UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Guilherme Kraemer

COMPARAÇÃO ENTRE MÉTODOS DE INTEGRAÇÃO DE DADOS PARA SENSORES IOT

Santa Maria, RS
2023

Guilherme Kraemer

COMPARAÇÃO ENTRE MÉTODOS DE INTEGRAÇÃO DE DADOS PARA SENSORES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Defesa realizada por videoconferência.

Orientador: Prof. Osmar Marchi dos Santos

Santa Maria, RS
2023

Guilherme Kraemer

COMPARAÇÃO ENTRE MÉTODOS DE INTEGRAÇÃO DE DADOS PARA SENSORES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

Aprovado em 20 de dezembro de 2023:

Osmar Marchi dos Santos, Dr. (UFSM)
(Presidente/Orientador)

Simone Regina Ceolin, Dra. (UFSM)

Andrei Piccinini Legg, Dr. (UFSM)

Santa Maria, RS
2023

AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão a todas as pessoas e instituições que contribuíram para a realização deste trabalho. Este projeto não teria sido possível sem o apoio e incentivo de muitos indivíduos, e é com imensa satisfação que dedico esta seção a todos aqueles que tornaram esta jornada acadêmica significativa.

- *Aos meus pais Gilmar e Janete, por tudo que representam em minha vida.*
- *Ao meu irmão Mateus, pelo apoio quando precisei levantar novamente.*
- *Ao meu orientador Osmar, por toda a dedicação e conhecimento compartilhado ao longo do projeto de pesquisa, TCC e disciplinas ministradas durante a graduação.*
- *À equipe da Node IoT, pelas oportunidades e experiências proporcionadas.*
- *E por fim, à todos os amigos, pelas risadas, desafios e momentos compartilhados.*

Cada um de vocês fez a diferença nesta trajetória, sou grato por compartilhar essa conquista com todos. Muito obrigado!

RESUMO

COMPARAÇÃO ENTRE MÉTODOS DE INTEGRAÇÃO DE DADOS PARA SENSORES IOT

AUTOR: Guilherme Kraemer
Orientador: Osmar Marchi dos Santos

Com expectativas de que a área de Internet of Things (IoT) gere um impacto econômico entre 3,9 a 11,1 trilhões de dólares por ano a partir de 2025, chegando a 22 bilhões de dispositivos conectados neste mesmo tempo, percebe-se o grande potencial que este segmento tem para produzir e se beneficiar com o surgimento de novas tecnologias. Neste sentido, este estudo de caso apresenta uma análise comparativa entre uma tecnologia recém criada (Node IoT) em relação a outra presente no mercado há quase 20 anos (Mongoose Web Server Library), utilizando-as na prática para integração de dados de sensores IoT. São desenvolvidas duas abordagens diferentes para o mesmo problema, expondo os desafios encontrados em projetos deste tipo e as qualidades que cada ferramenta possui para mitigar essas dificuldades. Ao final, em avaliação qualitativa dos aspectos práticos envolvidos nas implementações, percebe-se que a plataforma Node IoT é mais adequada quando se visa agilidade no desenvolvimento e manutenção, enquanto a biblioteca Mongoose é compatível com uma gama maior de arquiteturas de *hardware* e protocolos de comunicação, oferecendo mais flexibilidade nas maneiras de utilização.

Palavras-chave: IoT. Mongoose. Node IoT. Integração. Sensores. Dados. (...)

ABSTRACT

COMPARISON BETWEEN DATA INTEGRATION METHODS FOR IOT SENSORS

AUTHOR: Guilherme Kraemer
ADVISOR: Osmar Marchi dos Santos

With expectations that the Internet of Things (IoT) area generates an economic impact between 3,9 to 11,1 trillion dollars annually from 2025 and beyond, reaching 22 billion connected devices during the same period, the significant potential of this segment to produce and benefit from the emerging of new technologies becomes apparent. In this context, this case study presents a comparative analysis between a recently created technology (Node IoT) and another that has been in the market for almost 20 years (Mongoose Web Server Library), using them in practice for integrating IoT sensor data. Two different approaches are developed for the same problem, exposing the challenges encountered in projects of this kind and the qualities that each tool possesses to mitigate these difficulties. In the end, through a qualitative evaluation of the practical aspects involved in the implementations, it is observed that the Node IoT platform is more suitable when agility in development and maintenance is the focus, while the Mongoose library is compatible with a wider range of hardware architectures and communication protocols, offering more flexibility in ways of utilization.

Keywords: IoT. Mongoose. Node IoT. Integration. Sensor. Data. (...)

LISTA DE FIGURAS

LISTA DE SIGLAS

A	Amperes
API	*Application Programming Interface*
AWS	*Amazon Web Services*
CAN	*Controller Area Network*
CEO	*Chief Executive Officer*
CPU	*Central Processing Unit*
DNS	*Domain Name System*
DIY	*Do It Yourself*
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IA	Inteligência Artificial
IDE	*Integrated Development Environment*
IHM	Interface Homem-Máquina
IoT	*Internet of Things*
IoS	*Internet of Services*
IP	*Internet Protocol*
ISS	*International Space Station*
MAC	*Media Access Control*
MIT	*Massachusetts Institute of Technology*
MVP	*Minimum Viable Product*
MWSL	*Mongoose Web Server Library*
POC	*Proof of Concept*
POV	*Proof of Value*
RFID	*Radio Frequency Identification*
SMTP	*Simple Mail Transfer Protocol*
SoC	*System-on-a-Chip*
UI	*User Interface*
URL	*Uniform Resource Locator*
UX	*User Experience*
VSCode	*Visual Studio Code*

SUMÁRIO

1 INTRODUÇÃO

A transformação digital, tema muito discutido atualmente, é um fenômeno extremamente disruptivo e que vêm provocando mudanças profundas em empresas do mundo todo. CEOs e outros líderes seniores precisam entender as tecnologias que conduzem a transformação digital de hoje em muito mais detalhe do que era exigido deles no passado (SIEBEL, 2021). Como se não bastasse, claramente as tecnologias que impulsionam esta transformação estão mudando, ainda que estejamos no estágio inicial desta nova era.

O potencial para realização dessa transformação é enorme, sem dúvidas. Mesmo assim, os desafios envolvidos no desenvolvimento e escalada de aplicações de *Artificial Intelligence* (AI) e *Internet of Things* (IoT) em uma empresa podem ser apavorantes. No entanto, toda evolução tecnológica ocorrida na história humana está, intimamente, ligada a desafios.

De acordo com (ISAACSON, 2014), em 1837 foi publicado por Charles Babbage um artigo que descrevia um computador sofisticado, entretanto cem anos se passaram até que os avanços tecnológicos permitissem a construção de algo do gênero. Mesmo que não tenha sido completada na época, a Máquina Analítica de Babbage ficou mais tarde conhecida como o primeiro projeto para um computador de uso geral possível de se descrever, em termos modernos, como *Turing-complete*.

Poucos anos mais tarde, Ada Byron (condessa de Lovelace) publicara suas notas sobre a máquina de Babbage, trabalho pelo qual ficou conhecida como a primeira programadora da história. Em 1847, George Boole apresentou um sistema que usa álgebra para raciocínio lógico, algo ensinado e utilizado até o presente.

Estes e muitos outros, como Alan Turing por exemplo, contribuíram significativamente na transformação tecnológica, possibilitando a criação dos dispositivos hoje utilizados para as mais diversas finalidades. Adicionalmente, a quantidade de transistores nos *chips* aumentou de maneira gradativa ao longo dos anos desde a década de 1970, assim como a Lei de Moore previa.

Com isso, chega-se muitos anos depois à era da IoT, com estimativas de que o número de dispositivos conectados chegue a 22 bilhões em 2025 (Oracle Corporation, 2019).

> Segundo uma pesquisa realizada pelo McKinsey Global Institute (2015), a IoT deverá gerar um impacto econômico entre 3,9 trilhões de dólares a 11,1 trilhões de dólares por ano a partir de 2025, sendo que na indústria de manufatura a IoT tem potencial de reduzir o consumo de energia entre 10% a 20% e potencial de melhorar a eficiência do trabalho de 10% a 25%. (SACOMANO; GONçALVES; BONILLA, 2018).

Com tamanho impacto econômico e social, influenciando diretamente no estilo de

vida e trabalho das pessoas, percebe-se a importância de adquirir, desenvolver e compartilhar conhecimentos nessa área de estudo. Neste sentido, um estudo de caso prático realizando o desenvolvimento de um projeto de IoT, de maneira a perceber os desafios associados e avaliar o poder de diferentes ferramentas disponíveis, criadas no Brasil e no exterior, pode ser de grande valor para estimular o desenvolvimento de novos projetos neste segmento.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Este estudo de caso teve como objetivo avaliar o processo e desenvolver uma solução para comunicação de dados entre dispositivo embarcado e aplicativo *mobile*, utilizando duas tecnologias diferentes do segmento de *Internet of Things*, no âmbito de um projeto de pesquisa e iniciação científica voltado para o monitoramento e coleta de dados de bovinos em campo aberto.

Para isso, buscou-se utilizar e comparar duas abordagens diferentes, uma utilizando a biblioteca de comunicação *open-source* Mongoose para linguagens C/C++, outra utilizando a plataforma brasileira privada de *Internet of Things* Node IoT, disponibilizada para testes em sua versão *beta*.

1.1.2 Objetivos Específicos

De maneira a atingir o objetivo geral previamente definido, necessitou-se cumprir os seguintes objetivos específicos:

- Realizar uma revisão bibliográfica das soluções existentes no mercado de *Internet of Things* para o problema apresentado;
- Utilizando a *Mongoose Web Server Library*, desenvolver um *software* para atuar como *middleware* na comunicação entre dispositivo embarcado, banco de dados e aplicativo *mobile*;
- Na plataforma Node IoT, criar e configurar as variáveis do dispositivo utilizado para coleta de dados;
- Partindo do código base de comunicação Node IoT, realizar a integração com o *firmware* desenvolvido no projeto Tecnologias de Precisão para Agropecuária;

- Autenticar o dispositivo na plataforma;
- Criar um dashboard para visualização dos dados;
- Comparar, em termos qualitativos, as duas abordagens utilizadas para solução do problema de comunicação de dados no contexto do projeto de pesquisa, avaliando pontos como: tempo gasto e dificuldades relativas ao desenvolvimento, facilidade de uso e manutenção, limitações práticas da solução.

1.2 ESTRUTURA DO TRABALHO

Visando estruturar este trabalho de maneira adequada, dividiu-se o estudo de acordo com os seguintes capítulos:

- O Capítulo 1 visa fazer uma introdução ao tema, contextualizando sua evolução por meio de acontecimentos histórios marcantes até os tempos atuais, de maneira a corroborar com a importância do assunto e justificativa para realização do trabalho.
- O Capítulo 2 apresenta uma breve fundamentação teórica, cujo intuito principal é trazer aos leitores um entendimento superficial acerca do tema e tecnologias utilizadas no projeto. Portanto, foram abordados conceitos com relação a *Internet of Things*, comunicação e formatação de dados, bem como soluções comerciais existentes.
- O Capítulo 3 expõe, detalhadamente, o desenvolvimento de um servidor para comunicação utilizando *Mongoose Web Server Library*. O objetivo é ser algo semelhante a uma *Proof of Concept*, uma proposta de solução para o problema apresentado.
- O Capítulo 4 traz, como abordagem alternativa de solução ao problema, o uso da ferramenta em formato plataforma Node IoT para integração direta entre dispositivo embarcado e nuvem, com aquisição e visualização dos dados em tempo real.
- O Capítulo 5 faz uma avaliação comparativa dos aspectos práticos de implementação das soluções, destacando os desafios encontrados bem como pontos positivos e negativos de cada tecnologia.
- O Capítulo 6 apresenta uma breve recapitulação dos objetivos propostos e sintetiza a análise realizada no capítulo anterior, concluindo o trabalho destacando situações em que cada ferramenta se faz mais adequada.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho abrange conceitos relevantes ao campo de estudo relacionado a *Internet of Things* (IoT) e tecnologias comumente utilizadas em projetos desta área. Para entrar nesse tema, são abordados assuntos como *big data*, *cloud computing* e *Internet of Services* (IoS), bastante comentados já há algum tempo.

Passando para aspectos práticos associados a projetos de IoT, são conceituadas tecnologias que dizem respeito ao *hardware*, protocolos de comunicação e padrões de formatação de dados utilizados neste segmento. Além disso, soluções existentes no mercado são apresentadas para embasar a abordagem proposta no trabalho, servindo também como trabalhos relacionados. Com isso, pode-se ter um entendimento dos objetivos definidos e métodos utilizados.

2.1 INTERNET DAS COISAS (IOT)

A Internet das Coisas é usualmente conceituada como a capacidade de diferentes tipos de objetos estabelecerem e manterem uma conexão com a internet, de maneira que possam ser acessados remotamente por outro dispositivo conectado. Obviamente, isso não vale apenas para computadores e celulares, que já possuem essa capacidade há mais tempo, mas sim para uma gama muito mais ampla de dispositivos que até recentemente não tinham essa possibilidade, como sensores, eletrodomésticos e até automóveis.

O conceito para o termo IoT foi criado e apresentado, não exatamente com essas palavras, pelo britânico Kevin Ashton, que na época estava envolvido em um projeto relacionado à identificação por radiofrequência (RFID).

> Formulada em 1999, sua ideia descreveu um sistema no qual o mundo material se comunicaria com computadores por meio da troca de dados com sensores onipresentes. Quase uma década depois, na virada de 2008 para 2009, o número de dispositivos conectados à rede excedeu o número de habitantes do nosso planeta. Esse momento, de acordo com a Cisco, foi o verdadeiro nascimento da IoT, referida mais frequentemente como a internet de tudo. (SACOMANO; GONçALVES; BONILLA, 2018).

O que percebemos hoje em dia é um ambiente virtual integrado se estabelecendo, onde aparelhos recebem e transferem dados a todo momento utilizando a nuvem, podendo dessa forma agir em conjunto.

Essa comunicação constante e com aumento crescente no número de dispositivos conectados à rede gera quantidades massivas de dados. Associa-se a isso uma densidade enorme de informações, provenientes de processos industriais, comerciais, com-

portamento de consumidores, entre outras fontes, de maneira que essa massa passou, então, a ser conhecida como *big data*. Segundo (SACOMANO; GONçALVES; BONILLA, 2018), otimizações, reduções de desperdícios, sustentabilidade e oportunidades de negócios poderão ser geradas a partir dessas informações, se analisadas adequadamente por *software*.

Para armazenamento e processamento de tantas informações passaram a ser utilizados servidores de diversos locais do mundo, facilitando o acesso remoto aos arquivos gerados. Essa prática ficou conhecida como computação em nuvem (*cloud computing*), uma vez que não se tem certeza de onde está fisicamente localizado o servidor cujos dados estão sendo acessados. A finalidade, no entanto, é justamente possibilitar o acesso a partir de qualquer lugar do mundo onde exista conexão com a internet.

Nesta linha de transformação digital e intimamente ligado ao IoT, um novo e mais inteligente conceito de serviços passa a ser também oferecido aos consumidores, conhecido como *Internet of Services* (IoS). Com a informação de que o seu trajeto usual para o trabalho se encontra congestionado, um despertador inteligente poderia acordá-lo um pouco mais cedo, caso a sugestão de rota alternativa tome mais tempo do que a original tomaria em condições normais.

Um campo aberto se abre para a inovação, com inserção de inteligência em coisas e processos que até então pareciam já ter atingido seu estado final. Tais possibilidades se devem muito aos avanços na computação e tecnologias de fabricação para essa área, permitindo que capacidades computacionais consideráveis e conectividade sem fio fossem reunidos em dispositivos cada vez mais compactos, os microcontroladores.

2.2 MICROCONTROLADORES

Os microcontroladores são dispositivos que reúnem *Central Processing Unit* (CPU), memórias e periféricos de entrada e saída em um único circuito integrado. Não raramente confundidos com microprocessadores, diferem-se destes, que são apenas processadores de tamanho bastante reduzido. A importância dos microcontroladores reside basicamente na alta demanda por sistemas embarcados do mercado atual. Isto é, sistemas computacionais compactos e de custo acessível que atendem a uma demanda específica (Matheus Cardoso, 2020).

Não é de se surpreender que estejam, atualmente, contidos nos mais diversos tipos de dispositivos eletrônicos a nossa volta. Sistemas de alarme, periféricos de computador, relógios de pulso, eletrodomésticos, impressoras, automóveis, etc. Tudo passou a ser uma "coisa"conectada à internet.

