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

GUSTAVO RODRIGO DE ASSIS RODRIGUES

ESTUDO DA COMUNICAÇÃO BLUETOOTH LOW ENERGY (BLE) PARA MONITORAMENTO E CONTROLE DE DISPOSITIVOS

Sorocaba
2024

GUSTAVO RODRIGO DE ASSIS RODRIGUES

ESTUDO DA COMUNICAÇÃO BLUETOOTH LOW ENERGY (BLE) PARA MONITORAMENTO E CONTROLE DE DISPOSITIVOS

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

Orientador: Prof. Dr. Eduardo Paciencia Godoy

Sorocaba

2024

R696e

Rodrigues, Gustavo Rodrigo de Assis

Estudo da comunicação Bluetooth Low Energy (BLE) para monitoramento e controle de dispositivos / Gustavo Rodrigo de Assis Rodrigues. -- Sorocaba, 2024

88 p.

Trabalho de conclusão de curso (Bacharelado - Engenharia de Controle e Automação) - Universidade Estadual Paulista (UNESP), Instituto de Ciência e Tecnologia, Sorocaba

Orientador: Eduardo Paciencia Godoy

1. Sistemas de comunicação sem fio. 2. Internet das coisas. 3. Automação. 4. Tecnologias de informação e comunicação. 5. Controle remoto. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Universidade Estadual Paulista (UNESP), Instituto de Ciência e Tecnologia, Sorocaba. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

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

ESTUDO DA COMUNICAÇÃO BLUETOOTH LOW ENERGY (BLE) PARA MONITORAMENTO E CONTROLE DE DISPOSITIVOS

GUSTAVO RODRIGO DE ASSIS RODRIGUES

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

Prof. Dr. Everson Martins
Coordenador

BANCA EXAMINADORA:

Prof. Dr. Eduardo Paciencia Godoy
Orientador/UNESP – Câmpus de Sorocaba

Prof. Dr. Dhiego Fernandes Carvalho
UNESP – Câmpus de Sorocaba

Prof. Dr. Rodrigo Pita Rolle
IFSP – Câmpus Itapetininga.

Setembro de 2024

AGRADECIMENTOS

Primeiramente, expresso minha mais sincera gratidão a Deus, que esteve presente em cada etapa deste caminho, concedendo-me não apenas a oportunidade de cursar Engenharia de Controle e Automação na UNESP Sorocaba, mas também infundindo-me com a força, o vigor, a tenacidade e a sabedoria necessários para superar os desafios e alcançar a conclusão deste tão almejado capítulo de minha vida. Agradeço-Lhe, ainda, pela chance de disseminar os conhecimentos adquiridos neste projeto, beneficiando colegas, estudantes, pesquisadores e todos aqueles apaixonados pelo universo das tecnologias inteligentes.

Minha profunda gratidão se estende ao meu orientador, Prof. Dr. Eduardo Paciencia Godoy, cuja orientação e incentivo contínuos foram essenciais para o sucesso deste trabalho.

Um agradecimento especial aos meus pais, Oswaldo Augusto Rodrigues e Priscila de Assis Rodrigues, aos meus irmãos, Tales Augusto de Assis Rodrigues e Victor de Assis Rodrigues, a minha cunhada Renata Leme Nahum Rodrigues, ao meu sobrinho Eduardo, ao meu avô Oswaldo, aos meus tios e a toda a minha família, que foram luz para mim nesta jornada.

Não posso deixar de reconhecer a valiosa presença de meus amigos e colegas Guilherme Doubek Lopes, Matheus Salaroli, Lucas Okubo e Kaike Miranda, que estiveram ao meu lado nos momentos mais cruciais.

À UNESP Sorocaba e a todos os alunos com quem compartilhei experiências nestes anos, minha gratidão é imensa. Vocês foram a espinha dorsal de minha inspiração, impulsionando-me a desejar perpetuar este projeto entre vocês e as futuras gerações.

Este trabalho é o fruto de uma trajetória coletiva, e a cada um dos que contribuíram para a sua realização, o meu mais sincero e profundo agradecimento.

RESUMO

A comunicação sem fio de curto alcance tem se tornado cada vez mais essencial em um mundo onde a conectividade e a automação são cruciais para a inovação tecnológica. No campo da Engenharia de Controle e Automação, a compreensão e aplicação dessas tecnologias são fundamentais para o desenvolvimento de sistemas inteligentes que integram eficiência e flexibilidade em uma ampla gama de aplicações, desde a automação industrial até a Internet das Coisas (IoT). O presente trabalho tem como foco central trazer uma situação prática abordada na disciplina de Redes Industriais de Comunicação do curso de Engenharia de Controle e Automação da Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP) que envolve a utilização da comunicação Bluetooth Low Energy (BLE) para exercer o monitoramento e o controle de dispositivos. Com isso, este projeto se baseia na utilização de uma placa ESP32-LoRa (dotada de um display OLED, que permite a exposição dos dados de interesse) integrada a um sensor BME 280 para transmitir e receber dados de um dispositivo central (smartphone ou desktop Windows) utilizando uma aplicação desenvolvida com o MIT APP Inventor e a ferramenta Bluetooth LE Explorer. Dessa forma, com o experimento, busca-se estudar os principais conceitos e fundamentos que fazem parte da tecnologia BLE e estruturá-los através da criação de um sistema prático, permitindo a análise detalhada das funcionalidades da tecnologia, incluindo a criação de modelos de anúncio, a estruturação de perfis, serviços e características, e a utilização de ferramentas para análise de dados. Nesse sentido, o projeto demonstrou a viabilidade e as potencialidades de sistemas de comunicação ponto a ponto utilizando a ESP32, destacando também a existência de algumas limitações associadas à implementação da biblioteca BLECharacteristic, e explora práticas de implementação que permitiram a superação de desafios técnicos específicos. Através deste trabalho, espera-se fomentar a exploração de novas fronteiras no campo das comunicações sem fio e promover a integração de tecnologias *smart* no ambiente educacional.

Palavras-chave: Comunicação sem fio; Bluetooth Low Energy; Internet das Coisas; sistemas inteligentes; ambiente educacional

ABSTRACT

Short-range wireless communication has become increasingly essential in a world where connectivity and automation are crucial for technological innovation. In the field of Control and Automation Engineering, understanding and applying these technologies are fundamental for the development of intelligent systems that integrate efficiency and flexibility across a wide range of applications, from industrial automation to the Internet of Things (IoT). This work focuses on a practical situation addressed in the Industrial Communication Networks course of the Control and Automation Engineering program at Universidade Estadual Paulista “Júlio de Mesquita Filho” (UNESP), which involves the use of Bluetooth Low Energy (BLE) communication to monitor and control devices. This project is based on the use of an ESP32-LoRa board (equipped with an OLED display that allows the visualization of the data of interest) integrated with a BME280 sensor to transmit and receive data from a central device (smartphone or Windows desktop) using an application developed with MIT APP Inventor and the Bluetooth LE Explorer tool. The experiment aims to study the key concepts and fundamentals of BLE technology and structure them by creating a practical system, enabling a detailed analysis of the technology's functionalities, including the creation of advertisement models, the structuring of profiles, services, and characteristics, and the use of tools for data analysis. In this context, the project demonstrated the feasibility and potential of point-to-point communication systems using the ESP32, while also highlighting some limitations associated with the implementation of the BLECharacteristic library, and explores implementation practices that allowed overcoming specific technical challenges. Through this work, it is expected to foster the exploration of new frontiers in the field of wireless communications and promote the integration of smart technologies in the educational environment.

Keywords: Wireless communication; Bluetooth Low Energy; Internet of Things; intelligent systems; educational environment

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A comunicação wireless evoluiu significativamente desde suas origens na era pré-industrial, passando de métodos rudimentares como sinais de fumaça e heliógrafos até a sofisticação das atuais redes digitais sem fio. Avanços notáveis, como a invenção do telégrafo e do telefone, pavimentaram o caminho para a era das comunicações sem fio modernas. Um marco decisivo foi a transmissão de rádio pioneira de Guglielmo Marconi em 1895, que estabeleceu as bases para a comunicação por ondas de rádio (GOLDSMITH, 2005).

A partir dessas e outras conquistas, a tecnologia de transmissão de dados por ondas eletromagnéticas, em especial a radiofrequência, expandiu-se exponencialmente, permitindo comunicações mais distantes, de melhor qualidade e com menor consumo de energia (BANAFEA *et al.*, 2023). Isso viabilizou aplicações que vão desde a radiodifusão até as redes sem fio contemporâneas, como Wi-Fi e Bluetooth (ISLAM; JIN, 2019).

Entre as tecnologias emergentes, o Bluetooth Low Energy (também conhecido por Bluetooth 4.0, Bluetooth Smart, Bluetooth LE ou BLE), introduzido em 2009, destaca-se por oferecer comunicação eficiente e de baixo consumo energético para dispositivos em proximidade. Sua aplicabilidade abrange diversas áreas, desde dispositivos IoT (*Internet of Things*) até sistemas avançados de automação industrial, posicionamento indoor (*Indoor Positioning Systems* ou IPS) e geolocalização (GUGGEDAL, 2018).

No entanto, ainda persiste uma lacuna educacional no que tange ao engajamento de estudantes de nível médio e superior com as especificidades do BLE de maneira didática. Em resposta a essa necessidade, este trabalho propõe-se a elucidar o funcionamento de um sistema de comunicação BLE, com ênfase em funcionalidades de monitoramento e controle de dados, fazendo uma abordagem a um sistema utilizado no ensino da tecnologia na disciplina de Redes Industriais de Comunicação da Universidade Estadual Paulista “Julio de Mesquita Filho” (UNESP, Sorocaba).

Para tanto, foi desenvolvido um sistema de comunicação que integra uma placa da classe ESP32, como dispositivo servidor, e um dispositivo cliente para a transferência de dados e o monitoramento de informações dentro do alcance da conexão BLE. Com isso, este trabalho busca reunir os principais conceitos associados à implementação da tecnologia de modo a permitir a sua expansão para o desenvolvimento de outras aplicações práticas na área de tecnologias *smart*.

2 OBJETIVOS

O presente trabalho objetiva:

- Estudar a tecnologia de comunicação Bluetooth Low Energy a partir do desenvolvimento de um sistema de monitoramento e controle utilizando uma placa ESP32-LoRa conectada a um sensor BME 280, capaz de captar e transmitir dados de temperatura, pressão, altitude e umidade para um dispositivo cliente;

- Implementar a comunicação Bluetooth Low Energy entre a ESP32-LoRa e o dispositivo cliente, utilizando uma aplicação desenvolvida com MIT APP Inventor e o software Bluetooth LE Explorer;

- Permitir a interação bidirecional entre o dispositivo cliente e a ESP32-LoRa, incluindo a atualização dos valores coletados pelo sensor BME 280, a transmissão de mensagens de texto, o controle do LED interno da ESP32-LoRa e o monitoramento do botão PRG da mesma;

- Facilitar o entendimento da tecnologia BLE e seus principais conceitos no contexto da disciplina de Redes Industriais de Comunicação, aplicada no curso de Engenharia de Controle e Automação da UNESP – Sorocaba;

- Explorar e demonstrar as potencialidades e limitações da comunicação ponto a ponto utilizando a ESP32-LoRa, destacando a viabilidade e os desafios técnicos na implementação de um sistema BLE funcional.

3 REVISÃO CONCEITUAL

3.1 Bluetooth Low Energy

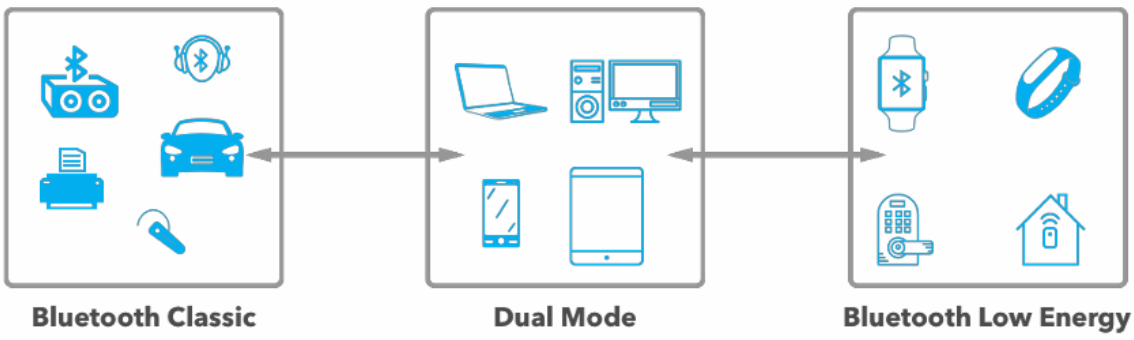
O Bluetooth (ou Bluetooth Clássico), tecnologia de comunicação sem fio de curta distância, evoluiu desde sua criação em 1994, permitindo o pareamento direto entre dispositivos (TRIGGS, 2023). No entanto, a busca por maior eficiência energética para aplicações de baixa taxa de dados e menor complexidade, como dispositivos IoT, levou ao desenvolvimento do Bluetooth Low Energy (BLUETOOTH SIG, 2024).

O BLE surgiu como versão 4.0 do Bluetooth. Porém, devido às significantes mudanças em seu funcionamento, foi considerado um novo protocolo de comunicação. Enquanto o Bluetooth Clássico é mais adequado para transferência de dados de alta velocidade, o BLE prioriza a economia de energia, tornando-o uma opção mais eficiente para aplicações que exigem comunicação de baixa taxa de dados e operações em dispositivos com restrições de bateria, como wearables e sensores (BLUETOOTH SIG, 2024).

3.1.1 Compatibilidade entre versões

Afaneh (2018) explica que as tecnologias Bluetooth Clássico e Bluetooth Low Energy (BLE) apresentam incompatibilidade, impedindo a comunicação direta entre dispositivos de cada tipo. No entanto, a demanda por maior versatilidade na comunicação sem fio impulsionou o desenvolvimento de dispositivos *Dual Mode*, capazes de operar tanto no padrão Clássico quanto no BLE. Smartphones e laptops são exemplos comuns dessa categoria, permitindo a interação com uma gama mais ampla de aparelhos Bluetooth. A Figura 1 ilustra os modelos de dispositivos predominantes em cada uma das classes de tecnologia Bluetooth: Clássico, *Dual Mode* e LE (AFANEH, 2018).

Figura 1 - Tipos de dispositivos Bluetooth



Fonte: AFANEH, 2018.

3.1.2 Métodos de conexão

O Bluetooth Low Energy oferece duas principais modalidades de comunicação: *broadcasting* e conexão ponto a ponto (*peer-to-peer* ou P2P). O *broadcasting* consiste na transmissão unidirecional de dados do dispositivo periférico para o central, sem a necessidade de um vínculo estabelecido. Por outro lado, a conexão ponto a ponto permite a troca bidirecional de informações entre dois dispositivos, estabelecendo um canal de comunicação dedicado (RENESAS, 2023).

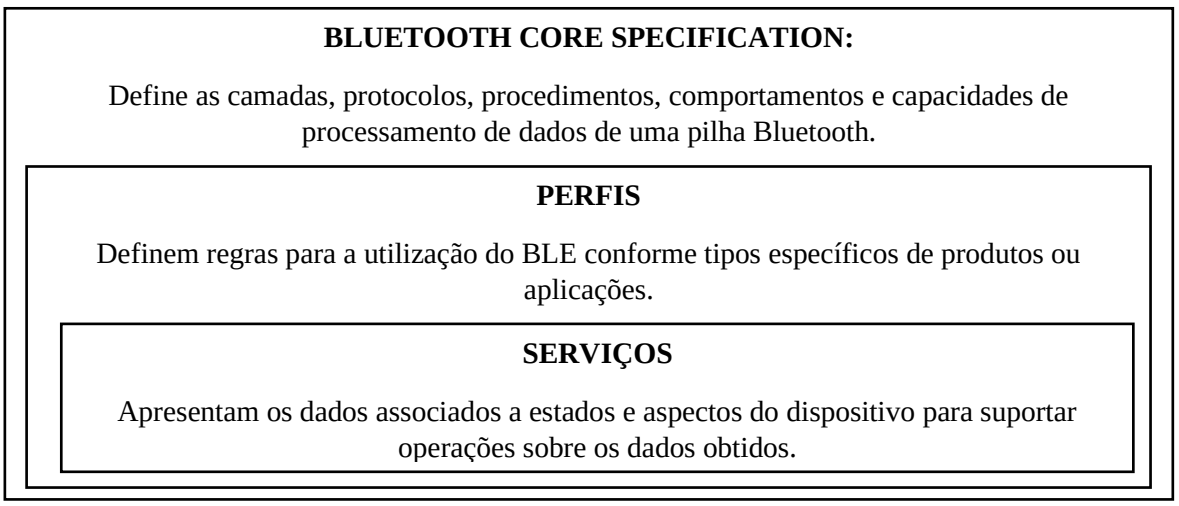
Nesse último caso, o dispositivo central monitora os pacotes de publicidade emitidos pelo periférico desejado. Ao identificar o dispositivo alvo, ele inicia uma solicitação de conexão. Uma vez estabelecida a conexão, ambos os dispositivos podem trocar dados livremente, sem as limitações de taxa de transferência e prioridade impostas pelo broadcasting (RENESAS, 2023).

Além disso, recentemente vem ganhando notoriedade a topologia *Bluetooh Mesh*. Com a criação de redes de múltiplos dispositivos interconectados, ela oferece uma solução para comunicação em larga escala (MOKOBLUE, 2021).

3.1.3 Arquitetura do BLE

A *Bluetooth Core Specification* detalha o funcionamento e a arquitetura do Bluetooth Low Energy. Essa especificação define os padrões e regras para a organização dos perfis e serviços, que determinam como dispositivos de diferentes fabricantes podem interagir de forma padronizada. A Figura 2 apresenta uma visão geral dos principais componentes da especificação BLE e suas relações (BLUETOOTH SIG, 2024).

Figura 2 - Especificações do BLE

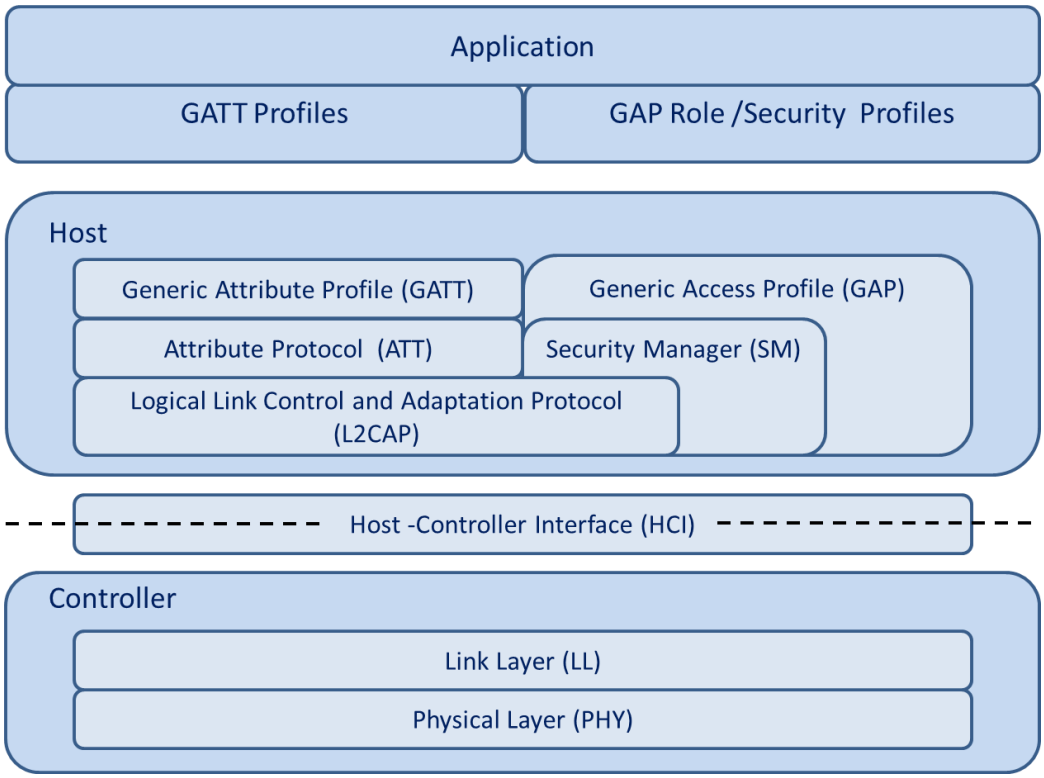


Fonte: Adaptado de BLUETOOTH SIG, 2024.

A arquitetura do BLE é composta por diversas camadas e módulos, que trabalham em conjunto para garantir o funcionamento do dispositivo. A pilha do protocolo BLE é dividida em uma camada de aplicação, um controlador e um *host*, além de uma interface lógica padrão (HCI) que permite a comunicação entre esses componentes (AFANEH, 2018).

A Figura 3 apresenta a arquitetura do BLE considerando as camadas supracitadas.

Figura 3 – Arquitetura do Bluetooth Low Energy



Fonte: RENESAS, 2023.

3.1.3.1 Camada de aplicação (application)

A camada de aplicação (*application*) é responsável pela implementação dos perfis de Acesso Genérico (GAP) e de Atributo Genérico (GATT), que definem as regras para descoberta, conexão e troca de dados entre dispositivos. Essa camada é altamente customizável e varia de acordo com a aplicação específica (RENESAS, 2023).

3.1.3.2 Camadas do host

O *host* abriga as camadas superiores do protocolo, responsáveis pela comunicação e gerenciamento de dispositivos. Os perfis GAP e GATT e o protocolo de Atributos (ATT) definem as regras para descoberta, conexão e troca de dados. O gerenciador de segurança cuida

da autenticação e criptografia. O L2CAP realiza a multiplexação de dados, enquanto a HCI (lado do host) permite a comunicação com o controlador (RENESAS, 2023).

3.1.3.3 Interface Host Controller (HCI)

A interface *Host Controller* (HCI) serve como ponte entre o *host* e o controlador, permitindo que o *host* envie comandos e receba informações do controlador (RENESAS, 2023).

3.1.3.4 Controlador (controller)

O controlador do Bluetooth Low Energy (BLE) é responsável pelas camadas inferiores da pilha de protocolos, gerenciando a comunicação física e o estado do rádio (RENESAS, 2023). Ele é composto por:

- ***Camada física (PHY):*** encarregada da transmissão e recepção dos sinais de rádio, convertendo dados digitais em sinais analógicos e vice-versa.
- ***Camada de link (enlace):*** interface entre a camada física e as camadas superiores, gerenciando o estado do rádio e garantindo a sincronização e a confiabilidade da comunicação. Realiza operações como CRC, geração de números aleatórios e criptografia para garantir a integridade dos dados.
- ***Modo de teste direto:*** utilizado para fins de diagnóstico e depuração do sistema.
- ***Interface Controlador-Host (HCI) — lado do controlador:*** permite a comunicação entre o controlador e o host, permitindo que o host configure e controle o controlador.

3.1.4 Estados de operação do Bluetooth Low Energy

Os dispositivos BLE podem operar em diferentes estados, que podem ser classificados conforme. o detalhamento expresso na Tabela 1 (BLUETOOTH SIG, 2024).

