deve-se armazenar nos registros o id da rede a qual cada pertence, conforme o esquema do banco de dados mostrado na Figura 9.

Figura 9 – Visualização do esquema do banco de dados



Fonte: Elaboração própria (2020).

Para criar estas tabelas em um banco de dados SQLite3 - ou qualquer banco de dados suportado por *Ruby on Rails* - executa-se o seguinte código em linguagem *ruby*:

```
class CreateRedes < ActiveRecord::Migration[7.0]
 def change
   create_table :redes do |t|
     t.string :nome
     t.string :token

     t.index :nome, unique: true
     t.index :token, unique: true
     t.timestamps
   end
 end
end
```

A função “create_table” executará o comando “SQL CREATE TABLE” e incluirá as colunas definidas com suas definições, por exemplo as colunas nome e *token* de uma rede devem apenas conter valores únicos. Isso adiciona segurança ao sistema, uma vez que o banco de dados garantirá que nunca existirão duas redes com mesmo nome ou *token*.

Depois de criada a tabela redes, podemos criar a tabela registro, também executando código *ruby*, sem necessidade de escrever SQL diretamente:

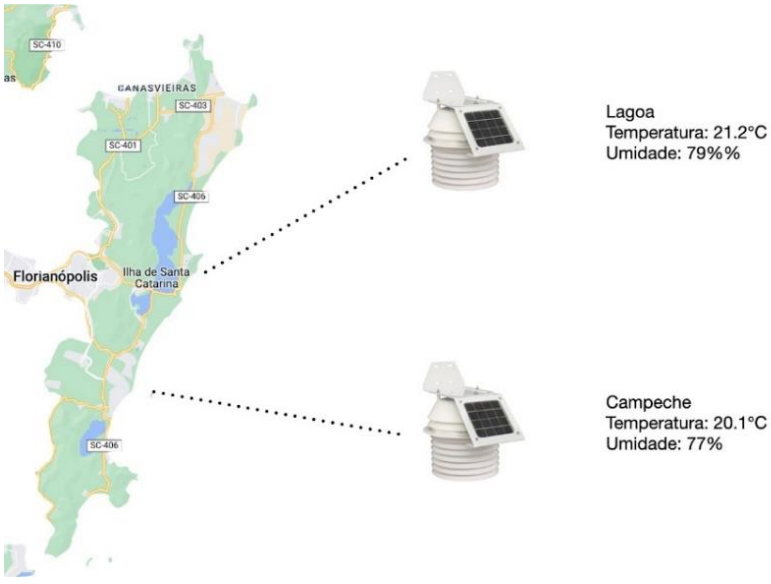
```
class CreateRegistros < ActiveRecord::Migration[7.0]
 def change
   create_table :registros do |t|
     t.string :nome
     t.string :valor
     t.integer :tipo
     t.references :rede, null: false, foreign_key: true

     t.index [:nome,:rede_id], unique: true
     t.timestamps
   end
 end
end
```

Neste código observa-se que a coluna rede é do tipo “references”, que denota uma ligação com outra tabela (redes). Também especifica-se usando o parâmetro “unique: true” que mesmo que possam existir registros com nomes repetidos no banco de dados, não podem existir registros com o mesmo nome e rede_id.

3.3.1 Exemplo de banco de dados

Um servidor da plataforma deste projeto utilizado para estações meteorológicas em que seus usuários decidiram separar as leituras de cada estação por redes. Na situação em que duas estações apresentam as leituras conforme apresentadas na Figura 10, as tabelas do banco de dados deste servidor conteriam os dados conforme as tabelas que seguem.

Figura 10 – Exemplo de estações meteorológicas



Fonte: Adaptado de imagens da internet (2022).

A Tabela 2, chamada “redes” no banco de dados, conforme padrão do *Ruby on Rails* para uma tabela que armazena modelos do tipo “rede”, possui uma linha para cada rede criada pelos usuários da plataforma neste exemplo, armazenando o nome escolhido, seu token gerado automaticamente e datas de atualização e criação de cada rede.

Tabela 4 – Tabela "redes" do banco de dados

id	nome	*token*	*created_at*	*updated_at*
1	Campeche	3SfLHM88hLoCRJoqRMzsGWkt	1656276478	1656276478
2	Lagoa	ymckKgDD2QUi65qvJnLQVhax	1656576837	1656576837

Fonte: Elaboração própria (2022).

Para cada estação meteorológica são armazenados dois parâmetros: temperatura e umidade. Logo, a Tabela 3, que simula o estado da tabela do banco de dados chamada “registros” possui um registro por rede para cada parâmetro armazenado.

Tabela 5 – Tabela "registros" do banco de dados.

Id	nome	valor	tipo	rede_id	*created_id*	*update_id*
1	temperatura	20.1	*float*	1	1656276478	1656277078
2	umidade	0.77	*float*	1	1656276478	1656277010
3	temperatura	21.2	*float*	2	1656576837	1656277103
4	umidade	0.79	*float*	2	1656576837	1656277122

Fonte: Elaboração própria (2022).

Observa-se que o esquema utilizado atende aos requisitos do projeto, oferecendo a possibilidade de que sejam criados diversos registros com o mesmo nome, porém ainda sejam facilmente diferenciados por pertencerem a redes diferentes.

3.4 Interface do aplicativo *web*

Podem-se identificar as funcionalidades necessárias para o aplicativo *web*:

1. Criar novas redes;
2. Listar todas as redes;
3. Editar redes existentes;
4. Excluir redes existentes;
5. Listar todos os registros de um *token*.

Em *Ruby on Rails*, os componentes da aplicação que definem a interface com o usuário são chamados *Views*, conforme a arquitetura de *software* MVC descrita na seção 2.4.2. Pode-se criar uma página (*View*) para oferecer a interface para cada funcionalidade apresentada acima, para simplificar pode-se incluir a opção de excluir uma rede – que também exclui todos os seus registros – na página que exibe todos os registros de uma rede.

As páginas do site são escritas em *HyperText Markup Language* (HTML) e usam *Tailwind* CSS (TAILWIND LABS INC, 2022), com os elementos que dependem de conteúdo dinâmico calculados em tempo real em pequenos blocos de código *Ruby* entre o HTML, como mostra o código abaixo.

```
<p class=my-5">
   <strong class="block font-medium mb-1">Nome:</strong>
   <%= rede.nome %>
 </p>

 <p class="my-5">
   <strong class="block font-medium mb-1">Token:</strong>
   <%= rede.token %>
 </p>
```

Neste exemplo, os elementos “rede.nome” e “rede.token” obtêm seus valores do banco de dados através do *controller*, e são preenchidos pelo servidor quando o navegador do usuário envia a requisição para a página.

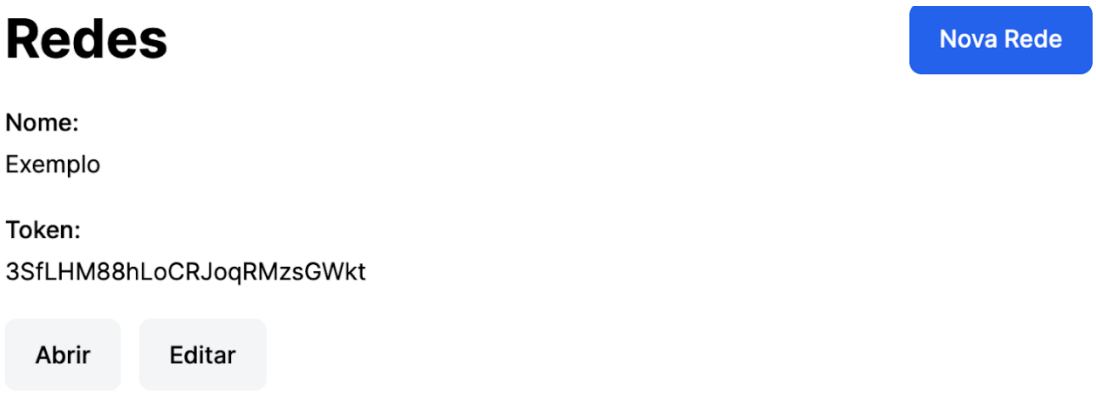
Seguindo essa estrutura, foram desenvolvidas as seguintes páginas para o site mostradas nas figuras a seguir.