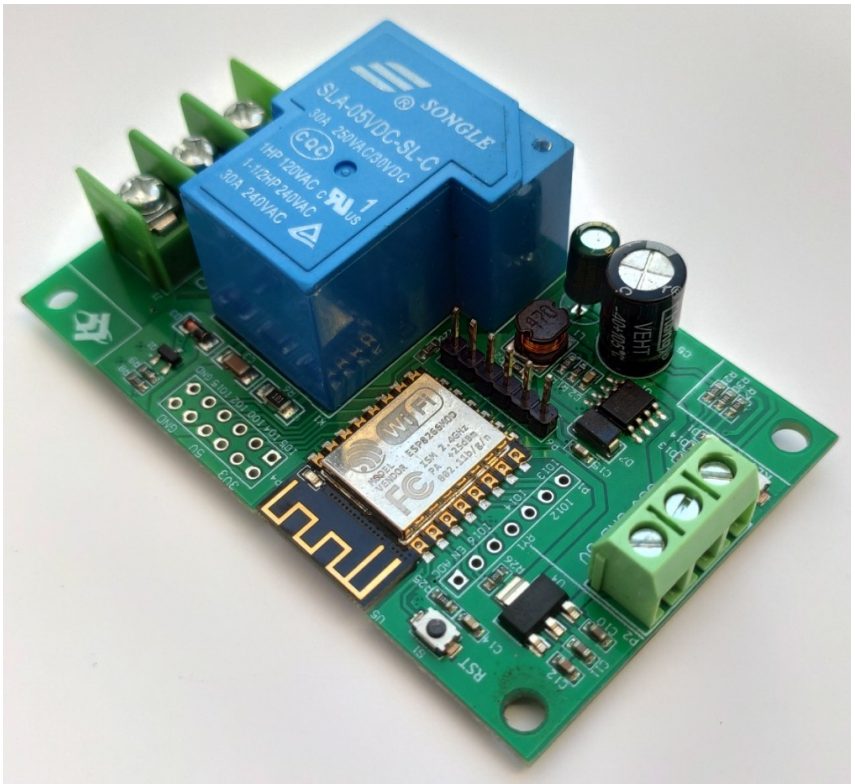
Segundo (KERSCHBAUMER, 2018), a crescente utilização de microcontroladores se deve principalmente a sua versatilidade, tendo em vista que o seu comportamento de-

pende do software nele gravado. Assim um mesmo microcontrolador pode ser utilizado para uma infinidade de aplicações bastando apenas mudar o seu software.

Devido a isso tem-se uma das principais vantagens desses dispositivos, a possibilidade de melhorar um produto apenas com uma atualização de software, o que não pode ser feito nos circuitos analógicos ou digitais tradicionais. Olhando para a versatilidade, também, há opções para os mais variados requisitos. O Arduino, por exemplo, possui diversos componentes, placas bastante completas certamente, mas para projetos específicos talvez muitos desses componentes não sejam aproveitados. Enquanto isso por outro lado, selecionando apenas um microcontrolador, pode-se desenvolver sua própria placa com as ferramentas exatas para atender seu projeto, economia e eficiência sob medida.

Entre as principais fabricantes de microcontroladores, têm-se Intel, Microchip, Atmel, STMicroelectronics e Texas Instruments. Com certeza muitos conhecem as linhas Intel MCS, PIC e Atmel AVR dessas marcas, respectivamente, mas outro nome que ganhou força nesse nicho é a chinesa Espressif Systems, que vêm se destacando pelas séries ESP32 e ESP8266, conhecidos por sua integração *Wi-Fi* e *Bluetooth*, baixo custo e baixo consumo de energia.

Figura 1 – Exemplo de módulo ESP8266.



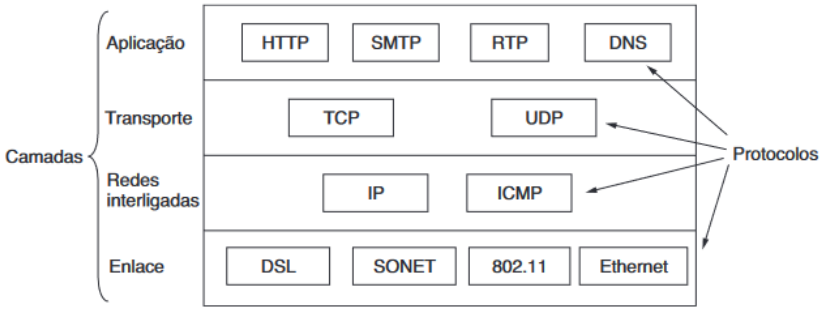
Fonte: Autor.

2.3 COMUNICAÇÃO DE DADOS APLICADA À INTERNET DAS COISAS

Já se sabe que todos esses dispositivos inteligentes citados até aqui estão, de alguma maneira, conectados a essa imensa infraestrutura física que conhecemos como Internet. Dessa maneira, assim como os já muito difundidos computadores e celulares, operam por meio do *Internet Protocol* (IP). O IP especifica o formato dos pacotes enviados através da Internet, bem como os mecanismos utilizados para encaminhar pacotes a partir de um computador, por meio de um ou mais roteadores, para um destino final (COMER, 2016). Além disso, o IP fornece um endereçamento único para cara dispositivo na rede.

Essencialmente, pode ser considerado o protocolo mais importante dentro da arquitetura de protocolos da Internet. Entretanto, para fins práticos normalmente não há necessidade de preocupação nesse sentido em projetos de IoT visando cliente final, uma vez que o *hardware* de partida já possui capacidade de conexão com a Internet via *Wi-Fi*.

Figura 2 – Arquitetura de protocolos de Internet (modelo TCP/IP).



Fonte: (TANENBAUM; WETHERALL, 2011)

No projeto de um produto ou solução para o mercado de IoT a atenção passa a ser, então, voltada àquilo que se usa na camada mais superior dessa arquitetura de protocolos de Internet. A Camada de Aplicação engloba os protocolos que vão, efetivamente, executar alguma tarefa para os usuários. Estes protocolos têm papel crucial na comunicação entre aplicativos e serviços em uma rede, pois definem as regras para o intercâmbio de dados entre sistemas distintos.

Bons exemplos disso são os protocolos *Single Mail Transfer Protocol* (SMTP) e *Domain Name System* (DNS). O SMTP é um protocolo muito popular, usado no envio e recebimento de *e-mail* desde a década de 80. Já o DNS, segundo (Melissa Cruz Cossetti, 2018), é um sistema de banco de dados utilizado na internet com objetivo de traduzir os endereços dos sites de uma complexa sequência numérica para algo mais simples e amigável, como os nomes dos endereços que escrevemos quando usamos o navegador. No entanto, aquele que receberá maior atenção neste trabalho é o *Hypertext Transfer Protocol* (HTTP).

2.3.1 Protocolo HTTP/HTTPS

Conforme dito por (TANENBAUM; WETHERALL, 2011), HTTP é um protocolo simples, do tipo solicitação-resposta, que roda sobre o *Transmission Control Protocol* (TCP). Ele especifica quais mensagens os clientes podem enviar para os servidores e quais respostas recebem de volta. Possui, também, uma versão com mecanismos de segurança adicionais, conhecida como *Hypertext Transfer Protocol Secure* (HTTPS).

Surgiu na década de 1990, devido à necessidade de um padrão na forma de se comunicar na internet, uma maneira leve, rápida e que todas as máquinas pudessem compreender. Esse modelo se tornou a base para quase toda comunicação na Internet. Qualquer servidor de hospedagem de sites possui um programa que recebe e responde solicitações HTTP. Basicamente, então, os navegadores são clientes HTTP que estão constantemente enviando solicitações a servidores, de maneira a obter o conteúdo que se deseja visualizar.

No entanto, é importante observar a evolução no modo como o HTTP é usado na Internet. Como um protocolo da Camada de Aplicação, rodando sobre o TCP, está intimamente ligado à *Web*.

> Porém, em outro sentido, o HTTP está se tornando mais um protocolo de transporte, que oferece um meio para os processos comunicarem conteúdo entre os limites de diferentes redes. Esses processos não precisam ser um navegador *Web* nem um servidor *Web*. (TANENBAUM; WETHERALL, 2011).

Uma evidência disso é justamente o uso do HTTP em aplicações de IoT. Bibliotecas e plataformas já difundidas nesse meio, como Mongoose e Ubidots, suportam comunicação por meio desse protocolo. E juntamente com a popularização de maneiras diferentes de se usar o HTTP, a forma de representação de dados *JavaScript Object Notation* (JSON) tem ganhado espaço em muitos projetos.

2.3.2 JSON

O *JavaScript Object Notation* é um padrão definido para troca e armazenamento de informações em formato texto. É bastante conhecido e utilizado pela facilidade com que pode ser entendido por humanos e máquinas, podendo ser gerado e interpretado facilmente. Além disso, é leve para ser transportado pela rede e suportado na maioria das linguagens de programação atuais.

Embora o nome remeta à linguagem *JavaScript*, é independente desta e qualquer outra linguagem de programação, contudo utiliza convenções familiares a programadores de linguagens da família C. Todas essas características fazem do JSON um formato ideal para troca de informações. Apesar de muito simples, tem sido bastante utilizado por aplicações *Web* devido a sua capacidade de estruturar informações de uma forma bem mais

compacta do que a conseguida pelo modelo *Extensible Markup Language* (XML), tornando mais rápido o parsing dessas informações (DEVMEDIA, 2012).

De acordo com (Diego Melo, 2021), a representação das informações em um arquivo JSON é feita por pares compostos de atributo e valor. Entre aspas indica-se o atributo, um identificador para o valor, da mesma maneira que o nome de uma variável em linguagem C, por exemplo. Após o atributo, utiliza-se dois pontos para separá-lo do valor. Este, por sua vez, pode ser fornecido como *string* (texto), número ou *booleano* (valor lógico), que são os tipos básicos. Entre cada par de atributo e valor é colocada uma vírgula e, por fim, um par de chaves englobando todos os atributos e valores define um objeto JSON.

A estrutura permite, ainda, a passagem de valores nos formatos *array* e objeto, ambos criados a partir dos tipos básicos. O array, diferente do objeto, é delimitado por colchetes com uma vírgula de separação entre cada um dos elementos. Dessa maneira, pode-se criar estruturas aninhadas de informações, com a configuração mais adequada para cada necessidade.

```
1 // exemplo de objeto JSON
2
3 {
4     "nome" : "Guilherme Kraemer",
5     "ano" : 2023,
6     "disciplinas" : ["TCC", "Estagio"]
7 }
```

Listing 2.1: exemploJSON.c

2.4 BIBLIOTECA MONGOOSE WEB SERVER

A *Mongoose Web Server Library* (MWSL) é uma ferramenta em formato de biblioteca para desenvolvimento de redes e comunicação em linguagens C/C++ que possibilita a implementação de *Application Programming Interfaces* (APIs) orientadas a eventos para os protocolos TCP, UDP, HTTP, WebSocket e MQTT. Por meio da criação de clientes e servidores, ela conecta dispositivos e coloca-os *online*, rodando em cima de sistemas operacionais como Windows, Linux, Mac e diversas arquiteturas embarcadas conhecidas, entre elas STM32, NXP, TI e ESP32.

Tornou-se popular ao longo dos anos entre desenvolvedores envolvidos em projetos que requerem a criação de aplicativos, comunicação com dispositivos IoT e serviços *web*. No mercado desde 2004, segundo (Cesanta Software Ltd, 2023b), é usada por um vasto número de produtos *open source* e comerciais e roda até mesmo na *International Space Station* (ISS). A MWSL torna a programação de rede embarcada rápida, confiável e simples.

O intuito principal da biblioteca é fornecer uma maneira eficaz e leve de comunicação entre dispositivos por meio das linguagens C/C++. Projetada para ser eficiente e otimizada, promete ser a escolha ideal para aplicações que necessitam escalabilidade e consumo reduzido de recursos.

Lançada no ano de 2004 por Sergey Luybka, fundador da Cesanta, como um *software* de código aberto sob a licença MIT, tem sido mantida desde então por uma comunidade ativa de desenvolvedores, que contribuiu significativamente ao longo dos anos para que isso fosse possível. Até hoje, orienta-se pela visão original do criador, que desejava criar uma biblioteca simples e eficiente para o desenvolvimento de servidores *web* em C.

A biblioteca fornece em sua documentação diversos tutoriais e códigos exemplo, que servem como ponto de partida para desenvolver uma aplicação funcional. Pensando no desenvolvimento de um servidor do tipo REST com comunicação HTTP/HTTPS, existem dois pontos principais que demandam mais atenção e trabalho, dependendo do caso.

O primeiro, que define a maneira como serão processadas as requisições do cliente, é o desenvolvimento de uma função responsável pelo tratamento de eventos. Essa função é chamada sempre que um evento ocorre na conexão, verificando qual foi o tipo ocorrido para saber como tratar a informação, de maneira a entender a requisição e/ou obter os dados da mensagem. Nela podem ser definidas ações específicas de acordo com a rota utilizada na requisição, lendo e gravando em banco de dados ou gerando mensagens de erro, por exemplo.

O outro ponto diz respeito à segurança da comunicação entre as partes, com um mínimo necessário para evitar exposições, podendo ainda demandar camadas extras dependendo da aplicação do sistema. Considerando uma automação residencial, é plausível pensar que os riscos envolvidos em casos de exposição da mensagem seriam muito menores do que em um processo industrial, mas não inexistentes, ambos os casos demandam análise. Dessa maneira se pode definir um nível que encaixe o projeto com a aplicação, evitando também um dispêndio maior de tempo e esforço na parte de desenvolvimento.

Entre *software*, eletrodomésticos, dispositivos e máquinas, existem milhões de produtos alimentados pela biblioteca Mongoose em produção atualmente. Entre os clientes divulgados pela Cesanta em sua página oficial estão gigantes da indústria como Samsung, Siemens, Caterpillar, Bosch e Google, que utilizam a biblioteca para implementar interfaces de usuário *web* em dispositivos, atualizações remotas de *software* e *firmware*, monitoramento remoto, troca de dados de telemetria e serviços do tipo API RESTful.

Adicionalmente, postados no site existem alguns casos de uso oriundos da parceria formada pela Cesanta com empresas diferentes das acima citadas. Nessas postagens pode-se ter uma ideia do que esses companhias buscavam desenvolver como produto e para quê utilizaram a MWSL. Entretanto, como esses casos tratam do uso para desenvolvimento de produtos/soluções comerciais, o nível de detalhamento fornecido é naturalmente limitado.

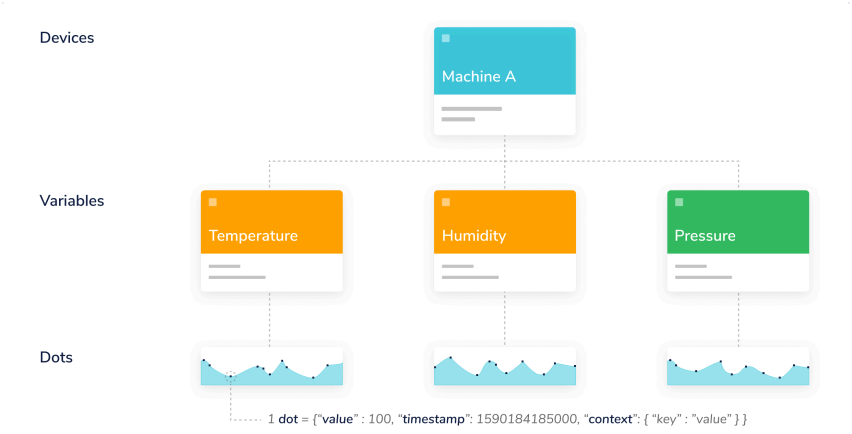
2.5 UBIDOTS

A Ubidots é uma *startup* que fornece uma plataforma de *Internet of Things* (IoT) baseada em nuvem, permitindo o desenvolvimento e construção de projetos nesse segmento de maneira a ter uma visualização amigável e eficiente dos dados gerados no sensoriamento. Com foco maior em aplicações industriais, mas não limitada a isso, a plataforma é capaz de se conectar a dezenas de dispositivos diferentes, de maneira a acessar dados obtidos por sensores em tempo real.

Na plataforma são criados dispositivos Ubidots, entidades virtuais que podem ser mapeadas a um ou mais dispositivos físicos, de acordo com a vontade do usuário. Em suas documentações, a empresa disponibiliza exemplos de conexão, envio e recebimento de dados utilizando diversos modelos de dispositivos diferentes, cada um com sua própria rotina de configurações.

Uma vez que o dispositivo esteja conectado, a Ubidots trabalha por meio da criação de pontos. Toda vez que um dispositivo atualiza o valor de um sensor em uma variável, um ponto de dados (*dot*) é criado. A plataforma armazena dentro de variáveis os pontos que vêm dos dispositivos e cada ponto possui um registro temporal associado.

Figura 3 – Hierarquia de dados Ubidots.



Fonte: Adaptação de (Ubidots, 2023).

Um ponto, como se pode notar na parte inferior da imagem acima, é uma mensagem em formato JSON que carrega as informações *value*, *timestamp* e *context*, por meio das quais a plataforma recebe dados do contexto da aplicação, juntamente com informações temporais. Por padrão, a plataforma armazena até 2 anos de informações em formato de série temporal.

O principal ponto com relação a essa plataforma, certamente, é o foco em ferramentas para análise de dados, com funções para limpar, filtrar e agregar dados, por exemplo, de maneira a extrair *insights* e contribuir com a tomada de decisões em alto nível. Para isso, dispõe de criação de *dashboards* com gráficos, tabelas, mapas e outros elementos

para visualizar e compartilhar os dados.

Através do site, é possível ter acesso a diversos tutoriais *Do It Yourself* (DIY) e exemplos de uso e integração com a plataforma, alguns publicados pela própria Ubidots e outros por usuários da comunidade. Diferente da biblioteca Mongoose, nesse caso não são apresentados exemplos de uso comercial da plataforma.

