UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

PAULO VITOR FERREIRA MARQUES

COMUNICAÇÃO E CONTROLE VIA WEB DE UM SISTEMA EMBARCADO UTILIZANDO O PROTOCOLO *WEBSOCKET* E A INFRAESTRUTURA *SERVERLESS*

UBERLÂNDIA
2023

PAULO VITOR FERREIRA MARQUES

COMUNICAÇÃO E CONTROLE VIA WEB DE UM SISTEMA EMBARCADO UTILIZANDO O PROTOCOLO *WEBSOCKET* E A INFRAESTRUTURA *SERVERLESS*

Trabalho de Conclusão de Curso submetido ao Curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica.

Orientador: Prof. Dr. Fernando Lourenço de Souza
Área do conhecimento: Sistemas embarcados

UBERLÂNDIA
2023

PAULO VITOR FERREIRA MARQUES

COMUNICAÇÃO E CONTROLE VIA WEB DE UM SISTEMA EMBARCADO UTILIZANDO O PROTOCOLO *WEBSOCKET* E A INFRAESTRUTURA *SERVERLESS*

Trabalho de Conclusão de Curso submetido ao Curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica.

Uberlândia, 28 de junho de 2023.

Orientador: _____
Prof. Dr. Fernando Lourenço de Souza
FEMEC/UFU

Examinador: _____
Prof. Me. Thiago Gomes Cardoso
CEFET - MG

Examinador: _____
Prof. Dr. Pedro Augusto Queiroz de Assis
FEMEC/UFU

AGRADECIMENTOS

Agradeço primeiramente a Deus por me permitir concluir esta etapa; aos meus pais que sempre foram pilares sólidos na minha vida, exemplos de perseverança e honestidade; aos amigos que me apoiaram diante das intempéries da vida; ao Professor Doutor Fernando Lourenço pela orientação e por ter concordado em trilhar comigo este último trecho rumo à conclusão da graduação.

RESUMO

Este estudo apresenta a implementação e a avaliação de um sistema de controle remoto para sistemas embarcados, utilizando o protocolo *WebSocket*, a infraestrutura *serverless* e o microcontrolador ESP32. A intenção é proporcionar uma solução robusta e eficiente para monitoramento e coleta de dados de dispositivos IoT (*Internet of Things*), os quais são dispositivos capazes de comunicarem entre si e com a Internet, demonstrando relevância para a comunidade acadêmica e aplicações práticas. São explorados os sistemas embarcados e IoT, a funcionalidade do protocolo *WebSocket* para comunicação bidirecional de baixa latência e o microcontrolador ESP32. Além disso, é discutida a arquitetura *serverless* e ferramentas relacionadas, como AWS API Gateway e AWS Lambda. Para tanto, foi idealizado um cenário de coleta de variáveis ambientais (temperatura e humidade) e o controle de um LED, que elucida de forma simples e eficaz a capacidade bidirecional de comunicação em tempo real, fazendo uso de uma infraestrutura *serverless*, juntamente com a criação de uma interface hospedada na *web* para o controle de forma remota. O sucesso da implementação deste sistema foi verificado através da observação durante seu funcionamento, com o fluxo em tempo real de dados do sensor sendo renderizados e visualizados corretamente na interface projetada e controle simultâneo não conflitante do LED, além da medição da latência média experimentada pelo usuário resultando no valor de 50 ms para o recebimento dos dados, e 100 ms para o controle do LED, com uma taxa de sucesso próxima de 100 % no período observado de duas horas. Portanto, tendo sido comprovada a eficácia da solução proposta, este trabalho contribui como uma abordagem detalhada para a consolidação do conhecimento em sistemas embarcados e IoT e para uma gama de aplicações nas quais se fazem necessárias as características providas pelo sistema, abrindo caminho para futuras investigações de diferentes protocolos e equipamentos com o intuito de otimizar a comunicação bidirecional em tempo real entre dispositivos e usuários, e facilitar o monitoramento e controle de sistemas embarcados.

Palavras-chave: Sistemas embarcados; IoT; *WebSocket*; Serverless; ESP32; Controle remoto; Comunicação bidirecional em tempo real.

ABSTRACT

This study presents the implementation and evaluation of a remote-control system for embedded systems, using the WebSocket protocol, the serverless infrastructure and the ESP32 microcontroller. The intention is to provide a robust and efficient solution for monitoring and collecting data from IoT (Internet of Things) devices, which are devices capable of communicating with each other and with the Internet, demonstrating relevance to the academic community and practical applications. Embedded and IoT systems, the functionality of the WebSocket protocol for bidirectional low-latency communication and the ESP32 microcontroller are explored. Additionally, serverless architecture and related tools such as AWS API Gateway and AWS Lambda are discussed. To this end, a scenario was designed to collect environmental variables (temperature and humidity) and control an LED, which simply and effectively elucidates the bidirectional capacity of real-time communication, making use of a serverless infrastructure, together with the creation of a web-hosted interface for remote control. The success of the implementation of this system was verified through observation during its operation, with the real-time flow of sensor data being correctly rendered and visualized in the designed interface and simultaneous non-conflicting control of the LED, in addition to the measurement of the average latency experienced by the user. resulting in a value of 50 ms for receiving the data, and 100 ms for controlling the LED, with a success rate close to 100% in the observed period of two hours. Therefore, having proven the effectiveness of the proposed solution, this work contributes as a detailed approach to the consolidation of knowledge in embedded systems and IoT and to a range of applications in which the characteristics provided by the system are necessary, opening the way for future investigations. of different protocols and equipment in order to optimize bidirectional communication in real time between devices and users, and to facilitate the monitoring and control of embedded systems.

Keywords: Embedded systems; IoT; WebSocket; Serverless; ESP32; Remote control; Real-time bidirectional communication.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

O futuro está enraizado na Indústria 4.0, indissociável do universo da Internet das Coisas (IoT), uma rede em expansão de dispositivos inteligentes dotados de computação embarcada que comunicam e trocam dados entre si e com pessoas por meio da Internet (OLIVEIRA, NIZU e BASSETO, 2017). Segundo Hermann, Pentek e Otto (2016), a Indústria 4.0 é alicerçada em quatro pilares principais: Sistemas Ciber-Físicos, Internet das Coisas, Internet de Serviços e Fábricas Inteligentes. Neste contexto, os sistemas embarcados, que são computadores especializados em realizar tarefas específicas, desempenham um papel indispensável, cuja familiaridade é encontrada no dia a dia, desde os automóveis aos eletrodomésticos, passando pelos sistemas industriais e infraestruturas, até no âmbito acadêmico para análise de dados e pesquisas.

Diante deste panorama, desponta a necessidade de monitoramento e controle remoto em tempo real dos sistemas embarcados. Esses recursos oferecem inúmeras possibilidades, especialmente no contexto de coleta e monitoramento de dados sensíveis ao tempo. Para atender a essa demanda, este trabalho aborda a implementação de um sistema de controle remoto para sistemas embarcados, utilizando o protocolo *WebSocket*, infraestrutura *serverless* e o microcontrolador ESP32.

A escolha do microcontrolador ESP32 (Espressif32) é justificada devido a características como robustez, versatilidade, baixo custo e conectividade, sendo ideal para um sistema de controle remoto eficaz para sistemas embarcados (KOLBAN, 2018). Além disso, a tecnologia *WebSocket* é um canal de comunicação bidirecional de baixa latência, essencial para o controle remoto eficiente, superando as limitações impostas pela distância física (KOTEVSKI, MIKROVSKI e JOLEVSKI, 2011).

A configuração do sistema é potencializada pela adoção de uma infraestrutura *serverless*, utilizando AWS API Gateway e AWS Lambda. Tal infraestrutura minimiza a necessidade de manutenção de servidores físicos, permitindo um desenvolvimento centrado na funcionalidade da aplicação e proporcionando economia de recursos.

Para o desenvolvimento do *front-end* da aplicação, foi utilizada a biblioteca *React*. Esta escolha permitiu a criação de uma interface de usuário eficiente e interativa, cumprindo os requisitos modernos da indústria de tecnologia.

A aplicabilidade potencial do sistema proposto abrange um amplo espectro de áreas. Embora tenha sido testado em um cenário específico - coleta de variáveis ambientais (temperatura e humidade) e controle de um LED - o sistema foi projetado para ser facilmente adaptável a outras aplicações e cenários, contanto que haja conexão *Wi-Fi* disponível. Sua capacidade de coleta de dados e controle remoto de sistemas embarcados pode ter aplicações em várias áreas, desde monitoramento de sistemas biológicos e automações residenciais, até a análise de desempenho e controle de máquinas e sistemas mecânicos em tempo real, incluindo também o âmbito de pesquisas e projetos acadêmicos, nas quais existe a necessidade de praticidade e versatilidade de observação e controle das aplicações.

Neste espaço dinâmico, caracterizado pela rápida evolução tecnológica, este trabalho contribui com a verificação de uma abordagem para enfrentar os desafios atuais, oferecendo uma solução prática, adaptável e versátil para monitoramento e controle remoto de sistemas embarcados.

2. OBJETIVOS

Para melhor entendimento do trabalho aqui exposto, os objetivos foram subdivididos entre geral e específicos, descritos a seguir.

2.1 Objetivo Geral

Desenvolver e implementar um sistema de controle remoto para sistemas embarcados, utilizando o microcontrolador ESP32, o protocolo *WebSocket* e uma infraestrutura *serverless*, visando facilitar o monitoramento em tempo real e a coleta de dados em diversas aplicações com acesso à rede *Wi-Fi*, incluindo, mas não se limitando a, ambientes de pesquisa e projetos dentro do âmbito acadêmico. A intenção é criar uma solução adaptável e eficiente, de baixo custo, que possa ser ajustada para atender às necessidades específicas de diferentes aplicações, promovendo avanços na coleta de dados e no controle remoto de dispositivos IoT, e explorando tecnologias relevantes na atualidade.

2.2 Objetivos Específicos

- **Explorar a importância e aplicações de sistemas embarcados e IoT:** Discutir o papel crítico que os sistemas embarcados e a IoT desempenham em nossa vida cotidiana e na indústria, com ênfase no monitoramento e controle remoto em tempo real.
- **Estudar o protocolo *WebSocket*:** entender o funcionamento do protocolo *WebSocket*, as respectivas vantagens para a comunicação bidirecional de baixa latência e como é facilitada a comunicação em tempo real em sistemas embarcados.
- **Compreender o microcontrolador ESP32:** discutir as características e vantagens do microcontrolador ESP32, incluindo a robustez, versatilidade, baixo custo e capacidade de comunicação.
- **Discutir a arquitetura *serverless* e ferramentas relacionadas:** explicar a infraestrutura *serverless*, as vantagens em termos de redução de custos e manutenção, e como as ferramentas AWS API Gateway e AWS Lambda se encaixam nesse contexto.

- **Analisar a biblioteca *React*:** discutir a importância do desenvolvimento *front-end* na criação de interfaces de usuário eficientes e interativas, destacando a escolha do uso da biblioteca *React*.
- **Implementar o sistema de controle remoto:** este objetivo é focado no desenvolvimento e implementação de um sistema de controle remoto para sistemas embarcados, utilizando as tecnologias e estratégias discutidas anteriormente. A finalidade principal é proporcionar um sistema eficaz e de baixo custo para coleta de dados em tempo real e controle remoto de dispositivos embarcados cuja aplicabilidade potencial abranja um espectro amplo de áreas.
- **Avaliar o sistema implementado:** este objetivo é dedicado à avaliação do sistema de controle remoto desenvolvido, considerando seu desempenho, eficiência e aplicações práticas.

3 JUSTIFICATIVA

A essência deste projeto reside na crescente demanda por uma abordagem multifacetada e inovadora na construção de sistemas de controle remoto aplicados a sistemas embarcados. As incessantes transformações no cenário tecnológico têm disponibilizado estratégias e instrumentos cada vez mais sofisticados para superar obstáculos presentes na gestão e intercomunicação de tais sistemas. Essa evolução inclui aumento das funcionalidades, simplificação e facilidade de uso, complexidade interna, diminuição de custo, entre outros (ALBERTIN, 2017).

Em suma, a razão de ser deste trabalho se ancora tanto na relevância pragmática - pela aplicação de tecnologias e estratégias inovadoras na resolução de problemas contemporâneos - quanto na relevância acadêmica - pela sua contribuição para a consolidação do conhecimento e a proposição de novos caminhos investigativos.

4 FUNDAMENTAÇÃO TEÓRICA E REVISÃO BIBLIOGRÁFICA

4.1 A Internet das Coisas

A Internet das Coisas (IoT) é um conceito que abrange a integração de vários dispositivos e tecnologias inteligentes em uma rede unificada, permitindo que se comuniquem e interajam entre si sem intervenção humana, como apontado por Al-Fuqaha, Guizani, Mohammadi, Aledhari e Ayyash (2015). Ainda segundo o autor, esta ampla conectividade desempenha um papel primordial na habilitação de sistemas autônomos, possibilitando a criação de serviços contextuais e aplicações sensíveis ao tempo. Nesse sentido, a importância da IoT transcende a funcionalidade imediata, servindo como o motor principal da interação moderna com o mundo, conectando uma diversidade crescente de dispositivos à internet de maneira cada vez mais inteligente.