O nome da rede deve ser único, caso o valor do campo da Figura 11 seja preenchido com um valor já existente o usuário receberá um erro e poderá escolher outro nome.

Figura 11 – Página para criação de uma rede (funcionalidade 1)

A lista exibida na Figura 12 exibe as redes cadastradas, no caso existe apenas uma chamada exemplo. Também nesta página o usuário tem a opção de cadastrar uma nova rede.

Figura 12 – Página que lista as redes cadastradas (funcionalidade 2)



A página de edição é muito similar à página para criação de uma rede. Percebe-se na Figura 13 que as diferenças se encontram na indicação do *token* para facilitar a confirmação da rede que está sendo modificada e na opção “abrir”, que redireciona o usuário a página da rede que está sendo modificada.

Figura 13 – Página de edição de uma rede existente (funcionalidade 3)

A lista de registros exibe o valor atual de todos os registros pertencente a uma rede. Na Figura 14, a rede “Exemplo” possui um registro chamado “temperatura”, com valor “21.2”.

Figura 14 – Página com a lista de registros de uma rede (funcionalidade 5) e exclusão (funcionalidade 4)

Além de elementos necessários à sua funcionalidade principal, como entradas de texto e botões de confirmação, as páginas contêm também botões de navegação para facilitar o acesso a todas as funcionalidades.

3.5 Implementação da versão pública

Com o modelo do banco de dados definido, e desenvolvidas as páginas do site, o aplicativo *web* pode ser hospedado em um servidor. O objetivo deste projeto é oferecer este código para qualquer pessoa conseguir hospedar sua versão da plataforma, porém, para tornar o desenvolvimento inicial ainda mais acessível, foi implementada uma versão completamente pública e gratuita da plataforma em um servidor operado pelo autor do projeto. Esta versão pública é disponibilizada em http://idcparatodos.com.br e este endereço será utilizado como exemplo nas indicações de rota neste trabalho, como a página que lista todos os registros para uma rede com o *token* "e7624sSjCuTdfGdrbafY1CrN" estaria na rota "https://idcparatodos.com.br/redes/e7624sSjCuTdfGdrbafY1CrN". Reforça-se que todo o código desenvolvido para o site idcparatodos.com.br é público no repositório deste projeto e pode ser utilizado e modificado para criar até um produto comercial em outro site.

3.6 Rotas do site

Desenvolvidas as páginas, devem-se mapear as rotas que acessam cada página. Buscando rotas claras, estabeleceram-se as rotas mostradas na Tabela 6, onde na coluna rotas está indicado o conteúdo que sucede a primeira barra "/" da *Uniform Resource Locator* (URL) do navegador. Por exemplo, “https://idcparatodos.com.br/redes” é indicado como "/redes", nos casos em que a rota contém um parâmetro (como o *token* de uma rede) ele é indicado por dois pontos ":" seguidos nome do parâmetro. Por exemplo, “https://idcparatodos.com.br/redes/e7624sSjCuTdfGdrbafY1CrN” é a rota “/redes/:token”.

Tabela 6 – Rotas do site

Rota	Página
/ ou /redes	Lista de todas as redes
/redes/new	Criar nova rede
/redes/:token	Exibe todos os registros da rede correspondente a este *token*
/redes/:token/edit	Editar a rede com este *token*

Fonte: Elaboração própria (2022).

3.7 Rotas da API

Os sistemas eletrônicos embarcados com recursos IoT que utilizarem a presente plataforma para seu componente de servidor demandam respostas mais simples que o HTML e *Cascading Style Sheets* (CSS) enviados ao navegador. Como discutido na seção 2, a comunicação entre o sistema embarcado e o servidor deve buscar redução dos recursos de rede e processamento utilizados. O uso do protocolo MQTT implicaria em grande complexidade adicional a plataforma, dificultando a modificação e expansão por outros desenvolvedores. Contudo, o ciclo de requisição e resposta HTTP implementado deve ser o mais eficiente possível, para isso foram definidas duas funções principais e suas rotas, como mostra a Tabela 7.

Tabela 7 – Rotas da API

Rota	Função
/redes/:token/:registro	Retorna exatamente o valor do registro especificado pelo nome no parâmetro :registro da rota pertencente a rede correspondente ao :*token*. Retorna o *string* "Registro não encontrado" caso não exista registro com este nome na rede.
/redes/:token/:registro/:valor	Escreve o valor definido em :valor ao :registro pertencente a rede correspondente ao :*token*. O registro é criado caso não exista. Retorna o *string* "OK".

Fonte: Elaboração própria (2022).

3.8 Lógica do controlador

No exemplo abaixo, o controlador recebe uma requisição para a escrita de um registro e atualiza o registro para o valor solicitado na requisição. Caso o registro ainda não exista, o controlador cai na condição “else” do código e cria um novo registro com o valor solicitado.

```
def update_simplificado
   @registro = @rede.registros.find_by_nome(params[:id])
   if @registro
     @registro.update(valor: params[:valor])
   else
     @rede.registros.create(nome: params[:id], valor: params[:valor], tipo:
'string')
   end
   render plain: "OK"
 end
```

3.9 Exemplo de requisições na API

No cenário estabelecido no exemplo da seção 3.3.1, foram apresentadas tabelas com dados das leituras das estações meteorológicas. Com a API da Tabela

7, pode-se descrever a sequência de ações que levam o banco de dados ao estado preenchido da Tabela 5, chamada "registros" do banco de dados. Primeiramente o usuário deve criar as redes de cada estação: "Lagoa" e "Campeche", que receberão um *token* seguro único. Programando o sistema embarcado de cada estação para usar o *token* de sua rede correspondente, as seguintes requisições da Tabela 8 devem ser feitas pela estação "Campeche" para que os dados armazenados no banco referentes a esta rede fiquem iguais aos do exemplo da seção 3.3.1.

Tabela 8 – Exemplo de requisições e suas respostas

Requisição	Resposta
http://idcparatodos.com.br/redes/3SfLHM88hLoCRJoqRMzsGWkt/registros/temperatura/20.1	"OK"
http://idcparatodos.com.br/redes/3SfLHM88hLoCRJoqRMzsGWkt/registros/umidade/0.77	"OK"

Fonte: Elaboração própria (2022).

3.10 Biblioteca Arduino para ESP32 e ESP8266

As requisições de escrita de registros documentadas no exemplo da seção anterior (seção 3.9) são de fácil implementação em qualquer dispositivo ou *software* capaz de atuar como cliente HTTP. Para os dispositivos acessíveis das linhas ESP32 e ESP8266, que são amplamente utilizados em produtos comerciais e em soluções didáticas, foi proposto o desenvolvimento de uma biblioteca compatível com o Ambiente de Desenvolvimento Integrado (do inglês, *Integrated Development Environment* – IDE) Arduino.

3.10.1 Definição das funções da biblioteca

A biblioteca, nomeada IDC (Internet Das Coisas), deve fornecer funções que permitam a implementação da comunicação com o servidor de idcparatodos.com.br da forma mais simples possível para iniciantes no aprendizado de sistemas embarcados e recursos IoT. Espera-se que desenvolvedores mais

avançados utilizarão diretamente as rotas de API descritas previamente, possibilitando cenários mais complexos.