2.6 NODE IOT

O Node IoT é um ecossistema no formato de plataforma, simplificando e agilizando a criação de aplicações ao integrar dispositivos *edge* (SOUZA; TEJADA, 2022). Desenvolvida por uma empresa gaúcha sediada em Guaíba/RS, sua concepção teve início em um grupo de estudo, motivado por interesses comuns na área, bem como pela ausência de uma ferramenta que proporcionasse conexão em tempo real, utilizando um padrão de protocolo seguro e que não se limitasse a uma área específica.

A ferramenta proporciona através de toda sua gama de funcionalidades a aceleração de processos vitais para empresas que buscam inovação e tecnologia, pois proporciona velocidade e escalabilidade na construção de *Proof of Concept*(POC), *Proof of Value*(POV) e *Minimum Viable Product*(MVP), auxiliando da prototipagem ao produto. Seu principal propósito é viabilizar o desenvolvimento e integração de projetos relacionados à automação, IoT, coleta, gerenciamento e visualização de dados, assim como a criação de dashboards e elementos visuais de controle e análise.

A Node IoT desenvolveu um padrão de protocolo baseado em HTTP com mensagens em formato JSON. Por meio de seu próprio broker, a plataforma oferece recursos de mensageria em tempo real, garantindo um tempo de resposta máximo de 500 ms em conexões estáveis. É importante destacar, também, a abordagem em segurança cibernética, com autenticação de dispositivos, verificação de MAC e chaves de acesso.

No que diz respeito ao sistema embarcado, a Node IoT disponibiliza à comunidade um código aberto em C/C++, que serve como base para projetos integrados à plataforma. Esse código, gerado diretamente pela ferramenta *web*, não apenas inclui funções de *edge computing* e conexões prontas para uso, mas também permite a implementação de controles e funções específicas para cada projeto ou necessidade.

Outro recurso disponível é a ferramenta para a criação de dashboards, destinados à exibição e tratamento de dados, envio de comandos a atuadores e integrações com outros serviços ou projetos. O diferencial desta funcionalidade reside na alta capacidade de personalização, com responsividade e praticidade, permitindo a aplicação de conceitos de *User Interface*(UI) e *User Experience*(UX) para projetos que até então utilizavam somente Interface Homem-Máquina(IHM).

Além do mencionado acima, a ferramenta conta ainda com armazenamento em

nuvem, acesso em uma variedade de dispositivos e compatibilidade com ampla gama de controladores. Tudo isso resulta em grande amplitude de possibilidades no que se refere a projetos, processos, integrações e utilização de sensores e atuadores.

3 TRABALHOS RELACIONADOS

3.1 MONGOOSE

3.1.1 Estudo de caso: Schneider Electric

A Schneider Electric é um grupo multinacional francês, que tem como especialidade produtos e serviços para distribuição elétrica, controle e automação, contando com mais de 135 mil colaboradores ao redor do mundo. Integra tecnologias de ponta em processo e energia, conecta físico à nuvem por meio de produtos, controle, *software* e serviços, bem como possibilita um gerenciamento integrado para casas, edifícios e indústrias. Com ações desse tipo, a Schneider Electric visa impulsionar a transformação digital no mundo.

A parceria entre Cesanta e Schneider Electric se dá por meio da utilização da MWSL em um produto chamado *Automation Runtimes*, que consiste em um sistema de controle distribuído especializado usando dispositivos embarcados multiplataforma. Nele, a biblioteca implementa a comunicação e uma API RESTful.

> Facilmente, Mongoose é um dos melhores servidores *web* no mercado. Mongoose tem nos ajudado a atingir o nível de performance requerido com integração perfeita. É muito estável e nós não encontramos qualquer *bug* significativo. (Santhosh Kumar P.B, 2023).

Segundo representante da Schneider Electric, o desafio que os levou a essa procura foi a necessidade de um servidor *web* que fosse leve e performático. Comparando as opções disponíveis no mercado, a única que atendeu todas as características buscadas foi a MWSL, que conta ainda com um bom suporte comercial, tornando a escolha óbvia.

3.1.2 Estudo de caso: Veethree

O Grupo Veethree é um conglomerado empresarial movido por tecnologias em comum e inovação, com colaboradores por todo o mundo e unidades em quatro continentes. A Veethree Technologies, sediada no Reino Unido, é uma parte do grupo presente na América do Norte, Ásia e Reino Unido, projetando, desenvolvendo, fabricando e fornecendo suporte a uma gama de displays CANvu.

Em seu portfólio de produtos existe um *gateway* genérico CAN para TCP/IP chamado *H1*, cuja comunicação e API foram desenvolvidas utilizando a biblioteca Mongoose.

O *H1* é um dispositivo que serve como central de comunicações, fazendo uma ponte entre barramentos CAN NMEA2000 e J1939, RS422 e redes TCP/IP *Ethernet* e sem fio para aplicações de controle marítimas, automotivas, agrícolas e industriais.

Figura 4 – Veethree H1 gateway.

Fonte: Adaptação de Cesanta Software Ltd (2023a).

De acordo com as palavras de Sam Edge, engenheiro de *software* na Veethree Technologies, para suportar o tipo de dispositivos que se comunicam com o *H1*, era necessário inicialmente um servidor de aplicação *web* HTTP/HTTPS com suporte a WebSocket, com possibilidade de acréscimo futuro de outros serviços IoT, como o MQTT por exemplo. Em vez de juntar essas partes ou lançar uma solução própria, procurou-se por uma solução única e compacta, que pudesse ser lançada rapidamente.

Com essa demanda e o desejo de atingir as metas técnicas e comerciais relacionadas ao projeto, a equipe envolvida no desenvolvimento do *H1* evitou usar uma plataforma Linux embarcada, comumente usadas em outros casos, optando por uma alternativa *bare metal* de sistema operacional de tempo real que pode ser implementada em um *hardware* muito mais eficiente em termos de custo e energia. A flexibilidade da configuração em tempo de compilação, possibilidade de rodar a partir de um único arquivo C e a integração com o TCP/IP e TLS escolhidos, fizeram da MWSL a escolha adequada para o *H1*.

No processo de implementação, a integração do Mongoose com as APIs da empresa foi ágil e a equipe conseguiu fazer funcionar os serviços necessários para a comunicação de maneira bastante rápida, destacando principalmente seu design movido a eventos. O fornecimento de um simples servidor WebDAV no Mongoose possibilitou aos desenvolvedores fazer *debug* e editar as aplicações *web* diretamente na memória *flash* do *hardware* utilizado no *H1*, o que foi muito conveniente a eles.

Como resultados, o *H1* é capaz de prover uma plataforma com *hardware* e *software* em C para aplicações usando comunicação CAN, Serial e redes TCP/IP com pouca ou nenhuma alteração no código compilado. Carregando diferentes recursos *web* via Mongoose, pode se adaptar a contínuos requisitos e com baixo risco.

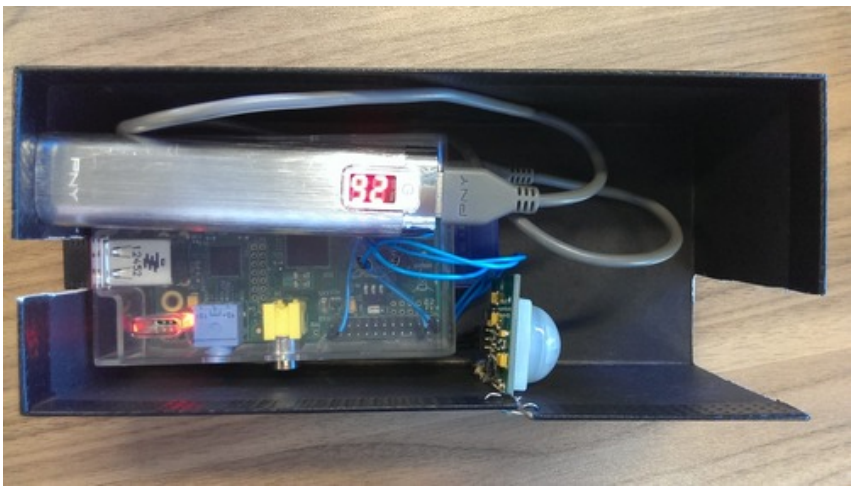
3.2 UBIDOTS

3.2.1 Estudo de caso: Contador de pessoas com Raspberry Pi

Muito utilizados no varejo para gerar mapas de calor das regiões de maior interesse de consumidores dentro de lojas, os contadores de pessoas também são encontrados em aplicações de segurança, gerenciamento de eventos e cidades inteligentes. Segundo (Agustin Pelaez, 2023), para implementação deste projeto são necessários:

- Raspberry Pi Modelo B;
- Módulo bateria pequeno, com cabo micro-USB;
- *Wi-fi USB dongle*;
- Sensor de movimento;
- 3 cabos fêmea-fêmea;
- Caixa pequena para encapsular os componentes.

Figura 5 – Contador de pessoas com Raspberry e Ubidots.



Fonte: Adaptação de (Agustin Pelaez, 2023).

Na postagem são passadas instruções sobre as conexões realizadas entre os pinos do sensor e as entradas e saídas de propósito geral do Raspberry. O autor também indica como fazer configurações básicas no dispositivo, instalação de bibliotecas, *download* do código e alterações nos parâmetros de dispositivo, variável e *token* da plataforma.

O *script* consiste em um *loop* que checa o estado do pino de sinal do sensor de movimento. Se a leitura for nível lógico alto, significa que houve movimento, então incrementa a variável de contagem de pessoas e aguarda 1,5 segundos até que o sensor retorne ao normal. O envio à plataforma ocorre a cada 10 contagens e uma vez que os dados estejam na nuvem, pode-se adicionar *widgets* como o mostrado abaixo no *dashboard* para visualização da atividade.

Figura 6 – *Dashboard* Ubidots.

Fonte: Adaptação de (Agustin Pelaez, 2023).

4 DESENVOLVIMENTO DO SERVIDOR COM MONGOOSE WEB SERVER LIBRARY

De maneira a desenvolver um servidor que atendesse aos requisitos do projeto, capaz de receber os dados coletados pelos dispositivos embarcados, armazená-los em banco de dados e disponibilizá-los à aplicação *mobile*, fez-se a escolha da biblioteca Mongoose como ferramenta para essa implementação.

Como uma biblioteca *open-source* e presente no mercado há muitos anos, possui uma documentação bem estruturada e com grande disponibilidade de exemplos, o que faz a diferença no processo de desenvolvimento. Construída para as linguagens C/C++ e com suporte ao protocolo HTTP, a familiariade com estas tecnologias vistas ao longo da graduação também contribuiu na escolha.

Adicionalmente, como ferramenta em formato biblioteca, permite um entendimento maior do fluxo e tratamento das informações, diferente de uma plataforma de IoT onde muito é abstraído do usuário. Dessa maneira, além de contribuir no crescimento acadêmico e profissional do estudante, serve também como bom objeto de comparação em relação à plataformas IoT como Ubidots e NodeIoT, por exemplo.

4.1 ESTRUTURA DO SERVIDOR

Considerando um aplicativo *mobile* que oferecesse funcionalidades de cadastro, exclusão e visualização de usuários, dispositivos e dados de sensoriamento, bem como *login* e configuração de coordenadas para cercamento virtual, planejou-se implementar o servidor de acordo com a estrutura apresentada abaixo.

Figura 7 – Estrutura de rotas do servidor Mongoose.

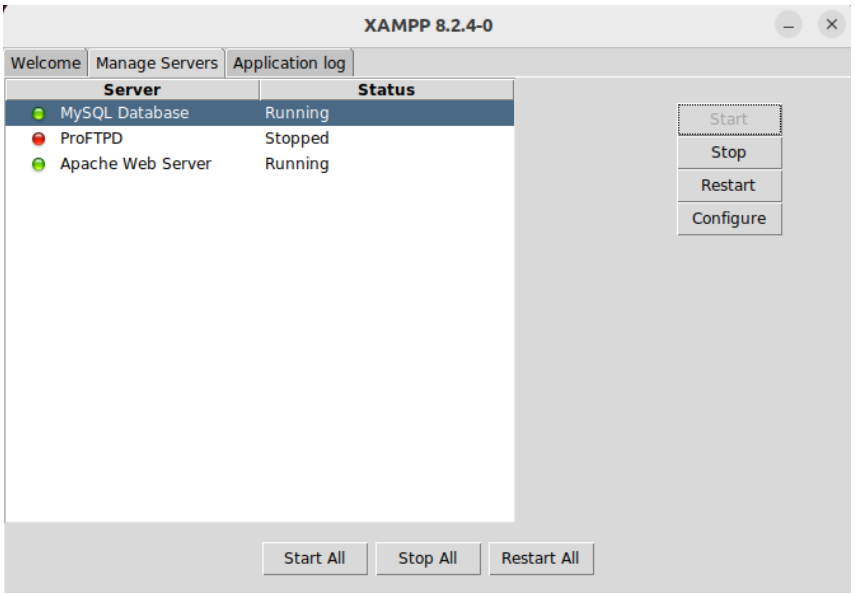
Fonte: Autor.

Ao todo, então, foram planejadas 6 rotas de acesso ao servidor, com algumas delas respondendo a diferentes métodos HTTP, atendendo portanto aos requisitos básicos estipulados para a aplicação. Para armazenar os dados manipulados por esse servidor, fez-se necessário o desenvolvimento, também, de um banco de dados estruturado em acordo com as necessidades do projeto. Com esse propósito, o pacote XAMPP foi escolhido para facilitar essa tarefa.

4.2 IMPLEMENTAÇÃO DO BANCO DE DADOS MYSQL

Segundo o (Apache Friends, 2023), o XAMPP é uma distribuição Apache presente no mercado há mais de 10 anos, que contém PHP, MySQL e Perl. Com ele, é possível iniciar um servidor *web* local para trabalhar com o banco de dados MySQL de maneira fácil e visual. Para isso, após realizar o *download* por meio do site oficial, basta instalar e executar o programa. Na sequência, deve-se fazer o *start* do *Apache Web Server* e do *MySQL Database*, conforme a imagem.

Figura 8 – *Start* no banco de dados MySQL no XAMPP.

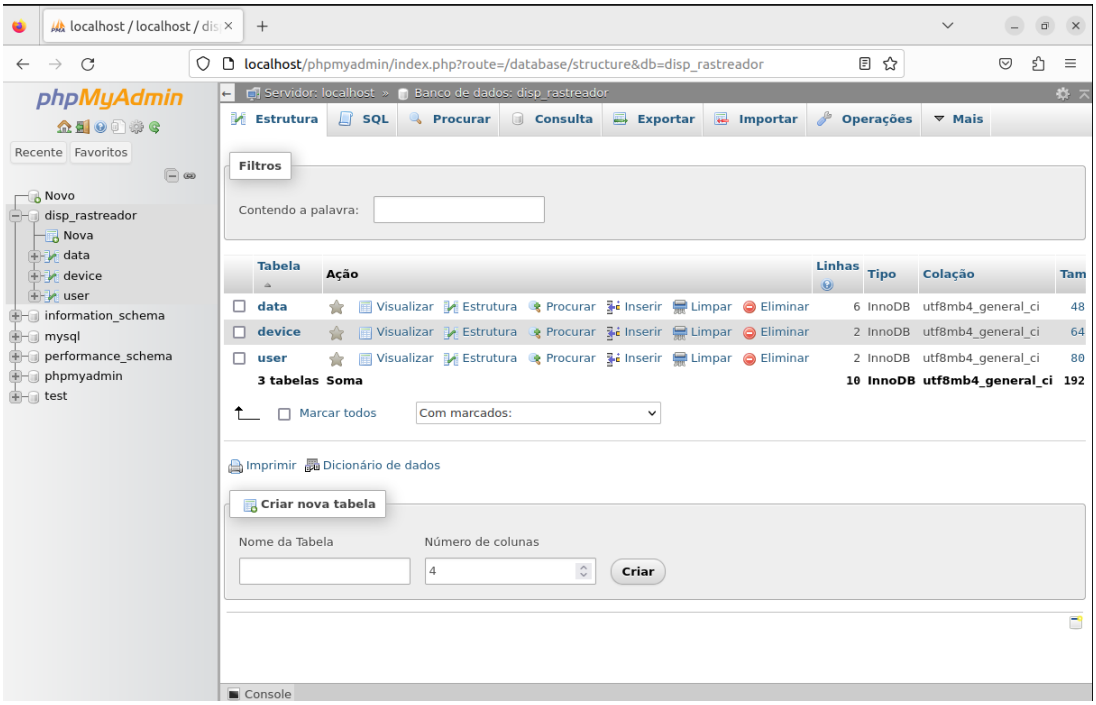


Fonte: Autor.

Após iniciar o MySQL, pode-se acessar o *MySQL Database* por meio da página phpMyAdmin. O *phpMyAdmin* é uma aplicação *web*, escrita em PHP, que roda sobre o Apache, fornecendo uma inteface gráfica para interações com o MySQL. O Apache, por outro lado, é o responsável por lidar com as solicitações HTTP. Em suma, ao executar o XAMPP e, consequentemente, *Apache Web Server* e MySQL, estão sendo ativados os serviços necessários para executar aplicações *web* no ambiente de desenvolvimento local.

Conforme imagem abaixo, nota-se no menu lateral esquerdo a existência do banco de dados *disp_rastreador*, criado para atender a este projeto com três tabelas, uma para usuários, uma para dispositivos e outra para os dados das leituras realizadas pelos dispositivos.