Embora o termo "Internet das Coisas" tenha sido cunhado por Kevin Ashton em 1999, durante sua apresentação na Procter & Gamble (P&G), a concepção da IoT remonta a décadas anteriores. Ashton vislumbrava uma realidade em que os objetos físicos, imbuídos com sensores e conectividade, pudessem comunicar dados sem a necessidade de interação humana (ASHTON et al., 2009). Embora a ideia inicial tenha sido revolucionária, foi o advento e a rápida evolução das tecnologias de comunicação sem fio e dos dispositivos inteligentes que realmente deram asas à visão de Ashton.

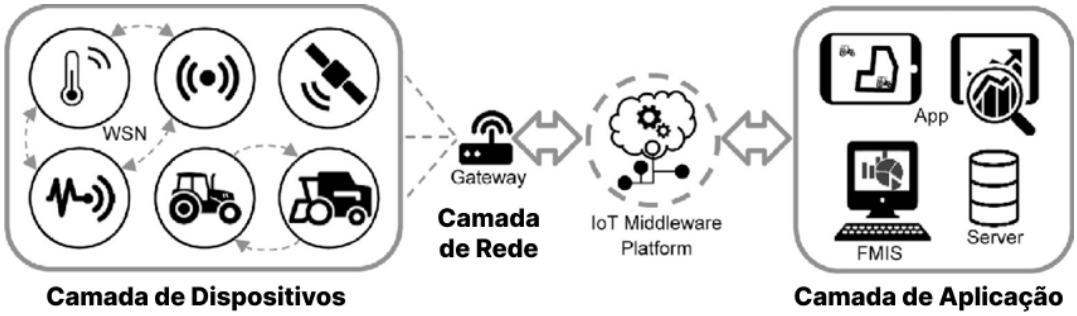
> “A Internet das Coisas (IoT), também chamada de Internet de Tudo ou Internet Industrial, é um novo paradigma tecnológico concebido como uma rede global de máquinas e dispositivos capazes de interagir uns com os outros. A IoT é reconhecida como uma das mais áreas importantes da tecnologia futura e está ganhando grande atenção de uma ampla gama de indústrias. O verdadeiro valor da IoT para as empresas pode ser totalmente percebido quando os dispositivos conectados são capazes de se comunicar uns com os outros e se integrar com sistemas de inventário gerenciados pelo fornecedor, sistemas de suporte ao cliente, aplicativos de inteligência de negócios e análise de negócios.”
>
> (LEE, I; LEE, K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, v. 58, n. 4, p. 431, 2015. Tradução nossa.)

Assim como o autor deixou claro o potencial da Internet das Coisas em 2015, ao longo dos anos a tecnologia em torno desse tema despontou de maneira vertiginosa e, para compreender a evolução da IoT, deve-se considerar

a evolução dos componentes fundamentais: dispositivos, redes e aplicativos (Figura 1). Os dispositivos IoT, variando de simples sensores a complexos robôs, têm evoluído em termos de capacidade computacional, eficiência energética e miniaturização (MIORANDI, SICARI, DE PELLEGRINI & CHLAMTAC, 2012).

Além disso, o controle em tempo real desses dispositivos tornou-se um foco importante, permitindo respostas mais rápidas e eficientes a eventos dinâmicos (LIN, YU, ZHANG, YANG, ZHANG & ZHAO, 2017). A conectividade de rede se expandiu do 2G para o 5G, com o último permitindo maior velocidade de transferência de dados, menor latência e conectividade massiva de dispositivos IoT. Finalmente, os aplicativos IoT têm experimentado uma explosão em termos de quantidade, qualidade e diversidade, abrangendo áreas como casas inteligentes, cidades inteligentes, saúde, agricultura e a indústria.

Figura 1 - Componentes fundamentais.



Fonte: Adaptado de Villa-Henriksen *et. Al.* (2020).

As expectativas são de que a Internet das Coisas continuará a evoluir a uma velocidade surpreendente. Conforme observado por Gubbi *et al.* (2013), as próximas etapas da evolução da IoT provavelmente envolverão a integração com a inteligência artificial (IA), permitindo sistemas IoT mais inteligentes e autônomos. Além disso, a segurança e a privacidade são áreas que requerem atenção contínua, à medida que a crescente conectividade traz consigo desafios significativos nessas áreas.

Em suma, a Internet das Coisas representa uma revolução na forma como interage-se com o mundo, tornando o dia a dia mais conectado e personalizado em termos das informações coletadas e apresentadas, como pode ser visto, por exemplo, através da tecnologia *wearable*, que se trata de acessórios e vestimentas dotados de tecnologia capazes de integração com a IoT. É um

campo de estudo em constante evolução que certamente continuará a moldar o futuro da inovação e transformação digital. É nesse contexto que o presente projeto busca explorar a utilização de tecnologias emergentes em soluções IoT. A escolha dessas tecnologias visa aproveitar os benefícios que podem oferecer ao universo IoT, como redução da latência, melhor eficiência na comunicação e maior economia de recursos. Dessa forma, o projeto contribui por buscar fornecer soluções que acompanhem esse ritmo de mudança e respondam efetivamente às demandas atuais de interconexão e interação de dispositivos.

4.2 Sistemas Embarcados: conceitos e aplicações na Internet das Coisas (IoT)

Os sistemas embarcados são computadores especializados projetados para realizar tarefas dedicadas com requisitos específicos. Ao contrário dos computadores de uso geral, como os desktops, que são projetados para executar uma ampla gama de aplicações, os sistemas embarcados são otimizados para eficiência e desempenho em suas tarefas dedicadas (MARWEDEL, 2011). Estes sistemas servem como a base de muitos dispositivos digitais e eletrônicos que usamos no nosso dia a dia.

A importância dos sistemas embarcados está no fato de permitirem a automação e a digitalização de inúmeros dispositivos e aplicações. Efetivamente, esses sistemas são encontrados em uma grande variedade de aplicações, desde automóveis, aviões e telefones celulares, até dispositivos médicos e eletrodomésticos (WOLF, 2022) (Figura 2). Graças a eficiência e especificidade dos sistemas embarcados, muitas das tecnologias modernas se tornaram possíveis, simplificando a interação com a tecnologia e proporcionando inovações contínuas em diversas áreas.

No contexto da IoT, os sistemas embarcados desempenham um papel ainda mais crucial. A IoT refere-se à interconexão digital de objetos cotidianos com a internet, permitindo que coletem e troquem dados (LEE & LEE, 2015). Isso não seria possível sem os sistemas embarcados, que atuam como o cérebro desses objetos conectados, permitindo que informações sejam processadas, armazenadas e transmitidas.

Figura 2 - Exemplos de objetos que possuem sistema embarcado.

Fonte: Sam Solutions (2023).

Os sistemas embarcados, quando integrados à IoT, podem desempenhar uma ampla gama de funções, desde monitorar a saúde de um paciente através de um dispositivo *wearable*, até otimizar a eficiência energética em uma casa inteligente (GRIMM, NEUMANN & MAHLKNECHT, 2012). Ao fornecer a capacidade de processar e transmitir informações, os sistemas embarcados facilitam a transformação de objetos físicos em inteligentes, habilitando-os a interagir e comunicar-se entre si.

Em conclusão, a importância dos sistemas embarcados na nossa sociedade digital é incontestável, não só permitindo a funcionalidade de inúmeros dispositivos e aplicações, mas também desempenhando um papel chave na facilitação da IoT. Com a crescente digitalização da sociedade, a importância dos sistemas embarcados só aumentará no futuro.

4.3 Microcontrolador ESP32: Aplicabilidade e vantagens na IoT e sistemas embarcados

O ESP32 é um microcontrolador altamente integrado desenvolvido pela Espressif Systems (Figura 3). Este chip de alto desempenho e baixo custo, equipado com capacidades de Wi-Fi e Bluetooth de baixo consumo energético,

é ideal para uma miríade de aplicações. Neste contexto, destaca-se o seu papel nas aplicações relacionadas à Internet das Coisas (IoT) e sistemas embarcados (ESPRESSIF SYSTEMS, 2023).

Figura 3 - ESP32

Fonte: Auto Core Robótica (2023).

É preciso enfatizar que as vantagens do ESP32 para aplicações de IoT são multifacetadas, dentre as quais o seu baixo consumo de energia e a capacidade de suportar diversas interfaces de comunicação tornam-se notáveis. Esses aspectos são cruciais para dispositivos IoT, que muitas vezes são alimentados por bateria e demandam eficiência energética (BAHGA; MADISETTI, 2014). Nesse sentido, uma característica que vale a pena ser discutida em detalhe é a capacidade de comunicação em tempo real do ESP32.

Pode-se dizer que os meios de comunicação em tempo real proporcionados pelo ESP32 são uma força motriz para sua crescente adoção. Por meio de sua capacidade de operar com protocolos como HTTP, MQTT e *WebSocket*, todos baseados no protoclo TCP, o ESP32 fornece uma comunicação em tempo real confiável, essencial para muitas aplicações de IoT. O mais interessante, contudo, é perceber a relação sinérgica entre o protocolo *WebSocket* e o ESP32.

Nesta linha de raciocínio, o uso do protocolo *WebSocket* com o ESP32 abre uma nova gama de possibilidades, permitindo uma comunicação bidirecional e em tempo real entre o microcontrolador e um servidor ou cliente. Este protocolo é especialmente útil em aplicações nas quais a rápida troca de mensagens é necessária, como em sistemas de controle e automação

(BAYILMIŞ, 2022). Assim, torna-se evidente o potencial do ESP32 em sistemas embarcados.

Não é exagero afirmar que o ESP32 tem encontrado uso extensivo em sistemas embarcados, devido à sua flexibilidade e poder de processamento. Desde sensores ambientais até dispositivos *wearable* e automação residencial, o ESP32 oferece uma solução compacta e eficiente para os diversos desafios associados à IoT (MAIER; SHARP; VAGAPOV, 2017).

Pode-se então constatar que o ESP32 é um microcontrolador poderoso e versátil para o desenvolvimento de aplicações IoT e sistemas embarcados. A eficiência energética, características técnicas e robustas capacidades de comunicação deste dispositivo tornam-no uma escolha ideal para uma ampla gama de aplicações.

4.4 Protocolo *WebSocket*: comunicação em tempo real na Internet das Coisas

A invenção do protocolo *WebSocket* marca uma era inovadora na comunicação web. Lançado em 2011 pelo Internet Engineering Task Force (IETF), foi concebido como uma resposta à demanda por uma comunicação bidirecional em tempo real mais eficaz, devido aos desafios associados aos métodos então em uso, o *polling* e o *long polling* (FETTE; MELNIKOV, 2011).

4.4.1 O Início do *WebSocket*: Abandonando o *Polling*

Antes do advento do *WebSocket*, as estratégias para comunicação em tempo real na web eram embasadas na ideia de *polling*. Essa abordagem envolvia fazer requisições HTTP frequentes ao servidor para verificar a existência de novos dados. Entretanto, tais métodos apresentavam limitações incontornáveis:

- *Polling*: aqui, requisições HTTP frequentes eram enviadas ao servidor. Embora útil, resultava em alto consumo de recursos e não garantia uma comunicação verdadeiramente em tempo real.
- *Long Polling*: este processo mantinha a resposta HTTP do servidor aberta até que novos dados estivessem disponíveis. Embora fosse um avanço em direção à comunicação em tempo real, ainda era uma solução provisória e distante do ideal.

4.4.2 Detalhes Técnicos do *WebSocket*

O protocolo *WebSocket* introduz uma metodologia fundamentalmente distinta para a comunicação em tempo real na web, sendo definido por duas características primordiais:

- Canal bidirecional: O *WebSocket* instaura um canal bidirecional completo por meio de uma única conexão TCP. Isso contrasta fortemente com os métodos prévios baseados em HTTP, que exigiam múltiplas conexões para a comunicação bidirecional.
- Comunicação constante: Após a abertura da conexão inicial, por meio de um "aperto de mão" HTTP (Figura 4), a conexão *WebSocket* se mantém aberta, permitindo o fluxo contínuo de dados entre o cliente e o servidor, eliminando a necessidade de requisições HTTP adicionais (WANG; SALIM; MOSKOVITS, 2013).

Figura 4 - Procedimento de conexão *WebSocket*

Fonte: Adaptado de PubNub (2023).