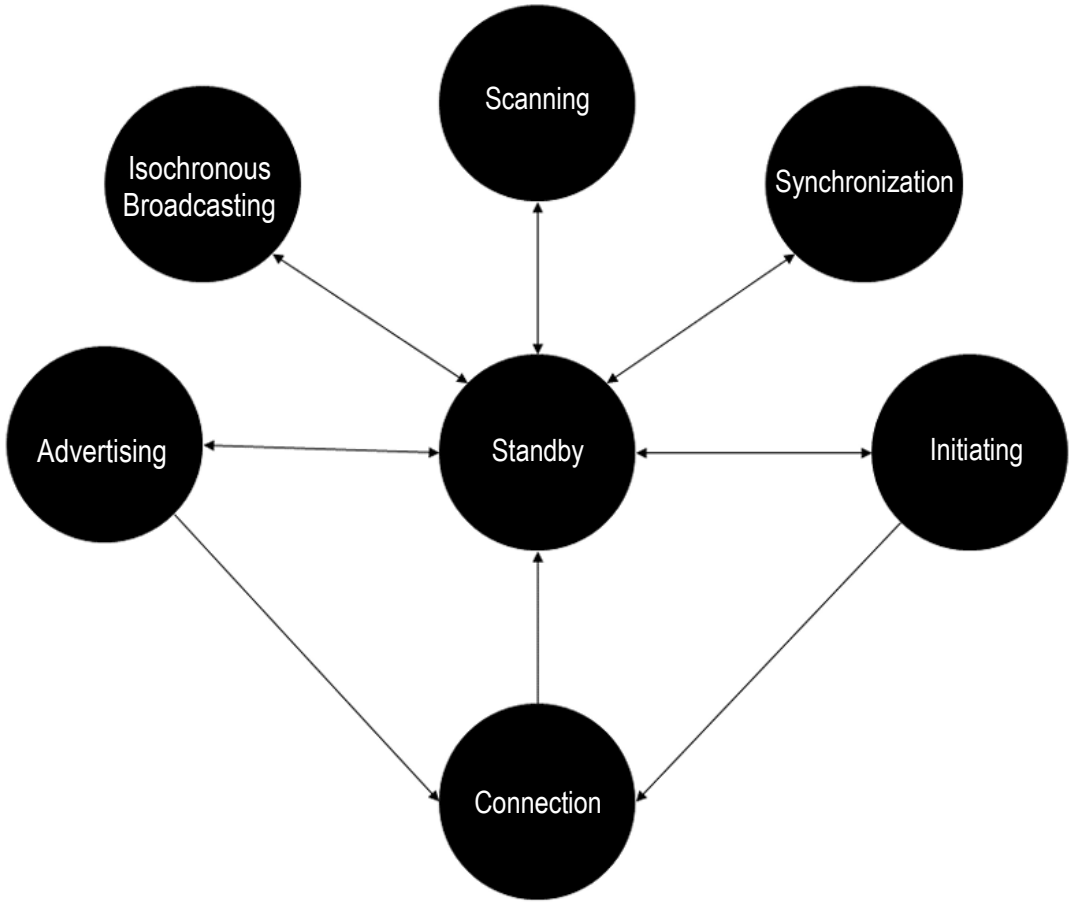
Tabela 1 – Estados de operação de dispositivos BLE

Standby	Dispositivo inativo, sem transmitir nem receber pacotes.
Initiating	Solicita conexão a outro dispositivo.
Advertising	Transmite pacotes de publicidade e pode responder a solicitações de conexão.
Connection	Mantém uma conexão ativa com outro dispositivo.
Scanning	Busca ativamente por outros dispositivos.
Isochronous Broadcast	Transmite dados em tempo real, com taxa de bits constante.
Synchronization	Sincroniza com um dispositivo específico, seguindo uma sequência de publicidade.

Fonte: Autoria própria.

O diagrama de máquina de estados da Figura 4 ilustra as transições entre os diferentes estados de operação da camada de link do Bluetooth LE. Cada estado representa uma condição específica do dispositivo, e as transições são desencadeadas por eventos como o recebimento de um pacote ou o término de uma conexão.

Figura 4 – Diagrama de máquina de estados da camada de enlace.



Fonte: BLUETOOTH SIG, 2024.

Uma característica distintiva do BLE é sua estrutura em pilha, que abrange todas as sete camadas do modelo OSI (Open Systems Interconnection). Essa arquitetura modular permite que o protocolo seja altamente flexível e adaptável a diferentes aplicações (SHIN-TING, 2019). Ao contrário de outros sistemas sem fio que se concentram apenas em algumas camadas, o BLE oferece um conjunto completo de funcionalidades, desde a física até a aplicação, facilitando sua evolução e integração com outros sistemas (BLUETOOTH SIG, 2024).

3.1.5 Anúncio (advertising) e escaneamento (scanning)

O Perfil de Acesso Genérico ou GAP (*Generic Access Profile*) define regras para gerenciar a descoberta de dispositivos e os recursos de segurança associados à camada de controle superior da conexão (RENESAS, 2023). Ele estabelece quatro papéis:

- *Broadcaster:* anuncia sua presença como dispositivo disponível para conexão.
- *Observer:* busca por dispositivos anunciantes.
- *Central:* inicia a conexão com outro dispositivo.
- *Peripheral:* pode atuar como anunciante ou responder a conexões.

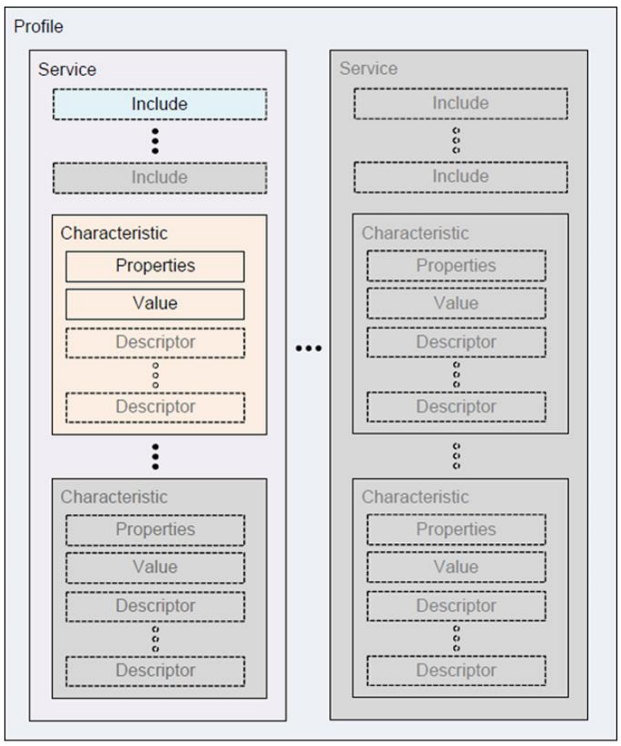
O Perfil de Atributo Genérico ou GATT (*Generic Attribute Profile*) é fundamental para a troca de dados entre dispositivos Bluetooth Low Energy. Enquanto o GAP define como os dispositivos se descobrem e se conectam, o GATT estabelece as regras para a organização e o acesso aos dados (RENESAS, 2023).

Os dois papéis atribuídos pelo GATT para classificar os dispositivos BLE são:

- *Cliente:* inicia a comunicação, solicitando informações ao servidor e recebendo respostas.
- *Servidor:* responde às solicitações do cliente e pode iniciar transmissões de dados.

Os papéis GAP e GATT são independentes, mas compatíveis entre si, permitindo que tanto um dispositivo central quanto um periférico atuem como cliente, servidor, ou ambos simultaneamente (BLUETOOTH SIG, 2024). Os dados são organizados de forma hierárquica em uma estrutura de perfis, serviços e características conforme indicado na Figura 5.

Figura 5 – Estrutura de perfis, serviços e características



Fonte: BLUETOOTH SIG, 2024.

3.1.6 Serviços

Serviços no contexto do Bluetooth Low Energy são agrupamentos de uma ou mais características e atributos relacionados, destinados a atender a uma funcionalidade específica no servidor. Essa estrutura permite que os dispositivos descubram e acessem os dados de forma eficiente e padronizada. Alguns modelos de serviços SIG (*Special Interest Group*) padronizados seriam serviços de leitura de nível da bateria de dispositivo ou serviços de saúde para medição da temperatura corporal ou batimentos cardíacos de um paciente (BLUETOOTH SIG, 2024).

3.1.7 Características

Características são os elementos fundamentais que compõem os serviços no GATT. Cada característica possui um *handle* (identificador numérico de 16 bits), um UUID (*Universally Unique Identifier* ou identificador único, globalmente reconhecido, de 1 28 bits), um valor e permissões que definem como ela pode ser acessada. Opcionalmente, podem ter descritores que fornecem informações adicionais sobre o valor. Por exemplo, em um serviço de bateria, a característica "nível de bateria" teria um UUID específico, um valor numérico indicando a porcentagem de bateria e permissões de leitura (BLUETOOTH SIG, 2024).

A Figura 6 ilustra a estrutura de uma característica.

Figura 6 – Classes de atributos de características.

Fonte: Autoria própria.

Cada atributo possui um *handle* único (16 bits) que serve como endereço para acesso, um UUID (16 ou 128 bits) que identifica o tipo de dado e um valor que armazena a informação. (AFANEH, 2018). Além disso, atributos podem ter permissões que definem as operações permitidas (leitura, escrita) e descritores que fornecem informações adicionais sobre o atributo.

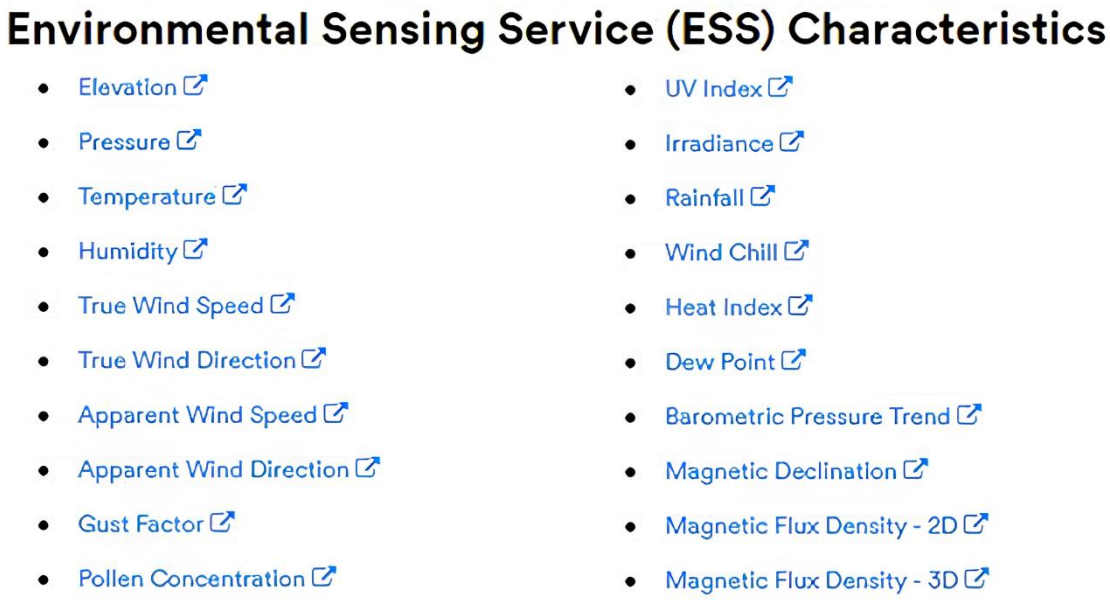
Ao realizar operações de leitura ou escrita, o cliente utiliza o *handle* do atributo para identificar o dado de interesse. O UUID informa o tipo de dado esperado, permitindo que o cliente interprete corretamente o valor retornado. O servidor valida as solicitações e garante a consistência dos dados (BLUETOOTH SIG, 2024).

3.1.8 Bluetooth Core Specification

A *Bluetooth Core Specification* define um conjunto de serviços e características padronizados para garantir a interoperabilidade entre dispositivos de diferentes fabricantes, isto é, que se comuniquem e compartilhem dados de forma eficiente. Esses padrões facilitam o desenvolvimento de diversas aplicações, como monitoramento ambiental, automação residencial e saúde (BLUETOOTH SIG, 2024).

O Environmental Sensing Service (ESS), por exemplo, oferece um conjunto de características para medir variáveis ambientais, como altitude, pressão atmosférica, temperatura e umidade (Figura 7).

Figura 7 – Características do Environmental Sensing Service (ESS)



Fonte: BLUETOOTH SIG, 2024.

3.1.9 Permissões de atributos

As permissões (propriedades) de um atributo GATT referem-se a uma sequência de bits que definem as operações autorizadas sobre ele, garantindo a segurança e o controle do acesso aos dados. Essas permissões são configuradas por bit e determinam se um atributo pode ser lido, escrito, notificado ou transmitido (RANDOM NERD TUTORIALS, 2024).

3.1.9.1 Permissão de leitura (READ)

A permissão de leitura (READ) permite que um cliente leia o valor de um atributo no servidor (por exemplo, a leitura da temperatura por um sensor conectado ao servidor). Em seu

processo de funcionamento, o dispositivo cliente envia uma resposta ao servidor informando que o sinal foi lido corretamente para dar sequência ao processo de comunicação (RANDOM NERD TUTORIALS, 2024).

3.1.9.2 Permissões de escrita (WRITE) e escrita sem resposta (WRITE NO RESPONSE)

As permissões de escrita (WRITE) e escrita sem resposta (WRITE NO RESPONSE) possibilitam a um cliente atribuir ou altear o valor de um atributo no servidor, sendo úteis para atualizar configurações ou enviar comandos para o dispositivo (RANDOM NERD TUTORIALS, 2024).

Enquanto a escrita exige um sinal de resposta enviado pelo servidor para prosseguimento da comunicação, a escrita sem resposta permite o envio dos dados sem que haja a confirmação dos dados. Dessa forma, a propriedade WRITE é preferível em situações que exigem alta confiabilidade dos dados, enquanto a propriedade WRITE NO RESPONSE, de maneira geral, é empregada em casos nos quais a rapidez e a otimização de recursos são prioritárias (WU; MARTIN, 2018).

3.1.9.3 Permissões de notificação (NOTIFY) e indicação (INDICATE)

As permissões de notificação (NOTIFY) e indicação (INDICATE) permitem que o servidor envie atualizações de um atributo sem que o cliente precise solicitar a informação.

NOTIFY é muito utilizada para dados que variam com frequência, como leituras de sensores, em que a latência é importante. Similar à propriedade WRITE NO RESPONSE, a notificação não requer a confirmação do dispositivo receptor da informação (cliente) para permitir a continuidade do processo (RANDOM NERD TUTORIALS, 2024).

INDICATE, por sua vez, requer a confirmação do para garantir que a atualização foi recebida. Isso a torna preferível na exposição de dados críticos que exigem alta confiabilidade, como comandos de controle, sincronização de dados entre dispositivos e confirmação de recebimento de informações (WU; MARTIN, 2018).

3.1.9.4 Permissão de publicidade (BROADCAST)

A permissão de transmissão (BROADCAST) permite que o atributo seja transmitido para todos os dispositivos ao alcance, sem a necessidade de uma conexão direta P2P. É utilizada para transmitir dados de forma contínua e pública, como o nome do dispositivo ou informações de status (RANDOM NERD TUTORIALS, 2024).

3.1.9.5 Segurança dos dados

Os níveis de segurança das propriedades READ, WRITE e WRITE NO RESPONSE podem ser configurados para diferentes graus de acesso, variando entre leitura e escrita com ou sem autenticação e criptografia (AFANEH, 2018). A propriedade NOTIFY não oferece controle direto de segurança, sendo habilitada pelo cliente. Já a INDICATE pode exigir segurança na camada de vínculo, dependendo da configuração do cliente. A propriedade BROADCAST não possui níveis de segurança, pois transmite dados publicamente (WU; MARTIN, 2018).

3.1.10 Descritores

Descritores são atributos que complementam as informações básicas de uma característica (RANDOM NERD TUTORIALS, 2024), fornecendo detalhes como:

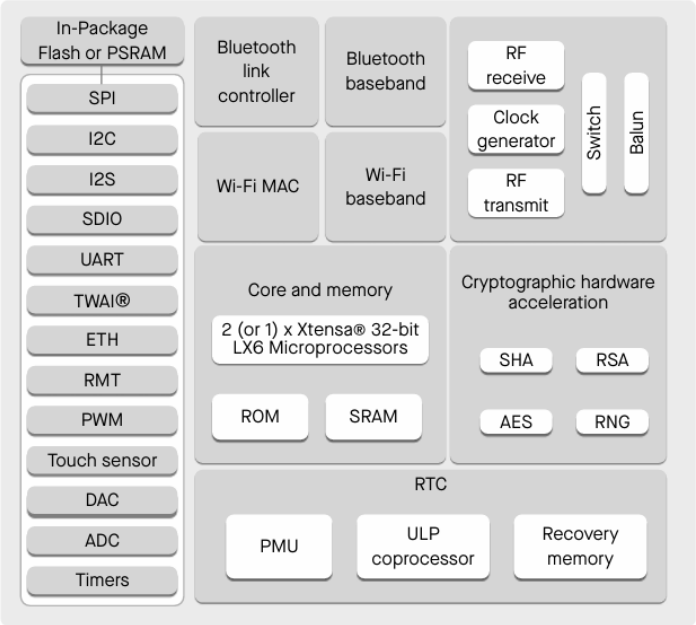
- ***Metadados:*** indicam se o sistema deve gerar notificações ou indicações quando o valor da característica é alterado.
- ***Formato dos dados:*** especificam o tipo de dado (numérico, texto, etc.), as unidades de medida (metros, segundos, etc.) e outros detalhes sobre a representação dos valores.
- ***Descrição:*** oferecem uma explicação clara e concisa sobre o significado da característica, facilitando a compreensão por parte do usuário.

3.2 ESP32-LoRa

ESP32 é uma série de microcontroladores de sistema em chip (SoC) de baixo custo e baixo consumo de energia criado e desenvolvido pela empresa chinesa *Espressif Systems* e fabricado pela TSMC. A família de placas integra as tecnologias Wi-Fi e Bluetooth, surgindo como um sucessor do microcontrolador ESP8266. Esse fato o faz ser amplamente utilizado em projetos de Internet das Coisas (IoT) e que envolvem funcionalidades similares. Inicialmente, a ESP32 foi projetada para oferecer excelente desempenho em termos de energia e rádio frequência, sendo robusta, versátil e confiável para diversas aplicações e cenários de uso (ESPRESSIF SYSTEMS, 2024).

O diagrama de blocos funcionais do ESP32 é mostrado na Figura 8.

Figura 8 – Diagrama em blocos do ESP32



Fonte: ESPRESSIF SYSTEMS, 2024.

Dentre os principais componentes e módulos internos do microcontrolador ESP32 denotam-se: a unidade central de processamento (CPU); os circuitos de memória ROM, SRAM e Flash, módulos de rádio para conectividade Wi-Fi e Bluetooth (com suporte às tecnologias Bluetooth Clássico e Bluetooth Low Energy), periféricos de comunicação (UART, SPI, I2C, I2S), portas de entrada e saída de uso geral configuráveis GPIO, conversores analógicos de sinal (ADC e DAC), temporizadores, módulos para modulação PWM, RTC (para manutenção do tempo de operações de baixo consumo energético), sensores internos (temperatura e capacitivo) e reguladores de tensão (ESPRESSIF SYSTEMS, 2024).

Entre os elementos da família ESP32, a ESP32-LoRa (mostrada na Figura 9) representa uma solução eficaz para aplicações que demandam comunicação remota e eficiente em termos de consumo de energia e potência de comunicação. Seu design compacto e versátil a torna adequada para uma variedade de projetos na área de IoT e sistemas embarcados.

Figura 9 – Módulo ESP32-LoRa com Display Oled (LCD)

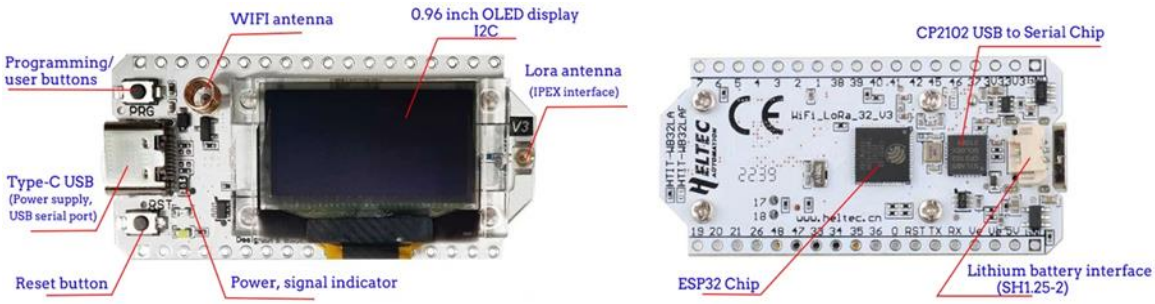
Fonte: AUTOCORE ROBÓTICA, 2024.

Para a implementação do projeto em questão, optou-se pela utilização da placa ESP32-LoRa em função das seguintes características:

- ***Comunicação wireless:*** integração de Bluetooth Low Energy, além de outras funcionalidades wireless.
- ***Pinos de encaixe:*** facilita a conexão com protoboards e outros componentes.
- ***Conectividade:*** suporta comunicação com sensores digitais e analógicos via I2C.
- ***Processamento:*** capacidade de processamento suficiente para diversas aplicações.
- ***LED e botão:*** permitem controle e interação com o usuário.
- ***Display LCD:*** facilita a visualização de dados e informações.

A placa possui o esquemático mostrado na Figura 10 a seguir.

Figura 10 – Esquemático da placa ESP32-LoRa com Display OLED



Fonte: Adaptado de ALIBABA GROUP, 2024.

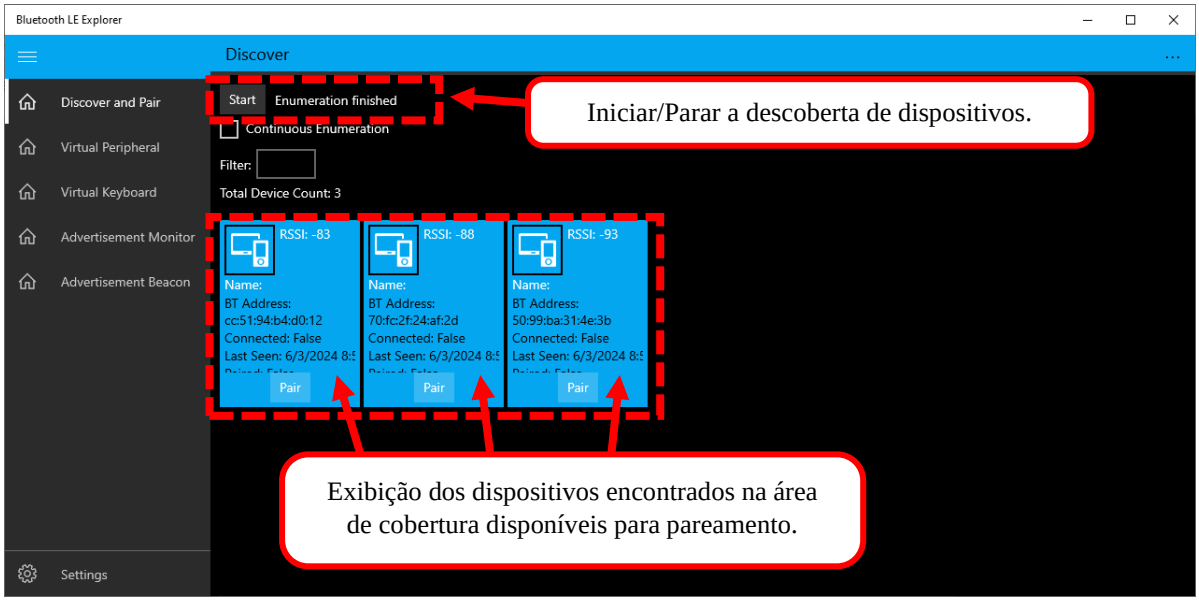
Conforme indicado na Figura 10, a placa é equipada com uma entrada MicroUSB, antena Wi-Fi, um botão de reset e um botão PRG. Além disso, ela conta com um processador ESP32 e um chip CP2102 USB para comunicação serial. Possui também uma interface para bateria de lítio (conector SH1.25-2), um conector para antena LoRa (interface IPEX) e um LED indicador de energia (além do LED interno controlável já citado anteriormente), que sinaliza se a placa está sendo alimentada por alguma fonte de energia.