Baseando-se em funções existentes na biblioteca padrão do Arduino, propôs-se uma estrutura similar e simplificada para esta biblioteca:

a) Um inicializador do tipo IDC que recebe o *token* correspondente a rede que será usada no programa;

b) Um método de leitura de registros que recebe o nome o registro a ser lido;

c) Um método de escrita de registro que recebe o nome do registro e o valor a ser escrito

Como exemplo da implementação proposta, tem-se:

a) Inicialização:

```
IDC idc("wvgY8Pi4R4xUng1ugC9Uy5Xy");
```

b) Leitura:

```
idc.lerRegistro("contagem");
```

c) Escrita:

```
idc.escreverRegistro("contagem", 23);
```

3.10.2 Implementação da biblioteca

Para oferecer a biblioteca como projetada acima, são definidas as funções implementadas em um arquivo .h com o nome da biblioteca:

```
class IDC {
 public:
   IDC(String token);
   String lerRegistro(String nome);
   boolean escreverRegistro(String registro, String valor);
 private:
   String _token;
   HTTPClient http;
};
```

Neste código em C++ observa-se a definição do inicializador da classe IDC, que recebe o *token* como argumento do tipo “String” e os demais métodos públicos conforme a especificação da biblioteca. Além disso, incluem-se duas definições privadas: "_token", do tipo “String” e "http", do tipo “HTTPClient”. A primeira será

utilizada para armazenar o valor do *token* que for passado ao inicializador e é prefixada por um caractere "_" para evitar conflitos com o nome do argumento do inicializador. A segunda define uma instância da classe “HTTPClient”, que é a biblioteca usada para fazer as requisições HTTP.

Além do trecho de código acima, que define a funcionalidade da classe IDC, é necessário que sejam declaradas as dependências da biblioteca, chegando a versão final do arquivo “IDC.h”:

```
#ifndef IDC_h
#define IDC_h

#include "Arduino.h"
#include <WiFiMulti.h>
#include <HTTPClient.h>

class IDC {
 public:
   IDC(String token);
   String lerRegistro(String nome);
   boolean escreverRegistro(String registro, String valor);
 private:
   String _token;
   HTTPClient http;
};

#endif
```

Observa-se neste arquivo completo que todo seu conteúdo está dentro da condição "#ifndef IDC_h", e que logo na segunda linha é definido “IDC_h”, tornando falsa a resolução de "#ifndef IDC_h" se chamando novamente. Trata-se de uma boa prática que evita que a biblioteca seja incluída no código múltiplas vezes, gerando erros. Um exemplo de situação solucionada por estas definições é se, em um dado programa de Arduino, o usuário incluir uma biblioteca que depende da biblioteca IDC (e, portanto, possui em seu código a linha “#include <IDC.h>”) e, em seguida, tentar incluir novamente a biblioteca IDC, sem saber que a biblioteca anterior já o fez. Como todo o código de IDC está envolto pela condicional "#ifndef IDC_h", que agora chamada depois de ter sido executada a segunda linha contendo "#define IDC_h", esta duplicidade não gera erros ou colisões, a segunda chamada para #include apenas pula ao final do arquivo, para a linha "endif", sem fazer mais nada.

Com os métodos definidos e organizados no arquivo “IDC.h”, o código de cada função pode ser programado no arquivo “IDC.cpp”. O inicializador de uma instância IDC é implementado com sua única responsabilidade de armazenar na variável privada “_token” conforme o código seguinte:

```
IDC::IDC(String token) {
 _token = token;
}
```

A função “lerRegistro()” retorna um valor *string*. Existem alguns cenários de erro para esta leitura, como a interrupção da conexão com a internet ou uma tentativa de ler um registro inexistente. Para os casos em que a requisição foi malsucedida, identificados pelo código de retorno HTTP retornado ter um valor diferente de "HTTP_CODE_OK" (200), a função retorna uma *string* vazia "". No caso de não existir o registro procurado, o erro é percebido no servidor, que retorna a *string* "Registro não encontrado." ao dispositivo. Neste caso não é necessário que a função “lerRegistro” faça qualquer tratamento de exceção, a *string* é retornada normalmente e será tratada pelo usuário ou, espera-se, impressa em uma porta serial para servir de indicação do erro. Segue a implementação desta função:

```
String IDC::lerRegistro(String registro) {
 String valor = "";
 String url = "http://idcparatodos.com.br/redes/";
 url += _token;
 url += "/registros/";
 url += registro;
 http.begin(url);
 int httpCode = http.GET();
 if (httpCode > 0) {
   if (httpCode == HTTP_CODE_OK) {
     String payload = http.getString();
     valor = payload;
   }
 }
 http.end();
 return valor;
}
```

Similar à função de leitura, a função de escrita “escreverRegistro” executa uma requisição ao servidor e trata a resposta recebida. No caso de escrita, pode-se eliminar a situação de erro em que o registro para o qual se deseja escrever não existe, uma vez que se pode imediatamente criá-lo e inicializá-lo com o valor desejado. Restam os erros em que, por uso de um *token* inexistente ou interrupção da comunicação com internet não foi recebida a confirmação do servidor de que o registro foi armazenado corretamente no banco de dados. Nestes casos, resta apenas indicar ao usuário da biblioteca que não aconteceu a escrita desejada, identificando a situação pela ausência da confirmação do servidor - exatamente a *string* "OK" - na resposta:

```
boolean IDC::escreverRegistro(String registro, String valor) {
 boolean sucesso = false;
 String url = "http://idcparatodos.com.br/redes/";
 url += _token;
 url += "/registros/";
 url += registro;
 url += "/";
 url += valor;
 http.begin(url);
 int httpCode = http.GET();
 if (httpCode > 0) {
   if (httpCode == HTTP_CODE_OK) {
     String payload = http.getString();
     if(payload == "OK") sucesso = true;
   }
 }
 http.end();
 return sucesso;
}
```

Existe uma expectativa, principalmente entre estudantes que estão começando a aprender programação com Arduino, que a IDE destaque com cores diferentes classes conhecidas e seus métodos. Para que isso aconteça com a biblioteca implementada neste item, deve ser adicionado mais um arquivo de texto à pasta contendo os arquivos “IDC.h” e “IDC.cpp”. O arquivo "keywords.txt" deve conter os termos para as classes e métodos definidos pela biblioteca. O leiaute do arquivo determina que deve ser escrito um termo por linha e que cada termo seja seguido de um único caractere “tab” (\t) e o seu tipo. As opções de tipo são "KEYWORD1" para classes e "KEYWORD2" para métodos. Conforme esta especificação, segue o conteúdo do arquivo "keywords.txt" para a biblioteca IDC:

```
IDC	KEYWORD1
lerRegistro	KEYWORD2
escreverRegistro	KEYWORD2
```

Esta implementação atende o objetivo de oferecer funções simples de usar e compreender, com o foco em usuários iniciantes. São apenas três funções, sem redundância das informações necessárias nos argumentos de cada uma.

4 RESULTADOS

Após os testes iniciais da plataforma desenvolvida, conduzidos pelo autor junto com o orientador deste trabalho, os primeiros usuários da plataforma foram alunos do curso de Engenharia Mecatrônica. Foi oferecido acesso à versão pública do site IDC para Todos em “idcparatodos.com.br” e acompanhamento do autor para que os alunos avaliassem o uso da plataforma em dois projetos do curso.