4.4.3 *WebSocket* - Uma Revolução na Comunicação em Tempo Real

A chegada do protocolo *WebSocket* não foi menos do que uma revolução na comunicação em tempo real na web. Este salto monumental no paradigma da comunicação online é resultado de vários fatores cruciais:

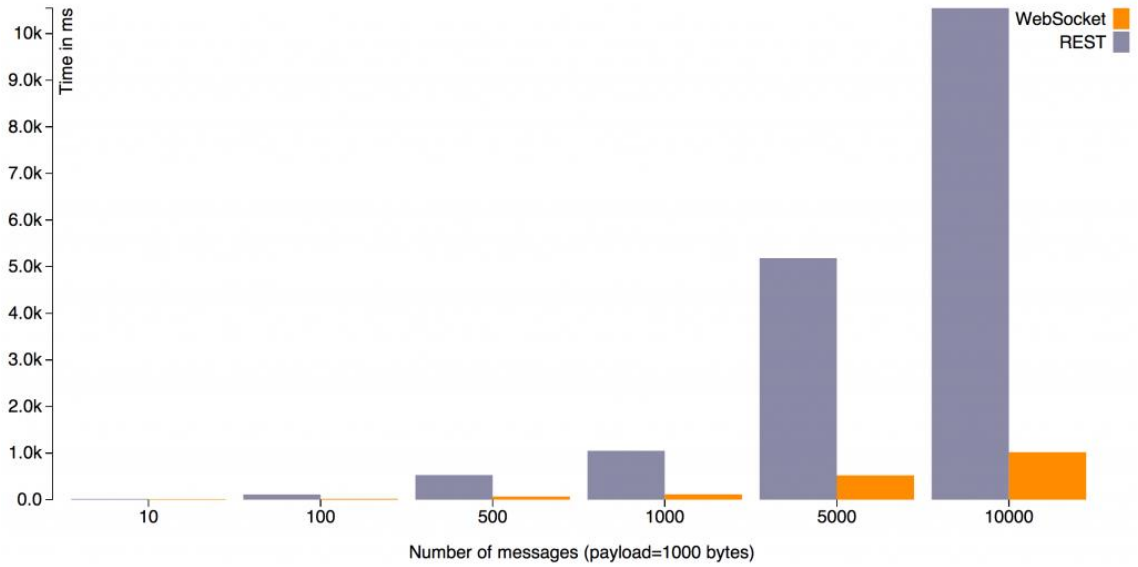
- Eficiência: o *WebSocket* evita o desperdício de recursos associado à abertura e fechamento constante de conexões, que era característico dos métodos baseados em HTTP, como o padrão

REST (*Representational State Transfer*) (Figura 5). Essa eficiência é de grande importância para a otimização do desempenho, especialmente em aplicações de grande escala e com alto tráfego de dados (WANG; SALIM; MOSKOVITS, 2013).

- Flexibilidade: a natureza do *WebSocket* permite que seja usado não apenas para comunicação de texto, mas também para transferência de dados binários. Esta característica torna o protocolo adequado para uma ampla gama de aplicações, desde a transmissão de dados simples até a streaming de áudio e vídeo em tempo real (WANG; SALIM; MOSKOVITS, 2013).
- Integração: o *WebSocket* foi projetado para operar por meio da porta 80, a mesma usada pelo protocolo HTTP. Isso permite que seja integrado facilmente com a infraestrutura web existente, incluindo proxies, roteadores e firewalls, sem a necessidade de grandes alterações (WANG; SALIM; MOSKOVITS, 2013).

Assim, graças a essas inovações e melhorias, o protocolo *WebSocket* tem reformulado o cenário da comunicação em tempo real na web, proporcionando a base para uma nova era de interatividade *online*.

Figura 5 - Comparação do tempo total de resposta entre o protocolo *WebSocket* e o padrão REST.



Fonte: Browsee (2023).

4.4.4 *WebSocket* e IoT

No universo de IoT, o *WebSocket* desempenha um papel de destaque, sendo uma solução de comunicação particularmente adequada para este ambiente. As razões para isso são:

- Comunicação em Tempo Real: devido à sua natureza bidirecional e à capacidade de manter conexões abertas, é perfeitamente adequado para dispositivos de IoT que precisam se comunicar de forma contínua e instantânea. Esta característica é especialmente relevante em aplicações de IoT, onde a transmissão em tempo real de dados pode ser crítica.
- Eficiência em termos de largura de banda: o *WebSocket* se destaca quando comparado aos protocolos HTTP tradicionais, uma vez que evita a sobrecarga associada à abertura e fechamento constantes de conexões. Essa eficiência é uma consideração importante em cenários de IoT, onde a largura de banda pode ser um recurso limitado.

Desta forma, o protocolo *WebSocket* mostra-se como uma solução promissora para superar os desafios da comunicação em tempo real no campo da IoT. Tendo o potencial de revolucionar a forma como os dispositivos de IoT se comunicam, o posicionamento do protocolo é reforçado como um protagonista na comunicação em tempo real na web.

4.5 Arquitetura *Serverless*

O conceito de arquitetura *serverless*, também conhecida como computação sem servidor, descreve um modelo de design de aplicativo e um paradigma de implantação no qual a infraestrutura que executa o código do aplicativo é totalmente gerenciada pelo provedor de nuvem, liberando os desenvolvedores das tarefas de gerenciamento de servidores (BALALAIE; HEYDARNOORI; JAMSHIDI, 2016).

Historicamente, a arquitetura *serverless* evoluiu como um passo natural além da computação em nuvem, advinda das limitações percebidas, onde ainda havia necessidade de gerenciar servidores, atualizar sistemas operacionais e

software de suporte, e ajustar o desempenho (ROBERTS; CHAPIN, 2017). Com escalabilidade e cobrança baseada no consumo real, essa arquitetura foi uma resposta a esses desafios.

As vantagens do paradigma *serverless* são particularmente notáveis em sistemas de IoT. A capacidade de processar dados em grande escala e de lidar com cargas de trabalho variáveis torna a arquitetura *serverless* uma escolha atraente. Além disso, com o *serverless*, as empresas de IoT podem manter seus custos operacionais baixos, reduzindo o excesso de capacidade e pagando apenas pelo que usam (LLOYD et al., 2018).

4.5.1 Ferramentas para Implementação de Arquitetura *Serverless*: Uma introdução ao AWS API Gateway e AWS Lambda

As ferramentas AWS API Gateway e AWS Lambda são cruciais para a implementação de uma arquitetura *serverless*. O AWS API Gateway é uma plataforma de serviço totalmente gerenciada que torna mais fácil para os desenvolvedores criar, publicar, manter, monitorar e proteger APIs em qualquer escala (AWS, 2023). Por outro lado, o AWS Lambda é um serviço que permite que você execute seu código sem provisionar ou gerenciar servidores, proporcionando uma resposta rápida a eventos e gerenciando os recursos de computação automaticamente (AWS, 2023).

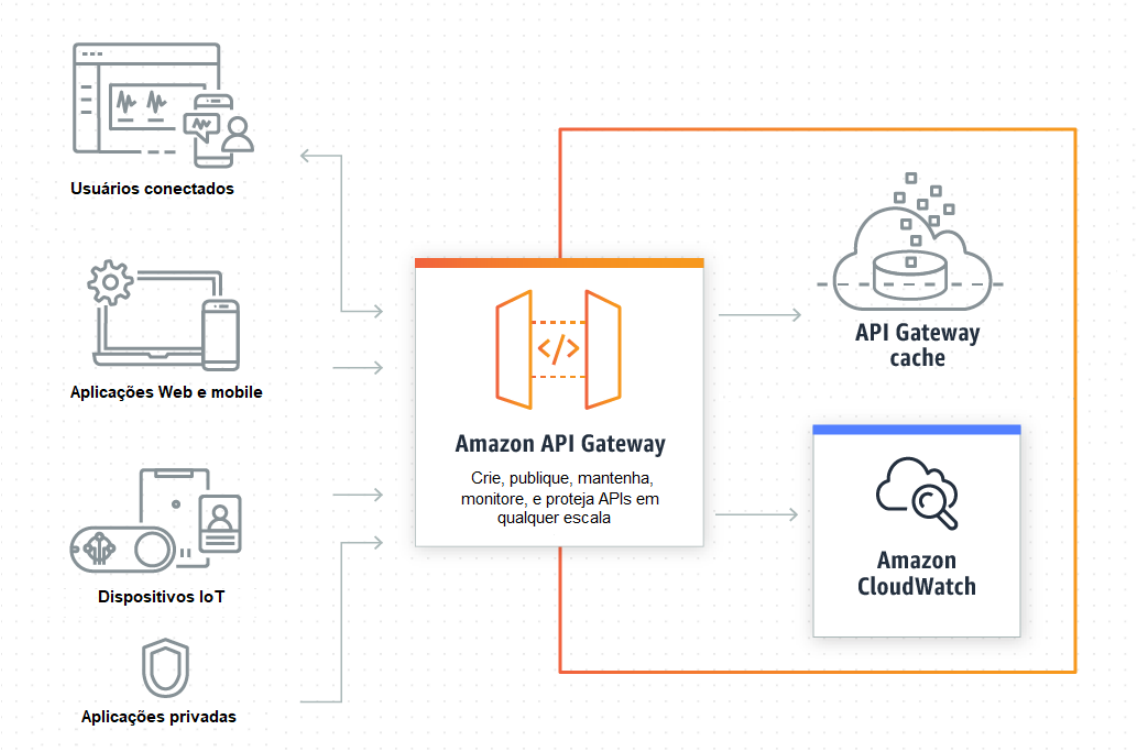
4.5.1.1 AWS API Gateway

A AWS API Gateway suporta uma variedade de APIs, como HTTP, RESTful e *WebSocket*. Essa versatilidade torna a ferramenta adaptável a várias situações de uso, desde serviços da web simples até integrações de *backend* complexas. Em particular, a capacidade do API Gateway de suportar APIs *WebSocket* é de grande importância para a IoT, já que este protocolo permite uma comunicação bidirecional em tempo real, crucial para muitas aplicações de IoT (AWS, 2023).

No API Gateway, as rotas podem ser definidas para canalizar as mensagens recebidas para diferentes funções do AWS Lambda com base no conteúdo da mensagem, tornando a ferramenta altamente flexível e capaz de gerenciar diferentes tipos de interações em tempo real (Figura 6). Além disso, o API Gateway tem integração nativa com outros serviços AWS, como o AWS

CloudWatch para monitoramento e o AWS Cognito para autenticação (AWS, 2023).

Figura 6 - AWS API Gateway



Fonte: AWS API Gateway (2023).

4.5.1.2 AWS Lambda

O AWS Lambda, por sua vez, é um ambiente de computação sem servidor que executa seu código em resposta a eventos. O serviço lida com todo o gerenciamento de recursos, incluindo a manutenção do sistema operacional e do ambiente de tempo de execução, o provisionamento de capacidade, o patch automático de software e o monitoramento e registro de código (AWS, 2023).

Além disso, o AWS Lambda tem uma profunda integração com o ecossistema AWS, permitindo que ele responda a eventos de mais de 200 serviços AWS. Na prática, isso significa que sempre que um evento ocorre - seja ele uma mensagem *WebSocket* recebida, uma alteração em um banco de dados ou um arquivo carregado em um bucket do S3 (Figura 7) - uma função Lambda pode ser acionada para processar esse evento (AWS, 2023).

Figura 7 - AWS Lambda

Fonte: AWS Lambda (2023).

4.5.2 AWS API Gateway, AWS Lambda e Comunicação *WebSocket* em Tempo Real para IoT

No contexto da arquitetura *serverless*, a AWS oferece ferramentas poderosas que facilitam a implementação de comunicação *WebSocket*. O AWS API Gateway e a AWS Lambda formam uma combinação eficiente para este propósito.

Ao combinar essas duas ferramentas, pode-se criar um sistema de comunicação em tempo real eficaz e escalável. O API Gateway pode gerenciar conexões *WebSocket*, enquanto a AWS Lambda pode processar as solicitações de entrada e saída de dados.

O processo de implementação de comunicação *WebSocket* é abordado do seguinte modo:

- **Solicitação de conexão *WebSocket***: o processo de comunicação em tempo real começa quando um cliente, seja um dispositivo de IoT ou um aplicativo web, envia uma solicitação de conexão *WebSocket* para o AWS API Gateway. Esta solicitação é feita usando o protocolo *WebSocket*, e é tipicamente enviada para um *endpoint* que foi definido no API Gateway (AWS, 2023).
- **Invocação da função AWS Lambda**: após receber a solicitação de conexão *WebSocket*, o AWS API Gateway invoca a função AWS Lambda correspondente. Esta função é responsável por processar a solicitação de conexão e retornar uma resposta. A lógica de processamento específica pode variar dependendo do propósito da comunicação em tempo real, mas em geral, pode

envolver a autenticação do cliente, a validação da solicitação e a preparação da resposta (AWS, 2023).