Figura 9 – Interface *web* phpMyAdmin para interações com MySQL.

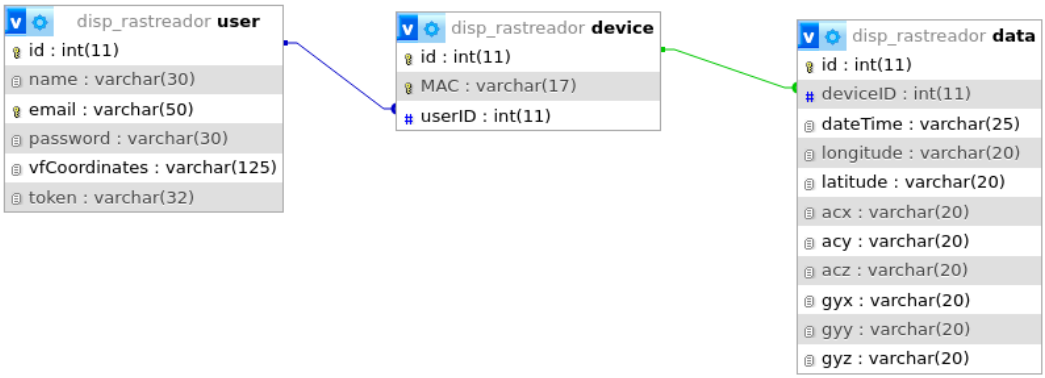


Fonte: Autor.

A tabela *user* armazena dados pessoais da conta do usuário, nome, *e-mail*, senha, *token* para autenticação e coordenadas para funcionalidade de cercamento virtual da propriedade, todos em formato *String*, além de um identificador inteiro único gerado automaticamente. Na tabela *device* ficam as informações quem dizem respeito aos dispositivos cadastrados pelo usuário, com identificador inteiro único também gerado automaticamente, MAC do dispositivo em formato *String* e identificador inteiro do usuário proprietário. Parâmetros como *e-mail* para usuário e MAC para dispositivo são únicos no banco de dados também, de modo que não há a possibilidade de mais de uma conta de usuário com o mesmo *e-mail* ou dispositivo com o mesmo MAC.

Por sua vez, a tabela *data* define aquilo que é armazenado dos resultados de sensoriamentos efetuados pelos dispositivos, guardando nesse caso, em formato *String*, informações de GPS (longitude, latitude e instante de tempo), informações de acelerômetro e giroscópio, ambos com relação aos eixos x, y e z, bem como identificadores inteiros, sendo um único para a leitura e outro para indicar o dispositivo responsável. Essa estrutura está representada na imagem abaixo.

Figura 10 – Estrutura de tabelas do banco de dados MySQL desenvolvido.



Fonte: Autor.

Na imagem acima é possível notar uma conexão entre parâmetros das tabelas. Mais especificamente, o identificador único de *user* é uma chave estrangeira de *device*, assim como o identificador único de *device* é uma chave estrangeira de *data*. O propósito disso é estabelecer uma relação entre essas tabelas. Um conjunto de dados está obrigatoriamente associado a um dispositivo, que por sua vez está associado a um usuário.

Com isso, pode-se definir o comportamento do banco de dados para determinadas ações, como excluir dispositivos ou usuários, por exemplo. Nesses casos, ao excluir um dispositivo, todos os dados associados a ele serão excluídos também, assim como ao excluir um usuário, todos os seus dispositivos (e dados associados) também serão excluídos. Foi definido um comportamento em cascata para essas ações, mas o uso de chaves estrangeiras não implica isso, essa decisão parte do desenvolvedor.

4.3 IMPLEMENTAÇÃO DAS ROTAS DO SERVIDOR

Para iniciar a implementação do servidor, primeiramente foi necessário preparar o ambiente de desenvolvimento. Para comunicação entre o código escrito em C e o banco de dados, foi instalado o pacote libmysqlclient-dev que inclui uma biblioteca cliente do MySQL e o *driver MySQL Connector/C*, juntos fornecendo uma interface C para manipulação de banco de dados. Por meio da diretiva *#include <mysql.h>*, é feita a inclusão da biblioteca.

Como estrutura principal do servidor, a biblioteca *open-source* Mongoose é importada armazenando seus dois arquivos principais (mongoose.c e mongoose.h), disponíveis no GitHub, no mesmo diretório do projeto. Em código, deve-se utilizar as diretivas *#include "mongoose.c"* e *#include "mongoose.h"*.

Como padrão escolhido para o formato das mensagens trocadas entre as partes, devido à facilidade de interpretação tanto por máquinas quanto humanos bem como ampla aceitação entre linguagens, o JSON foi utilizado por meio da biblioteca *open-source*

cJSON, disponível no GitHub. Sua importação ocorreu da mesma maneira que o Mongoose, armazenando os arquivos (cJSON.c e cJSON.h) no diretório do projeto e utilizando as diretivas *#include "cJSON.c"* e *#include "cJSON.h"*.

Com estas configurações necessárias prontas, o ponto de partida foi o código exemplo minimal-HTTP-server, disponibilizado pela Cesanta em sua documentação oficial e mostrado abaixo. Por meio dele serão explicados os pontos principais do código e funcionamento do servidor.

```
1  #include "mongoose.h"
2
3  static void fn(struct mg_connection *c, int ev, void *ev_data, void *fn_data) {
4    if (ev == MG_EV_HTTP_MSG) {
5      struct mg_http_message *hm = (struct mg_http_message *) ev_data;
6      if (mg_http_match_uri(hm, "/api/hello")) {              // On /api/hello requests,
7        mg_http_reply(c, 200, "", "{%m:%d}\n",
8                      MG_ESC("status"), 1);                   // Send dynamic JSON response
9      } else {                                                // For all other URIs,
10       struct mg_http_serve_opts opts = {.root_dir = "."};   // Serve files
11       mg_http_serve_dir(c, hm, &opts);                      // From root_dir
12     }
13   }
14 }
15
16 int main(int argc, char *argv[]) {
17   struct mg_mgr mgr;
18   mg_mgr_init(&mgr);                                        // Init manager
19   mg_http_listen(&mgr, "http://0.0.0.0:8000", fn, &mgr);    // Setup listener
20   for (;;) mg_mgr_poll(&mgr, 1000);                         // Event loop
21   mg_mgr_free(&mgr);                                        // Cleanup
22   return 0;
23 }
```

Listing 4.1: minimal-http-server.c

O funcionamento do servidor é, felizmente, bastante simples de se compreender. Observando o código apresentado na imagem acima, em sua função principal (*main*) que inicia na linha 16, é feita a declaração de uma variável chamada *mgr*, cujo tipo é uma estrutura definida na biblioteca e que possui a função de gerenciar os eventos ocorridos na conexão, armazenando informações sobre conexões ativas bem como outros parâmetros necessários para o funcionamento do servidor, mas que nesse caso não necessitaram da atenção do desenvolvedor.

Abaixo, na linha 18 é executada uma função para inicializar a estrutura *mgr*, definindo a lista ativa de conexões como nula, além de servidores DNS para as versões 4 e 6 do protocolo IP. Na linha 19 ocorre a execução de uma função que indica à estrutura gerenciadora de eventos em qual endereço IP ela deve aguardar por conexões, indicando também pelo parâmetro seguinte qual será a função responsável por tratar os eventos (nesse caso, a função *fn*). Assim, sempre que ocorrer um evento na conexão, a estrutura gerenciadora de eventos irá chamar a função *fn* para responder a esse evento. O último parâmetro representa um ponteiro arbitrário.

Na sequência, inicia-se um *loop* de execução por tempo indeterminado, que fica verificando se há algum dado sendo recebido, gerando diferentes tipos de eventos de acordo com isso. A linha 21 tem a finalidade de fechar todas as conexões e liberar todos os recursos, caso o laço de execução seja quebrado por algum motivo.

Basicamente, no que diz respeito à implementação das rotas de acesso ao servidor, todo o trabalho ocorre na função *fn*, responsável por responder aos eventos ocorridos na conexão. Toda vez que for executada, terá em seu primeiro parâmetro um ponteiro para a conexão que originou o evento, dessa maneira a resposta pode ser dada ao cliente correto. No segundo parâmetro existe a informação do tipo de evento ocorrido, segundo definições da biblioteca disponíveis em documentação. Por meio do tipo de evento se pode saber, também, qual será o tipo de dados recebidos no terceiro parâmetro, para que a conversão possa ser feita corretamente. O último parâmetro, também um ponteiro, é utilizado em casos de conexão HTTPS, o que não ocorre neste projeto.

Dessa maneira, no corpo dessa função o único tipo de evento que nos interessa tratar é o *MG_EV_HTTP_MSG*, o que implica segundo a documentação que o parâmetro apontando para os dados recebidos deve ser convertido para uma estrutura de mensagem HTTP. Desse ponto em diante, assim como feito no *if* logo abaixo, todo o desenvolvimento consiste em acessar os campos da mensagem recebida e realizar o tratamento necessário das informações.

Portanto, abaixo será explicado o comportamento da função responsável pelo tratamento de eventos neste servidor, cujo nome foi alterado para se diferenciar do exemplo fornecido pela biblioteca. Como IDE escolhida para realização desse projeto, utilizou-se o VSCode da Microsoft. É importante mencionar que essa função tem, ao todo, mais de 400 linhas, por isso será fornecida uma explicação mais generalizada juntamente com o exemplo de uma das rotas implementadas, uma vez que muitos dos comportamentos são semelhantes para as demais. O código completo estará na seção de anexos deste trabalho, além de disponível no GitHub.

```
1  static void event_handler(struct mg_connection *c, int ev, void *ev_data, void *fn_data)
       {
2
3      if(ev == MG_EV_HTTP_MSG) { // Identifies HTTP connection
4
5          struct mg_http_message *hm = (struct mg_http_message *) ev_data;
6          const char *method = hm->method.ptr;
7          char tokenStr[32];
8          int i, id, deviceID;
9          cJSON *root = cJSON_Parse(hm->body.ptr);
10         cJSON *aux, *arrayItem;
11
12         printf("\n\t...message...\n%s\n", hm->message.ptr);
13
14         if (checkParse(root) == 0) {
15             mg_http_reply(c, 200, "", "{\"result\": \"Body da mensagem é nulo!\"}\n");
16             cJSON_Delete(root);
17         }
```

```
18          else {
19
20              if (method[1] == 'U' || method[1] == 'A') { // PUT e PATCH indisponíveis
21                  mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível.\"}\n");
22              }
23
24              //rota --- /api/user
25              else if(mg_http_match_uri(hm, "/api/user")) {
26
27                  switch(method[0]) {
28
29                      case 'G': // GET - Obtém informações do usuário
30
31                          if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
32
33                              // verifica qual usuário possui o token informado
34                              id = dbRead(dbConnection, 2, root);
35
36                              if (id != 0) {
37
38                                  cJSON_AddNumberToObject(root, "userID", id);
39
40                                  // consulta os dados do usuário
41                                  if (dbRead(dbConnection, 4, root) != 0) {
42                                      cJSON_DeleteItemFromObjectCaseSensitive(root, "token"
    );
43                                      cJSON_DeleteItemFromObjectCaseSensitive(root, "userID
    ");
44                                      char *jsonString = cJSON_Print(cJSON_GetObjectItem(
    root, "rows"));
45                                      mg_http_reply(c, 200, "", jsonString);
46                                      free(jsonString);
47                                  }
48                                  else {
49                                      mg_http_reply(c, 200, "", "{\"result\": \"Usuário não
     encontrado!\"}\n");
50                                  }
51                              }
52                              else {
53                                  mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado!\"}\n");
54                              }
55                          }
56                          break;
57
58                      case 'P': // POST - Registrar usuário
```

Listing 4.2: Trecho da função de tratamento de eventos.

No corpo da função *event handler*, mais especificamente na linha 3, está definida a condição que filtra somente eventos do tipo HTTP, exatamente como explicado no exemplo anterior, desse modo quaisquer outros tipos de eventos serão ignorados. Por se tratar de um evento deste tipo, na linha 5 é feita a conversão da mensagem recebida para o formato *mg_http_message*, cujo resultado é atribuído à variável *hm*. A partir daí se pode extrair as informações dela, processo que ocorre já nas linhas 6 e 9.

Na linha 6 atribui-se à variável *method* a informação que indica o método HTTP

utilizado na requisição recebida, acessado por meio de um dos campos da estrutura *hm*. Como convencionado entre as partes envolvidas no projeto Tecnologias de Precisão para a Agropecuária, o formato das mensagens trocadas nas comunicações seria o JSON, portanto na linha 9 utiliza-se a biblioteca cJSON para criar um objeto deste tipo (variável *root*) e realizar o *parsing* da informação contida no *body* da mensagem HTTP. As demais variáveis declaradas nesse trecho são auxiliares, utilizadas ao longo do código para armazenamento temporário de dados recebidos/enviados ao banco de dados, bem como para construção das mensagens de resposta ao cliente.

Na linha 14 é chamada uma função que verifica se o *body* da mensagem recebida é nulo e, caso seja, envia uma resposta ao cliente e libera os recursos de memória alocados para o objeto cJSON. Se não for nulo, passa-se então ao escopo do *else* que inicia na linha 18. De acordo com a imagem apresentada na seção 3.1, este servidor foi planejado para responder a requisições com os métodos POST, GET e DELETE, dependendo da rota associada. Para melhor legibilidade de código, foi utilizada uma lógica com *switch* para identificar o método por meio do seu primeiro caractere. Dessa maneira, para evitar que métodos PUT e PATCH produzissem o mesmo resultado de um POST, foi necessário adicionar um trecho de verificação nas linhas 20, 21 e 22.

Após este ponto, todo o código consiste em verificar se a rota de destino da mensagem bate com alguma das definidas na Figura 6 e, caso positivo, utilizar um *switch* parar verificar o método HTTP utilizado (entre os disponíveis) e executar as ações pertinentes de maneira a produzir o resultado esperado para a rota e método utilizados.

Como exemplo, pode-se observar a implementação da rota */api/user* no Listing 3.2. Nesta parte consta o trecho que define o comportamento do servidor para uma requisição GET nesta rota. Se a requisição tiver esse formato, significa que o cliente está solicitando os dados de sua conta de usuário. Para obtê-los, a requisição deve conter ainda um *token*, passado em formato *String* dentro de um JSON por meio do *body* da mensagem HTTP.

Primeiramente, verifica-se na condição da linha 25 se a rota é a citada no parágrafo anterior, caso seja, o *switch* da linha 27 vai olhar para o primeiro caractere do método da requisição e o *case 'G'* vai identificar um GET. Na linha 31 é verificado se o objeto *root* carrega o *token* necessário para autorizar a ação. Se o *token* não for nulo, passa-se a uma consulta ao banco de dados que ocorre na linha 34 por meio da função *dbRead*.

O primeiro parâmetro dessa função é a conexão estabelecida com o banco de dados, que foi criada globalmente e inicializada na função *main* do servidor. O segundo parâmetro é um inteiro que serve para definir como será a *query* utilizada na consulta ao banco, uma vez que diversos tipos de consultas diferentes ocorrem no servidor, fez-se necessário criar um *switch* nesta função para este propósito. O último parâmetro é o objeto JSON que contém o *token*, que será usado para identificar o usuário. Dessa maneira, o retorno dessa consulta será o identificador único do usuário, diferente de 0, caso o *token* seja válido.

Se o identificador for diferente de 0, será conforme linha 38 adicionado ao objeto JSON e utilizado em uma nova consulta ao banco (linha 41), onde se espera obter informações sobre nome, *e-mail* e coordenadas de cercamento virtual dessa conta de usuário. Na sequência o *token* e o identificador serão excluídos do objeto JSON, que agora conterá apenas os dados requisitados pelo cliente. Na linha 44 esse objeto JSON será formatado para *String* e então enviado de volta ao cliente por meio da função *mg_http_reply*, liberando a memória no final.

Para teste desta funcionalidade, utilizou-se o Postman, um software cujo propósito é facilitar o desenvolvimento de API's, possibilitando a realização de testes, requisições HTTP, definições de cabeçalhos e corpo de mensagens, entre outras opções. Pode-se observar na imagem abaixo os dados atualmente armazenados na tabela *user* do banco de dados, mais especificamente para a linha cujo usuário possui o identificador 18.