3.3 Bluetooth LE Explorer

O Bluetooth LE Explorer é uma ferramenta de software desenvolvida pela Microsoft Corporation para dispositivos Windows 10 que suportam BLE. A aplicação permite aos usuários interagir com dispositivos Bluetooth LE próximos, visualizar suas características e enviar

solicitações e comandos para eles. A Figura 11 apresenta a tela inicial do aplicativo com suas funcionalidades básicas.

Figura 11 – Tela de descoberta de dispositivos BLE no aplicativo Bluetooth LE Explorer



Fonte: Autoria própria.

O Bluetooth LE Explorer destaca-se por permitir:

- ***Descoberta e visualização:*** permite encontrar e explorar dispositivos BLE em detalhes.
- ***Interação:*** facilita a comunicação com dispositivos, enviando e recebendo dados.
- ***Monitoramento:*** permite acompanhar a qualidade da conexão e identificar problemas.
- ***Usabilidade***: possui uma interface gráfica intuitiva, facilitando o uso.

3.4 MIT APP Inventor

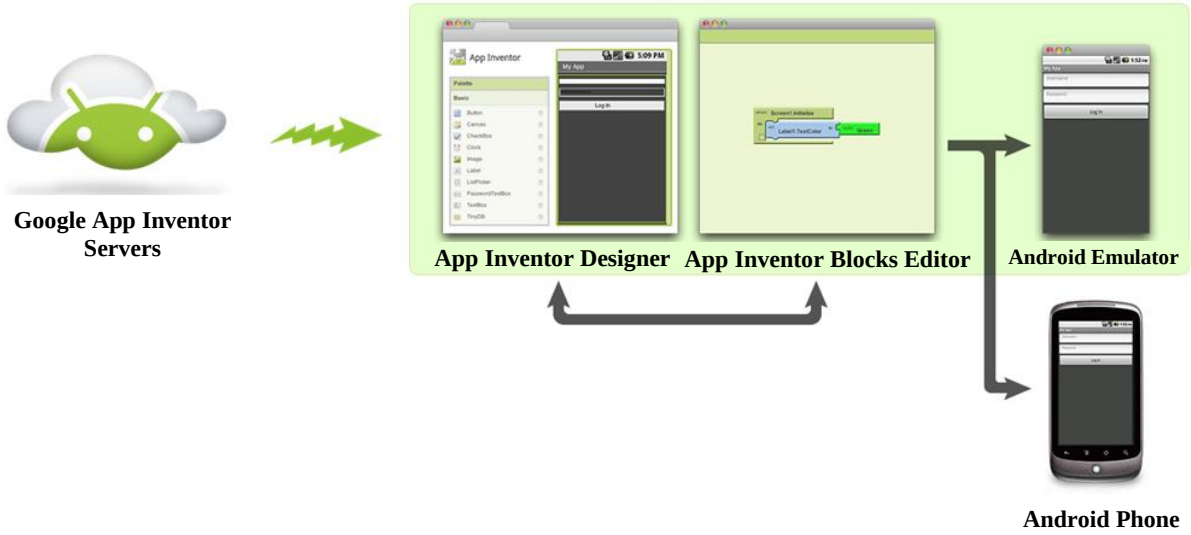
O MIT App Inventor é uma plataforma de código aberto desenvolvida pela Google e mantida pelo Massachusetts Institute of Technology (MIT) que permite a criação de aplicativos para dispositivos Android sem a necessidade de conhecimento prévio em programação. Ele oferece uma abordagem visual baseada no desenvolvimento de algoritmos com blocos, que simplifica e torna mais intuitivo o processo de desenvolvimento de aplicativos (ANDROID PRO, 2024).

A plataforma oferece suporte para uma ampla variedade de componentes e funcionalidades, incluindo conectividade com dispositivos, como sensores internos, GPS e câmera, recursos para conexão wireless (Bluetooth Clássico, Low Energy, Wi-Fi), além de

armazenamento local e serviços da web. Os aplicativos criados podem ser testados instantaneamente em dispositivos Android conectados ou em sistemas de emulação, e podem ser compartilhados via APK ou publicados na Google Play Store e lojas de aplicativos similares (ANDROID PRO, 2024).

A Figura 12 ilustra uma representação esquemática do fluxo de desenvolvimento de aplicativos utilizando o MIT App Inventor.

Figura 12 – Fluxo de desenvolvimento de aplicativos utilizando o MIT APP Inventor



Fonte: adaptado de ANDROID PRO, 2024.

O servidor do MIT App Inventor hospeda a plataforma de desenvolvimento de aplicativos e permite aos usuários acessarem a ferramenta via internet. A primeira etapa no desenvolvimento é realizada no *App Inventor Designer*, por meio da qual os desenvolvedores projetam a interface do usuário do aplicativo. A próxima etapa é no *App Inventor Blocks Editor*, em que os desenvolvedores programam a lógica do aplicativo utilizando blocos de construção visuais (ANDROID PRO, 2024).

Após a definição do design e da lógica, o aplicativo pode ser testado no *Android Emulator*, que simula um dispositivo Android no computador, permitindo a verificação do funcionamento do aplicativo em um ambiente controlado. Por fim, o aplicativo pode ser instalado e executado em um dispositivo Android real (ANDROID PRO, 2024).

4 MATERIAIS E MÉTODOS

Nesta seção, é apresentada a proposta do trabalho, o diagrama geral que correlaciona os materiais utilizados, os métodos empregados para a concepção e implementação do sistema de comunicação Bluetooth Low Energy entre a placa ESP32-LoRa e um dispositivo cliente, bem como as principais regras de funcionamento do sistema elaborado.

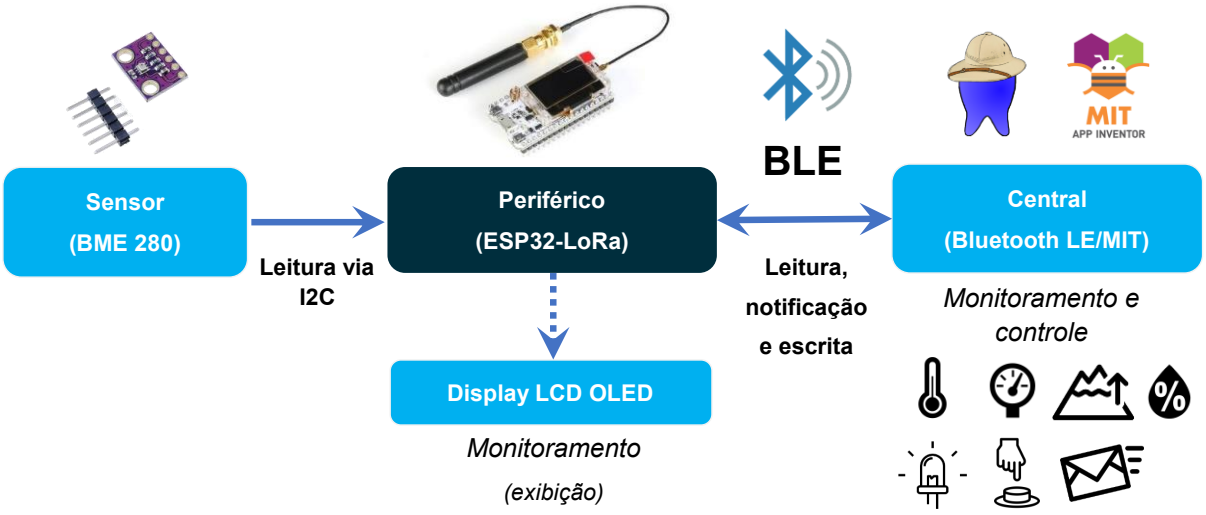
4.1 Proposta do trabalho

A proposta deste projeto é desenvolver um sistema de comunicação bidirecional capaz de transmitir dados coletados de um sensor ambiental conectado a um dispositivo periférico ESP32-LoRa (medindo temperatura, pressão, altitude e umidade) e de atuar sobre o botão PRG da mesma. Além disso, ele deve enviar comandos para controlar o estado do LED interno da placa ESP32 e enviar mensagens de texto para o dispositivo periférico utilizando o Bluetooth LE Explorer (Desktop) ou uma aplicação própria desenvolvida para smartphone na plataforma MIT App Inventor.

4.1.1 Diagrama geral

O diagrama geral que representa o sistema elaborado é apresentado na Figura 13.

Figura 13 – Diagrama esquemático do sistema com relações de leitura e escrita.



Fonte: Autoria própria.

4.1.2 Materiais utilizados

A seguir, são descritos os elementos que fazem parte do sistema apresentado na Figura 13.

- ***Sensor BME 280:*** dispositivo utilizado para medição da umidade relativa, temperatura e pressão barométrica.

- ***Placa ESP32-LoRa:*** integra o sensor BME 280 via comunicação I2C e dispõe de um display módulo LCD, permitindo a visualização dos dados coletados, bem como dados enviados pelo dispositivo central através de comunicação BLE.

- ***Firmware:*** o firmware utilizado para a programação da ESP32 foi desenvolvido para permitir o envio de dados do sensor e a recepção de comandos de um dispositivo central. Utilizou-se a IDE do Arduino (Arduino Integrated Development Environment) com a linguagem C devido a sua capacidade de suporte a múltiplas bibliotecas e componentes BLE e à facilidade de integração com o sensor BME e a placa ESP32-LoRa. Ele foi estruturado para implementar o GATT, com a definição do perfil, serviços, características, descritores e propriedades habilitadas para cada caso.

Foram utilizadas bibliotecas compatíveis com a IDE do Arduino para permitir a integração dos dispositivos. As bibliotecas utilizadas estão listadas na Tabela 2.

Tabela 2 – Bibliotecas utilizadas e suas versões

Biblioteca	**Versão**
Adafruit_BME280 Library	2.2.4
Adafruit BusIO	1.15.0
Adafruit GFX Library	1.11.9
Adafruit SSD1306	2.5.9
Adafruit SSD1306 EMULATOR	0.1.0
Adafruit Unified Sensor	1.1.14
ESP8266 and ESP32 OLED driver for SSD1306 displays by ThingPulse	4.4.1
LiquidCrystal	1.0.7
ESP32 by Espressif Systems	2.0.11

Fonte: Autoria própria.

- **Dispositivo central:** os testes de integração foram realizados utilizando um notebook com sistema operacional Windows 10, equipado com capacidade de transmissão Bluetooth, versão do driver 10.0.19041.4412. Isso permitiu o uso do software Bluetooth

LE Explorer. Além disso, foi criado um aplicativo para dispositivos smartphones Android utilizando a plataforma MIT App Inventor.

A principal diferença na utilização do Bluetooth LE e do MIT APP Inventor e é que a primeira permite a exposição de uma vasta gama de elementos envolvidos no GATT (informações de potência de sinal de transmissão, RSSI, descritores, UUIDs dos serviços e características, entre outros), enquanto a segunda oferece uma exibição menos detalhada (não permite a exibição de descritores e UUIDs, entre outros).

4.2 Estrutura GATT

Para implementar a comunicação entre os dispositivos, foi adotado o perfil GATT, utilizando os serviços Automation IO e Environmental Sensing. O serviço Automation IO foi utilizado para controlar o LED interno da placa ESP32-LoRa, monitorar o estado do botão PRG e enviar mensagens de texto. Já o serviço Environmental Sensing, como mencionado anteriormente, foi responsável pela transmissão dos dados do sensor BME280.

4.2.1 Serviço Automation IO

O serviço Automation IO, definido na *Bluetooth Core Specification*, promove a interoperabilidade entre dispositivos que utilizam soluções de IoT para o envio e recebimento de sinais digitais (como sensores digitais) ou cadeias de dados (numéricos, textos, etc.). Esse serviço foi utilizado para implementar características que envolvem a ativação e o desligamento do LED interno da placa ESP32, o envio de mensagens de texto do dispositivo central para o periférico e o monitoramento do estado do botão PRG (simulando a participação de um sensor digital, como um sensor de presença ou toque capacitivo).

4.2.1.1 Característica "LED"

Para a operação de controle do estado ON/OFF do LED interno da placa ESP32-LoRa, foi adicionada uma característica com as propriedades WRITE e READ. A propriedade de escrita foi utilizada para permitir a alteração do valor de uma variável do tipo char associada aos níveis lógicos alto e baixo no firmware, enquanto READ foi utilizada para possibilitar atualizações do status em requisições de leitura do dispositivo cliente.

O UUID associado à característica LED foi 0x2A57, estabelecido na documentação oficial do Bluetooth como identificador padrão de características de saída digital, destinadas a dispositivos que podem controlar o estado de um sinal digital, como uma lâmpada ou LED.

4.2.1.2 Característica "Status do Botão"

Para monitorar o estado do botão PRG da placa ESP32-LoRa, foi adicionada uma característica com as propriedades READ e NOTIFY. A propriedade READ foi usada para possibilitar atualizações do status em requisições de leitura do dispositivo cliente e NOTIFY para transmissão automática das modificações realizadas no status do componente.

O UUID associado à característica Status do Botão foi 0x2A56, utilizado para identificar características de sinais e estados digitais no geral (como ligado/desligado ou 0/1), sendo comumente utilizada em elementos que possuem estados binários simples, como interruptores ou botões.

4.2.1.3 Característica "MSG"

A característica "MSG" foi criada para fazer o envio de mensagens de texto escritas e processadas em um dispositivo cliente para a placa ESP32-LoRa. Com o intuito de permitir a sua escrita e a leitura (quanto a atualização do valor desta for solicitada pelo usuário), utilizaram-se as propriedades WRITE e READ.

O UUID associado ao envio das mensagens foi o 0x2A22, padrão da Característica BootKeyboardInputReport. Características dessa classe são originalmente projetadas para dispositivos como teclados e controles, que necessitam enviar estados de tecla, como em computadores ou smartphones. Para o caso em questão, utilizou-se a característica para fazer referência ao envio de cadeias de texto.

4.2.2 Serviço Environmental Sensing (ESS)

O serviço Environmental Sensing foi utilizado para o gerenciamento das características ambientais de temperatura, pressão, altitude e umidade. As características foram definidas de acordo com os padrões estabelecidos na *Bluetooth Core Specification*, permitindo que os nomes de cada uma sejam associados automaticamente na visualização do Bluetooth LE Explorer.

Para ambas as características do ESS, foram utilizadas as propriedades READ, com o intuito de permitir que o dispositivo cliente obtenha a atualização dos valores de temperatura, umidade, pressão e altitude nos momentos de requisições de leitura do usuário, e NOTIFY a fim de possibilitar a transmissão automática das modificações realizadas no valor das variáveis conforme intervalo programado no firmware para o monitoramento.

Os UUIDs vinculados a essas características estão expressos na Tabela 3.

Tabela 3 – UUIDs e descritores vinculados às características do ESS

Característica	**UUID**
Temperature Characteristic	0x2A6E
Humity Characteristic	0x2A6F
Pressure Characteristic	0x2A6D
Elevation Characteristic	0x2AB3

Fonte: Autoria própria.

4.2.3 Descritores genéricos

Foram inseridos descritores genéricos de texto (cujo UUID padrão é 0x2901) para apresentar informações pertinentes às definições dos elementos ou às unidades de medida das variáveis mensuradas. Dessa forma, eles auxiliam a clarificar a funcionalidade dos mecanismos atuadores e as leituras do sensor para os usuários, proporcionando uma melhor compreensão dos dados apresentados.

4.3 Diagrama da Estrutura do Perfil GATT

Nos APÊNDICE A e APÊNDICE B, são apresentados o diagrama da estrutura do perfil GATT com os serviços, características correspondentes e descritores associados às funcionalidades do Environmental Sensing e Automation IO respectivamente. Para cada serviço e característica, são descritas as propriedades e métodos utilizados para a passagem e interpretação dos sinais.

4.3.1 Regras de funcionamento do sistema

O sistema consiste em uma placa ESP32-LoRa conectada ao sensor BME280 por meio dos pinos Vcc, GND, 22, e 21, correspondentes às portas Vcc, GND, SCL, e SDA do sensor, respectivamente. O sensor BME280 transmite continuamente os dados coletados para a ESP32-LoRa, que os converte em valores de temperatura, pressão, umidade e altitude (esta última calculada com base na pressão local definida no código na Arduino IDE).

Ao ser energizada, a ESP32-LoRa inicializa, configurando perfis, serviços e características. Em seguida, o dispositivo verifica as conexões I2C e entra em modo de espera (*advertising*), anunciando-se para possíveis conexões. Quando um dispositivo cliente estabelece conexão P2P com a ESP32, esta inicia um ciclo contínuo de leitura dos dados do sensor BME280, enviando-os ao cliente através da propriedade NOTIFY. A cada 3 segundos,

uma das características do Environmental Sensing Service (ESS) é atualizada e o valor correspondente é exibido no display LCD.

Se o cliente enviar uma solicitação READ, os valores das características são imediatamente atualizados na interface do cliente, sem alterar a sequência de exibição no display LCD, que continua mostrando o valor mais recente de cada característica em sua vez.

Ao receber uma solicitação WRITE para alterar o estado do LED interno da ESP32 (ON/OFF), o valor é instantaneamente atualizado na tela do cliente e exibido no display LCD por três segundos, após o que a exibição das características do ESS é retomada.

Pressionar o botão PRG da ESP32-LoRa interrompe a exibição atual no display para mostrar o novo estado do botão por três segundos, a menos que uma atualização do LED ou uma mensagem de texto intervenha. Depois, a exibição das características do ESS continua a partir da próxima.

O dispositivo cliente também pode enviar mensagens de texto para a ESP32 usando a propriedade WRITE. Quando uma mensagem é recebida, a exibição da característica atual é interrompida e a mensagem é exibida no display OLED por três segundos, salvo interrupções por atualizações do LED ou do botão PRG. Após esse tempo, a exibição da próxima característica do ESS é retomada. A mensagem também é exibida na tela do dispositivo cliente no campo reservado para ela.

5 DESENVOLVIMENTO

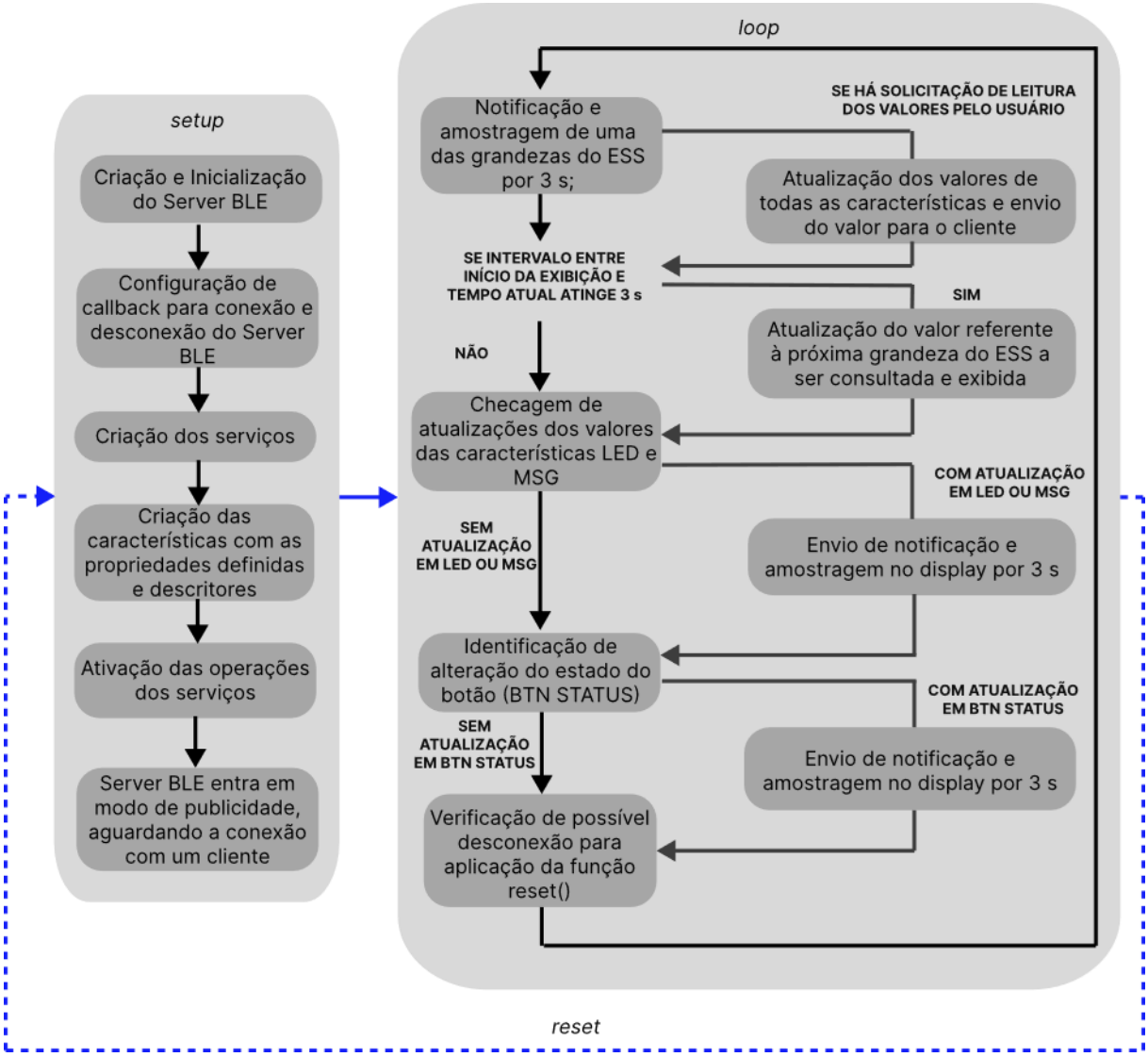
Esta seção tem como objetivo descrever as configurações realizadas nos dispositivos central e periférico para o estabelecimento da comunicação BLE.

5.1 Configurações do dispositivo servidor

5.1.1 Fluxo de execução

Nesta subseção, é apresentado o fluxograma das principais funções implementadas no firmware do ESP32-LoRa para viabilizar a comunicação BLE com um dispositivo cliente (Figura 14). O fluxograma ilustra as etapas executadas desde a inicialização do servidor BLE até o monitoramento e a notificação das grandezas do Environmental Sensing Service (ESS) e Automation IO Service (AIO), além do tratamento de alterações nos valores das características e a possibilidade de reinício do código (função *reset*).

Figura 14 – Diagrama representando a lógica de programação implementada no firmware



Fonte: Autoria própria.