- **Estabelecimento da conexão *WebSocket***: uma vez que a função Lambda processou a solicitação e retornou uma resposta, o API Gateway então envia essa resposta de volta ao cliente. Se a resposta for bem-sucedida, uma conexão *WebSocket* é estabelecida entre o cliente e o servidor. Esta conexão é persistente, o que significa que permanece aberta até que seja fechada por um dos participantes, permitindo uma comunicação bidirecional em tempo real (AWS, 2023).
- **Comunicação em Tempo Real**: com a conexão *WebSocket* estabelecida, o cliente e o servidor agora podem trocar mensagens em tempo real. Cada mensagem enviada pelo cliente é direcionada ao API Gateway, que então invoca outra função Lambda para processar a mensagem. Após a mensagem ser processada, a função Lambda pode, por exemplo, enviar uma mensagem de resposta de volta ao cliente através do API Gateway, completando assim o ciclo de comunicação (AWS, 2023).

A arquitetura *serverless* é uma solução eficiente para o desenvolvimento de sistemas IoT. Ao oferecer alta escalabilidade, eficiência operacional e menor necessidade de gerenciamento de infraestrutura, o paradigma *serverless* permite maior foco na criação e otimização de aplicações robustas e escaláveis.

Adotando ferramentas como a AWS API Gateway e a AWS Lambda, é possível criar e gerenciar APIs eficientes e seguras. Essas ferramentas, juntamente com a capacidade de criar conexões *WebSocket*, abrem caminho para a comunicação em tempo real em sistemas IoT. Isso permite que os dados sejam trocados entre dispositivos e servidores de maneira rápida e eficiente, melhorando significativamente a capacidade de resposta e a experiência do usuário final.

O aprofundamento na técnica e na prática dessas ferramentas é essencial para explorar todo o seu potencial. Como abordado, o uso da AWS API Gateway e da AWS Lambda pode ser bastante complexo, envolvendo várias etapas e considerações. No entanto, com a devida compreensão e experiência, essas

ferramentas podem ser incrivelmente poderosas, permitindo a criação de sistemas IoT sofisticados.

Em suma, a arquitetura *serverless* e as ferramentas associadas, como a AWS API Gateway e a AWS Lambda, oferecem um caminho promissor para o futuro do desenvolvimento de sistemas IoT. Através do estudo e aplicação prática desses conceitos e ferramentas, este projeto apresentará uma solução simulada, funcional, para o controle e monitoramento de um sistema embarcado simples, onde serão expostos os passos para a configuração e disponibilização na *Web*.

4.6 Desenvolvimento *front-end* e a biblioteca *React*

O desenvolvimento *front-end* refere-se à construção da frente de um site ou aplicação, a parte que é diretamente visível e interativa para o usuário final. Não se limita apenas à estética de um website, mas envolve também a funcionalidade e a experiência do usuário. Quando um indivíduo navega por uma página na web, cada elemento com o qual interage - desde a barra de navegação, botões, imagens, até formulários e animações - é o resultado do trabalho meticuloso do desenvolvimento *front-end*. Esta camada de interface do usuário é construída usando uma combinação de HTML para a estrutura, CSS para o estilo, e JavaScript para adicionar interatividade e funcionalidades adicionais (MDN WEB DOCS, 2023).

A importância do desenvolvimento *front-end* na era digital atual é inegável. Com o crescente número de serviços digitais e negócios online, uma presença digital robusta e eficaz é crucial para o sucesso. Uma interface de usuário bem projetada e funcional não só atrai e retém os usuários, como também tem o potencial de aumentar a conversão e o engajamento dos usuários. Além disso, no contexto da IoT, a interface do usuário pode servir como um ponto crucial de interação entre os usuários e os dispositivos conectados, o que ressalta a importância de um design de interface de usuário eficiente e eficaz.

4.6.1 Evolução do ecossistema *front-end*

O ecossistema *front-end* tem passado por uma evolução significativa ao longo das últimas duas décadas. Inicialmente, as páginas web eram estáticas,

compostas apenas por HTML e CSS. O JavaScript entrou em cena no final dos anos 90, trazendo a possibilidade de interatividade para a web (FLANAGAN, 2006). No entanto, a complexidade das páginas web começou a crescer e os desenvolvedores precisavam de maneiras mais eficientes de criar interfaces de usuário interativas.

A partir dessa necessidade surgiram várias bibliotecas e frameworks JavaScript, como jQuery, que proporcionaram maneiras mais convenientes e menos verbosas para manipular o DOM (*Document Object Model*), que se trata de uma representação de dados dos objetos que compõe a estrutura e o conteúdo de uma página *web*, e lidar com eventos do navegador. Apesar de ter facilitado muitas tarefas, jQuery ainda deixou algumas lacunas. A manutenção do estado da aplicação tornou-se um desafio à medida que as aplicações web evoluíram para se tornarem mais dinâmicas e complexas (MARAS, 2016).

A resposta a esses desafios veio na forma de frameworks modernos de JavaScript como AngularJS, Ember e, posteriormente, *React*, *Vue* e *Angular*. Estas ferramentas não só ofereceram uma maneira de estruturar o código de maneira mais eficiente, mas também introduziram conceitos avançados como vinculação de dados bidirecional, componentes reutilizáveis e atualização eficiente do DOM. Tais inovações deram origem à era moderna do desenvolvimento *front-end*, permitindo a criação de aplicações *web* de página única, SPA (*Single Page Application*), ricas em recursos e altamente interativas.

4.6.2 Introdução à biblioteca *React*

A biblioteca *React* é construída em JavaScript e utilizada para a criação de interfaces de usuário, mantida principalmente pela Meta (antigo Facebook) e uma comunidade de desenvolvedores individuais e empresas. A biblioteca foi lançada pela primeira vez em 2013, durante a conferência JavaScript do Facebook, conhecida como JSConf (JSCONF, 2013). Desde então, tem ganhado popularidade devido à sua eficiência, flexibilidade e abordagem centrada no componente.

Essa biblioteca foi introduzida para resolver um problema fundamental enfrentado pelos desenvolvedores *front-end*: como gerenciar e minimizar a complexidade à medida que as aplicações web se tornam mais dinâmicas e interativas. Para isso, a *React* introduziu o conceito de "componentes". Os

componentes são elementos independentes e reutilizáveis que contêm sua própria lógica e podem ser compostos para criar interfaces de usuário complexas (JSCONF, 2013).

Um diferencial significativo da *React* é o uso de uma tecnologia conhecida como Virtual DOM. Em vez de atualizar diretamente o DOM do navegador, o que pode ser um processo lento e ineficiente, a *React* cria uma representação virtual do DOM. Quando ocorrem alterações no estado do aplicativo, a *React* atualiza essa representação virtual de maneira eficiente, e depois sincroniza as alterações com o DOM do navegador, otimizando o desempenho (BANK & PORCELLO, 2017).

Desde o seu lançamento, a *React* tem sido adotado por uma variedade de empresas de alto perfil, como Airbnb, Netflix e Instagram. A sua crescente popularidade é um testemunho do seu desempenho, capacidades e adaptabilidade em uma variedade de projetos, desde pequenas aplicações pessoais até aplicações empresariais de grande escala.

4.6.3 Uso e vantagens da *React* para o desenvolvimento de aplicações IoT

A Internet das Coisas (IoT) é um paradigma que exige soluções de software robustas e altamente responsivas. A biblioteca *React*, com seu modelo de programação baseado em componentes e gerenciamento eficiente de atualizações da interface de usuário, oferece várias vantagens para o desenvolvimento de aplicações IoT.

À medida que a IoT evolui, as interações entre usuários e dispositivos se tornam cada vez mais complexas. A biblioteca *React* permite a criação de componentes reutilizáveis que podem ser compostos para criar interfaces de usuário complexas. Isso permite aos desenvolvedores projetar experiências de usuário que se adaptam a uma variedade de cenários da IoT (FISCHER, 2017).

Além disso, a abordagem da *React* à atualização da interface de usuário - através do uso da Virtual DOM - é particularmente relevante na IoT. As aplicações IoT frequentemente envolvem a transmissão e o processamento de grandes volumes de dados em tempo real. A biblioteca *React* consegue lidar com essas atualizações frequentes de maneira eficiente, garantindo que a interface do usuário permaneça responsiva e atualizada (SUBRAMANIAN, 2017).

Outro benefício significativo da *React* é a sua ampla comunidade de desenvolvedores e o vasto ecossistema de bibliotecas e ferramentas disponíveis (FISCHER, 2019). Isso inclui bibliotecas especializadas para a IoT, que podem acelerar o desenvolvimento de aplicações, facilitando a integração com uma variedade de dispositivos e protocolos IoT.

Sendo assim, o projeto a ser desenvolvido fará uso de tais bibliotecas especializadas de forma a facilitar o desenvolvimento da aplicação *front-end*, usando a formatação JSON *(JavaScript Object Notation* - Notação de Objetos JavaScript) a qual é uma forma simples e leve de intercâmbio de informações. Este formato, legível para humanos e fácil de processar por máquinas, tem suas raízes na linguagem de programação JavaScript, especificamente no padrão ECMA-262 3ª Edição de dezembro de 1999. Sendo baseado em texto e independente de linguagem, o JSON é compatível com diversas linguagens como C, C++, C#, Java, JavaScript, Perl, Python, entre outras. Devido a essas características, é considerado um formato ideal para troca de dados (JSON ORG [s.d.]).

5. IMPLEMENTAÇÃO DO PROJETO

O presente capítulo destina-se a ilustrar a aplicação prática da tecnologia *WebSocket*, juntamente com o uso de arquitetura *serverless* e a biblioteca *React* para o desenvolvimento *front-end*. Como um dos principais objetivos deste trabalho, o projeto proposto visa implementar um sistema IoT com comunicação bidirecional em tempo real. Na sequência apresenta-se um relato detalhado de como o projeto foi executado.

5.1 Descrição

O projeto deste trabalho se concentra no desenvolvimento de uma solução baseada em IoT utilizando um ESP32 como dispositivo embarcado. Este sistema permite o monitoramento em tempo real da temperatura e umidade, além do controle de um LED como forma de exemplificar a capacidade bidirecional de receber e enviar informações. Além disso, a solução foi feita de forma que seja possível ser monitorada por vários usuários. A escolha do escopo dessa solução se deve pela facilidade de demonstração prática, sem perdas no que tange a proposta deste trabalho, do uso das tecnologias apresentadas ao longo da fundamentação teórica.

O sistema desenvolvido consiste em três componentes principais: o módulo do microcontrolador ESP32, a infraestrutura *serverless* e a interface de usuário. O microcontrolador ESP32, acoplado a um sensor DHT11 de temperatura e umidade, adquirido separadamente, coleta dados ambientais e os transmite em tempo real para os usuários conectados, ao passo que "escuta" comandos para o controle do LED. A infraestrutura *serverless* é implementada na AWS, utilizando o Gateway API para receber dados tanto do ESP32 quanto da interface do usuário e o serviço Lambda para processar, armazenar esses dados e gerir as conexões de múltiplos dispositivos e usuários. O *front-end* é desenvolvido usando a biblioteca *React*, permitindo aos usuários visualizar os dados em tempo real em uma interface de usuário amigável e o controle do LED, contando com o uso de bibliotecas apropriadas.

O objetivo principal deste projeto é demonstrar a aplicabilidade das tecnologias estudadas neste trabalho - em especial o protocolo *WebSocket* e a arquitetura *serverless* - no desenvolvimento de aplicações IoT escaláveis e de

fácil manutenção. Ao mesmo tempo, espera-se que o sistema desenvolvido seja funcional e possa ser usado para fins práticos de validação da proposta.

A implementação bem-sucedida do projeto será confirmada através de testes de funcionamento do sistema. Isso incluirá a verificação do fluxo de dados em tempo real entre o ESP32 e o servidor, bem como a visualização correta desses dados no *front-end*.

5.2 Ferramentas e materiais necessários

O desenvolvimento deste projeto exigiu uma variedade de ferramentas e tecnologias para garantir que todos os componentes funcionassem de maneira adequada. Cada uma dessas ferramentas foi escolhida com base em sua capacidade de atender aos requisitos específicos do projeto e contribuir para o objetivo geral de criar um sistema de controle de sistemas embarcados robusto e funcional.

Inicialmente, o material físico do projeto inclui o microcontrolador ESP32, modelo ESP-WROOM-32, e o sensor de temperatura e umidade DHT11. Além disso, uma protoboard foi usada para conectar esses componentes, juntamente com LEDs, resistores e jumpers para garantir o correto funcionamento do sistema proposto.

No desenvolvimento do código para o microcontrolador ESP32, a linguagem C++ foi utilizada, com a IDE Visual Studio Code (VSCode), empregando a extensão PlatformIO para facilitar o processo de desenvolvimento. Diversas bibliotecas foram usadas, incluindo *DHT.h* para leitura do sensor DHT11, *Arduino.h* para interação com o hardware do microcontrolador, *ArduinoJson.h* para formatação de dados JSON, *WifiMulti.h* e *WebsocketsClient.h* para lidar com conexões *Wi-Fi* e *WebSocket*, respectivamente. A biblioteca *timer.h* foi utilizada para controlar a frequência de leitura do sensor DHT11, definida para um intervalo de um segundo.