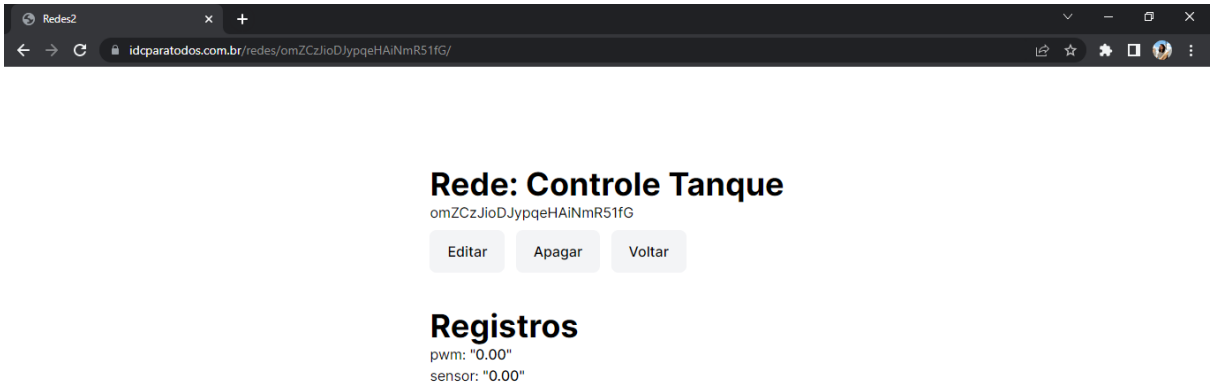
4.1 Aplicação em protótipo de planta de controle de nível

Em uma primeira experiência de aplicação da plataforma, e como teste de sua viabilidade e adequação ao objetivo de ser fácil de aprender, foi proposta a sua utilização em um projeto das disciplinas de Projeto Integrador 3 e de Aplicações de Instrumentação Virtual, do Curso de Graduação em Engenharia Mecatrônica do IFSC. Segundo Silvano, Silva e Padilha (2022, p. 2), o trabalho desenvolvido “tem como objetivo apresentar a comunicação entre uma interface VI e página *web* IDC para todos, simulando o controle de PWM e nível de uma bancada de tanque”.

O autor desta plataforma orientou os alunos sobre o uso da versão pública, e eles desenvolveram o projeto com acompanhamento do Prof. Francisco Edson Nogueira de Mélo (orientador deste TCC).

No decorrer do desenvolvimento do projeto, observou-se interesse e fácil compreensão dos alunos sobre o funcionamento da plataforma. O protótipo do sistema eletrônico utilizado simula as leituras de sensores com potenciômetros, favorecendo a experimentação com o envio destes valores para o servidor do aplicativo *web* “idcparatodos.com.br” (Figura 15).

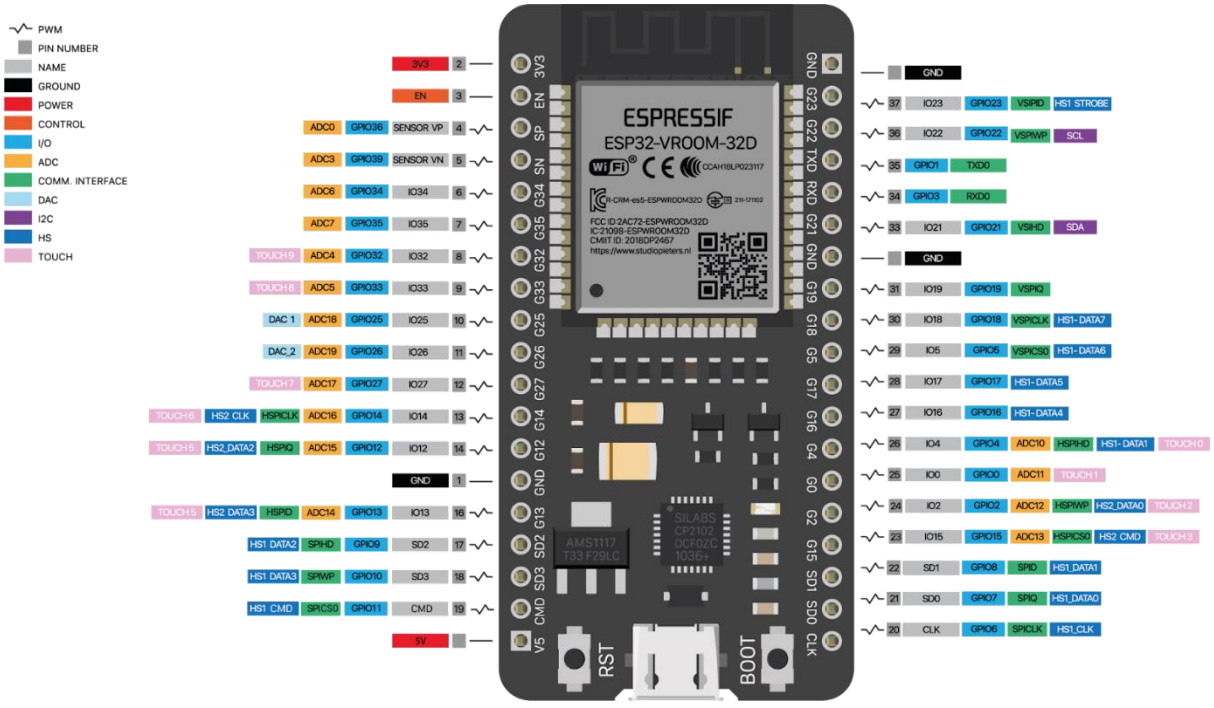
Figura 15 – Servidor do aplicativo *web* idcparatodos.com.br para a rede Controle Tanque



Fonte: Silvano, Silva e Padilha (2022, p. 2).

No projeto feito pelos alunos foi utilizado o Módulo WIFI ESP 32 *NodeMcu*, conforme a Figura 16, que mostra descrição dos pinos deste módulo.

Figura 16 – Descrição dos pinos Módulo WIFI ESP 32 *NodeMcu*



Fonte: Silvano, Silva e Padilha (2022, p. 3).

Para comunicação com a página *web* foi utilizada a biblioteca IDC, apresentada na seção 3.10. A biblioteca é inicializada criando-se um objeto chamado “idc”, passando-se como parâmetro ao construtor o *token* referente à rede utilizada, conforme a parte do código mostrado na Figura 17.

Figura 17 – Parte do código que mostra o uso da biblioteca “IDC.h”

tanque_AIV

```
#include <IDC.h>
#include <WiFi.h>

#define PIN_SENSOR 34
#define PIN_PWM 35
#define MAX_VALUE_ANALOG_READ 4095

WiFiMulti wifiMulti;
IDC idc("omZCzJioDJypqeHAiNmR51fG"); //token

float sensor, pwm;

```

Fonte: Silvano, Silva e Padilha (2022, p. 3).

Para efetuar os registros na página é utilizado o método “idc.escreverRegistro()”, conforme a parte do código mostrada na Figura 18. Este método recebe como parâmetros o nome da variável criada na rede e o valor a ser atribuído a essa variável, ambas do tipo “String”.