5.1.2 Importação das bibliotecas no firmware da Arduino IDE

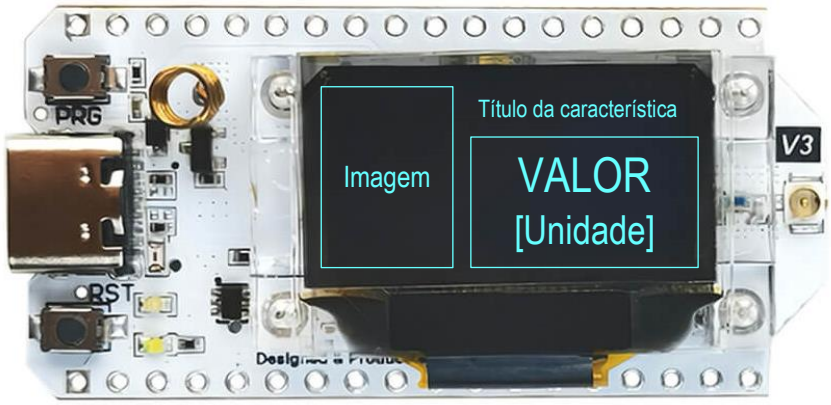
No firmware, fez-se a importação de bibliotecas referentes à comunicação I2C, à captura dos sinais das portas do sensor BME 280, ao driver do display de LCD OLED, das bibliotecas para utilização do protocolo BLE e emprego de descritores (*BLEDevice* e *BLE2902*). Essas inicializações são apresentadas no APÊNDICE C.

5.1.3 Exibição dos dados no Display OLED LCD

Para indicar as características mensuradas e referidos valores na tela do display OLED, foi utilizado um layout padrão (Figura 15), consistindo em uma ilustração representativa da

característica a ser exibida na parte esquerda do display, e, na parte direita, o nome e o valor da grandeza mensurada (ou elemento estado está sendo informado, nos casos do botão e do LED ou mensagem, para fazer referência ao texto digitado e enviado pelo usuário).

Figura 15 – Diagrama esquemático do sistema com relações de leitura e escrita



Fonte: Adaptado de SMART KITS, 2024.

Os arquivos de frames utilizados são bibliotecas com conteúdo na estrutura de XBM, originados a partir de imagens bitmap monocromático de 16 bits, que contém uma estrutura de distribuição de cadeia de pixels que pode ser lida e convertida para apresentação no display LCD. A Tabela 4 apresenta a imagem e as informações exibidas para cada um dos elementos na tela do display OLED. Os vetores foram convertidos para arquivos de frame e colocados no diretório do arquivo (código do Arduino) de modo que possam ser referenciados como bibliotecas .h conforme mostrado no APÊNDICE D.

Tabela 4 – Estrutura de exibição de cada característica no display OLED LCD

Imagem:	**Informações:**
	**Título:** Temperatura: **Valor:** XX.XX °C
	**Título:** Pressão: **Valor:** XXX.XX hPa

	**Título:** Altitude: **Valor:** XXX.XX m
	**Título:** Umidade: **Valor:** XX %
	**Título:** LED: **Valor:** *ON / OFF*
	**Título:** Botão: **Valor:** *ON / OFF*
	**Título:** Mensagem: **Valor:** *<Texto>*

Fonte: Autoria própria.

5.1.4 Definições de pinos e variáveis globais utilizadas

Definiram-se variáveis para referência aos pinos e componentes internos da placa ESP32-LoRa. Ao led controlável interno, cuja pinagem é 25, foi atribuída a variável ONBOARD_LED. Igualmente, para o botão PRG da placa, cujo pino é 0, definiu-se uma variável, btn_PRG.

Foi realizada a definição dos pinos para a conexão da placa ESP32-LoRa com o sensor BME280 (o pino 21 da ESP32 foi ligado ao SDA do sensor e a variável utilizada no código recebeu o nome I2C_SDA. O pino 22 da placa foi ligado ao SCL do sensor BME 280 e a variável para se trabalhar com essas operações foi denominada I2C_SCL). Foram também feitas as inicializações de *TwoWire I2C_BME = TwoWire(1)* e *Adafruit_BME280 BME*. A sentença *TwoWire I2C_BME = TwoWire(1)* configura o segundo controlador I2C do ESP32, permitindo a comunicação com dispositivos conectados ao barramento I2C1. *Adafruit_BME280 BME* cria uma instância do objeto Adafruit_BME280, permitindo a comunicação com o sensor BME280 para obter leituras de temperatura, pressão e umidade. A Figura 16 indica essas configurações.

Figura 16 – Configuração dos pinos e entradas do sensor BME e de componentes da ESP32

```
35 //PLACA ESP32-LORA:
36 #define ONBOARD_LED 25  // (25 é a pinagem do led interno da
   ESP32_LoRa, usada nesse projeto)
37 #define btn_PRG 0       // (0 é a pinagem referente ao botão PRG)
38
39 //PINOS DO SENSOR BME-280:
40 #define I2C_SDA 21
41 #define I2C_SCL 22
42 TwoWire I2CBME = TwoWire(1);
43 Adafruit_BME280 bme;
```

Fonte: Autoria própria.

Além disso, como mostrado no APÊNDICE E, foi realizada a configuração do display LCD. Esse processo envolveu a definição de parâmetros importantes, como o endereço I2C do display, definido como 0x3C, o pino de reset configurado no pino 16 do microcontrolador, o pino de dados SDA configurado no pino 4, e o pino de *clock* SCL no pino 15. Para a inicialização e controle do display, foi utilizada a função *SSD1306 display(Addr, SDA, SCL)*, pertencente à biblioteca específica para displays LCD SSD1306, que permite a comunicação adequada entre o microcontrolador e o display.

A Figura 17 apresenta as definições do nome do servidor (*"ESP32_AIO_BME280"*), nomes de referência para os UUIDs dos serviços BLE (*SERVICE_UUID_AIO* para o Automation IO e *SERVICE_UUID_BME* para o Environmental Sensing Service), para os UUIDs das características BLE e UUIDs dos descritores, que foi utilizado para permitir a exibição das descrições das características no aplicativo Bluetooth LE Explorer.

Figura 17 – Definição dos identificadores dos elementos do protocolo BLE

```
52 #define ServerName "ESP32_AIO_BME280"
53 #define SERVICE_UUID_AIO (BLEUUID((uint16_t)0x1815))
54 #define SERVICE_UUID_BME (BLEUUID((uint16_t)0x181a))
55 #define CHARACTERISTIC_UUID_LED (BLEUUID((uint16_t)0x2a57))
56 #define CHARACTERISTIC_UUID_BTN_STATUS (BLEUUID((uint16_t)0x2a56))
57 #define CHARACTERISTIC_UUID_MSG (BLEUUID((uint16_t)0x2a22))
58 #define CHARACTERISTIC_UUID_Temp (BLEUUID((uint16_t)0x2a6e))
59 #define CHARACTERISTIC_UUID_Pre (BLEUUID((uint16_t)0x2a6d))
60 #define CHARACTERISTIC_UUID_Alt (BLEUUID((uint16_t)0x2ab3))
61 #define CHARACTERISTIC_UUID_Umi (BLEUUID((uint16_t)0x2a6f))
64 #define DESCRIPTOR_UUID (BLEUUID((uint16_t)0x2901))
```

Fonte: Autoria própria.

Por fim, foi configurada a pressão ao nível do mar para a obtenção do valor da altitude local, como 1014,5 hPa conforme pode ser visto na Figura 18.

Figura 18 – Definição da pressão atmosférica ao nível do mar

```
66 //Pressão atmosférica ao nível do mar
67 #define SEALEVELPRESSURE_HPA (1014.5)
```

Fonte: Autoria própria.

5.1.5 Definições dos ponteiros do BLE

Foi implementado o modelo de comunicação BLE via conexão P2P para permitir que o dispositivo anuncie sua presença e seja detectado por outros dispositivos BLE. Inicialmente, o elemento do servidor foi definido como pServer = NULL, uma prática recomendada para aumentar a segurança do código ao evitar erros. A criação do servidor, assim como a definição das características e descritores, ocorre exclusivamente na função void setup().

As características utilizadas também foram inicializadas com NULL, seguindo a mesma lógica de segurança. O mesmo procedimento foi aplicado aos descritores associados a cada uma delas. A Tabela 5 apresenta as funções utilizadas.

Tabela 5 – Funções utilizadas para implementação da ESP32 como servidor BLE

Ponteiro	**Uso**
BLEServer*	*função para definir um dispositivo servidor*
BLECharacteristic*	*função para definir um elemento característica*
BLEDescriptor*	*função para definir um elemento descritor*

Fonte: Autoria própria.

No APÊNDICE F, são apresentados os ponteiros declarados para atribuição de parâmetros ao dispositivo, características e descritores.

5.1.6 Declaração das variáveis utilizadas ao longo do projeto

Primeiramente, foram declaradas as variáveis para calcular o tempo em que os valores são atualizados na visualização do LCD e notificados ao cliente. A função *millis()* é utilizada para a contagem do tempo, retornando o número de milissegundos desde que o Arduino começou a executar o programa atual. Já a variável *lastMsg* é utilizada para programação dos

intervalos de amostragens e atualizações de cada característica no display LCD e dispositivo cliente. A Figura 19 apresenta essas declarações.

Figura 19 – Declaração das variáveis de cálculo de tempo

```
108 unsigned long now = millis();
109 unsigned long lastMsg = 0; //instante de tempo inicial, definido
    como zero para o início da contagem
```

Fonte: Autoria própria.

Em seguida, conforme o APÊNDICE G, são definidas variáveis para armazenar os valores do botão (*btn_signal*), LED (*led_signal*), mensagem enviada pelo usuário (*msg*), temperatura (*msg_temp*), pressão (*msg_pre*), altitude (*msg_alt*) e umidade (*msg_umi*). Estas variáveis permitem monitorar e controlar o estado do sistema no display OLED.

Para a visualização das informações no display LCD, foram declaradas variáveis que definem as imagens na tela. Estas variáveis, como apresentado no APÊNDICE H, configuram a posição (*pos_x* e *pos_y*), largura (*frame_width*), altura (*frame_height*), cadeia de bit (*frame_bits*) da imagem, posição horizontal do título e do valor da característica, respectivamente, *pos_x_titulo* e *pos_x_valor*.

Por fim, é definida uma estrutura (*struct*) para o recebimento simultâneo dos valores obtidos do sensor BME280, que mede temperatura, pressão, altitude e umidade. Esta estrutura é indicada na Figura 20.

Figura 20 – Declaração da estrutura para armazenar valores das variáveis do ESS

```
144 //Estrutura (struct) para recebimento simultâneo dos valores ob-
    tidos do sensor BME280
145 struct BME_Dados{
146   float temperatura;
147   float pressao;
148   float altitude;
149   float umidade;
150 };
```

Fonte: Autoria própria.

5.1.7 Inicialização do sensor BME e display LCD

Por meio da função *inicializacao_bme_and_display*, apresentada no APÊNDICE I, realiza-se a inicialização do sensor BME280 e do display LCD, de modo que sejam verificadas as conexões antes da liberação da transferência de dados e exibidas as informações de status no

monitor serial e no próprio display. Inicialmente, é definida a porta do display LCD (OLED_RESET) como terminal de saída. Em seguida, é realizado um reset inicial do display, apagando qualquer sinal presente para garantir que a tela esteja limpa ao iniciar a aplicação. Após o *reset*, o display é inicializado, configurando-se a fonte de texto padrão e o alinhamento, como indicado no APÊNDICE J. Caso a inicialização do display falhe, é exibida a mensagem "Display indisponível!" no monitor serial. Se bem-sucedida, a mensagem "Display OLED – OK" é mostrada tanto na tela do display como no monitor serial.

Em seguida, as portas I2C são configuradas para a comunicação com o sensor BME280. Como pode ser visto na Figura 21, a função *bme.begin* é usada para iniciar o sensor na porta especificada. Se falhar, é exibida a mensagem "Sensor BME280 - Não OK" no display e no monitor serial, e o programa entra em um loop infinito para forçar o usuário a verificar as conexões. Caso a inicialização do sensor seja bem-sucedida, a mensagem "Sensor BME 280 – OK" é mostrada.

Figura 21 – Inicialização da comunicação I2C do sensor BME 280 e configurações das mensagens de log para confirmação da procedência do processo

```
191   // se definindo portas I2C
192   I2CBME.begin(I2C_SDA, I2C_SCL, 100000);
193   bool status;
194   status = bme.begin(0x76, &I2CBME);  // se definindo portas I2C
195   if (!status) {
196     Serial.println("Sensor BME280 - Não OK");
197     display.drawString(0, 16, "Sensor BME280 - Não OK"); //posição x
    na tela do display = 0; posição y na tela do display = 16; texto =
    "Sensor BME280 - Não OK"
198     display.display();
199     vTaskDelay(1000 / portTICK_PERIOD_MS);
200     while (1);  //Se BME não disponível, gera-se um loop infinito para
    barrar o fluxo mantendo a exibição da mensagem de que é necessário ve-
    rificar as conexões dos cabos que ligam os sensores e pressionar o RE-
    SET para tentar de novo
201   } else {
202     Serial.println("Sensor BME280 - OK");
203     display.drawString(0, 16, "Sensor BME280 - OK"); //posição x na
    tela do display = 0; posição y na tela do display = 16; texto = "Sen-
    sor BME280 - OK"
204     display.display();
205     vTaskDelay(500 / portTICK_PERIOD_MS);
```

Fonte: Autoria própria.

5.1.8 Mostrar dados das características no display LCD

A função *mostrar_dados_display_lcd* foi criada para configurar e exibir imagens e textos no display LCD, referentes às diferentes características monitoradas pelo sistema. Esta função, apresentada no APÊNDICE K, recebe como parâmetros as posições, dimensões e a cadeia de bits da imagem, além dos textos a serem exibidos e suas respectivas posições.

Primeiramente, a função limpa o display, carregando o buffer com os dados da imagem monocromática que será exibida. As posições iniciais *pos_x* e *pos_y*, a largura, a altura e a cadeia de bits da imagem são passadas como argumentos, bem como o título do elemento, seu valor e a variável *long_text* (usada para diferir o tamanho da fonte para textos longos e sequências de caracteres curtas, de forma a otimizar a visualização no display).

Em seguida, a fonte do texto é definida para ArialMT_Plain de tamanho 10, e o nome da grandeza é desenhado na tela na posição especificada por meio da função *display.drawString*. Para a exibição do valor e unidade da característica, caso o texto não seja longo, a fonte do valor da grandeza é alterada para ArialMT_Plain de tamanho 16 e desenhado na posição correspondente com a função *display()*. O APÊNDICE L demonstra essa sequência de comandos.

5.1.9 Função Reset

A função *funcReset*, apresentada na Figura 22 , reinicia o sistema quando chamada com o valor zero. Ao ser executada, a CPU retorna ao início do código, permitindo que o dispositivo cliente reinicie o processo de descoberta.

Figura 22 – Função *(*funcReset)()*

```
249   void (*funcReset)() = 0;
```

Fonte: Autoria própria.

5.1.10 Função print_LCD_and_APP

A função *print_LCD_and_APP* foi declarada para permitir a configuração e exibição de informações das características do Environmental Sensing no display LCD e, ao mesmo tempo, notificar o aplicativo cliente via BLE (Bluetooth Low Energy). A função recebe dois parâmetros: *type_var*, que varia conforme a característica da sequência pré-definida que deve ser atualizada e exibida, e *valor*, que é uma estrutura contendo o valor da respectiva

característica do ESS. Primeiramente, são declaradas variáveis locais para armazenar os valores das características ambientais, conforme mostrado no APÊNDICE M.

Em seguida, a instrução *switch case* é utilizada para determinar qual característica será processada, com base no valor de *type_var*. Para cada caso, o valor correspondente é extraído da estrutura *valor* e convertido para *string* com a função *dtostrf*. Os textos de *título* e *valor* são configurados, assim como as propriedades do display e a notificação via BLE é habilitada.

Na Figura 23, é mostrada a estrutura criada para definir a exibição e atualizar o valor da característica “temperatura” no display.

Figura 23 – Trecho da estrutura *switch case* para envio do valor das características do ESS e configuração dos parâmetros de visualização no display

```
263   switch (type_var){
264     case 1:
265       temp = valor.temperatura;
266       dtostrf(temp, 4, 2, msg_temp);
267       var_titulo = String("Temperatura: ");
268       var_valor = String(temp) + ("°C");
          pCharacteristic_Temp->notify();  //envio via BLE habilitado
269 pela propriedade NOTIFY
270       frame_width    =   frame_temperatura_width;
271       frame_height   =   frame_temperatura_height;
272       frame_bits     =   frame_temperatura_bits;
273       pos_x_titulo = 64;
274       pos_x_valor = 64;
275       break;
```

Fonte: Autoria própria.

Para as demais características, segue-se o mesmo procedimento alternando os valores conforme indicado na Tabela 6.

Tabela 6 – Valores das variáveis para cada característica

Variável	Temperatura	Pressão	Altitude	Umidade
type_var	1	2	3	4
titulo	"Temperatura: "	"Pressão: "	"Altitude: "	"Umidade: "
valor	String(temp) + ("°C")	String(pres) + (" hPa")	String(alt) + (" m")	String(umi) + ("%")
pos_x_titulo	64	61	64	70
pos_x_valor	64	43	58	64

Fonte: Autoria própria.

Após a definição dos parâmetros específicos para cada característica, o valor da grandeza é impresso no monitor serial. Em seguida, a função *mostrar_dados_display_lcd*, apresentada na Figura 24, é chamada para atualizar o display com os dados configurados.

Figura 24 – Chamada para a função *mostrar_dados_display_lcd*

```
314   Serial.println(var_valor);
316   mostrar_dados_display_lcd(pos_x, pos_y, pos_x_titulo, pos_x_valor,
    frame_width, frame_height, frame_bits, var_titulo, var_valor);
317 }
```

Fonte: Autoria própria.

5.1.11 Classe MyServerCallbacks

A classe *MyServerCallbacks* é responsável por configurar as funções de retorno de chamada (*callback*) para os momentos nos quais o dispositivo BLE se conecta (*onConnect*) ou desconecta (*onDisconnect*). Essas funções são essenciais para gerenciar o estado de conexão do dispositivo Bluetooth com o ESP32. São declaradas duas variáveis booleanas (*deviceConnected* e *temporary_connected*) como ilustrado na Figura 25. A variável *deviceConnected* é utilizada para verificar se um dispositivo Bluetooth está conectado ao ESP32, enquanto *temporary_connected* é usada para validar se o dispositivo já se conectou e desconectou ou está aguardando a primeira conexão. *temporary_connected* é útil para liberar a execução da função *Reset* somente após a perda de conexão (e não no início do código).

Figura 25 – Declaração das variáveis para o estado de conexão

```
333 bool deviceConnected = false;
334 bool temporary_connected = false;
```

Fonte: Autoria própria.

A classe *MyServerCallbacks* é uma classe pública do modelo *BLEServerCallbacks* que implementa as funções *onConnect* e *onDisconnect*. Quando um dispositivo cliente se conecta ao servidor, a função *onConnect* é chamada. Esta função altera as variáveis *deviceConnected* e *temporary_connected* para *true* e imprime uma mensagem no monitor serial indicando que o dispositivo cliente está conectado. Já quando ocorre um evento de desconexão, é chamada a função *onDisconnect*. Esta função altera a variável *deviceConnected* para *false* e imprime uma mensagem no monitor serial indicando que o dispositivo cliente está desconectado. A Figura 26 apresenta a estrutura da classe referente.

Figura 26 – Classe *MyServerCallbacks*

```
337 class MyServerCallbacks : public BLEServerCallbacks{
      void onConnect(BLEServer* pServer){//onConnect: A função altera as
338 variáveis deviceConnected e temporary_connected para TRUE
339     deviceConnected = true;
340     Serial.println("Dispositivo cliente conectado!");
341     temporary_connected = true;
342   };
      void onDisconnect(BLEServer* pServer){ //onDisconnect: A função
343 altera a variável deviceConnected para FALSE.
344     deviceConnected = false;
345     Serial.println("Dispositivo cliente desconectado!");
```

Fonte: Autoria própria.

5.1.12 Classe LedCallbacks

Para executar comandos de controle do estado do LED da ESP32-LoRa por meio do Bluetooth Low Energy, definiu-se a classe *LedCallbacks* (classe pública do tipo *BLECharacteristicCallbacks*) para armazenar funções de retorno de chamada (*callback*) a serem executadas quando a característica *pCharacteristic_LED* tem seu valor alterado pelo dispositivo cliente (WRITE) ou quando é solicitada uma leitura do valor (READ).

Quando um dispositivo cliente solicita a leitura do valor da característica *pCharacteristic_LED*, a função *onRead*, ilustrada na Figura 27, é chamada. Esta função verifica o estado atual do LED embarcado (*ONBOARD_LED*). Se o LED estiver aceso (HIGH), o valor 1 é atribuído a *led_status*. Caso contrário, o valor 0 é atribuído. O valor de *led_status* é então configurado na característica *pCharacteristic_LED*.

Figura 27 – Descrição da função *onRead* da classe *LedCallbacks*

```
359   void onRead(BLECharacteristic* pCharacteristic_LED){ //suporta um
    pedido de leitura do valor pelo dispositivo cliente (habilitado com a
    propriedade READ)
360     char led_status[2];
361     if(digitalRead(ONBOARD_LED) == HIGH){ led_status[0] = '1';}
363     else{ led_status[0] = '0';}
366     pCharacteristic_LED->setValue(led_status); //a conversão para
    char é necessária para que os dados sejam corretamente identificados
    no MIT APP Inventor. String não é tipo aceito e int gera erro de in-
    terpretação do código: a função setValue da característica BLE espera
    uma sequência de bytes (uint8_t *) e um tamanho como argumentos
    quando você está passando dados que não são uma string C-style ou um
    array de bytes.
```

Fonte: Autoria própria.

Quando um dispositivo cliente escreve um valor na característica *pCharacteristic_LED*, a função *onWrite*, apresentada no APÊNDICE N, é chamada. Esta função obtém o valor escrito pelo usuário (que pode ser 1 ou 0) e o converte para um inteiro. Dependendo do valor recebido, o estado do LED é alterado: se o valor for 1, o LED é aceso; se for 0, o LED é apagado. Além disso, a função imprime o valor recebido e o estado atual do LED no monitor serial e atualiza o display com essas informações.

5.1.13 Classe MsgCallbacks

Similarmente à classe *LedCallbacks*, *MsgCallbacks* define funções de retorno de chamada que são executadas quando a característica *pCharacteristic_MSG* tem seu valor alterado pelo dispositivo cliente (WRITE). Dessa forma, possibilita a comunicação de texto entre o dispositivo cliente e o ESP32, permitindo que mensagens enviadas pelo cliente sejam exibidas no display e registradas no monitor serial.

Foi definida uma classe pública *BLECharacteristicCallbacks* denominada *MsgCallbacks*. A classe contém uma função principal que é chamada em eventos de escrita de um novo valor para *pCharacteristic_MSG*. A função *onWrite* é acionada quando um dispositivo cliente escreve um valor na característica. Dentro dessa função, o valor recebido é armazenado como uma string em *rxValue*. O valor da mensagem é então impresso no monitor serial para verificação. A estrutura desta função pode ser observada na Figura 28.