Para a implementação da infraestrutura *serverless*, foi utilizada a IDE WebStorm, juntamente com a linguagem Typescript e o *runtime* Node.js v18. A ferramenta Serverless foi usada para definir e implementar a infraestrutura na AWS através de arquivos YAML, incluindo políticas IAM de acesso e privacidade e uma tabela no banco de dados não-relacional *DynamoDB*. Para interação

direta com as ferramentas correspondentes no ecossistema AWS, foi utilizado o SDK (*software development kit*) da AWS para a linguagem Javascript.

Na criação da interface de usuário *front-end*, utilizou-se novamente a IDE WebStorm, juntamente com a biblioteca *React* v18 e outras bibliotecas e ferramentas auxiliares como *react-use-websocket* para conexão com o servidor, *react-chartjs-2* para visualização de dados, e Sass para estilização. A ferramenta AWS Amplify foi empregada para a implantação do código e integração contínua/entrega contínua (CI/CD). Finalmente, para gerenciamento de versões e hospedagem de código, foram usados o Git e o GitHub, respectivamente.

5.3 Módulo do microcontrolador

O código, o qual pode ser visto no Anexo 1, implementado no microcontrolador ESP32, foi desenvolvido para ler os dados do sensor de temperatura e umidade DHT11 e enviar esses dados para um servidor via *WebSocket*. Abaixo, cada seção desse código é explicada detalhadamente.

5.3.1 Importação das bibliotecas

O código começa importando todas as bibliotecas necessárias para seu funcionamento (Figura 8). Isso inclui bibliotecas para interagir com o sensor DHT11 (*DHT.h*), gerenciar a conexão Wi-Fi (*esp_wifi.h* e *WiFiMulti.h*), enviar e receber dados via *WebSocket* (*WebSocketsClient.h*), processar dados JSON (*ArduinoJson.h*) e gerenciar temporizadores (*timer.h*).

Figura 8 - Inclusão das bibliotecas usadas no código do ESP32

```c++
#include <DHT.h>
#include <esp_wifi.h>
#include <Arduino.h>
#include <ArduinoJson.h>
#include <WiFiMulti.h>
#include <WebSocketsClient.h>
#include <timer.h>
```

Fonte: Autoria própria (2023).

5.3.2 Definição das constantes

Depois de importar as bibliotecas, o código define algumas constantes que serão usadas ao longo do programa (Figura 9). O WIFI_SSID e WIFI_PWD são o nome e a senha da rede Wi-Fi à qual o ESP32 vai se conectar. O LED_PIN é o pino do ESP32 conectado ao LED que será usado como indicativo da conexão bidirecional. O DHT_PIN e DHT_TYPE se referem ao pino de conexão e tipo do sensor DHT, neste caso, um DHT11.

Em relação ao servidor *WebSocket*, WS_HOST e WS_PORT representam o endereço do servidor e a porta a ser utilizada, no caso, a porta 443 para conexões seguras (SSL). É relevante mencionar que o WS_HOST é definido usando um parâmetro disponibilizado pelo serviço API Gateway, após a infraestrutura *serverless* ser devidamente implantada, que será abordado na seção 5.4. Já o WS_PATH é o caminho para a conexão *WebSocket* no servidor, onde o dispositivo deve anexar seu ID. O WS_FINGERPRINT e WS_PROTOCOL são a impressão digital do certificado SSL do servidor e o protocolo a ser utilizado, respectivamente, que é o WSS (*WebSocket Secure*).

Por fim, o JSON_DOC_SIZE e MSG_SIZE são o tamanho em bytes do documento JSON a ser processado e o tamanho máximo da mensagem a ser enviada ao servidor.

Figura 9 - Definição dos parâmetros globais

```
#define WIFI_SSID "your_wifi_ssid"
#define WIFI_PWD "your_wifi_password"
#define LED_PIN led_pin_number
#define DHT_PIN dht_pin_number
#define DHT_TYPE DHT11
#define WS_HOST "your_websocket_host"
#define WS_PORT port_number
#define WS_PATH "your_websocket_path"
#define WS_FINGERPRINT "your_ssl_fingerprint"
#define WS_PROTOCOL "wss"
#define JSON_DOC_SIZE json_document_size
#define MSG_SIZE message_size
```

Fonte: Autoria própria (2023).

5.3.3 Instanciação dos objetos

Em seguida, o código cria instâncias dos objetos que serão usados (Figura 10): WiFiMulti para gerenciar a conexão Wi-Fi, WebSocketsClient para a comunicação *WebSocket*, Timer para a programação de eventos, e DHT para o sensor DHT11.

Figura 10 – Criação das instâncias dos objetos.

Fonte: Autoria própria (2023).

5.3.4 Funções auxiliares

O código define algumas funções auxiliares que irão ajudar na implementação das funcionalidades desejadas. Isso inclui funções para enviar mensagens de erro para o servidor (*sendErrorMessage*), lidar com o *payload* recebido do servidor (*handlePayload*), gerenciar eventos do *WebSocket* (*webSocketEvent*) e enviar os dados lidos do sensor DHT11 para o servidor (*sendDHTData*).

- Função *sendErrorMessage*

Essa função recebe uma mensagem de erro como parâmetro e envia essa mensagem ao servidor como uma *string* JSON. Ela usa a função *sprintf*() para formatar a mensagem de erro em um objeto JSON, que inclui o tipo de ação ("*msg*"), o tipo de mensagem ("*error*") e o corpo da mensagem (o erro em si).

- Função *handlePayload*

A função *handlePaylod*() processa os dados recebidos do servidor, desserializando o *payload* através da biblioteca ArduinoJson e verificando possíveis erros. Se uma ação do tipo "*toggle_led*" é identificada, a função manipula o estado do LED conforme especificado no *payload* e retorna o novo estado para o servidor. Em caso de erros durante a desserialização, erros

comuns do servidor ou falha na operação do LED, mensagens de erro adequadas são enviadas de volta ao servidor.

- Função *webSocketEvent*

Essa função é chamada sempre que um evento ocorre na conexão *WebSocket*. Ela usa um switch para verificar o tipo de evento e executa diferentes ações dependendo do tipo.

- Função *notifyLedState*

A função *notifyLedState*() é responsável por atualizar o servidor em relação ao estado do LED do dispositivo. O estado atual é lido, e cria-se uma mensagem de texto formatada (JSON) indicando se o LED está ligado ou desligado. Por fim a mensagem é enviada ao servidor por meio do cliente *WebSocket*.

Essa função é vinculada a um timer, definido na função setup(), que a aciona em um intervalo específico, neste caso, a cada 750 milissegundos. Esse procedimento constante de envio do estado atual do LED ajuda a manter novos usuários conectados atualizados sobre o status.

- Função *sendDHTData*

Esta função é chamada a cada segundo para ler os dados do sensor DHT11 e os transmitir aos usuários. São lidos os valores de temperatura e umidade, é verificado se a leitura foi bem-sucedida e, se sim, os dados são formatados em uma *string* JSON enviados aos usuários conectados.

- Função *setup*

A função *setup*() é executada apenas uma vez após a inicialização do microcontrolador ou após um *reset*. Além disso, configura o hardware e estabelece conexões necessárias para o funcionamento do programa.

Aqui, o código executa uma série de ações:

a) Inicia a comunicação serial com uma velocidade de 115200 bits por segundo.
b) Configura os pinos LED_BUILTIN e LED_PIN como saída.

c) Adiciona as credenciais da rede Wi-Fi que o ESP32 deve se conectar.
d) Executa um loop até que a conexão com a Wi-Fi seja estabelecida, piscando o LED embutido durante a tentativa de conexão.
e) Imprime no console serial que a conexão foi estabelecida e mostra o endereço IP do ESP32 na rede.
f) Cria uma *string deviceId* única para este ESP32 com base no endereço MAC do dispositivo.
g) Cria o caminho final para a conexão *WebSocket* adicionando o *deviceId* ao WS_PATH.
h) Inicia a conexão *WebSocket* segura (WSS) com o servidor, vincula a função de manipulação de eventos e inicia o sensor DHT.
i) Configura o timer para chamar a função *sendDHTData* a cada 1 segundo.
j) Configura o timer para chamar a função *notifyLedState* a cada 750 milissegundos e anexa o timer ao loop.

- Função loop

A função loop() é executada continuamente após o término da função setup(), e é responsável pela lógica principal do programa.

Na função loop():

a) O LED embutido é ligado se a conexão Wi-Fi estiver ativa e desligado se não estiver.
b) A função wsClient.loop() é chamada para manter a conexão *WebSocket* ativa e processar qualquer dado recebido.
c) A função timer.tick() é chamada para atualizar o estado do timer e executar a função sendDHTData() se o intervalo de tempo especificado tiver passado.

5.4 Infraestrutura *Serverless*

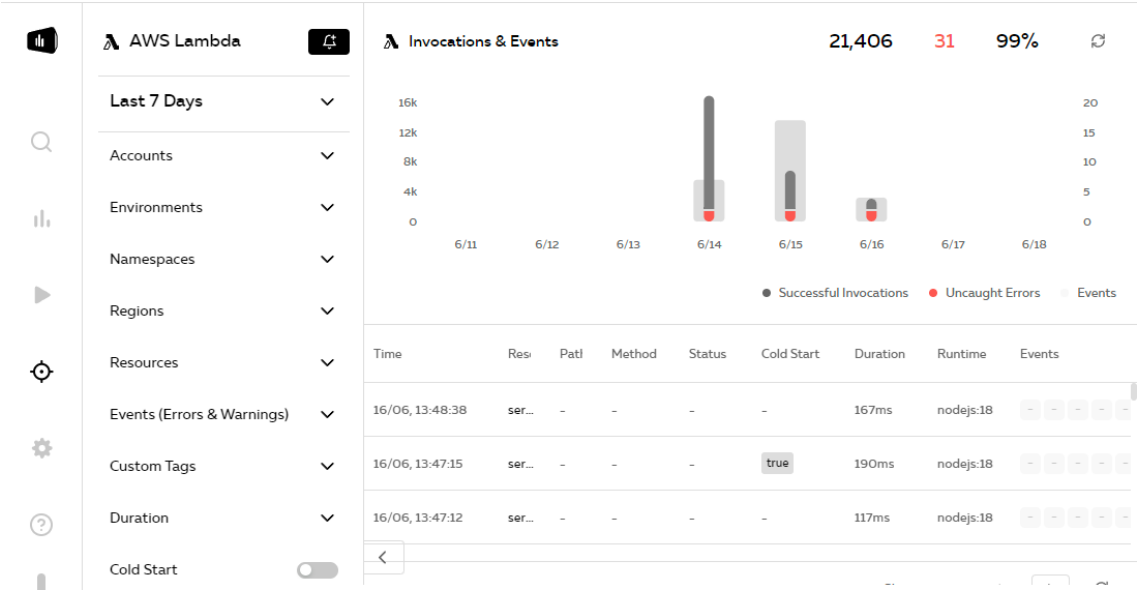
A aplicação deste projeto contou com a efetiva utilização do framework *Serverless* para o gerenciamento e provisionamento dos recursos necessários. Desta maneira, a definição dos recursos, incluindo funções AWS Lambda, API

Gateway, tabelas no banco de dados não-relacional *DynamoDB*, entre outros, foi estabelecida de forma declarativa através do arquivo *serverless.yml*. Esse arquivo é uma representação YAML da infraestrutura que deve ser criada.

O *Serverless* fornece, além disso, um painel - o *Serverless Dashboard* (Figura 11) - que proporciona o monitoramento e rastreamento dos recursos em execução. Este painel revela-se de grande utilidade para a identificação de eventuais erros e para a análise do comportamento da aplicação, uma vez que podem ser observadas métricas como a latência, o número de execuções das funções Lambda, incluindo os sucessos e falhas, o detalhamento do funcionamento interno de cada Lambda executado e uma visão geral da infraestrutura ao longo do tempo através de gráficos.

No cerne deste projeto encontram-se as funções Lambda, responsáveis por lidar com as solicitações recebidas pela API. Cada função é disparada por uma rota *WebSocket* específica: *$connect*, *$disconnect* e *msg*. A seguir, detalha-se a lógica por trás de cada função, iniciando pela função responsável pela rota *$connect*.

Figura 11 - *Serverless Dashboard*.



Fonte: Autoria própria (2023).

Essa função é acionada sempre que um cliente estabelece uma nova conexão *WebSocket*. A implementação é encontrada no arquivo *connect.ts*. O trecho de código a seguir mostra sua interface (Figura 12).

