Costa, José Miguel Peixoto da

Sistema universal de aquisição, processamento e monitorização de variáveis sensoriais com base em tecnologia IoT

http://hdl.handle.net/11067/7655

Metadados

Data de Publicação 2023

Resumo A Internet das Coisas (IoT) tem ganho cada vez mais importância na sociedade atual, permitindo a conexão e comunicação entre objetos do quotidiano, como eletrodomésticos, veículos, dispositivos médicos e até mesmo cidades inteiras, através da internet. Este avanço tecnológico tem proporcionado inúmeras vantagens, desde a automação de tarefas domésticas até à otimização de processos industriais. Nos últimos anos, o mercado IoT tem registado um crescimento significativo e é expectável que esta te...

The Internet of Things (IoT) has gained increasing importance in today's society, allowing the connection and communication between everyday objects, such as home appliances, vehicles, medical devices, and even entire cities, through the Internet. This technological advance has provided numerous advantages, from the automation of domestic tasks to the optimization of industrial processes. In recent years, the IoT market has seen significant growth, and it is expected that this trend will contin...

Palavras Chave Internet das coisas, Interface de Utilizador, Microcontroladores

Tipo masterThesis

Revisão de Pares no

Coleções [ULF-FET] Dissertações

UNIVERSIDADE LUSÍADA
Vila Nova de Famalicão

SISTEMA UNIVERSAL DE AQUISIÇÃO, PROCESSAMENTO E MONITORIZAÇÃO DE VARIÁVEIS SENSORIAIS COM BASE EM TECNOLOGIA IOT

José Miguel Peixoto da Costa

Dissertação para obtenção do Grau de Mestre em Engenharia Eletrónica e Informática

Vila Nova de Famalicão – julho 2023

UNIVERSIDADE LUSÍADA
Vila Nova de Famalicão

SISTEMA UNIVERSAL DE AQUISIÇÃO, PROCESSAMENTO E MONITORIZAÇÃO DE VARIÁVEIS SENSORIAIS COM BASE EM TECNOLOGIA IOT

José Miguel Peixoto da Costa

Orientadores: Professor Doutor Pedro Reis
Professor Doutor Vitor Pereira

Dissertação para obtenção do Grau de Mestre em Engenharia Eletrónica e Informática

Agradecimentos

O trabalho apresentado nesta dissertação foi realizado sob a orientação dos Professores Doutor Pedro Reis e Doutor Vitor Pereira, aos quais expresso o meu sincero agradecimento pela disponibilidade e conselhos transmitidos ao longo deste percurso.

Agradeço também ao Professor António Nicolau pela dedicação e prontidão demonstrada, disponibilizando-se sempre para atender aos pedidos solicitados.

A todos os meus colegas e amigos mais próximos que me acompanharam ao longo do percurso académico, gostaria de expressar o quão importante foi a vossa presença e boa disposição para tornar esta jornada ainda mais memorável.

Gostaria de expressar um agradecimento especial à minha Mãe, por me permitir manter-me concentrado e focado nos meus estudos e objetivos. A sua constante dedicação e apoio incondicional foram e continuam a ser fundamentais para o meu crescimento pessoal.

Por fim, um obrigado à Universidade Lusíada por todo o conhecimento e oportunidades que me proporcionou ao longo da minha jornada académica.

Resumo

A Internet das Coisas (IoT) tem ganho cada vez mais importância na sociedade atual, permitindo a conexão e comunicação entre objetos do quotidiano, como eletrodomésticos, veículos, dispositivos médicos e até mesmo cidades inteiras, através da internet. Este avanço tecnológico tem proporcionado inúmeras vantagens, desde a automação de tarefas domésticas até à otimização de processos industriais.

Nos últimos anos, o mercado IoT tem registado um crescimento significativo e é expectável que esta tendência se mantenha nos próximos anos. Contudo, o rápido crescimento deste mercado, aliado à ampla diversidade de tecnologias disponíveis para o desenvolvimento de soluções IoT, acarreta alguns desafios significativos para os seus utilizadores. Um desses desafios é a seleção das tecnologias, que se torna numa tarefa complexa que requer tempo e conhecimentos técnicos.

As plataformas IoT surgem com o propósito de facilitar a tarefa do utilizador no desenvolvimento e integração de soluções IoT. Estas plataformas fornecem uma infraestrutura composta por determinados recursos e componentes, que possibilitam a criação de uma camada intermediária entre os utilizadores e as suas soluções projetadas. No entanto, após um estudo realizado às plataformas IoT atualmente disponíveis no mercado, verificou-se uma considerável heterogeneidade entre as mesmas. Algumas plataformas oferecem apenas a infraestrutura necessária para interligar os utilizadores às suas soluções, enquanto outras, para além da infraestrutura, disponibilizam uma interface de utilizador (IU). Contudo, existe uma escassez de plataformas que ofereçam uma alternativa completa, que inclua hardware responsável a ser implementado no desenvolvimento da solução IoT.

Face à diversidade de tecnologias e à ausência de opções complementares às plataformas existentes, neste trabalho desenvolveu-se um sistema IoT. Este sistema é composto por uma plataforma, que por sua vez, é constituída por uma IU e uma infraestrutura que conecta os utilizadores às suas soluções projetadas. Além da plataforma, é fornecido também por este sistema um dispositivo, capaz de auxiliar e simplificar o processo de desenvolvimento de soluções IoT aos utilizadores.

O desenvolvimento do sistema IoT dividiu-se em dois blocos: plataforma e dispositivo, centrando-se mais no desenvolvimento do dispositivo, visto que é o componente de um sistema IoT menos explorado pelas opções existentes no mercado. Assim, além da plataforma, desenvolveu-se um dispositivo constituído por dois microcontroladores,

denominados Master e Slave MCU. Cada um destes microcontroladores possui específicas responsabilidades: o Master é responsável pela conectividade com a plataforma desenvolvida, enquanto o Slave executa o programa ou, por outras palavras, a solução desenvolvida pelo utilizador. Além de dois microcontroladores, este dispositivo possui múltiplas tecnologias de comunicação, entre as quais: Wi-Fi, Bluetooth, LoRa, I^2C, SPI e UART.

A utilização de dois microcontroladores possibilitou a conceção de um dispositivo "*Plug and Play*" (PnP), eliminando quaisquer preocupações dos utilizadores relacionadas com a conexão dos seus dispositivos à plataforma e, permitindo-lhes assim, focarem-se exclusivamente no desenvolvimento da solução. A divisão de tarefas entre microcontroladores possibilita um melhor aproveitamento dos seus recursos, que por si só já são limitados. Esta abordagem também viabiliza o surgimento de novas contingências, como a implementação de um ambiente de multilinguagem de programação, permitindo ao utilizador programar o Slave utilizando diferentes linguagens compatíveis com a plataforma (C, Python, JavaScript e uma variante de C++).

Com este modelo, os utilizadores podem conectar-se ao universo de desenvolvimento IoT, sem a necessidade de se preocuparem com questões relacionadas com a plataforma, uma vez que por si só, o sistema fornecido assume essa responsabilidade.

Palavras-chave: Internet das Coisas (IoT), Sistema IoT, Plataforma, Interface de Programação de Aplicações (API), Interface de Utilizador (IU), Dispositivo, Microcontrolador, Utilizadores, Soluções IoT

Abstract

The Internet of Things (IoT) has gained increasing importance in today's society, allowing the connection and communication between everyday objects, such as home appliances, vehicles, medical devices, and even entire cities, through the Internet. This technological advance has provided numerous advantages, from the automation of domestic tasks to the optimization of industrial processes.

In recent years, the IoT market has seen significant growth, and it is expected that this trend will continue in the coming years. However, the rapid growth of this market, combined with the wide range of technologies available for the development of IoT solutions, poses some significant challenges for its users. One of these challenges is the selection of technologies, which becomes a complex task that requires time and technical knowledge.

IoT platforms emerge with the purpose of facilitating the user's task in the development and integration of IoT solutions. These platforms provide an infrastructure consisting of specific resources and components, which enable the creation of an intermediate layer between users and their designed solutions. However, after a study of the IoT platforms currently available on the market, there was considerable heterogeneity between them. Some platforms only offer the necessary infrastructure to connect users to their solutions, while others, in addition to the infrastructure, provide a user interface. Nevertheless, there is a scarcity of platforms that provide a comprehensive solution encompassing responsible hardware to support the developed IoT solutions.

Given the diversity of technologies and the lack of complementary options to existing platforms, this work developed an IoT system. This system consists of a platform, which, in turn, comprises a user interface and an infrastructure that connects users to their designed solutions. In addition to the platform, this system also provides a device capable of assisting and simplifying the process of developing IoT solutions for users.

The development of the IoT system was divided into two blocks: platform and device, with a greater focus on the development of the device since it is the component of an IoT system that is less explored by the existing options in the market. Thus, in addition to the platform, a device consisting of two microcontrollers called the Master and Slave MCU, was developed. Each of these microcontrollers has specific responsibilities: the Master is responsible for connectivity with the developed platform, while the Slave executes the program or, in other words, the solution developed by the user. In addition to two

microcontrollers, this device is compatible with multiple communication technologies, including Wi-Fi, Bluetooth, LoRa, I2C, SPI, and UART.

The use of two microcontrollers enabled the design of a "Plug and Play" (PnP) device, eliminating any concerns users had with connecting their devices to the platform and thus allowing them to focus exclusively on developing the solution. The division of tasks between microcontrollers allows for better use of their resources, which are inherently limited. This approach also makes possible the emergence of new contingencies, such as the implementation of a multilingual programming environment, allowing the user to program the Slave using different languages compatible with the platform (C, Python, JavaScript, and a variant of C++).

With this model, users can connect to the IoT development universe without worrying about platform-related issues, as the provided system itself takes on that responsibility.

Keywords: Internet of Things (IoT), IoT System, Platform, Application Programming Interface (API), User Interface, Device, Microcontroller, Users, IoT Solutions

Índice

Índice de Figuras

Índice de Tabelas

Abreviaturas, Acrónimos e Siglas

AMQP	*Advanced Message Queuing Protocol* (Protocolo de Fila de Mensagens Avançadas)
aPaaS	*Application Platform as a Service* (Plataforma de Aplicação como Serviço)
API	*Application Programming Interface* (Interface de Programação de Aplicação)
AWS CLI	*Amazon Web Services Command Line Interface*
AWS IoT	*Amazon Web Services IoT*
BD	Base de dados
CoAP	*Constrained Application Protocol*
CSS	*Chirp Spread Spectrum* (Espectro Espalhado Chirp)
DIY	*Do it yourself* (Faça por si mesmo)
DoS	*Denial of Service* (Ataques de Negação de Serviço)
GFSK	*Gaussian Frequency Shift Keying*
GPIO	*General Purpose Input/Output* (Entrada/Saída de Uso Geral)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HTTPS	*Hyper Text Transfer Protocol Secure* (Protocolo de Transferência de Hipertexto Seguro)
I^2C	*Inter-Integrated Circuit*
ID	*Identifier* (Identificador)
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
IETF	*Internet Engineering Task Force* (Força-Tarefa de Engenharia da Internet)
IoT	*Internet of Things* (Internet das Coisas)
IoT-A	*IoT Architectural Reference Model* (Modelo de Referência de Arquitetura IoT)
IP	*Internet Protocol* (Protocolo de Internet)

ISM	*Industrial, Scientific, and medical* (Industrial, Científico e Médico)
JSON	*JavaScript Object Notation* (Notação de Objetos JavaScript)
JWT	*JSON Web Token*
LDAP	*Lightweight Directory Access Protocol* (Protocolo de Acesso a Diretório Leve)
LoRaWAN	*Long Range Wide-Area Network* (Rede de Área Ampla de Longo Alcance)
M2M	*Machine-to-Machine* (Máquina para Máquina)
MAC	*Media Access Control* (Controlo de Acesso de Media)
MCU	*Microcontroller Unit* (Unidade de Microcontrolador)
MQTT	*Message Queuing Telemetry Transport* (Transporte de Telemetria de Mensagens em Fila)
NAT	*Network Address Translation* (Tradução de Endereço de Rede)
NVS	*Non-Volatile Storage* (Armazenamento Não Volátil)
ONU	Organização das Nações Unidas
PaaS	*Platform as a Service* (Plataforma como Serviço)
PLA	*Polylactic Acid* (Ácido Polilático)
PnP	*Plug and Play* (Ligar e Utilizar)
QoS	*Quality of Service* (Qualidade de Serviço)
RESTful	*Representational State Transfer* (Transferência de Estado Representativo)
RFID	*Radio Frequency Identification* (Identificação por Radiofrequência)
SDK	*Software Development Kit* (Kit de Desenvolvimento de Software)
SDN	*Software-Defined Networking* (Redes Definidas por Software)
SPI	*Serial Peripheral Interface*
SRDs	*Short Range Devices* (Dispositivos de Curto Alcance)
TCP	*Transmission Control Protocol* (Protocolo de Controlo e Transmissão)
TLS	*Transport Layer Security* (Segurança de Camada de Transporte)

UART	*Universal Asynchronous Receiver-Transmitter*
IU	Interface de Utilizador
URI	*Universal Resource Identifier* (Identificador Uniforme de Recurso)
UUID	*Universally Unique Identifier* (Identificador Único Universal)
WAN	*Wide Area Network* (Rede de Longa Distância)
WS	*WebSocket*
WSS	*WebSocket Secure*

1 Introdução

Este capítulo contextualiza o tema do presente trabalho, apresentando uma visão geral sobre o mesmo e as suas principais motivações, bem como os objetivos desta dissertação.

1.1 Enquadramento

A Internet evoluiu significativamente nas últimas décadas, tornando-se uma tecnologia determinante da Era da Informação [1]. O seu crescimento gradual conduziu a grandes oportunidades sociais e económicas, com um enorme potencial para gerar eficiências que produzem, num vasto número de áreas, um aumento da qualidade dos serviços de interesse social, promovendo assim o crescimento económico sustentável com ganhos de produtividade notáveis [2]. Os benefícios e potencialidades associados a esta rede global justificam a sua acentuada utilização, resultando num aumento do número de dispositivos conectados à Internet. Dentro deste conjunto de dispositivos incluem-se: computadores pessoais, tablets, smartphones, sistemas embebidos, entre outros. A maioria destes dispositivos dispõe de diferentes sensores e atuadores que lhes permitem adquirir informações, realizar cálculos, tomar decisões e transmitir dados pela internet, possibilitando assim a interação entre mundo digital e o mundo físico [3], [4]. A junção destes dispositivos – que numa outra perspetiva podem ser vistos como objetos inteligentes –, integrados numa rede que interliga vários utilizadores através de diferentes tecnologias, permitiu o aparecimento de um novo conceito tecnológico: '*Internet of Things*' (IoT).

Não existe uma definição padronizada em relação ao conceito '*Internet of Things*' [5]. Consta-se que o respetivo conceito tecnológico foi designado pela primeira vez numa apresentação feita por Kevin Ashton em 1999 – investigador britânico do Instituto de Tecnologia de Massachusetts (MIT) em Cambridge – no contexto de etiquetar eletronicamente os produtos da empresa *Procter & Gamble (P&G)*, com o objetivo de facilitar a logística da cadeia de produção da empresa em questão, através de identificadores de radiofrequência (RFID) [6]. Contudo, ao longo dos últimos anos, com o desenvolvimento deste conceito, a definição tornou-se mais inclusiva, cobrindo hoje uma ampla gama de aplicações em vários setores, como na saúde, nos serviços públicos e transportes. Atualmente, '*Internet of Things*' ou Internet das Coisas, pode ser definido como uma infraestrutura global destinada à sociedade da informação, que concede serviços avançados

através da interligação de objetos físicos e virtuais, utilizando tecnologias de informação e comunicação interoperáveis existentes ou em constante evolução [7].

Ainda que a sua definição tenha sofrido algumas alterações, resultantes da evolução tecnológica, o objetivo principal permanece: fazer um computador adquirir informações sem o auxílio da intervenção humana, permitindo a troca de dados entre o mundo digital e o real por meio de uma interligação autónoma e segura [3], [8].

A um ritmo sem precedentes, a quantidade de dispositivos ligados à Internet tem vindo aumentar ano após ano. Em 2010, o número de dispositivos conectados à Internet ultrapassou a população mundial [9]. Segundo os dados da *IoT Analytics* [10], é possível verificar que, desde 2010, além de existir uma tendência positiva em relação ao crescimento do número de dispositivos conectados à internet, é possível também concluir que o número de dispositivos IoT[1] conectados têm aumentado em relação aos dispositivos não IoT[2]. No final de 2020, pela primeira vez, dos 21,7 milhares de milhões de dispositivos conectados em todo o mundo, 11,3 milhares de milhões (ou seja, 52%) eram dispositivos IoT [10], [11]. Segundo as projeções da *IoT Analytics*, apresentadas na Figura 1, prevê-se para 2023 um crescimento de 16% face ao ano anterior, estimando-se que cerca de 16,7 milhares de milhões de dispositivos IoT estarão conectados até ao final do ano. Segundo esta análise, em 2027 espera-se que haja mais de 29 milhares de milhões de dispositivos IoT conectados [11].

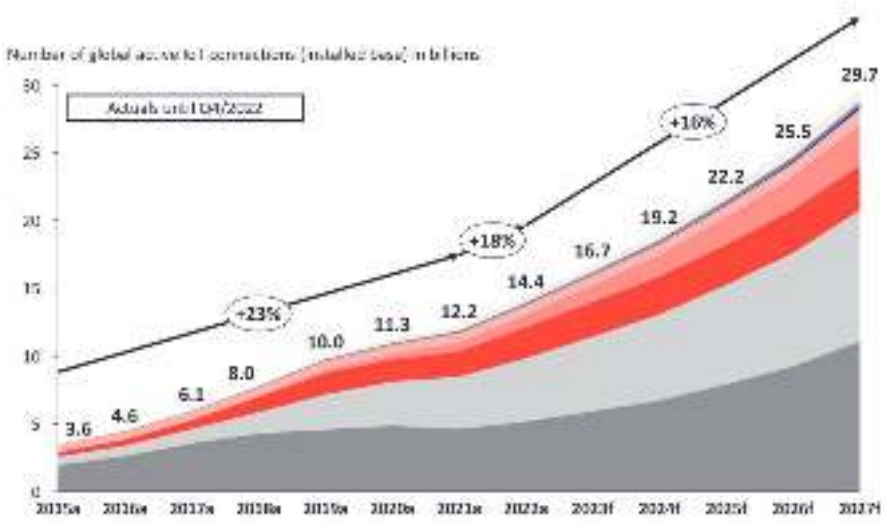


Figura 1 - Total de dispositivos IoT e não IoT conectados à internet desde 2015 e até 2027 (Adaptado de [11])

[1] Dispositivos que não dependem da interação humana para comunicar com outros dispositivos. Alguns exemplos de dispositivos IoT: câmaras de segurança, dispositivos domésticos, sensores baseados em tecnologia IoT, etc. [81].

[2] Dispositivos que depende da interação humana para comunicar com outros dispositivos. Alguns exemplos de dispositivos não IoT: *desktop*, *laptop*, *smartphone*.

Apesar da pandemia COVID-19, o mercado associado à Internet das Coisas continuou em crescimento. Impulsionada por novos padrões tecnológicos, como o 5G, espera-se que até 2025 haja mais de 30 milhares de milhões de conexões IoT, o que perfaz quase 4 dispositivos IoT por pessoa [10].

Na última década, mediante a evolução tecnológica, o leque de aplicações associadas à Internet das Coisas aumentou consideravelmente, existindo hoje vários tipos de sistemas IoT para diferentes contextos [12]. O crescimento e adoção dos sistemas IoT permitiu também o crescimento de outros mercados, como, por exemplo, o mercado das casas inteligentes (*Smart Homes*) e das cidades inteligentes (*Smart Cities*). Segundo o *MordorIntelligence*, o mercado das casas inteligentes foi avaliado em 79,1 mil milhões de dólares em 2020, e deve atingir 313,9 mil milhões dólares até 2026. O maior mercado das casas inteligentes está localizado na América do Norte, prevendo-se que seja este subcontinente que mais cresça até 2026. Ao contrário do que se possa considerar, neste mercado, o principal efeito sentido pela pandemia COVID-19 não foi a desaceleração do seu crescimento, mas sim uma mudança de paradigma nos espaços residenciais. Isto deve-se ao conjunto totalmente novo de comodidades que ressurgiu – como, por exemplo, o distanciamento social – caracterizando-se assim, como o 'novo normal' [13]. Em relação às cidades inteligentes, segundo a mesma fonte, o mercado foi avaliado em 739,8 mil milhões dólares em 2020 e deverá superar os 2 biliões de dólares até 2026, estando o maior mercado localizado na Europa. De salientar que nos próximos anos é esperado que a região Ásia-Pacífico testemunhe o maior crescimento deste mercado, sendo a China e a Índia os países proeminentes, justificando-se assim o forte investimento que estes dois países têm feito no contexto das cidades inteligentes [14].

O mercado IoT cresceu e segundo as projeções apresentadas na Figura 1, deverá continuar nesse sentido nos próximos anos. Contudo, um forte crescimento num curto espaço de tempo poderá implicar alguns desafios. Presentemente, este conceito tecnológico pode ser implementado em inúmeros setores; porém, as aplicações que sustentam o seu funcionamento podem ser desenvolvidas a partir de tecnologias de uma enorme heterogeneidade. Tal escala de diferentes abordagens gera um ponto de divergência: se, por um lado, existe um leque considerável de alternativas disponíveis a serem exploradas, por outro, uma ampla diversidade de opções faz com que o processo de seleção se torne numa árdua tarefa. Além de tempo, é necessário também conhecimento técnico para entender e

comparar os diferentes conceitos, de modo a optar pelas tecnologias mais adequadas para um determinado campo de aplicação [12].

Uma solução que poderá ajudar a ultrapassar esta adversidade são as plataformas IoT. Estas plataformas podem ser vistas como uma infraestrutura, que se comporta como um *middleware*[3], capaz de fornecer um conjunto de recursos e componentes que facilitam a tarefa do utilizador no desenvolvimento e na integração de uma solução IoT [15].

Estima-se que existam mais de 500 plataformas IoT [16] e, de acordo com os dados da *IoT Analytics* [17], comprova-se que o mercado das plataformas IoT tem vindo a solidificar-se. Na Figura 2 é possível observar que entre 2015 e 2019 o número de plataformas oferecidas pelo mercado mais que duplicou. Posteriormente, no período de 2019 a 2021, houve um ligeiro decréscimo de 1% que se pode traduzir num início de consolidação deste mercado.

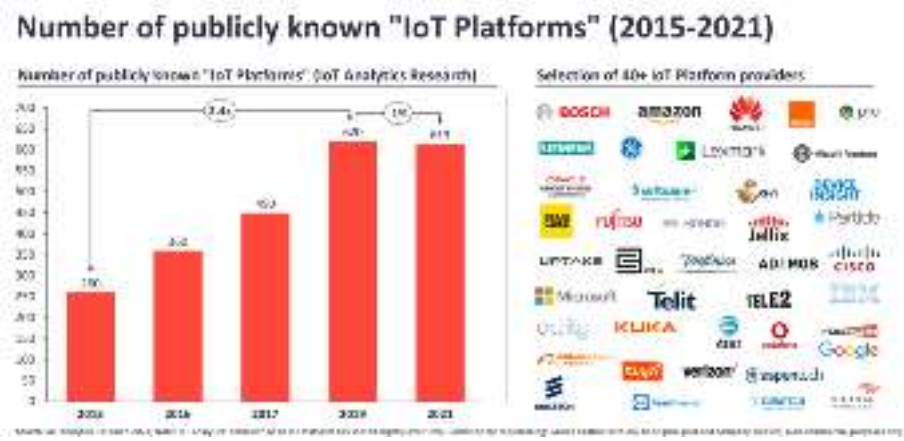


Figura 2 - Número de plataformas IoT presentes no mercado entre 2015 e 2021 [17]

Existem alguns indícios que apontam para a consolidação deste mercado, entre os quais se destacam: (1) cada vez mais este mercado está a centrar-se em apenas 10 empresas (Microsoft, AWS, Google Cloud, Alibaba, entre outras), que controlam 68% da sua totalidade; (2) algumas empresas que forneciam este tipo de serviços alteraram o seu modelo de negócio, outras agregaram-se ou então deixaram de existir. Ainda assim, e dado o forte crescimento esperado na adoção de dispositivos IoT, é expectável que o mercado das plataformas IoT ainda sofra um aumento substancial, com uma taxa de crescimento anual composta de 33% no período de 2020 a 2026 [17].

[3] Software que atua como um intermediário entre duas ou mais aplicações, facilitando, desta forma, a interação entre ambos.

As plataformas IoT são constituídas por importantes ferramentas, que são um precioso auxílio para o utilizador na construção de uma solução IoT [15]. Contudo, a existente falta de padronização no interior do ecossistema da Internet das Coisas afeta também as plataformas IoT, fazendo com que um dos maiores desafios seja encontrar uma plataforma adequada para um determinado campo de aplicação. Estes desafios são caracterizados pelos seguintes aspetos: (1) uma enorme diversificação de tecnologias, conceitos, terminologias utilizadas ou arquiteturas subjacentes, mesmo em plataformas de utilidade semelhante; (2) os próprios dispositivos, responsáveis por processar todos os dados adquiridos pelos sensores, dependendo da plataforma, podem gerar algumas incompatibilidades; (3) por parte do utilizador, é necessário conhecimento técnico que possibilite a escolha de uma plataforma que se adeque à solução que pretenda projetar. Estes inconvenientes, fazem com que o processo de seleção e adaptação por parte do utilizador seja lento e exaustivo, o que para alguns, como utilizadores menos experientes na área de tecnologia da informação, poderá ser visto como um impedimento para a criação da sua própria solução IoT [18].

1.2 Motivação e Objetivos

A evolução tecnológica fez com que o universo da Internet das Coisas passasse a ter uma grande diversidade de aplicações (agricultura, saúde, transportes, automação industrial ou residencial, entre muitas outras áreas), que não só contribuem para o aumento da qualidade de vida dos seus utilizadores, como também para o crescimento económico [19]. As suas infindas potencialidades de aplicação traduzem-se em necessidades sem fim à vista, fazendo com que o mercado não consiga satisfazer todas as necessidades dos seus consumidores. Tudo isto acaba por incentivar os utilizadores a se adaptarem – ou até mesmo a criarem – as suas soluções IoT, de acordo com as suas necessidades pessoais. Consequentemente, esta constante necessidade por parte dos entusiastas deste ecossistema, facilitou a formação de uma forte e ampla comunidade ao redor da Internet das Coisas que vai desde amadores até a investigadores. A cultura *do-it-yourself* (DIY) tornou-se efetivamente num ponto crucial no desenvolvimento IoT [20].

A agilização do processo de desenvolvimento de uma solução IoT tem o potencial de estimular os utilizadores menos experientes a juntarem-se ao universo da Internet das Coisas. Além do mais, poderá ainda servir de ponto de partida para outros rumos, como, por exemplo, a estimulação da curiosidade pela programação, ou até mesmo o ensino da

programação na educação. A formação do pensamento computacional (*Computational Thinking*) é um importante desafio na educação para os próximos anos, permitindo o desenvolvimento de habilidades relacionadas com o processo de resolução de problemas, criatividade e pensamento crítico. Nos últimos anos, um pouco por todo o mundo, as escolas têm apostado cada vez mais neste tipo de formação, existindo esforços para introduzi-la cada vez mais cedo [21].

Um dos principais desafios que relaciona a tecnologia e o meio-ambiente é a gestão e a sustentabilidade dos resíduos eletrónicos (*E-Waste*). O consumo exponencial de equipamentos digitais e elétricos tem consequentemente aumentado a quantidade de lixo eletrónico no planeta [22]. Segundo os dados da ONU [23], em 2019 foi produzido um valor recorde de 53,6 milhões de toneladas de resíduos de equipamentos elétricos e eletrónicos. Dessa quantidade, apenas 17,4% foi reciclado. Um baixo reaproveitamento deste tipo de resíduos é significativamente prejudicial para o meio-ambiente, colocando em risco a saúde pública, o ambiente, e a sustentabilidade do planeta. É essencial avaliar o impacto desta prática e tomar medidas em relação a esta situação. Em 2020, o Parlamento Europeu concordou em promover a sustentabilidade através de incentivos à reutilização e reparação. Esses incentivos passam pela uniformização dos carregadores para dispositivos móveis, o prolongamento das garantias e o direito à reparação dos produtos eletrónicos, através da disponibilização de peças sobresselentes por um determinado período [24]. Estas medidas entraram em vigor em Portugal a 1 de janeiro de 2022[4].

Contribuir para a sustentabilidade ambiental é essencial e, para tal, as entidades governamentais têm implementado medidas para tentar contornar o fluxo crescente e descontrolado de resíduos eletrónicos dos últimos tempos. Além da reutilização e da reciclagem, outra forma de combater o lixo eletrónico é pela implementação de múltiplas funções num único produto eletrónico[5]. Se, através de um produto, o utilizador, consoante as próprias necessidades pessoais, conseguir facilmente moldá-lo para diferentes casos de uso, não haverá, desta forma, a necessidade de adquirir outro produto para uma distinta função. Para ser possível tal aproveitamento, é necessário fornecer e potencializar determinados recursos que acompanhem o avanço tecnológico, e que contribuem no processo de flexibilização do respetivo produto. Um exemplo de um produto deste tipo são

[4] Decreto-Lei n.º 84/2021. https://dre.pt/dre/detalhe/decreto-lei/84-2021-172938301
[5] Sustainability at Harvard | 6 ways to minimize your e-waste. https://green.harvard.edu/tools-resources/how/6-ways-minimize-your-e-waste

as plataformas oferecidas pelo mercado que simplificam e agilizam o processo de desenvolvimento e integração de soluções IoT [20]. Para além de contribuírem para a criação de uma determinada solução, permitem também que o utilizador adapte a solução desenvolvida a necessidades futuras.

Tal como foi referido na secção anterior, numerosas e diversificadas plataformas IoT estão disponíveis no mercado. Contudo, os utilizadores deste tipo de plataforma costumam deparar-se com dificuldades na respetiva seleção. Ainda que o mercado disponha de uma enorme quantidade de plataformas, a maioria delas apenas oferece soluções parciais, podendo até não ser o suficiente para se enquadrarem ao nível de uma plataforma IoT. Por outro lado, as plataformas que oferecem soluções mais completas normalmente não cooperam entre si, o que, consequentemente, provoca a adoção de diferentes conceitos, terminologias e tecnologias para propósitos análogos [25]. Para a sua utilização, além de ser necessário entender alguns fundamentos relacionados com a área da tecnologia da informação, o utilizador deve ainda possuir conhecimento prévio em relação à disparidade de tecnologias, conceitos e arquiteturas, adotadas nas próprias plataformas. A indefinição ao seu redor pode resultar no afastamento de possíveis entusiastas da área da Internet das Coisas.

A presente dissertação de mestrado tem como objetivo principal estudar as características e a composição das plataformas presentes no mercado IoT, identificando as suas mais-valias e limitações, com o propósito de desenvolver uma proposta que supere os obstáculos identificados e torne o processo de desenvolvimento de soluções mais simples e acessível. Inicialmente, pretende-se realizar uma análise detalhada às principais plataformas do mercado, de modo a identificar quais aspetos podem ser melhorados. Com base no resultado desta avaliação, o objetivo é projetar e construir um sistema que sirva como prova de conceito. Assim, pretende-se implementar as melhorias identificadas e aferir a sua eficácia na prática, para que, no fim, se proceda à avaliação da viabilidade das mesmas. Em síntese, o sistema a desenvolver deve cumprir os seguintes objetivos:

1. Projetar e desenvolver um dispositivo de aquisição de dados, com suporte a múltiplas tecnologias de comunicação (Wi-Fi, Bluetooth e pelo menos uma tecnologia de longo alcance) que permita ao utilizador criar e executar código, agregando a capacidade sensorial, de forma a solucionar um determinado problema;

2. Fornecer condições de portabilidade, desenvolvendo uma aplicação móvel que permita ao utilizador efetuar uma comunicação bidirecional com o dispositivo de aquisição de dados, em qualquer momento e em qualquer lugar com ligação à internet.
3. Introduzir mecanismos que facilitem o uso do sistema pelo utilizador como, por exemplo, proporcionar um ambiente de multilinguagem de programação, possibilitando a programação do dispositivo mediante diferentes linguagens de programação disponíveis na plataforma.
4. Implementação das Interfaces de Programação de Aplicação (APIs) da plataforma que possibilitem a integração e comunicação entre todos os componentes do sistema, destacando-se em particular a integração entre dispositivos e utilizadores.