Figura 28 – Trecho inicial da função *onWrite* da classe *MsgCallbacks*

```
411   void onWrite(BLECharacteristic* pCharacteristic_MSG){ //onWrite é
    executado quando o dispositivo cliente (PC ou smartphone) ESCREVE
    um valor no dispositivo servidor (ESP32)
412     std::string rxValue = pCharacteristic_MSG->getValue(); //rxVa-
    lue é uma string que recebe o valor escrito pela pCharacteris-
    tic_MSG
413     Serial.print("Mensagem digitada: ");
414     // Converta a std::string para uma string de estilo C antes de
    concatenar
415     Serial.print(rxValue.c_str());
416     Serial.println();
```

Fonte: Autoria própria.

O valor da mensagem recebida (*rxValue*) é convertido para uma string e atribuído à variável *var_valor*, usada para exibir a mensagem no display. O título da mensagem é definido como “Mensagem:”.

A função *mostrar_dados_display_lcd* é chamada para atualizar o display com a mensagem recebida. As configurações de largura, altura e bits da imagem no display são definidas de acordo com a imagem correspondente à característica. Além disso, o horário da última alteração (*lastMsg*) é registrado para que a mensagem seja exibida no display por 3 segundos, a menos que outra característica seja lida ou escrita, o que interromperia essa exibição. Esses procedimentos são expressos no APÊNDICE O.

5.1.14 Função setup: inicialização do ESP32 e configuração BLE

A função *setup* contém as instruções que configuram a placa ESP32-LoRa para iniciar suas operações, incluindo a configuração da taxa de transmissão de dados, de pinos, criação de um servidor BLE e definição das características e serviços BLE. A Figura 29 contém as instruções iniciais formuladas na função.

Figura 29 – Trecho inicial da função *setup()*

```
442 void setup() {
443 Serial.begin(115200);// 115200 => Taxa de transmissão do monitor serial
444 Serial.println("Iniciando...");
446 inicializacao_bme_and_display();
```

Fonte: Autoria própria.

A comunicação serial foi configurada para uma taxa de transmissão de 115200 bps (bits por segundo). Em seguida, a função *inicializacao_bme_and_display* é chamada para inicializar o sensor BME280 e o display OLED.

Conforme mostrado na Figura 30, os pinos de GPIO são configurados (definiu-se o LED interno como porta de saída e o botão como de entrada). Assim, é utilizada a função *BLEDevice::createServer()* para inicializar o dispositivo BLE, com a criação do servidor e configuração dos *callbacks* para gerenciar conexões e desconexões.

Figura 30 – Definição de pinos e criação e configuração do servidor BLE

```
449 pinMode(ONBOARD_LED, OUTPUT); // Configura o LED interno como saída
450 pinMode(btn_PRG, INPUT); // Configura o botão como entrada
451 //Criação do servidor BLE
452 BLEDevice::init(ServerName); // Inicializa o dispositivo BLE com o nome
    fornecido
453 pServer = BLEDevice::createServer();// Cria um servidor BLE
454 pServer->setCallbacks(new MyServerCallbacks());// Configura callbacks
    para conexões e desconexões BLE.
```

Fonte: Autoria própria.

Foi criado o serviço BLE *pService_AIO* referente a Automation IO. Definiu-se o UUID SERVICE_UUID_AIO, criado anteriormente para esta funcionalidade e o parâmetro *handle* definido como 30 (de modo a permitir a manipulação de até 15 características no serviço simultaneamente), como exibido na Figura 31.

Figura 31 – Criação do serviço AIO (*pService_AIO*)

```
456   //criação do serviço do Automation IO
457   BLEService* pService_AIO = pServer->createService(SERVICE_UUID_AIO,
    30); //definido como 30 o handle, irá aceitar no máximo 15 caracterís-
    ticas no serviço SERVICE_UUID_AIO.
```

Fonte: Autoria própria.

Em seguida, foi configurada a característica *pCharacteristic_LED* para controlar o LED, com propriedades de leitura e escrita. A característica utiliza a classe *LedCallbacks* para gerenciar as interações do cliente com o LED. Foi gerado e configurado, utilizando a função *addDescriptor()*, um objeto de descritor *pDesc_LED* para atribuir informações adicionais sobre a característica associada ao LED. A Figura 32 demonstra essas configurações.

Figura 32 – Criação da característica LED

```
462 // CHARACTERISTIC --> LED:
463 pCharacteristic_LED = pService_AIO->createCharacteristic(CHARACTERIS-
    TIC_UUID_LED, BLECharacteristic::PROPERTY_WRITE | BLECharacteris-
    tic::PROPERTY_READ);
464 pCharacteristic_LED->setCallbacks(new LedCallbacks()); //ativa callback
    do led com as configurações/funções inseridas na classe LedCallbacks()
465   // DESCRIPTOR --> LED
466   //ponteiro descritor genérico para informação do LED no Bluetooth LE
    Explorer
467   pDesc_LED = new BLEDescriptor(DESCRIPTOR_UUID); //objeto do descritor
    para o LED com UUID de descritor genérico
468   pDesc_LED->setValue("Valor do LED (0 - OFF; 1 - ON)"); //descritor do
    led associa valor passado como parâmetro
469   pCharacteristic_LED->addDescriptor(pDesc_LED); //descritor do led com
    as configurações associadas é relacionado à característica correspon-
    dente
```

Fonte: Autoria própria.

De modo similar, foi criada a característica *pCharacteristic_BTN_STATUS* para o botão, com propriedades de leitura e notificação, como apresentado na Figura 33. O descritor *pDesc_BTN_STATUS* para fornecer informações sobre o status do botão também foi adicionado por meio da função *addDescriptor()*.

Figura 33 – Criação da característica STATUS

```
474 pCharacteristic_BTN_STATUS = pService_AIO->createCharacteristic(CHARAC-
    TERISTIC_UUID_BTN_STATUS, BLECharacteristic::PROPERTY_READ  | BLEChar-
    acteristic::PROPERTY_NOTIFY);
475   // DESCRIPTOR --> BUTTON PRESS
476 pDesc_BTN_STATUS = new BLEDescriptor(DESCRIPTOR_UUID);
477 pDesc_BTN_STATUS->setValue("Mostra o status do Botão (0 - OFF; 1 -
    ON)");
478   pCharacteristic_BTN_STATUS->addDescriptor(pDesc_BTN_STATUS);
```

Fonte: Autoria própria.

Outra característica BLE, *pCharacteristic_MSG*, foi criada para receber mensagens do cliente, com propriedades de leitura e escrita. *Callbacks* foram configurados para gerenciar as interações do cliente, e um descritor é adicionado para fornecer informações sobre a mensagem, como indicado na Figura 34.

Figura 34 – Criação da característica MSG

```
483 pCharacteristic_MSG = pService_AIO->createCharacteristic(CHARACTERIS-
    TIC_UUID_MSG, BLECharacteristic::PROPERTY_WRITE | BLECharacteris-
    tic::PROPERTY_READ);
484 pCharacteristic_MSG -> setCallbacks(new MsgCallbacks());
485 //DESCRIPTOR --> MENSAGEM
486 pDesc_MSG = new BLEDescriptor(DESCRIPTOR_UUID);
487 pDesc_MSG -> setValue("Mensagem digitada pelo usuário.");
488 pCharacteristic_MSG -> addDescriptor(pDesc_MSG);
```

Fonte: Autoria própria.

Definiu-se um serviço BLE para o monitoramento das variáveis ambientais analisadas (temperatura, pressão, altitude e umidade), mostrado na Figura 35. Cada variável é associada a uma característica BLE com propriedades de leitura e notificação.

Figura 35 – Criação do serviço ESS (*pService_BME*) e definição das suas características

```
490 //definir o serviço de variáveis do meio ambiente (ENVIRONMENTAL SEN-
    SING)
491 BLEService* pService_BME = pServer->createService(SERVICE_UUID_BME,
    30); //handle = 30 -> com isso, no máximo 15 características podem ser
    incluídas nesse serviço
493   pCharacteristic_Temp = pService_BME->createCharacteristic(CHARACTER-
    ISTIC_UUID_Temp, BLECharacteristic::PROPERTY_READ | BLECharacteris-
    tic::PROPERTY_NOTIFY);
494   pService_BME->addCharacteristic(pCharacteristic_Temp);
496   pCharacteristic_Pre = pService_BME->createCharacteristic(CHARACTERIS-
    TIC_UUID_Pre, BLECharacteristic::PROPERTY_READ | BLECharacteris-
    tic::PROPERTY_NOTIFY);
497   pService_BME->addCharacteristic(pCharacteristic_Pre);
```

```
499   pCharacteristic_Alt = pService_BME->createCharacteristic(CHARACTERIS-
    TIC_UUID_Alt, BLECharacteristic::PROPERTY_READ | BLECharacteris-
    tic::PROPERTY_NOTIFY);
500   pService_BME->addCharacteristic(pCharacteristic_Alt);
502   pCharacteristic_Umi = pService_BME->createCharacteristic(CHARACTERIS-
    TIC_UUID_Umi, BLECharacteristic::PROPERTY_READ | BLECharacteris-
    tic::PROPERTY_NOTIFY);
503   pService_BME->addCharacteristic(pCharacteristic_Umi);
```

Fonte: Autoria própria.

Os descritores *pDesc_Alt*, *pDesc_Umi*, *pDesc_Temp* e *pDesc_Pre* foram adicionados às características ambientais para fornecer informações detalhadas sobre cada variável monitorada (temperatura, pressão, altitude e umidade), conforme apresentado na Figura 36.

Figura 36 – Criação dos descritores do serviço ESS

```
507   pDesc_Alt = new BLEDescriptor(DESCRIPTOR_UUID);
508   pDesc_Umi = new BLEDescriptor(DESCRIPTOR_UUID);
509   pDesc_Temp = new BLEDescriptor(DESCRIPTOR_UUID);
510   pDesc_Pre = new BLEDescriptor(DESCRIPTOR_UUID);
512   pDesc_Temp->setValue("Temperatura em graus Celsius (°C)");
513   pCharacteristic_Temp->addDescriptor(pDesc_Temp);
514   pDesc_Pre->setValue("Pressão Atmosférica em hectopascal (hPa)");
515   pCharacteristic_Pre->addDescriptor(pDesc_Pre);
516   pDesc_Alt->setValue("Altitude em metros (m)");
517   pCharacteristic_Alt->addDescriptor(pDesc_Alt);
518   pDesc_Umi->setValue("Umidade Relativa do Ar (%)");
519   pCharacteristic_Umi->addDescriptor(pDesc_Umi);
```

Fonte: Autoria própria.

Por fim, foram iniciados os serviços BLE, de forma que o dispositivo pudesse começar a anunciar sua presença para conexões de clientes disponíveis. Uma mensagem é impressa no monitor serial indicando que o dispositivo está esperando por conexões de clientes para enviar notificações. A Figura 37 ilustra a inicialização desses serviços.

Figura 37 – Inicialização dos serviços AIO e ESS

```
523   pService_AIO->start(); //ativa a operação de serviço do AIO com as
    características definidas no escopo
525   pService_BME->start(); //ativa a operação de serviço do ENVIRONMENTAL
    SENSING com as características definidas no escopo
527   pServer->getAdvertising()->start();
529   Serial.println("Waiting a client connection to notify...");
530 }
```

Fonte: Autoria própria.

5.1.15 Estrutura loop()

A função *loop()* é responsável por monitorar a conexão BLE e executar várias ações dependendo do estado da conexão entre o servidor e o cliente. A Figura 38 apresenta o trecho inicial de sua estrutura. Foi definida uma variável do tipo inteiro *i* para iterar de 1 a 4 de forma a permitir a atualização intercalada das características do Environmental Sensing Service.

Figura 38 – Trecho inicial da função *loop()*

```
558 void loop() {
559 if (deviceConnected) {
560   int i = 1;
561   while(i <= 4){
562     now = millis();
563     //Envio periódico das variáveis do sensor BME280 via BT e serial
564     BME_Dados ler_dados = read_bme_data(); //   READ       ::   quando
    o usuário solicitar a leitura dos valores das características
565       pCharacteristic_LED->setValue(led_signal);
566       pCharacteristic_MSG->setValue(btn_signal);
```

Fonte: Autoria própria.

Quando o dispositivo está conectado, inicia-se um ciclo em que a cada período de 3 segundos é realizada a leitura de um dos elementos do sensor BME280 e são atualizados os valores das características do LED e da mensagem. Sucessivamente, são inseridas instruções para o monitoramento do estado do botão e, em casos de alterações, é atualizado o valor da característica do botão e enviada uma notificação para o cliente, conforme mostra a Figura 39.

Figura 39 – Comandos para verificação do acionamento do botão PRG

```
567       buttonState = digitalRead(btn_PRG);
568       if (buttonState != lastButtonState) {
569         Serial.println("Início button status alterado");
570         StatusBtn = 0;
571         if (buttonState == LOW) { //se é alterado para PRESSIONADO
572           StatusBtn = 1;
573           var_valor = String("ON");
574         }else{ // se é alterado retornando para NÃO PRESSIONADO
575           StatusBtn = 0;
576           var_valor = String("OFF");
577         }
578         String(StatusBtn).toCharArray(sig, 2); //sig é um char de 2
    posições obtido a partir da string de StatusBtn
579         pCharacteristic_BTN_STATUS->setValue(sig);
580         pCharacteristic_BTN_STATUS->notify();
```

Fonte: Autoria própria.

Quando o dispositivo não está conectado, ele aguarda por uma conexão. Se uma desconexão for detectada, a função *funcReset* é chamada para reiniciar o fluxo, como indicado na Figura 40.

Figura 40 – Configuração de parâmetros para exibição do estado atual do botão no display OLED

```
582         var_valor.toCharArray(sig, 50);
583         Serial.println(sig);
584         var_titulo = String("Botão:");
585         frame_width    =   frame_push_button_width;
586         frame_height   =   frame_push_button_height;
587         frame_bits     =   frame_push_button_bits;
588         pos_x_titulo   =   64;
589         pos_x_valor    =   64;
590         mostrar_dados_display_lcd(pos_x, pos_y, pos_x_titulo,
    pos_x_valor, frame_width, frame_height, frame_bits, var_titulo,
    var_valor);
592         lastMsg=now;
593       }
594       lastButtonState = buttonState; //altera-se o valor do
    lastButtonState para o mesmo do último estado constatado
595       if (now - lastMsg > 3000) {  //Intervalo de envio a cada 3
    segundos
596         lastMsg = now; //atualiza o horário de início de exibição para
    reiniciar o ciclo de 3 s.
597         Serial.println("--Variáveis do Sensor--");
598         print_LCD_and_APP(i, ler_dados); //   NOTIFY     ::   atualizar
    os valores das variáveis ambientais no display e no cliente
599         i++;}
601       vTaskDelay(500 / portTICK_PERIOD_MS);  // Pausa a tarefa por 500
    ms -> similar a //delay(500); no entanto, permite a execução de outras
    ações do código durante a espera.
602     }
603   } else{
604     if (temporary_connected == true){
    Serial.println(temporary_connected);;funcReset();}
605   }
606   vTaskDelay(500 / portTICK_PERIOD_MS); //delay(500);
607 }
```

Fonte: Autoria própria.

5.1.16 Estrutura de dados struct() para leitura de sensores

Para a leitura dos valores de temperatura, pressão, altitude e umidade do sensor BME280, foi utilizada a estrutura *BME_Dados* apresentada em 5.1.6. Dessa forma, foi desenvolvida a função *read_bme_data()* para realizar a leitura dos dados do sensor BME280 e atribuir os valores obtidos às características BLE (Bluetooth Low Energy) definidas no sistema. Esta

função é chamada periodicamente para garantir que os dados estejam atualizados e disponíveis para transmissão para os dispositivos clientes conectados, como ilustrado na Figura 41.

Figura 41 – Estrutura para leitura dos valores das características ESS

```
625 BME_Dados read_bme_data(){
627   BME_Dados dados;
629   dados.temperatura = bme.readTemperature();
630   dtostrf(dados.temperatura, 4, 2, msg_temp);
631   pCharacteristic_Temp->setValue(msg_temp);   //O valor da temperatura
    com duas casas decimais é passado como uma string
633   dados.pressao = bme.readPressure()/100.0F;
634   dtostrf(dados.pressao, 4, 2, msg_pre);
635   pCharacteristic_Pre->setValue(msg_pre);     //O valor da pressão com
    duas casas decimais é passado como uma string
637   dados.altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
638   dtostrf(dados.altitude, 4, 2, msg_alt);
639   pCharacteristic_Alt->setValue(msg_alt);     //O valor da altitude
    com duas casas decimais é passado como uma string
641   dados.umidade = bme.readHumidity();
642   dtostrf(dados.umidade, 4, 2, msg_umi);
643   pCharacteristic_Umi->setValue(msg_umi);     //O valor da umidade com
    duas casas decimais é passado como uma string para a BLE_Characteris-
    tic
645   return dados;}
```

Fonte: Autoria própria.

Na função *read_bme_data*, a leitura e processamento de cada parâmetro ambiental seguem um fluxo específico. Para a temperatura, a função *bme.readTemperature()* é usada para obter o valor, que é então convertido para uma *string* com duas casas decimais (utilizando *dtostrf()*).

A *string* resultante é atribuída à característica BLE correspondente (*pCharacteristic_Temp*) através do método *setValue()*. No caso da pressão, o valor é lido com *bme.readPressure()* e dividido por 100.0 para converter de Pa para hPa.

Da mesma forma, o valor convertido para *string* é atribuído à característica BLE *pCharacteristic_Pre*. Para a altitude, a leitura é feita utilizando a pressão atmosférica ao nível do mar por meio da função *bme.readAltitude(SEALEVELPRESSURE_HPA)*, e o valor convertido para *string* é atribuído à característica BLE (*pCharacteristic_Alt*). A umidade é lida com *bme.readHumidity()*, e após a conversão para *string*, o valor é atribuído à característica BLE *pCharacteristic_Umi*.

5.2 Configurações do dispositivo cliente

5.2.1 Construção do aplicativo Android no MIT APP Inventor para controle e monitoramento de variáveis

Para o monitoramento e controle das características dos serviços AIO e ESS, de modo a proporcionar uma interface amigável ao usuário, foi desenvolvido um aplicativo no MIT APP Inventor, denominado *BLE AIO BME*. O aplicativo foi estruturado de maneira hierárquica, se baseando na sequência de operações principais da conexão BLE: escaneamento, solicitação de conexão (necessárias para o estabelecimento de uma conexão bidirecional entre dois dispositivos), e interação com características (monitoramento e controle dos seus valores). Dessa forma, o usuário primeiramente seleciona o dispositivo BLE para estabelecer a conexão P2P. Após o êxito da conexão, são exibidas as características monitoradas e fornecidas opções para alterar os valores dos itens controláveis. Os principais componentes e funcionalidades utilizados para confeccionar a lógica de funcionamento do sistema estão descritos na Tabela 7.

Tabela 7 – Principais componentes utilizados para desenvolvimento da arquitetura do aplicativo BLE AIO BME

Componente/Bloco de função	Funcionalidade
When (tela) Initialize	Permite a execução de ações no momento em que a tela atual do aplicativo é acessada (exemplo: verificação de permissões, estado de conexão do dispositivo etc.).
When (tela) OrientationChanged	Utilizada para executar ações (como alterar a imagem de plano de fundo ou redimensionar objetos na tela) ao alterar a tela do dispositivo entre os modos retrato e paisagem.
When (botão/imagem/elemento) click	Utilizada para executar ações quando um elemento (botão, imagem ou outro objeto) é pressionado.
AskForPermission	Solicitar permissões específicas do dispositivo em tempo de execução que devem ser concedidas pelo usuário para executar determinada ação (exemplos: permissão Bluetooth e localização).
ListPicker	Possibilita ao usuário selecionar um item de uma lista e acionar um evento, permitindo que o aplicativo execute ações com base na escolha realizada.
BeforePicking (ListPicker)	Definição das ações a serem executadas antes que um elemento do ListPicker seja selecionado.
AfterPicking (ListPicker)	Definição das ações a serem executadas após a seleção de um elemento do ListPicker.
TinyDB	Componente de banco de dados local no MIT App Inventor que permite armazenar pequenos volumes de dados diretamente no dispositivo (dados armazenados não são voláteis durante a execução da aplicação).

StoreValue (TinyDB)	Utilizado para armazenamento de parâmetros de variáveis (funcional para a obtenção do valor da variável em outras páginas do aplicativo, por exemplo).
GetValue (TinyDB)	Utilizado para acesso a parâmetros de variáveis pré-armazenados.
Initialize global	Inicializa uma variável global e armazena o valor do bloco a ele anexado.
Notifier	Exibir alertas e notificações na interface do aplicativo
ShowAlert notice	Exibe uma mensagem simples ao usuário, que desaparece após alguns segundos ou quando o usuário interage com a tela.

Fonte: Autoria própria.

5.2.1.1 Recursos de permissão do usuário

Inicialmente, foram construídas telas para solicitar ao usuário a ativação das permissões de Bluetooth e localização, ambas necessárias para a manipulação, aquisição e transmissão dos dados das variáveis conforme a Figura 42. Este processo foi realizado utilizando os blocos *call.AskForPermission* para solicitar as permissões *FineLocation*, *LocationExtraCommands*, *CoarseLocation* e *BluetoothConnect*, e, adicionalmente, o bloco *ActivityStarter.Action* para direcionar o usuário à tela de configuração da localização nos dispositivos Android (*android.settings.LOCATION_SOURCE_SETTINGS*).

Figura 42 – Tela de permissões do APP e respectiva estrutura em blocos configurada

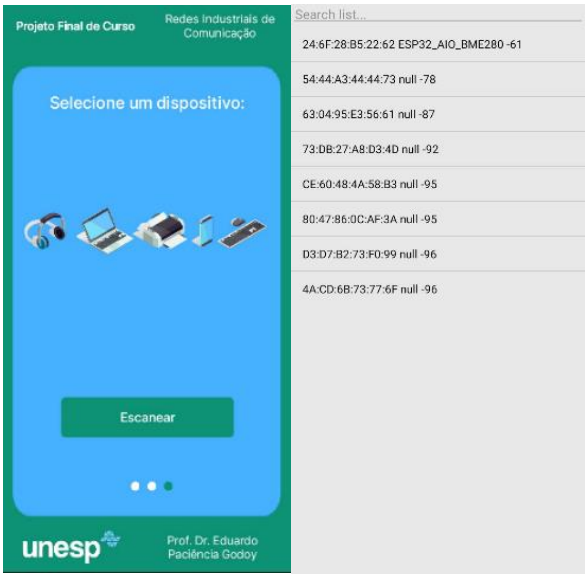
Fonte: Autoria própria.

Quando a tela é iniciada, são verificadas se as permissões Bluetooth e de localização do dispositivo estão ativadas e, em caso negativo (*PermissionDenied*), são invocadas as funções necessárias para a ativação de ambos os recursos. Por fim, ao clicar sobre o botão “Próximo” (*btn_iniciar*), o usuário é destinado à tela de seleção do dispositivo ao qual deve se conectar.