Figura 12 - Função Handler

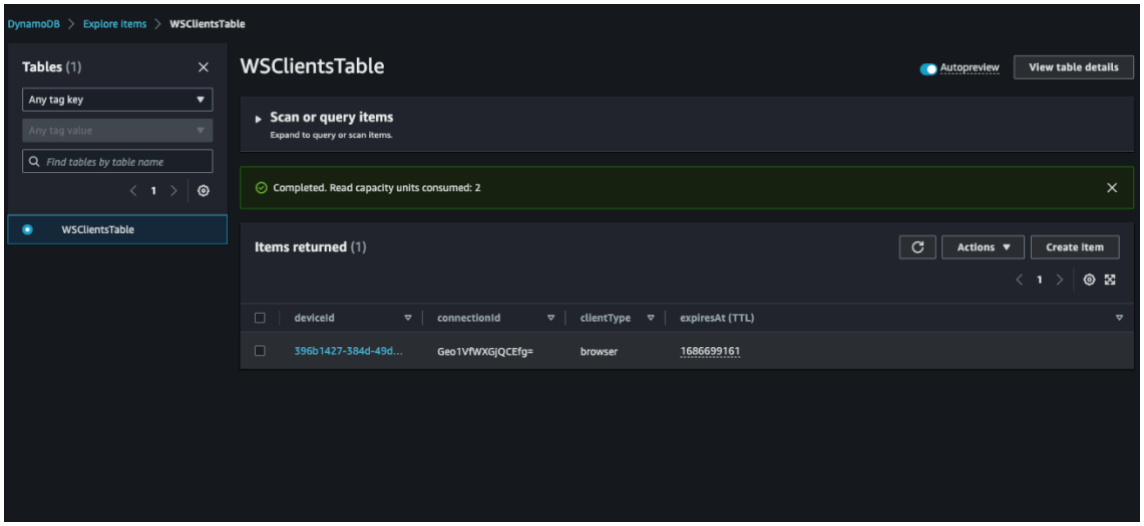
```
export const handler = async (
    event: APIGatewayProxyEvent, context: any
): Promise<APIGatewayProxyResult> => {...};
```

Fonte: Autoria própria (2023).

A função *handler* recebe um evento APIGatewayProxyEvent, que representa a requisição recebida, e um objeto de contexto, que contém informações sobre o ambiente de execução. O retorno pode ser uma resposta bem-sucedida ou uma mensagem de erro.

O comportamento desta função é o seguinte: inicialmente, extrai o ID da conexão e os parâmetros da solicitação. Em seguida, verifica no banco *DynamoDB* se já existe um cliente com o mesmo *deviceId*. Caso exista, o registro antigo é substituído pelo novo. Se não existir, simplesmente adiciona-se o novo cliente à tabela (Figura 13). Este comportamento garante que cada dispositivo tenha uma única entrada válida na tabela a qualquer momento.

Figura 13 - Tabela de conexões.



Fonte: Autoria própria (2023).

Caso ocorra algum problema durante a execução, o erro é capturado e registrado, facilitando a posterior análise e correção.

A função retorna, por fim, uma resposta com status HTTP 200 para indicar que a conexão foi bem-sucedida, ou um status HTTP 500, com a mensagem do erro ocorrido, caso haja falhas durante sua execução.

Após detalhar a função que trata a conexão de clientes, a próxima etapa natural é examinar a função responsável por lidar com o desligamento dessas conexões. Esta função é acionada pela rota *$disconnect* da API *WebSocket*. Tal função encontra-se no arquivo *disconnect.ts*, e possui a mesma interface que a função da rota *$connect*, conforme a Figura 12.

Semelhante à função *connect*, a função *disconnect* recebe como entrada um evento APIGatewayProxyEvent e um objeto de contexto. A lógica por trás dessa função é mais simples: primeiro, o ID da conexão é extraído do evento recebido. Em seguida, realiza-se uma busca na tabela *DynamoDB* para encontrar o cliente correspondente a esse ID de conexão.

Se o cliente for encontrado, este é excluído da tabela *DynamoDB*, o que efetivamente desassocia o dispositivo da conexão *WebSocket*. Caso ocorra um erro durante a execução da função, o *Serverless SDK* é novamente utilizado para capturar e registrar o erro.

Assim como na função *connect*, a resposta padrão para uma execução bem-sucedida é um status HTTP 200. No caso de falhas durante a execução, um status HTTP 500 é retornado, juntamente com a mensagem do erro ocorrido.

É importante destacar que o comportamento da função *disconnect* é crucial para a manutenção da tabela *DynamoDB* atualizada. Ao eliminar as entradas de clientes que não estão mais conectados, a tabela permanece otimizada e o desempenho geral da aplicação é mantido.

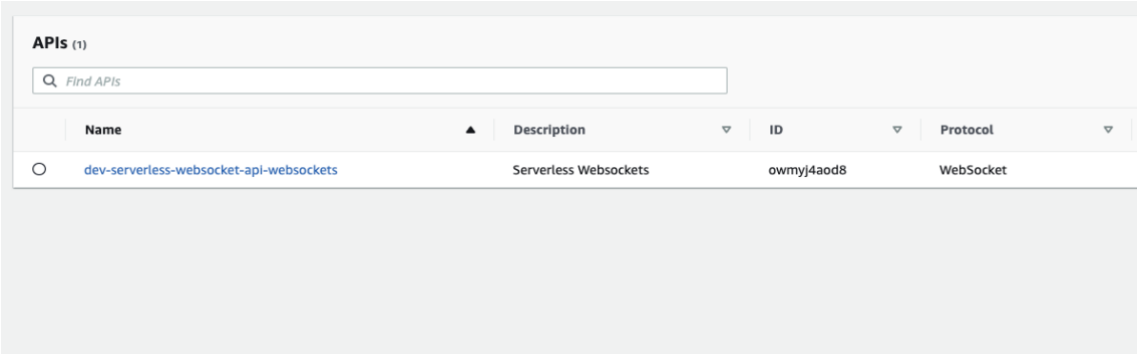
A última das funções Lambda criadas para esse projeto é a função que lida com o recebimento de mensagens dos clientes. Essa função é acionada pela rota *msg* da API *WebSocket*, sendo definida no arquivo *message.ts*. A definição e interface mantém o padrão das demais.

Nesta função, mais uma vez, o evento APIGatewayProxyEvent e o objeto de contexto são recebidos como entrada. O ID da conexão, o ID do dispositivo e o tipo de cliente são extraídos do evento recebido. Além disso, o corpo da mensagem é obtido do evento. Uma instância do APIGatewayManagement é criada, com o *endpoint* sendo definido a partir da variável de ambiente WSS_API_GATEWAY_ENDPOINT, gerada a partir do provisionamento de

recursos no arquivo *serverless.yml*. Essa instância é usada posteriormente para enviar mensagens para os clientes.

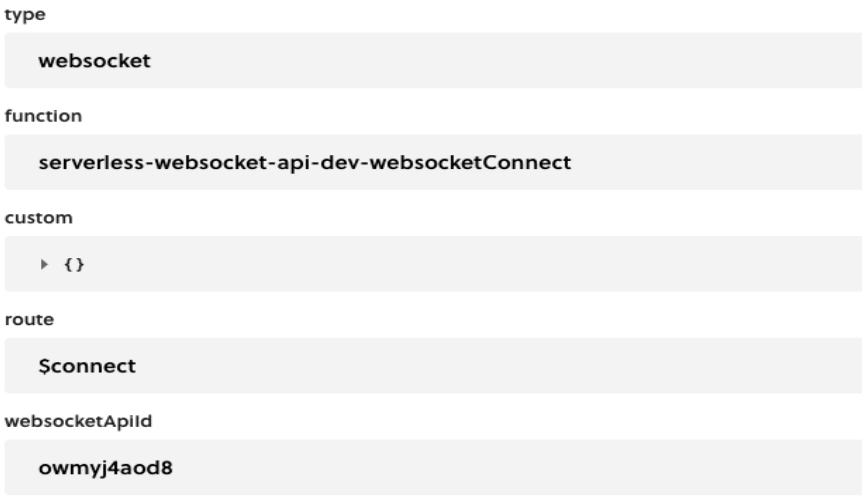
É importante notar que a variável de ambiente acima é criada usando o parâmetro WebsocketsApi, que se trata do ID fornecido pelo serviço API Gateway, ao ser provisionado pela ferramenta *Serverless*, e, conforme mencionado no tópico 5.3.2, este é o parâmetro usado para a definição da constante WS_HOST. Podemos conferir esse ID dentro da página do serviço (Figura 14), ou no *Serverless Dashboard* (Figura 15).

Figura 14 - ID fornecido no *API Gateway*.



Fonte: Autoria própria (2023).

Figura 15 - *Serverless Dashboard*.



Fonte: Autoria própria (2023).

A função *handler* obtém a lista completa de clientes conectados fazendo uma varredura no banco de dados. Em seguida, define-se uma função *sendMessage* interna que é usada para enviar mensagens aos clientes. Esta

função recebe um ID de conexão e um corpo de mensagem como argumentos. A função *sendMessage* tenta enviar uma mensagem para a conexão especificada. Caso ocorra um erro durante a operação e o erro seja do tipo *GoneException*, o que implica na conexão não ser mais existente, o cliente correspondente é excluído do banco.

Todos os clientes retornados pela varredura da tabela *DynamoDB* são percorridos pela função. Se um cliente não é o remetente da mensagem atual e não é do mesmo tipo e não tem o mesmo ID de dispositivo do remetente, a mensagem é enviada para este cliente. Isso garante que a mensagem seja enviada a todos os clientes que deveriam recebê-la. Se não houver clientes conectados, a função envia uma mensagem de aviso ao remetente.

Assim, o Lambda da rota *msg* serve como um encaminhador de mensagens, garantindo que sejam distribuídas corretamente entre os clientes conectados. Essa lógica de encaminhamento de mensagens é fundamental para a funcionalidade do dispositivo IoT e da interface do usuário web. A comunicação eficiente e precisa entre os componentes é crucial para a operação correta do sistema como um todo. O código desta seção consta no Anexo 3.

5.5 Criação da interface

A Interface do Usuário (IU) desempenha um papel crucial neste projeto para o estabelecimento de uma comunicação bidirecional eficaz entre os usuários e os dispositivos. A IU em questão foi desenvolvida utilizando a *React*, uma biblioteca JavaScript popular para a construção de interfaces interativas. Este tópico descreve sua implementação e o código está presente no Anexo 2.

5.5.1 Implementação e distribuição da IU usando *AWS Amplify*

Utilizando o serviço AWS Amplify, a interface do usuário foi implementada e distribuída. Esse serviço, integrado ao ecossistema AWS, oferece uma solução robusta para o desenvolvimento de aplicações.

Posteriormente ao desenvolvimento da interface, o código-fonte foi armazenado em um repositório Git, configurado junto a um novo aplicativo no AWS Amplify. Dessa forma, a cada atualização enviada ao repositório, o serviço identifica as alterações, executa a construção da aplicação e realiza o *deploy* da versão atualizada - um processo automatizado de *Continuous Deployment*.

O AWS Amplify também providencia um URL seguro (HTTPS), vide o Anexo 4, garantindo uma transmissão de dados protegida entre cliente e interface. Através desse processo, a aplicação tornou-se acessível na *web*, facilitando a interação do usuário com os dispositivos microcontroladores.

5.5.2 Desenvolvimento

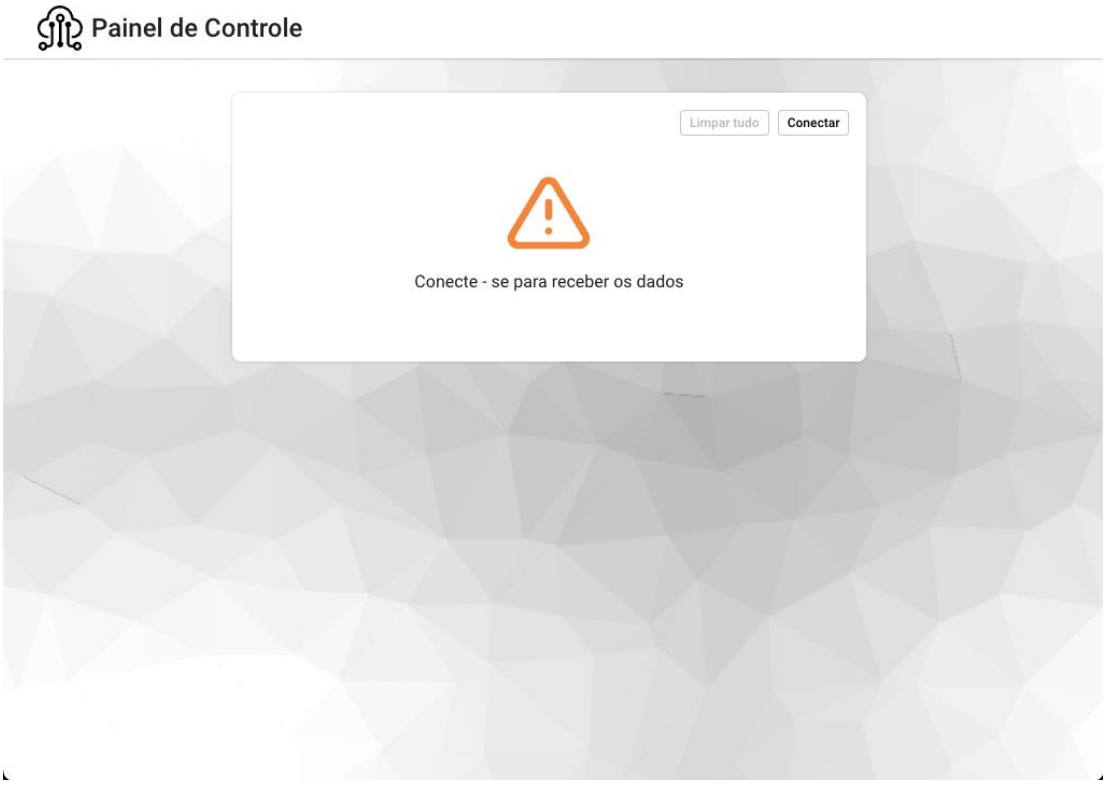
A IU foi projetada para permitir uma conexão dinâmica com o servidor *WebSocket*, armazenar e gerenciar o estado do LED, e representar os dados dos sensores em gráficos de linha. Cada uma dessas características é implementada como componentes independentes, garantindo modularidade e facilitando a manutenção do código.