1.3 Estrutura da Dissertação

Esta dissertação encontra-se dividida em 5 capítulos.

Neste capítulo introdutório, efetua-se o enquadramento e as motivações do trabalho, estabelecendo-se os objetivos que se pretende alcançar.

No segundo capítulo é abordado o enquadramento teórico do presente trabalho, concentrando-se inicialmente no relacionamento entre os modelos e arquiteturas de referência no contexto da Internet das Coisas. São também apresentadas e comparadas algumas plataformas IoT disponíveis no mercado, discutindo-se ainda, temas relacionados com protocolos e tecnologias de comunicação de longo alcance. No final é apresentado o microcontrolador utilizado na conceção do dispositivo pertencente ao sistema IoT a implementar.

No terceiro capítulo é descrita a metodologia para alcançar o sistema a projetar, identificando-se as possíveis dificuldades que deverão ser encaradas, realizando-se uma análise das abordagens e técnicas a serem utilizadas nos dois principais componentes do sistema (dispositivos e plataforma), de modo a alcançar os objetivos propostos.

No quarto capítulo é descrito o desenvolvimento e funcionamento do sistema projetado, bem como a apresentação dos resultados obtidos. Finalmente, no quinto capítulo, são expostas as principais conclusões e delineadas algumas sugestões para trabalhos futuros.

2 Tecnologias IoT

Um sistema de aquisição, processamento e monitorização de variáveis sensoriais com base em tecnologia IoT, ou simplesmente sistema IoT, deve ser constituído por: (1) hardware, como sensores ou outros tipos dispositivos; (2) conectividade, garantindo que os dados adquiridos pelo hardware sejam transmitidos para a plataforma; (3) APIs e bases de dados (BDs), responsáveis por processar e armazenar os dados enviados pelos dispositivos, tomando decisões sobre os mesmos; (4) IU, para garantir que os utilizadores interajam com os componentes do sistema. Tal como demonstra a Figura 3, e contrariamente àquilo que habitualmente se deduz, as plataformas IoT são responsáveis por conectar todos os componentes presente num sistema IoT, enquadrando-se no ponto três (3) e quatro (4) anteriormente mencionados [26].

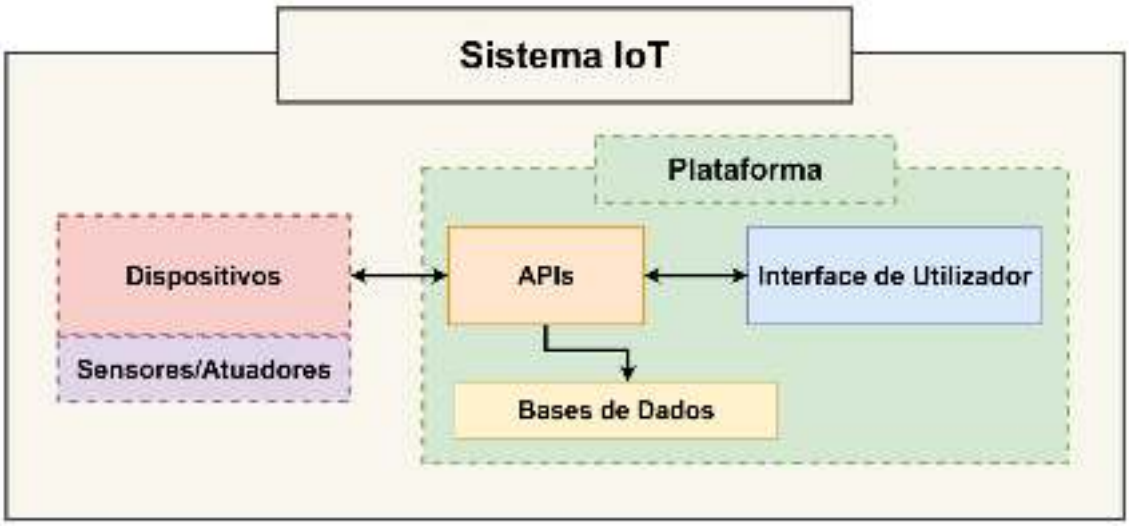


Figura 3 - Composição de um sistema baseado em tecnologia IoT

Ao longo deste capítulo, dada a existente indefinição em torno das plataformas, abordar-se-á os componentes que constituem as mesmas. Inicialmente faz-se uma análise dos modelos e arquiteturas de referências utilizadas nas plataformas até à sua conceção final. Depois, são apresentadas e comparadas algumas das maiores plataformas do mercado, de modo a perceber o que cada uma pode oferecer. Nos subcapítulos finais, são estudadas algumas tecnologias de comunicação e de hardware complementares para a elaboração do sistema proposto.

2.1 Modelos de Referência no domínio da Internet das Coisas

Um modelo de referência pode traduzir-se num paradigma composto por um conjunto de conceitos e de princípios que devem ser independentes de padrões, tecnologias, implementações ou de qualquer outro detalhe. O seu objetivo consiste em facilitar a identificação dos elementos que compõe um sistema num domínio específico, sendo comummente utilizado para fins didáticos. Por sua vez, uma arquitetura de referência pode derivar de um, ou mais, modelos de referência. O seu propósito passa pela definição de padrões arquiteturais, boas práticas de desenvolvimento e componentes de software e hardware necessários na construção de uma arquitetura específica. Deste modo, as arquiteturas de referência podem ser utilizadas como um meio de padronização para permitir a interoperabilidade entre os sistemas ou os componentes que compõem um sistema [27].

A Figura 4 demonstra a relação entre modelos de referência e arquiteturas de referência. Um ou mais modelos de referência podem servir de base comum no desenvolvimento de uma determinada arquitetura de referência, que, por conseguinte, reproduzirá uma arquitetura específica que poderá ser adotada na criação de um sistema. De salientar ainda que, embora não seja obrigatório, é recomendável que as arquiteturas de referência sejam estabelecidas com base em modelos de referência.

Figura 4 - Relação entre modelos de referência, arquiteturas de referência e arquiteturas (Adaptado de [27])

No domínio da Internet das Coisas, não há consenso em relação à padronização de um modelo com o propósito de fornecer uma estrutura de referência [28]. O que na realidade existe, são diferentes propostas de modelos constituídos por um número variado de camadas que variam de acordo com os requisitos e de tarefas a serem tratadas [29]. Na Figura 5 são apresentadas duas propostas recorrentes de modelo para o domínio da Internet das Coisas.

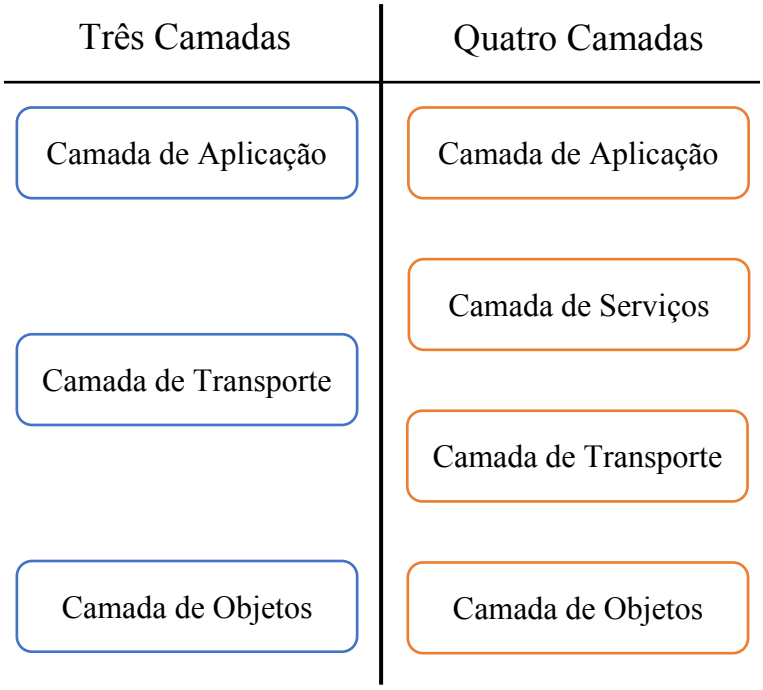


Figura 5 - Duas propostas diferentes para um modelo de referência no domínio da Internet das Coisas (três e quatro camadas) (Adaptado de [30])

I. Modelo de Três Camadas

O modelo de três camadas apresentado na Figura 5 é uma das principais abordagens no domínio da Internet das Coisas. As três camadas presentes neste modelo são as camadas de objetos, de transporte e de aplicação. Em relação a outros modelos, a principal vantagem deste reside na simplicidade de implementação [29].

As três camadas deste modelo concentram-se no funcionamento fundamental da Internet das Coisas. Contudo, com o aumento da heterogeneidade e da complexidade dos sistemas baseados em IoT, o modelo de três camadas revelou-se incapaz de efetuar a gestão dos processos competentes, não fornecendo, portanto, uma solução confiável. Logo, em virtude do surgimento de novos dispositivos tecnologicamente mais avançados, foi necessária a inclusão de novos serviços para suportar o aumento significativo de novas funcionalidades e de dados produzidos, o que motivou e resultou no acréscimo de novas camadas complementares ao atual modelo de três camadas [29], [31].

II. Modelo de Quatro Camadas

O modelo de quatro camadas surgiu com o objetivo de resolver algumas limitações inerentes ao modelo de três camadas. Além das três camadas apresentadas no modelo anterior, no atual modelo foi adicionada uma nova camada: a camada de serviços.

A nova camada adicionada neste modelo situa-se entre a camada de transporte e camada de aplicação (Figura 5), e tem por objetivo garantir a segurança na comunicação entre as duas camadas adjacentes. Esta camada é responsável por verificar a autenticidade dos dispositivos (presentes na camada de objetos) ou dos utilizadores (presentes na camada de aplicação) durante a transmissão de dados pela camada de transporte. Para além disso, esta camada ainda deve ser capaz de proteger a camada de transporte de ataques informáticos, tais como: ataques de negação de serviço (DoS), acessos não autorizados ou ameaças internas [30].

Devido aos desafios atuais relacionados com a privacidade e segurança no mundo digital, a camada adicionada a este modelo é de extrema importância, o que faz com que este modelo se torne mais completo do que o modelo de três camadas. Dada a relevância deste modelo, as secções seguintes apresentam uma descrição mais pormenorizada de cada camada.

2.1.1 Camada de Objetos

A camada de objetos, comummente conhecida como camada sensorial ou de perceção, é composta por dispositivos que, por sua vez, são constituídos por três elementos [30]:

1. **Sistemas Embebidos:** considerados a porta de entrada para o mundo digital, os sistemas embebidos, através do uso de software, poderão conectar-se e processar os dados produzidos pelos sensores, atuando de acordo com os mesmos. Os microcontroladores ou os computadores de placa única (*single-board computers*) são dois exemplos de sistemas embebidos.
2. **Sensores:** permitem adquirir determinados parâmetros de um ambiente físico – como a temperatura de um ambiente – traduzindo-os em sinais elétricos.
3. **Atuadores:** atuam, controlam ou manipulam um determinado ambiente físico. Contrariamente aos sensores, os atuadores recebem comandos enviados pelos microcontroladores, transformando-os em algum tipo de ação física.

Sendo esta considerada a camada física de um modelo IoT, a sua função passa por coletar e processar informações de um determinado ambiente, remetendo-os posteriormente para a camada adjacente, para que determinadas ações possam ser efetuadas com base nos dados coletados. Também é possível executar qualquer ação, sem a necessidade da

intervenção das camadas superiores, visto que, por si só, um sistema embebido tem a capacidade de processar as informações adquiridas pelos sensores e tomar decisões sobre o estado dos atuadores (Figura 6).

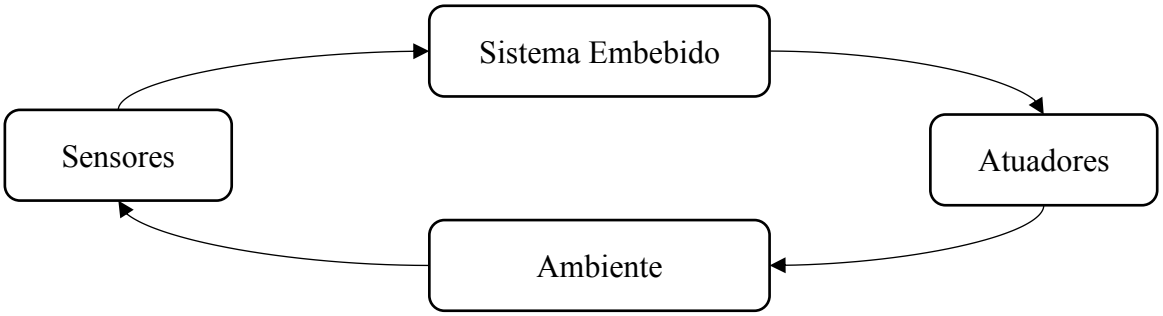


Figura 6 - Com base nos dados obtidos pelos sensores, os sistemas embebidos estabelecem uma ação nos atuadores

Os sensores e os atuadores tanto podem ser configurados por hardware como por software. Contudo, no caso de a configuração ser realizada por software, deve ocorrer por meio de um sistema embebido, de modo a processar a configuração via software [18].

2.1.2 Camada de Transporte

A camada de transporte, apelidada igualmente de camada de comunicação ou de rede, tem a responsabilidade principal de transferir os dados coletados na camada de objetos para outras camadas [32]. Esta transferência de dados pode ocorrer por meio de diferentes tecnologias e protocolos de comunicação, tais como: Wi-Fi, 5G, Bluetooth, NFC, RFID, ZigBee, LoRa, 6LoWPAN (*IPv6 over Low-power Wireless Personal Area Networks*), entre outras [3].

A atual diversidade de tecnologias de comunicação existentes faz com que os dispositivos responsáveis por processar os dados produzidos pelos sensores possam processá-los por meio de diferentes tecnologias de comunicação. Esta condição pode apresentar alguns problemas de interoperabilidade, pois, se um dispositivo comunicar através de uma tecnologia de comunicação não suportada pela camada de transporte, a comunicação não irá ser efetuada com sucesso. Há, portanto, a necessidade de implementar uma interface com o objetivo de transformar as diversas tecnologias que poderão ser adotadas pelos dispositivos, numa tecnologia de comunicação compatível com as camadas pelas quais os dados irão circular. A transformação de uma determinada tecnologia é, geralmente, concretizada por um dispositivo externo denominado *gateway*.

Os dispositivos *gateways*, também conhecidos por *proxys* reversos, funcionam como interfaces entre os dispositivos e os elementos nas camadas adjacentes. Além de serem capazes de proceder à conversão de uma determinada tecnologia de comunicação, estes dispositivos fornecem também, algumas funções de processamento de dados, como agregação e conversão, necessárias para traduzir os dados destinados a outros elementos do sistema [18].

Na Figura 7 é apresentado um exemplo de comunicação entre dois ecossistemas (Sistema A e Sistema B) que possuem compatibilidade com dois protocolos de comunicação. Enquanto o Sistema A possui elementos compatíveis com o Protocolo Y (Servidor S1 e Clientes) e com o Protocolo X (Servidor S2), os dispositivos do Sistema B (D1 e D2) são apenas compatíveis com o Protocolo Y.

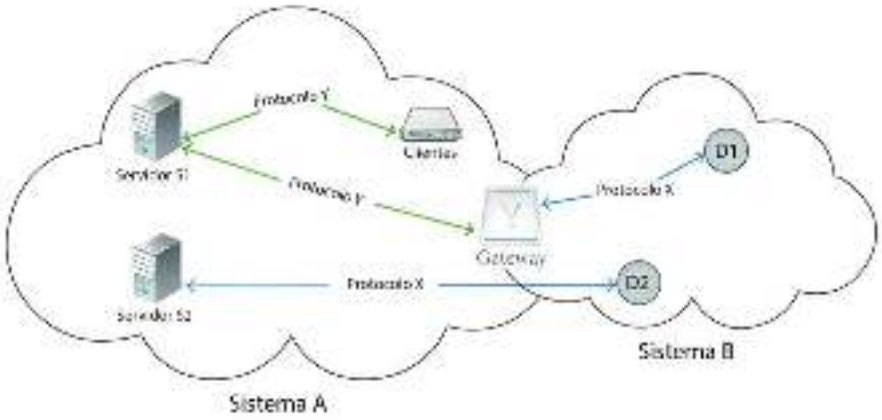


Figura 7 - O dispositivo D1 utiliza um *gateway* para comunicar com os elementos do Sistema A que possuem apenas compatibilidade com o Protocolo Y (Adaptado de [33])

As condições apresentadas neste exemplo obrigam a que a comunicação entre o dispositivo D1 e o Servidor S1 decorra por meio de um *gateway*. Assim, este *gateway* deve ser capaz de converter o protocolo Y para o protocolo X (e vice-versa). No caso do dispositivo D2, como a comunicação com o Servidor S2 é realizada pelo mesmo protocolo, não será necessário o auxílio de um *gateway*.

Sucintamente, a comunicação deve passar pelo *gateway* sempre que uma das seguintes condições se verifique:

1. A tecnologia de comunicação utilizada pelo servidor não é suportada;
2. O protocolo de transporte correspondente não é suportado;
3. Utilização de um formato de *payload* incompatível.

2.1.3 Camada de Serviços e Camada de Aplicação

A camada de serviços (ou de *middleware*), pode ser descrita como uma camada de integração entre os dispositivos IoT e as aplicações. A sua função está relacionada com o processamento e armazenamento dos dados enviados pela camada de transporte [29]. Não se limitando só às funcionalidades descritas acima, esta camada possui outras funções, como a configuração e o controlo dos utilizadores e dispositivos IoT, bem como a agregação dos dados provenientes destes.

Acima, a camada de aplicação é a camada que se situa no topo do modelo de quatro camadas e, consequentemente, é a que está mais próxima do utilizador. A sua função passa por aceder aos dados armazenados pelos serviços presentes na camada de serviços, e interpretá-los com a finalidade de estabelecer uma relação entre as regras de negócio estipuladas e os dados recebidos pela aplicação.

As regras de negócio são padrões que controlam e definem o comportamento das aplicações presentes na camada de aplicação. Por outras palavras, é através da estipulação destas regras que são instituídas todas as diretrizes de validação que garantem a consistência e a privacidade dos dados recebidos. O sucesso de um sistema IoT depende também de bons modelos de negócios [3], [30].

2.2 Arquitetura de Referência no domínio da Internet das Coisas

A definição e implementação de arquiteturas de referência, assim como em noutros domínios, é um relevante aspeto a ser considerado na Internet das Coisas. Independente do domínio em questão, há três principais vantagens na adoção de uma arquitetura de referência [27]:

1. **Simplificação do desenvolvimento de sistemas.** As arquiteturas de referência possibilitam um ganho de produtividade e de qualidade no desenvolvimento, visto que a estrutura arquitetónica já foi considerada e definida por um grupo de investigadores;
2. **Interoperabilidade entre sistemas heterogéneos.** Com a heterogeneidade existente entre sistemas, é fundamental desenvolver arquiteturas mutuamente interoperáveis, facilitando assim a integração e compatibilidade entre sistemas heterogéneos no domínio em questão;

3. **Acompanhar a evolução dos sistemas existentes.** Com o intuito de melhorar os sistemas já desenvolvidos, é essencial adaptá-los para atender a novos padrões ou até mesmo para proporcionar melhorias nos serviços já prestados.

Nas próximas duas secções (2.2.1 e 2.2.2), são analisadas as seguintes propostas de arquiteturas de referência para IoT: o *IoT Architectural Reference Model* [34], desenvolvido pelo projeto europeu *Internet of Things Architecture* (IoT-A) e a arquitetura de referência proposta pelo WSO2 [35].

2.2.1 Arquitetura de Referência - IoT-A

O IoT-A é um projeto europeu [36], que envolveu várias equipas de investigação com a missão de desenvolverem um modelo arquitetural de referência (MAR) que possibilitasse a interoperabilidade entre os sistemas IoT, definindo princípios e diretrizes para o design técnico de protocolos, interfaces e algoritmos.

O IoT-A fornece diferentes visões e perspetivas relevantes na construção e implementação de uma específica arquitetura. Uma perspetiva arquitetural é constituída por um conjunto de atividades, táticas e diretrizes que garantem que um sistema apresente um conjunto de atributos de qualidade capazes de influenciar um determinado número de visões arquiteturais de um sistema [27]. Além das perspetivas, o IoT-A ainda define as visões arquiteturais. As visões fornecidas dividem-se em quatro categorias: (1) funcional; (2) informação; (3) operação e; (4) implantação.

A visão funcional (1) engloba as principais funcionalidades adotadas pela arquitetura e que devem ser consideradas na construção de um sistema IoT. Já a visão da informação (2) concentra-se na descrição, no tratamento e no ciclo de vida da informação, bem como no fluxo de informação nos componentes do sistema. Por fim, a visão de operação (3) e a visão de implantação (4) concedem aos utilizadores um conjunto de diretrizes para auxiliá-los a decidir sobre as diferentes opções de design que terão de enfrentar na implementação dos seus serviços. Como a análise que se pretende efetuar é em relação às funcionalidades da arquitetura de referência do IoT-A, o foco desta secção concentra-se na visão funcional. Mais detalhes acerca de outras visões podem ser encontrados na respetiva referência [37].

Conforme é apresentado na Figura 8, a visão funcional do MAR do IoT-A, é definida por nove grupos de funcionalidades (GFs): (1) Aplicação; (2) Gestão; (3) Organização de

Serviços; (4) Gestão de Processos IoT; (5) Entidades Virtuais; (6) Serviços IoT; (7) Segurança; (8) Comunicação e; (9) Dispositivo.

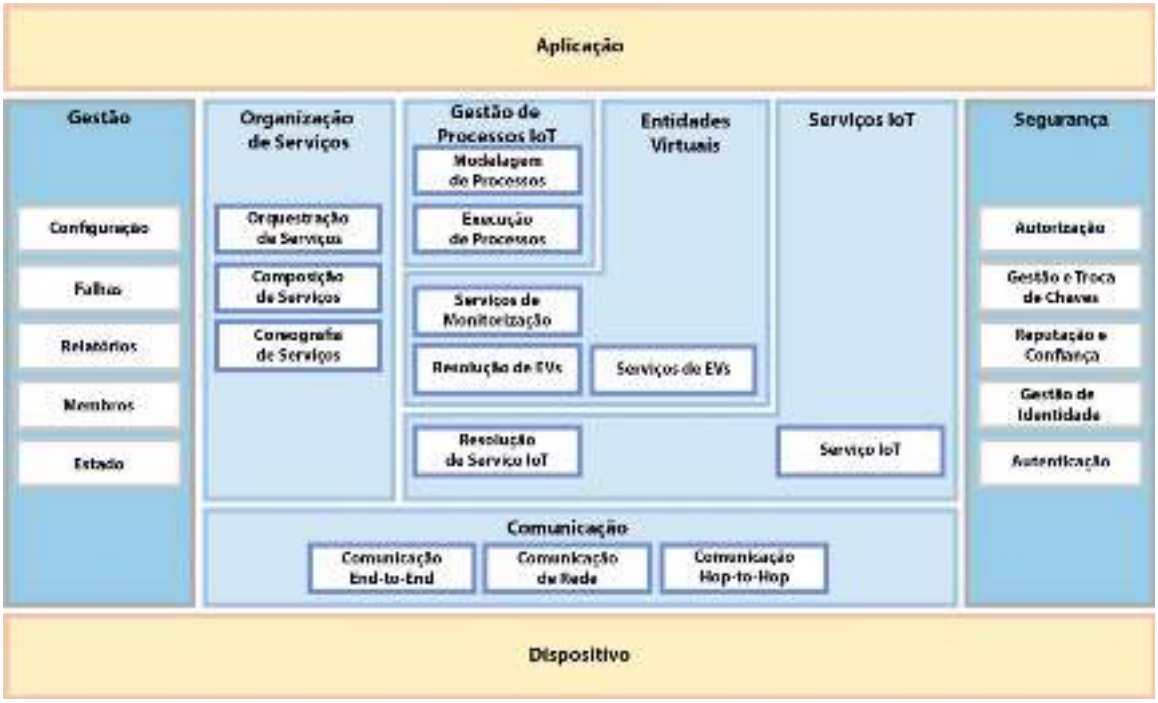


Figura 8 - Distribuição dos GFs da visão funcional da arquitetura IoT-A (Adaptado de [37])

Um GF pode conter um ou mais componentes funcionais (CFs), conforme ilustrado na figura anterior (retângulos brancos). Contudo, apesar de a visão funcional descrever os CFs, ela não específica as interações que ocorrem entre estes elementos, uma vez que tais interações dependem tipicamente de escolhas de projetos e, portanto, são realizadas durante o desenvolvimento da arquitetura concreta. É importante observar que os GF de aplicação e de dispositivo estão fora do escopo da arquitetura de referência do IoT-A, enquanto os GF de Gestão e de Segurança são transversais aos demais GFs.

De uma forma breve, cada um dos GF são apresentados nas próximas secções.

2.2.1.1 Gestão

O **GF de Gestão** engloba cinco CFs (Figura 9): Configuração, Falha, Relatório, Membro e Estado. O CF de Configuração é responsável por carregar a configuração do sistema, bem como coletar e armazenar as configurações dos restantes CFs e dispositivos. Por sua vez, o CF de Falhas tem como objetivo identificar, tratar, monitorizar e recuperar eventuais falhas que possam ocorrer num sistema IoT. Sempre que ocorra uma falha, o respetivo componente funcional deve notificar imediatamente o CF de Falta. Adicionalmente, as notificações geradas também podem ser difundidas para os restantes CFs.

O CF de Membros é responsável pela gestão das associações e informações associadas de qualquer entidade relevante (GFs e CFs) de um sistema IoT. O CF de Membros tem três funções padrão: monitorização, recuperação e atualização dos membros que se enquadrem num filtro específico.

O CF de Relatórios fornece informações em relação a outros CFs pertencentes ao GF de Gestão. Estes relatórios permitem retirar várias conclusões, como determinar a eficiência do sistema. Apenas existe uma função padrão para este CF: a criação de relatórios referentes ao sistema.

O CF de Estado controla a condição do sistema IoT, podendo oferecer o estado passado, atual e futuro do sistema. As funções do CF de Estado são as de monitorizar, definir ou alterar um determinado estado no sistema. Com este CF é possível também, definir um estado por um determinado período, recuperar o anterior estado do sistema através do histórico de estados ou atualizá-lo em função de uma alteração ou criação de um outro estado.

Figura 9 - GF de Gestão (Adaptado de [37])

2.2.1.2 Organização de Serviços

O **GF de Organização de Serviços** atua como um ponto central de comunicação entre os vários GFs, assegurando o controlo dos serviços em diferentes níveis de abstração. Este GF é constituído por três CFs (Figura 10): Orquestração de Serviços, Composição de Serviços e Coreografia de Serviços.

O CF de Orquestração de Serviços é responsável por coordenar os serviços IoT, identificando o serviço mais apropriado para responder às solicitações dos utilizadores ou

de outros CFs. O CF de Composição de Serviços determina a composição de cada serviço. Por último, o CF de Coreografia de Serviços disponibiliza um *broker* que possibilita que os serviços IoT comuniquem pelo modelo *Publish/Subscribe*. Desta forma, qualquer cliente pode solicitar um serviço com determinados recursos ao broker, e no caso de não existir nenhum serviço correspondente ao desejado, a solicitação é memorizada até que fique disponível um serviço que atenda à solicitação.

Figura 10 - GF de Organização de Serviços (Adaptado de [37])

2.2.1.3 Gestão de Processos IoT

O **GF de Gestão de Processos IoT** tem como principal objetivo fornecer os conceitos e interfaces necessárias para adaptar os processos convencionais aos processos presentes em ambientes IoT. Para tal, as suas principais funções estão divididas em dois CFs (Figura 11): Modelagem de Processos e Execução de Processos.

Figura 11 - GF de Gestão de Processos IoT (Adaptado de [37])

O CF de Modelagem de Processos fornece um conjunto de ferramentas que permitem descrever e modelar os diversos processos de um ambiente IoT. Por sua vez, o CF de Execução de Processos é responsável por implementar e executar os processos previamente

modelados pelo CF de Modelagem de Processos. Por isso, é essencial que o CF de Execução de Processos consiga comunicar com o GF de Organização de Serviços.

2.2.1.4 Entidades Virtuais

O **GF de Entidades Virtuais (EV)** permite interagir com os elementos do Sistema IoT com base em EVs, recorrendo a funcionalidades relacionadas com a pesquisa de serviços que forneçam informações de outros EVs. Uma EV representa uma entidade física constituída por um identificador, um tipo e uma série de atributos que definem as suas funcionalidades.

As EVs podem ser classificadas como ativas ou passivas. Uma EV ativa são os elementos que executam as aplicações, serviços ou agentes de *software*, enquanto as passivas são os elementos que constituem os recursos de um sistema, como por exemplo, os registos de uma BD. Por fim, as EVs interagem com os serviços através de associações que indicam qual informação, ou ação, pode ser obtida e/ou executada numa EV.

Este GF é constituído por três CFs (Figura 12): Resolução de EVs, Serviços de Monitorização e Serviços de EVs. O CF de Resolução de EVs é responsável por fornecer determinadas funcionalidades aos utilizadores do sistema IoT que permitem restaurar as associações entre EVs e serviços IoT. O CF de Serviços de Monitorização é responsável por encontrar automaticamente novas associações que são introduzidas no CF de Resolução de EVs. Por último, o CF Serviços de EVs opera com serviços de entidade, que representam um ponto de acesso para uma entidade específica, disponibilizando meios para exercer a manipulação de estado de uma entidade.

Figura 12 - GF de Entidades Virtuais (Adaptado de [37])

2.2.1.5 Serviços IoT

O **GF de Serviços IoT** contém serviços IoT, bem como funcionalidades para pesquisar e modificar informações relativas aos serviços de um sistema IoT. Este GP é composto por dois CFs (Figura 13): Serviço IoT e Resolução de Serviço IoT.

O CF de Serviço IoT permite facultar um recurso específico para outras partes do sistema IoT. Além disso, um serviço IoT também pode fornecer informações a um recurso do sistema de modo a controlar os atuadores de um dispositivo ou até configurá-lo. Os recursos podem ser configurados em termos não funcionais, como segurança e confiabilidade (por exemplo, controlo de acessos), resiliência (disponibilidade, por exemplo) e desempenho (escalabilidade).

O CF de Resolução de Serviço IoT fornece aos utilizadores todas as funcionalidades necessárias para comunicarem com os Serviços IoT. O utilizador tanto pode ser uma pessoa como um elemento de software. Este CF também permite filtrar e modificar as descrições dos serviços (normalmente armazenadas numa BD), para que mais tarde possam auxiliar o utilizador a identificar determinados serviços.

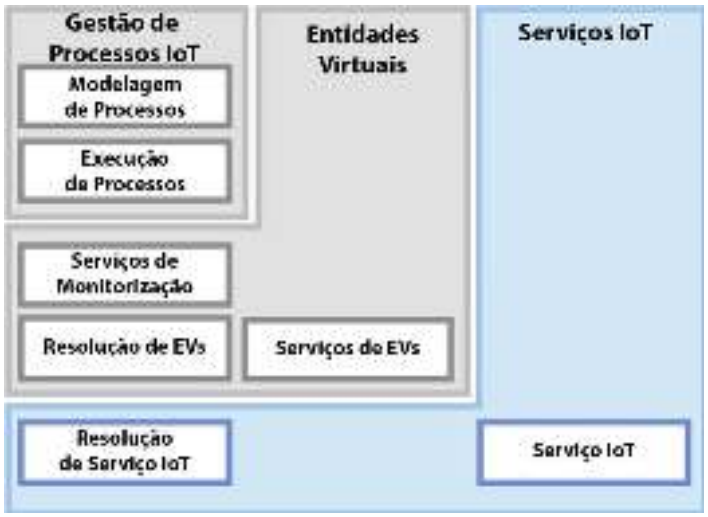


Figura 13 - GF de Serviços IoT (Adaptado de [37])

2.2.1.6 Segurança

O **GF de Segurança** é responsável por garantir a segurança e a privacidade dos componentes do sistema. É composto por cinco CFs (Figura 14): Autorização, Gestão e Troca de Chaves, Reputação e Confiança, Gestão de Identidade e Autenticação

O CF de Autorização é responsável por o controlo de acessos aos recursos do sistema. As suas principais funcionalidades incluem: determinar se uma ação é ou não autorizada; e gerir as políticas de acesso.