5.2.1.2 Recursos do Bluetooth Low Energy

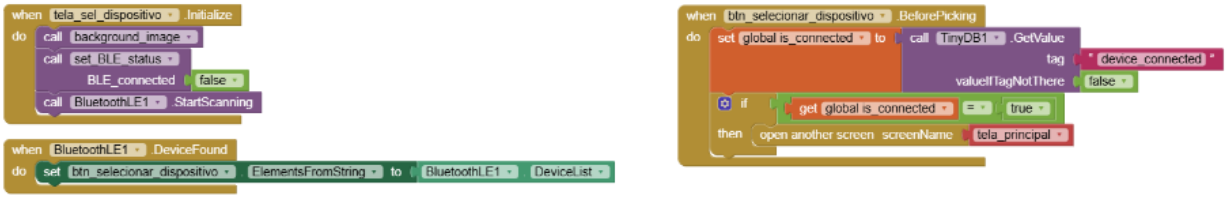
Para ter acesso às funções do protocolo Bluetooth Low Energy, foi importado o componente *BluetoothLE*, disponível no repositório de extensões adicionais do MIT App Inventor Extensions (MIT App Inventor, 2024). Com isso, desenvolveu-se um ambiente para a seleção do dispositivo Bluetooth Low Energy ao qual o usuário deseja se conectar. Quando é clicado sobre o botão "Selecionar dispositivo", é aberto um *ListPicker* e invocada a função *BluetoothLE .StartScanning* para iniciar a busca por dispositivos Bluetooth Low Energy disponíveis para conexão. Uma vez que se tenha um dispositivo selecionado e a conexão é bem estabelecida, é utilizada a função *BluetoothLE .StopScanning* para parar a procura por dispositivos e dar início às transmissões de dados. A Figura 43 e a Figura 44 apresentam respectivamente a tela e a estrutura em blocos desenvolvida.

Figura 43 – Tela de seleção de dispositivo e ListPicker com listagem dos elementos disponíveis



Fonte: Autoria própria.

Figura 44 – Estrutura de blocos da tela de seleção de dispositivos

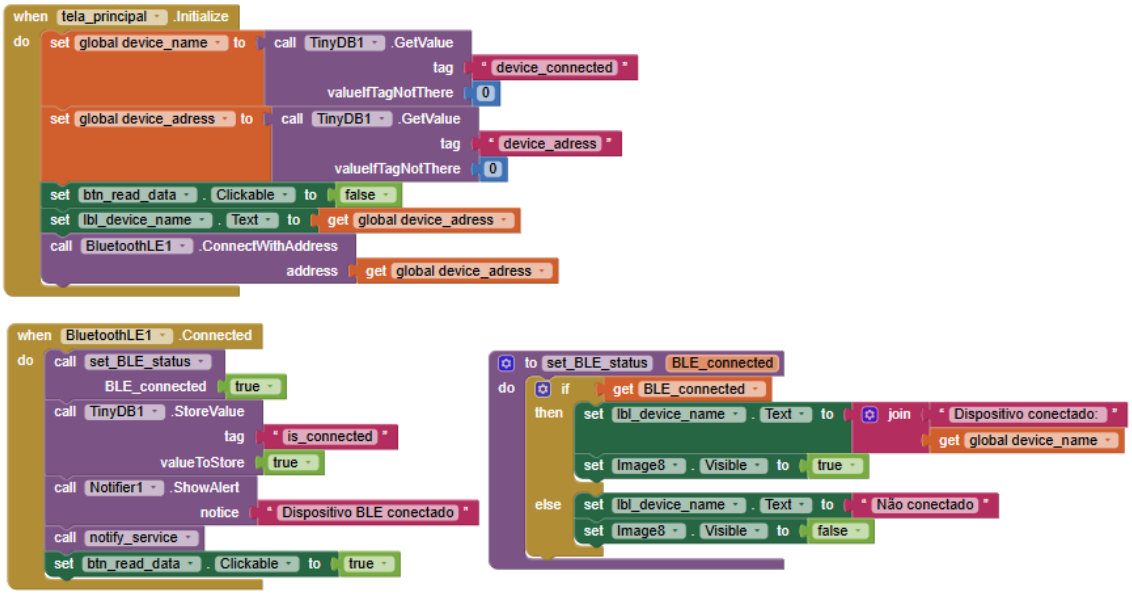


Fonte: Autoria própria.

5.2.1.3 Transferência de parâmetros de conexão e atributos entre telas

Para a transferência dos valores referentes aos parâmetros de conexão e dos atributos de uma tela para outra no MIT APP Inventor, de forma a manter a conexão estabelecida com o servidor, foi utilizado o *TinyDB*, que possibilita o armazenamento de dados para transferência entre telas. Com ele, são atribuídas duas funções principais: *StoreValue* e *GetValue*, respectivamente, para armazenar e obter valores de variáveis trabalhadas na tela original para serem transmitidos. A Figura 45 apresenta o diagrama de blocos configurado para atribuir os parâmetros da conexão e as funções usadas para alertar o usuário acerca do estado da conexão.

Figura 45 – Atribuição dos parâmetros da conexão e chamada das funções de alerta de status



Fonte: Autoria própria.

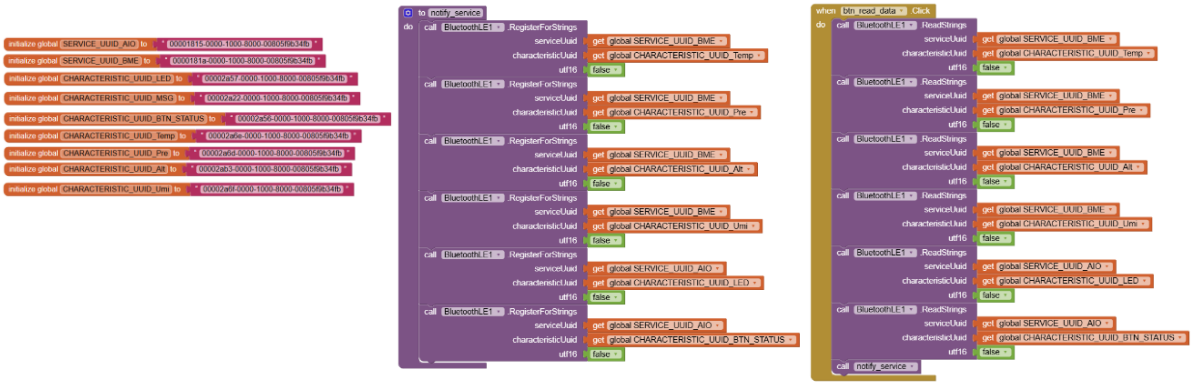
5.2.1.4 Amostragem de mensagens de alertas de conexão

A fim de mostrar alertas informando sucesso ou insucesso na conexão do dispositivo cliente ao servidor, foi utilizado o componente *Notify*. A partir da função *ShowAlert .notice*,é exibida uma mensagem de texto no centro da tela (“Não conectado” ou “Dispositivo conectado”).

5.2.1.5 Monitoramento e controle de características

A plataforma permite ao usuário monitorar e controlar as características sem a necessidade de interação direta com os valores UUID, já que estes estão referenciados na lógica principal do programa, como ilustrado no trecho do diagrama em blocos da Figura 46. Vale ressaltar que, para que seja realizada a identificação da característica correspondente, os UUIDs devem ser inseridos no formato de 128 bits (por exemplo, “0000xxxx-0000-1000-8000-00805F9B34FB”), uma vez que a forma breve (0x???) não é reconhecida pela plataforma. Com isso, o aplicativo realiza a varredura dos dispositivos disponíveis para conexão BLE, utilizando os UUIDs pré-programados para as características e serviços ESS e AIO. Quando a conexão é bem-sucedida, o aplicativo começa a exibir os dados transmitidos pelo servidor.

Figura 46 – Diagrama em blocos com UUIDs inicializados e funções *RegisterForStrings* e *ReadStrings*

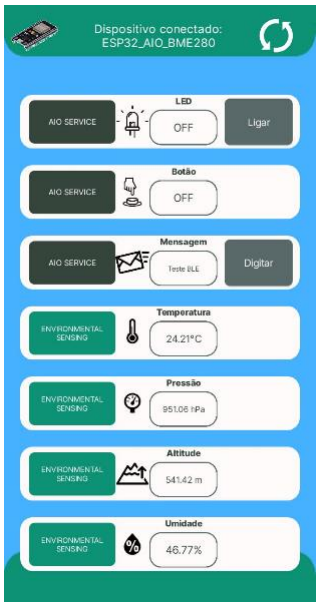


Fonte: Autoria própria.

Analogamente ao uso das funções *callback* utilizadas no firmware, a função *notify_service* reúne uma série de funções que são invocadas quando há atualização no valor da característica de referência, permitindo o recebimento dos dados do dispositivo periférico para o central.

Para exercer permitir que as atualizações ocorram conforme a solicitação do usuário, foi inserido um botão de atualização no canto superior direito da tela principal de monitoramento das variáveis (Figura 47). Ele permite ao usuário realizar uma leitura instantânea dos valores das características, atualizando-os conforme solicitado. Com seu pressionamento, são chamadas as funções *call .ReadStrings* para realização da coleta dos valores armazenados em cada característica de referência. Se há alteração no valor, a função *notify_service* é aplicada para possibilitar a atualização.

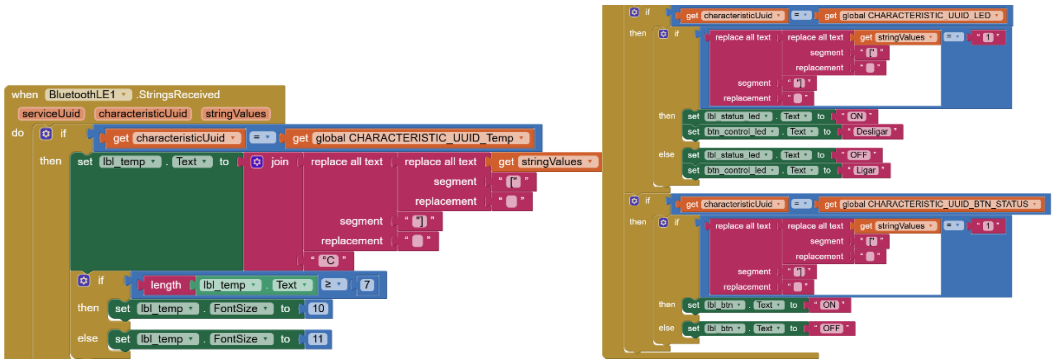
Figura 47 – Tela principal para monitoramento e controle das características em tempo real



Fonte: Autoria própria.

Os valores das características obtidos como cadeias de texto são lidos pelo dispositivo cliente com a função *when BluetoothLE.StringsReceived* no formato ["valor+unidade"]. Para a remoção dos colchetes e aspas duplas contidos neste padrão de cadeia de texto (e manter apenas valor+unidade) para exibição nos rótulos de texto da tela principal, foi utilizado o conjunto dos blocos *join* e *replace .text* para substituir os "*[*"" e ""*]*" por um elemento vazio, de modo a removê-los. A Figura 48 apresenta os modelos das funções utilizadas para a extração dos dados recebidos nos serviços ESS e AIO.

Figura 48 – Diagrama em blocos apresentando modelos das funções utilizadas para a extração dos dados recebidos nos serviços ESS e AIO

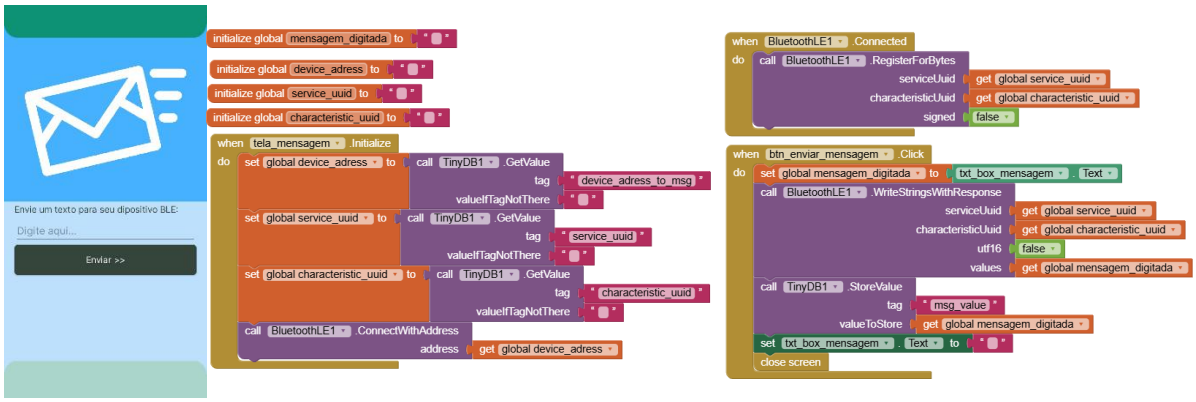


Fonte: Autoria própria.

Para envio das mensagens, foi confeccionada uma tela própria com um campo de entrada de texto *TextBox*, no qual o usuário pode inserir o texto desejado, e um botão "Enviar", pelo

qual se faz a chamada da função *BluetoothLE .WriteStringsWithResponse* para envio da cadeia de caracteres ao servidor. Também é utilizado o bloco de função *TinyDB .StoreValue* para permitir a exibição do texto transmitido na tela principal do aplicativo. A Figura 49 exibe a tela respectiva e a configuração lógica realizada.

Figura 49 – Diagrama em blocos apresentando modelos das funções utilizadas para a extração dos dados



Fonte: Autoria própria.

O APÊNDICE P apresenta os blocos de funções do componente BluetoothLE que foram utilizados neste projeto.

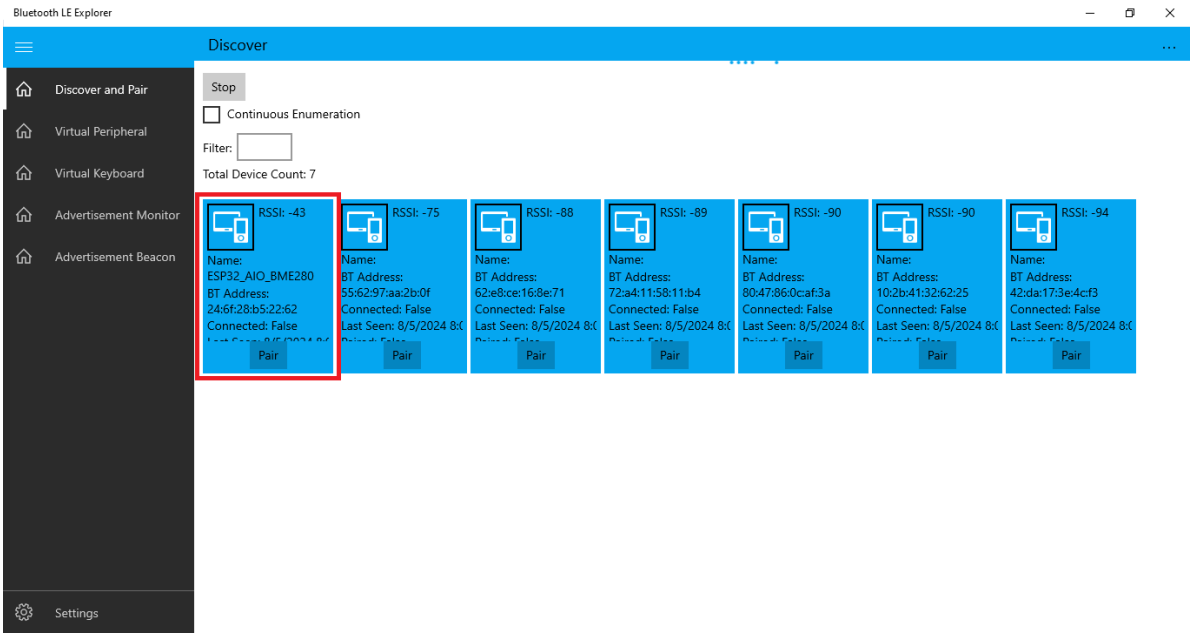
5.2.2 Bluetooth LE Explorer

Adicionalmente à aplicação desenvolvida no MIT APP Inventor, para a realização dos testes de comunicação BLE, utilizou-se o aplicativo Bluetooth LE Explorer como cliente desktop. Com essa ferramenta, pôde-se obter uma visualização detalhada das propriedades e configurações do dispositivo servidor, bem como dos serviços envolvidos.

Para iniciar a comunicação, primeiro é necessário que o computador (dispositivo cliente) esteja com os recursos Bluetooth e a localização ativados. Após selecionar o botão *scan* (escanear), o software inicia a varredura dos dispositivos BLE que estão disponíveis para estabelecer uma comunicação. Uma vez localizado o dispositivo desejado (ESP32_AIO_BME280), é possível clicar sobre o cartão do dispositivo indicado para estabelecer a comunicação P2P.

A Figura 50 apresenta a tela inicial do software, após uma solicitação de escaneamento de dispositivos Bluetooth Low Energy próximos.

Figura 50 – Tela de seleção de dispositivos do Bluetooth LE Explorer

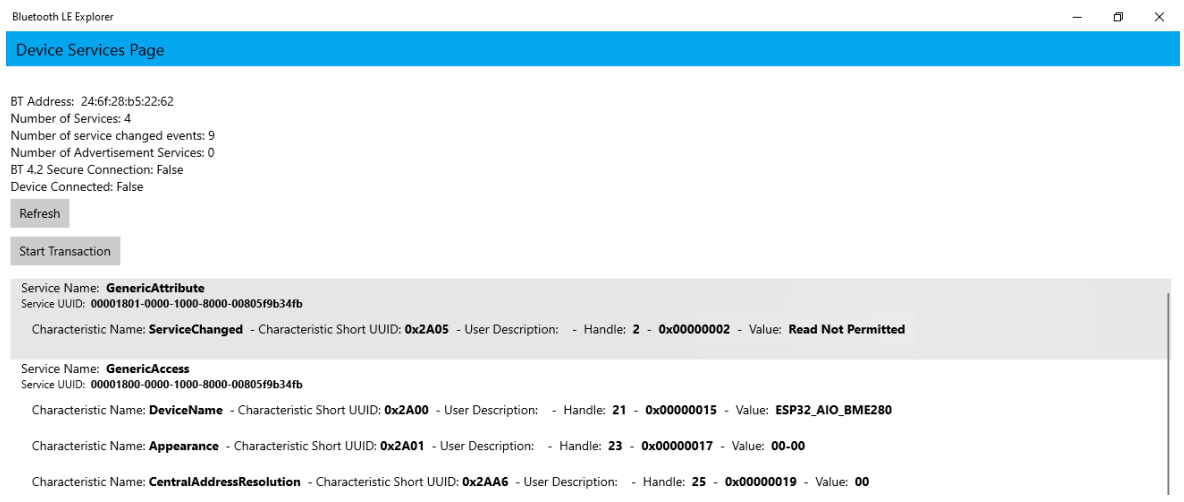


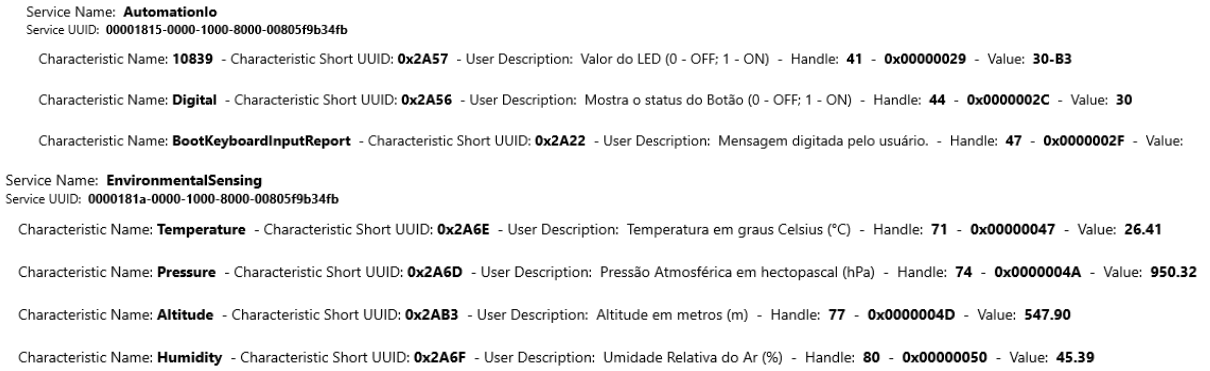
Fonte: Autoria própria.

O Bluetooth LE Explorer já possui em background as configurações necessárias ao reconhecimento automático dos nomes dos serviços por meio dos UUIDs padronizados pré-definidos na tabela do Bluetooth SIG, tais como os serviços de Environmental Sensing e Automation IO. Além disso, os descritores de texto padrão foram exibidos juntamente com suas respectivas características, conforme as definições do Bluetooth SIG.

Após a conexão, é exibida uma interface detalhada que mostra os serviços registrados no servidor, as características de cada serviço e seus respectivos descritores. A Figura 51 exibe a janela de listagem dos serviços, características e propriedades associadas, inclusive os descritores de texto definidos a cada qual.

Figura 51 – Telas geral de serviços e detalhes de características do Bluetooth LE Explorer

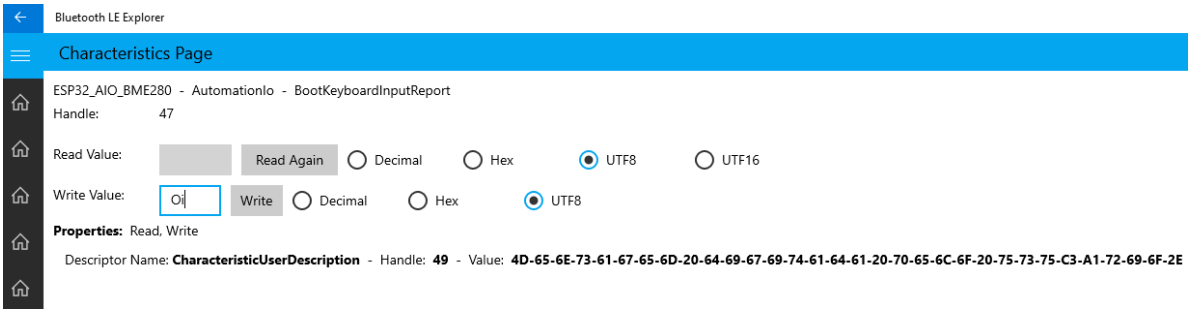




Fonte: Autoria própria.

Para controlar os valores das características MSG (como mostra a Figura 52) e o Status do LED, pode-se selecionar o item desejado e localizar o campo para digitar o valor atrelado à propriedade *Write Value:* Seleciona-se o tipo do dado associado e deve pressionar o botão *Write*. Com isso, o valor é atualizado no servidor e exibido no display OLED. Além disso, é possível alterar os valores das características, caso elas permitam operações de escrita (WRITE) ou escrita sem resposta (WRITE NO RESPONSE).

Figura 52 – Escrita de valores de texto à característica MSG



Fonte: Autoria própria.

As informações disponibilizadas sobre o dispositivo incluem: *RSSI (*indicação da potência do sinal, relacionada diretamente à distância entre o servidor e o cliente); *nome e UUID (*nome e identificador universal único do dispositivo); *endereço Bluetooth (*endereço físico do dispositivo); e*stado de conexão (*status atual da conexão com o dispositivo).

Já em relação aos serviços, são apresentados: *nome do serviço* (denominação do serviço fornecido pelo dispositivo); *UUID* (identificador universal único do serviço); *propriedades* (capacidades da característica); *descritores* (descrições adicionais associadas à característica).

6 RESULTADOS E DISCUSSÕES

Nesta seção, são apresentados os resultados obtidos a partir dos testes realizados para verificar a efetividade da comunicação Bluetooth Low Energy entre o dispositivo servidor ESP32-LoRa e os dispositivos clientes. São exibidos os valores expressos no display OLED LCD, no aplicativo Bluetooth LE Explorer e no MIT APP Inventor para cada uma das situações de interesse, incluindo a visualização de características ambientais, envio de mensagens, controle do estado do LED interno e acionamento do botão PRG da placa ESP32-LoRa.