O principal componente, denominado App, é responsável por estabelecer a conexão com o servidor *WebSocket*, manter o estado do LED e dos dados do gráfico, além de gerenciar o armazenamento local do identificador do dispositivo. Para isso, é utilizado o pacote *react-use-websocket*, que permite a conexão *WebSocket* e oferece diversos *hooks* e métodos úteis.

No momento da conexão, o App se comunica com o servidor *WebSocket* através do URL de acesso ao API Gateway e consulta os parâmetros, que incluem o tipo de cliente e o identificador do dispositivo (*deviceId*) presentes no armazenamento local do navegador. Se nenhum *deviceId* existir, um novo é gerado usando a biblioteca *uuid* e é armazenado no *local storage*. Os estados do LED e dos dados do gráfico são inicialmente definidos como objetos vazios e são atualizados conforme as mensagens chegam do servidor *WebSocket*.

O componente *Main* da IU serve como o componente de exibição principal e aceita uma variedade de parâmetros, incluindo o estado do LED, os dados do gráfico, a conexão *WebSocket* e os manipuladores de evento para limpar dados e alternar o estado do LED. Além disso, este componente apresenta ao usuário a possibilidade de conectar ou encerrar a conexão *WebSocket* (Figura 16), limpar todos os dados recebidos e, se os dados estiverem presentes, ligar ou desligar o LED. Os dados dos sensores são exibidos em gráficos de linha.

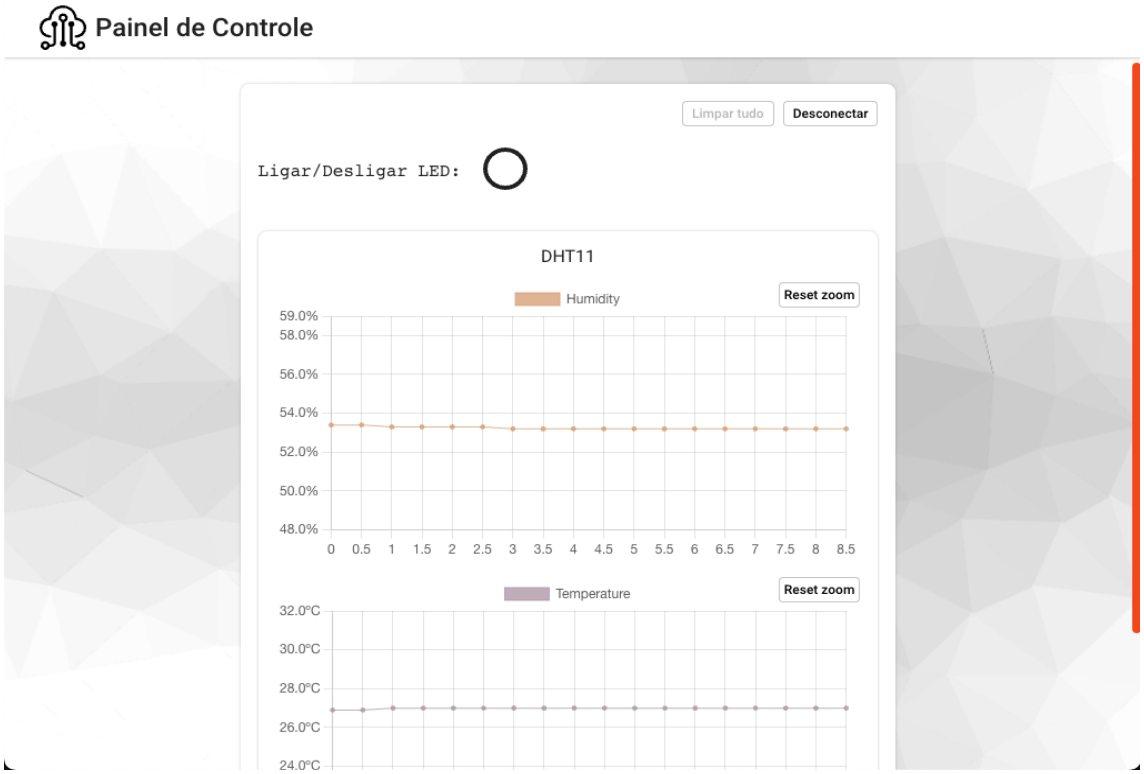
Figura 16 - Interface do usuário - desconectado.



Fonte: Autoria própria (2023).

Esta interface fornece uma ponte importante entre o código do microcontrolador e o código *serverless*. Os dados enviados pelo microcontrolador são recebidos pelo serviço API Gateway e transmitidos para a interface através da conexão *WebSocket*. A IU, por sua vez, processa esses dados e os exibe de forma intuitiva para o usuário (Figura 17). Além disso, as ações do usuário na interface, como alternar o estado do LED, são enviadas de volta ao servidor e posteriormente para o microcontrolador.

Figura 17 - Interface do usuário - conectado



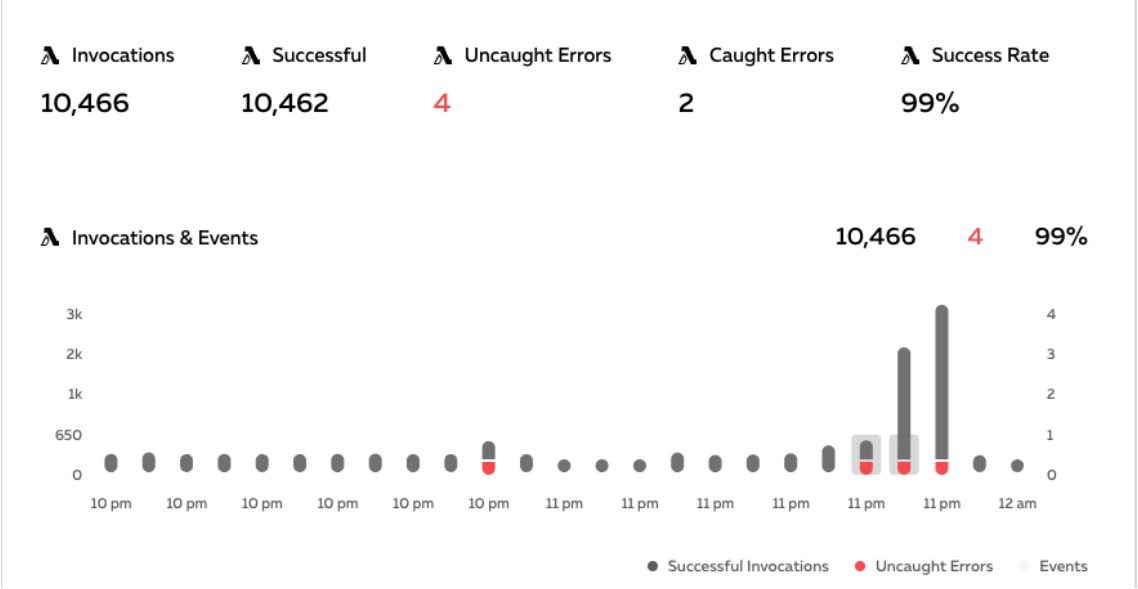
Fonte: Autoria própria (2023).

6. RESULTADOS E DISCUSSÕES

Neste capítulo, busca-se elucidar e analisar os resultados obtidos ao longo da fase de experimentação, na qual o sistema embarcado proposto foi monitorado por um período de duas horas ininterruptas. A análise leva em conta o desempenho de comunicação entre o dispositivo e o usuário, bem como a eficiência do sistema.

Ao longo do período observado, registrou-se um total de 10466 invocações Lambda, provenientes da interação usuário-dispositivo. Nesse contexto, é notável que somente quatro invocações resultaram em falhas, sendo elas devido a desconexão do usuário por parte do navegador em momentos de inatividade, que culminou no disparo do evento de erro *GoneException*, quando o destinatário de uma mensagem não está mais presente. Este comparativo entre invocações bem-sucedidas e eventuais falhas demonstram uma eficiência de aproximadamente 99,96 % na execução das funções lambda (Figura 18). Esta taxa de sucesso ressalta a confiabilidade da estrutura adotada e evidencia a robustez do sistema em lidar com a interação intensiva e contínua entre usuário e dispositivo.

Figura 18 - Total de invocações Lambda durante o período de observação.



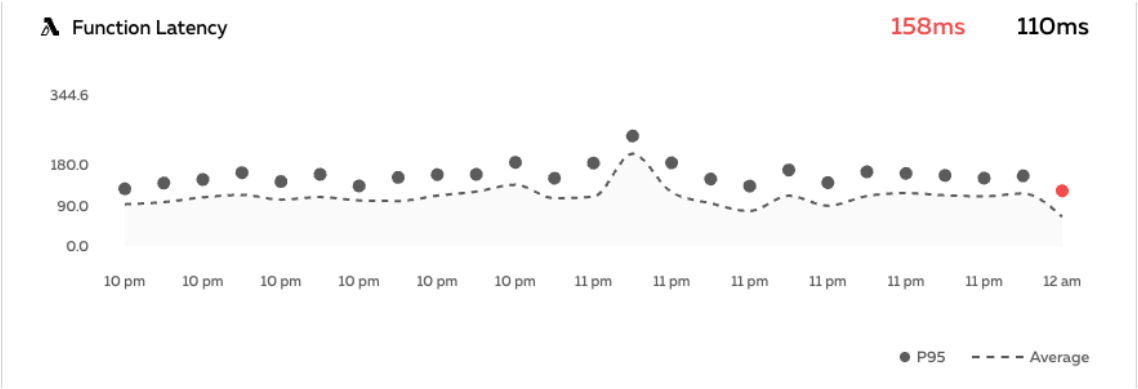
Fonte: Autoria própria (2023).

É interessante notar os picos de invocações na Figura 18, que aconteceram durante um período de dez minutos devido á conexão de outros

dois usuários, totalizando três, que passaram a se comunicar com o sistema embarcado simultaneamente. Apesar do expressivo aumento, o sistema se manteve estável, sem aumento de latência (Figura 19).

Focando na latência do sistema, é essencial destacar que esse parâmetro desempenha um papel crucial na avaliação da performance da comunicação em sistemas embarcados. A latência média passível de cobrança pela AWS durante o período de monitoramento foi de 110 ms, enquanto a latência P95 – um indicador que denota que 95 % das invocações foram concluídas dentro de determinado tempo ou menos – alcançou o valor de 158 ms (Figura 19). Ainda que essa não seja a latência real percebida pelo usuário, é uma métrica importante para a validação da performance do sistema, visto que este resultado demonstra um comportamento de latência extremamente consistente, onde a grande maioria das invocações foi processada em um tempo inferior a 200 ms.

Figura 19 - Progressão das latências médias durante o período de observação



Fonte: Autoria própria (2023).

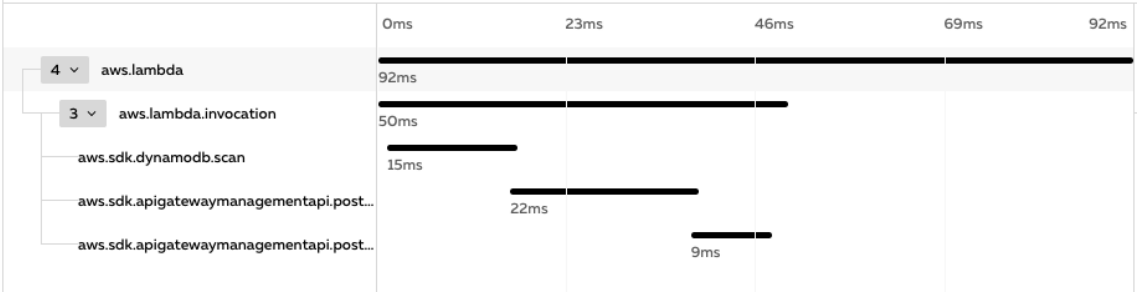
Um fenômeno que precisa ser considerado quando se analisa a latência de um sistema *serverless* é o chamado *cold start*. Este ocorre quando uma instância do serviço precisa ser iniciada do zero pela AWS, geralmente após períodos de inatividade, mas também quando a instância cai por algum motivo. Essa inicialização pode pontualmente aumentar a latência percebida pelo usuário, que, no contexto deste projeto, foi de 1.5 s em média (Figura 20) durante as situações de *cold start* ao somar também o tempo de execução do Lambda.