O CF de Autenticação está relacionado com a autenticação de utilizadores e serviços. As suas principais funcionalidades são a autenticação dos utilizadores com base nas credenciais fornecidas e a validação do *token* de acesso enviado pelo utilizador.

O CF de Gestão e Troca de Chaves garante a segurança nas comunicações entre dois ou mais pontos que ainda não realizaram a troca de chaves ou cuja interoperabilidade não é garantida, assegurando assim a integridade e confidencialidade da comunicação. Este CF possui duas funcionalidades: distribuição de chaves entre os vários pontos de comunicação e o registo de todos os recursos de segurança do sistema, registando as capacidades de cada nó e fornecendo chaves na estrutura correta.

O CF de Gestão da Identidade tem como finalidade criar identidades fictícias (pseudónimos ou identidades de grupo) associadas a credenciais de segurança para que os utilizadores e serviços usem-nas durante o processo de autenticação.

Por fim, o CF de Reputação e Confiança distribui níveis de reputação pelos utilizadores e calcula os níveis de confiança dos serviços. Este CF tanto pode responder a solicitações de informações de reputação sobre uma entidade, bem como receber informações de reputação em relação a uma entidade.

Figura 14 - GF de Segurança (Adaptado de [37])

2.2.1.7 Comunicação

O **GF de Comunicação** tem como objetivo moldar as diferentes tecnologias de comunicação utilizadas em sistemas IoT. O atual GF é constituído por três componentes funcionais (Figura 15): Comunicação *Hop-to-Hop*, Comunicação de Rede e Comunicação *End-to-End*.

O CF de Comunicação *Hop-to-Hop* fornece a primeira camada de abstração na comunicação com os dispositivos. As suas principais funções estão relacionadas com a

transmissão de *frames* entre o CF de Comunicação de Rede e os dispositivos, permitindo definir argumentos como confiabilidade, a integridade, o tipo de criptografia e o controlo de acessos. Este CF permite ainda efetuar a gestão da fila de *frames*, definindo o seu tamanho e as suas prioridades estruturais.

O CF da Comunicação de Rede possibilita a comunicação entre diferentes redes através de endereços e resolução de IDs. As funções do CF de Comunicação de Rede resumem-se na transmissão de pacotes entre o CF de Comunicação *Hop-to-Hop* e o CF de Comunicação de *End-to-End*, permitindo configurar argumentos como confiabilidade, integridade, tipo de criptografia, endereçamento *unicast/multicast* e controlo de acessos. Este CF também possibilita a tradução de diferentes protocolos de rede e a gestão da fila de pacotes, definindo o seu tamanho e prioridades estruturais.

O CF de Comunicação *End-to-End* deve garantir o transporte, a tradução e ajuste de parâmetros de configuração de uma mensagem, sempre que a comunicação intercepta diferentes ambientes de rede. Este CF é responsável por transmitir as mensagens entre CF de Comunicação de Rede e o GF Serviço IoT, permitindo definir argumentos como confiabilidade, integridade, criptografia, controlo de acessos e multiplexação. Outras funções são o armazenamento de mensagens em *cache* e a tradução entre diferentes protocolos *End-to-End*, como por exemplo, a tradução do protocolo HTTP (*Hypertext Transfer Protocol*) para CoAP (*Constrained Application Protocol*).

Figura 15 - GF de Comunicação (Adaptado de [37])

2.2.2 Arquitetura de Referência - WSO2

A arquitetura de referência do WSO2, conforme demonstrado na Figura 16, é composta por um conjunto de camadas, das quais se destacam: (1) Comunicações Externas (Web/Portal, *Dashboards*, APIs); (2) Processamento e Análise de Eventos (incluindo armazenamento de dados); (3) Agregação/Barramento; (4) Comunicação entre Dispositivos; Dispositivos; (5) Gestão de Dispositivos; (6) Gestão de Acesso e Identidade.

Figura 16 - Arquitetura de referência WSO2

A **camada dos Dispositivos**, a mais inferior desta arquitetura de referência, é composta por dispositivos que devem se conectar e comunicar pela Internet, de forma direta ou indireta. Conexões diretas ocorrem quando um dispositivo pode se conectar, de forma natural, sem intermediários. Já em conexões indiretas, os dispositivos dependem de outros dispositivos auxiliares para estabelecerem uma conexão com a Internet, tais como, dispositivos ZigBee conectados a um *gateway* que seja compatível com as mesmas tecnologias de comunicação entre os dois pontos.

A cada dispositivo deve ser fornecido um identificador exclusivo, conhecido como UUID (*Universally Unique Identifier*), de modo a garantir-lhe uma identidade única. Este identificador pode ser gravado no dispositivo (armazenado em memória não volátil ou num circuito integrado secundário), fornecido pelo próprio hardware (como o endereço da interface de Bluetooth ou endereço MAC - *Media Access Control*) ou até por meio de um *Bearer Token*.

A **camada de Comunicações** é responsável por garantir a conectividade dos dispositivos com os outros elementos do sistema. Existem diversos protocolos que podem ser utilizados para a comunicação entre os dispositivos e o servidor, entre os quais os mais conhecidos: HTTPS (*Hyper Text Transfer Protocol Secure*), MQTT (*Message Queuing Telemetry Transport*) e CoAP. Na arquitetura de referência do WSO2, o MQTT foi o protocolo preferencialmente escolhido para tal função, enquanto o HTTPS é uma opção alternativa. Face aos outros protocolos, a preferência pelo MQTT deve-se pelo suporte mais

amplo de bibliotecas e pela simplicidade na comunicação, prevenindo potenciais problemas relacionados com *firewall* e NAT (*Network Address Translation*). As comunicações entre dispositivos que utilizam o protocolo HTTP serão atualizadas para uma conexão bidirecional WebSocket. Esta atualização justifica-se pela ineficiência e pelos elevados custos associados ao HTTP, tanto em termos de tráfego de rede como de requisitos de energia para os dispositivos.

A **camada de Agregação/Barramento** tem a responsabilidade de agregar e intermediar as comunicações do sistema. Para desempenhar esta função, é necessário utilizar um servidor HTTP e/ou um *broker* MQTT capazes de suportar a comunicação com os dispositivos, podendo, mediante um *gateway*, converter os protocolos de comunicação utilizados pelos dispositivos. É fundamental que esta camada tenha um foco especial em termos de segurança, devendo assim: realizar a validação dos *bearer tokens* recorrendo a um servidor de *OAuth2* e efetuar o controlo de acessos baseando em políticas de acesso.

A **camada de Processamento e Análise de Eventos** trata dos eventos provenientes da camada de agregação/barramento, processando-os e tomando determinadas decisões relacionadas a estes eventos. Os dados agregados a cada evento podem ser armazenados numa BD, através da execução de uma aplicação do lado servidor, tais como aplicações RESTful (*Representational State Transfer*). Em alternativa, uma abordagem de maior agilidade é o uso de plataformas *Big Data* em *cloud,* que suportam diversas tecnologias que fornecem análises altamente escaláveis dos dados provenientes dos dispositivos. Esta camada possui ainda a capacidade de interagir com plataformas externas de processamento de dados.

A **camada de Comunicações Externas** disponibiliza uma arquitetura de páginas web modulares, que permitem criar *dashboards* para visualizar os dados provenientes dos dispositivos, bem como interagir através de APIs com camada de processamento/análise de eventos e com sistemas externos. A gestão e a monitorização das APIs são realizadas pelo Gestor de APIs.

A **camada de Gestão de Dispositivos** é composta por uma aplicação chamada de gestor de dispositivos, que viabiliza a comunicação com os dispositivos através de vários protocolos, proporcionando um controlo individual para cada dispositivo. Se necessário, esta aplicação pode bloquear um determinado dispositivo ou até mesmo manipular remotamente o software nele presente. Com o auxílio da camada de gestão de identidade e acesso, a

camada de gestão de dispositivos pode controlar os acessos num dispositivo, dependendo das permissões atribuídas.

Por fim, a **camada de Gestão de Identidade e Acesso,** é responsável por fornecer serviços como: emissão e validação de *OAuth2 tokens* de acesso aos serviços, monitorização das políticas de acesso e diretório central LDAP (*Lightweight Directory Access Protocol*).

2.3 Plataformas no domínio da Internet das Coisas

Uma arquitetura de referência é especialmente útil para projetar arquiteturas de software específicas (instâncias) destinadas a sistemas de um determinado domínio de aplicação. No contexto da Internet das Coisas, uma instância de uma arquitetura de referência pode ser relevante para o desenvolvimento de uma plataforma.

Uma plataforma da Internet das Coisas atua como uma ponte que interliga vários componentes heterogêneos, facilitando a adaptação, identificação e a gestão de dados e de outros tipos de recursos. Além do mais, promovem a reutilização de serviços genéricos que podem ser configurados para facilitar o desenvolvimento de aplicações IoT de forma mais eficiente. O seu desenvolvimento tem estado em constante expansão, atraindo cada vez mais a atenção das comunidades acadêmicas e da indústria [38]. Nas próximas subsecções são apresentados alguns exemplos de plataformas.

2.3.1 WSO2

A plataforma WSO2 [35] é uma plataforma modular e de código aberto que disponibiliza diversos componentes de software ao nível da camada de serviços e de dispositivos (Figura 5, do subcapítulo 2.1). Esta plataforma pode ser implementada em três destinos diferentes: servidores locais, em *clouds* publicas (incluindo as plataformas de serviços AWS e Microsoft Azure) e em *clouds* híbridas ou privadas, tais como OpenStack, Suse Cloud, Amazon Virtual Private Cloud e Apache Stratos.

Esta plataforma oferece aos utilizadores determinados recursos para complementar as funcionalidades desejadas. Os seus recursos podem ser adicionados ou removidos conforme as necessidades do utilizador.

A Figura 17 associa os componentes correspondentes da plataforma WSO2 à arquitetura de referência analisada na secção 2.2.2.

Figura 17 - Arquitetura da plataforma WSO2

Na camada de Agregação/Barramento, são implementados dois componentes: o *WSO2 Message Broker* (MB) e o *WSO2 Enterprise Service Bus* (ESB). O MB atua como um broker MQTT, enquanto o ESB é responsável por tratar mais de duas mil milhões de solicitações por dia, suportando vários protocolos, como HTTP, MQTT, AMQP (*Advanced Message Queuing Protocol*), entre outros. O ESB disponibiliza recursos, como transformação de dados, ponte entre protocolos, suporte OAuth2 e gestão das políticas de acesso.

Na camada de Análise e Processamento de Eventos, o WSO2 integra uma ferramenta para análise de dados, o *WSO2 Data Analytics Server*. Já na camada de Comunicações Externas, encontram-se os componentes *WSO2 User Engagement Server* (UES) e *WSO2 API Manager* (AM). O UES oferece suporte à criação e monitorização de IUs Web, enquanto o AM gere o ciclo de vida das APIs, oferecendo suporte aos programadores das respetivas APIs.

Na camada de Gestão de Dispositivos encontra-se o componente o *WSO2 Enterprise Mobility Management*, que permite a gestão dos dispositiso da plataforma através de um equipamento móvel. Por último, na camada de Gestão de Identidade e Acesso, subsiste o componente *WSO2 Identity Server* que oferece vários recursos relacionados com permissões de acesso a outros componentes da plataforma, tais como a emissão, revogação e gestão de *tokens*.

2.3.2 Amazon Web Services IoT

O objetivo da Amazon Web Services IoT ou AWS IoT [39] consiste em facilitar a tarefa do utilizador no desenvolvimento de soluções IoT. Para isso, a AWS IoT disponibiliza um conjunto de serviços *cloud* e ferramentas que possibilitam e facilitam a conexão dos dispositivos e aplicações a outros serviços associados à *cloud* da AWS (Figura 18).

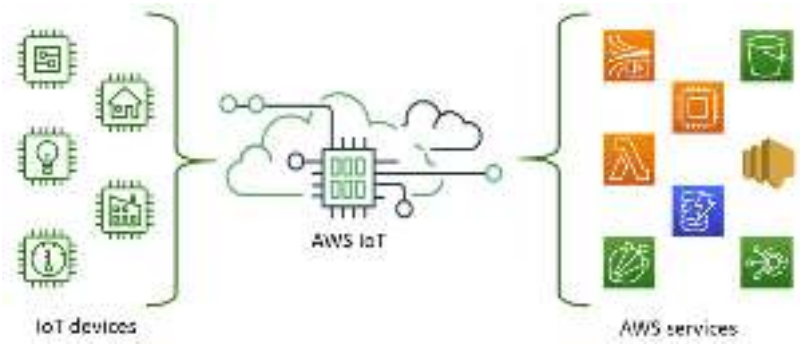


Figura 18 - A AWS IoT pode ser visto como uma ponte entre os dispositivos e os restantes serviços AWS [39].

Os dispositivos e aplicações desenvolvidas pelo utilizador, podem aceder aos serviços da AWS IoT através de diferentes tipos de interfaces, tais como os SDKs[6] de dispositivos da AWS IoT, interfaces de linha de comando (AWS CLI), APIs, entre outros.

Dento do universo da AWS IoT, o serviço *cloud* AWS IoT Core assegura a interação entre dispositivos conectados com as aplicações *cloud* da plataforma e outros dispositivos. A arquitetura da AWS IoT Core é representada na Figura 19 e inclui os seguintes componentes responsáveis por fornecer as principais funcionalidades deste serviço (ilustrados a verde) [39]:

1. *Message Broker*: responsável por gerir a troca de mensagens entre os dispositivos conectados e as aplicações na nuvem. Garante a entrega eficiente e segura das mensagens.
2. *Device Shadows*: são representações virtuais dos dispositivos conectados, permitindo que as aplicações interajam com eles mesmo quando não estão disponíveis ou conectados. Este componente armazenam informações sobre o estado atual e as configurações dos dispositivos.

[6] "*Software Development Kit*" em inglês ou "Kit de Desenvolvimento de Software" em português, é um conjunto de ferramentas e recursos que ajudam os programadores a criarem aplicações para um determinado sistema operativo, plataforma ou serviço.

3. *Rules Engine*: responsável por processar e tomar ações baseadas em eventos ou condições específicas. Permite definir regras e lógicas personalizadas para automatizar ações ou acionar eventos com base nos dados recebidos dos dispositivos.

4. *Security and Identity*: este componente garante a segurança das comunicações entre os dispositivos e a plataforma de nuvem, além de fornecer autenticação e autorização para garantir que apenas dispositivos confiáveis possam interagir com o serviço.

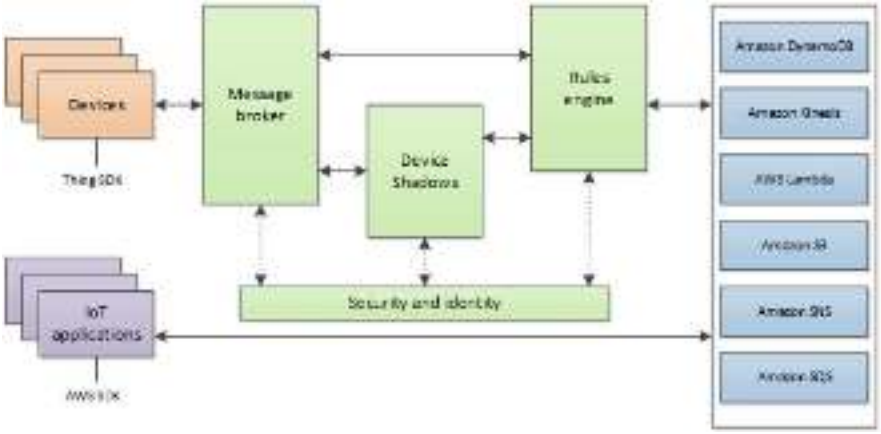


Figura 19 - Arquitetura do AWS IoT Core

O AWS IoT Core suporta diversos protocolos, como o MQTT, MQTT sobre WSS (*WebSocket Secure*), HTTPS e LoRaWAN (*Long Range Wide-Area Network*). Possui também ferramentas de autenticação e autorização para garantir a segurança do fluxo de dados. Para proteger a confidencialidade dos seus protocolos de comunicação, todo o tráfego que entra e sai deste serviço deve ser cifrado pelo protocolo *Transport Layer Security* (TLS). A utilização do sistema FreeRTOS[7] pode assegurar a segurança na conexão dos dispositivos de baixo consumo de energia com a AWS IoT.

Um serviço adicional relevante do AWS IoT é o AWS IoT Analytics, que possibilita realizar eficientemente análises complexas de grandes volumes de dados coletados e não estruturados. Este serviço filtra e transforma os dados coletados antes de os armazenar, tornando-os passíveis de serem visualizados e analisados por meio de uma série temporal.

2.3.3 Microsoft Azure IoT

O Microsoft Azure IoT [40] é uma coleção de serviços *cloud* geridos pela Microsoft Azure que conectam, monitorizam e controlam milhares de milhões de recursos IoT. Esta

[7] O FreeRTOS é um sistema operativo de código aberto e em tempo real desenvolvido para microcontroladores. Destaca-se por oferecer um ambiente de execução multitarefa e escalonamento de tarefas, permitindo aos programadores criar sistemas eficientes e robustos.

plataforma permite criar soluções IoT, utilizando serviços PaaS (*Platform as a Service*) ou aPaaS (*Application Platform as a Service*).

1. **Plataforma como serviço (PaaS)** é um modelo de computação em *cloud,* onde o utilizador pode personalizar as ferramentas de hardware e software de acordo com uma tarefa ou objetivo específico. Com os serviços PaaS, o utilizador é responsável pela escalabilidade e configuração das aplicações desenvolvidas, sem se preocupar com assuntos relacionados com infraestrutura.
2. **Plataforma de aplicação como serviço (aPaaS)** proporciona um ambiente em *cloud* para construir, gerir e entregar aplicações aos clientes. Com a aPaaS deixa de existir a preocupação com escalabilidade e da maioria das configurações relacionadas. Contudo, ainda requerem um programador para produzir uma solução final.

O Azure IoT Central (Figura 20) é uma plataforma de aplicação como um serviço (aPaaS) concebida para simplificar e acelerar a montagem e o funcionamento da solução IoT. Nesta plataforma é fornecida uma IU web, que permite monitorizar as condições dos dispositivos, criar regras e gerir milhões de dispositivos e dados.

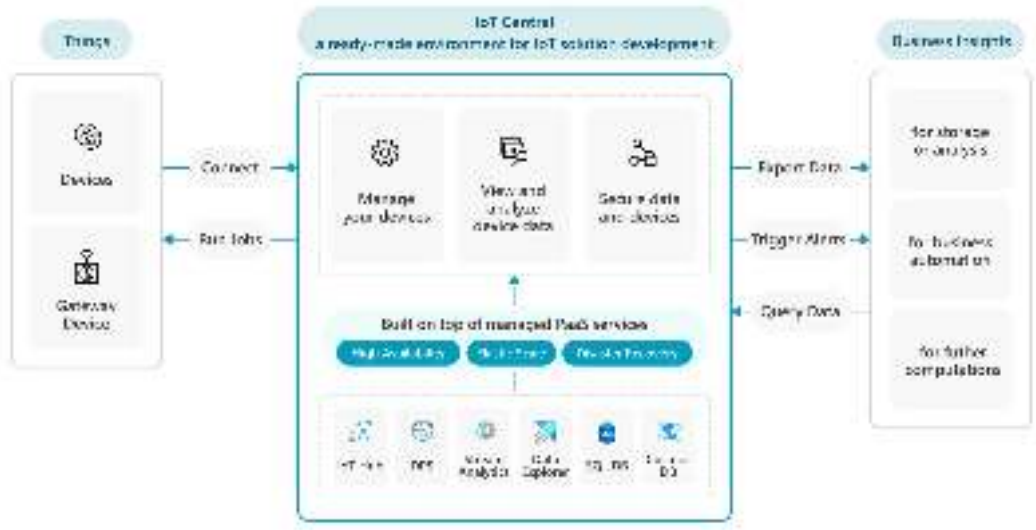


Figura 20 - Arquitetura aPaaS do Azure IoT Central [40]

Num outro panorama, pode ser necessário um controlo e personalização mais aprofundado do que aquele que o Azure IoT Central oferece. Para esses casos, a Microsoft oferece uma plataforma individual como serviço (PaaS), que sucintamente, são serviços *cloud* (Figura 21) que possibilitam a construção de soluções IoT personalizadas. A construção de uma solução IoT pode ser alcançada pela combinação de alguns dos seguintes serviços PaaS:

1. Serviço Device Provisioning e Hub IoT: distribuição de mensagens entre as aplicações IoT e os seus dispositivos, garantindo assim a conectividade e a gestão do dispositivo;
2. Serviço Azure Time Series Insights: armazenamento e análise de séries temporais interativas, de modo a acelerar a utilização dos dados adquiridos pelos dispositivos;
3. Serviço Azure Stream Analytics: projetado para analisar e processar grandes volumes de dados de várias fontes em tempo real;
4. Serviço Azure IoT Edge: gestão das análises executadas por inteligência artificial, por serviços de terceiros ou por lógica de negócios personalizada;

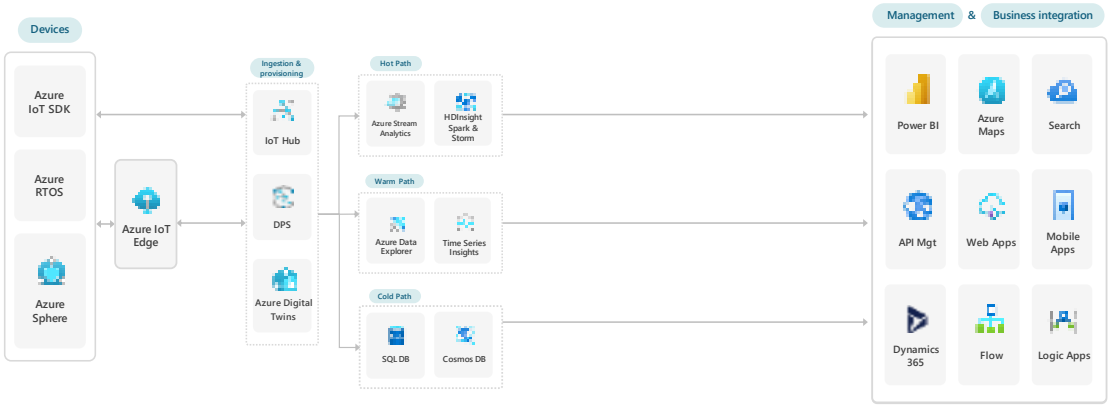


Figura 21 - Arquitetura PaaS do Azure IoT [40]

A Microsoft fornece ainda SDKs de código aberto destinados para os dispositivos com o objetivo de simplificar e acelerar o desenvolvimento das soluções IoT. Os SDKs dos dispositivos IoT e o Hub IoT são compatíveis com os protocolos de comunicação HTTP, MQTT e AMQP, tornando mais fácil a integração com outras plataformas e serviços.

2.3.4 Arduino IoT Cloud

O Arduino IoT Cloud [41] é uma plataforma que possibilita aos seus utilizadores desenvolver projetos IoT. Com o objetivo de fornecer uma solução integral, esta plataforma oferece recursos essenciais, como a configuração de dispositivos, edição, compilação e carregamento de código. Além destes, esta plataforma oferece ainda, os seguintes recursos: (1) Monitorização de dados: monitorização dos valores enviados pelos dispositivos através de uma IU; (2) Sincronização de variáveis: permite sincronizar o valor das variáveis definidas entre dispositivos; (3) Agendar tarefas: programação de tarefas num período específico; (4) Carregamento *Over-the-Air* (OTA): recurso que permite efetuar o

carregamento sem fios de programas no dispositivo. Nesta plataforma, esta funcionalidade é suportada apenas por algumas placas, tais como: Arduino MKR WiFi 1010, Arduino Nano 33 IoT, Arduino Nano RP2040 Connect e Portenta H7. (5) *Webhooks*: é um recurso que envia e recebe mensagens automatizadas de e para outros serviços. Por exemplo, pode-se utilizar este recurso para receber uma notificação quando o estado de uma varável é alterado; (6) Suporte à Amazon Alexa; (7) Partilha de *dashboards*;

A plataforma Arduino IoT Cloud é compatível com uma variedade de placas de desenvolvimento, incluído todas as placas oficiais Arduino e placas baseadas no microcontrolador ESP32/ESP8266. Em relação às opções de conexão, esta plataforma suporta Wi-Fi, LoRaWAN (via *The Things Network*[8]) e redes móveis.

Destaca-se ainda, a disponibilização de um Web Editor que permite aos utilizadores projetar e armazenar online todos os seus programas. Assim, os programas armazenados podem ser carregados numa placa de desenvolvimento compatível através do Web Editor. Adicionalmente, nesta interface, as bibliotecas mais populares são automaticamente disponibilizadas, retirando ao utilizador a tarefa de as descarregar e de as adicionar ao seu editor de código.

2.4 Comparação entre Plataformas IoT

Após a análise de quatro plataformas IoT, pretende-se nesta secção apresentar uma análise comparativa entre determinadas características funcionais, nomeadamente:

1. Escalabilidade: refere-se à capacidade que um sistema possui para crescer, adequando-se a novos fluxos de clientes e permitindo o aumento da sua estrutura sem substituição ou prejuízo das funcionalidades existentes;
2. Gestor de dispositivos: capacidade de associar, configurar e dissociar dispositivos à plataforma, como também a visualização dos dados enviados pelos mesmos;
3. IU amigável: uma interface intuitiva e eficiente é essencial para fazer com que o utilizador se sinta confortável ao utilizar a ferramenta em questão;
4. Fornecimento de hardware: além de software, algumas plataformas disponibilizam kits de hardware compostos por atuadores, sensores e placas de desenvolvimento;

[8] The Things Network é uma rede global de comunicação de dados para dispositivos IoT. Mais informações em: https://www.thethingsnetwork.org/

5. Ferramentas de análise de dados (FAD): o aumento do número de dispositivos IoT produz um aumento de dados gerados a cada segundo, tornando fundamental a oferta de ferramentas para tratar e analisar esses dados;
6. Automação: capacidade de automatizar ações baseadas em determinadas ocorrências;
7. Interoperabilidade: possibilidade de integração e comunicação entre dois ou mais sistemas;
8. Segurança: implementação de estratégias de segurança para evitar ataques cibernéticos;

Com o objetivo de obter resultados mais representativos, além das quatro plataformas IoT originalmente analisadas, serão adicionadas a esta análise comparativa mais quatro plataformas (Blynk, Ubidots, ThingWorx e ThingSpeak). Assim, de forma sucinta, nas tabelas 1 e 2 é apresentada a correspondência entre cada plataforma analisada e as características funcionais apresentadas anteriormente. Para tal, foi adotada uma determinada simbologia para categorizar os resultados, cuja legenda segue-se abaixo:

✓ - Característica funcional implementada;

○ - Característica funcional parcialmente implementada;

✗ - Característica funcional não implementada;

Tabela 1 - Tabela de caraterísticas (1º Parte)

Plataforma	Escalabilidade	Gestor de dispositivos	IU amigável	Fornecimento de *hardware*
Arduino Cloud	✓	✓	✓	✓
AWS IoT Core	✓	✓	✗	✗
Blynk	✓	✓	✓	✗
Microsoft Azure IoT	✓	✓	○	✗
Ubidots	✓	✓	✓	✗
ThingSpeak	✓	✗	✓	✗
ThingWorx	✓	✓	✗	✗
WSO2	✓	✓	○	✗

Tabela 2 - Tabela de características (2º Parte)

Plataforma	FAD	Automação	Interoperabilidade	Mecanismos de Segurança
Arduino Cloud	○	✗	○	✓
AWS IoT Core	✓	✓	✓	✓
Blynk	○	✓	○	✓
Microsoft Azure IoT	✓	✓	✓	✓
Ubidots	○	✓	✓	✓
ThingSpeak	✓	✓	○	✓
ThingWorx	✓	✓	✓	✓
WSO2	✓	✓	✓	✓

Através de uma sucinta análise, as tabelas 1 e 2 evidenciam a heterogeneidade existente entre as características inerentes das plataformas. Por exemplo, no que diz respeito

à característica FAD, a maioria das plataformas já disponibiliza ferramentas e mecanismos capazes de lidar com os dados gerados pelos dispositivos. Contudo, neste aspeto, algumas plataformas estão consideravelmente mais desenvolvidas do que outras. O mesmo acontece com a interoperabilidade, havendo plataformas que demonstram maior preocupação no desenvolvimento de capacidades interoperáveis.

Em termos de IU amigável, devido às extensas funcionalidades fornecidas, as interfaces das plataformas acabam por se tornarem menos intuitivas, dificultando a tarefa dos seus utilizadores em relação à compreensão do seu funcionamento. Para terminar, uma das maiores diferenças constata-se no fornecimento de hardware. A maioria das plataformas não fornecem qualquer tipo de equipamento, focando-se apenas em fornecer um meio de comunicação que permita aos utilizadores comunicarem com os seus dispositivos de aquisição de dados. Deste modo, considera-se que, das várias características funcionais analisas, esta em particular é aquela que as plataformas ainda não exploram amplamente.

2.5 Protocolos de Comunicação

A padronização da comunicação é um aspeto importante no desenvolvimento IoT. Esforços têm sido feitos para padronizar e unificar os protocolos de comunicação no âmbito da Internet das Coisas. Contudo, a existência de uma proporção significativa de tecnologias, resultante dos avanços obtidos por diversos fabricantes, têm dificultado a sua padronização, o que, por conseguinte, torna a interoperabilidade entre sistemas deste domínio num dos seus principais desafios [42].

Devido à ampla diversidade de protocolos, cada qual com características distintas, torna-se necessário investigar as diferentes opções disponíveis, de modo a obter uma base sólida de conhecimento que permita proceder à escolha de um conjunto de tecnologias que melhor se adequem a um caso específico. Por este motivo, além de fornecerem diferentes tecnologias de comunicação, as plataformas devem ser capazes de auxiliar os seus utilizadores na seleção das tecnologias mais adequadas de acordo com a solução que pretendem projetar [18].

Analisando as tecnologias ao nível da camada de aplicação, dois dos principais protocolos de comunicação utilizados no atual contexto são o MQTT e o CoAP [35], [43].

2.5.1 CoAP

Constrained Application Protocol (CoAP) é um protocolo projetado pela comunidade *Internet Engineering Task Force* (IETF) que atua na camada de aplicação do modelo TCP/IP. Foi desenvolvido com o propósito de permitir a troca de mensagens de forma segura e confiável entre dispositivos com recursos limitados e em redes restritas[9] [44]. O CoAP é utilizado em comunicações máquina-a-máquina (M2M), apresentando um modelo de comunicação semelhante ao do protocolo HTTP, conhecido como modelo cliente/servidor. Ao efetuar um pedido CoAP associado a um determinado método (GET, POST, PUT, DELETE), o cliente poderá solicitar uma ação num determinado recurso identificado por um *Universal Resource Identifier* (URI). Diferentemente do protocolo HTTP, o CoAP atua sobre o protocolo UDP para a troca de mensagens entre o cliente e o servidor (Figura 22).

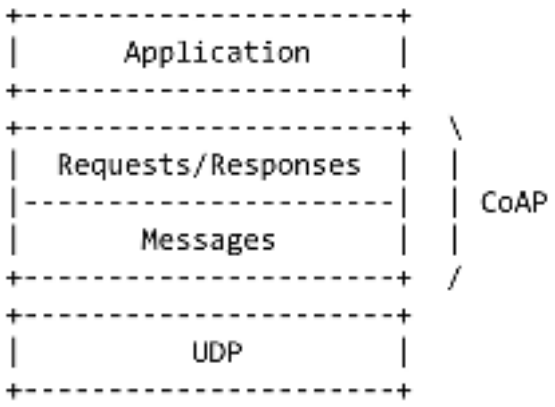


Figura 22 - Camadas abstratas do CoAP [45]

O protocolo COAP dispõe de quatro tipos de mensagens:

1. *Confirmable* (CON): o remetente da mensagem aguarda a confirmação após o seu envio;
2. *Non-Confirmable* (NON): o remetente da mensagem não aguarda a confirmação.
3. *Acknowledgement* (ACK): confirmação da receção de uma mensagem do tipo CON.
4. *Reset* (RST): a mensagem do tipo CON ou NON não foi processada pelo destinatário.

A troca de mensagens do tipo CON pode ser semelhante ao esquema apresentado na Figura 23.

[9] Redes com baixa largura de banda e conectividade limitada.