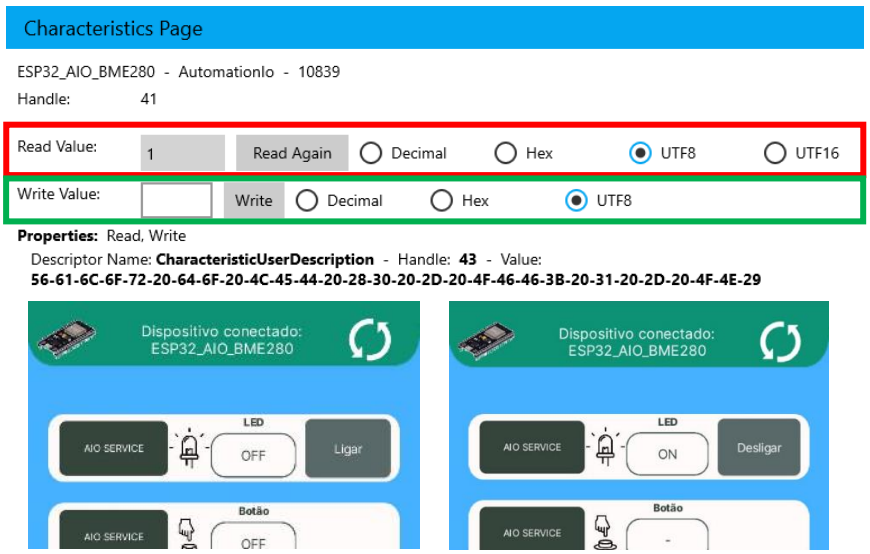
6.1 Característica LED

O controle do LED interno da placa ESP32-LoRa foi realizado por meio de comandos enviados pelos aplicativos BLE AIO BME e Bluetooth LE Explorer.

- **Controle via MIT APP Inventor:** no controle por meio do aplicativo BLE AIO BME, o usuário pode acionar o LED utilizando o botão "Ligar/Desligar" (*btn_control_led*). A comunicação é realizada através da propriedade WRITE. Quando a característica LED apresenta um valor diferente de "1", o botão exibe o rótulo "Ligar". Ao ser pressionado nessa condição, o valor "1" é enviado ao servidor com o bloco de função *WriteStringsWithResponse*, acionando o LED. Após essa ação, o rótulo do botão é automaticamente alterado para "Desligar". Nesse estado, ao pressionar o botão, o valor "0" é enviado ao servidor, resultando no desligamento do LED.
- **Controle via Bluetooth LE Explorer:** no aplicativo Bluetooth LE Explorer, o usuário seleciona a característica do LED, escolhe o formato de dados UTF-8, digita "1" para acender o LED ou "0" para apagá-lo, e clica em "Write" para enviar os dados ao servidor via BLE. O comando é transmitido utilizando a propriedade WRITE, e a resposta pode ser confirmada com a propriedade READ, atualizando o valor da característica nos dispositivos clientes.

Na Figura 53, nota-se um exemplo de envio de valor "1" (*Write Value* na área sinalizada em verde) e constatação do valor na solicitação de leitura (*Read Value*, na área sinalizada em vermelho) para controle do LED via Bluetooth LE Explorer e no MIT APP Inventor. Os valores utilizados estão no formato de texto codificado em UTF-8.

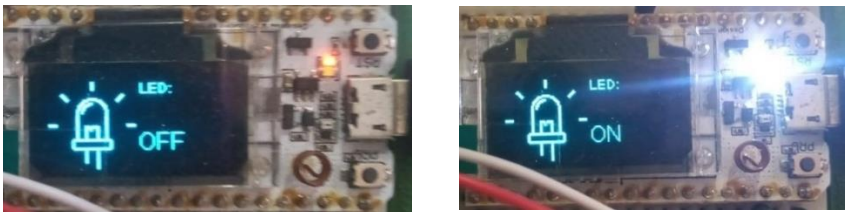
Figura 53 – Escrita e leitura de valores para a característica LED no Bluetooth LE Explorer e BLE AIO BME



Fonte: Autoria própria.

No display OLED LCD, o envio das cadeias de caracteres únicas “1” ou “0” é demonstrado pela visualização do ícone e do valor, como ilustrado nas imagens da Figura 54.

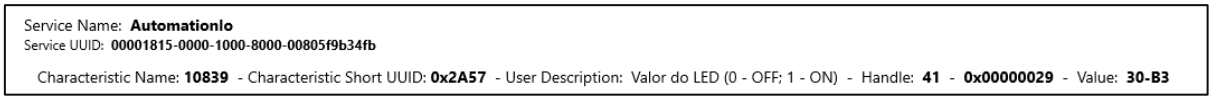
Figura 54 – Exibição da mudança do estado do LED para OFF (na imagem à esquerda) e mudança para ON na imagem à direita



Fonte: Autoria própria.

O descritor de texto (*User Description*) configurado no firmware da placa ESP32-LoRa é apresentado na tela de visão geral do Bluetooth LE Explorer, junto com as demais propriedades associadas à característica, conforme mostrado na Figura 55.

Figura 55 – Visão geral das propriedades da característica LED no Bluetooth LE Explorer



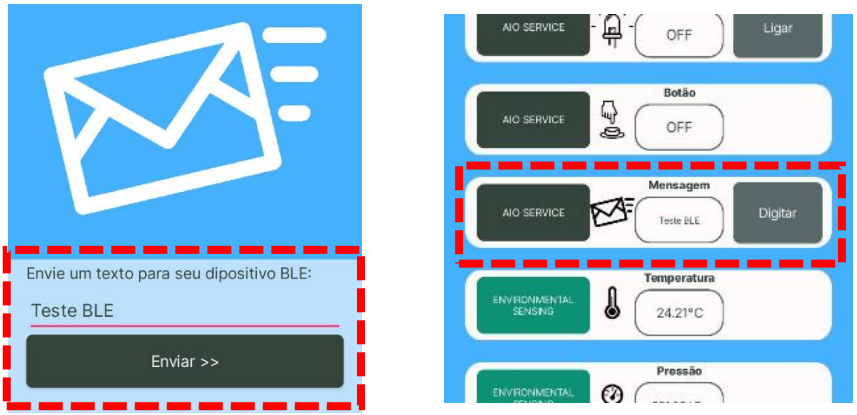
Fonte: Autoria própria.

6.2 Característica MSG

O envio de mensagens de texto do dispositivo cliente para o dispositivo servidor também foi realizado pelos aplicativos BLE AIO BME e Bluetooth LE Explorer.

- **Envio via MIT APP Inventor:** utilizando o aplicativo BLE AIO BME, a última mensagem enviada é exibida no campo na tela principal do aplicativo. Para enviar uma nova mensagem ao dispositivo Bluetooth Low Energy conectado, ao clicar sobre o botão "Enviar" (*btn_enviar_mensagem*), a função *WriteStringsWithResponse* é chamada e o texto digitado no campo *txt_box_mensagem* transferido ao servidor, atualizando a visualização na tela principal do aplicativo e no display OLED, conforme demonstrado na Figura 56 e na Figura 57.

Figura 56 – Elementos para envio de mensagem e valor da característica MSG atualizado na tela principal



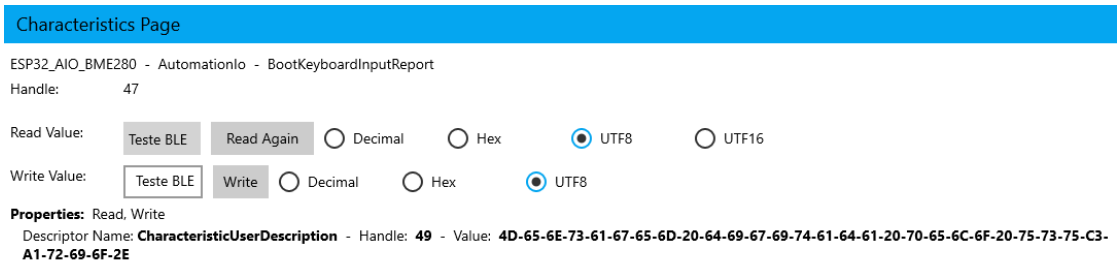
Fonte: Autoria própria.

Figura 57 – Valor da característica MSG atualizado na tela do display OLED

Fonte: Autoria própria.

- **Envio via Bluetooth LE Explorer:** no Bluetooth LE Explorer, o usuário seleciona a característica MSG apresentada na Figura 59, escolhe o formato de dados UTF-8, digita uma cadeia de caracteres no campo de texto indicado, e clica em "Write" para submissão da frase ao servidor. O comando é transmitido utilizando a propriedade WRITE, e a resposta pode ser confirmada com a propriedade READ, atualizando o valor da característica nos dispositivos clientes, conforme visto na página da característica MSG indicada na Figura 58.

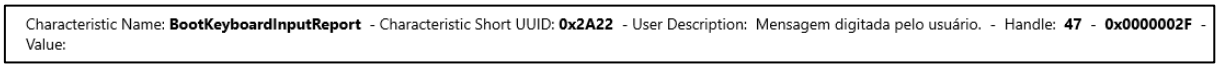
Figura 58 – Escrita e leitura de valores para a característica MSG no Bluetooth LE Explorer



Fonte: Autoria própria.

O descritor de texto (*User Description*) da característica MSG configurado no firmware da placa ESP32-LoRa é apresentado na tela de visão geral do Bluetooth LE Explorer, junto com as demais propriedades associadas, como mostrado na Figura 59.

Figura 59 – Visão geral das propriedades da característica MSG no Bluetooth LE Explorer



Fonte: Autoria própria.

6.3 Característica Status do Botão

Para monitorar o estado do *push button* PRG da ESP32-LoRa, a propriedade READ foi habilitada na característica Status do Botão no firmware, de forma que seu valor pudesse ser atualizado em eventos de transferência dos dados. A cada pressionamento ou soltura do botão, por meio do comando *notify()* no servidor, o caractere correspondente ao estado da característica é atualizado. Quando o botão é pressionado (estado "LOW"), a característica assume o valor "1", enviado do servidor para o cliente. Em caso de ser solto (estado "HIGH"), o valor "0" é enviado.

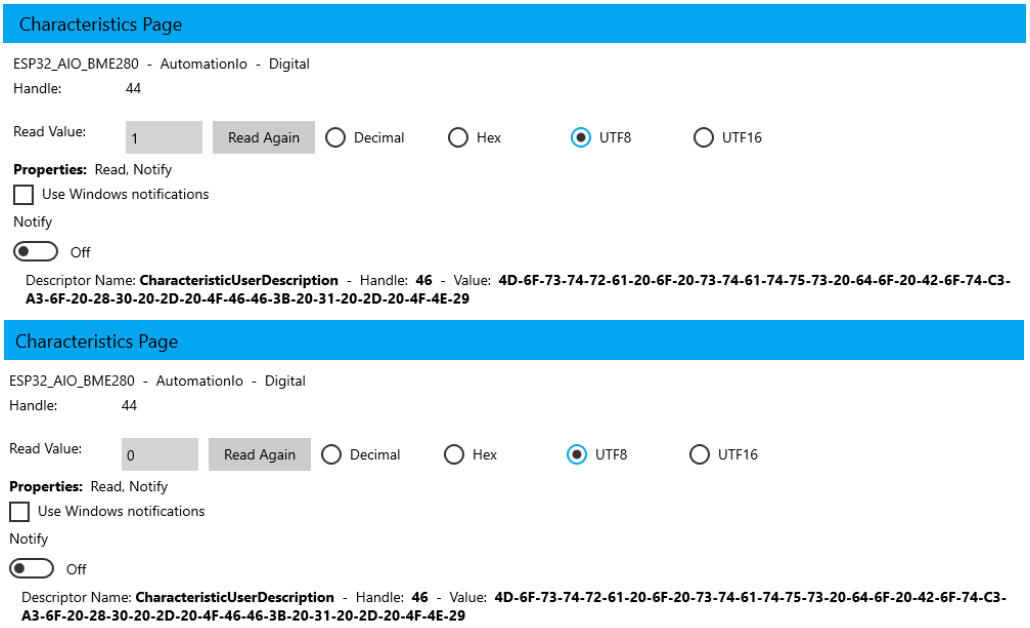
- **Monitoramento do status via MIT APP Inventor:** utilizando o aplicativo BLE AIO BME, o bloco *when .StringsReceived* permitiu a atualização do valor da característica Status do Botão a cada evento. Em caso do recebimento do valor string ["1"], o campo de texto *lbl_btn* recebe o valor "ON". Em caso de receber ["0"], o campo assume o valor "OFF", como evidenciado na Figura 60.

- **Monitoramento via Bluetooth LE Explorer:** no aplicativo Bluetooth LE Explorer, o usuário pode fazer a leitura dos valores "1" e "0" correspondentes aos valores associados à característica na página da característica em questão (Figura 61). O descritor da característica já indica a correspondência desses valores para os status LOW e HIGH do botão, como mostrado na Figura 62.

Figura 60 – Seção de monitoramento do status do botão na tela principal do BLE AIO BME

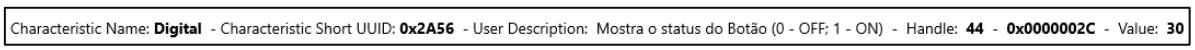
Fonte: Autoria própria.

Figura 61 – Página da característica Status do Botão permitindo a leitura do valor “1” (imagem acima) ou “0” (imagem abaixo) correspondente ao status do botão



Fonte: Autoria própria.

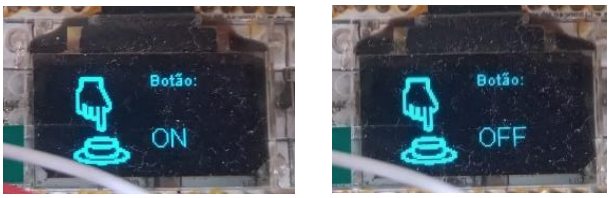
Figura 62 – Visão geral das propriedades da característica Status do Botão no Bluetooth LE Explorer



Fonte: Autoria própria.

No display OLED LCD, o envio das cadeias de caracteres únicas “1” ou “0” é demonstrado pela visualização do ícone e do valor, como ilustrado nas imagens da Figura 63.

Figura 63 – Valor da característica Status do Botão atualizado na tela do display OLED



Fonte: Autoria própria.

6.4 Características ambientais: temperatura, pressão, altitude e umidade

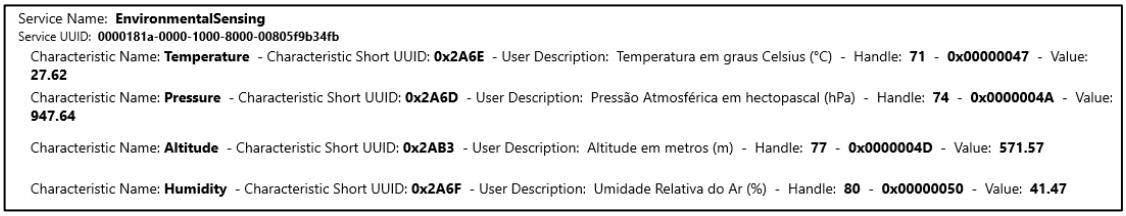
O monitoramento das características ambientais também foi permitido por meio do aplicativo BLE AIO BME e do Bluetooth LE Explorer devido à implementação das

propriedades READ e NOTIFY. A propriedade READ permitiu a atualização dos valores das características quando pressionado o botão “Atualizar” (*btn_read_data*) no aplicativo ou (Refresh) no Bluetooth LE Explorer. A propriedade NOTIFY permitiu a amostragem dos valores das características com a notificação intercalada da atualização de uma característica a cada três segundos, conforme programado no firmware.

- **Monitoramento das variáveis ambientais via MIT APP Inventor:** com o aplicativo BLE AIO BME, o bloco *when .StringsReceived* permitiu a atualização dos valores das características Temperatura, Pressão, Altitude e Umidade conforme o intervalo de atualização e envio de notificações para sobrescrever os valores anteriores ou através de solicitação do dispositivo cliente.
- **No Bluetooth LE Explorer:** para contemplar a atualização dos valores das características ambientais é necessário pressionar o botão *Refresh* (atualizar) na tela geral de visualização dos serviços ou o botão *Read Again* (ler novamente) na tela de visualização da característica, de modo a obter os valores mais atualizados presentes em cada uma das características até o momento.

A Figura 64 apresenta a seção da tela principal de serviços contendo as características do ESS e suas principais propriedades (nome, UUID, valor, handle e descritor de texto personalizado).

Figura 64 – Visão geral das propriedades das características ambientais no Bluetooth LE Explorer



Fonte: Autoria própria.

Na Figura 65, são indicadas as visualizações das características ambientais no display OLED LCD.

Figura 65 – Exibição das características ambientais na tela do display OLED

Fonte: Autoria própria.

A Figura 66 ilustra as características do Environmental Sensing Service e os campos referentes aos valores de cada característica. No canto superior direito, é destacado o botão de atualizar, por meio do qual é realizada uma solicitação de leitura para obtenção dos valores atualizados das características de forma imediata.

Figura 66 – Atualização das variáveis ambientais na tela principal do AIO BME280

Fonte: Autoria própria.

6.5 Testes de confiabilidade das propriedades

Também foram realizados testes modificando as permissões aplicadas às características ambientais a fim de validar o correto funcionamento das propriedades READ e NOTIFY utilizadas. Ao remover-se ambas propriedades de leitura (READ e NOTIFY) e inserir uma propriedade WRITE (que não habilita o modo de leitura para dispositivos clientes), ponderou-se que as notificações (através do comando *notify()*) ainda são enviadas periodicamente do dispositivo servidor para o cliente. Em contrapartida, a atualização no aplicativo BLE AIO BME não ocorre como esperado quando pressionado o botão “Atualizar”, indicando que a permissão de solicitações de leituras que podem ser feitas pelo cliente fica desabilitada.

7 CONCLUSÃO

Este trabalho teve como objetivo estudar os princípios da tecnologia Bluetooth Low Energy aplicada no monitoramento e controle sem fio de dispositivos próximos, explorando os conceitos necessários à comunicação entre um servidor e um cliente por meio da confecção de um sistema com topologia de rede ponto a ponto. Através da análise detalhada do funcionamento dos protocolos envolvidos, desde a modelagem de anúncios de dispositivos até a estruturação de perfis, serviços e características, desenvolveu-se uma aplicação que vem a servir de referência para o desenvolvimento de diversas outras correlatas dentro do âmbito educacional.

Conforme demonstrado com a aplicação prática do projeto, o uso de microcontroladores ESP32 para implementar sistemas de comunicação BLE mostrou-se uma alternativa de simples configuração e de baixo custo, indicando um bom potencial para aplicações que requerem comunicação sem fio com baixo consumo de energia. Apesar dos benefícios e vantagens, ainda ressaltam algumas limitações encontradas.

Com os testes realizados, foi constatado que o módulo BLECharacteristic incorporado na biblioteca BLEDevice possui algumas restrições, especialmente no que se refere à habilitação de propriedades. Observou-se que o comando *notify()* funcionou efetivamente, mesmo sem a definição das propriedades READ, NOTIFY ou INDICATE, permitindo a atualização de valores sem a necessidade de permissões de leitura explícitas.

Por outro lado, a ausência da propriedade READ impactou diretamente na capacidade de realizar requisições de leitura, comportamento este de acordo com os princípios da comunicação BLE. Dessa forma, sugere-se que ainda haja problemas e ambiguidades na implementação do Bluetooth Low Energy nas bibliotecas BLE para microcontroladores ESP32, que impactam sobretudo projetos nos quais existem preocupações de criptografia e segurança de acesso aos dados envolvidas. Apesar disso, para aplicações meramente educacionais, nas quais o intuito é facilitar a compreensão dos tópicos centrais da tecnologia, ela ainda é uma ferramenta didática e que oferece recursos suficientes para a sua implementação.

Adicionalmente, embora não tenha sido explorada a propriedade BROADCAST neste projeto, pode-se inferir que o conhecimento adquirido e as técnicas empregadas podem ser extrapolados para o entendimento e aplicação dessa propriedade em outros contextos, dada a similaridade dos processos de anúncio e publicidade de valores.

Nesse sentido, valida-se o uso do BLE utilizando o microcontrolador ESP32 para o desenvolvimento de uma ampla gama de projetos, nos quais geralmente o foco principal está associado à efetividade da transmissão dos dados, sem necessariamente serem relevantes os aspectos de segurança.

Em suma, este projeto serviu para aprimorar o método de ensino da tecnologia BLE na disciplina de Redes Industriais de Comunicação do Curso de Engenharia de Controle e Automação da UNESP Sorocaba, podendo inclusive servir de base para o desenvolvimento de trabalhos futuros na área. As limitações encontradas na biblioteca BLECharacteristic apontam para oportunidades de pesquisa e desenvolvimento que podem ser exploradas em projetos subsequentes, contribuindo para a melhoria contínua das ferramentas disponíveis e para a expansão do ecossistema de comunicação BLE.

REFERÊNCIAS

AFANEH, Mohammad. *Intro to Bluetooth low energy*. Novel Bits, 2018.

ANDROID PRO. *App Inventor: o que é, para que serve, tutoriais, vantagens e desvantagens. Android Pro*, 2024. Disponível em: https://www.androidpro.com.br/blog/desenvolvimento-android/app-inventor. Acesso em: 27 jun. 2024.

ALIBABA GROUP. *2 conjuntos ESP32 LoRa V3 placa de desenvolvimento 868MHz-915MHz SX1262 0.96 polegada display OLED BT + WiFi LoRa kit para Arduino IoT casa inteligente*. Disponível em: https://pt.aliexpress.com/item/1005005241276202.html. Acesso em: 27 ago. 2024.

BANAFEA, Mohammed; SHAYEA, Ibraheem; DIN, Jafri; AZMI, Marwan Hadri; ALASHBI, Abdulaziz; DARADEKH, Yousef Ibrahim; ALHAMMADI, Abdulraqeb. 6G Mobile Communication Technology: Requirements, Targets, Applications, Challenges, Advantages, and Opportunities. *Alexandria Engineering Journal*, v. 64, p. 245-274, 2023. ISSN 1110-0168. Disponível em: https://doi.org/10.1016/j.aej.2022.08.017. Acesso em: 8 jul. 2024.

BLUETOOTH SIG. *CORE Specification 5.1*. 2024. Disponível em: https://www.bluetooth.com/specifications/specs/core-specification-5-1/. Acesso em: 21 mar. 2024.

BLUETOOTH SIG. *The Bluetooth® Low Energy Primer*. Document Version 1.2.0. 15 mar. 2024. Disponível em: https://www.bluetooth.com/wp-content/uploads/2022/05/the-bluetooth-le-primer-v1.2.0.pdf. Acesso em: 29 jan. 2024.

BLUETOOTH Wireless Technology. *Bluetooth SIG*, 2024. Disponível em: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/. Acesso em: 8 abr. 2024.

ESPRESSIF SYSTEMS. *ESP32 Series Datasheet*. Version 4.6. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 20 abr. 2024.

ESPRESSIF SYSTEMS. *How to customize BLE services*. Disponível em: https://docs.espressif.com/projects/esp-at/en/latest/esp32/Compile_and_Develop/How_to_customize_BLE_services.html. Acesso em: 13 abr. 2024.