Figura 20 - Invocação Lambda com *cold start*

Fonte: Autoria própria (2023).

No entanto, a latência média percebida pelo usuário se manteve na faixa de 50 ms (Figura 21). Esta diferença em relação à latência média geral de 110 ms e à latência P95 de 158 ms é atribuída ao tempo de pós-processamento da função Lambda, que continua mesmo após o envio do resultado ao usuário. Através da Figura 21 é possível constatar a duração efetiva da invocação do Lambda no item *aws.lambda.invocation*.

Figura 21 - Invocação Lambda padrão



Fonte: Autoria própria (2023).

É igualmente importante destacar que, durante o período de monitoramento, não foram observados conflitos no sistema ao se acionar o LED através da interface enquanto ela estava em processo de recebimento de dados do sensor. Esta observação é de particular relevância, pois indica a capacidade do sistema de manejar comandos de forma bidirecional e simultânea, mantendo sua estabilidade e funcionalidade. Considerando o fluxo de informação entre o usuário e o sistema embarcado para o acionamento do LED, que envolve a transmissão e recebimento de dados (ida e volta), a latência média experimentada pelo usuário foi de 100 ms.

Os dados apresentados, juntamente com a análise subsequente, demonstram a eficiência do sistema proposto na comunicação entre o dispositivo embarcado e o usuário, fornecendo uma experiência de interação suave e responsiva. A alta taxa de sucesso nas invocações lambda e a latência baixa e consistente são indicativos da robustez do sistema implementado.

7. CONCLUSÃO

De posse dos resultados obtidos e da verificação do projeto, alguns pontos chaves podem ser destacados. O sistema embarcado proposto demonstrou robustez e confiabilidade na coleta e transmissão de dados. A latência média percebida pelo usuário se manteve em níveis baixos, oferecendo uma experiência de quase tempo real.

O esquema de arquitetura adotado com a divisão das responsabilidades entre o microcontrolador, a infraestrutura *serverless* e a interface de usuário provou ser eficaz, escalável e flexível. Ao manter cada módulo com um propósito específico e limitado, foi possível isolar falhas e otimizar cada componente de acordo com suas necessidades particulares.

A escolha pelo microcontrolador ESP32 se mostrou acertada, dada sua robustez, versatilidade e baixo custo. Este componente se demonstrou capaz de suportar a carga de trabalho exigida, mantendo a comunicação bidirecional não conflitante com a interface de usuário e o serviço AWS Lambda, reafirmando seu potencial como uma solução viável para uma gama de aplicações em sistemas embarcados.

Finalmente, a aplicação da computação em nuvem por meio do AWS API Gateway e AWS Lambda possibilitou um processamento de dados eficiente e escalável, minimizando a latência e oferecendo alta disponibilidade. O entendimento das características particulares desse serviço, como o fenômeno do *cold start*, é fundamental para projetar soluções otimizadas.

Uma questão a se notar é o fato de que a latência observada, ainda que pequena, deve estar sujeita aos critérios de construção do sistema no qual esta solução seja utilizada. Caso seja necessária uma frequência de amostragem muito rápida, próximo de 100 ms ou inferior, a latência medida pode representar a inviabilidade do uso da solução.

Considerando a supracitada observação, o sistema proposto ainda demonstra, portanto, aplicabilidade potencial em um amplo espectro de áreas, uma vez que não limita quais dados podem ser coletados, ou qual o nível de controle pode ser exercido sobre as aplicações em que esteja empregado. Apesar de ter sido testado em um cenário específico, ele foi projetado para ser facilmente adaptável a outros, reforçando a sua versatilidade.

Entre as aplicações imediatas deste projeto, dada as devidas adaptações para cada cenário e necessidade, pode-se listar:

- **Monitoramento de laboratório:** em laboratórios de pesquisa, o sistema poderia ser usado para coletar dados de diversos equipamentos de forma remota, permitindo o monitoramento e controle em tempo real de variáveis como temperatura, pressão, pH, entre outros. Isso seria útil tanto para experimentos que precisam de um acompanhamento constante, quanto para a manutenção das condições ideais de trabalho.
- **Controle de equipamentos de pesquisa:** o sistema poderia ser aplicado ao controle remoto de equipamentos utilizados em pesquisas acadêmicas, permitindo que os pesquisadores ajustem as configurações e parâmetros de experimentos à distância, o que pode aumentar a eficiência dos procedimentos.

De forma mais específica, trazendo para o contexto acadêmico da Universidade Federal de Uberlândia onde este trabalho se insere, a solução proposta tem aplicabilidade em um dos projetos desenvolvidos pelo Centro Brasileiro de Referência em Inovações Tecnológicas para Esportes Paralímpicos – CINTESP.Br. O projeto em questão consiste, de forma resumida, na criação de uma bicicleta para o treinamento e avaliação ao longo do tempo de atletas paralímpicos, através da configuração de mecanismos que permitem a variação do esforço exigido pelo atleta. Nesse sentido, a solução de controle e monitoramento é útil para que tais configurações sejam feitas de forma facilitada, sem envolver o desmonte de equipamentos embarcados, e os dados possam ser monitorados e armazenados conforme necessário. Além disso, por se tratar de um projeto em desenvolvimento, é ideal que as ferramentas que o circundam sejam flexíveis e de fácil adaptação.

Como toda pesquisa, este trabalho abre portas para investigações futuras. Aspectos como a implementação de um mecanismo de cache para maior otimização da latência, a exploração de outras plataformas de microcontroladores, ou a validação de diferentes protocolos de comunicação dentro do mesmo cenário, podem ser investigados em trabalhos subsequentes.

Por fim, é possível afirmar que o objetivo principal do projeto foi atingido: a implementação e avaliação de um sistema de controle remoto eficiente e de baixo custo para sistemas embarcados utilizando *WebSocket* e a infraestrutura *serverless* com o microcontrolador ESP32. Este trabalho, portanto, contribui para a consolidação do conhecimento no campo dos sistemas embarcados e abre novos caminhos para investigações futuras.

REFERÊNCIAS

ALBERTIN, A. L.; ALBERTIN, R. M. A Internet das Coisas irá muito além das Coisas. GV-Executivo, vol. 16, n. 2, p. 12-17, mar./abr. 2017.

AL-FUQAHA, A. et al. Internet of Things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & Tutorials, v. 17, n. 4, p. 2347-2376, 2015.

AUTO CORE ROBÓTICA. Placa de Desenvolvimento Wifi Bluetooth ESP32. Disponível em: <https://www.autocorerobotica.com.br/placa-de-desenvolvimento-wifi-bluetooth-esp32>. Acesso em: 10 de junho de 2023.

ASHTON, K., et al. That "Internet of Things thing". RFID Journal, v. 22, n. 7, p. 97-114, 2009.

AWS. AWS API Gateway. Disponível em: <https://aws.amazon.com/api-gateway/?nc2=type_a>. Acesso em: 10 de junho de 2023.

AWS. AWS Lambda. Disponível em: <https://aws.amazon.com/lambda/?nc1=h_ls>. Acesso em: 10 de junho de 2023.

BAHGA, A.; MADISETTI, V. Internet of Things: A hands-on approach. Estados Unidos. Vijay Madisetti, 2014.

BALALAIE, A.; HEYDARNOORI, A.; JAMSHIDI, P. Microservices architecture enables DevOps: Migration to a cloud-native architecture. Ieee Software, v. 33, n. 3, p. 42-52, 2016.

BANKS, A.; PORCELLO, E. Learning React: functional web development with React and Redux. 1ª ed. Estados Unidos. O'Reilly Media, 2017.

BAYILMIŞ, C., et al. A survey on communication protocols and performance evaluations for Internet of Things. Digital Communications and Networks, 2022.

BROWSEE. WebSocket vs HTTP Calls – Performance Study, 2019. Disponível em: < https://browsee.io/blog/websocket-vs-http-calls-performance-study/>. Accesso em: 8 de jun. de 2023.

ESPRESSIF SYSTEMS. ESP32 Series Datasheet. Espressif Systems, 2023. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 8 de jun. de 2023.

FETTE, I.; MELNIKOV, A. RFC 6455: The WebSocket Protocol. Internet Engineering Task Force, 2011. Disponível em: < https://www.rfc-editor.org/rfc/rfc6455>. Acesso em: 8 de jun. de 2023.

FISCHER, L. React for Real: Front-End Code, Untangled. Pragmatic Bookshelf, 2017.

FLANAGAN, D. JavaScript: the definitive guide. 5ª ed. Estados Unidos. O'Reilly, 2006.

GRIMM, C.; NEUMANN, P.; MAHLKNECHT, S. (Ed.). Embedded Systems for Smart Appliances and Energy Management. Vol. 3. Springer Science & Business Media, 2012.

GUBBI, J., et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, v. 29, n. 7, p. 1645-1660, 2013.

HERMANN, M.; PENTEK, T.; OTTO, B. Design principles for industrie 4.0 scenarios. In: Hawaii International Conference on Systems Science. 2016. p. 3928–3937.

INTRODUCING JSON. JSON Org. Disponível em: <http://www.json.org/json-pt.html>. Acesso em: 10 de junho de 2023.

LIN, J.; YU, W.; ZHANG, N.; YANG, X.; ZHANG, H.; W. ZHAO, W., "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," in IEEE Internet of Things Journal. 2017. Vol. 4, no. 5, pp. 1125-1142.

JSCONF. Rethinking best practices in MVC. YouTube, 30 de outubro de 2013. Disponível em: <https://www.youtube.com/watch?v=x7cQ3mrcKaY&ab_channel=JSConf>. Acesso em: 10 de junho de 2023.

KOLBAN, N. Kolban's book on ESP32, 2ª Ed. Leanpub, 2018.
KOTEVSKI, Aleksandar; MIKROVSKI, Gjorgi; JOLEVSKI, Ilija. HTML5 Web Sockets.

LEE, I.; LEE,K. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. Business Horizons, v. 58, n. 4, p. 431-440, 2015.

LLOYD, W. et al. Serverless computing: An investigation of factors influencing microservice performance. In: 2018 IEEE international conference on cloud engineering (IC2E). IEEE, 2018. p. 159-169.

MAIER, A.; SHARP, A.; VAGAPOV, Y. Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things. In: 2017 Internet Technologies and Applications (ITA). IEEE, 2017. p. 143-148.

MARAS, J. Secrets of the JavaScript Ninja. Simon and Schuster, 2016.

MARWEDEL, P. Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things. Springer Nature, 2021.

MDN WEB DOCS. Aprendendo desenvolvimento web. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn>. Acesso em: 10 de junho de 2023.

MIORANDI, D., SICARI, S., DE PELLEGRINI, F. & CHLAMTAC, I. Internet of things: Vision, applications and research challenges. Ad hoc networks, v. 10, n. 7, p. 1497-1516, 2012.

OLIVEIRA, V. G.; NIIZU, F. Y.; BASSETO, F. Internet das Coisas. O Setor Elétrico, n. 132, p. 32-35, 2017. ISSN 1983-0912.

JAVA WebSocket Programming with Android and Spring Boot. PubNub. 18 nov. 2022. Disponível em: < https://www.pubnub.com/blog/java-websocket-programming-with-android-and-spring-boot/>. Acesso em: 10 de junho de 2023.

ROBERTS, M.; CHAPIN, J. What is Serverless?. O'Reilly Media, Incorporated, 2017.

SAM Solutions. Disponível em: <https://www.sam-solutions.com/blog/all-you-need-to-know-about-embedded-system-programming/>. Acesso em: 13 de junho de 2023.

SUBRAMANIAN, V. Pro MERN Stack. Apress, 2017.

VILLA-HENRIKSEN, A., EDWARDS, G. T., PESONEN, L. A., GREEN, O. & SORENSEN, C. A. Internet of Things in arable farming: Implementation, applications, challenges, and potential. Biosystems Engineering. Vol. 191, p. 60-84. 2020.

WANG, V.; SALIM, F.; MOSKOVITS, P. The definitive guide to HTML5 WebSocket. New York. Apress, 2013.

WOLF, M. Computers as components: principles of embedded computing system design. 5ª ed. São Francisco, EUA. Morgan Kaufmann, 2022.

ANEXOS

Anexo 1 - Código fonte do microcontrolador ESP32.

Disponível em: <https://github.com/paulovfmarques/esp32-client>.

Anexo 2 - Código fonte da interface do usuário.

Disponível em: <https://github.com/paulovfmarques/iot-frontend>.

Anexo 3 - Código fonte para o *backend serverless.*

Disponível em: <https://github.com/paulovfmarques/serverless-websocket-api>.

Anexo 4 - Website da interface do usuário.

Disponível em: <https://main.d1c1drp9ho5pkf.amplifyapp.com/>.