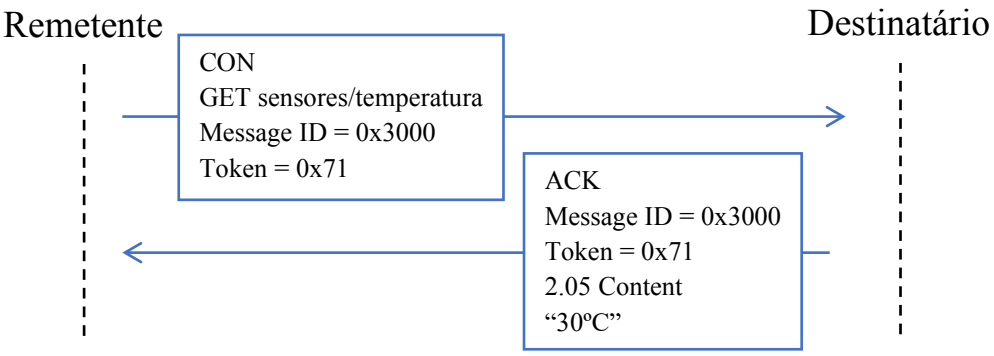


Figura 23 - Exemplo ilustrativo de uma comunicação do tipo CON

Na Figura 23, a comunicação inicia com o envio de uma mensagem do tipo CON pelo remetente ao destinatário. Esta mensagem possui o URI-Path "sensores/temperatura" e utiliza o método GET. O parâmetro "*Message ID*" armazena um valor de 16 bits com a função de identificar a mensagem, prevenindo assim que o destinatário receba mensagens duplicadas. O parâmetro "*Token*" é usado para correlacionar solicitações e respostas. Logo após a receção da mensagem, o destinatáriodeve responder com um *Acknowledgement* (uma mensagem de reconhecimento), que contém o mesmo *Message ID* e *Token* do pedido do remetente, bem como os dados solicitados. Adicionalmente, a resposta possui um código de mensagem, que neste exemplo, é o "2.05 *Content*" (assemelham-se aos códigos de estado HTTP). A mensagem do tipo CON é utilizada para garantir a entrega entre remetente e destinatário.

No caso das mensagens do tipo NON (Figura 24), o comportamento de comunicação é similar ao tipo CON, com a diferença de que o remetente envia uma mensagem NON para o destinatário. No entanto, se a mensagem não for entregue com sucesso, o remetente não irá retransmiti-la automaticamente. Este tipo de mensagem é aconselhável para pedidos que não requerem confirmação ou retransmissão [44].

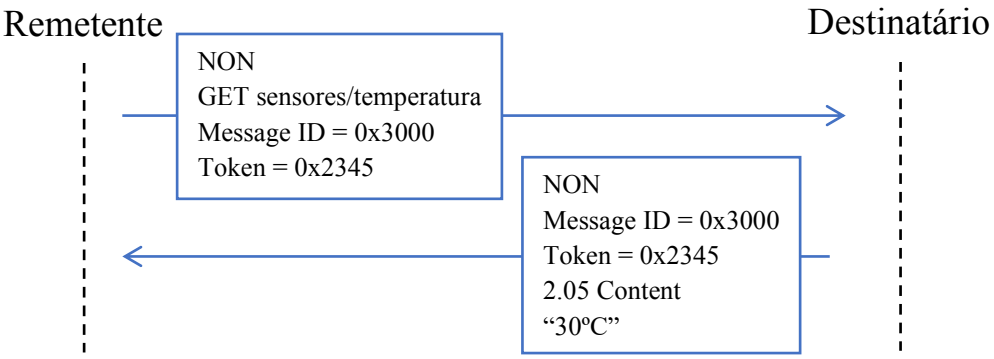


Figura 24 - Exemplo ilustrativo de uma comunicação do tipo NON

2.5.2 MQTT

O protocolo *Message Queue Telemetry Transport* (MQTT) é um protocolo de comunicação entre máquinas (M2M) que atua na camada de aplicação. Desenvolvido pela IBM, o MQTT possibilita a comunicação entre dispositivos com recursos limitados e em redes não confiáveis. Ao contrário do CoAP, este protocolo opera sobre o protocolo TCP, assegurando a confiabilidade das mensagens e comunicações bidirecionais entre os nós [46], [47].

A arquitetura deste protocolo baseia-se no paradigma de publicação/subscrição, implementado por um *middleware* designado de *Broker*. Constituído por tópicos (ou endereços), um MQTT *Broker* garante a troca de mensagens entre os diferentes nós conectados nele próprio. Assim que uma mensagem é publicada num determinado tópico, os nós subscritos a esse endereço receberão a mensagem do *Broker,* tal como é ilustrado na Figura 25. Importante salientar que um único nó pode estar subscrito simultaneamente em vários tópicos. [47], [48].

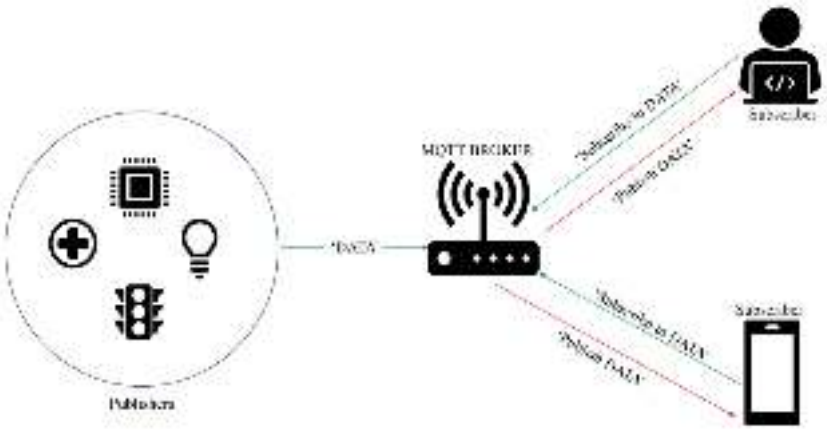


Figura 25 - Neste exemplo, o broker garante a entrega das mensagens publicadas no tópico 'DATA' aos correspondentes nós subscritores [49]

Um MQTT *Broker*, além de controlar o tráfego gerado pelos participantes, implementa também alguns mecanismos de segurança no transporte de mensagens. Todas as comunicações entre nós podem ser cifradas através do protocolo TLS, garantindo assim a integridade e privacidade dos dados contidos na mensagem [50].

O MQTT fornece um mecanismo chamado Nível de Qualidade de Serviço (QoS). O QoS consiste num acordo entre o remetente e o destinatário de uma mensagem, que determina o nível de confiabilidade na entrega da mensagem a enviar-se. O QoS é composto por três níveis de segurança [47], [48]:

1. **QoS 0** (*at most once*): cada mensagem é enviada no máximo uma vez, o que significa que neste nível, não há garantia que a mensagem seja entregue ao destinatário. Este nível é normalmente dominado de *"fire and forget"* pelo facto de a mensagem não ser armazenada ou retransmitida pelo remetente.
2. **QoS 1** (*at least once)*: cada mensagem é entregue pelo menos uma vez, sendo necessário a confirmação da sua receção por parte do destinatário. O remetente mantém a mensagem armazenada até que o destinatário confirme a sua receção. Mensagens duplicatas podem ocorrer.
3. **QoS 2** (*exactly once)*: Através de um mecanismo de *handshake* de quatro vias entre o remetente e o destinatário, é assegurada a entrega da mensagem exatamente uma vez.

Quanto maior o nível de QoS, maior será a confiabilidade na entrega de uma mensagem. Contudo, é importante destacar que o aumento do nível de QoS também acarreta algumas desvantagens, como o aumento de latência e da largura de banda, o que consequentemente resultará num aumento do consumo de energético [47].

2.6 Tecnologias de comunicação de longo alcance

Socialmente, as tecnologias de comunicação desempenham um papel vital na conexão e troca de informações. Elas oferecem uma variedade de meios de comunicação, desde a comunicação por voz e por escrito, até à transmissão de imagens e vídeos. Ao longo dos anos, estas tecnologias evoluíram e tornaram-se cada vez mais sólidas e eficientes.

Entre os tipos de tecnologias de comunicação existentes, atualmente continuam a existir meios mais tradicionais, como telefones e correios, e meios mais recentes, como comunicações sem fios. As tecnologias sem fios, tais como Wi-Fi e o Bluetooth, têm ganho cada vez mais popularidade devido à sua facilidade de uso e ao seu baixo custo. Com o aparecimento da Internet das Coisas, este tipo de tecnologias ganhou ainda mais adeptos.

O aumento da adoção de tecnologias IoT em vários setores, como na saúde, transporte, indústria, agricultura e automação residencial, provocou o aparecimento novas tecnologias de comunicação, tais como as tecnologias sem fios de longo alcance. Estas tecnologias permitem a comunicação entre dispositivos que se encontram distantes entre si, geralmente em zonas remotas. Alguns exemplos de tecnologias de comunicação de longo

alcance incluem o LoRa (*Long Range*), Sigfox, Zigbee, e as *Wide Area Networks* (WANs) como o 4G e 5G.

Nas duas seguintes secções, são exploradas as tecnologias de comunicação LoRa e GFSK (*Gaussian Frequency Shift Keying*). Como se pretende efetuar testes em ambiente de campo, estas duas tecnologias foram escolhidas por proporcionarem comunicações virtualmente gratuitas, e por possuírem uma favorável disponibilidade e acessibilidade no mercado.

2.6.1 LoRa

A tecnologia LoRa (*Long Range*) foi desenvolvida pela empresa Semtech, responsável pelo desenvolvimento de componentes eletrónicos. A LoRa Alliance, uma associação de indústrias, foi criada em 2015 para promover e desenvolver a tecnologia LoRa, com o objetivo de padronizar e ampliar sua utilização em diferentes setores [51].

O LoRa é uma técnica de modulação de espectro espalhado que se baseia numa outra técnica de modulação, chamada de *Chirp Spread Spectrum* (CSS). Esta técnica consiste em utilizar sinais de rádio de baixa potência, conhecidos como *chirps*, para transmitir dados com um alcance significativo. Os *chirps* são sinais de rádio curtos e de alta frequência que variam continuamente, permitindo a transmissão de dados através de várias frequências. Esta técnica de modulação possui uma apreciável robustez contra interferências, uma vez que opera abaixo do nível de ruído [52].

Existem cinco importantes parâmetros associados ao LoRa: a frequência de transmissão (*Carrier Frequency* - CF), potência de transmissão (*Transmission Power* - TP), fator de espalhamento (*Spreading Factor* - SF), largura de banda (*Bandwidth* - BW) e taxa de código (*Coding Rate* - CR).

1. **Frequência de transmissão (CF)**: é a frequência em que o sinal LoRa é transmitido. O LoRa opera em frequências abaixo de 1 GHz, podendo a sua frequência de operação ser definida entre 137 MHz e 1020 MHz. Contudo, as frequências comummente mais utilizadas são a de 433 MHz na Ásia, 868 MHz na Europa e 915 MHz nos Estados Unidos [52], [53].
2. **Potência de transmissão (TP)** é a potência de saída do transmissor. O TP do LoRa pode ser configurado até 20 dBm. Este parâmetro pode ser ajustado para aumentar o alcance da transmissão, diminuindo consequentemente a eficiência energética [54].

3. **Fator de Espalhamento (SF)** é um parâmetro utilizado para ajustar a relação sinal-ruído e a sensibilidade do recetor. O SF define o número de símbolos *chirps* enviados por segundo, podendo ser configurado de 6 a 12. Quanto maior o SF, maior será a sensibilidade do recetor, permitindo assim um maior alcance de comunicação e aumentando consequentemente o tempo de antena [53].
4. **Largura de Banda (BW)** refere-se à largura do espectro de frequência utilizada na transmissão. No LoRa as frequências podem ser configuradas entre 7,8 kHz e 500 kHz [54]. Quanto maior a BW, maior será a taxa de dados, resultando num menor tempo de antena. No entanto, valores de BW mais baixos permitem maior sensibilidade, porém com uma taxa de transferência de dados menor [53].
5. **Taxa de Código (CR)** é um parâmetro utilizado para aumentar a robustez da transmissão contra ruídos. Quanto maior for o CR, maior será a robustez, porém menor será velocidade de transmissão [53].

Ao ser comparada com soluções IoT que utilizam Wi-Fi ou Bluetooth, esta tecnologia destaca-se por possuir um alcance de comunicação superior e uma largura de banda mais ampla. Além disso, a técnica de modulação CSS consiste em espalhar o sinal de transmissão de forma pseudoaleatória, dificultando a interceção ou decodificação do sinal por parte de terceiros. Esta medida proporciona um reforço na segurança dos sistemas LoRa, quando complementada com outras medidas de segurança, como criptografia e autenticação [52].

2.6.2 GFSK

A modulação Gaussiana FSK (GFSK – *Gaussian Frequency Shift Keying*) é uma técnica de modulação digital, utilizada em comunicações sem fios, especialmente em dispositivos de baixo custo e baixo consumo de energia como o Bluetooth [55].

Para compreender o funcionamento do GFSK, é crucial conhecer a técnica de modelação da qual o mesmo deriva, a FSK. A modelação FSK baseia-se no deslocamento da frequência do sinal que é transmitido (frequência portadora), atribuindo-lhe diferentes frequência em função do bit que é transmitido. Por exemplo, ao transmitir-se um bit de valor zero, a frequência do sinal emitido assume uma frequência correspondente a um bit igual a zero durante o período em que é transmitido. Analogamente, quando um bit de valor um é transmitido, a frequência do sinal transmitido é modificada para um valor correspondente,

permanecendo nesta frequência durante o período de transmissão do bit, conforme o ilustrado na Figura 26.

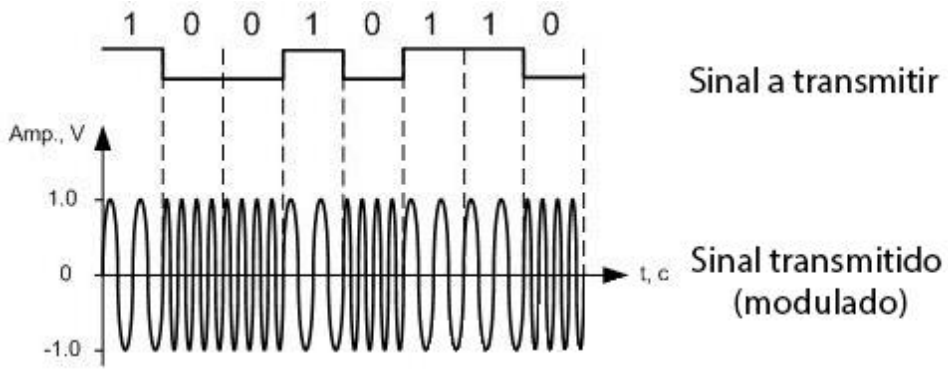


Figura 26 - Modulação FSK (Adaptado de [56])

A modulação GFSK é semelhante à modulação FSK, com a diferença que os sinais passam por um filtro antes de serem transmitidos. Este filtro, chamado filtro gaussiano, ajuda a reduzir a largura do sinal e a suavizar a transição entre os sinais, garantindo assim uma maior eficiência na transmissão de sinal em relação à modulação FSK [57].

2.6.3 Licenciamento e Disponibilidade Espectral

As bandas de frequência ISM (*Industrial, Scientific, and Medical*) são uma série de frequências de rádio (6,78 KHz até aos 245 GHz) não licenciadas e reservadas internacionalmente para o desenvolvimento industrial, científico e médico. Cada região possui a sua própria regulamentação para a utilização destas frequências de rádio, especificando a potência máxima de transmissão, ciclo de trabalho e periodicidade do sinal [58].

A regulamentação de bandas isentas de licença na Europa é uma questão que envolve a Comissão Europeia (CE), o Comité Europeu de Comunicações (ECC) e o Instituto Europeu de Padrões de Telecomunicações (CEPT), que trabalham em estreita colaboração para estabelecer as normas e regulamentações para o uso de bandas de frequência não licenciadas. As medidas resultantes, tanto podem ser impostas pela CE, como podem ser simplesmente recomendações do ECC, cabendo a cada país europeu decidir adoção ou não dessas medidas [59]. Com base na regulamentação europeia, a ANACOM[10] disponibiliza um extenso documento contendo o quadro de atribuição de frequências isentas de licenciamento [60].

[10] A Autoridade Nacional de Comunicações (ANACOM) é a autoridade reguladora nacional (ARN) que tem por missão a regulação do sector das comunicações em Portugal.

As técnicas de modulação discutidas nos capítulos anteriores utilizam frequências de rádio que se enquadram no espectro de bandas de frequência ISM. No entanto, a distribuição das bandas de frequência ISM podem variar de acordo com o país ou região, podendo a sua atribuição destinar-se para diferentes propósitos de acordo com as regulamentações locais. Por exemplo, na América do Norte a frequência livre de licença e compatível com comunicações LoRa situa-se entre a faixa dos 902-928 MHz, enquanto na Europa esta faixa não licenciada situa-se nos 433 MHz e entre os 863-870 MHz [61].

A recomendação ERC/REC70-03 define os parâmetros de utilização espectral para aplicações SRDs (*Short Range Devices*) na Europa. Estes requisitos incluem: bandas de frequência, potência máxima difundida (PMD), largura de banda máxima e ciclo de trabalho (*duty cycle*) [62]. A Tabela 3 apresenta alguns dos parâmetros de utilização espectral estipuladas no ERC/REC70-03.

Tabela 3 - Parâmetros de utilização espectral para dispositivos não específicos de curto alcance na Europa [62]

Frequência (MHz)	PMD	*Bandwidth*	*Duty Cycle*
433.05 – 434.79	10 mW	Não especificado	≤ 10%
433.05 – 434.79	1 mW	Não especificado	Nenhuma exigência
434.04 – 434.79	10 mW	≤ 25 kHz	Nenhuma exigência
862 – 863	25 mW	≤ 350 kHz	≤ 0,1%
865 – 868	25 mW	Não especificado	≤ 1%
869.4 – 869.65	500 mW	Não especificado	≤ 10%

Como referido anteriormente, as modulações Lora e GFSK permitem efetuar comunicações de longo alcance. No entanto, é importante destacar que quando estas técnicas de modulação são aplicadas em dispositivos de comunicação, esses dispositivos são geralmente classificados como SRDs. Isto deve-se ao facto destas duas técnicas de modulação apresentarem um baixo risco de interferência a outros serviços de rádio devido à baixa potência irradiada nas suas comunicações.

2.7 Microcontroladores ESP32

A série de microcontroladores ESP32 é composta por diversos chips, denominados pela Espressif de SoCs (*system-on-a-chips*), que são implementados em diferentes módulos. Apesar do seu tamanho reduzido, todos os SoCs da série ESP32 são constituídos por um processador de 32 bits, 448 KB de memória ROM, 520 KB de memória SRAM, diferentes interfaces para periféricos (4x SPI, 2x I^2C, 3x UART, etc.), duas tecnologias de comunicação sem fios (Wi-Fi e Bluetooth), temporizadores e recursos de segurança (como cifragem da memória *flash* e acelerador de hardware para algoritmos criptográficos) [63], [64]. Em resumo, na Figura 27 é apresentado todos os blocos funcionais do ESP32.

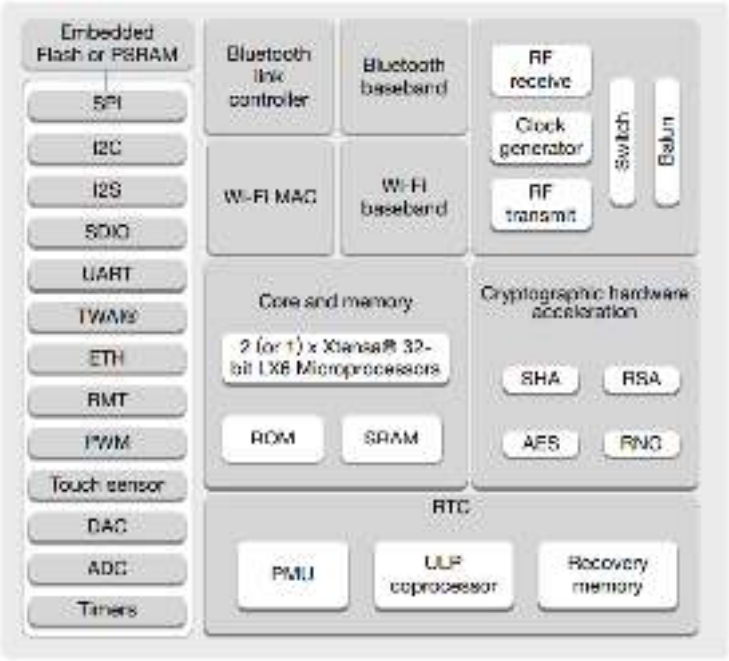


Figura 27 - Diagrama de Blocos Funcionais [63]

Por norma, os módulos fornecidos pela Espressif são constituídos por três principais componentes: SoCs (Figura 28, sinalizado a amarelo), memória *flash* (sinalizado a azul) e uma antena Wi-Fi (sinalizado a vermelho).

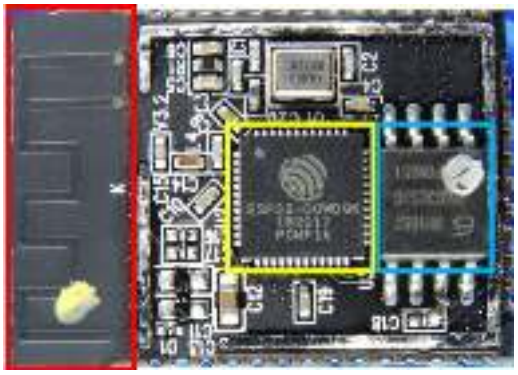


Figura 28 - Módulo ESP32 (sem a *shield*) (Adaptado de [65])

A Espressif fornece mais de uma centena de diferentes módulos, sendo que as principais diferenças residem no SoC utilizado, no tamanho da memória *flash*, nas interfaces de comunicação fornecidas, no tipo de antena Wi-Fi e no número de GPIO (*General Purpose Input/Output*) [66]. Embora os módulos sejam uma solução conveniente para os utilizadores que desejam desenvolver as suas soluções IoT, o uso destes módulos pode originar um aumento de tempo de desenvolvimento e complexidade do projeto. Isto deve-se à possível necessidade de integrar os respetivos módulos com outros componentes e interfaces, visto que os módulos em si não estão otimizados para este tipo de integração.

Para alguns utilizadores, o uso destes módulos poderá ser inconveniente, podendo atrasar o processo de desenvolvimento. Assim, é disponibilizado pela fabricante uma alternativa: os kits de desenvolvimento. Estes kits são pacotes completos de ferramentas que ajudam os utilizadores a criar soluções eletrónicas com uma maior facilidade. Geralmente, incluem uma placa de desenvolvimento (exemplo na Figura 29), ferramentas de desenvolvimento de software e documentação, permitindo assim que os utilizadores se concentrem na criação da solução a implementar.

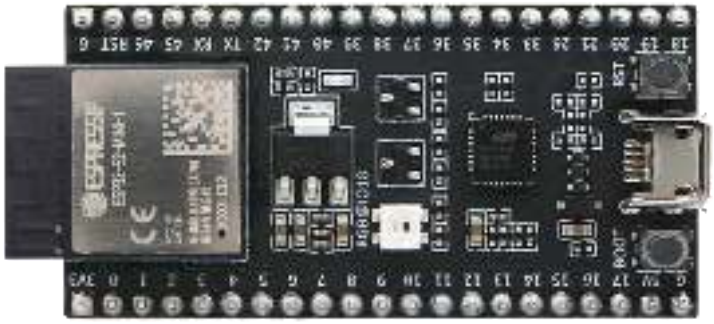


Figura 29 - Exemplo de uma placa de desenvolvimento ESP32 [67]

Além do módulo ESP32, a placa de desenvolvimento inclui essencialmente dois interruptores (um para *reset* do microcontrolador e outro para entrar no modo de programação), um conversor USB-Serial que conecta a interface serial do módulo à interface USB da placa, permitindo ao utilizador carregar facilmente qualquer tipo de *firmware* no módulo. As portas GPIO do módulo também são disponibilizadas nas extremidades laterais da placa de desenvolvimento para facilitar o acesso às mesmas.

No que diz respeito às ferramentas de desenvolvimento de software, existem diversas opções disponíveis, oferecidas tanto pela própria Espressif, como pela vasta comunidade em torno da Internet das Coisas. Estas ferramentas são abordadas no próximo subcapítulo.

2.7.1 Ferramentas de Desenvolvimento

O ESP32 é um microcontrolador que suporta uma ampla variedade de ferramentas de desenvolvimento, desde IDEs[11] até SDKs, que ajudam a simplificar o processo de conceção de uma solução IoT. Existem diversos ambientes de desenvolvimento à disposição do utilizador, aptos a oferecerem recursos que auxiliam a programação entre diversas linguagens, tais como o Arduino IDE, Visual Studio Code e Eclipse. Estes ambientes são indispensáveis no desenvolvimento de aplicações, especialmente quando se trata do microcontrolador em questão. O ESP32 é compatível com várias linguagens de programação, incluindo-se o C/C++, Python, Javascript, Lua, etc., o que permite aos programadores escolherem a linguagem que melhor se adapta às suas necessidades e preferências. A compatibilidade com estas diferentes linguagens só é possível graças à existência de interpretadores e compiladores, responsáveis por executar ou traduzir o código fonte em código de máquina. Alguns exemplos de tais ferramentas de desenvolvimento são:

1. Arduino: trata-se de uma plataforma de código aberto que disponibiliza tanto hardware quanto software integrado. A programação do hardware é realizada utilizando uma versão adaptada de C++ (baseada na *framework* Wiring[12]). Para auxiliar na programação, a plataforma oferece uma IDE[13] que integra várias ferramentas de desenvolvimento que permitem ao utilizador projetar, compilar e carregar os seus projetos [68]. Para realizar estas operações, por padrão a IDE do Arduino utiliza um conjunto de ferramentas de software (*toolchain*) fornecidas pela Atmel AVR.

2. Espressif IoT Development Framework (ESP-IDF): é uma *framework* de desenvolvimento fornecida pela Espressif, projetada para a programação dos SoCs ESP32. Esta *framework* de código aberto, utiliza a linguagem C como base para a criação de aplicações e soluções IoT. O ESP-IDF fornece uma estrutura de desenvolvimento integral que inclui bibliotecas de baixo nível para o acesso aos recursos do microcontrolador, assim como bibliotecas de alto nível destinadas a tarefas como conectividade Wi-Fi e gestão de eventos. Por fim, são também

[11] *Integrated Development Environments* (Ambientes de Desenvolvimento Integrado)
[12] O Wiring é uma plataforma de desenvolvimento que disponibiliza uma estrutura de software e hardware para a criação de projetos relacionados com microcontroladores. Mais informações em https://wiring.org.co/
[13] A IDE desenvolvida pelo Arduino é baseada na IDE fornecida pela plataforma Processing, que é utilizada para criar imagens, animações e interações de forma intuitiva [68].

fornecidas ferramentas para ajudar na configuração, implementação e compilação da aplicação que será carregada no dispositivo [69].

3. Espruino: é um interpretador de linguagem JavaScript destinado para microcontroladores, e projetado para dispositivos com apenas 128 kB de memória Flash e 8 kB de memória RAM [70].
4. MicroPython: é um interpretador de linguagem Python que possui um prompt interativo (REPL) para execução de comandos, bem como a capacidade de executar e importar *scripts* diretamente do sistema de arquivos integrado. Um dos seus principais objetivos é tornar-se o mais compatível possível com a linguagem Python, de modo a tornar a sua experiência de utilização o mais padronizado com a linguagem de origem. Além de implementar algumas das principais bibliotecas de Python, este interpretador inclui bibliotecas de baixo nível para aceder ao hardware do microcontrolador [71].

3 Abordagem de Implementação

As plataformas estão em constante evolução, tornando-se cada vez mais escaláveis, interoperáveis e autónomas. Com a análise realizada nas secções 2.3 e 2.4, concluiu-se que o foco de desenvolvimento das plataformas incide sobretudo nas APIs (incluindo-se o armazenamento de dados) e na IU. Já em relação aos dispositivos, tal como foi demonstrado no decorrer do Capítulo 2, é comum que as plataformas forneçam apenas SDKs ou bibliotecas, que incluem exemplos de códigos padrão e documentação para ajudar no processo de integração. Neste sentido, cabe ao utilizador integrar o dispositivo na plataforma, utilizando o SDK ou as bibliotecas fornecidas. Por norma, este é um processo divido essencialmente em três partes:

1. Encontrar um dispositivo compatível: o primeiro passo é encontrar um dispositivo que seja compatível quer com a solução que o utilizador pretende implementar, quer com os requisitos da plataforma escolhida.
2. Obter o SDK ou biblioteca de integração: antes de conectar o dispositivo, o utilizador deve obter o SDK ou a biblioteca de integração à plataforma. Para a realização desta tarefa, normalmente é necessário consultar a documentação fornecida pela própria plataforma.
3. Integrar o dispositivo com a plataforma: a próxima etapa e última etapa consiste em estabelecer a comunicação do dispositivo com a plataforma. Além da solução que pretende implementar, o utilizador deve também adicionar o código de conectividade ao *firmware* do seu dispositivo, utilizando para esse efeito, o SDK e as bibliotecas disponibilizadas pela plataforma.

Parte deste processo pode ser visto como desnecessário para quem pretende desenvolver uma solução IoT, podendo até funcionar como um desincentivo no caso de utilizadores menos experientes.

No que diz respeito aos dispositivos, torna-se notório que as plataformas não fornecem uma abordagem tão consistente à que normalmente proporcionam nas APIs e IUs. Com base nisso, pretende-se desenvolver um sistema IoT que sirva de proposta para uma abordagem mais consistente entre os dispositivos e plataformas. Nos próximos subcapítulos, é descrito a metodologia utilizada de forma a atingir os objetivos propostos.

3.1 Proposta do Sistema IoT

O sistema a desenvolver será constituído pelos dois componentes fundamentais representados na Figura 3 (capítulo 2): a Plataforma e os Dispositivos. A plataforma a desenvolver deverá ser composta por APIs que disponibilizarão serviços a serem utilizados quer pelos utilizadores quer pelos dispositivos. Para tal, os utilizadores terão à disposição uma interface que lhes permitirá aceder a estes serviços. Todas as informações relevantes, provenientes tanto dos dispositivos como dos utilizadores, serão armazenadas nas BDs geridas pelas APIs.

Quanto aos dispositivos, pretende-se que sejam incorporados com tecnologia Wi-Fi para que possam conectar-se e comunicar com a plataforma. Como estes dispositivos devem ser utilizados essencialmente pelos utilizadores para projetarem as suas soluções IoT, é importante que os mesmos suportem outras tecnologias de comunicação, como Bluetooth, I^2C, SPI, entre outras, de modo a serem compatíveis com o maior número de cenários possíveis de soluções projetadas pelos utilizadores.

Pretende-se também que a programação do dispositivo seja efetuada via *wireless* (sem fios), podendo o mesmo ser programado com diferentes arquiteturas de linguagem. O maior desafio na implantação desta abordagem reside justamente no dispositivo, que deve suportar e executar duas principais funções:

1. Solução do utilizador: o dispositivo deve executar as soluções IoT idealizadas pelo utilizador.
2. Conectividade à plataforma: o dispositivo deve, autonomamente, permanecer conectado à plataforma e receber as atualizações dos *firmware* produzidos pelos utilizadores com diferentes linguagens de programação.

Estes dois principais segmentos de código, além de serem implementados e coexistirem na memória do dispositivo, devem ser executados em paralelo. Analogamente, é como se tratasse de dois diferentes *firmwares*, armazenados em distintos endereços da memória do dispositivo, e fossem ambos executados simultaneamente.

Adicionalmente, embora normalmente seja da responsabilidade do utilizador desenvolver a sua própria solução IoT, o mesmo não se aplica ao segmento referente à conectividade com a plataforma. No sistema proposto, o objetivo passa por se disponibilizar meios que permitam aos seus utilizadores concentrarem-se exclusivamente na criação da solução, sem se preocuparem com questões técnicas relacionadas com a plataforma. Logo,

é necessário que haja um *firmware* de origem pré-instalado no dispositivo que se encarregue deste tipo de questões.

Para satisfazer as exigências mencionadas no parágrafo anterior, propõe-se que o dispositivo seja constituído por dois microcontroladores (MCUs), denominados Master e Slave (Figura 30).