GOLDSMITH, Andrea. *Wireless communications*. Cambridge university press, 2005.

GUGGEDAL, Jan Tore. *Tracking and positioning of objects using a network of Bluetooth Low Energy devices*. 2018. Dissertação de Mestrado. NTNU.

ISLAM, Mohaiminul; JIN, Shangzhu. An overview research on wireless communication network. *Networks*, v. 5, n. 1, p. 19-28, 2019.

MIT App Inventor. *Extensions*. Disponível em: https://mit-cml.github.io/extensions/. Acesso em: 10 jul. 2024.

MOKOBLUE. *What is Bluetooth Mesh?*. Março 2021. Disponível em: https://www.mokoblue.com/pt/what-is-bluetooth-mesh/. Acesso em: 24 fev. 2024.

RANDOM NERD TUTORIALS. *ESP32 BLE Server and Client*. Disponível em: https://randomnerdtutorials.com/esp32-ble-server-client/. Acesso em: 1 fev. 2024.

RANDOM NERD TUTORIALS. *ESP32 BLE Server with Environmental Sensing Service*. Disponível em: https://randomnerdtutorials.com/esp32-ble-server-environmental-sensing-service/. Acesso em: 1 fev. 2024.

RANDOM NERD TUTORIALS. *ESP32 Bluetooth Low Energy (BLE) with Arduino IDE*. Disponível em: https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/#more-63505. Acesso em: 15 mai. 2024.

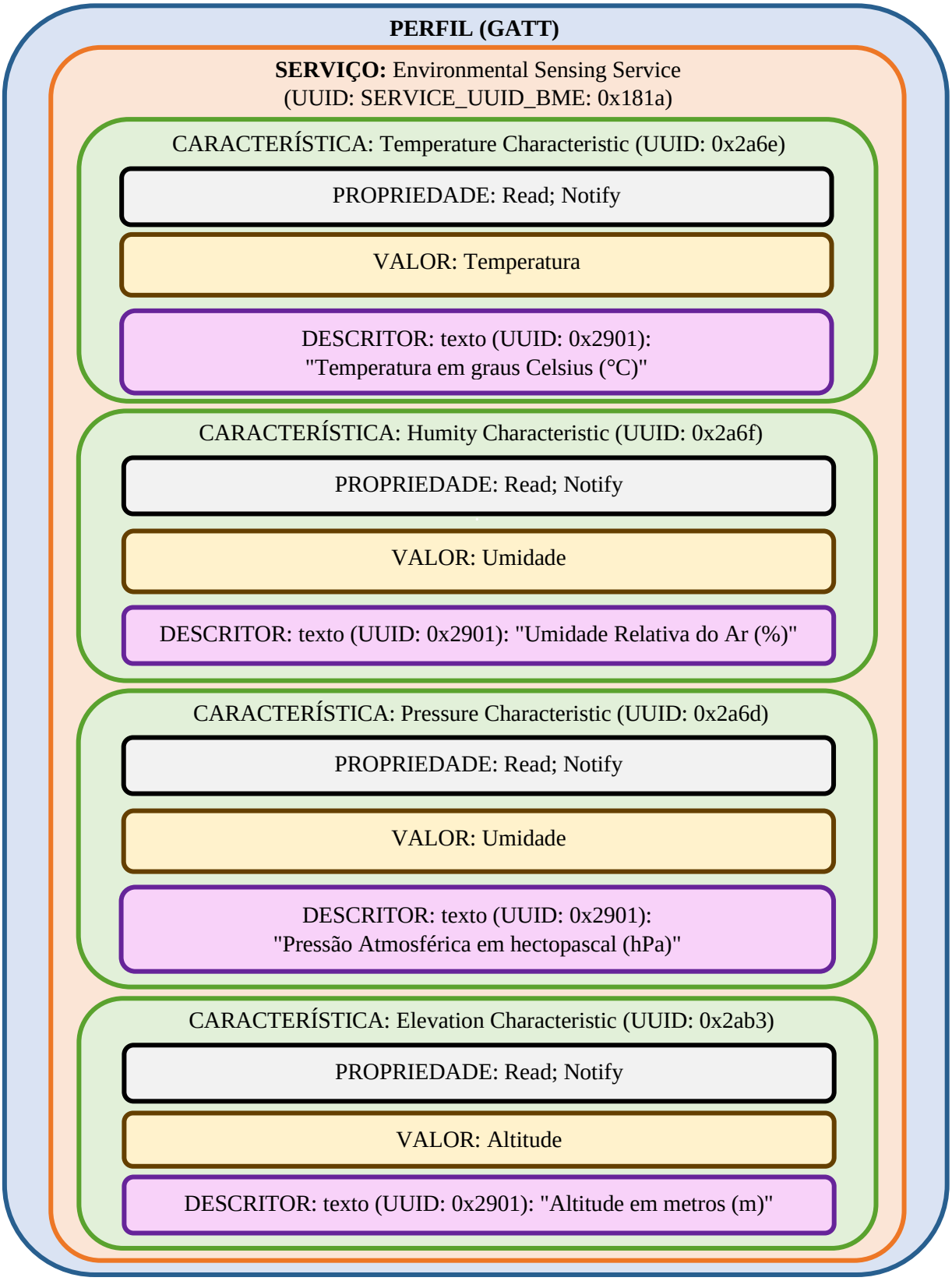
RENESAS. *Introdução — Manual de Referência da Plataforma SW DA14585/DA14531*. Disponível em: https://lpccs-docs.renesas.com/UM-B-119_DA14585-DA14531_SW_Platform_Reference/Introduction/Introduction.html#protocol-stack. Acesso em: 11 abr. 2024.

SHIN-TING, Wu. *Tópico 13*. 2019.

TRIGGS, C. W. R. *A little history of Bluetooth*. 2023. Disponível em: https://www.androidauthority.com/history-bluetooth-explained-846345/. Acesso em: 2 mai. 2024.

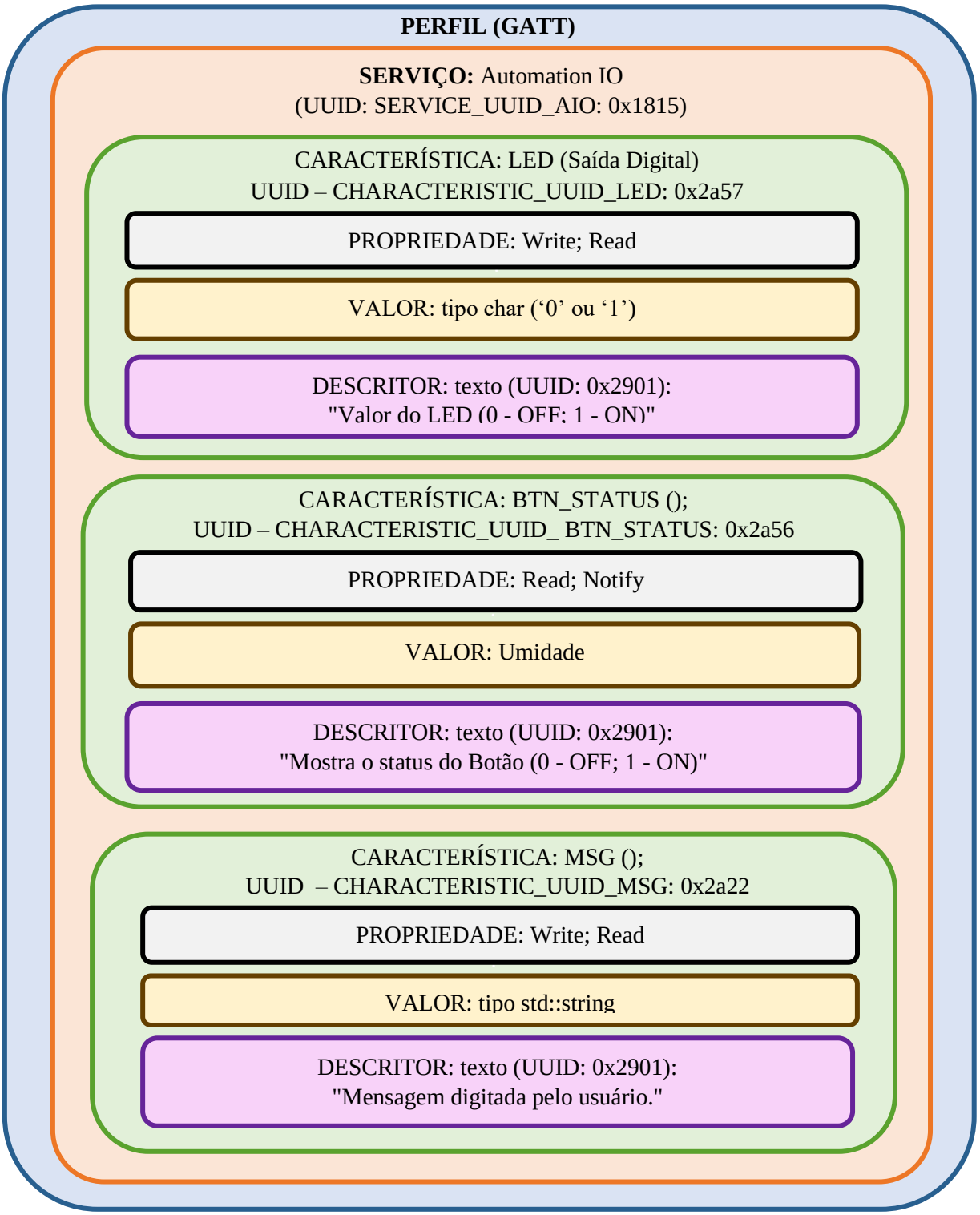
WU, Tina; MARTIN, Andrew. Bluetooth Low Energy Used for Memory Acquisition from Smart Health Care Devices. In: *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018. p. 1256-1261.

APÊNDICE A – DIAGRAMA DA ESTRUTURA GATT COM SERVIÇO ENVIRONMENTAL SENSING



Fonte: Autoria própria.

APÊNDICE B – DIAGRAMA DA ESTRUTURA GATT COM SERVIÇO AUTOMATION IO



Fonte: Autoria própria.

APÊNDICE C – IMPORTAÇÃO DE BIBLIOTECAS UTILIZADOS NO PROJETO

```
 3  PARTE 1. IMPORTAÇÃO DAS BIBLIOTECAS E ARQUIVOS UTILIZADOS*/
 6  #include <Wire.h>             //Biblioteca do I2C
 7  #include <Adafruit_BME280.h>  //Biblioteca do sensor BME
 8  #include <SSD1306.h>          //Driver do display OLED
 9  #include <BLEDevice.h>        //Biblioteca para uso do protocolo BLE
10  #include <BLE2902.h>          //Biblioteca para uso dos descritores
```

Fonte: Autoria própria.

APÊNDICE D – IMPORTAÇÃO DOS ARQUIVOS DE FRAME

```
12  #include "frame_btn.h"
13  #include "frame_msg.h"
14  #include "frame_led.h"
15  #include "frame_tem.h"
16  #include "frame_pre.h"
17  #include "frame_alt.h"
18  #include "frame_umi.h"
```

Fonte: Autoria própria.

APÊNDICE E – CONFIGURAÇÃO DOS PINOS DO DISPLAY OLED LCD

```
45 //Configurações do display LCD
46 #define OLED_I2C_ADDR 0x3c
47 #define OLED_RESET 16
48 #define OLED_SDA 4
49 #define OLED_SCL 15
50 SSD1306 display(OLED_I2C_ADDR, OLED_SDA, OLED_SCL);
```

Fonte: Autoria própria.

APÊNDICE F – CRIAÇÃO DOS PONTEIROS QUE REPRESENTAM O SERVIDOR, AS CARACTERÍSTICAS E OS DESCRITORES UTILIZADOS

```
80 BLEServer*         pServer                  = NULL;
81 BLECharacteristic* pCharacteristic_LED      = NULL;
82 BLECharacteristic* pCharacteristic_BTN_STATUS = NULL;
83 BLECharacteristic* pCharacteristic_MSG      = NULL;
84 BLECharacteristic* pCharacteristic_Temp     = NULL;
85 BLECharacteristic* pCharacteristic_Pre      = NULL;
86 BLECharacteristic* pCharacteristic_Alt      = NULL;
87 BLECharacteristic* pCharacteristic_Umi      = NULL;
88 BLEDescriptor*     pDesc_LED                = NULL;
89 BLEDescriptor*     pDesc_BTN_STATUS         = NULL;
90 BLEDescriptor*     pDesc_MSG                = NULL;
91 BLEDescriptor*     pDesc_Temp               = NULL;
92 BLEDescriptor*     pDesc_Pre                = NULL;
93 BLEDescriptor*     pDesc_Alt                = NULL;
94 BLEDescriptor*     pDesc_Umi                = NULL;
```

Fonte: Autoria própria.

APÊNDICE G – DECLARAÇÃO DE VARIÁVEIS DE CONTROLE E MONITORAMENTO DO SISTEMA

```
114 #define MSG_BUFFER_SIZE (50)
115 int buttonState = 0;      // estado atual do botão
116 int lastButtonState = 0;  // estado anterior do botão, inicialmente
    definido no código como LOW (0)
117 char msg[MSG_BUFFER_SIZE];// armazenar valor char da mensagem
118 int led_signal; // armazenar valor int do estado do led que será
    enviado do servidor para o cliente
119 int btn_signal; // armazenar valor int do estado do botão
120 char msg_temp[MSG_BUFFER_SIZE];// armazenar valor char da tempera-
    tura
121 char msg_pre[MSG_BUFFER_SIZE];// armazenar valor char da pressão
122 char msg_alt[MSG_BUFFER_SIZE];// armazenar valor char da altitude
123 char msg_umi[MSG_BUFFER_SIZE];// armazenar valor char da umidade
124 char sig[MSG_BUFFER_SIZE]; //sig será definido como um char e seu
    valor será obtido a partir da string StatusBtn
125
126 //std::string led;
127 int led; // armazenar valor int do estado do led que será recebido
    no servidor
128 int StatusBtn; //int correspondente ao estado do botão (LOW = 1 =
    ON = Pressionado; HIGH = 0 = OFF = Não pressionado)
```

Fonte: Autoria própria.

APÊNDICE H – DECLARAÇÃO DAS VARIÁVEIS DE POSIÇÃO DOS ELEMENTOS NA TELA DO DISPLAY

```
132 //definições de posição da imagem e variáveis usadas como parâmetro
    de largura (width) e altura (height)
133 #define pos_x 0
134 #define pos_y 0
135 //definições dos parâmetros do comprimento, altura e cadeia de bits
    da imagem que será mostrada no display
136 int frame_width;
137 int frame_height;
138 static const unsigned char * frame_bits;
139 String var_titulo; //variável usada como parâmetro do título/identi-
    ficação da grandeza/elemento mostrado, usado na função mostrar_da-
    dos_display_lcd
140 String var_valor;  //variável usada como parâmetro do valor da gran-
    deza/elemento mostrado, usado na função mostrar_dados_display_lcd
141 int pos_x_titulo;  //as posições x do título (pos_x_titulo) e do va-
    lor (pos_x_valor) foram configuradas para poderem variar para cada
    característica. As posições y foram configuradas com valores fixos,
    definidos diretamentes no corpo da função
142 int pos_x_valor;
```

Fonte: Autoria própria.

APÊNDICE I – DECLARAÇÃO DA ESTRUTURA PARA ARMAZENAR VALORES DAS VARIÁVEIS DO ESS

```
167 void inicializacao_bme_and_display(){
168   //Definição do display LCD (OLED_RESET) como saída
169   pinMode(OLED_RESET, OUTPUT);
171   //Início da operação de RESET = Definir inicialmente o display como
    apagado (sem sinal nenhum), para garantir que a tela esteja "limpa" ao
    iniciar a aplicação
172   digitalWrite(OLED_RESET, LOW);
173   vTaskDelay(50 / portTICK_PERIOD_MS); //similar a delay(50)
174 //Conclusão da operação de RESET = Inicialização dos pixels do display
175   digitalWrite(OLED_RESET, HIGH);
176   vTaskDelay(50 / portTICK_PERIOD_MS);
```

Fonte: Autoria própria.

APÊNDICE J – INICIALIZAÇÃO E CONFIGURAÇÃO DO DISPLAY OLED

```
178 display.init(); //comando necessário para
179 display.setFont(ArialMT_Plain_10); //Fonte padrão (Arial Plain tamanho
    10) para todos os textos que se sucedem a essa definição até a próxima
180 display.setTextAlignment(TEXT_ALIGN_LEFT);
182 if (!display.init()) {
183   Serial.println("Display indisponível!");
184 } else {
185   display.drawString(0, 0, "Display OLED - OK"); //posição x na tela
    do display = 0; posição y na tela do display = 0; texto = "Display
    OLED - OK"
186   Serial.println("Display OLED - OK!");
187   display.display();
188   vTaskDelay(1000 / portTICK_PERIOD_MS);
189 }
```

Fonte: Autoria própria.

APÊNDICE K – PARAMETRIZAÇÃO E COMANDOS INICIAIS DA FUNÇÃO “mostrar_dados_display_lcd”

```
227 void mostrar_dados_display_lcd(int posicao_x_imagem, int posicao_y_ima-
    gem, int posicao_x_nome, int posicao_x_valor, int largura_imagem, int
    altura_imagem, const unsigned char cadeia_bits[], String nome_grandeza,
    String valor_grandeza, bool long_text = false) {
228   display.clear();  //limpa tela
230   display.drawXbm(posicao_x_imagem, posicao_y_imagem, largura_imagem,
    altura_imagem, cadeia_bits); //mostra a imagem
```

Fonte: Autoria própria.

APÊNDICE L – CONFIGURAÇÕES PARA AMOSTRAGEM DO FRAME RESULTANTE NO DISPLAY OLED

```
231   display.setFont(ArialMT_Plain_10);
    //seta a fonte do que for escrito na sequência do código até que esse
    parâmetro seja novamente alterado
232   display.drawString(posicao_x_nome, 0, nome_grandeza);
    //texto do título, escrito em letra ArialMT_Plain_10, com x definido a
    cada característica e y fixo em 0, e o nome constatado.
233   if(long_text == false) display.setFont(ArialMT_Plain_16);
    //seta a fonte do que for escrito na sequência do código até que esse
    parâmetro seja novamente alterado
234   display.drawString(posicao_x_valor, 34, valor_grandeza);
    //texto do valor, escrito em letra ArialMT_Plain_16 com x definido a
    cada característica e y fixo em 34, e o valor constatado.
235   display.display();
    //após realizadas todas as configurações, mostra o frame resultante no
    display
236 }
```

Fonte: Autoria própria.

APÊNDICE M – FUNÇÃO “print_lcd_and_app()”

```
257 void print_LCD_and_APP(int type_var, const BME_Dados& valor){
258   float temp = 0;
259   float pres = 0;
260   float alt = 0;
261   float umi = 0;
```

Fonte: Autoria própria.

APÊNDICE N – DESCRIÇÃO DA FUNÇÃO “onWrite()” DA CLASSE LEDCALLBACKS

```
368   void onWrite(BLECharacteristic* pCharacteristic_LED){ //suporta um pedido de leitura do valor pelo dispositivo cliente (habilitado com a propriedade READ)
369 //usada para que o valor recebido do usuário (1 ou 0) possa modificar o estado do LED
370     led = std::stoi(pCharacteristic_LED->getValue()); //led é um inteiro obtido a partir de uma string "1" ou "0" recebida do dispositivo cliente.
371     Serial.print("O valor recebido é: ");
372     Serial.print(led);
373     Serial.println();
374       if (led == 1) {
375         digitalWrite(ONBOARD_LED, HIGH); //acende o LED
376         Serial.println("LED ON");
377         Serial.println();
378         var_valor = String("ON");
379       }
380       if (led == 0) {
381         digitalWrite(ONBOARD_LED, LOW); //apaga o LED
382         Serial.println("LED OFF");
383         Serial.println();
384         var_valor = String("OFF");
385       }
386       var_titulo = String("LED:");
387       //configurações e referências para exibição da característica pCharacteristic_LED no display
388       frame_width    =    frame_led_width;     //frame_led_width é o valor da largura do frame da imagem definido em "frame_led.h"
389       frame_height   =    frame_led_height;    //frame_led_height é o valor da altura do frame da imagem definido em "frame_led.h"
390       frame_bits     =    frame_led_bits;      //frame_led_bits é a cadeia de bits do frame da imagem definida em "frame_led.h"
391       pos_x_titulo   =    64; //posição x do título do LED no display (64 representa a quantidade de pixels)
392       pos_x_valor    =    64; //posição x do valor do LED no display (64 representa a quantidade de pixels)
393       mostrar_dados_display_lcd(pos_x, pos_y, pos_x_titulo, pos_x_valor, frame_width, frame_height, frame_bits, var_titulo, var_valor);
394       lastMsg = now; //o horário da última alteração é gravado para que se possa exibir a característica do LED no display por 3 segundos (se não for interrompido pela leitura/escrita de outra característica. Nesse caso, essa outra característica passa a ser exibida).
395   }
396 };
```

Fonte: Autoria própria.

APÊNDICE O – CONFIGURAÇÃO DE PARÂMETROS DE EXIBIÇÃO DOS DADOS NO OLED E CHAMADA DA FUNÇÃO “mostrar_dados_display_lcd()”

```
418     var_valor = rxValue.c_str();
419     var_titulo = String("Mensagem:");
420     frame_width    =   frame_mensagem_width;
421     frame_height   =   frame_mensagem_height;
422     frame_bits     =   frame_mensagem_bits;
423     pos_x_titulo   =   64;
424     pos_x_valor    =   64;
425     mostrar_dados_display_lcd(pos_x, pos_y, pos_x_titulo, pos_x_valor,
    frame_width, frame_height, frame_bits, var_titulo, var_valor, true);
426     lastMsg = now; //adicionado para permitir que o fluxo aguarde o
    tempo de 3 segundos mostrando a mensagem no display
427   }
428 };
```

Fonte: Autoria própria.

APÊNDICE P – BLOCOS DE FUNÇÕES UTILIZADOS NO MIT APP INVENTOR PARA CONFIGURAÇÃO DO PROTOCOLO BLE

Função	*Objetivo*
when – DeviceFound do set .ElementsFromString to .DeviceList	Adicionar um novo dispositivo disponível para conexão ao *ListPicker*.
call FoundDeviceName	Obter nome do dispositivo encontrado.
call FoundDeviceAdress	Obter endereço do dispositivo encontrado.
call BluetoothLE1.StartScanning	Ativar a varredura de dispositivos disponíveis para conexão.
call BluetoothLE1.StopScanning	Desativar a varredura de dispositivos para conexão.
call BluetoothLE1.ConnectWithAddress	Conectar ao dispositivo usando o endereço obtido.
when .Connected / when .Disconnected	Atribuir uma ação durante eventos de conexão ou desconexão do dispositivo cliente ao servidor.
BluetoothLE1.WriteStringsWithResponse	Enviar valores do LED e mensagem de texto.
when .StringsReceived	Usado para substituir valores exibidos das características quando são recebidos novos valores.
call .RegisterForStrings	Usado para liberar a transferência de dados do tipo *string* entre o dispositivo cliente e o dispositivo servidor para uma dada característica.
call .ReadStrings	Passar valores das características via BLE.
.DeviceFound	Verificar se está conectado antes de proceder com as ações. Caso contrário, mostrar alerta.

Fonte: Autoria própria.