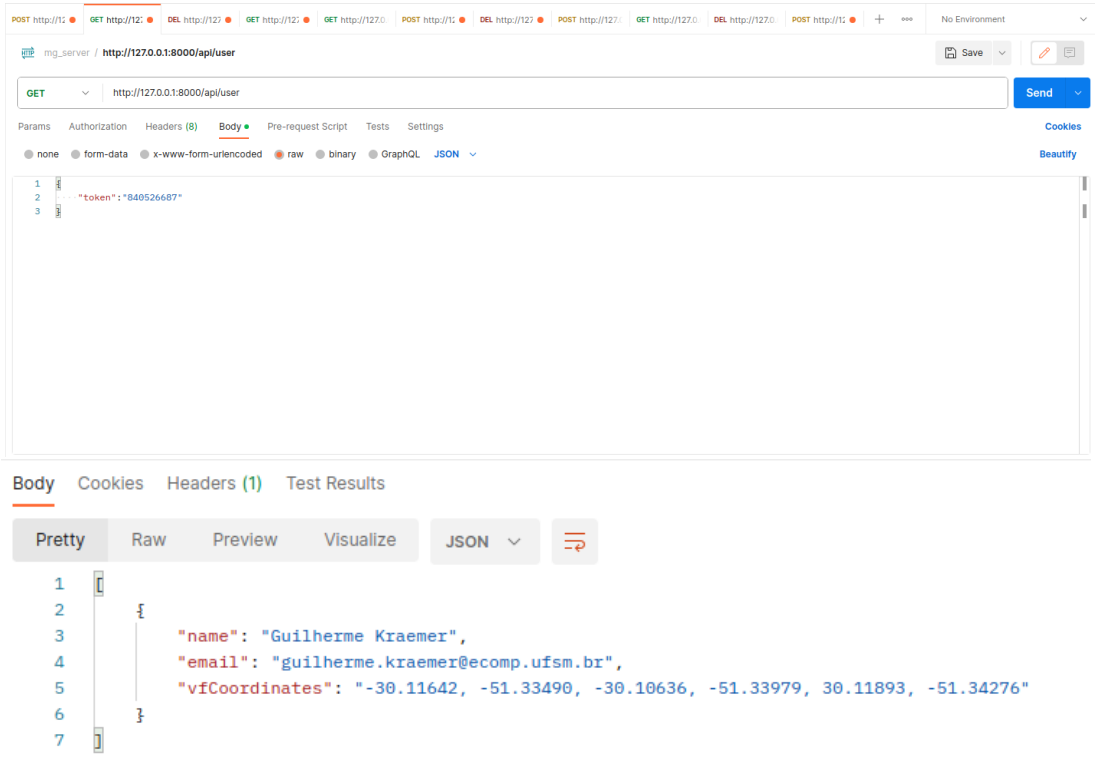
Figura 11 – Dados para teste na tabela *user* do banco de dados.

	id	name	email	password	vfCoordinates	token
Editar Copiar Remover	18	Guilherme Kraemer	guilherme.kraemer@ecomp.ufsm.br	12345678	-30.11642, -51.33490, -30.10636, -51.33979, 30.118...	840526687
Editar Copiar Remover	22	Guilherme Kraemer	gkraemer@ecomp.ufsm.br	1asdsd5678	0	413199984
Editar Copiar Remover	25	Guilherme Kraemer	gkraemer2024@ecomp.ufsm.br	1asdsd5678	-30.11642, -51.33490, -30.10636, -51.33979, 30.118...	1413862530
Editar Copiar Remover	26	Guilherme Kraemer	gkraemer2025@ecomp.ufsm.br	1asdsd5678	0	1290482309
Editar Copiar Remover	27	Guilherme Kraemer	gkraemer20257@ecomp.ufsm.br	1asdsd5678	0	284868340
Editar Copiar Remover	28	Guilherme Kraemer	gkraemer2027@ecomp.ufsm.br	1asdsd5678	0	2097358905
Editar Copiar Remover	29	Guilherme Kraemer	gkraemer2031@ecomp.ufsm.br	1asdsd5678	0	106442805

Fonte: Autor.

Será feito um teste que simula uma requisição de dados por parte deste usuário, que possui o token *840526687* armazenado em seu dispositivo *mobile*, que será automaticamente inserido no corpo da mensagem HTTP pelo cliente utilizado (aplicativo). A figura abaixo mostra o ambiente de testes Postman onde essa requisição é efetuada. Na imagem seguinte é apresentado o retorno (HTTP *response*), cujos dados podem ser conferidos na tabela *user* para se certificar de que estão corretos.

Figura 12 – Teste de requisição GET na rota */api/user*.



Fonte: Autor.

Para as demais funcionalidades do servidor, a implementação segue o mesmo padrão utilizado neste exemplo. Verifica-se qual é a rota, qual é o método e se o corpo da mensagem traz as informações necessárias para executar a ação solicitada. Caso positivo, executa-se aquilo que for necessário para cada caso, enviando ao cliente uma resposta no final. Em casos onde uma verificação tem retorno negativo, também é enviada ao cliente uma mensagem de resposta. O código completo do servidor encontra-se em anexo e também no *GitHub*.

5 DESENVOLVIMENTO DA SOLUÇÃO UTILIZANDO NODE IOT

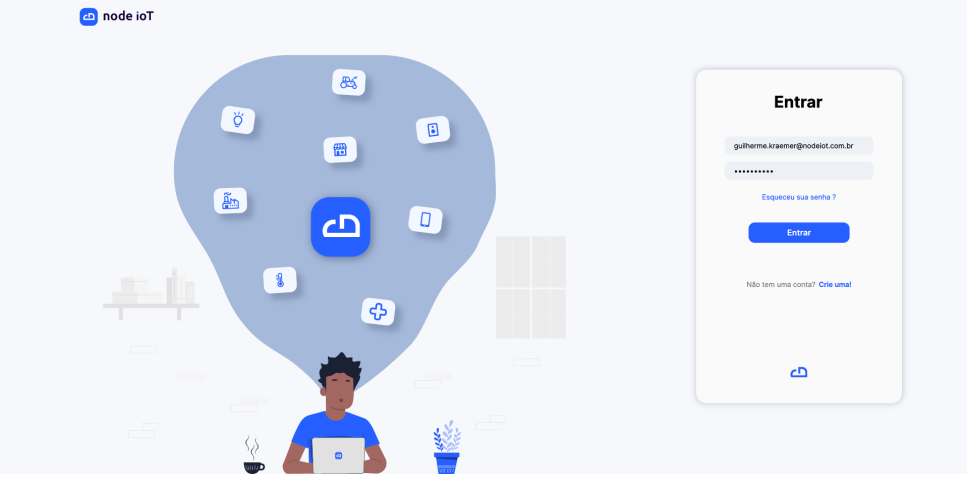
Como alternativa à utilização da biblioteca Mongoose, buscou-se uma ferramenta que pudesse trazer ganhos no ponto de vista prático de implementação, reduzindo o trabalho ou até eliminando, se possível, parte dos processos envolvidos nessa comunicação de dados. Nesse sentido, optou-se por utilizar a plataforma NodeIoT em relação à também citada neste trabalho Ubidots, e alguns motivos tiveram importância nesta escolha.

A Node IoT é uma empresa brasileira sediada no município de Guaíba, próximo à capital do Rio Grande do Sul, Porto Alegre. Entre seus fundadores, encontra-se um egresso do curso de graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Maria. Devido a essa proximidade acadêmica bem como interesses em comum com relação ao aperfeiçoamento e aquisição de conhecimentos na área de IoT, surgiu durante conversas a oportunidade de conhecer a plataforma IoT desenvolvida pela empresa e atualmente em fase de testes (*beta*).

Se por um lado o uso da plataforma neste trabalho seria uma boa maneira de testá-la, por outro se mostrou uma ótima abordagem alternativa à biblioteca Mongoose, sem contar no valor de poder contribuir com o desenvolvimento de uma empresa nacional em um mercado bastante promissor e que estará, nos próximos anos, absorvendo muitos formandos de cursos de tecnologia.

No caso de uma implementação utilizando a Node IoT, as necessidades de projeto passam a ser no sentido de integração, uma vez que a plataforma já possui uma estrutura construída para receber as comunicações, com servidor e banco de dados hospedados na nuvem (AWS). Para isso, primeiramente foi necessário criar uma conta de acesso *beta* e fazer *login* na plataforma em (Node IoT, 2023a).

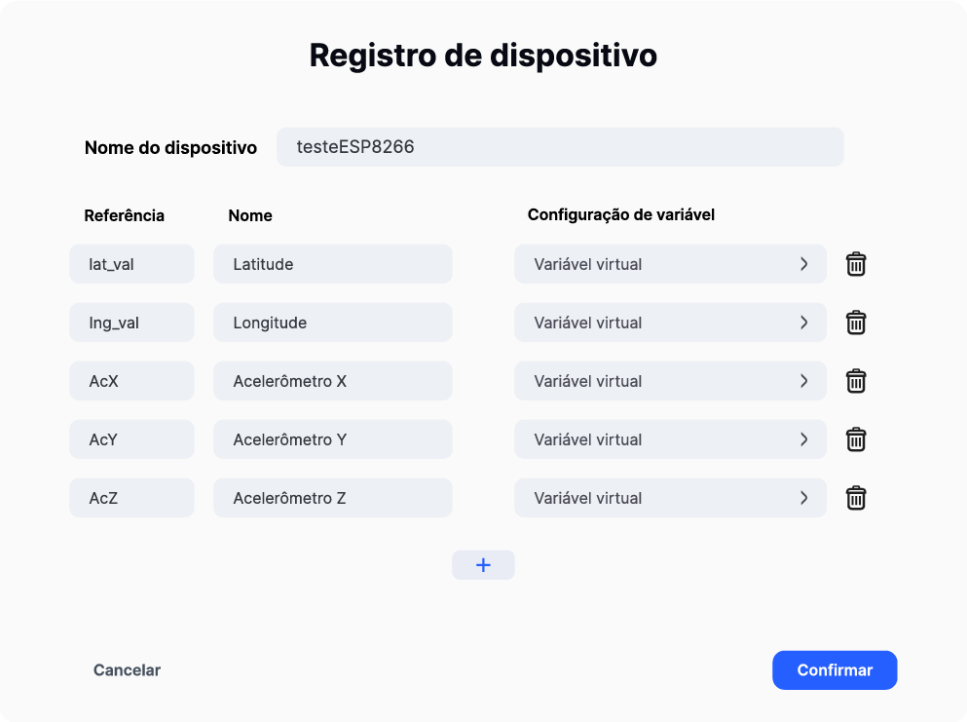
Figura 13 – Tela de *login* Node IoT.



Fonte: Adaptado de (Node IoT, 2023b).

Com o acesso realizado, inicia-se na opção Dispositivos do menu lateral esquerdo, onde é criado o dispositivo que será conectado à plataforma. Nesta etapa devem ser criadas variáveis para receber as informações coletadas pelos sensores integrados ao ESP8266 utilizado para testes no projeto Tecnologias de Precisão para a Agropecuária. Como os dados de interesse são providos por acelerômetro, giroscópio e GPS, a declaração de variáveis ocorreu de maneira semelhante ao que foi feito no banco de dados MySQL utilizando com o servidor Mongoose. Neste caso, a única diferença conforme imagem abaixo foi que não houve necessidade de uma informação temporal no conjunto de dados providos pelo ESP, uma vez que a plataforma já guarda informações referentes aos instantes de tempo em que ocorrem atualizações nos valores.

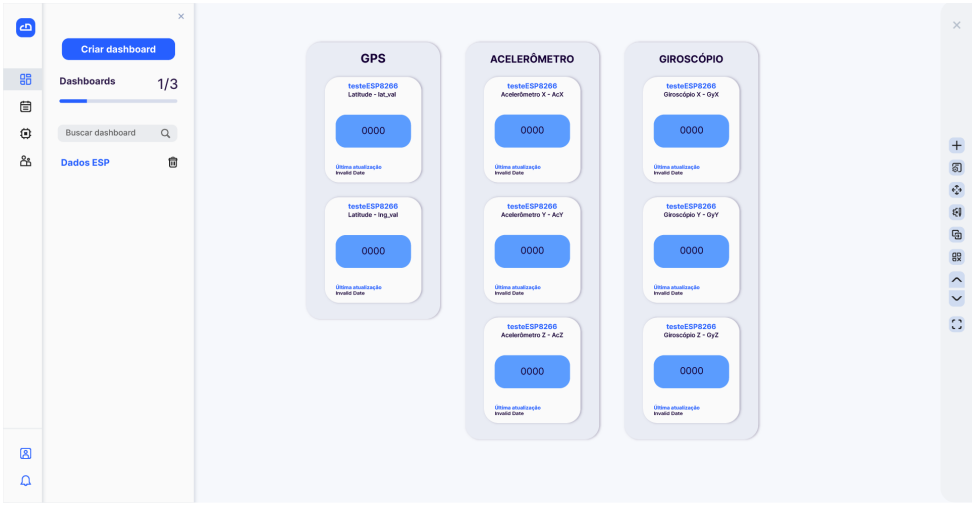
Figura 14 – Tela de criação de dispositivos Node IoT.



Fonte: Adaptado de (Node IoT, 2023b).

Portanto, as variáveis criadas no registro deste dispositivo foram: lat-val, lng-val, AcX, AcY, AcZ, GyX, GyY e GyZ. Agora, como meio de visualização dessas informações, passa-se à opção Dashboard do menu, que permite a criação de painéis personalizáveis para visualizar os dados. Aqui foi criado o dashboard Dados ESP, ao qual foram adicionados, utilizando as ferramentas da lateral direita da tela, componentes que permitem a visualização dos dados praticamente em tempo real, uma vez que o atraso médio da comunicação que sai do dispositivo embarcado, passa pelo servidor e chega ao *front-end* é de aproximadamente 500 ms.

Figura 15 – Dashboard Dados ESP.



Fonte: Adaptado de (Node IoT, 2023b).

Com o dashboard preparado, o próximo passo é integrar o código base de comunicação Node IoT ao *firmware* desenvolvido para testes com sensores no projeto de pesquisa. O *firmware* não faz parte do escopo delimitado para este trabalho em específico, contudo está englobado pelo mesmo projeto de pesquisa de modo que existe um relacionamento entre essas partes. Portanto, o código apresentado neste capítulo diz respeito apenas à integração com a plataforma e não inclui as funções e bibliotecas responsáveis pela parte de sensoriamento, entretanto as informações enviadas à plataforma são providas por estes sensores, por esse motivo haverá um destaque nestes trechos de código para lembrete deste detalhe.

O código base completo para realizar a integração entre *hardware* e plataforma se encontra disponível em anexo e também em repositório da Node IoT no GitHub. Na sequência serão mostradas imagens apenas dos trechos com as alterações para este projeto, de modo que possam ser comparadas com o original em anexo.

Basicamente, para conectar o dispositivo à plataforma e fazer o envio dos dados dos sensores, foram necessárias alterações em duas partes do código base: as variáveis para configuração de conexão *wi-fi* e autenticação do dispositivo na conta Node IoT e a função principal do ESP (*loop*). Na imagem abaixo é mostrada a alteração feita nas variáveis citadas, adequando-as para a rede *wi-fi* utilizada, nome da companhia definido na criação da conta Node IoT e nome do dispositivo definido em sua criação na plataforma.

```
1    const char ssid[] = "1010";
2    const char password[] = "dsetdonut30cm";
3
4    String companyName = "UFSM";  // Nome da companhia que o dispositivo está conectado
     // TODO: codificar os espaços
5    String deviceId = "testeESP8266"; // Nome do dispositivo mesmo nome configurado na
     Plataforma NodeIOT // TODO: codificar os espaços
```

Listing 5.1: Variáveis de conexão e autenticação.

Por sua vez, no *loop* foi utilizada a função *espPOST* para atualizar o valor de cada uma das variáveis definidas na plataforma com os dados obtidos localmente.

```
// Função principal
    void loop()
    {
        socketIO.loop();

        if(WiFi.status() != WL_CONNECTED)
        {
            WiFi.disconnect();
            Connected = false;
            LedControl(Connected);

            nodeIotConnection();
        }
        else if (!Connected)
        {
            LedControl(Connected);
            SocketIOConnect();
        }
        else
        {
            // código de acordo com a aplicação

            // segundo parâmetro representa a referência da variável criada no
    dispositivo na plataforma
            // terceiro parâmetro representa o valor que será enviado à variável
            espPOST(appPostData, "lat_val", lat_val);
            espPOST(appPostData, "lng_val", lng_val);
            espPOST(appPostData, "AcX", AcX);
            espPOST(appPostData, "AcY", AcY);
            espPOST(appPostData, "AcZ", AcZ);
            espPOST(appPostData, "GyX", GyX);
            espPOST(appPostData, "GyY", GyY);
            espPOST(appPostData, "GyZ", GyZ);
        }
    }
```

Listing 5.2: Envio de dados à plataforma no *loop.*

Com essas alterações, o código deve ser compilado e gravado no ESP8266 para proceder com a autenticação do dispositivo na plataforma. Assim que pronto, deve-se ligar o controlador e acessar a aba autenticação no dispositivo testeESP8266 criado na plataforma.

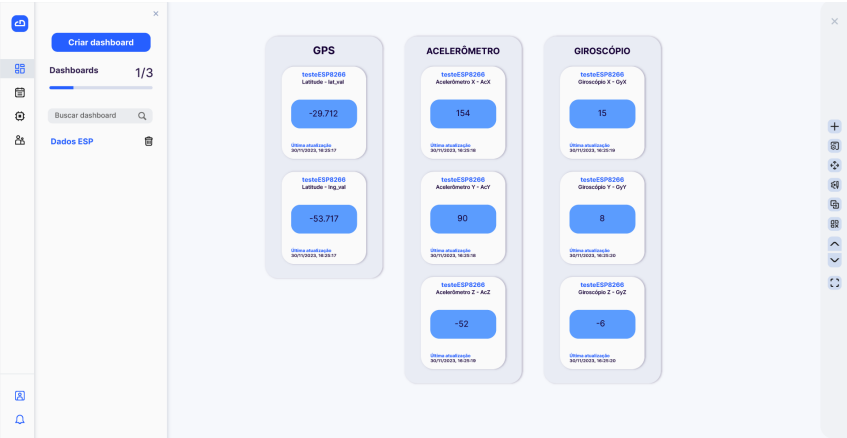
Figura 16 – Aba de autenticação do dispositivo.