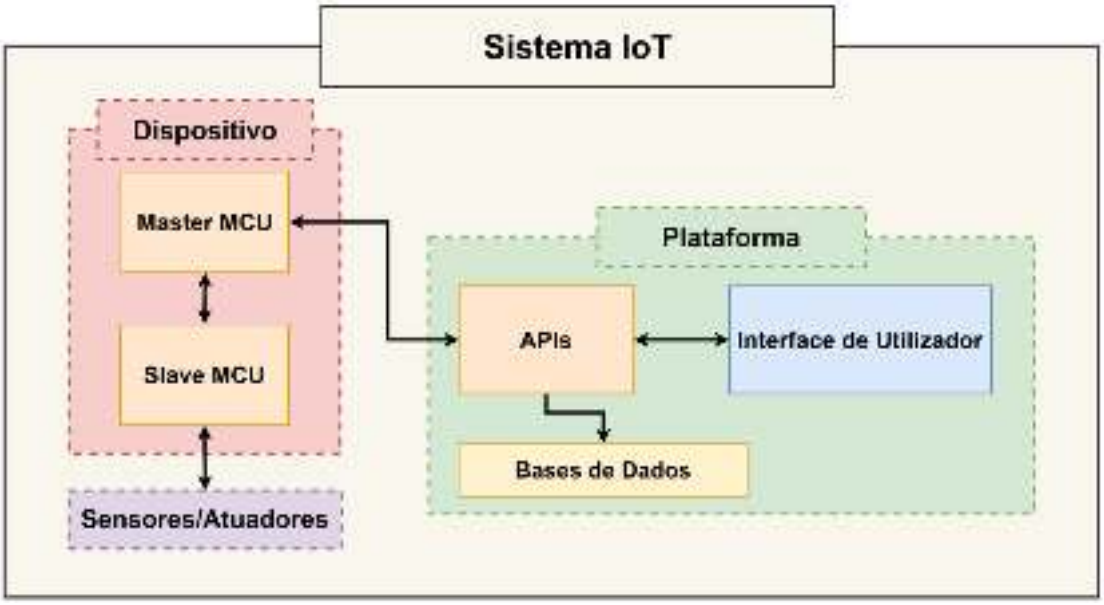


Figura 30 - Composição do sistema IoT a desenvolver

Sinteticamente, o microcontrolador Master ficará responsável pela conectividade do dispositivo à plataforma, devendo possuir apenas o *firmware* pré-instalado pelos responsáveis do desenvolvimento do sistema. Este microcontrolador deverá receber exclusivamente atualizações de *firmware* desenvolvidas pelo fabricante, não sendo permitido aos utilizadores qualquer modificação ao software de origem. Essa função será atribuída ao microcontrolador Slave, que ficará responsável por receber e executar o *firmware* desenvolvido pelo utilizador.

Para além da conectividade à plataforma, a transferência de todos os dados entre a plataforma e o dispositivo deve ser realizada pelo Master MCU, funcionando assim como uma "ponte de comunicação" entre estes dois componentes do sistema. A Figura 31 demonstra um exemplo prático deste processo, onde o Slave MCU lê a humidade num determinado ambiente através de um higrómetro, processando os dados adquiridos e enviando-os para o Master MCU que, por sua vez, os enviará para a plataforma.

Figura 31 - Processo de comunicação do sistema

Em relação às APIs, para serem eficazes, é fundamental que sejam oferecidos os serviços necessários para alcançar os objetivos propostos. Estes componentes devem fornecer suporte às funcionalidades necessárias tanto para os dispositivos quanto para os seus utilizadores. No sistema a projetar, as APIs são consideradas elementos centrais que possibilitam a comunicação entre os diferentes componentes do sistema, permitindo o fornecimento de dados e informações. Caso ocorra algum problema que prejudique o seu funcionamento, a execução do resto do sistema também ficará em risco. Dada a sua importância, o desenvolvimento das APIs da plataforma seguirá os padrões estabelecidos e utilizados numa abordagem de microsserviços.

Ao contrário da abordagem monolítica, em que a aplicação é desenvolvida num único software, na abordagem de microsserviços a aplicação é dividida em pequenas partes independentes de software (denominadas por microsserviço). Cada microsserviço desenvolvido terá a responsabilidade de lidar com um pequeno conjunto de funcionalidades atribuídas. Quando comparada à abordagem monolítica, a abordagem de microsserviços traz os seguintes benefícios:

1. Acoplamento: as aplicações monolíticas são altamente acopladas, o que significa que uma mudança numa parte da aplicação pode afetar outras partes. Por outro lado, como os microsserviços são desacoplados, as mudanças efetuadas num microsserviço não afetam os outros microsserviços.
2. Escalabilidade: as aplicações monolíticas são horizontalmente escaláveis, ou seja, novas instâncias são adicionadas na aplicação como um todo, o que pode ser considerado um processo dispendioso e complexo. Em contrapartida, os microsserviços são verticalmente escaláveis, o que significa que novos recursos são

adicionados apenas num microsserviço em específico, tornando a tarefa do programador mais facilitada e eficiente.

3. Tecnologias: uma aplicação monolítica geralmente segue um padrão de tecnologias utilizadas, o que pode limitar a escolha de tecnologias e ferramentas disponíveis para o seu desenvolvimento. Com os microsserviços, como são softwares independentes, as aplicações podem ser construídas utilizando diferentes tecnologias, consoante as suas necessidades.
4. Robustez: a abordagem de microsserviços torna as aplicações mais robustas quando comparadas às aplicações monolíticas. Como cada microsserviço é independente e responsável por um pequeno grupo de funcionalidades, a falha de um microsserviço não afetará os outros microsserviços inseridos no sistema. Isto significa que a aplicação pode continuar a funcionar normalmente, mesmo se um microsserviço falhar, aumentando a robustez e a resiliência da aplicação como um todo.

Na implementação da abordagem de microsserviços no sistema a desenvolver, para se obter os benefícios descritos no parágrafo anterior, é fundamental definir especificamente quais os serviços que serão oferecidos por cada API. Cada API comportar-se-á como um microsserviço e deverá ter a sua própria BD. Será utilizado um ou mais protocolos de comunicação, de modo que os clientes (sejam dispositivos ou utilizadores) possam aceder aos serviços fornecidos por meio do protocolo de comunicação correspondente, tal como é ilustrado na Figura 32.

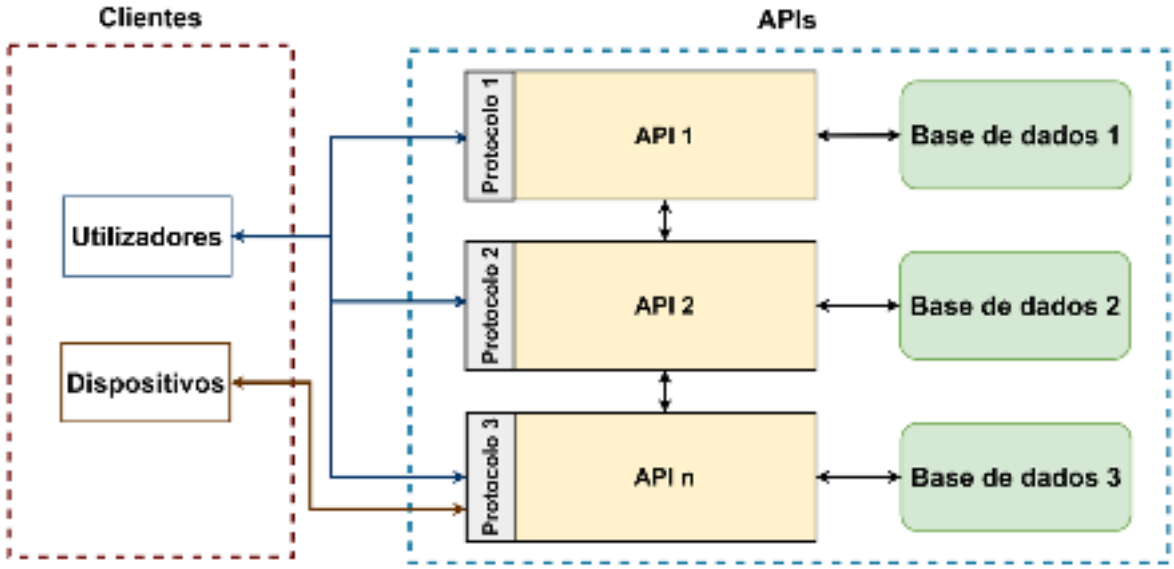


Figura 32 - Abordagem de desenvolvimento das APIs

3.2 Descrição dos Componentes do Sistema

Neste subcapítulo, são apresentados e analisados os componentes que compõem o sistema IoT a ser desenvolvido. Cada componente é descrito em maior detalhe, incluindo a sua função, características, limitações e interações com os demais componentes. Nas próximas secções é fornecida uma visão geral dos elementos que compõem o sistema, justificando-se as decisões tomadas na implementação de cada componente, com o objetivo de cumprir os requisitos inicialmente estipulados do projeto.

3.2.1 Plataforma

Uma plataforma IoT deve ser constituída por um conjunto de ferramentas e tecnologias que viabilizem a comunicação dos componentes inseridos num sistema IoT. Como já discutido no subcapítulo 3.1, uma plataforma pode ser dividida em duas partes:

1. Interface de Programação de Aplicação: software responsável pela conexão entre os clientes e o restante do sistema, permitindo a coleta/distribuição, processamento e armazenamento dos dados do sistema.
2. Interface de Utilizador: componente da plataforma visível e acessível ao utilizador final, consistindo numa interface que permite aos utilizadores proceder ao controlo e gestão dos seus dispositivos.

Nas próximas duas secções, é efetuado o planeamento que estabelece a abordagem de implementação utilizada nestes dois componentes da plataforma. Deste modo, é definido um conjunto de requisitos e estratégias que servirão como um guia de desenvolvimento de ambas as interfaces.

3.2.1.1 Interfaces de Programação de Aplicação

Conforme mencionado no final do subcapítulo 3.1, a abordagem escolhida para a implementação de todas as APIs será baseada na arquitetura de microsserviços. Cada API será responsável por uma coleção de serviços autónomos. Deste modo, é essencial atribuir os serviços que devem ser providenciados por cada API:

1. Gestão de Utilizadores: no sistema a desenvolver, é necessário fornecer um conjunto de serviços responsáveis pela gestão dos utilizadores, de modo a garantir que apenas utilizadores autenticados e autorizados tenham acesso a determinados serviços do sistema.

2. Gestão de Dispositivos: tal como a gestão de utilizadores, é importante ter serviços que permitam a gestão de todos os dispositivos conectados à plataforma. Estes serviços devem permitir, essencialmente, o registo dos dispositivos, autenticação, autorização e associação a utilizadores da plataforma.
3. Transmissão de Dados entre os Clientes: a plataforma deve disponibilizar meios que garantam a comunicação eficiente e precisa entre utilizadores e dispositivos. Assim, para melhorar a experiência de utilização, é importante que a plataforma seja capaz de transmitir todos os dados relevantes em tempo real, permitindo que os utilizadores recebam (ou enviem) informações atualizadas e precisas dos seus dispositivos.
4. Atualização do *Firmware* do Slave MCU: embora a responsabilidade seja do utilizador de carregar o *firmware* a ser executado pelo Slave MCU, a plataforma deve assegurar que o processo de atualização seja realizado de forma simples e eficiente, auxiliando se necessário o utilizador nesta tarefa, de modo a reduzir as hipóteses de interrupções ou falhas durante o processo de atualização de *firmware* do dispositivo.
5. Notificação dos Clientes: é essencial manter os utilizadores e dispositivos informados sobre ocorrências que ocorram no sistema. Exemplo de tais ocorrências podem ser: informar se atualização do *firmware* do dispositivo foi executada com ou sem sucesso, comunicação de problemas que estejam a ocorrer na plataforma, entre outros tipos de ocorrências.

Atendendo às necessidades apresentadas, pretende-se então desenvolver as seguintes cinco APIs (Figura 33) em Node.js: Gestão de Utilizadores, Gestão de Dispositivos, Transmissão de Dados, Atualização de *Firmware* e Notificações. De acordo com as recomendações estipuladas na arquitetura de microsserviços, cada API deverá possuir a sua própria BD independente. As APIs de Transmissão de Dados, Atualização de *Firmware* e Notificações utilizarão uma BD não relacional (MongoDB), enquanto as APIs de Gestão de Utilizadores e de Dispositivos utilizarão uma BD relacional (PostgreSQL).

Durante a tomada de decisão sobre o uso de uma BD relacional ou não relacional, foram levadas em consideração as vantagens oferecidas por ambas para o atual contexto. Por um lado, uma BD relacional é mais adequada para gerir os relacionamentos entre os dados armazenados pelas APIs de Gestão de Utilizadores e de Dispositivos. Por outro lado, uma BD não relacional é mais apropriada para lidar com grandes volumes de dados, como os que podem ser processados pelas APIs de Transmissão de Dados, Atualização de *Firmware* e

Notificações. No caso deste projeto em particular, a escolha de utilizar o MongoDB e o PostgreSQL é essencialmente fundamentada na experiência e conhecimento prévio sobre o uso destas tecnologias de armazenamento de dados.

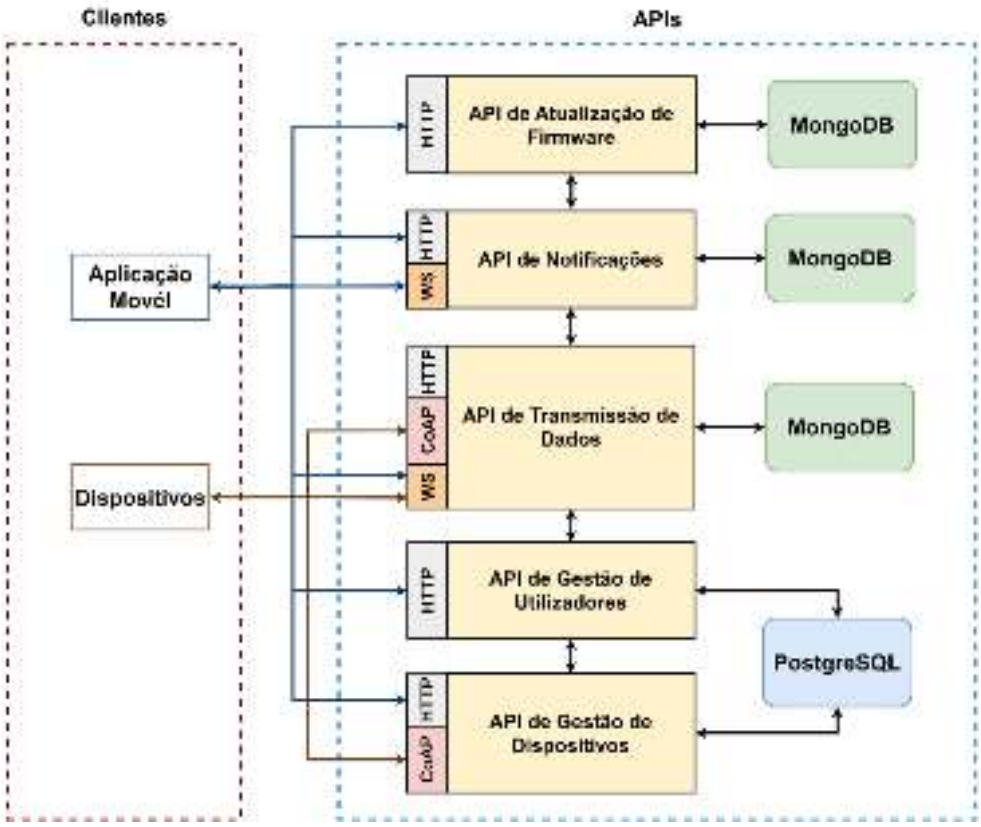


Figura 33 - Escalonamento final da abordagem de desenvolvimento das APIs

Ao analisar-se os objetivos propostos no final do subcapítulo 1.2, conclui-se que o foco deste trabalho não se prende com questões de segurança do sistema. Apesar do atual projeto concentrar-se em fornecer funcionalidades que facilitem e melhorem a usabilidade do sistema IoT, pretende-se que o tópico de segurança não seja totalmente desconsiderado.

Uma medida de segurança que se pretende implementar é o controlo dos acessos às APIs da plataforma a desenvolver. Esta medida é fundamentada em dois fatores:

1. A ausência do controlo de acessos pode conduzir a potenciais ameaças críticas, como acessos não autorizados e violação de informações sensíveis;
2. De modo a validar a usabilidade e praticidade das funcionalidades que se pretende implementar, é importante que o sistema IoT desenvolvido seja comparável com um sistema real.

Para garantir que os acessos efetuados a cada uma das API sejam autorizados, utilizar-se-á o padrão JWT (*JSON Web Token*). O JWT é um padrão aberto que estabelece

uma forma segura de transmitir informações em formato JSON (*JavaScript Object Notation*) entre diferentes sistemas. É comummente utilizado para autenticar utilizadores, permitindo que as aplicações verifiquem a identidade de um utilizador de maneira segura e confiável. Além destas, o JWT possui outras vantagens, tais como:

1. Escalabilidade: o JWT é facilmente escalável, uma vez que não requer armazenamento de estado do lado do servidor. O servidor apenas precisa de ter acesso à respetiva chave privada;
2. Redução de consultas às BDs: visto que as informações necessárias estão contidas no próprio token JWT, não é necessário consultar a BD;
3. Carga útil: as informações não sensíveis podem ser armazenadas no próprio token, permitindo manter a autenticidade e integridade das informações armazenadas.

Com a utilização do padrão JWT para a autenticação e controlo de acesso às APIs, é possível aumentar a integridade e autenticidade dos acessos realizados.

Nos parágrafos seguintes, é realizada uma descrição mais aprofundada de cada uma das cinco APIs a serem desenvolvidas.

1. Gestão de Utilizadores

   A API de Gestão de Utilizadores será responsável pelo registo e autenticação dos utilizadores na plataforma. No registo do utilizador, serão armazenadas algumas informações na BD, tais como: nome do utilizador, endereço de e-mail, senha e data de criação e alteração do registo.

   Na autenticação do utilizador, caso a mesma seja efetuada com sucesso, devem ser gerados dois JWT, que deverão ser enviados e armazenados pela aplicação de IU. Estes dois *tokens,* designados como Token de Acesso de Utilizador e Token de Atualização de Utilizador, serão abordados em maior detalhe no subcapítulo 4.1.1.

   Após o utilizador concluir o processo de autenticação, a API deve ainda permitir a apresentação e alteração dos dados de registo do utilizador, como também, a inserção de uma imagem de perfil. De salientar que no decorrer da autenticação e registo do utilizador, os dados recebidos pela API, só deverão ser armazenados na BD após passarem por um processo de validação.

   Por fim, a comunicação com os serviços fornecidos pela API de Gestão de Utilizadores deverá ocorrer pelo protocolo HTTP.

2. Gestão de Dispositivos

A API de Gestão de Dispositivos fornecerá serviços semelhantes aos da API de Gestão de Utilizadores, com a principal diferença de que esta API será exclusivamente destinada a serviços relacionados com os dispositivos. Esta API deve suportar dois protocolos diferentes: o protocolo HTTP, destinado a responder às solicitações dos utilizadores, e o protocolo CoAP, utilizado para responder às solicitações dos dispositivos.

Em termos de funcionalidades, através do protocolo HTTP, será disponibilizada a possibilidade aos seus utilizadores de associar ou desassociar dispositivos à sua conta, filtrar por dispositivos através de determinados parâmetros e confirmar se um dos seus dispositivos estão ou não online. Esta última funcionalidade será especialmente útil para verificar com quais dispositivos o utilizador pode comunicar em tempo real. Ainda dentro do grupo de serviços mencionados, haverá também um com a responsabilidade de registar novos dispositivos. No entanto, este será um serviço de uso restrito, podendo ser utilizado apenas por utilizadores autenticados.

Nesta API, através do protocolo CoAP, os dispositivos poderão autenticar-se e associar-se aos utilizadores.

3. Transmissão de Dados

A API de Transmissão de Dados intermediará a comunicação dos dados transmitidos entre os dispositivos e os utilizadores. A razão pela qual esta API se encontra posicionada entre os dispositivos e utilizadores, deve-se ao facto de estes estarem geralmente conectados em diferentes redes locais protegidas por NAT. Esta condição impede-os de comunicar diretamente entre si, o que limitaria significativamente o potencial do sistema. Opta-se, assim, pela implementação de uma API para contornar esta adversidade.

Para garantir a eficiência e a fiabilidade da transmissão de dados entre clientes, é essencial que esta API possua robustez suficiente para lidar com múltiplos protocolos de comunicação. Assim, para atender as necessidades que possam surgir por parte dos clientes, esta API suportará três protocolos de comunicação: HTTP (1), WebSocket (2) e CoAP (3).

1. Na implementação desta API, o protocolo HTTP é utilizado para responder às solicitações provenientes dos utilizadores e de outras APIs do sistema.

2. O WebSocket é um protocolo de comunicação bidirecional que reutiliza a conexão inicialmente estabelecida. Desta forma, os utilizadores deverão utilizar este protocolo para transacionarem dados em tempo real com os seus dispositivos.

3. Por último, o CoAP é um protocolo de comunicação especialmente desenvolvido para dispositivos IoT. Nesta API, este protocolo deve ser utilizado pelos dispositivos para encaminhar os dados adquiridos para o utilizador.

Na Figura 34 é apresentado o diagrama de comunicação desta API, o qual evidencia o que foi discutido nos últimos parágrafos:

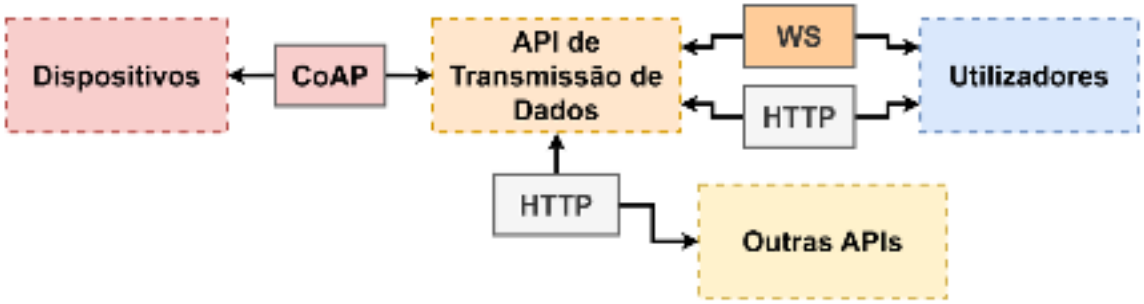


Figura 34 - Diagrama de comunicação da API de Transmissão de Dados

4. Atualização de *Firmware*

A API de Atualização de *Firmware* terá a função de descompilar o processo de atualização de *firmware* dos dispositivos (mais concretamente, do Slave MCU). Com esta API, para atualizar os dispositivos, os utilizadores apenas deverão fornecer os respetivos ficheiros de *firmware* pré-compilados e os dados associados ao processo. Uma vez fornecidos, a API encarregar-se-á do resto do processo de atualização.

O princípio de funcionamento desta API baseia-se em agrupar todos os ficheiros de *firmware* enviados pelo utilizador, convertendo-os para esse efeito num único arquivo binário, denominado por padrão de *spiffs.bin*. Logo após a conclusão deste processo, o ficheiro gerado pela API é enviado para o dispositivo. Todos os dados resultantes deste processo serão armazenados na BD, incluindo os ficheiros de *firmware* enviados pelo utilizador, o ficheiro *spiffs.bin* e os dados associados ao respetivo processo. Ao proceder-se ao armazenamento dos dados resultantes do processo de atualização, para além de se efetuar o registo das atualizações de *firmware* realizadas – o que pode ser útil para o suporte e manutenção da solução desenvolvida pelo utilizador – pretende-se também que, caso seja necessário, os utilizadores possam aceder facilmente a todos os ficheiros de atualização armazenados.

Uma vez que um dos objetivos desta dissertação é fornecer um ecossistema de multilinguagem, é necessário que esta API esteja preparada para lidar com vários tipos de binários, que poderão ser gerados por diferentes ferramentas de desenvolvimento, tais como Arduino, ESP-IDF, Espruino e MicroPython. Destaca-se ainda que as comunicações com os serviços fornecidos pela API correspondente, deverão ser realizadas pelo protocolo HTTP.

5. Notificações

A API de Notificações tem como objetivo notificar os utilizadores sobre eventos ocorridos no sistema IoT, utilizando para esse efeito dois protocolos de comunicação: HTTP (1) e WebSocket (2).

1. Será através do protocolo HTTP que as restantes APIs poderão solicitar a emissão de determinadas notificações. As notificações emitidas devem ser armazenadas por esta API na BD, para que dessa forma, seja possível aceder e consultar as notificações emitidas, bem como filtrá-las com diferentes parâmetros, ou até alterar os dados associados de cada notificação.
2. O protocolo WebSocket tem como função primordial notificar os utilizadores em tempo real sobre a ocorrência de eventos, que estejam relacionados diretamente ou indiretamente com a atividade do utilizador dentro da plataforma.

Na Figura 35 é demonstrado o funcionamento desta API. Sempre que for emitida uma nova notificação, espera-se que esta API armazene a notificação e a encaminhe posteriormente para os utilizadores destinatários.

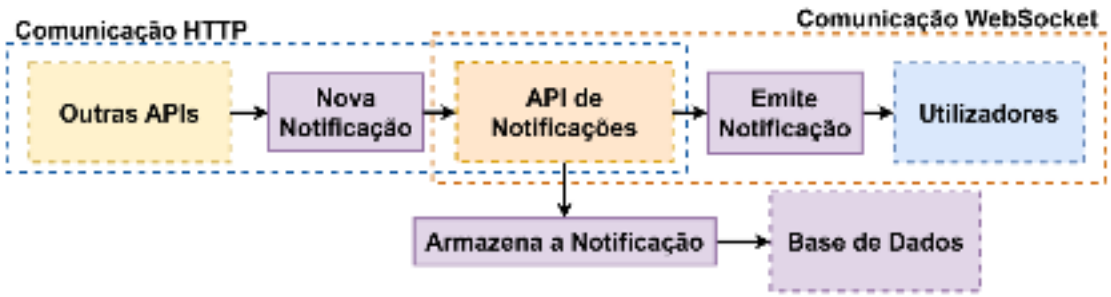


Figura 35 - Processo de funcionamento da API de Notificações

3.2.1.2 Interface de Utilizador

Na plataforma a desenvolver, a IU é uma ferramenta que deve proporcionar uma experiência intuitiva na interação e controlo dos elementos do sistema IoT.

Para testar as funcionalidades pretendidas, será desenvolvida uma interface que deve permitir aos utilizadores programar e controlar os seus dispositivos, além de outras funcionalidades típicas, como o acesso a dados específicos e configurações associadas à conta do utilizador. Também deverá permitir o acesso aos dados em tempo real provenientes dos seus dispositivos. Para garantir uma melhor experiência, é fundamental implementar um sistema de notificações que alerte o utilizador em tempo real, informando-o sobre determinadas ocorrências do sistema.

Para o desenvolvimento desta interface, utilizar-se-á a *framework* React-Native. A escolha de uma *framework* em Javascript deve-se ao facto de ser a mesma linguagem de programação que é utilizada pelo Node.js, permitindo assim que ambos os softwares (APIs e IU) sejam escritos de forma similar. Por si só, esta abordagem garante uma maior uniformidade de código, reduzindo o tempo de desenvolvimento e de futuras manutenções, uma vez que alguns aspetos como a sintaxe, disponibilização de bibliotecas, entre outras características, podem variar dependendo da linguagem de programação utilizada.

Nos próximos parágrafos, realizar-se-á uma projeção da IU a desenvolver, onde serão descritas as páginas que a interface irá conter, bem como as suas funcionalidades e o respetivo layout.

1. <u>Página de Início de Sessão e Registo</u>

   Um simples formulário que permita ao utilizador iniciar sessão com a sua conta, ou, caso ainda não possua uma, efetuar o registo. Na Figura 36 é apresentado o layout das páginas que se pretende projetar: a página de Início de Sessão e de Registo.

Figura 36 - Layout da página de Início de Sessão e de Registo

2. Página de Painel de Controlo

O painel de controlo é a página central da aplicação (Figura 37). Nesta página os utilizadores devem conseguir aceder às informações da sua conta, podendo realizar algumas ações, tais como atualizar o nome, a sua senha, o endereço de e-mail ou a imagem de perfil. O painel de controlo deve permitir aos utilizadores gerir os seus dispositivos, podendo associá-los ou removê-los da sua conta. Por fim, ainda deve ser incluída uma barra de navegação inferior para que os utilizadores possam percorrer as outras páginas da aplicação de forma rápida e intuitiva.

Figura 37 - Layout da página do Painel de Controlo

3. Página de Monitorização de Dados

Figura 38 - Layout da página de monitorização de dados

O monitor de dados (Figura 38) é uma página cujo objetivo é permitir a comunicação remota com os dispositivos associados à conta do utilizador. Ao utilizar este monitor, deve ser possível monitorizar todos os dados enviados pelos dispositivos, recebendo as informações em tempo real. Por exemplo, através deste monitor, o utilizador poderá verificar o estado atual de diferentes atuadores, receber os dados adquiridos pelos sensores ou até enviar comandos para controlar determinados dispositivos associados. De destacar que tanto os dados enviados pelos dispositivos como os comandos enviados pelo utilizador, dependerão do *firmware* carregado no dispositivo Slave MCU, *firmware* esse a ser implementado pelo utilizador.

4. Página de Carregamento de Firmware

Nesta página (Figura 39), deverá ser possível realizar o carregamento de *firmware* no Slave MCU. De modo a facilitar o processo de desenvolvimento desta funcionalidade, serão disponibilizados vários botões de opção, que permitirão ao utilizador selecionar o *firmware* que deseja carregar no Slave MCU. Cada opção conterá um conjunto de ficheiros binários pré-compilados, armazenados localmente no telemóvel, que serão enviados para o dispositivo após o utilizador selecionar uma das opções disponíveis. A

principal diferença entre cada uma das opções, reside no uso de diferentes linguagens de programação no desenvolvimento do *firmware*. O Objetivo desta página passa por demonstrar o conceito de multilinguagem.

É importante destacar que, idealmente, seria desejável que o utilizador fornecesse os ficheiros *firmware* a serem carregados no Slave MCU. No entanto, uma vez que a implementação do conceito de multilinguagem ainda se encontra num estágio inicial, seria inviável avançar de imediato para essa etapa sem antes verificar a sua eficácia. Assim, optou-se por seguir uma abordagem mais prudente, como a descrita no parágrafo anterior.

Figura 39 - Layout da página responsável pelo carregamento do *firmware*

5. Página de Notificações

Figura 40 - Layout da página de apresentação das notificações

Uma simples página de notificações (Figura 40) que deve informar o utilizador de eventos ocorridos no sistema em tempo real. Neste caso em específico, o principal objetivo das notificações é informar o utilizador de que a atualização de *firmware* foi concluída, com ou sem sucesso. Uma vez que o processo atualização pode levar alguns minutos, é importante que ocorra de forma assíncrona. Desta forma, o utilizador poderá realizar outras tarefas enquanto aguarda pela notificação da conclusão deste processo.

3.2.2 Dispositivo

O dispositivo que se pretende projetar deve satisfazer determinados requisitos, de modo a corresponder aos objetivos propostos do sistema que se pretende implementar. A escolha do microcontrolador é um passo fundamental na projeção de um dispositivo IoT, uma vez que é responsável por controlar todas as funcionalidades do dispositivo, como a interação com sensores e atuadores, processamento de dados, comunicação com outros sistemas, entre outras.

Ao considerar a escolha de um microcontrolador, diversos fatores devem ser considerados. Entre eles, destacam-se: a quantidade de interfaces periféricas disponibilizadas, a sua capacidade de processamento, a facilidade de programação e a compatibilidade com diferentes tecnologias de comunicação. Nesse sentido, o ESP32 evidencia-se como uma opção adequada para a projeção deste dispositivo. Por possuir várias interfaces periféricas, o ESP32 demostra uma notável compatibilidade para se conectar com diversos tipos de transdutores. Além do mais, devido ao seu poder de processamento, pode ser utilizado em soluções mais complexas. Em termos de linguagens suportadas, o ESP32 é conciliável com diversas linguagens de programação, o que permite aos seus utilizadores escolherem a linguagem que consideram mais adequada para projetar a sua solução.

Por fim, destaca-se que o ESP32 apresenta um baixo custo de aquisição, é amplamente disponível no mercado, possui uma comunidade ativa e oferece suporte a diversas tecnologias de comunicação, tais como Wi-Fi, Bluetooth, SPI, UART e I^2C. A combinação destes fatores, tornam o ESP32 uma opção atraente para aqueles que desejam desenvolver soluções eficientes, com alto desempenho e custo reduzido [72].