Figura 18 – Parte do código que mostra o uso do método idc.escreverRegistro()

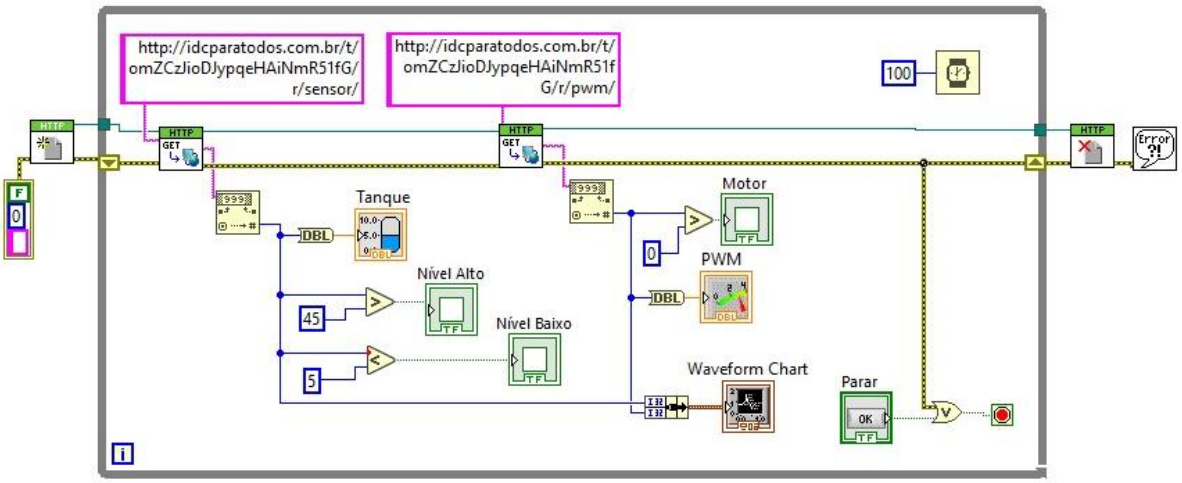
```
void loop() {
  //Leitura dos potenciometros e tratamento dos dados
  sensor = analogRead(PIN_SENSOR);
  sensor = map(sensor, 0, MAX_VALUE_ANALOG_READ, 0, 50);
  pwm = analogRead(PIN_PWM);
  pwm = map(pwm, 0, MAX_VALUE_ANALOG_READ, 0, 100);
  String sensor_str = String(sensor);
  String pwm_str = String(pwm);
  //Registrando dados no server IDC
  idc.escreverRegistro("sensor", sensor_str);
  idc.escreverRegistro("pwm", pwm_str);

  delay(500);
}
```

Fonte: Silvano, Silva e Padilha (2022, p. 4).

A interface do protótipo foi desenvolvida pelos alunos no ambiente de desenvolvimento LabVIEW, que é um ambiente de programação para desenvolvimento de sistemas automatizados, bem como seu controle e supervisão. Segundo a empresa desenvolvedora NI (2019, p. 1), o LabVIEW “fornece um ambiente de desenvolvimento de aplicações de fácil uso, projetado especificamente para atender às necessidades dos engenheiros e pesquisadores”. A empresa também aponta a facilidade de integrar diversas plataformas, possibilitando que, uma vez integrado com o LabVIEW, o IDC para Todos possa ser usado com essas outras plataformas: “Através de suas poderosas funcionalidades, o LabVIEW pode ser facilmente integrado a uma grande variedade de plataformas de *hardware* ou *software*” (NI, 2019, p. 1).

A Figura 19 mostra os blocos de conexões feito no LabVIEW para funcionamento da interface. Observa-se a interação com o servidor da plataforma IDC para Todos em dois blocos identificados por “HTTP GET”, em que o LabVIEW faz requisições ao servidor da plataforma para receber o valor de dois registros: “sensor” e “pwm”.

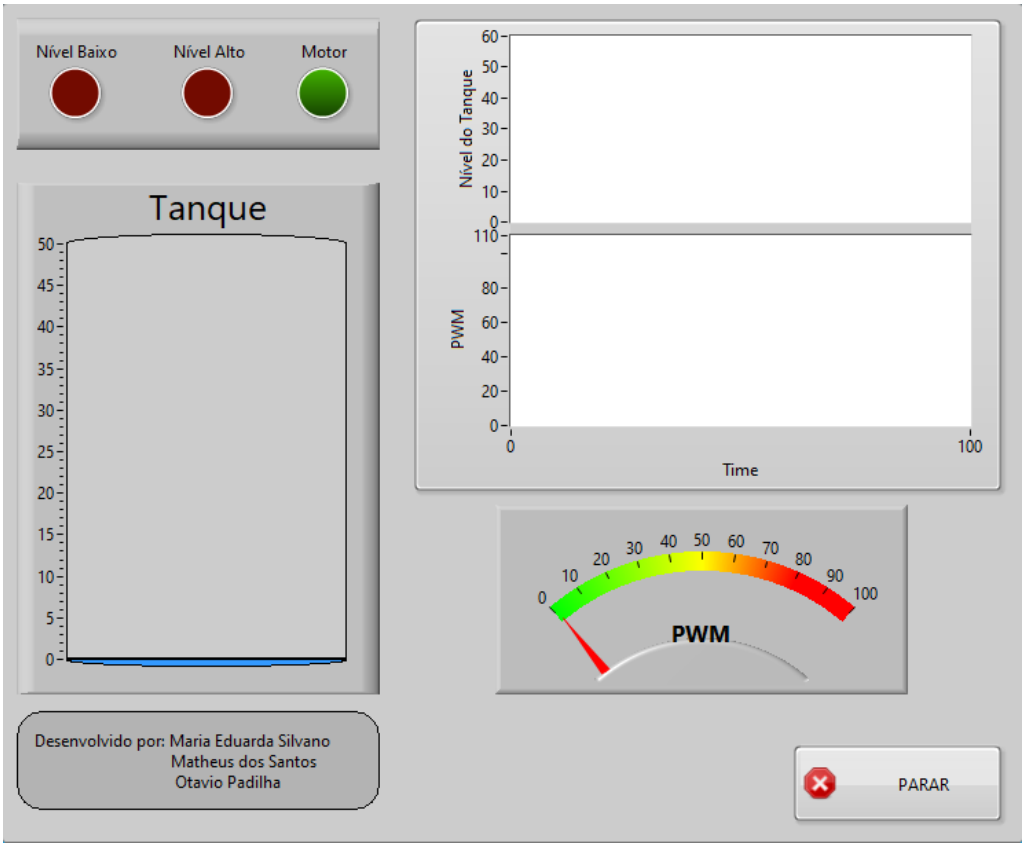
Figura 19 – Blocos de conexões feito no LabVIEW para funcionamento da interface



Fonte: Silvano, Silva e Padilha (2022, p. 5).

A Figura 20 mostra a interface de visualização dos dados do protótipo desenvolvido, através de VI do LabVIEW, que são atualizados conforme os valores recebidos do servidor IDC para Todos, após mínima adequação dos valores para sua exibição nos elementos de interface.

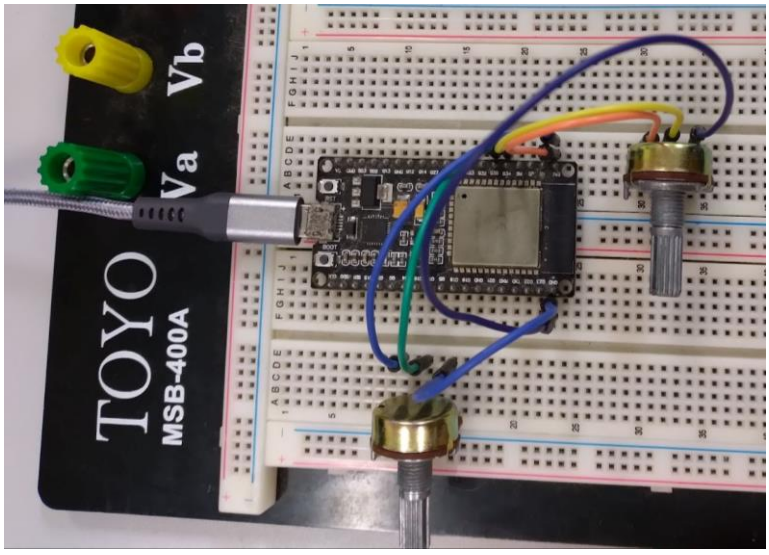
Figura 20 – Interface de visualização dos dados do protótipo



Fonte: Silvano, Silva e Padilha (2022, p. 5).

Para simular a variação do PWM e nível de um tanque foram utilizados 2 potenciômetros conectados aos pinos 34 e 35 da ESP 32 para um controle manual. A Figura 21 mostra o protótipo montado em uma matriz de contato.