Fonte: Autor.

Quando o ESP tiver feito a conexão com a internet, irá executar a função de autenticação com a plataforma, alterando o status e MAC mostrados na tela acima. Neste momento, uma confirmação será solicitada ao usuário e, quando feita, o dispositivo estará apto a transmitir os dados do sensoriamente para as variáveis criadas anteriormente.

Para visualização destes dados, basta acessar o dashboard criado anteriormente, que passará a receber as atualizações de valores, mostrando também o instante de tempo da alteração mais recente.

Figura 17 – Dashboard Dados ESP com o dispositivo operando.



Fonte: Autor.

6 AVALIAÇÃO DOS ASPECTOS PRÁTICOS DE IMPLEMENTAÇÃO

De maneira a atender os objetivos definidos para este estudo de caso, fez-se uma avaliação dos aspectos práticos envolvidos na implementação, comparando qualitativamente os diferentes métodos utilizados para integração de dados obtidos em sensores. A figura abaixo apresenta resumidamente alguns desses aspectos que são discorridos para cada uma das abordagens nas seções abaixo.

Figura 18 – Resumo comparativo dos aspectos práticos de implementação.

Fonte: Autor.

6.1 MONGOOSE WEB SERVER LIBRARY

Quanto ao Mongoose, a biblioteca possui recursos para servir como uma solução completa para projetos deste tipo, permitindo o desenvolvimento do cliente embarcado, servidor e interfaces *web* para visualização de dados. A documentação é abundante, organizada e com exemplos básicos que servem como ponto de partida, além de possuir uma comunidade ativa nos fóruns.

Por outro lado, todas essas partes requerem bastante trabalho adicional, sem mencionar nas horas de estudo necessárias para adquirir o entendimento de como utilizar a biblioteca e seus recursos. Neste trabalho em específico, onde a MWSL foi utilizada como uma solução parcial atuando apenas como *middleware*, o estudo e desenvolvimento do código envolvido, juntamente com o banco de dados, teve início há mais de 9 meses.

Obviamente, é importante ressaltar que essa é a experiência de um formando e participante de projeto de pesquisa e iniciação científica, provavelmente muito diferente das conclusões que seriam obtidas de um profissional com anos de carreira desenvolvendo um produto comercial. No entanto, as observações servem de base para conhecer os desafios envolvidos, bem como avaliar pontos positivos e negativos da biblioteca em condições semelhantes.

Sem um banco de dados criado e estruturado de acordo com as necessidades do projeto, com a organização adequada para acessar apenas o necessário de maneira simples, não seria possível definir como seriam as rotas do servidor. Portanto esta etapa tomou um tempo considerável no início, passando por refatorações até atingir um estado satisfatório. Somando isso com o planejamento e implementação do *middleware*, dezenas de horas foram consumidas nisso ao longo destes meses.

Um aspecto positivo é se ter um controle maior sobre tudo que acontece no sistema, de cima a baixo, possibilitando a utilização da biblioteca como solução parcial, a exemplo deste trabalho, bem como a definição da própria API. Isso torna o desenvolvimento mais flexível. Contudo, para conseguir concluir o trabalho dentro do prazo, optou-se por não utilizar a versão mais segura do protocolo (HTTPS), uma vez que envolvia uma carga de trabalho extra para entender como configurar e utilizar arquivos de certificação e chave primária.

Portanto, as configurações relativas à segurança da comunicação fazem parte das dificuldades encontradas no desenvolvimento. Junto a isso, a manipulação do formato JSON embora suportado pela própria MWSL, foi realizada utilizando a biblioteca *cJSON*, que traz funções e sintaxe mais amigáveis e passa uma noção melhor de um objeto. Diferente disso, as funções próprias da MWSL não conseguem abstrair as diversas estruturas envolvidas na manipulação de dados nesse formato.

Outro ponto positivo interessante é a disponiblidade das opções MQTT e WebSocket quanto ao protocolo utilizado na comunicação, que podem ser mais adequadas em certos casos, como largura de banda limitada e comunicação bidirecional em tempo real, por exemplo. Quanto ao gerenciamento e escalabilidade da solução, faz parte do trabalho do desenvolvedor avaliar as opções de hospedagem e garantir que o servidor consiga atender ao crescimento no número de solicitações caso ocorra uma escalada no número de clientes.

Por fim, com relação aos custos financeiros envolvidos, a biblioteca é *open-source*(GPLv2 *License*) e para uso dessa maneira a Cesanta oferece suporte apenas por meio de docu-

mentações e fóruns. Para suporte técnico prioritário e direto com engenheiros da empresa, existe a *Commercial License*, cujo custo não é explicitado no site e deve ser negociado via contato direto.

6.2 NODE IOT

Na solução utilizando Node IoT, tem-se a vantagem de contar com uma estrutura genérica e pronta para utilização nos mais diversos casos, entretanto por estar ainda em fase *beta*, não dispõe de documentações e exemplos em seu site ainda, de modo que foi necessário contato direto com a empresa para obter um tutorial para utilização da ferramenta.

Uma vez que o tutorial foi disponibilizado, a configuração do dispositivo e criação do *dashboard* para visualização dos dados foi bastante fácil e intuitiva, tomando apenas alguns minutos. Da mesma maneira, a integração do *firmware* para utilização dos sensores com o código base para comunicação com a plataforma é simples de fazer, destacando a agilidade que a ferramenta traz na produção de um MVP, possível de se realizar em poucas horas.

Somado a isso, a plataforma já conta com comunicação segura, além de abstrair quaisquer manipulações no que diz respeito ao servidor e banco de dados, poupando o desenvolvedor de trabalhos adicionais nesse sentido. Isso implica que não há como conhecer ou ter controle sobre o *core* da aplicação e qualquer problema ocorrido deve ser solucionado em contato com a empresa. A vantagem, além da agilidade já mencionada, é não se preocupar com o gerenciamento de tudo, podendo escalar o número de dispositivos e *dashboards* sabendo que a responsabilidade de manter tudo funcionando é da Node IoT.

Não foram encontradas dificuldades no desenvolvimento, seja na configuração realizada no ambiente *web* ou na integração de software embarcado. Por outro lado, durante a fase *beta* a plataforma não conta ainda com outras opções de protocolo de comunicação, o que pode ser um problema para ambientes onde o sinal é fraco e a conexão instável, uma vez que o HTTP não costuma funcionar bem sob essas condições.

Não existe custo ao usuário para uso da plataforma considerando uma conta de acesso *beta*, limitada a 2 dispositivos e 3 *dashboards*. Para realização de trabalhos acadêmicos, se necessário, a empresa pode ainda disponibilizar recursos adicionais na mesma modalidade, com a ressalva de que para estes casos o *hardware* e sensores devem ser adquiridos e integrados por conta do usuário. Para projetos pessoais ou comerciais os custos variam conforme número de dispositivos e *dashboards*, negociados diretamente com a empresa.

7 CONCLUSÃO

Cumprindo os objetivos estipulados para este estudo de caso, o trabalho apresenta o desenvolvimento e análise de dois métodos diferentes para integração de dados entre dispositivo embarcado e aplicativo *mobile*. Na abordagem utilizando a biblioteca Mongoose, o *middleware* implementado serve para atender às requisições tanto do dispostivo embarcado quanto do aplicativo *mobile*, armazenando e acessando informações em banco de dados conforme necessidade para fornecer a resposta adequada ao cliente. Com o servidor rodando localmente, todas as rotas implementadas foram testadas usando o *software* Postman, definindo a API utilizada para comunicação com o *hardware* embarcado e aplicativo, atualmente sendo desenvolvidos por outros alunos de graduação membros do projeto de pesquisa.

A implementação utilizando Node IoT é simples e extremamente ágil de se fazer, possibilitando a visualização dos dados por meio de *dashboards* na plataforma, que podem ser acessados tanto em dispositivos *mobile* quanto computadores utilizando o navegador *web*. Por outro lado, o conjunto de componentes disponíveis para uso em *dashboards* ainda é pequeno e entre eles não há algum que inclua integração com APIs como *Google Maps*, por exemplo.

Dessa maneira, para visualização de coordenadas e uso de funcionalidades como *heat map*, faz-se necessário uma implementação própria de componente para isso, que pode ser incluído ao *dashboard* por meio da ferramenta que permite inserir código em JavaScript. A Node IoT não dispõe ainda de aplicativo *mobile* próprio, então uma possibilidade opcional seria requisitar os dados por meio de API disponibilizada e visualizá-los em aplicativo próprio, como no caso Mongoose.

Conforme o resumo comparativo e análises apresentadas no capítulo de avaliação, percebe-se que a plataforma Node IoT tem uma clara vantagem no que diz respeito ao tempo empregado no desenvolvimento e gerenciamento da aplicação, uma vez que retira do usuário diversas das responsabilidades que teria utilizando Mongoose, por exemplo. Destacam-se também aspectos relacionados à personalização de interfaces (*dashboards*) e experiência de usuário, com funcionalidades simples e intuitivas de se usar.

Em contrapartida, é uma ferramenta ainda em fase *beta* e necessita de testes mais estressantes para provar sua robustez, principalmente sob condições em que a conexão é instável. De qualquer modo, é um produto muito promissor no segmento IoT e que surge com identidade própria, potencial de atrair atenção e crescer nos próximos anos.

A biblioteca Mongoose, por sua vez, está no mercado de maneira sólida há muitos anos, com suporte a uma variedade maior de arquiteturas de *hardware* e protocolos de comunicação, documentações bem escritas e com exemplos, além de comunidade ativa. Essas características trazem uma amplitude maior às possibilidades e modos de uso, po-

dendo atuar como solução parcial de diferentes maneiras, tal como é feito neste trabalho e em diversos projetos comerciais apresentados no site da Cesanta.

Adicionalmente, a grande quantidade de projetos concluídos traz também sua parcela de importância. No entanto, a carga de trabalho envolvida para construir algo do zero é bastante alta, aspecto negativo e que ressalta as diferenças de uma tecnologia desenvolvida em 2004 em relação a outra atual, como Node IoT. Cada um desses aspectos deve ser levado em consideração juntamente a uma análise precisa das necessidades de projeto, levando portanto a escolha a ser baseada nisso.

Espera-se que as análises e implementações desenvolvidas neste estudo sirvam de base e estímulo para o desenvolvimento de novos trabalhos na área de IoT. A contínua evolução deste campo representa um indicativo de sua importância, que já influencia mudanças na estrutura curricular de diversos cursos de nível superior no Brasil. Neste sentido, este trabalho apresenta uma nova tecnologia surgindo no mercado comparando-a com outra já conceituada, assim contribui com o desenvolvimento do conhecimento em IoT.

REFERÊNCIAS

Agustin Pelaez. **Building a People Counter with Raspberry Pi and Ubidots**. Ubidots, 2023. Acesso em outubro de 2023. Disponível em: <https://help.ubidots.com/en/articles-/927987-building-a-people-counter-with-raspberry-pi-and-ubidots>.

Apache Friends. Apache Friends, 2023. Acesso em novembro de 2023. Disponível em: <https://www.apachefriends.org/pt_br/index.html>.

Cesanta Software Ltd. **Mongoose CASE STUDY: Veethree**. Cesanta, 2023. Acesso em outubro de 2023. Disponível em: <https://mongoose.ws/case-studies/veethree/>.

____. **Mongoose Features**. Cesanta, 2023. Acesso em outubro de 2023. Disponível em: <https://mongoose.ws/features/>.

COMER, D. E. **Redes de Computadores e Internet**. 6. ed. [S.l.]: Bookman, 2016. 557 p.

DEVMEDIA. **JSON Tutorial**. DEVMEDIA, 2012. Acesso em novembro de 2023. Disponível em: <https://www.devmedia.com.br/json-tutorial/25275>.

Diego Melo. **O que é JSON? [Guia para Iniciantes]**. Tecnoblog, 2021. Acesso em novembro de 2023. Disponível em: <https://tecnoblog.net/responde/o-que-e-json-guia-para-iniciantes/>.

ISAACSON, W. **Os Inovadores**: Uma biografia da revolução digital. [S.l.]: Editora Companhia das Letras, 2014. 568 p.

KERSCHBAUMER, R. **Microcontroladores**. [S.l.], 2018. 181 p. Acesso em novembro de 2023. Disponível em: <https://professor.luzerna.ifc.edu.br/ricardo-kerschbaumer/wp-content/uploads/sites/43/2018/02/Apostila-Microcontroladores.pdf>.

Matheus Cardoso. **O Que É Um Microcontrolador?** IEEE RAS UFCG, 2020. Acesso em novembro de 2023. Disponível em: <https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/>.

Melissa Cruz Cossetti. **O que é DNS?** Tecnoblog, 2018. Acesso em novembro de 2023. Disponível em: <https://tecnoblog.net/responde/o-que-e-dns/>.

Node IoT. **ACESSO BETA**. 2023. Acesso em novembro de 2023. Disponível em: <https:-//nodeiot.com.br/>.

____. **Plataforma Node IoT**. 2023. Acesso em novembro de 2023. Disponível em: <https:-//nodeiot.app.br/>.

Oracle Corporation. **O que é IoT?** Oracle, 2019. Acesso em novembro de 2023. Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot/#why-is-iot-important>.

SACOMANO, J. B.; GONçALVES, R. F.; BONILLA, S. H. **Indústria 4.0 : conceitos e fundamentos**. São Paulo: Editora Edgard Blücher Ltda, 2018. 182 p.

Santhosh Kumar P.B. **CASE STUDY: Schneider Electric**. Cesanta, 2023. Acesso em outubro de 2023. Disponível em: <https://mongoose.ws/case-studies/schneider-electric/>.

SIEBEL, T. M. **Transformação Digital: Como Sobreviver e Prosperar em uma Era de Extinção em Massa**. Rio de Janeiro: Editora Alta Books, 2021. 251 p.

SOUZA, C. L. de; TEJADA, C. T. **Tecnologias Disruptivas, Digitalização de Processos e Integração de Sistemas: O Desafio da Indústria 4.0**. 2022. 34 f. 24. Monografia (Trabalho de Conclusão de Curso de Especialização) — Curso de Especialização em Especialista de Engenharia de Automação e Controle Industrial, Porto Alegre, 2022. Acesso em novembro de 2023.

TANENBAUM, A. S.; WETHERALL, D. **Redes de Computadores**. 5. ed. São Paulo: Pearson Education, Inc, 2011. 600 p.

Ubidots. **How Ubidots works?** Ubidots, 2023. Acesso em outubro de 2023. Disponível em: <https://docs.ubidots.com/reference/how-ubidots-works>.