Este dispositivo será constituído por dois microcontroladores ESP32, designando-se cada um deles de Master e Slave MCU. Antes de se abordar cada um deles individualmente, é importante discutir sobre dois tópicos relacionados com a abordagem de se utilizar dois microcontroladores num único dispositivo.

1. Eficiência

   A eficiência de um dispositivo constituído por dois microcontroladores é um aspeto que deve ser discutido. Se por um lado, a adição de mais um microcontrolador traduz-se num maior consumo de energia para o dispositivo, por outro lado, a existência de um segundo microcontrolador pode libertar valiosos recursos de processamento e armazenamento do primeiro. Um pequeno exemplo em anexo pode demonstrá-lo. No Anexo A, encontra-se um código de referência da plataforma Microsoft Azure IoT, que realiza a leitura e o envio periódico da humidade e temperatura relativa do ar. Este *firmware* é dividido em duas partes: a parte da conectividade com a plataforma (assinalada com retângulos azuis) e a parte da solução (assinalada com retângulos amarelos), que nada mais é do que a leitura da humidade e temperatura.

   Ao utilizar-se a IDE do Arduino para carregar no ESP32 o *firmware* que contenha apenas o código da solução (excluindo-se a parte da conectividade), o espaço de armazenamento

utilizado pelo programa será de 20% da memória total disponível. Note-se que estes 20% também incluem o núcleo do Arduino para o ESP32, conforme ilustrado na Figura 41.

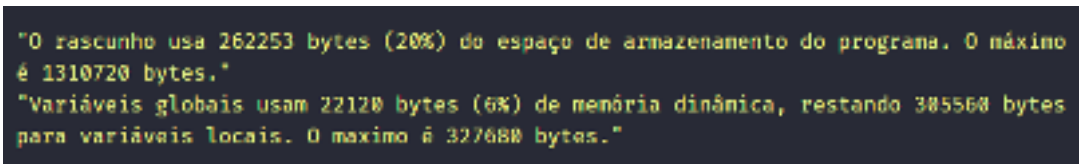


Figura 41 - Consumo de memória *flash* da solução e núcleo do Arduino para ESP32

Por sua vez, se nas mesmas condições às anteriores for realizado o carregamento do *firmware,* mas com a adição da parte da conectividade à plataforma, então já será utilizado 66% da memória total disponível (Figura 42).

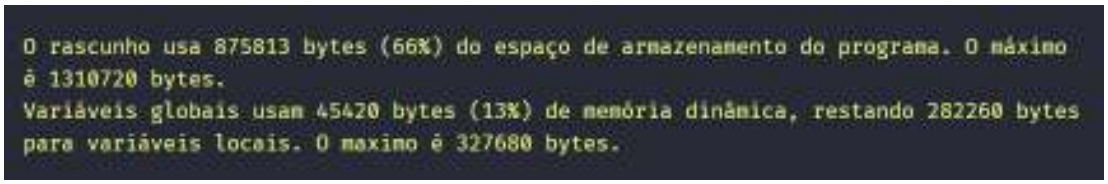


Figura 42 - Consumo de memória *flash* do *firmware* responsável pela conectividade com a plataforma, solução e núcleo do Arduino para ESP32

Neste exemplo, os resultados apresentados na Figura 41 e Figura 42, demonstram que somente a parte do código relacionada à conectividade com a plataforma é responsável por um acréscimo de 46% de utilização de memória. No caso do dispositivo que se pretende desenvolver, ao atribuir-se ao Master MCU a responsabilidade pela conectividade entre o dispositivo e a plataforma, será possível armazenar o mencionado acréscimo de 46% na sua memória *flash*, o que consequentemente libertará alguma capacidade processamento e espaço na memória *flash* do Slave MCU. Assim, o utilizador terá a possibilidade de construir soluções mais complexas e robustas, aproveitando o espaço de memória e o processamento suplementar do Slave MCU.

2. Custo de Aquisição

O custo de aquisição de um microcontrolador é um fator relevante e a ter em conta. Ao ser acrescentado ao dispositivo um segundo microcontrolador, naturalmente que o custo final do dispositivo aumentará. O custo final do dispositivo dependerá do microcontrolador que se pretenda adicionar, devendo este ser um ponto a ter em consideração na sua seleção. Contudo, um aumento de custo de aquisição deve responder a um aumento de qualidade e funcionalidades promovidas. Neste caso em específico, com a adição de outro microcontrolador, pretende-se que o mesmo trate de tarefas relacionadas com a conectividade à plataforma, que noutros casos, teriam de ser

resolvidas pelo próprio utilizador. Nessas tarefas, incluem-se: validação de formatos, gestão de *tokens*, controlo de acessos, tratamento de erros, cache, entre outras. Neste segmento, o importante é sobretudo encontrar um equilíbrio entre o aumento do custo de aquisição e as mais-valias que esse aumento acrescentará.

Nas próximas duas secções, são analisados individualmente cada um dos microcontroladores, discutindo-se especificamente as funções atribuídas a cada um deles.

3.2.2.1 Master MCU

O Master MCU terá a principal função de agir como um *gateway* entre o Slave MCU e a plataforma, devendo auxiliar o utilizador a implementar a solução pretendida. Contudo, as funções do Master não se limitam somente em transacionar os dados de comunicação entre estes dois elementos do sistema, podendo ficar encarregue de outras funções que deverão provocar um aumento de interoperabilidade, segurança, controlo e monitorização das comunicações, e em certo modo, um aumento de eficiência e acessibilidade entre o dispositivo e a plataforma.

1. Interoperabilidade

   O Master MCU poderá controlar e garantir a compatibilidade das mensagens transferidas entre a plataforma e o Slave MCU. Esta capacidade pode ser útil em situações inerentes à validação dos dados da mensagem (incluindo-se a validação de formatos e parâmetros utilizados), bem como na conversão de protocolos de comunicação.

2. Segurança

   Devido a situar-se num ponto intermédio da comunicação, o Master poderá atuar como uma firewall, garantindo uma menor exposição dos pontos mais sensíveis dos *end-points* da plataforma.

3. Controlo e monitorização das comunicações

   O Master MCU pode realizar operações de controlo e monitorização relativamente à comunicação com os *end-points* da plataforma. Nessas operações incluem-se: a gestão de tráfego entre o Slave e plataforma, salvaguardando que ambos os elementos da comunicação não gerem um volume de tráfego que comprometa o funcionamento do sistema; Controlo dos acessos aos recursos da plataforma; Monitorização do estado da comunicação e dos elementos constituintes, podendo coletar dados estatísticos,

identificar problemas na comunicação e lançar notificações aos utilizadores no caso de ocorrer alguma eventualidade inesperada.

4. Atualizações de *firmware* sem complicações

   Ao permanecer conectado à plataforma, o Master poderá receber atualizações sem fios, utilizando o método de atualização OTA (Over The Air). Estas atualizações permitirão aos utilizadores manter o *firmware* sempre atualizado consoante as necessidades da sua solução, podendo também corrigir assim eventuais erros que possam surgir ou até implementar possíveis melhorias. Este é um processo que se torna mais conveniente para o utilizador, uma vez que do ponto de vista do dispositivo, bastará apenas conectá-lo à internet para que este processo ocorra naturalmente.

5. Atualização do Slave MCU com diferentes linguagens de programação

   Em função da linguagem de programação utilizada para atualizar o *firmware* do Slave MCU, dependendo da linguagem de programação escolhida, o processo de atualização poderá sofrer variações. Por exemplo, o processo de atualização de um binário desenvolvido através da plataforma Arduino, diferirá daquele que foi implementado com MicroPython. Neste exemplo, do ponto de vista do processo de atualização, as diferenças resumem-se à quantidade de ficheiros que compõem o *firmware* e aos endereços de memória nos quais esses ficheiros serão armazenados. Cabe ao Master a responsabilidade deste processo de atualização, enquanto Slave MCU apenas precisará de carregar e executar os binários.

O *firmware* do dispositivo que será desenvolvido neste trabalho, deverá ser carregado especificamente neste microcontrolador. Numa perspetiva de produto final, prevê-se que este microcontrolador venha pré-programado de fábrica, inabilitando o utilizador final de programá-lo. Desta forma, o dispositivo poderá executar as funções para as quais foi projetado, dispensando a necessidade de qualquer programação adicional por parte do utilizador.

Por fim, é importante salientar que este *firmware* será concebido através de uma variante específica de C++, a mesma que é utilizada pela plataforma Arduino. A escolha desta linguagem e respetiva variante está diretamente relacionada com a compatibilidade e facilidade de programação. Isto deve-se à plataforma Arduino contar com uma comunidade ativa de entreajuda, que disponibiliza milhares de bibliotecas para diferentes propósitos, desde um modelo específico de um sensor até soluções mais completas e complexas.

3.2.2.2 Slave MCU

O microcontrolador Slave deve ser projetado para ser utilizado inteiramente pelo utilizador final. Em contraste com o Master MCU, o utilizador poderá carregar o seu *firmware*, adicionando sensores e outros dispositivos de instrumentação que irão complementar a sua solução desenvolvida.

Como será utilizado para projetar as soluções do utilizador, é importante fornecer múltiplos recursos que cubram as possíveis necessidades dos seus utilizadores. Por esse motivo, este microcontrolador estará equipado com múltiplas tecnologias de comunicação, como Wi-Fi, Bluetooth, UART, SPI e I^2C. A disponibilidade de múltiplas formas de comunicação no dispositivo, pode ser um fator determinante para lidar com as condicionantes que normalmente um projeto pode enfrentar. Assim, ao proporcionar o aumento da flexibilidade e adaptabilidade do dispositivo, será possível aumentar as chances de o dispositivo atender às exigências necessárias de cada projeto.

O ESP32 é um microcontrolador que já dispõem das tecnologias de comunicação anteriormente mencionadas. No entanto, há um objetivo de disponibilizar uma nova tecnologia que possibilite comunicações de longo alcance. As tecnologias LoRa e GFSK foram apresentadas como possíveis opções para atingir esse objetivo, tal como foi descrito nas secções 2.6.1 e 2.6.2. Após a análise das respetivas tecnologias, foi necessário procurar por dispositivos compatíveis com estas técnicas de modulação, levando em consideração o custo, a facilidade de uso e a sua disponibilidade no mercado. Assim, foram escolhidos os módulos SX1276/78 para se efetuar testes à tecnologia LoRa e o HC-12 para a tecnologia GFSK.

Fornecidos pela SEMTCH, o SX1276 e o SX1278 são dois módulos transcetores de rádio sem fios, que possuem o modem LoRaTM de longo alcance e operam entre a faixa de frequência de 137 - 1020 MHz e de 137 - 525 MHz, respetivamente. São comummente utilizados em dispositivos IoT que requerem comunicações de longa distância, devido ao seu consumo de energia reduzido e a sua elevada imunidade a interferências. De acordo com o fabricante, a técnica de modulação LoRa implementada permite que estes dispositivos alcancem uma sensibilidade de -148 dBm. A combinação da alta sensibilidade com o amplificador de potência integrado de +20 dBm, torna estes dispositivos ideais para qualquer aplicação que requeira alcance e robustez [54].

O HC-12 é um módulo transcetor de rádio sem fios, que opera entre a faixa de frequência de 433,4 - 473,0 MHz, com a possibilidade de ser configurado entre 100 canais de comunicação, cada um com uma largura de banda de 400 kHz. A potência máxima de transmissão do módulo é de 100 mW (equivalente a 20 dBm) e a sensibilidade do recetor é de -116 dBm, com uma taxa de transmissão de 5000 bps e com uma capacidade de distância de comunicação até cerca de 500 metros [73].

4 Desenvolvimento e Discussão de Resultados

Após a definição dos requisitos para a implementação do sistema IoT, procede-se ao desenvolvimento do sistema. Neste capítulo, são apresentados os resultados obtidos com base no estipulado no capítulo anterior.

4.1 Plataforma

Nos próximos subcapítulos é descrito o desenvolvimento das APIs e da IU, incluindo as principais dificuldades encontradas e as soluções adotadas para ultrapassá-las.

4.1.1 Interfaces de Programação de Aplicação

No decorrer do desenvolvimento deste trabalho, com o propósito de oferecer interfaces que possibilitam que outros componentes do sistema consigam se integrar e aceder a outros recursos, foram criadas cinco APIs: Gestão de Utilizadores, Gestão de Dispositivos, Transmissão de Dados, Atualização de *Firmware* e Notificações. Tratando-se de uma abordagem de arquitetura de microsserviços, cada um destes cinco serviços tem a sua própria BD.

Como já anteriormente enunciado no subcapítulo 3.2.1.1, as APIs de Atualização de *Firmware*, Notificações e Transmissão de Dados utilizam uma BD não relacional (MongoDB), enquanto as restantes utilizam individualmente uma BD relacional (PostgreSQL). Cada BD é responsável por armazenar e gerir diferentes conjuntos de dados específicos. A diversidade de dados obriga a que cada uma das BDs seja composta por diferentes atributos e, se necessário, sejam estabelecidas relações entre si. Deste modo, na Figura 43 é apresentado o diagrama de entidade-relacionamento que descreve as entidades, atributos e relacionamentos das BDs da plataforma desenvolvida. Esta figura divide as BDs constituintes da plataforma em dois grupos: MongoDB (1) e PostgreSQL (2). Nos próximos dois parágrafos, é feita uma breve análise a cada um destes dois grupos, descrevendo-se a composição e as relações de cada uma das BDs pertencentes a cada um dos grupos.

No primeiro grupo, a BD integrada na API de Atualização de *Firmware* é responsável por armazenar as atualizações feitas no Slave MCU. Esta BD é composta por apenas uma coleção denominada *"Firmware Uploads"*. A coleção *"Firmware Uploads"* possui várias propriedades essenciais para o registo das atualizações solicitadas pelo utilizador, tais como o ID do utilizador requerente, o ID do dispositivo destinatário, a linguagem do *firmware* e

todos os ficheiros constituintes da atualização. Por sua vez, a BD da API de Notificações mantém um registo completo de todas as notificações emitidas pela plataforma. Esta BD possui várias propriedades relevantes, incluindo o título e conteúdo da notificação, o ID do cliente para quem a notificação é direcionada, o tipo de notificação (classificado em quatro categorias: erro, aviso, informação e sucesso) e um indicador que determina se a notificação foi lida ou não. Por último, a BD pertencente à API de Transmissão de Dados, regista todos os dados transacionados entre os utilizadores e os dispositivos. É composta por duas coleções: *"User Data"* e *"Device Data"*. A diferença entre estas duas coleções, reside no facto de a coleção *"User Data"* registar os dados enviados dos utilizadores para os seus dispositivos, enquanto a coleção *"Device Data"* armazena o mesmo tipo de informação, mas no sentido oposto, ou seja, dos dispositivos para os utilizadores.

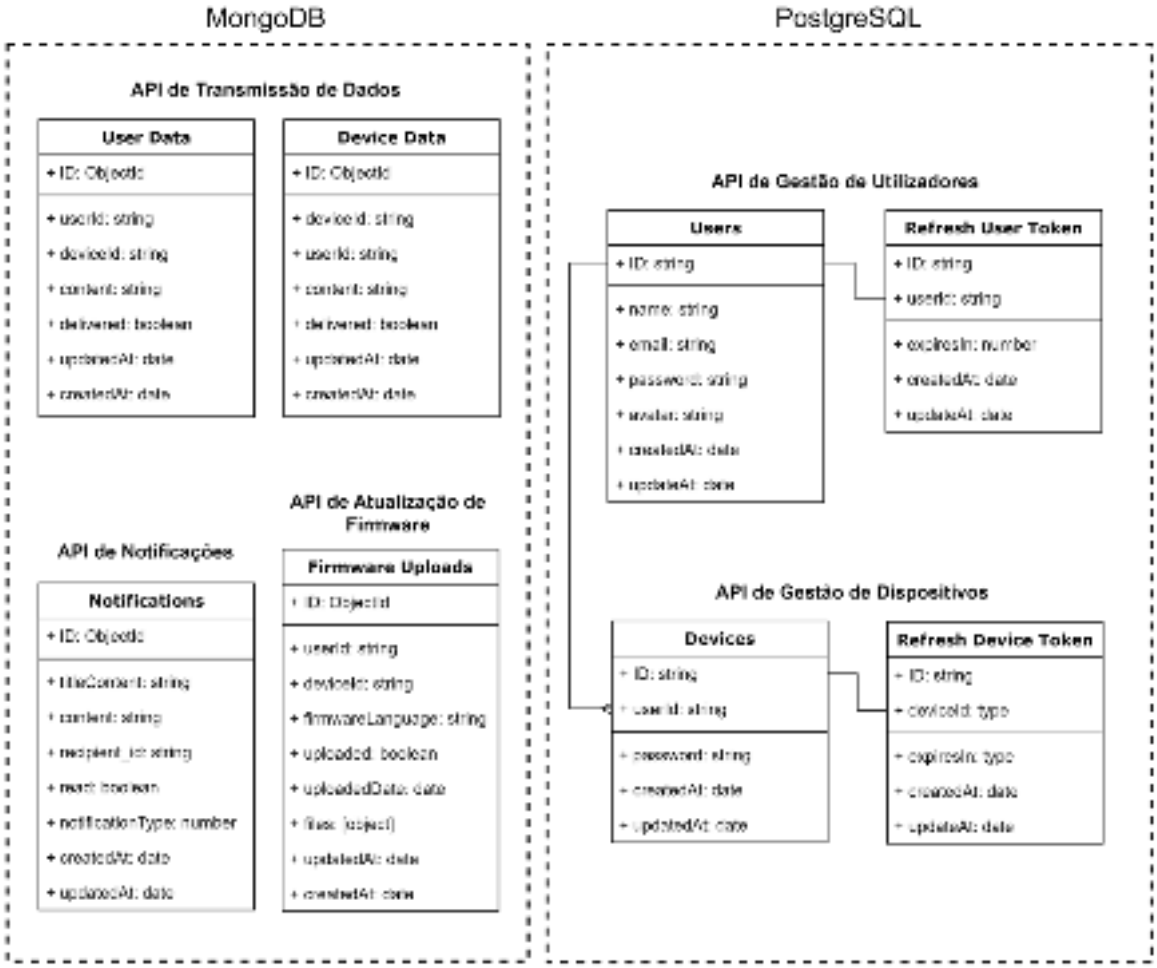


Figura 43 - Diagrama Entidade-Relacionamento

No segundo grupo, encontra-se a BD da API de Gestão de Utilizadores e Dispositivos, que se destinam a armazenar os dados referentes aos utilizadores e aos dispositivos, respetivamente. No caso da API de Gestão de Utilizadores, a BD é composta

por duas tabelas: a tabela *"Users"* armazena os dados de registo de cada utilizador (nome, endereço de email, senha, etc.), enquanto a outra tabela, *"Refresh User Token"*, armazena os *tokens* de atualização de cada utilizador. No caso da API de Gestão de Dispositivos, a BD também é constituída por duas tabelas: *"Devices"* e *"Refresh Device Token"*. Estas duas tabelas seguem o mesmo princípio utilizado nas tabelas da API de Gestão de Utilizadores, aplicando-se neste caso, ao contexto dos dispositivos.

Para identificar e validar a autenticidade dos clientes que pretendem aceder a determinados recursos ou funcionalidades, são utilizados pelas APIs desta plataforma seis JSON Web Tokens (JWT):

1. Token de Acesso de Utilizador (*User Token*): utilizado para autenticar um utilizador na plataforma e permitir o acesso a recursos específicos. O seu *payload* é constituído pelo ID do utilizador e pela validade de expiração do *token*, definida em formato *Unix Timestamp* com a validade de um dia;
2. Token de Atualização de Utilizador (*Refresh User Token*): atualiza o Token de Acesso de Utilizador expirado, permitindo que o utilizador continue autenticado sem a necessidade de iniciar a sessão novamente. Tem uma validade de trinta dias, e ao fim deste período, é gerado e enviado um novo Token de Atualização de Utilizador.
3. Token de Acesso do Dispositivo (*Device Token*): tem a função de autenticar um dispositivo na plataforma e fornecer acesso a recursos específicos;
4. Token de Atualização do Dispositivo (*Refresh User Token*): atualiza o Token de Acesso do Dispositivo expirado, permitindo que o dispositivo continue autenticado sem precisar de iniciar a sessão novamente;
5. Token de Associação de Dispositivos (*Device Association Token*): associa um dispositivo a uma determinada conta de utilizador. O seu *payload* é constituído pelo ID do utilizador e pela validade de expiração do *token*, definido em dois minutos;
6. Token de Acesso de API (*API Token*): utilizado para autenticar as solicitações provenientes das APIs da plataforma.

Nos próximos subcapítulos, são apresentados os resultados de desenvolvimento de cada API, tais como: os seus *endpoints* e fluxogramas que representem o respetivo processo de funcionamento; os protocolos de comunicação utilizados; os parâmetros de entrada e de

saída de cada protocolo de comunicação; os *tokens* utilizados e associados aos respetivos *endpoints*; entre outras possíveis informações relevantes.

4.1.1.1 Gestão de Utilizadores

Esta API oferece suporte ao protocolo de comunicação HTTP, e é destinada aos utilizadores do sistema. Mediante a aplicação móvel, os utilizadores através desta API, poderão adicionar, validar e alterar os dados associados à sua conta. Na Tabela 4, são listados os *endpoints* HTTP fornecidos por esta API, como também uma breve descrição de cada um deles:

Tabela 4 - Endpoints HTTP da API de Gestão de Utilizadores

N.º	**Endpoint**	**Método**	**Ação**
1	/profile	GET	Retorna os dados de registo do utilizador
2	/profile	PUT	Altera os dados de registo do utilizador
3	/profile/avatar	PATCH	Associa uma imagem de perfil ao registo do utilizador
4	/sessions	POST	Autentica o utilizador
5	/users	POST	Adiciona um novo utilizador
6	/users/refresh_sessiontoken	POST	Retorna um novo token de acesso de utilizador

No Apêndice A estão disponíveis os fluxogramas correspondentes de cada *endpoint* apresentado na Tabela 4, identificados pelo nome da respetiva API e número de *endpoint*. Já na Tabela 5, são apresentados os parâmetros de entrada e de saída de cada um dos *endpoints* expostos na Tabela 4.

Tabela 5 - Parâmetros de entrada e de saída da API de Gestão de Utilizadores

N.º de Endpoint	Parâmetros de Entrada	Parâmetros de Saída
1	Token de acesso de utilizador	ID do utilizador; Nome; E-mail; Data de criação de registo; Data de alteração de registo; URL da imagem de utilizador
2	Nome; E-mail; Senha atual; Nova senha; Confirmação da nova senha; Token de acesso de utilizador	ID do utilizador; Nome; E-mail; Data de criação de registo; Data de alteração de registo; URL da imagem de utilizador
3	Ficheiro de imagem; Token de acesso de utilizador	ID do utilizador; Nome; E-mail; Data de criação de registo; Data de alteração de registo; URL da imagem de utilizador
4	E-mail; Senha	ID do utilizador; Nome; E-mail; Data de criação de registo; Data de alteração de registo; URL da imagem de utilizador; Token de acesso de utilizador; Token de atualização de utilizador
5	Nome; E-mail; Senha	ID do utilizador; Nome; E-mail; Data de criação de registo; Data de alteração de registo; URL da imagem de utilizador
6	Token de atualização de utilizador	Token de acesso de utilizador

Para garantir a autenticidade dos seus clientes, esta API utiliza dois tipos de JWT:

1. Token de Acesso de Utilizador;
2. Token de Atualização de Utilizador;

O Token de Acesso de Utilizador é emitido após a autenticação do utilizador (fluxograma na Figura 95, no Apêndice A), de modo a verificar a sua autenticidade e permitir o seu acesso aos recursos protegidos da API. É obrigatório fornecer este *token* nos *endpoints* n.º 1, 2, 3, 6 da Tabela 4. Após a autenticação do utilizador ser bem-sucedida, o Token de Atualização de Utilizador é emitido juntamente com o Token de Acesso de Utilizado. A

emissão do Token de Acesso de Utilizador permite que o utilizado renove o seu Token de Acesso após o seu prazo de validade expirar (fluxograma no Apêndice A - Figura 97).

4.1.1.2 Gestão de Dispositivos

A API de Gestão de Dispositivos suporta dois protocolos de comunicação: HTTP e CoAP. O protocolo HTTP destina-se a responder a solicitações provenientes dos utilizadores ou de outras APIs da plataforma. O protocolo CoAP destina-se para um efeito semelhante ao do protocolo HTTP, mas com a particularidade de ser direcionado especificamente para dispositivos.

Na Tabela 6 são listados os *endpoints* HTTP fornecidos por esta API, como também uma breve descrição de cada um deles.

Tabela 6 - Endpoints HTTP da API de Gestão de Dispositivos

N.º	**Endpoint**	**Método**	**Ação**
1	/device/create	POST	Regista um novo dispositivo
2	/device/generate_associationtoken	POST	Retorna um novo token de associação de dispositivos
3	/device/filter/deviceid/{id}	GET	Retorna os dados registados de um determinado dispositivo
4	/device/filter/online/deviceid/{id}	GET	Informa se um determinado dispositivo está ou não online
5	/device/filter/userid/{id}	GET	Retorna os dispositivos associados a um utilizador
6	/device/link	PUT	Associa um dispositivo a uma conta de utilizador
7	/device/unlink/	PUT	Desassocia um dispositivo a uma conta de utilizador

No Apêndice A estão disponíveis os fluxogramas correspondentes a cada *endpoint* HTTP, identificados pelo nome da respetiva API e número de *endpoint*. É importante reforçar que todos os *endpoints* apresentados na Tabela 6 destinam-se apenas aos utilizadores ou a outras APIs pertencentes à plataforma.

Para os dispositivos, a Tabela 7 apresenta os *endpoints* CoAP disponíveis nesta API e uma breve descrição de cada um deles. Na Tabela 8 são apresentados os parâmetros de entrada e de saída de cada um dos *endpoints* apresentados na Tabela 7.

Tabela 7 - Endpoints CoAP da API de Gestão de Dispositivos

N.º	Endpoint	Método	Ação
8	/connection_device	POST	Valida a autenticação do dispositivo, possibilitando a associação de um dispositivo a um utilizador, caso assim seja solicitado.
9	/refresh_sessiontoken	POST	Retorna um novo token de acesso do dispositivo.

Mais uma vez, no Apêndice A estão disponíveis os fluxogramas correspondentes a cada *endpoint*, identificados pelo nome da respetiva API e número de *endpoint,* com o objetivo de clarificar o funcionamento de cada um. Na Tabela 8 são apresentados os parâmetros de entrada e de saída de cada um dos *endpoints* apresentados na tabela anterior.

É importante salientar que no *endpoint* n.º 8 da Tabela 8 os parâmetros de saída podem variar dependendo dos parâmetros de entrada inseridos, existindo três possibilidades distintas. Na primeira, é realizado o típico processo de início de sessão, onde devem ser fornecidas as respetivas credenciais: o ID do dispositivo e a senha. Em caso de sucesso, é devolvido o Token de Acesso do Dispositivo, permitindo posteriormente a atualização do estado da sessão sem necessidade de voltar a fornecer o ID do dispositivo e a senha (terceira possibilidade). Caso seja adicionado um Token de Associação de Dispositivos com as credenciais (segunda possibilidade), o dispositivo iniciará primeiramente a sessão e, em seguida, associar-se-á ao ID do utilizador anexado no *payload* do Token de Associação de Dispositivos fornecido.

Na plataforma projetada, por norma, os *endpoints* de cada API aceitam um conjunto de *tokens* que derivam apenas de um tipo de cliente (utilizadores, dispositivos, APIs e administradores do sistema). Contudo, esta API em específico, possui *endpoints* que aceitam *tokens* que derivam de diferentes tipos de clientes. Por exemplo, o *endpoint* n.º 3 desta API aceita dois tipos de *tokens*: tanto aceita solicitações que contenham um Token de Acesso de Utilizador como de API. O *endpoint* n.º 1 destina-se apenas a clientes que realizam solicitações locais. Este tipo de ponto de comunicação é dedicado exclusivamente aos administradores da plataforma, visto que apenas clientes com acesso local a esta API têm permissão para acederem aos seus serviços.

Tabela 8 - Parâmetros de entrada e de saída da API de Gestão de Dispositivos

N.º de Endpoint	Parâmetros de Entrada	Parâmetros de Saída
1	Senha; ID do utilizador	ID do dispositivo; ID do utilizador; Data de criação de registo; Data de alteração de registo
2	Token de acesso de utilizador	Token de associação de dispositivos
3	ID do dispositivo; Token de acesso de API	ID do dispositivo; ID do utilizador; Data de criação de registo; Data de alteração de registo
4	ID do dispositivo; Token de acesso de API	Estado do dispositivo (online ou offline)
5	ID do utilizador; Token de acesso de utilizador	ID do dispositivo; ID do utilizador; Data de criação de registo; Data de alteração de registo
6	ID do dispositivo; Token de acesso de utilizador	ID do dispositivo; ID do utilizador; Data de criação de registo; Data de alteração de registo
7	ID do dispositivo; Token de acesso de utilizador	ID do dispositivo; ID do utilizador; Data de criação de registo; Data de alteração de registo
8	ID do dispositivo; Senha	Mensagem informativa; Código de resposta; Endereço de URL; Token de acesso do dispositivo; Token de atualização do dispositivo
	ID do dispositivo; Senha; Token de associação de dispositivos	Mensagem informativa; Código de resposta; Endereço de URL; Token de acesso do dispositivo; Token de atualização do dispositivo
	Token de acesso do dispositivo	Mensagem informativa; Payload do token de acesso do dispositivo; Código de resposta; Endereço de URL
9	Token de atualização do dispositivo	Token de acesso do dispositivo

São utilizados cinco tipos de JWT por esta API:

1. Token de Acesso do Dispositivo;
2. Token de Atualização do Dispositivo;
3. Token de Associação de Dispositivos;

4. Token de Acesso de Utilizador;
5. Token de Acesso de API.

Ambos os Tokens de Acesso do Dispositivo e de Atualização do Dispositivo são gerados após o dispositivo efetuar a autenticação com sucesso (fluxograma no Apêndice A - Figura 105). Quando o Token de Acesso do Dispositivo expirar, o dispositivo deve enviar automaticamente uma solicitação para o *endpoint* n.º 9, utilizando o Token de Atualização do Dispositivo para obter um novo *token* de acesso (conforme detalhado no Apêndice A - Figura 106). O Token de Associação de Dispositivos é apenas necessário fornecê-lo como parâmetro de entrada no *endpoint* n.º 8. Esta API utiliza o Token de Acesso de Utilizador para aceder aos dados dos dispositivos associados ao utilizador, bem como para desassociar esses mesmos dispositivos à sua conta (fluxogramas no Apêndice A - Figura 102 e Figura 104, respetivamente). Por sua vez, o Token de Acesso de API deve ser anexado ao cabeçalho da solicitação do *endpoint* n.º 4, visto apenas possuir serviços destinados para APIs.

Durante o desenvolvimento desta API, ficou evidente a necessidade de implementar métodos que assegurassem a fiabilidade do processo de associação entre dispositivos e utilizadores. Sem a implementação de métodos de segurança adequados, o processo de associação ficaria condicionado ao surgimento de situações inesperadas, como por exemplo, um utilizador associar indevidamente um dispositivo. Para reduzir as chances de tais ocorrências, foi definido especificamente um *token*, denominado por Token de Associação de Dispositivos. Assim, sempre que um utilizador tentar adicionar um novo dispositivo à sua conta, a aplicação móvel deve enviar uma solicitação ao *endpoint* n.º 2 desta API, com o propósito de obter um Token de Associação de Dispositivos (fluxograma no Apêndice A - Figura 99). Após a API gerar e devolver o *token* solicitado ao utilizador, o mesmo deve ser encaminhado para o dispositivo. Por sua vez, o dispositivo comunicará com a API de Gestão de Dispositivos a intenção de associar-se ao utilizador[14]. Para a conclusão do processo de associação, API deve verificar se o dispositivo não está associado a nenhum utilizador e se o utilizador requerente se encontra registado. Se ambas as condições forem atendidas, a associação do dispositivo será realizada com êxito. Na Figura 44 apresenta-se um diagrama do processo descrito neste parágrafo.

[14] O ID do utilizador está incluído no *payload* do respetivo Token de Associação de Dispositivos.