Figura 21 – Protótipo montado em uma matriz de contato



Fonte: Silvano, Silva e Padilha (2022, p. 7).

Os alunos, em suas considerações finais conforme Silvano, Silva e Padilha (2022, p. 9), descrevem não terem encontrado dificuldades no uso da plataforma. Eles complementam:

> A utilização da biblioteca do IDC se mostrou muito simples e direta, sendo rapidamente possível registrar e ler dados na página, por consequência é possível rapidamente montar um protótipo com comunicação a internet sem dificuldades que normalmente se mostram em outras plataformas (SILVANO; SILVA; PADILHA, 2022, p. 9).

É também apresentada uma análise comparativa com outras soluções familiares aos alunos: “notou-se a grande facilidade e praticidade no uso da página web IDC para todos em comparação com outras presentes no mercado (exemplo: MQTT e *FireBase*).” (SILVANO; SILVA; PADILHA, 2022, p. 9).

4.2 Aplicação didática com célula de carga

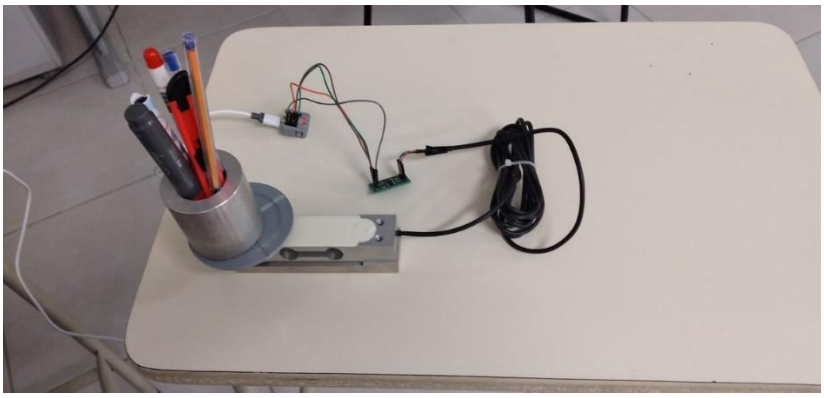
Em um projeto do curso de Engenharia Mecatrônica, conforme Rzatki (2022), foi utilizada a plataforma IDC para Todos em “idcparatdodos.com.br” na integração de um sistema de medição de uma célula de carga, utilizando um *hardware* ESP32 da *Espressif*, para aquisição e processamento de valores utilizando o LabVIEW para sinalização e registro dos resultados. A Figura 22 mostra a rede Balança IOT criada no servidor do aplicativo *web* “idcparatodos.com.br” para envio e armazenamento dos valores medidos no experimento.

Figura 22 – Servidor do aplicativo *web* idcparatodos.com.br para a rede Balança IOT

Fonte: Adaptado de Rzatki (2022).

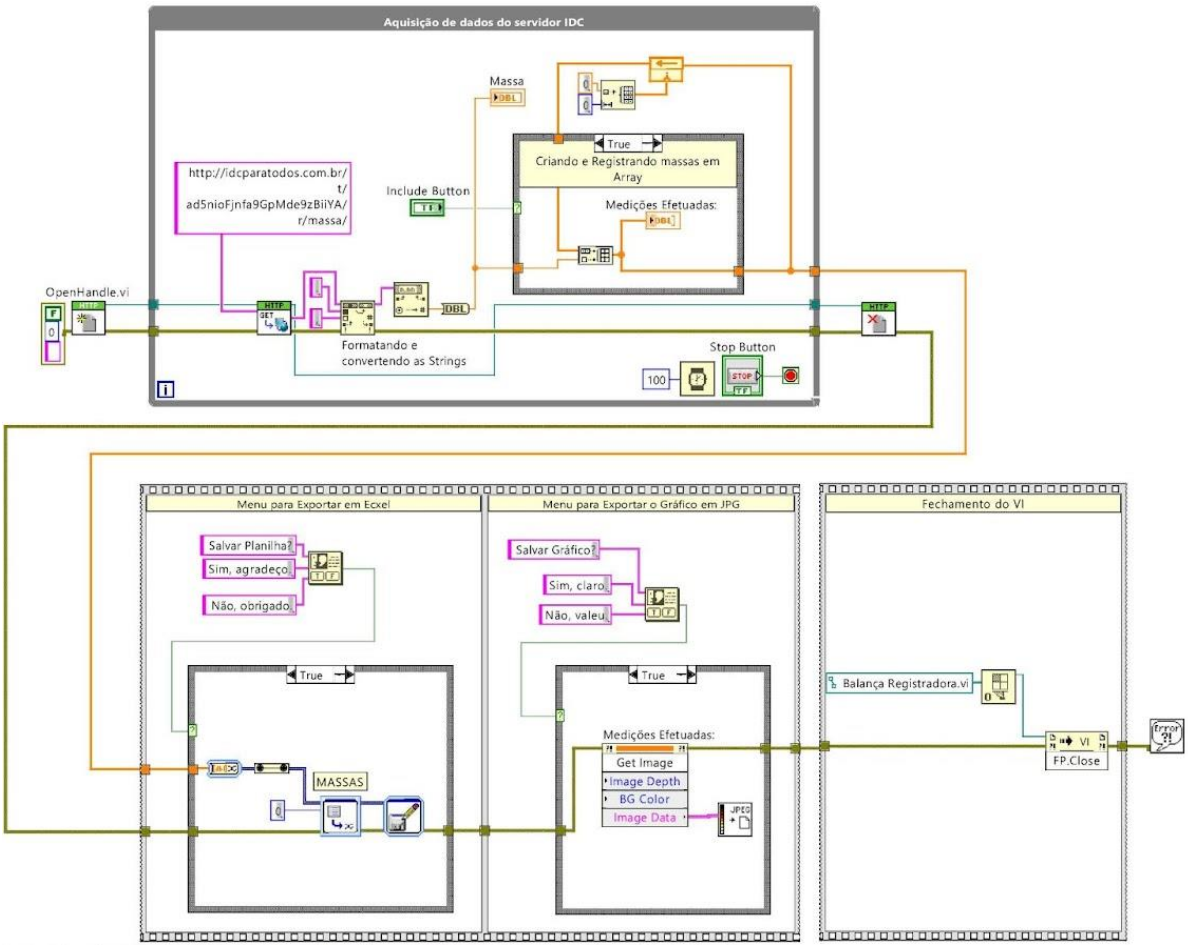
A Figura 23 mostra o protótipo do sistema de medição de uma célula de carga elaborado pelo aluno.

Figura 23 – protótipo do sistema de medição de uma célula de carga



A Figura 24 destaca o ciclo principal do instrumento virtual e exibe em um mostrador analógico que obtém as leituras da célula de carga usando o servidor IDC para Todos, no bloco identificado como “HTTP GET”.

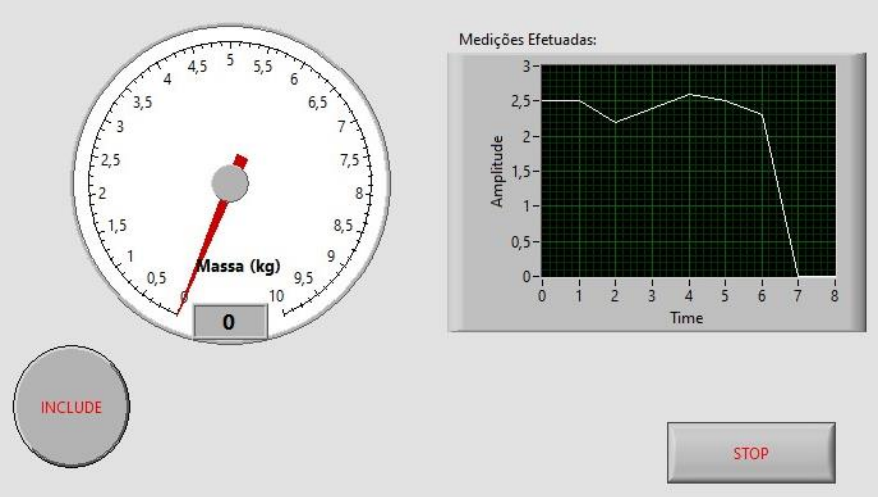
Figura 24 – ciclo principal do instrumento virtual



Fonte: Rzatki (2022, p. 11).