ANEXO A – CÓDIGO-FONTE SERVIDOR MONGOOSE

```
#include "dbFunctions.c"

int checkParse (cJSON *msg);
long int generateToken();

MYSQL *dbConnection;

static const char *s_http_addr = "http://127.0.0.1:8000";    // HTTP port

// The function is called every time an event happens

static void event_handler(struct mg_connection *c, int ev, void *ev_data, void *fn_data)
    {

    if(ev == MG_EV_HTTP_MSG) { // Identifies HTTP connection

        struct mg_http_message *hm = (struct mg_http_message *) ev_data;
        const char *method = hm->method.ptr;
        char tokenStr[32];
        int i, id, deviceID;
        cJSON *root = cJSON_Parse(hm->body.ptr);
        cJSON *aux, *arrayItem;

        printf("\n\t...message...\n%s\n", hm->message.ptr);

        if (checkParse(root) == 0) {
            mg_http_reply(c, 200, "", "{\"result\": \"Body da mensagem é nulo!\"}\n");
            cJSON_Delete(root);
        }
        else {

            if (method[1] == 'U' || method[1] == 'A') { // PUT e PATCH indisponíveis
                mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível.\"}\n");
            }

            //rota --- /api/user
            else if(mg_http_match_uri(hm, "/api/user")) {

                switch(method[0]) {

                    case 'G': // GET - Obtém informações do usuário

                        if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {

                            // verifica qual usuário possui o token informado
                            id = dbRead(dbConnection, 2, root);

                            if (id != 0) {

                                cJSON_AddNumberToObject(root, "userID", id);

                                // consulta os dados do usuário
                                if (dbRead(dbConnection, 4, root) != 0) {
                                    cJSON_DeleteItemFromObjectCaseSensitive(root, "token"
    );
```

```
54                                  cJSON_DeleteItemFromObjectCaseSensitive(root, "userID
   ");
55                                  char *jsonString = cJSON_Print(cJSON_GetObjectItem(
   root, "rows"));
56                                  mg_http_reply(c, 200, "", jsonString);
57                                  free(jsonString);
58                              }
59                              else {
60                                  mg_http_reply(c, 200, "", "{\"result\": \"Usuário não
    encontrado!\"}\n");
61                              }
62                          }
63                          else {
64                              mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
   com o token informado!\"}\n");
65                          }
66                      }
67                      break;
68
69                  case 'P': // POST - Registrar usuário
70
71                      // converte o token em uma string e adiciona ao objeto cJSON root
72                      snprintf(tokenStr, sizeof(tokenStr), "%ld", generateToken());
73                      cJSON_AddStringToObject(root, "token", tokenStr);
74
75                      // registra usuário se não houver existente com o email informado
76                      if(dbWrite(dbConnection, 1, root) == 1){
77                          char jsonString[50] = "";
78                          sprintf(jsonString, "{\"token\": \"%s\"}", tokenStr);
79                          printf("%s\n", jsonString);
80                          mg_http_reply(c, 200, "", jsonString);
81                      }
82                      else{
83                          mg_http_reply(c, 200, "", "{\"result\": \"Já existe usuário
   registrado com o email informado!\"}\n");
84                      }
85                      break;
86
87                  case 'D':  // DELETE - Excluir usuário, dispositivos e dados
   associados
88
89                      if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
90
91                          // verifica qual usuário possui o token informado
92                          id = dbRead(dbConnection, 2, root);
93
94                          if (id != 0) {
95
96                              cJSON_AddNumberToObject(root, "userID", id);
97
98                              // consulta os dados do usuário
99                              if (dbDelete(dbConnection, 1, root) != 0) {
100
101                                 mg_http_reply(c, 200, "", "{\"result\": \"Usuário
   excluído com sucesso.\"}\n");
102                             }
103                             else {
104                                 mg_http_reply(c, 200, "", "{\"result\": \"Falha ao
   excluir usuário.\"}\n");
```

```
105                                 }
106                             }
107                             else {
108                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário com o token informado.\"}\n");
109                             }
110                         }
111                         break;
112 
113                     default:
114                         mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível.\"}\n");
115                         break;
116                 }
117             }
118             //rota --- /api/user/device
119             else if(mg_http_match_uri(hm, "/api/user/device")){
120 
121                 switch(method[0]) {
122 
123                     case 'G': //GET obtém o MAC de todos os dispositivos do usuário
124 
125                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
126 
127                             // verifica qual usuário possui o token informado
128                             id = dbRead(dbConnection, 2, root);
129 
130                             if (id != 0) {
131 
132                                 cJSON_AddNumberToObject(root, "userID", id);
133 
134                                 // consulta os dados do usuário
135                                 if (dbRead(dbConnection, 5, root) != 0) {
136                                     cJSON_DeleteItemFromObjectCaseSensitive(root, "token");
137                                     cJSON_DeleteItemFromObjectCaseSensitive(root, "userID");
138                                     char *jsonString = cJSON_Print(root);
139                                     printf("%s\n", jsonString);
140                                     mg_http_reply(c, 200, "", jsonString);
141                                     free(jsonString);
142                                 }
143                                 else {
144                                     mg_http_reply(c, 200, "", "{\"result\": \"Usuário não encontrado!\"}\n");
145                                 }
146                             }
147                             else {
148                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário com o token informado!\"}\n");
149                             }
150                         }
151                         break;
152 
153                     default:
154                         mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível.\"}\n");
155                         break;
156                 }
```

```
157             }
158             // rota --- /api/user/login
159             else if (mg_http_match_uri(hm, "/api/user/login")) {
160 
161                 switch(method[0]) {
162 
163                     case 'G': //GET verifica se o email e senha estão corretos
164 
165                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
166 
167                             // verifica qual usuário possui o token informado
168                             id = dbRead(dbConnection, 2, root);
169 
170                             if (id != 0) {
171 
172                                 // consulta os dados do usuário
173                                 if (dbRead(dbConnection, 6, root) == id) {
174                                     mg_http_reply(c, 200, "", "{\"result\": \"Login
    efetuado com sucesso!\"}\n");
175                                 }
176                                 else {
177                                     mg_http_reply(c, 200, "", "{\"result\": \"Dados
    incorretos!\"}\n");
178                                 }
179                             }
180                             else {
181                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado!\"}\n");
182                             }
183                         }
184                         break;
185 
186                     default:
187                         mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível
    .\"}\n");
188                         break;
189                 }
190 
191             }
192             //rota --- /api/user/coordinates
193             else if(mg_http_match_uri(hm, "/api/user/coordinates")){
194 
195                 switch(method[0]) {
196 
197                     case 'P': // post de coordenadas para cercamento virtual
198 
199                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
200 
201                             // verifica qual usuário possui o token informado
202                             id = dbRead(dbConnection, 2, root);
203 
204                             if (id != 0) {
205 
206                                 if (cJSON_GetObjectItem(root, "coordinates")->valuestring
     != NULL){
207 
208                                     // insere os dados no banco
209                                     if(dbWrite(dbConnection, 4, root) == 1){
210                                         mg_http_reply(c, 200, "", "{\"result\": \"
```

```
    Coordenadas registradas com sucesso!\"}\n");
211                                 }
212                                 else {
213                                     mg_http_reply(c, 200, "", "{\"result\": \"
    Coordenadas já armazenadas anteriormente!\"}\n");
214                                 }
215 
216                             }
217                         }
218                         else {
219                             mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado!\"}\n");
220                         }
221                     }
222                     break;
223 
224                 default:
225                     mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível
    .\"}\n");
226                     break;
227             }
228 
229         }
230         //rota --- /api/device
231         else if (mg_http_match_uri(hm, "/api/device")) {
232 
233             switch(method[0]) {
234 
235                 case 'D':   // delete device
236 
237                     if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
238 
239                         // verifica qual usuário possui o token informado
240                         id = dbRead(dbConnection, 2, root);
241 
242                         if (id != 0) {
243 
244                             cJSON_AddNumberToObject(root, "userID", id);
245 
246                             // consulta os dados do usuário
247                             if (dbDelete(dbConnection, 2, root) != 0) {
248 
249                                 mg_http_reply(c, 200, "", "{\"result\": \"Dispositivo
     excluído com sucesso.\"}\n");
250                             }
251                             else {
252                                 mg_http_reply(c, 200, "", "{\"result\": \"Falha ao
    excluir dispositivo.\"}\n");
253                             }
254                         }
255                         else {
256                             mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado.\"}\n");
257                         }
258                     }
259                     break;
260 
261                 case 'P':   // device register
262 
```

```
263                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
264
265                             // verifica qual usuário possui o token informado
266                             id = dbRead(dbConnection, 2, root);
267
268                             if(id != 0){
269
270                                 cJSON_AddNumberToObject(root, "userID", id);
271
272                                 // insere os dados no banco
273                                 if(dbWrite(dbConnection, 2, root) == 1){
274                                     mg_http_reply(c, 200, "", "{\"result\": \"Dispositivo
     registrado com sucesso!\"}\n");
275                                 }
276                                 else {
277                                     mg_http_reply(c, 200, "", "{\"result\": \"Já existe
    dispositivo registrado com o MAC informado!\"}\n");
278                                 }
279                             }
280                             else {
281                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado!\"}\n");
282                             }
283                         }
284                         else {
285                             mg_http_reply(c, 200, "", "{\"result\": \"Token nulo!\"}\n");
286                         }
287                         break;
288
289                     default:
290                         mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível
    .\"}\n");
291                         break;
292                 }
293             }
294             else if(mg_http_match_uri(hm, "/api/data")){
295
296                 switch(method[0]){
297
298                     //
299                     case 'G': // obtém os dados do dispositivo informado (no período
    informado ou o registro completo)
300
301                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
302
303                             // verifica qual usuário possui o token informado
304                             id = dbRead(dbConnection, 2, root);
305
306                             if (id != 0) {
307
308                                 cJSON_AddNumberToObject(root, "userID", id);
309
310                                 // verifica qual dispositivo possui o MAC informado
311                                 deviceID = dbRead(dbConnection, 3, root);
312
313                                 printf("device ID: %i\n", deviceID);
314                                 if (deviceID != 0) {
315
316                                     cJSON_AddNumberToObject(root, "deviceID", deviceID);
```

```
317
318                                     // todos os dados do dispositivo
319                                     if (dbRead(dbConnection, 7, root) != 0) {
320                                         cJSON_DeleteItemFromObjectCaseSensitive(root, "
    token");
321                                         cJSON_DeleteItemFromObjectCaseSensitive(root, "
    MAC");
322                                         cJSON_DeleteItemFromObjectCaseSensitive(root, "
    deviceID");
323                                         cJSON_DeleteItemFromObjectCaseSensitive(root, "
    userID");
324
325                                         char *jsonString = cJSON_Print(root);
326                                         mg_http_reply(c, 200, "", jsonString);
327                                         free(jsonString);
328                                     }
329                                 }
330                             }
331                             else {
332                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
    com o token informado!\"}\n");
333                             }
334                         }
335                         break;
336
337                     // Deleta todos os dados referentes ao dispositivo
338                     case 'D':
339
340                         if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {
341
342                             // verifica qual usuário possui o token informado
343                             id = dbRead(dbConnection, 2, root);
344
345                             if (id != 0) {
346
347                                 cJSON_AddNumberToObject(root, "userID", id);
348
349                                 // verifica qual dispositivo possui o MAC informado
350                                 deviceID = dbRead(dbConnection, 3, root);
351
352                                 printf("device ID: %i\n", deviceID);
353                                 if (deviceID != 0) {
354
355                                     cJSON_AddNumberToObject(root, "deviceID", deviceID);
356
357                                     // exclui todos os dados do dispositivo
358
359                                     if(dbDelete(dbConnection, 3, root) == 1){
360                                         mg_http_reply(c, 200, "", "{\"result\": \"Dados
    excluídos com sucesso!\"}\n");
361                                     }
362                                     else{
363                                         mg_http_reply(c, 200, "", "{\"result\": \"Falha
    ao excluir os dados!\"}\n");
364                                     }
365                                 }
366                             }
367                             else {
368                                 mg_http_reply(c, 200, "", "{\"result\": \"Não há usuário
```

```
com o token informado!\"}\n");
                            }
                        }
                        break;

                    // data register
                    case 'P':

                        if (cJSON_GetObjectItem(root, "token")->valuestring != NULL) {

                            // verifica qual usuário possui o token informado
                            id = dbRead(dbConnection, 2, root);

                            if(id != 0){

                                cJSON_AddNumberToObject(root, "userID", id);

                                // verifica qual dispositivo possui o MAC informado
                                deviceID = dbRead(dbConnection, 3, root);

                                if (deviceID != 0) {

                                    cJSON_AddNumberToObject(root, "deviceID", deviceID);

                                    // insere os dados no banco
                                    if (dbWrite(dbConnection, 3, root) == 1){
                                        mg_http_reply(c, 200, "", "{\"result\": \"Dados
registrados com sucesso!\"}\n");
                                    }
                                    else {
                                        mg_http_reply(c, 200, "", "{\"result\": \"Erro na
 inserção de dados.\"}\n");
                                    }
                                }
                                else {
                                    mg_http_reply(c, 200, "", "{\"result\": \"MAC
incorreto/inexistente.\"}\n");
                                }
                            }
                            else {
                                mg_http_reply(c, 200, "", "{\"result\": \"Token incorreto
.\"}\n");
                            }
                        }
                        break;

                    default:
                        mg_http_reply(c, 200, "", "{\"result\": \"Método indisponível
.\"}\n");
                        break;
                }
            }
            else {
                mg_http_reply(c, 200, "", "{\"result\": \"Rota não encontrada.\"}\n");
            }
            cJSON_Delete(root);
        }
    }
    /* else {
```

```
422         mg_http_reply(c, 200, "", "{\"result\": \"Evento desconhecido.\"}\n");
423     } */
424 }
425
426 // verifica erro no cJSON_Parse()
427 int checkParse (cJSON *msg) {
428     if (msg == NULL) {
429         // Erro de parsing
430         printf("Erro no parsing do JSON.\n");
431         return 0;
432     }
433     else return 1;
434 }
435
436 long int generateToken(){
437     srand((unsigned)time(NULL));
438     return rand();
439 }
440
441 int main (void) {
442
443     struct mg_mgr mgr;                          // Event manager
444     mg_log_set(MG_LL_DEBUG);                    // Set log level
445
446     dbConnection = getConnection();           // Connect to database
447
448     mg_mgr_init(&mgr);                          // Initialise event manager
449
450     mg_http_listen(&mgr, s_http_addr, event_handler, NULL);//(void *) &connection  //
    Create HTTP listener
451
452     for (;;) mg_mgr_poll(&mgr, 1000);                   // Infinite event loop
453     mg_mgr_free(&mgr);
454
455     return 0;
456 }
```

Listing A.1: mg-http-server.c

```
1  /* Arquivo com funções relacionadas ao banco de dados */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <mysql.h>
6  #include <string.h>
7  #include "mongoose.h"
8  #include "mongoose.c"
9  #include "cJSON.h"
10 #include "cJSON.c"
11
12 void errorMsg(MYSQL *connection) {
13     fprintf(stderr, "\n%s\n", mysql_error(connection));
14     mysql_close(connection);
15     exit(1);
16 }
17
18 MYSQL * getConnection(){
19     MYSQL *conexao;
20     MYSQL_RES *res;
21     MYSQL_ROW row;
```

```
22      char *server = "127.0.0.1";
23      char *user = "root";
24      char *password = ""; /* set me first */
25      char *database = "disp_rastreador";
26
27      conexao = mysql_init(NULL);
28
29      /* Connect to database */
30      if (!mysql_real_connect(conexao, server, user, password, database, 0, NULL, 0)) {
31          printf("Erro na conexao!\n");
32          errorMsg(conexao);
33      }
34      else {
35          printf("Conexao realizada com sucesso!\n");
36      }
37
38      return conexao;
39  }
40
41  int dbWrite(MYSQL *connection, int wType, cJSON *data){
42
43      char query[250];
44
45      // wType representa o tipo de escrita a ser feita
46      switch (wType){
47
48          // registro de usuário (name, email, password, token)
49          case 1:
50              sprintf(query, "INSERT INTO user(name, email, password, token) VALUES ('%s',
        '%s', '%s', '%s');",
51                  cJSON_GetObjectItem(data, "name")->valuestring,
52                  cJSON_GetObjectItem(data, "email")->valuestring,
53                  cJSON_GetObjectItem(data, "password")->valuestring,
54                  cJSON_GetObjectItem(data, "token")->valuestring);
55              break;
56
57          // registro de dispositivo (MAC, user.id)
58          case 2:
59              sprintf(query, "INSERT INTO device(MAC, userID) VALUES ('%s', '%i');",
60                  cJSON_GetObjectItem(data, "MAC")->valuestring,
61                  cJSON_GetObjectItem(data, "userID")->valueint);
62              break;
63
64          // registro de dados (deviceID, dateTime, longitude, latitude, acx, acy, acz, gyx
        , gyy, gyz)
65          case 3:
66              sprintf(query, "INSERT INTO data(deviceID, dateTime, longitude, latitude, acx
        , acy, acz, gyx, gyy, gyz) VALUES ('%i', '%s', '%s', '%s', '%s', '%s', '%s', '%s', '%
        s', '%s');",
67                  cJSON_GetObjectItem(data, "deviceID")->valueint,
68                  cJSON_GetObjectItem(data, "dateTime")->valuestring,
69                  cJSON_GetObjectItem(data, "longitude")->valuestring,
70                  cJSON_GetObjectItem(data, "latitude")->valuestring,
71                  cJSON_GetObjectItem(data, "acx")->valuestring,
72                  cJSON_GetObjectItem(data, "acy")->valuestring,
73                  cJSON_GetObjectItem(data, "acz")->valuestring,
74                  cJSON_GetObjectItem(data, "gyx")->valuestring,
75                  cJSON_GetObjectItem(data, "gyy")->valuestring,
76                  cJSON_GetObjectItem(data, "gyz")->valuestring);
```

```
77              break;
78
79          // registro de coordenadas para cercamento virtual
80          case 4:
81              sprintf(query, "UPDATE user SET vfCoordinates = '%s' WHERE token = '%s';",
    cJSON_GetObjectItem(data, "coordinates")->valuestring,
82                  cJSON_GetObjectItem(data, "token")->valuestring);
83              break;
84      }
85
86      printf("%s\n", query);
87      if (mysql_query(connection, query)) {
88
89          printf("\nErro ao inserir no banco de dados!\n");
90          errorMsg(connection);
91          return 0;
92      }
93      else {
94
95          printf("\nDados inseridos com sucesso!\n");
96          return 1;
97      }
98  }
99
100 int dbRead(MYSQL *connection, int rType, cJSON *data) {
101
102     MYSQL_RES *res;
103     MYSQL_ROW row;
104     MYSQL_FIELD *current_field, *fields;
105     char query[250];
106     char *field_name, *field_value;
107     int value;
108     unsigned int num_fields, i;
109     unsigned long *lengths;
110     cJSON *dataArray;
111     cJSON *rowObject;
112
113
114     // switch utilizado para montagem da query
115     switch (rType) {
116
117         case 1:
118             sprintf(query, "SELECT id FROM user WHERE email = '%s';", cJSON_GetObjectItem
    (data, "email")->valuestring);
119             break;
120
121         case 2:
122             sprintf(query, "SELECT id FROM user WHERE token = '%s';", cJSON_GetObjectItem
    (data, "token")->valuestring);
123             break;
124
125         case 3:
126             sprintf(query, "SELECT id FROM device WHERE userID = %i AND MAC = '%s';",
127                 cJSON_GetObjectItem(data, "userID")->valueint,
128                 cJSON_GetObjectItem(data, "MAC")->valuestring);
129             break;
130
131         case 4:
132             sprintf(query, "SELECT name, email, vfCoordinates FROM user WHERE id = %i;",
```

```
    cJSON_GetObjectItem(data, "userID")->valueint);
            break;

        case 5:
            sprintf(query, "SELECT MAC FROM device WHERE userID = %i;",
    cJSON_GetObjectItem(data, "userID")->valueint);
            break;

        case 6:
            sprintf(query, "SELECT id FROM user WHERE email = '%s' and password = '%s';",
                cJSON_GetObjectItem(data, "email")->valuestring,
                cJSON_GetObjectItem(data, "password")->valuestring);
            break;

        case 7:
            sprintf(query, "SELECT dateTime, longitude, latitude, acx, acy, acz, gyx, gyy
    , gyz FROM data WHERE deviceID = %i;", cJSON_GetObjectItem(data, "deviceID")->
    valueint);
            break;

    }
    printf("query %s", query);

    if (mysql_query(connection, query)) {
        printf("\nErro ao consultar no banco de dados!\n");
        errorMsg(connection);
        return 0;
    }

    printf("\nConsulta realizada com sucesso!\n");

    res = mysql_use_result(connection);

    if (res == NULL) {
        printf("res = NULL\n");
        errorMsg(connection);
        mysql_free_result(res);
        return 0;
    }
    else {
        if ((rType == 2) || (rType == 3) || (rType == 6)) {
            row = mysql_fetch_row(res);
            value = atoi(row[0]);
            mysql_free_result(res);
            return value;
        }
        else {
            num_fields = mysql_num_fields(res);

            fields = mysql_fetch_fields(res);

            dataArray = cJSON_CreateArray();

            while ((row = mysql_fetch_row(res))) {
                //lengths = mysql_fetch_lengths(res);
                rowObject = cJSON_CreateObject();

                for (i = 0; i < num_fields; i++) {