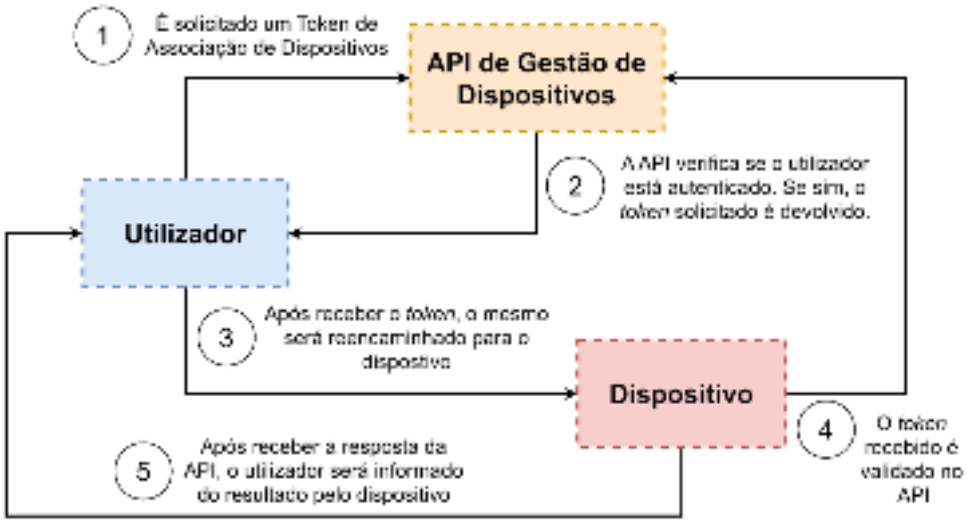


Figura 44 - Diagrama de associação de um dispositivo a um utilizador

4.1.1.3 Transmissão de Dados

A API de Transmissão de Dados suporta três protocolos de comunicação: HTTP, WebSocket e CoAP. O protocolo WebSocket destina-se aos utilizadores que necessitam de enviar dados da aplicação móvel para os seus dispositivos, enquanto o protocolo CoAP é utilizado pelos dispositivos para encaminhar os dados adquiridos para a aplicação móvel. Em relação ao protocolo HTTP, embora não esteja atualmente a ser utilizado nesta API, foi concedido o suporte para no caso ser necessário no futuro.

Na Tabela 9 é listado o único *endpoint* WebSocket fornecido por esta API, juntamente com uma breve descrição:

Tabela 9 - Endpoints WebSocket da API de Transmissão de Dados

N.º	**Endpoint**	**Ação**
1	/data_exchange_user-to-device	Recebe os dados provenientes do utilizador e encaminha-os para o respetivo dispositivo associado

Na Tabela 10 é listado o *endpoint* CoAP disponível nesta API:

Tabela 10 - Endpoints CoAP da API de Transmissão de Dados

N.º	**Endpoint**	**Método**	**Ação**
2	/data_exchange_device-to-user	PUT	Recebe os dados provenientes do dispositivo e encaminha-os para o utilizador associado

No Apêndice A estão disponíveis os fluxogramas correspondentes a cada *endpoint*, identificados pelo nome da respetiva API e número de *endpoint*. Na Tabela 11 são apresentados os parâmetros de entrada e de saída de cada um dos *endpoints*.

Tabela 11 - Parâmetros de entrada e de saída da API de Transmissão de Dados

N.º de Endpoint	**Parâmetros de Entrada**	**Parâmetros de Saída**
1	Token de acesso de utilizador; ID do dispositivo; Conteúdo da mensagem	ID da mensagem; ID do utilizador; ID do dispositivo; Conteúdo da mensagem; Data de criação de registo; Data de alteração de registo
2	Token de acesso do dispositivo; Conteúdo da mensagem	ID da mensagem; ID do utilizador; ID do dispositivo; Conteúdo da mensagem; Data de criação de registo; Data de alteração de registo

São utilizados dois JWT por esta API:

1. Token de Acesso de Utilizador;
2. Token de Acesso do Dispositivo;

O Token de Acesso do Utilizador deve ser enviado no cabeçalho da mensagem a ser recebida pelo dispositivo. Por outras palavras, sempre que o utilizador pretender enviar uma mensagem para o dispositivo, é obrigatório passar este *token*, de modo que a API possa verificar se o utilizador em questão tem ou não autorização para comunicar com o dispositivo pretendido. A referida autorização resume-se em dois pontos: o utilizador deve estar com a sessão iniciada; o utilizador deve estar associado ao dispositivo para o qual deseja comunicar. Com apenas o *token*, é possível realizar tais validações, uma vez que, se o cliente possui um Token de Acesso de Utilizador válido, isso significa que se trata de um utilizador autenticado. Adicionalmente, uma vez que este *token* contém o ID do utilizador no seu *payload*, é possível extraí-lo e comunicar com a API de Gestão de Dispositivos, para verificar se o respetivo ID está associado ao dispositivo que o utilizador pretende comunicar.

Por sua vez, o Token de Acesso do Dispositivo tem uma função semelhante ao Token de Acesso de Utilizador, aplicando-se o mesmo princípio de funcionamento, mas com uma diferença: em vez de ser direcionado aos utilizadores, destina-se aos dispositivos.

4.1.1.4 Atualização de Firmware

A API de Atualização de Firmware tem como responsabilidade realizar atualizações sem fios nos dispositivos, mais especificamente ao *firmware* do Slave MCU. Em termos de suporte a protocolos de comunicação, apenas o protocolo HTTP é suportado. Dado o facto de que esta API tem uma função particularmente específica, o protocolo HTTP é suficiente para atender ao seu propósito, visto que todas as atualizações ocorrem por meio deste protocolo. Na Tabela 12 é listado o único *endpoint* HTTP fornecido por esta API, como também uma breve descrição:

Tabela 12 - Endpoints HTTP da API de Atualização de Firmware

N.º	**Endpoint**	**Método**	**Ação**
1	/upload/device	POST	Atualiza o *firmware* do dispositivo

Este único *endpoint* permite a atualização do *firmware* do Slave MCU através de binários desenvolvidos em diferentes linguagens de programação, tais como: uma variante de C++ (Arduino), C (ESP-IDF), Javascript (Espruino) e Python (MicroPython). Para isso, os utilizadores só precisam de selecionar na aplicação móvel os binários correspondentes com a linguagem que pretendem utilizar. A Tabela 13 especifica quais os ficheiros que o utilizador deve fornecer obrigatoriamente, conforme a linguagem selecionada.

Tabela 13 - Ficheiros necessários em função da linguagem de programação selecionada

Linguagem	**Ficheiros que devem ser fornecidos**
C++ (Arduino)	bootloader.bin; partitions.bin; firmware.bin; data.txt
C (ESP-IDF)	bootloader.bin; partitions.bin; firmware.bin; data.txt
Javascript (Espruino)	bootloader.bin; partitions.bin; firmware.bin; data.txt
Python (MicroPython)	bootloader.bin; data.txt

Todos os ficheiros apresentados na Tabela 13 são guardados na BD. Contudo, só os ficheiros com a extensão .bin serão enviados e carregados nos dispositivos. Como ilustrado na tabela, o número de ficheiros a enviar varia consoante a linguagem de programação utilizada na conceção do *firmware*. Cada um dos ficheiros apresentados na Tabela 13 tem a seguinte função:

1. bootloader.bin: é um programa que é executado pelo microcontrolador assim que o dispositivo é ligado ou reiniciado. A sua principal função é carregar o *firmware* do dispositivo a partir da sua memória flash, executando-o posteriormente.
2. partitions.bin: é utilizado para definir as partições de memória num microcontrolador. Uma partição é uma área da memória reservada para um determinado fim, como o armazenamento de dados, configurações ou código executável. Neste ficheiro são definidos o tamanho e a localização de cada partição na memória do dispositivo, permitindo que o *firmware* aceda às áreas definidas de forma eficiente.
3. firmware.bin: é o programa principal que é executado pelo microcontrolador após ser carregado pelo *bootloader*, contendo o código executável responsável por controlar o funcionamento do dispositivo.
4. data.txt: trata-se de um ficheiro que contém informações adicionais e relevantes, que serão utilizadas pela API para processar os ficheiros de *firmware* enviados pelo utilizador. É neste ficheiro que se encontra o ID do dispositivo que se pretende atualizar, bem como a linguagem de programação utilizada na conceção dos binários.

Na Figura 45 é apresentado um exemplo para melhor ilustrar o processo de atualização de um dispositivo, visto sob a perspetiva da API. Neste exemplo, simula-se a atualização de *firmware* do dispositivo, nas condições em que o mesmo é concebido na linguagem C++ e compilado através das ferramentas (SDKs) fornecidas pela plataforma Arduino. Após serem gerados os três binários (bootloader.bin, partitions.bin e firmware.bin), mais o ficheiro instrutivo data.txt, estes serão enviados para a API de Atualização de Firmware (mais precisamente para o *endpoint* n.º 1, da Tabela 13). A título de exemplo, o ficheiro data.txt terá o seguinte conteúdo em formato JSON:

```
{
	“deviceId”: “c20df4b9-20eb-4390-a252-4d0e2ac15db8”,
	“firmwareLanguage”: “arduino”
}
```

Após todos os ficheiros serem enviados da aplicação móvel, a API valida as informações recebidas. Se tudo estiver em conformidade, a API deve fundir os três binários de atualização, gerando um único ficheiro (sppifs.bin). De seguida, todos os ficheiros enviados pelo utilizador, juntamente com o ficheiro gerado pela API (sppifs.bin), serão

armazenados na BD. Se este processo ocorrer sem qualquer falhas, no final, será enviado para o dispositivo apenas um único ficheiro, o spiffs.bin (fluxograma do único *endpoint* desta API no Apêndice A - Figura 109).

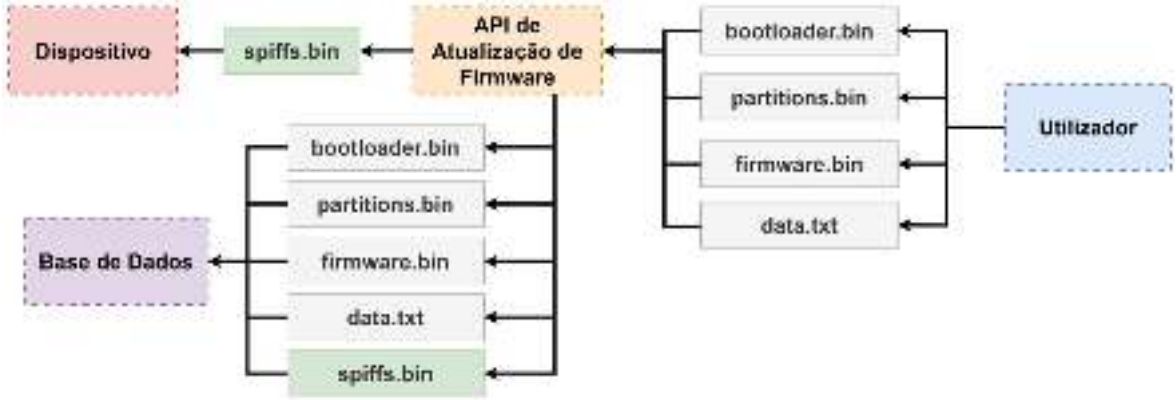


Figura 45 - Exemplo ilustrativo do processo de atualização do dispositivo

Para uma melhor compreensão, é importante realçar que a conclusão do processo de atualização do dispositivo não ocorre logo após o envio do ficheiro spiffs.bin. Na realidade, depois deste ficheiro ser enviado, a API aguardará por uma resposta do dispositivo para validar se a atualização do *firmware* foi bem-sucedida. No entanto, para que o processo de atualização seja concluído com êxito, existem alguns procedimentos que são da responsabilidade do próprio dispositivo. Esses procedimentos são abordados na secção 4.2.4, uma vez que estão diretamente relacionados com o dispositivo.

Cabe destacar que, independentemente da linguagem de programação utilizada pelo utilizador, o processo de atualização é bastante similar. A principal diferença está no número de ficheiros que devem ser fornecidos pelo utilizador, conforme é demonstrado na Tabela 13. Assim como nos subcapítulos anteriores, na Tabela 14 são apresentados os parâmetros de entrada e de saída do único *endpoint* desta API.

Tabela 14 - Parâmetros de entrada e de saída da API de Atualização de Firmware

Nº. de Endpoint	**Parâmetros de Entrada**	**Parâmetros de Saída**
1	ID do dispositivo; Linguagem do *firmware*; Token de acesso do utilizador; Ficheiros de atualização de *firmware*	ID da operação; ID do utilizador; ID do dispositivo; Ficheiros de *firmware*; Carregado; Data de carregamento; Data de criação de registo; Data de alteração de registo

É utilizado apenas um JWT por esta API:

1. Token de Acesso de Utilizador;

O Token de Acesso de Utilizador nesta API desempenha uma função semelhante à descrita anteriormente em outras APIs, tendo a finalidade de validar a autenticidade dos pedidos solicitados. No caso em particular do *endpoint* n.º 1 da Tabela 14, depois de receber a solicitação de atualização de *firmware*, a API utilizará o Token de Acesso de Utilizador, não apenas para autenticar o pedido, mas também para verificar se o dispositivo que se pretende atualizar está associado à conta do utilizador.

4.1.1.5 Notificações

A API de Notificações suporta dois protocolos de comunicação: HTTP e WebSocket. O protocolo HTTP é utilizado para receber notificações emitidas por outras APIs do sistema. Por sua vez, o protocolo WebSocket é utilizado para reencaminhar as notificações recebidas para os utilizadores correspondentes. A Tabela 15 lista os *endpoints* HTTP fornecidos por esta API, com uma breve descrição de cada um deles. Na Tabela 16 são listados os *endpoints* WebSocket fornecidos pela API.

Tabela 15 - Endpoints HTTP da API de Notificações

N.º	**Endpoint**	**Método**	**Ação**
1	/create	POST	Cria uma notificação
2	/filter?{start}&{end}	GET	Retorna todas as notificações criadas
3	/filter/id/{id}	GET	Retorna uma notificação filtrada pelo ID
4	/filter/lasthours/{hour}	GET	Retorna um conjunto de notificações filtradas por um determinado intervalo de tempo
5	/update/read_field	POST	Atualiza na BD o campo *read* (“lido”) de uma notificação

Tabela 16 - Endpoints WebSocket da API de Notificações

N.º	**Endpoint**	**Ação**
6	/connection	Efetua a conexão WebSocket, retornando um conjunto de notificações filtradas num especificado intervalo de tempo e utilizador
7	/update/read_field	Atualiza o campo *read* (“lido”) de uma notificação

No Apêndice A estão disponíveis os fluxogramas correspondentes de cada *endpoint*, identificados pelo nome da respetiva API e número de *endpoint*. Na Tabela 17 são apresentados os parâmetros de entrada e de saída de todos os *endpoints* presente nas tabelas 15 e 16.

Tabela 17 - Parâmetros de entrada e de saída da API de Notificações

N.º de Endpoint	**Parâmetros de Entrada**	**Parâmetros de Saída**
1	Título; Descrição; ID do destinatário; Tipo de notificação; Token de acesso de API	ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo
2	Data inicial; Data final	[ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo]
3	ID do utilizador	[ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo]
4	Data inicial; Data final; Token de acesso de utilizador	[ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo]
5	ID da notificação; Registo de leitura; Token de acesso de API	ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo
6	Data inicial; Data final; Token de acesso de utilizador	[ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo]
7	ID da notificação; Registo de leitura	ID da notificação; Título; Descrição; ID do destinatário; Tipo de notificação; Registo de leitura; Data de criação de registo; Data de alteração de registo

Com exceção do *endpoint* n.º 1, todos os outros *endpoints* HTTP apenas retornam uma resposta válida para solicitações locais. No caso do *endpoint* n.º 1, além de aceitar solicitações provenientes do *localhost*, também aceita solicitações que incluam um Token de Acesso de API válido. Assinala-se ainda, que os parâmetros de saída entre parênteses retos listados na tabela anterior, indicam que o *endpoint* correspondente retorna um vetor contendo os parâmetros mencionados entre os respetivos parênteses retos.

Por fim, são utilizados dois JWT por esta API:

1. Token de Acesso de Utilizador;
2. Token de Acesso de API.

Ambos os *tokens* identificam e validam os utilizadores e as APIs, respetivamente.

4.1.2 Apresentação da Interface de Utilizador

A IU é o componente que interliga o utilizador aos restantes elementos do sistema IoT. É através desta interface que os utilizadores interagem com os seus dispositivos e têm acesso aos dados adquiridos por estes. Após a definição dos requisitos que a interface deve cumprir no subcapítulo 3.2.1.2, nos capítulos subsequentes são apresentados os resultados obtidos com o desenvolvimento da IU.

4.1.2.1 Início de Sessão

Na interface desenvolvida, a página de Início de Sessão da aplicação móvel (Figura 46) é a primeira página que é apresentada aos utilizadores quando abrem a primeira vez a aplicação.

Figura 46 - Página de Início de Sessão

Esta página permite aos seus utilizadores acederem à sua conta, fornecendo para esse efeito as credenciais de acesso, que consistem apenas num endereço de e-mail e uma senha. Após serem fornecidas essas informações, os utilizadores devem clicar no botão "Iniciar sessão" para validar os dados de autenticação fornecidos. No caso de a autenticação ser bem-sucedida, a aplicação redirecionará o utilizador para a página de Painel de Controlo (Figura 48, no subcapítulo 4.1.2.3). No caso contrário, será acionada uma

janela pop-up, informando o utilizador que os dados fornecidos são inválidos, mantendo-o na página de Início de Sessão.

Antes de encaminhar os dados para a API correspondente (API de Gestão de Utilizadores), é realizada a validação dos dados inseridos pelo utilizador nas duas caixas de texto (E-mail e Senha). Logo, antes dos dados serem enviados, é verificado se o endereço de e-mail está no formato correto e se a senha tem entre 8 e 32 caracteres. Se esses requisitos não forem cumpridos, o utilizador será informado através de uma mensagem exibida na própria página. Caso contrário, os dados inseridos serão enviados para a API, ficando assim o utilizador a aguardar por uma resposta. No caso de as credenciais inseridas não corresponderem a nenhum utilizador, a API retornará uma mensagem de erro que será apresentada na aplicação móvel, devendo o utilizador repetir o processo.

Por fim, existe ainda um botão "Criar uma conta" que, quando clicado, redirecionará o utilizador para a página de Registo de Conta.

4.1.2.2 Registo de Conta

A página de registo de conta (Figura 47) tem a função de permitir que os utilizadores criem uma conta, devendo ser fornecidas algumas informações básicas nas três caixas de texto disponíveis: nome, endereço de e-mail e senha.

Tal como acontece na página de Início de Sessão, estes três campos são obrigatórios e a validação dos dados é realizada no momento da submissão do formulário. O endereço de e-mail deve ter o formato "nome@dominio .com" e a senha deve ter entre 8 e 32 caracteres. Caso os requisitos não sejam cumpridos, o utilizador será informado através de uma mensagem de erro exibida na própria página, devendo corrigir os erros antes de prosseguir novamente com o registo. Se o registo for efetuado com sucesso, o utilizador será redirecionado para a página de Início de Sessão, onde poderá efetuar a autenticação com as credenciais fornecidas durante o registo. Se a autenticação for bem-sucedida, o utilizador será redirecionado para o Painel de Controlo.

Figura 47 - Página de Registo de Conta

4.1.2.3 Painel de Controlo

A página de painel de controlo (Figura 48) é uma das principais páginas da aplicação móvel, pois permite que o utilizador aceda rapidamente às outras páginas presentes na aplicação. Nesta página é apresentada a imagem de perfil do utilizador, uma mensagem de boas-vindas que inclui o nome do utilizador, e três botões: Perfil, Dispositivos e Terminar Sessão.

O botão "Terminar Sessão" permite que o utilizador encerre a sua sessão atual, apagando todos os dados da sessão do utilizador armazenados no telemóvel (nome do utilizador, e-mail e Token de Acesso de Utilizador), impedindo assim acessos não autorizados. Os outros dois botões são discutidos nas secções 4.1.2.4 e 4.1.2.5.

Figura 48 - Página de Painel de Controlo

Abaixo, encontra-se uma barra de separadores (*tab bar*), que possibilita a navegação entre outras páginas da aplicação, mais especificamente:

3. HOME - redireciona para a página do Painel de Controlo;
4. DATA - redireciona para a página de Monitor de Dados;
5. LOAD - redireciona para a página de Carregar Firmware;
6. NOTIF - redireciona para a página de Notificações.

De salientar que a barra de separadores é único elemento comum em todos as páginas exibidas após a autenticação do utilizador.

4.1.2.4 Perfil

A página de perfil (Figura 49) possibilita a visualização e modificação das informações da conta do utilizador. Os dados da conta apresentados nesta página são a imagem de perfil do utilizador, o nome e o endereço de e-mail. Além de apresentar essas informações, é possível alterá-las (incluindo-se aqui a senha), garantindo assim que as informações da conta permaneçam sempre atualizadas.

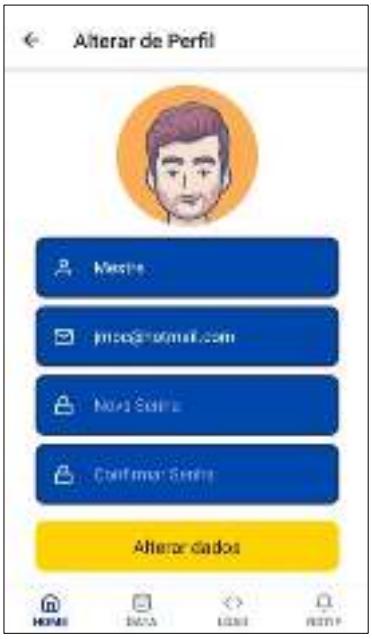


Figura 49 - Página de Perfil

Para alterar o nome, o endereço de e-mail ou a senha, basta preencher as quatro caixas de texto apresentadas na Figura 49, e posteriormente clicar no botão "Alterar dados". Antes de os dados serem submetidos, a aplicação irá validá-los. Se o utilizador pretender alterar o nome e/ou e-mail, os dados inseridos nas caixas de texto correspondentes, terão de conter pelo menos um nome de 3 caracteres e um e-mail de formato válido. No caso de se alterar a senha, é necessário inserir uma senha entre 8 e 32 caracteres, que contenha pelo menos uma letra minúscula, uma letra maiúscula e um número. No campo seguinte, é solicitada a confirmação da senha inserida anteriormente, com o propósito de evitar erros de digitação. Após clicar no botão "Alterar dados", se os dados inseridos corresponderem aos critérios descritos anteriormente, será então acionada uma janela de confirmação da ação solicitada, devendo o utilizador confirmar se pretende ou não avançar com as alterações.

Figura 50 - Janela apresentada depois de clicar na imagem de perfil

Por fim, para alterar a imagem de perfil, deve-se inicialmente clicar na imagem de perfil apresentada na Figura 49. Após este procedimento, abrirá uma janela (Figura 50) que terá três opções: carregar foto, tirar foto e voltar. Se o utilizador escolher a opção "Carregar foto", será redirecionado para o gestor de ficheiros do telemóvel, onde poderá escolher uma imagem previamente guardada. Por outro lado, se for escolhida a opção "Tirar foto", a aplicação da câmara do telemóvel será iniciada, podendo o utilizador tirar uma fotografia e defini-la como a sua nova imagem de perfil. De salientar que este processo exige que sejam concebidas as devidas permissões de armazenamento e de acesso à câmara do telemóvel. Por fim, se for selecionada a opção "Voltar", a janela apresentada na Figura 50 é fechada.

4.1.2.5 Dispositivos

A página de Dispositivos na aplicação móvel desenvolvida (Figura 51), permite que os utilizadores adicionem e gerem os seus dispositivos IoT. Nesta página, os dispositivos associados à conta do utilizador são apresentados em formato de lista, contendo cada elemento da lista o ID do dispositivo e uma opção de remover os dispositivos indesejados (ícone de caixote de lixo). Para evitar remoções acidentais, o utilizador tem de confirmar a ação de remoção.

No canto inferior direito da página, existe um botão que permite adicionar novos dispositivos à conta do utilizador, o qual será discutido na próxima secção.

Figura 51 - Página de Dispositivos

4.1.2.6 Adicionar Dispositivos

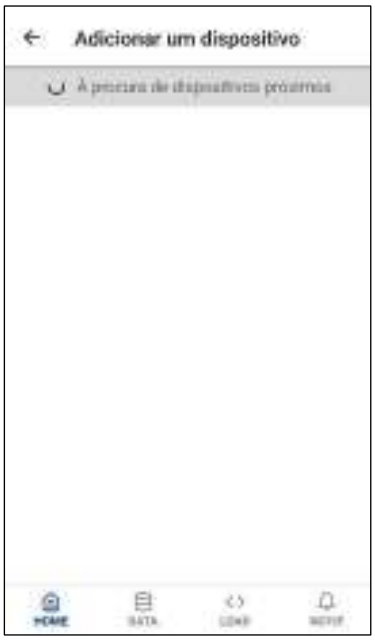


Figura 52 - Página inicial de Adicionar dispositivos

Para facilitar a associação entre dispositivos e utilizadores, foi incluída uma funcionalidade na aplicação que permite adicionar os dispositivos através da tecnologia Wi-Fi. Ao clicar no botão de adicionar dispositivos (localizado no canto inferior esquerdo da Figura 51), o utilizador será redirecionado para a página inicial de adicionar dispositivos (Figura 52). Nesta página, a aplicação móvel irá iniciar a procura por dispositivos próximos ao telemóvel. A procura por dispositivos consiste em encontrar redes Wi-Fi que possuam o prefixo "Master_" no SSID, apresentando-os nesta página conforme vão sendo encontrados (exemplo na Figura 53). Quando for encontrado, o utilizador deve então pressionar sobre o dispositivo que deseja adicionar.

Figura 53 - Exemplo de um dispositivo encontrado

Após escolher o dispositivo, o utilizador será encaminhado para uma outra página (Figura 54), onde deverá inserir o nome (SSID) e a senha da rede Wi-Fi à qual o dispositivo se deva conectar. De destacar que o campo "SSID", é uma lista suspensa (*dropdown*) que apresenta todas as redes Wi-Fi disponíveis e acessíveis pelo telemóvel, exceto aquelas criadas pelos dispositivos (com o prefixo "Master_"). Uma vez esses dados inseridos, o utilizador deve clicar no botão "Seguinte" e aguardar que os dados sejam comunicados ao dispositivo. Durante este processo, a aplicação irá conectar o telemóvel à rede gerada pelo dispositivo, enviando posteriormente o Token de Associação de Dispositivos, o nome da rede (SSID) e a senha inseridos, ficando assim, o utilizador a aguardar por uma resposta do dispositivo (Figura 55).

Figura 54 - Página de autenticação da rede Wi-Fi

Ao receber estas informações, o dispositivo irá tentar ligar-se à rede Wi-Fi com o mesmo SSID indicado pelo utilizador, utilizando a senha apenas se a mesma for fornecida (a senha do SSID correspondente é considerada uma informação facultativa no processo de associação).

Figura 55 - Dispositivo a conectar-se à rede Wi-Fi

Figura 56 - Dispositivo associado com sucesso à conta do utilizador

Figura 57 - Exemplo de uma mensagem de erro na associação do dispositivo ao utilizador

Após este procedimento, o utilizador deverá aguardar pelo resultado, recebendo em tempo real informações relativas à fase em que o dispositivo se encontra no processo de associação à conta do utilizador (Figura 55). Se o dispositivo se conectar à rede Wi-Fi desejada, deverá então comunicar com a API de Gestão de Dispositivos, iniciando a sessão com as credenciais de fábrica armazenadas na sua memória não volátil. A sessão ao ser iniciada com sucesso, o dispositivo associar-se-á ao respetivo utilizador, fornecendo para tal o Token de Associação de Dispositivos à API de Gestão de Dispositivos. Se nenhum erro ocorrer, o dispositivo enviará uma mensagem para a aplicação móvel, informando que foi associado com sucesso (Figura 56). Caso o processo de associação falhe, será enviada uma mensagem de erro, descrevendo detalhadamente a origem do erro (Figura 57). Possíveis erros podem estar relacionados com a desserialização da mensagem enviada pela aplicação móvel, parâmetros inválidos ou em falta, falha na comunicação com a API, erros de memória, entre outros. Sempre que este processo falhar, o utilizador deve restaurar as configurações do dispositivo e repetir o processo.

4.1.2.7 Monitor de Dados

A página “Monitor de Dados” (Figura 58) tem a função de possibilitar a troca de dados sem fios entre o utilizador e os seus dispositivos (mais especificamente, o Slave MCU).

Figura 58 - Página de monitor de dados

Ao abrir a página, por defeito, todos os seus componentes encontram-se desabilitados, com exceção da caixa de seleção (*dropbox*) localizada no topo da página (Figura 58). Nesta caixa de seleção, estão listados todos os dispositivos associados ao utilizador, devendo-se escolher o dispositivo com o qual se deseja comunicar. Ao selecionar o dispositivo, é estabelecida uma conexão WebSocket com a API de Transmissão de Dados, que por sua vez, terá a responsabilidade de interligar o utilizador com o dispositivo selecionado. Relembrando que esta conexão só será bem-sucedida se o cliente fornecer um Token de Acesso de Utilizador válido. Logo, no *payload* desta conexão com a API, a aplicação móvel deve incluir um *token* válido.

Após a conexão entre utilizador e dispositivo ser estabelecida com sucesso, os restantes componentes da página (as duas caixas de texto e o botão) ficam habilitados. A primeira caixa de texto da Figura 58, tem a responsabilidade de apresentar os dados provenientes dos dispositivos. O utilizador não tem autorização para inserir dados neste componente. No final desta página, encontra-se a outra caixa de texto que, ao contrário da primeira, permite ao utilizador inserir o seu próprio texto. Todas as informações inseridas nesta caixa serão enviadas para o dispositivo selecionado, assim que o utilizador clicar no botão “Enviar”.

4.1.2.8 Carregar Firmware

Na terceira aba da barra de navegação, encontra-se a página “Carregar Firmware” (Figura 59). Esta página tem a função de carregar novos *firmwares* nos dispositivos, suportando quatro ferramentas de desenvolvimento/*frameworks*: Arduino (variante de C++), ESP-IDF (C), Espruino (JavaScript) e MicroPython (Python).

A escolha e utilização destas ferramentas de tem como objetivo principal testar o conceito de multilinguagem de programação, permitindo a atualização sem fios do *firmware* (via OTA). Para esse efeito, a aplicação móvel armazenará os ficheiros *firmware* pré-compilados (Tabela 13, no subcapítulo 4.1.1.4), correspondentes a cada uma das ferramentas desenvolvimento. O utilizador precisa apenas escolher o dispositivo alvo, selecionar a linguagem de *firmware* desejada e confirmar a escolha. Ao realizar esta ação, todos os elementos da página ficam desativados e é exibida uma animação de *loading* até que a atualização seja concluída.

Figura 59 - Página de Carregar Firmware

Enquanto o utilizador aguarda pela conclusão do processo de atualização do dispositivo, a aplicação móvel envia os ficheiros de *firmware*, juntamente com um ficheiro denominado "data.txt". Este é um ficheiro essencial para prosseguir com o processo de atualização, contendo informações em formato JSON, tais como o ID do dispositivo e a linguagem do *firmware* selecionada. Ainda no mesmo pedido HTTP, são incluídos no corpo da solicitação os binários necessários de acordo com a escolha do *firmware*, juntamente com o Token de Acesso de Utilizador no cabeçalho. Devido à necessidade de enviar dados binários, como arquivos, juntamente com outras informações no corpo desta solicitação HTTP, é necessário utilizar o formato de solicitação *multipart/form-data*.

4.1.2.9 Notificações

Na última aba da barra de navegação, encontra-se a página de Notificações (Figura 60). Esta é uma funcionalidade essencial da aplicação, pois permite manter os utilizadores informados sobre o estado das soluções IoT projetadas.

Sempre que o utilizador inicia a sessão, a aplicação estabelece uma conexão via WebSocket com a API de Notificações, permitindo que o utilizador seja informado em tempo real sobre eventos que ocorram nos elementos pertencentes ao sistema, tais como:

1. Atualizações de *firmware* realizadas nos dispositivos, sejam elas bem-sucedidas ou não;
2. Alertas de ocorrências emitidos pela plataforma;

Desta forma, ao manter-se sempre informado sobre o estado dos seus dispositivos, o utilizador pode tomar medidas adequadas de modo a garantir que estes estejam sempre a funcionar corretamente.

Cada notificação apresentada na IU é acompanhada pelo tempo decorrido desde a sua emissão. É também incluído uma animação nas notificações não visualizadas, localizado no canto superior esquerdo, com o intuito de facilitar a distinção entre as notificações visualizadas e não visualizadas pelo utilizador. Por fim, salienta-se ainda a inclusão de um sinal sonoro e visual – junto ao ícone correspondente da página de notificações, localizado na barra de navegação – que alerta e indica o número de notificações não lidas pelo utilizador, respetivamente.