As medidas são então armazenadas em um vetor de dados cada vez que o operador do instrumento virtual da aplicação célula de carga pressiona o botão circular “include”, conforme mostra a Figura 25. Ao final da execução, o operador pressiona o botão “STOP” para sair do ciclo principal de armazenamento de dados, e tem a opção de exportar as leituras salvas e uma imagem do gráfico.

Figura 25 – Instrumento virtual da aplicação célula de carga



Fonte: Rzatki (2022, p. 16).

Como resultado, após todas as configurações, calibrações e montagens completas, foram feitos testes com variados elementos de formas e massas diferentes, conforme mostra a Figura 26.

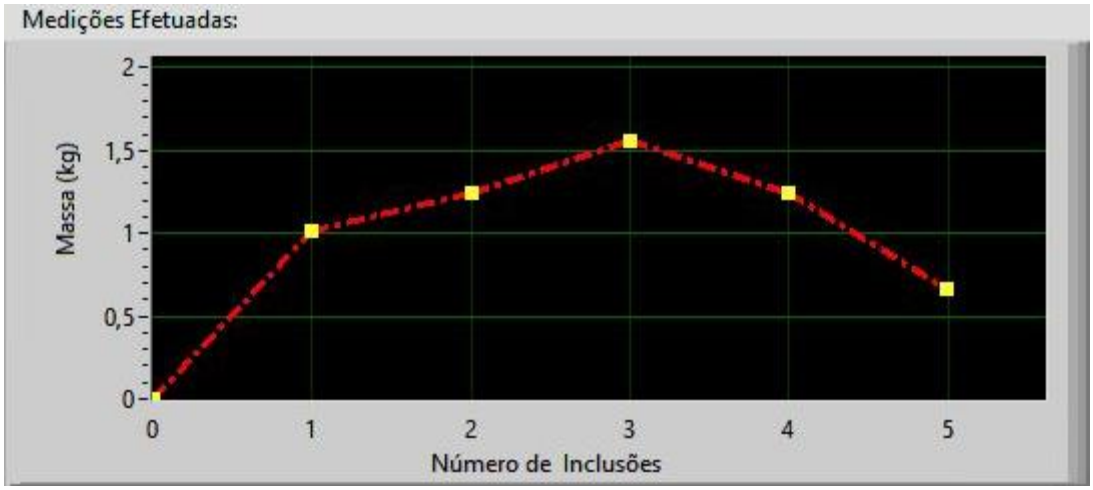
Figura 26 – Lista de massas

Massa (kg)
0
1,01
1,24
1,56
1,25
0,67

Fonte: Adaptado de Rzatki (2022).

A Figura 27 mostra o gráfico das leituras efetuadas.

Figura 27 – Gráfico das leituras efetuadas



Fonte: Rzatki (2022, p. 20).

Ao fim do trabalho, o autor conclui quanto ao servidor IDC que:

> Este se demonstrou extremamente útil para cumprir o objetivo proposto de captar os dados da balança e transmiti-los pela internet, uma vez que a interface é plenamente compatível e simples de conectar entre a ESP32 e o Labview, e por ser baseada no protocolo HTTP. (RZATKI, 2022, p. 20).

4.3 Uso em aplicações didáticas

Os primeiros resultados indicam que a plataforma é fácil de usar, atendendo um de seus objetivos principais. Também foram percebidos pontos para desenvolvimento futuro, conforme Silvano, Silva e Padilha (2022, p. 10), “Um pequeno empecilho a se considerar se dá pelo fato de ter que atualizar a página da rede sempre que é mandado um dado novo, a atualização não ocorre automaticamente”. Este ponto foi incluído na seção 5.1 deste trabalho.

Observa-se que, além da facilidade de uso, a plataforma deste trabalho oferece algumas vantagens quando comparada com soluções existentes para uso em sala de aula. A versão pública da plataforma pode ser acessada imediatamente sem nenhuma configuração, instalação de aplicativos ou criação de cadastros com e-mail e senha. Para projetos em que múltiplas equipes ou turmas de alunos desejem trabalhar com o mesmo conjunto de dados, pode ser criada uma rede pelo professor e adicionados ilimitados dispositivos de alunos a esta rede. Isto é possível pois o único “custo” (a plataforma é completamente gratuita, custo é usado em termos de recursos limitados do servidor) são as requisições feitas ao servidor.

Finalmente, aponta-se o fato de todo o código da plataforma estar disponível publicamente como vantagem para o seu uso em sala de aula. Espera-se que alunos interessados possam estudar e modificar esse código.

4.4 Uso em aplicações gerais

Adquirir um sistema de servidor para projetos de internet das coisas implica em riscos que devem ser considerados pelos projetistas. A empresa que vende o sistema pode mudar seus preços, impor novos limites e até descontinuar completamente a oferta do serviço de servidor para internet das coisas. Estes riscos podem ser reduzidos em projetos de grande valor comercial através de contratos de disponibilidade futura do servidor e limitação da variação dos preços cobrados. Projetos menores e não comerciais ficam sem opções em relação às mudanças impostas pela empresa de servidor. Nestes casos, espera-se que a plataforma desse trabalho seja uma alternativa viável, oferecendo um sistema de servidor que pode ser executado completamente em um computador comum. A plataforma deste trabalho também pode ser instalada em um servidor comercial, e, como não faz uso de funções proprietária, pode ser instalada no servidor de qualquer empresa. Limita-se assim a dependência a uma empresa específica. Caso ocorram alterações na oferta da empresa, um usuário pode apenas levar o mesmo código de sua versão modificada da plataforma para outra empresa de servidor.

Dentre os critérios usados para se decidir qual solução de servidor utilizar, espera-se que esta plataforma se destaque por ser código aberto e utilizar somente componentes código aberto; não impor nenhum limite artificial na quantidade de dados armazenados, leituras ou escritas que são feitas do servidor; usar um esquema de requisições muito simples, porém seguro devido a inclusão do *token*; e oferecer uma versão pública completamente gratuita que permite prototipar e validar rapidamente seu uso em determinado projeto.

5 CONSIDERAÇÕES FINAIS

O desenvolvimento da plataforma objeto deste projeto, e de sua versão pública gratuita, atingiu nível funcional e todo seu código foi disponibilizado publicamente em IDCPARATODOS (2022), conforme descrito na definição do problema e especificado nos objetivos do trabalho. Para isso foram avaliados sistemas de servidor para internet das coisas utilizando-se do conhecimento adquirido nas matérias do curso que abordaram redes industriais, sistemas embarcados, programação e projeto de sistemas eletrônicos. Tudo isso pode ser contextualizado com base na experiência adquirida nos diversos projetos realizados durante o curso de Engenharia Mecatrônica.

A solução desenvolvida oferece uma interface descomplicada, acessada de forma coerente, sempre através de requisições HTTP que aderem ao mesmo formato em todos os programas e dispositivos que se comunicam com o servidor, inclusive o navegador *web*.

A biblioteca para ESP32 e a disponibilidade de uma versão gratuita da plataforma tornaram acessível a experimentação com o sistema.