```

```
188                     field_name = fields[i].name;
189                     field_value = row[i] ? row[i] : "NULL";
190 
191                     //printf("[%s : %s] ", field_name, field_value);
192 
193                     if(cJSON_AddStringToObject(rowObject, field_name, field_value) ==
    NULL){
194                         printf("Erro ao adicionar ao objeto cJSON rowObject.\n");
195                     }
196                 }
197                 cJSON_AddItemToArray(dataArray, rowObject);
198                 printf("\n");
199 
200             }
201             cJSON_AddItemToObject(data, "rows", dataArray);
202         }
203     }
204 
205     mysql_free_result(res);
206     return (-1);
207 }
208 
209 int dbDelete(MYSQL *connection, int dType, cJSON *data) {
210 
211     char query[250];
212 
213     switch (dType) {
214 
215         case 1:
216             sprintf(query, "DELETE FROM user WHERE id = %d;", cJSON_GetObjectItem(data, "
    userID")->valueint);
217             break;
218 
219         case 2:
220             sprintf(query, "DELETE FROM device WHERE MAC = '%s' AND userID = %d;",
    cJSON_GetObjectItem(data, "MAC")->valuestring, cJSON_GetObjectItem(data, "userID")->
    valueint);
221             break;
222 
223         case 3:
224             sprintf(query, "DELETE FROM data WHERE deviceID = %d;", cJSON_GetObjectItem(
    data, "deviceID")->valueint);
225             break;
226     }
227 
228     printf("%s\n", query);
229 
230     if (mysql_query(connection, query)) {
231 
232         printf("\nErro ao excluir do banco de dados!\n");
233         errorMsg(connection);
234         return 0;
235     }
236     else {
237 
238         printf("\nDados excluídos com sucesso!\n");
239         return 1;
240     }
```

```
241 }
```

Listing A.2: dbFunctions.c

ANEXO B – CÓDIGO-BASE PARA INTEGRAÇÃO COM A PLATAFORMA NODE IOT

```
/*
########################################################################
##      Integração das tecnologias da REMOTE IO com Node IOT          ##
##                          Version 1.1                               ##
##   Código base para implementação de projetos de digitalização de   ##
##   processos, automação, coleta de dados e envio de comandos com    ##
##   controle embarcado e na nuvem.                                   ##
##                                                                    ##
########################################################################
*/

#define USE_SERIAL Serial
#define JSON_DOCUMENT_CAPACITY 1024

#include <Arduino.h>
#include <ArduinoJson.h>

#if defined(ESP8266)
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#elif defined(ESP32)
#include <WiFi.h>
#include <HTTPClient.h>
#endif

#include <WebSocketsClient.h>
#include <SocketIOclient.h>

SocketIOclient socketIO;

const char ssid[] = SSID;
const char password[] = PASSWORD;

String companyName = COMPANY_NAME;  // Nome da companhia que o dispositivo está conectado
    // TODO: codificar os espaços
String deviceId = DEVICE_ID; // Nome do dispositivo mesmo nome configurado na Plataforma
    NodeIOT // TODO: codificar os espaços

String appHost = "54.88.219.77";
uint16_t appPort = 5000;

String appBaseUrl = "http://" + appHost + ':' + appPort + "/api";
String appVerifyUrl = appBaseUrl + "/devices/verify";
String appLastDataUrl = appBaseUrl + "/devices/getdata/" + companyName + "/" + deviceId;
String appPostData = appBaseUrl + "/broker/data/";

String appSocketPath = "/socket.io/?companyName=%22" + companyName + "%22&deviceId=%22" +
    deviceId + "%22&EIO=4";

/*Configuração do Device*/

String state = "";
String token = "";

int ledPin = 5;
```

```
53
54 /*Armazenamento das RemoteIO's em JSON*/
55 StaticJsonDocument<JSON_DOCUMENT_CAPACITY> configurationDocument;
56 JsonArray configurations = configurationDocument.to<JsonArray>();
57 JsonObject setIO = configurations.createNestedObject();
58 JsonObject inputObj = configurations.createNestedObject();
59 /*Sinais para controle*/
60 int dataUpdated = 0;
61 int DataSend = 0;
62
63 /*Variáveis Fixas para rotina de comunição*/
64 bool Connected = false;
65 int Socketed = 0;
66 unsigned long messageTimestamp = 0;
67
68 void tryAuthenticate();
69 void fetchLatestData();
70
71 void UpdatePinOutput(String Ref);
72
73 #if defined(ESP32)
74 void hexdump(const void *mem, uint32_t len, uint8_t cols = 16)
75 {
76   const uint8_t *src = (const uint8_t *)mem;
77
78   USE_SERIAL.printf("\n[HEXDUMP] Address: 0x%08X len: 0x%X (%d)", (ptrdiff_t)src, len,
       len);
79
80   for (uint32_t i = 0; i < len; i++)
81   {
82     if (i % cols == 0)
83     {
84       USE_SERIAL.printf("\n[0x%08X] 0x%08X: ", (ptrdiff_t)src, i);
85     }
86     USE_SERIAL.printf("%02X ", *src);
87     src++;
88   }
89
90   USE_SERIAL.printf("\n");
91 }
92 #endif
93
94 void LedControl(int Status)
95 {
96   digitalWrite(ledPin, Status);
97 }
98
99 /*Função SocketIO.ON*/
100 void socketIOEvent(socketIOmessageType_t type, uint8_t *payload, size_t length)
101 {
102   switch (type)
103   {
104   case sIOtype_DISCONNECT:
105     USE_SERIAL.printf("[IOc] Disconnected!\n");
106     Connected = false;
107     LedControl(0);
108     break;
109   case sIOtype_CONNECT:
110     USE_SERIAL.printf("[IOc] Connected to url: %s\n", payload);
```

```
111       socketIO.send(sIOtype_CONNECT, "/");
112       LedControl(1);
113       break;
114     case sIOtype_EVENT:
115     {
116       char *sptr = NULL;
117       int id = strtol((char *)payload, &sptr, 10);
118
119       USE_SERIAL.printf("[IOc] get event: %s id: %d\n", payload, id);
120
121       if (id)
122       {
123         payload = (uint8_t *)sptr;
124       }
125
126       StaticJsonDocument<JSON_DOCUMENT_CAPACITY> doc;
127       DeserializationError error = deserializeJson(doc, payload, length);
128
129       if (error)
130       {
131         USE_SERIAL.print(F("[IOc]: deserializeJson() failed: "));
132         USE_SERIAL.println(error.c_str());
133         return;
134       }
135
136       String eventName = doc[0];
137
138       USE_SERIAL.printf("[IOc] event name: %s\n", eventName.c_str());
139
140       String ref = doc[1]["ref"];
141       String value = doc[1]["value"];
142
143       setIO[ref]["value"] = value;
144
145       if (setIO[ref]["Mode"] == "OUTPUT")
146       {
147         UpdatePinOutput(ref);
148       }
149     }
150     break;
151     case sIOtype_ACK:
152       USE_SERIAL.printf("[IOc] get ack: %u\n", length);
153       hexdump(payload, length);
154       break;
155     case sIOtype_ERROR:
156       USE_SERIAL.printf("[IOc] get error: %u\n", length);
157       hexdump(payload, length);
158       break;
159     case sIOtype_BINARY_EVENT:
160       USE_SERIAL.printf("[IOc] get binary: %u\n", length);
161       hexdump(payload, length);
162       break;
163     case sIOtype_BINARY_ACK:
164       USE_SERIAL.printf("[IOc] get binary ack: %u\n", length);
165       hexdump(payload, length);
166       break;
167     }
168 }
169
```

```
170 void nodeIotConnection()
171 {
172     /*Conexão WiFi*/
173     Connected = false;
174     LedControl(Connected);
175     WiFi.mode(WIFI_STA);
176     WiFi.begin(ssid, password);
177 
178     WiFi.waitForConnectResult();
179 
180     while (WiFi.status() != WL_CONNECTED)
181     {
182       delay(500);
183       USE_SERIAL.print(".");
184     }
185 
186     USE_SERIAL.printf("[SETUP] WiFi Connected %s\n", WiFi.localIP().toString().c_str());
187 
188     appVerifyUrl.replace(" ", "%20");
189     appLastDataUrl.replace(" ", "%20");
190     appSocketPath.replace(" ", "%20");
191     pinMode(ledPin, OUTPUT);
192 
193     // Função de verificar permissão do dispositivo
194     while (state != "accepted")
195     {
196       USE_SERIAL.println("trying to authenticate");
197       tryAuthenticate();
198 
199       delay(4000);
200     }
201 
202     // Depois de confirmado a permissão, chama função para carregar os ultimos valores
    salvos para/por este dispositivo
203     fetchLatestData();
204 
205     // Conexão Websocket
206     //<nome da companhia> e <nome do controlador> se tiver espaço subtituir por %20
207     socketIO.begin(appHost, appPort, appSocketPath); // TODO: Não há necessidade do %22
    (")
208     socketIO.onEvent(socketIOEvent);
209 }
210 
211 void setup()
212 {
213   USE_SERIAL.begin(115200);
214   USE_SERIAL.setDebugOutput(true);
215   nodeIotConnection();
216 }
217 
218 // Função WebSocket Connect
219 void SocketIOConnect()
220 {
221   uint64_t now = millis();
222   if (Socketed == 0)
223   {
224     StaticJsonDocument<JSON_DOCUMENT_CAPACITY> doc;
225     JsonArray array = doc.to<JsonArray>();
226     array.add("connection");
```

```
227     JsonObject query = array.createNestedObject();
228     query["Query"]["CompanyName"] = companyName;
229     query["Query"]["deviceId"] = deviceId;
230 
231     String output;
232     serializeJson(doc, output);
233     Socketed = socketIO.sendEVENT(output);
234     Socketed = 1;
235   }
236   if ((Socketed == 1) && (now - messageTimestamp > 2000) && (Connected == 0))
237   {
238     StaticJsonDocument<JSON_DOCUMENT_CAPACITY> doc;
239     JsonArray array = doc.to<JsonArray>();
240     messageTimestamp = now;
241     array.add("joinRoom");
242     JsonObject conectionId = array.createNestedObject();
243     conectionId["conectionId"] = companyName + ":" + deviceId;
244     String output;
245     serializeJson(doc, output);
246     Connected = socketIO.sendEVENT(output);
247     USE_SERIAL.println(Connected);
248   }
249 }
250 
251 // Função Atualizar Pinos de saída
252 void UpdatePinOutput(String Ref)
253 {
254   int PinRef = setIO[Ref]["pin"].as<int>();
255   int ValueRef = setIO[Ref]["value"].as<int>();
256 
257   digitalWrite(PinRef, !ValueRef);
258 }
259 
260 void ReadInput()
261 {
262 }
263 
264 // Função principal
265 void loop()
266 {
267   socketIO.loop();
268 
269   if (WiFi.status() != WL_CONNECTED)
270   {
271     WiFi.disconnect();
272     Connected = false;
273     LedControl(Connected);
274 
275     nodeIotConnection();
276   }
277   else if (!Connected)
278   {
279     LedControl(Connected);
280     SocketIOConnect();
281   }
282   else
283   {
284     // código de acordo com a aplicação
285   }
```

```
}

void tryAuthenticate()
{
  WiFiClient client;
  HTTPClient http;

  StaticJsonDocument<JSON_DOCUMENT_CAPACITY> document;
  String request;

  document["companyName"] = companyName;
  document["deviceId"] = deviceId;
  document["mac"] = WiFi.macAddress();

  serializeJson(document, request);

  http.begin(client, appVerifyUrl);
  http.addHeader("Content-Type", "application/json");

  int statusCode = http.POST(request);

  if (statusCode == HTTP_CODE_OK)
  {
    document.clear();
    deserializeJson(document, http.getStream());

    state = document["state"].as<String>();

    if (state == "accepted")
    {
      token = document["token"].as<String>();

      String string;
      serializeJsonPretty(document, string);

      for (size_t i = 0; i < document["gpio"].size(); i++)
      {
        String ref = document["gpio"][i]["ref"];

        int pin = document["gpio"][i]["pin"].as<int>();
        String type = document["gpio"][i]["type"];
        Serial.println(type);
        Serial.println(ref);
        Serial.println(pin);

        if (type == "INPUT")
        {
          setIO[ref]["pin"] = pin;
          setIO[ref]["Mode"] = type;
          inputObj[ref]["pin"]=pin;
          inputObj[ref]["ref"]=ref;
          pinMode(pin, INPUT);
        } else if(type == "OUTPUT"){
          setIO[ref]["pin"] = pin;
          setIO[ref]["Mode"] = type;
          pinMode(pin, OUTPUT);
        } else {
          setIO[ref]["pin"] = pin;
          setIO[ref]["Mode"] = "N/L";
```

```
345           }
346
347       }
348     }
349
350     http.end();
351   }
352 }
353
354 void fetchLatestData()
355 {
356   WiFiClient client;
357   HTTPClient http;
358
359   http.begin(client, appLastDataUrl);
360   http.addHeader("Content-Type", "application/json");
361   http.addHeader("Authorization", "Bearer " + token);
362
363   int statusCode = http.GET();
364
365   if (statusCode == HTTP_CODE_OK)
366   {
367     StaticJsonDocument<JSON_DOCUMENT_CAPACITY> document;
368     deserializeJson(document, http.getStream());
369
370     for (size_t i = 0; i < document.size(); i++)
371     {
372       String auxRef = document[i]["ref"];
373       String auxValue = document[i]["data"]["value"];
374
375       if (auxValue == "null")
376       {
377         auxValue = "0";
378       }
379
380       setIO[auxRef]["value"] = auxValue;
381
382       if (setIO[auxRef]["Mode"] == "OUTPUT")
383       {
384         UpdatePinOutput(auxRef);
385       }
386     }
387   }
388
389   http.end();
390 }
391
392 void espPOST(String Router, String variable, String value)
393 {
394   if ((WiFi.status() == WL_CONNECTED))
395   {
396     WiFiClient client;
397     HTTPClient http;
398     StaticJsonDocument<JSON_DOCUMENT_CAPACITY> document;
399     String request;
400
401     document["deviceId"] = deviceId;
402     document["ref"] = variable;
403     document["value"] = value;
```

```
404     serializeJson(document, request);
405
406     http.begin(client, Router); // HTTP
407     http.addHeader("Content-Type", "application/json");
408     http.addHeader("Authorization", "Bearer " + token);
409
410     int httpCode = http.POST(request);
411
412     if (httpCode == HTTP_CODE_OK)
413     {
414         document.clear();
415     }
416     else if (httpCode == -1)
417     {
418         WiFi.disconnect();
419         nodeIotConnection();
420     }
421     else
422     {
423         // Serial.printf("[HTTP] POST... failed, error: %s\n", http.errorToString(
    httpCode).c_str());
424     }
425     DataSend = 0;
426     http.end();
427   }
428 }
```

Listing B.1: NodeIoT-Esp32-Esp8266.ino

NUP: 23081.162292/2023-11 **Prioridade:** Normal

Homologação de ata de defesa de TCC e estágio de graduação

125.322 - Bancas examinadoras de TCC: indicação e atuação

COMPONENTE

Ordem	Descrição	Nome do arquivo
14	TCC versão final corrigida	TCC_GuilhermeKraemer_MDT_UFSM.pdf

Assinaturas

01/02/2024 10:19:54

GUILHERME KRAEMER (Aluno de Graduação - Aluno Regular)
07.09.09.01.0.0 - Curso de Engenharia de Computação - 121624