A plataforma pôde passar por uma validação inicial de seu uso no cenário acadêmico. Os alunos foram bem-sucedidos na implementação de todos os recursos da plataforma, além da integração com o LabVIEW.

Pretende-se manter disponível a versão pública da plataforma em idcparatodos.com.br, e todo seu código em IDCPARATODOS (2022) para qualquer desenvolvedor interessado.

5.1 Sugestões para trabalhos futuros

Este trabalho apresenta uma plataforma em desenvolvimento, para a qual novos recursos estão sendo projetados e adicionados.

A versão publica da plataforma lista todas as redes criadas. Esta configuração pode ser a mais simples e interessante para desenvolvedores que estão hospedando sua própria versão do servidor, mas pelo potencial de uso da versão publica identificado durante o desenvolvimento da plataforma, acredita-se que

implementar a opção de remover uma rede da listagem publica oferecerá mais segurança aos usuários que optarem por usar apenas a versão pública. Esta opção pode ser implementada adicionando-se uma coluna booleana “não listado” a tabela que armazena as redes, e, no momento da criação de uma nova rede oferecer ao usuário a opção de marcá-la como não listada. Essa rede então não apareceria na lista, e poderia ser acessada apenas pelo seu *token*, que poderia ser oferecido como um arquivo de texto que o usuário salvaria em seu computador ou enviado por e-mail. Recomenda-se que esta opção seja oferecida apenas na criação de uma nova rede e não seja disponibilizada a opção de modificar uma rede existente para “não listada”, uma vez que nesse caso qualquer usuário poderia marcar a rede de outro como “não listada”.

A escrita de variáveis por uma interface no site, como botões “+” e “-” para alteração de valores dos registros numéricos e caixas de texto para *strings* foi abandonada em favor de que sempre se use as mesmas URLs para leitura e escrita dos valores, já demonstrando imediatamente para novos usuários como as leituras e escritas serão feitas pelos dispositivos embarcados, ferramentas como LabVIEW e aplicativos web. Esta escolha pode ser reconsiderada, porém considera-se importante que essa demanda venha de usuários da plataforma e sua implementação seja priorizada no contexto de todas as melhorias que estes usuários apontarem como sugestões.

Um aspecto da interface de registros, que não modifica o funcionamento da interface, apenas torna seu uso mais agradável, seria a atualização em tempo real de seus valores, eliminado a necessidade de recarregar a página para receber os valores mais atualizados de cada registro.

A capacidade de acesso ao histórico de valores dos registros pode ser útil para algumas aplicações, e pode ser facilmente implementada graças à escolha de banco de dados da plataforma, que pode armazenar os valores antigos sem mudança na sua estrutura de colunas, apenas a lógica de escrita passaria a incluir cópias dos registros a cada mudança e, para a leitura de um registro, seria retornada a cópia mais recente. Com esta melhoria, podem ser implementados gráficos e elementos de interface que apresentem visualmente aspectos como variação, máximo, e mínimo de cada registro.

O protocolo MQTT apresenta melhor eficiência comparado ao HTTP, além da redução do tamanho das mensagens trocadas entre o servidor e os dispositivos eletrônicos embarcados. Uma vez que este trabalho já previa a implementação de uma API HTTP para a comunicação com navegadores da internet e outros *softwares* como LabVIEW, optou-se por utilizar o HTTP para comunicação com os dispositivos eletrônicos também. Esta escolha foi feita pela priorização da simplicidade do projeto final, que busca ser facilmente copiado e modificado por outros desenvolvedores. A inclusão de suporte ao protocolo MQTT pode ser interessante para tornar este projeto viável em situações de recursos de rede limitados e projetos que usam baterias pequenas, porém deve ser pensada de modo que não sacrifique a simplicidade do uso e ensino da plataforma e do seu código.

REFERÊNCIAS

AWS. **O que é a IoT?**. Amazon Web Services Inc. 2022a. Disponível em: https://aws.amazon.com/pt/what-is/iot/. Acesso em: 11 ago 2022.

AWS. **Módulo 4: Criar uma tabela de dados - Implantar uma aplicação Web e adicionar interatividade com uma API e um banco de dados**. Amazon Web Services Inc. 2022b. Disponível em: https://aws.amazon.com/pt/getting-started/hands-on/build-web-app-s3-lambda-api-gateway-dynamodb/module-four/. Acesso em: 10 jun 2022.

BANKS, Andrew (ed.); BRIGGS, Ed (ed.); BORGENDALE, Ken (ed.); GUPTA, Rahul (ed.). ***MQTT Version 5.0***. OASIS Standard. 07 mar. 2019. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Acesso em: 20 jun. 2022.

BLYNK. ***A fully integrated suite of IoT software***. Blynk Inc. 2022. Disponível em: https://blynk.io. Acesso em: 20 jun. 2022.

ESP RAINMAKER. ***Harness the Power of Cloud for Your Business with ESP RainMaker.*** Disponível em: https://rainmaker.espressif.com. Acesso em: 20 jun. 2022.

ESPRESSIF. ***ESP Rainmaker – Get Started***. Disponível em: https://rainmaker.espressif.com/docs/get-started.html. Acesso em: 20 de jun. 2022.

EUGSTER, Simon A. ***File:MQTT protocol example without QoS.svg***. 06 Jul. 2018. Disponível em: https://commons.wikimedia.org/wiki/File:MQTT_protocol_example_without_QoS.svg. Acesso em: 25 jun. 2022.

IDC. **Previsões da IDC para 2022 apontam crescimento de 8,2% para o mercado de TIC no Brasil**. 08 fev. 2022. Disponível em: https://www.idc.com/getdoc.jsp?containerId=prLA49041022. Acesso em: 05 jul. 2022.

NI, I**nstrumentação Virtual**. 05 mar. 2019. Disponível em: https://www.ni.com/pt-br/innovations/white-papers/06/virtual-instrumentation.html. Acesso em: 10 jun. 2022.

RAILS. **Compress the complexity of modern web apps**. 09 maio 2022. Disponível em: https://rubyonrails.org/. Acesso em: 10 jul. 2022.

RUBY. **O melhor amigo do programador.** Disponível em: https://www.ruby-lang.org/pt/. Acesso em: 10 jul. 2022.

RZATKI, Jorge R. B. **CÉLULA DE CARGA ONLINE: Unindo o Servidor IDC ao LabView**. Projeto da disciplina de Metrologia e de Aplicações Instrumentação Virtual. Florianópolis: IFSC, 2022. 25 p. Não publicado.

SCHUTT, K.; BALCI, O. ***Cloud software development platforms: A comparative overview***. 2016. *IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*. doi:10.1109/sera.2016.7516122.

SQLITE. ***About SQLite***. Disponível em: https://www.sqlite.org/about.html. Acesso em: 10 jul. 2022.

SILVANO, Maria E.; SILVA, Matheus dos S.; PADILHA, Otávio S. **Monitoramento e comando para controle de nível de tanque**. Projeto Integrador. Florianópolis: IFSC, 2022, 10 p. Não publicado.

TAILWIND LABS INC, ***Rapidly build modern websites without ever leaving your HTML***. Disponível em: https://tailwindcss.com. Acesso em: 10 jun. 2022.

W3TECHS. ***Usage statistics and market share of Linux for websites***. Disponível em: https://w3techs.com/technologies/details/os-linux. Acesso em: 10 jun. 2022.

WANG, Charlie. HTTP vs. MQTT: ***A tale of two IoT protocols***. *Google Cloud*. 26 nov. 2018. Disponível em: https://cloud.google.com/blog/products/iot-devices/http-vs-mqtt-a-tale-of-two-iot-protocols. Acesso em: 01 jul. 2022.

IDCPARATODOS. **IDC para todos**. Disponível em: https://github.com/idcparatodos/idcparatodos. Acesso em: 08 jul. 2022.