Figura 60 - Página de Notificações

4.2 Dispositivo

No subcapítulo 1.2 foram identificados os seguintes objetivos diretamente relacionados com o dispositivo:

1. Disponibilizar múltiplas tecnologias de comunicação, tais como Wi-Fi, Bluetooth e pelo menos uma tecnologia de longo alcance.
2. Permitir ao utilizador estabelecer uma comunicação sem fios e bidirecional com o dispositivo, em qualquer momento e lugar com ligação à internet.
3. Possibilitar a programação do dispositivo através de diferentes linguagens de programação.

De modo a alcançar os três objetivos mencionados anteriormente, nos próximos subcapítulos são apresentados os desenvolvimentos e os resultados obtidos em relação ao dispositivo.

4.2.1 Múltiplas Tecnologias de Comunicação

As tecnologias de comunicação Wi-Fi e Bluetooth estão incorporadas no lote de tecnologias fornecidas pelos microcontroladores ESP32. Como o dispositivo que se pretende projetar é composto por dois microcontroladores ESP32 (Master e Slave MCU), significa que ambos já possuem estas duas tecnologias incorporadas. Contudo, os objetivos mencionados anteriormente especificam a disponibilização de uma tecnologia de longo alcance destinada ao utilizador. Partindo deste pressuposto, entende-se que a nova tecnologia de comunicação deve ser implementada apenas no Slave MCU, uma vez que, dos dois microcontroladores disponíveis no dispositivo, este é justamente o microcontrolador que se destina a executar a solução desenvolvida pelo utilizador. Dito isto, é necessário analisar, estudar e testar as tecnologias de comunicação de longo alcance, de modo a concluir qual a tecnologia que melhor se enquadra para este propósito.

Nos subcapítulos 2.6.1 e 2.6.2, analisou-se duas tecnologias (LoRa e GFSK) que possibilitam estabelecer comunicações fiáveis a longas distâncias. Mais adiante, no subcapítulo 3.2.2.2, três módulos compatíveis com tais tecnologias (SX1276/78 e HC-12) foram apresentados.

Existindo a necessidade de determinar a tecnologia mais viável a implementar-se na placa de aquisição de dados em desenvolvimento, foram conduzidos testes de alcance e fiabilidade com o objetivo de efetuar um estudo comparativo entre as tecnologias LoRa e GFSK. Na sequência destes testes, as próximas secções descrevem a metodologia utilizada e os resultados obtidos.

4.2.1.1 Alcance

O teste de alcance teve como objetivo avaliar a amplitude de comunicação de cada uma das duas tecnologias estudadas, utilizando-se para esse efeito três módulos distintos: SX1276, SX1278 e HC-12. O método utilizado na realização deste teste consistiu na emissão de tramas a cada cinco segundos, durante cinco minutos, a partir de um planalto identificado com a letra E na Figura 61 (ponto de emissão). Para verificar a receção das tramas enviadas,

foram estipulados cinco diferentes locais, identificados como 1, 2, 3, 4 e 5 na Figura 61, que assinalam a localização de cada recetor correspondente durante o teste de alcance.

Durante os cinco ensaios realizados, a principal diferença considerada entre cada um dos pontos estipulados foi o aumento gradual da distância entre o emissor (E) e o recetor (1, 2, 3, 4, 5). É importante destacar a ausência de obstáculos físicos que pudessem prejudicar a comunicação entre os dispositivos, garantindo-se uma linha de vista do local do emissor para o recetor entre todos os pontos estipulados.

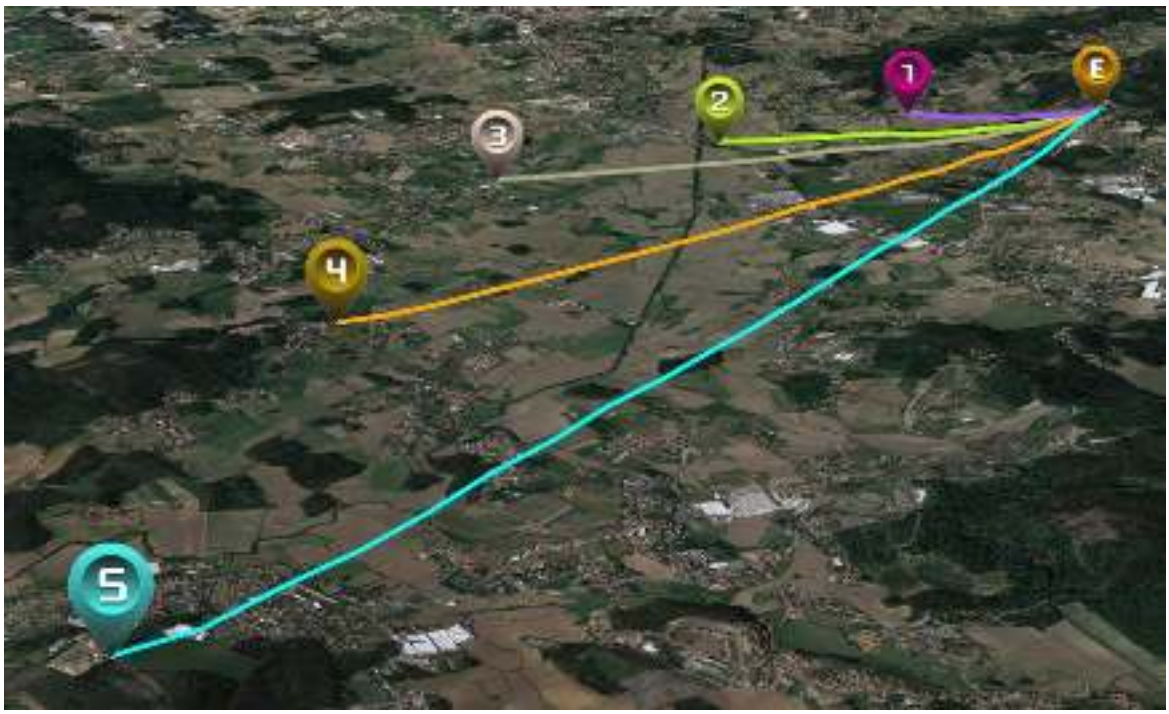


Figura 61 - Mapeamento dos pontos de comunicação do teste de alcance

Na Tabela 18 são apresentadas as distâncias aproximadas entre o emissor e o recetor de cada um dos cinco ensaios:

Tabela 18 - Distância entre módulos em cada ensaio

Ensaio (Emissor-Recetor)	**Distancia aprox. ente recetor e emissor (m)**	**Diferença entre o ponto anterior (m)**
1 (E-1)	850	---
2 (E-2)	1740	+ 890
3 (E-3)	2740	+ 1000
4 (E-4)	4200	+ 1460
5 (E-5)	6200	+ 2000

Durante este teste, os dispositivos LoRa SX1276/78 assumiram as seguintes configurações padrão: *Bandwidth* de 125 kHz e *Spreading Factor* 7. As tramas enviadas a cada cinco segundos pelo emissor, continham uma mensagem de apenas seis bytes, juntamente com um contador. Ao incluir-se um contador na mensagem, é possível gerar um identificador para cada mensagem enviada, permitindo verificar quais tramas foram entregues ao destinatário no final de cada ensaio. Nos próximos parágrafos, são apresentados de forma resumida os resultados obtidos em cada um dos cinco locais de teste.

Na localização mais próxima ao emissor, durante o primeiro ensaio (Figura 62), os recetores dos módulos SX1276 e SX1278, rececionaram as tramas provenientes do emissor sem apresentarem dificuldades. Com o SX1276 obteve-se um RSSI médio de -111 dBm, enquanto no SX1278 registou-se um RSSI médio de -98 dBm, apresentando menor perda de sinal em comparação com o SX1276.

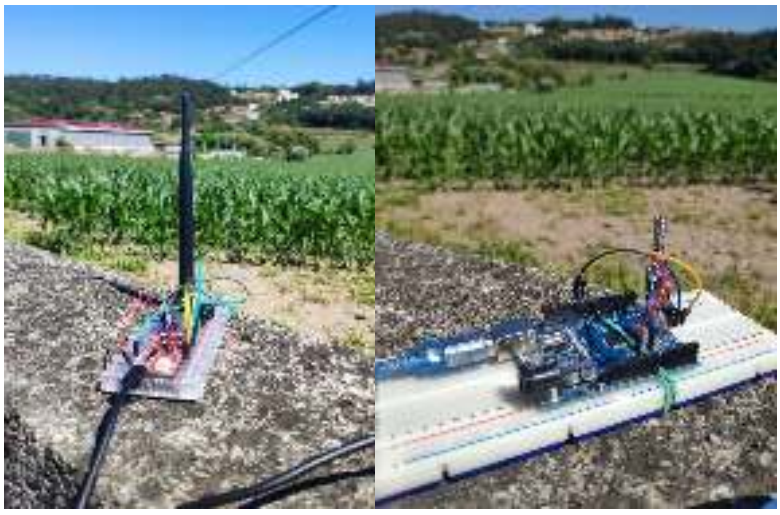


Figura 62 - Ensaio 1: recetores de rádio SX1276 (figura do lado esquerdo) e HC-12 (figura do lado direito)

Neste primeiro ensaio, o HC-12 apresentou dificuldades em receber as tramas do emissor, recebendo apenas 10% das tramas emitidas.

No segundo ensaio (Figura 63), a uma distância de 1,74 km entre emissor e recetor, com os módulos SX1276 conseguiu-se novamente comunicar sem comprometimentos, apresentando-se no módulo recetor uma ligeira quantidade de tramas recebidas com ruido. O RSSI médio foi de -111 dBm, o mesmo resultado obtido no primeiro ensaio. Ainda neste ensaio, o módulo SX1278 também recebeu as tramas do emissor com relativa facilidade, obtendo-se um RSSI médio de -103 dBm. Já o HC-12 não conseguiu capturar nenhuma trama emitida pelo emissor. Devido a este resultado, este módulo foi descartado para os próximos testes.

Figura 63 - Ensaio 2: recetores de rádio SX1276 (figura do lado esquerdo) e SX1278 (figura do lado direito)

No terceiro ensaio (Figura 64), a uma distância de 2,74 km, não foram verificadas variações significativas em relação ao segundo ensaio. Com o módulo SX1276, obteve-se um valor médio de RSSI de -113 dBm, enquanto com o módulo SX1278, o resultado obtido foi igual ao do ensaio anterior (-103 dBm).

Figura 64 - Ensaio 3: recetores de rádio SX1276 (figura do lado esquerdo) e SX1278 (figura do lado direito)

No penúltimo ensaio (Figura 65), com o recetor a 4,20 km do emissor, o módulo SX1276 apresentou dificuldades na entrega de algumas tramas, tendo algumas sido perdidas e outras recebidas com ruído. Ainda assim, este módulo conseguiu receber perto de 90% das tramas emitidas, obtendo um valor médio de RSSI de -118 dBm. Por outro lado, ainda no ensaio quatro, o SX1278 manteve o desempenho dos ensaios anteriores, obtendo um RSSI médio de -103 dBm.

Figura 65 - Ensaio 4: recetores de rádio SX1276 (figura do lado esquerdo) e SX1278 (figura do lado direito)

No último ensaio (Figura 66), separados entre 6,20 km de distância, apenas o módulo SX1276 conseguiu receber as tramas do emissor. Embora não na sua totalidade, o módulo recetor conseguiu capturar aproximadamente 70% das tramas enviadas pelo emissor. Já o módulo SX1278 enfrentou dificuldades em receber as tramas, perdendo ou recebendo com ruído uma parte significativa das tramas emitidas.

Figura 66 - Ensaio 5: recetor de rádio SX1276

Em resumo, a Tabela 19 apresenta os resultados dos testes de cada um dos cinco locais para cada um dos três módulos de rádio. Para facilitar a análise dos resultados, recorreu-se a uma determinada simbologia para categorizá-los. O significado de cada símbolo encontra-se legendado abaixo.

✓ - Recebeu 90% ou mais das tramas emitidas;

☐ - Recebeu entre 89% a 75% das tramas emitidas;

○ - Recebeu entre 74% a 51% das tramas emitidas;

✕ - Recebeu 50% ou menos das tramas emitidas;

— - Teste descartado.

Tabela 19 - Análise dos resultados do teste de alcance

Ensaio	SX1276	SX1278	HC-12
1	✓	✓	✗
2	✓	✓	✗
3	✓	✓	—
4	□	✓	—
5	○	✗	—

Com base nos resultados obtidos nos cinco ensaios, pode-se concluir que dos três módulos testados, o módulo SX1276 foi aquele que apresentou um desempenho mais sólido e consistente no somatório de todos os ensaios. Embora este módulo tenha apresentado dificuldades em receber algumas tramas no penúltimo ensaio, ainda assim, conseguiu receber a maioria das tramas provenientes do emissor. Salienta-se também, que o seu RSSI médio se manteve consistente em todos os locais de teste.

Excetuando-se o último teste, o módulo SX1278 demonstrou uma excelente capacidade de comunicação na maioria dos ensaios, sem apresentar grandes comprometimentos. Se apenas fossem considerados os quatro primeiros testes, este seria o módulo com os melhores resultados entre os três transcetores testados. Por outro lado, o HC-12 não conseguiu receber nenhum pacote desde o segundo ensaio, o que o torna inadequado para utilização em projetos que requerem confiabilidade e consistência em transmissões de longo alcance.

4.2.1.2 Fiabilidade

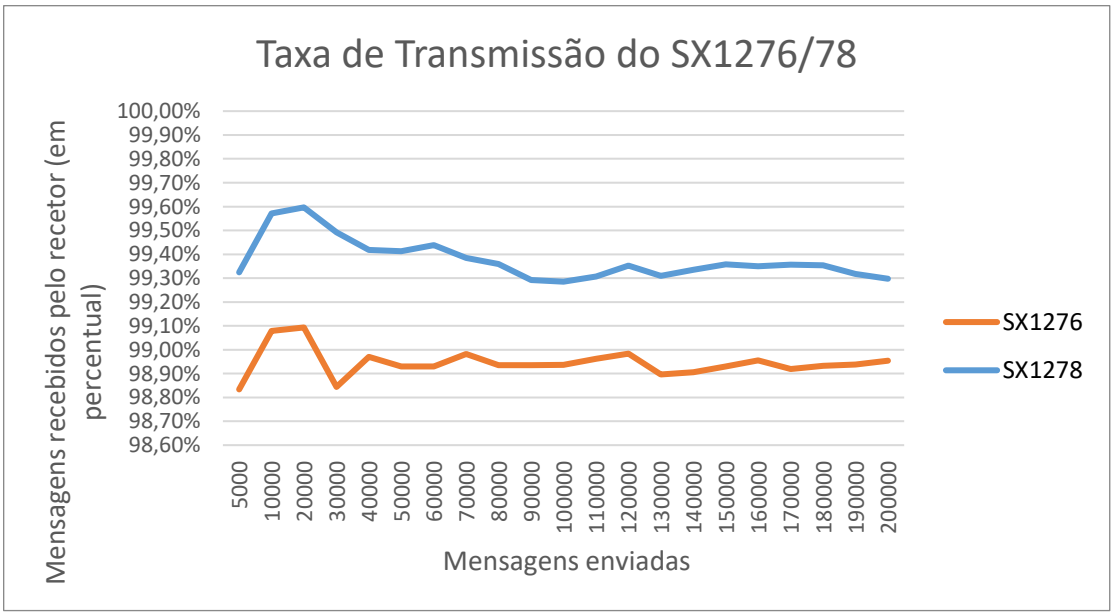
Após o teste de alcance, foi realizado o teste de fiabilidade com os módulos SX1276 e SX1278, com o propósito de avaliar a consistência da transmissão destes dois módulos de rádio ao longo do tempo. O HC-12 foi excluído deste teste devido ao resultado insatisfatório no teste de alcance.

As configurações dos módulos foram mantidas: *Bandwidth* 125 kHz e *Spreading Factor* 7. Através da tecnologia LoRa, durante aproximadamente 12 dias, em intervalos de 5 segundos, o emissor enviou tramas de 256 bytes contendo uma mensagem aleatória e um

contador (contabilização do número de envios do emissor). O teste de fiabilidade destes dois módulos ocorreu separadamente, utilizando-se apenas um emissor e um recetor responsável por armazenar as tramas recebidas. O local de execução do teste foi nos laboratórios da universidade Lusíada, ficando o emissor no laboratório de projetos de engenharia e o recetor no laboratório de processamento de sinal, conforme é mostrado na Figura 67.

Figura 67 - Posicionamento dos respetivos módulos durante o teste de fiabilidade nos laboratórios da universidade

Após os 12 dias de teste para cada módulo de rádio, mais de 200 mil mensagens foram recebidas pelos respetivos recetores. Para efeitos de se obter uma amostra de dados comparável, somente foram consideradas as primeiras 200 mil mensagens enviadas pelo emissor. No gráfico abaixo é apresentado a taxa de transmissão de dados entre o emissor e recetor do SX1276 (linha laranja) e SX1278 (linha azul).



A taxa de transmissão é determinada pelas mensagens recebidas pelo recetor em função das mensagens enviadas pelo emissor. Para complementar a atual análise, na Tabela 20 são apresentados alguns parâmetros referentes como a taxa de transmissão média (T. T. Média), valores médios de transmissão máximos e mínimos (V. M. T. Máximo e Mínimo), variações máximas entre taxas de transmissão, tempo de teste da amostra e as tramas não recebidas pelo recetor.

Tabela 20 - Análise dos resultados obtidos

	SX1276	**SX1278**
T. T. Média	98,96%	99,38%
V. M. T. Máximo	99,09%	99,60%
V. M. T. Mínimo	98,83%	99,29%
Variação Max.	0,26%	0,31%
Tempo de Teste	299:41:24	300:18:03
Tramas Perdidos	1932 (0,97%)	1305 (0,65%)

A partir dos resultados da Tabela 20, pode-se concluir que ambos os módulos de rádio obtiveram taxas de transmissão elevadas, aproximando-se de 99%. No entanto, o módulo SX1278 obteve um desempenho ligeiramente melhor que o SX1276, com um valor maior na taxa de transmissão média, o que resulta numa maior consistência na transmissão de dados. Quando se trata de mensagens não recebidas, o SX1278 também apresentou um melhor resultado, perdendo apenas 0,65% das tramas enviadas pelo emissor. Com base nestes resultados, pode-se concluir que o módulo de rádio com o melhor desempenho neste teste foi o SX1278.

4.2.1.3 Análise e conclusões finais

Após uma análise individual dos testes de alcance e fiabilidade, foi possível determinar que no alcance o SX1276 foi mais consistente, enquanto na fiabilidade o SX1278 alcançou ligeiramente melhores resultados.

Do ponto de vista técnico, o SX1276 possui uma vantagem significativa em relação ao SX1278: suporta um grupo de faixas de frequências mais amplo. O SX1276 suporta frequências entre 137 e 1020 MHz, enquanto o SX1278 suporta uma faixa de frequências

menor, variando entre 137 e 525 MHz. Quanto mais frequências de operação suportar, maior será a compatibilidade e usabilidade do módulo de rádio em função da localização geográfica de operação.

Analisando em termos disponibilidade e custo de aquisição, verifica-se que ambos são relativamente acessíveis, encontrando-se disponíveis em diversos fornecedores de produtos eletrónicos. No entanto, o custo de aquisição de cada um difere. Atualmente, o custo de aquisição do chip SX1276 é sensivelmente 20% superior ao custo do SX1278. Esta informação comprova-se verificando os preços catalogados pelos distribuidores oficiais da marca [74], [75].

Com base na análise apresentada, pode-se concluir que a escolha do módulo de rádio SX1276 é a mais adequada para se implementar no dispositivo a desenvolver. Embora o SX1278 seja uma opção mais económica, a banda de frequências suportada é significativamente menor do que a do SX1276, o que pode limitar a sua utilização em determinados contextos. Por outro lado, o SX1276 apresenta suporte a uma faixa de frequências mais ampla, aumentando a sua versatilidade e capacidade de adaptação a diferentes disponibilidades espectrais, em função de cada região geográfica. Ainda que o SX1276 seja uma opção com um maior custo do que o SX1278, como se trata de valores relativamente baixos, a diferença de preço não é significativa o suficiente para justificar a escolha da opção mais económica em detrimento da mais adequada.

4.2.2 Configuração do Dispositivo

Antes de utilizar o dispositivo, o utilizador deve proceder à sua configuração. A configuração do dispositivo é suportada pelo Master MCU e divide-se em duas fases: (1) configuração de uma rede Wi-Fi – de modo a comunicar com as APIs do sistema – e (2) a associação ao utilizador.

Começando pela primeira fase, uma vez que o dispositivo não possui inicialmente nenhuma rede Wi-Fi configurada, o mesmo colocará automaticamente o seu chip de Wi-Fi no modo de ponto de acesso (WIFI_MODE_AP), criando uma rede para que os utilizadores possam conectar os seus smartphones. Para se conectarem à respetiva rede, basta aceder à aplicação móvel da plataforma, iniciar sessão e adicionar um novo dispositivo (Adicionar Dispositivos). Depois de estarem conectados na mesma rede, o utilizador deverá introduzir as credenciais da rede Wi-Fi à qual deseja que o dispositivo se conecte. Depois de conectado, o dispositivo irá guardar na memória não volátil as credenciais recebidas. Desta forma, na

próxima vez que for iniciado, o dispositivo irá tentar conectar-se novamente à rede sem a necessidade de o utilizador ter de inserir novamente as credenciais de acesso à rede.

Após esta primeira fase, o dispositivo terá de se conectar à API antes de passar para a segunda fase de configuração. Antes de se conectar, o seu chip de Wi-Fi será colocado no modo de estação (WIFI_MODE_STA). Ao conectar-se à API, o dispositivo irá verificar na memória não volátil se existe algum Token de Acesso de Dispositivo. No caso de ser a primeira configuração, não existirá nenhum *token*, e por esse motivo, o dispositivo utilizará as suas próprias credenciais[15] para se autenticar na API. Se a autenticação for bem-sucedida, o dispositivo irá receber dois *tokens*: um de acesso e outro de atualização. O Token de Acesso de Dispositivo poderá ser utilizado pelo dispositivo em autenticações futuras, enquanto estiver dentro do prazo de validade. Após o término desse prazo, o dispositivo deverá utilizar o Token de Atualização de Dispositivo para obter um novo *token* de acesso válido.

Depois da conexão à API ser concluída com sucesso, passará para a próxima fase de associação de dispositivo. Esta fase tem a função, tal como o nome indica, de associar o dispositivo ao utilizador. Para esse efeito, o dispositivo deve encaminhar o Token de Associação de Dispositivos enviado pelo utilizador para a respetiva API (Figura 44, no subcapítulo 4.1.1.2). Este é um *token* de curta validade que assegurará à API que o pedido de associação do dispositivo é autorizado pelo utilizador. Na figura seguinte é exemplificada a estrutura JSON de um ficheiro de configuração armazenado na memória não volátil. Nela estão as respetivas propriedades abordadas nos anteriores parágrafos deste subcapítulo.


```
{
    "ssid":"",
    "password":"",
    "deviceID":"c20df4b9-20eb-4390-a252-4d0e2ac15db8",
    "devicePass":"d0b7357f4a267fd4782bf0e9fd9d5a07",
    "refreshDeviceToken":"",
    "deviceSessionToken":"",
    "deviceAssociationToken":""
}
```


Figura 68 - Exemplo da estrutura do ficheiro de configuração de um dispositivo

4.2.3 Comunicação de Dados Sensoriais entre Dispositivos e Utilizador

Para que haja uma comunicação sem fios bidirecional entre dispositivos e utilizadores, é necessário que ambos estejam conectados à internet ou ligados numa mesma rede privada. Assim, tal como explicado no subcapítulo anterior, além de o utilizador ter de

[15] As credenciais do dispositivo são compostas por um ID (deviceID) e uma senha (devicePass). Estas credenciais são geradas de fábrica e armazenadas na memória não volátil.

conectar o seu telemóvel a uma rede Wi-Fi disponível, terá também a responsabilidade de efetuar o mesmo processo com os seus dispositivos. Só depois do utilizador e os seus dispositivos conectarem-se a uma rede que possibilite a comunicação entre ambos, é que poderão iniciar a troca de dados, encaminhando-os para a API de Transmissão de Dados. Consequentemente, todos os dados transacionados entre o utilizador e os seus dispositivos são apresentados no Monitor de Dados.

Durante o desenvolvimento e implementação desta funcionalidade, surgiram algumas dificuldades devido ao dispositivo projetado ser composto por dois microcontroladores. Uma dessas dificuldades esteve relacionada com a entrega dos dados comunicados entre o utilizador e o Slave MCU. Conforme ilustrado na Figura 69, os dados além de passarem pela API de Transmissão de Dados, terão também de passar obrigatoriamente pelo Master MCU.

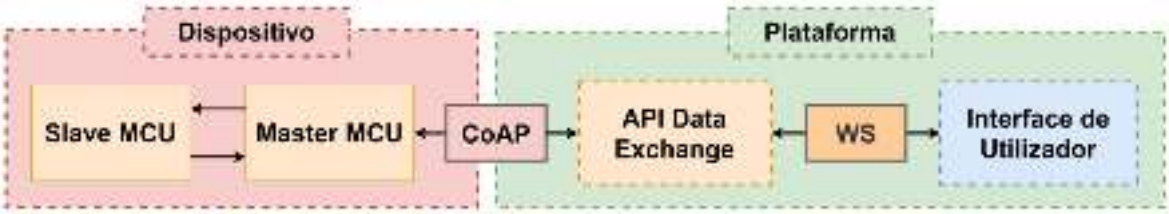


Figura 69 - Diagrama de comunicação entre dispositivos e utilizadores

Uma vez que o Master e Slave MCU se encontram fisicamente no mesmo circuito, considerou-se a utilização de um canal serial para permitir a comunicação entre ambos. Contudo, de modo a verificar a viabilidade desta possível solução, foi necessário analisar os canais seriais fornecidos pelo ESP32 (Tabela 21).

Tabela 21 - Interfaces seriais suportadas pelo ESP32 e as portas correspondentes por padrão

UART	RX IO	TX IO	CTS	RTS
UART0	GPIO3	GPIO1	N/A	N/A
UART1	GPIO9	GPIO10	GPIO6	GPIO11
UART2	GPIO16	GPIO17	GPIO8	GPIO7

De acordo com as informações presentes no seu datasheet, o ESP32 possui três interfaces seriais (UART). A UART0 é utilizada para programar o microcontrolador, enquanto a UART1 é uma interface livre para utilização. Contudo, os fabricantes responsáveis pela produção das placas de desenvolvimento ESP32, não costumam

disponibilizar as GPIOs da interface UART1. Esta decisão justifica-se pelo facto de as GPIOs da interface UART1 serem as mesmas utilizadas pelas interfaces SPI 0/1. Estas são duas importantes interfaces SPI – que são abordadas no próximo capítulo – responsáveis pela comunicação e carregamento do *firmware* na memória flash. Por fim, a UART2 é uma interface disponível para utilização, com GPIOs configuráveis e utilizáveis. Com base nas informações apresentadas, foi escolhida a interface UART2 como meio de comunicação entre o Master e Slave MCU, conforme ilustrado na Figura 70. Esclarece-se que na Figura 70 (e Figura 71), os barramentos seriais são designados pelo prefixo RX (receção) e TX (transmissão), seguidos de um número que identifica a interface serial.

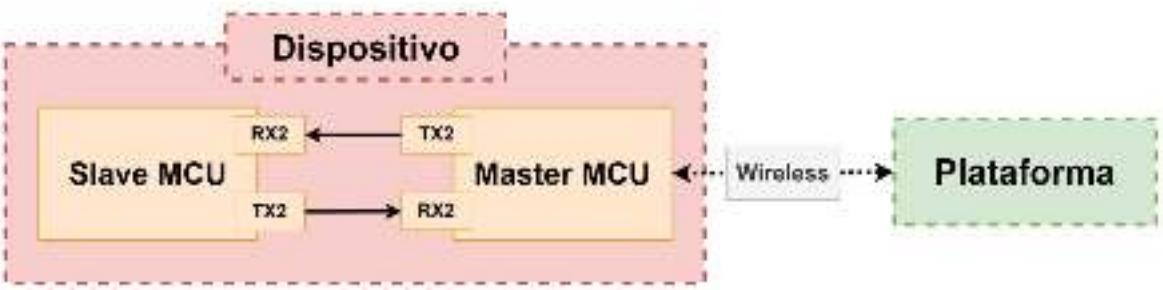


Figura 70 - Comunicação serial entre Master e Slave MCU

Ao utilizar a interface UART2 para este propósito, foi possível obter uma vantagem não planeada: o Slave MCU pode agora comunicar os dados que adquiriu com o utilizador tanto através de *wireless*, como via serial pela interface UART0 (Figura 71).

Figura 71 - A comunicação com utilizador tanto pode ocorrer por *wireless* (lado direito), como pelo barramento serial (lado esquerdo)

É importante salientar que não seria possível realizar o demonstrado na figura anterior com apenas uma única interface serial, devido ao facto de o protocolo UART suportar uma topologia de ponto-a-ponto e não de barramento, tal como acontece nos protocolos I^2C e SPI.

Durante os testes à comunicação serial entre os dois microcontroladores, foi identificado um problema recorrente: sempre que o Slave MCU era reiniciado, era transmitido ruido para o Master MCU através do respetivo barramento serial. Após uma análise detalhada, consultando a documentação do microcontrolador, foi concluído que este problema estava relacionado com flutuações do nível lógico – conhecido por alta impedância (Hi-Z) – presente nas respetivas GPIOs do barramento UART2 (GPIO16 e GPIO17). Durante a inicialização do microcontrolador, segundo o *Pin List* do ESP32 [76], as portas do respetivo barramento são configuradas com o *pull-down* e *pull-up* interno, e o modo input e output desabilitados. Nesta situação, o estado destas duas portas digitais está propenso a flutuações causadas principalmente pelo ruido elétrico presente no circuito, gerando consequentemente flutuações nos níveis de tensão nestas portas de comunicação.

Para resolver este problema, foram adicionadas duas resistências de *pull-up* de 4,7 kΩ em cada um dos dois barramentos seriais, conforme ilustrado na Figura 72. Estas resistências ajudam a reduzir o ruído elétrico presente no circuito, melhorando a qualidade da transmissão serial e minimizando a possibilidade de flutuações nos níveis de tensão do circuito. Com esta solução, foi possível estabilizar a transmissão serial entre os dois microcontroladores e garantir uma comunicação eficaz e sem erros.

Figura 72 - *Pull-up* no barramento serial UART2 do Master e Slave MCU

Clarificando que, na figura anterior, as legendas "M16" e "M17" representam as GPIO16 e GPIO17 do Master, respetivamente. Do mesmo modo, as legendas "S16" e "S17", representam as GPIO16 e GPIO17 do Slave.

4.2.4 Programação Multilinguagem do Slave MCU

A programação do Slave pode ser realizada através de USB, um método mais tradicional e comum, ou através da atualização OTA. Ao contrário da programação via USB, na atualização OTA, o master atua como intermediário entre o utilizador e o Slave, recebendo os binários do utilizador para, posteriormente, carregá-los no Slave MCU.

Na atualização OTA, após o utilizador conectar o seu telemóvel e dispositivos a uma rede que possibilite a comunicação entre ambos, poderá então proceder à atualização de *firmware* do Slave MCU. Para isso, bastará aceder à página "Carregar Firmware" da aplicação móvel, escolher o dispositivo que pretende atualizar e selecionar um dos quatro *firmwares* pré-compilados disponíveis na aplicação. Depois disso, o utilizador apenas precisará de aguardar pela notificação – apresentada na página de Notificações – que irá informá-lo do resultado deste processo. Ao aguardar a conclusão deste processo, os binários correspondentes ao *firmware* escolhido pelo utilizador são transferidos para a API de Atualização de Firmware. Esta, por sua vez, irá validar e armazenar os ficheiros de upload antes de os enviar para o Master MCU. Após a transferência dos binários da API para o Master, cabe a este último a responsabilidade de carregá-los no Slave.

Nos próximos dois capítulos, será apresentado o processo de atualização do dispositivo. No subcapítulo 4.2.4.1 será abordado o processo de carregamento dos binários na memória Slave MCU, por parte do Master MCU. Já no subcapítulo 4.2.4.2 será analisado o processo de atualização OTA entre o dispositivo e a plataforma.

4.2.4.1 Leitura/Escrita na Memória Flash do Slave MCU

O desenvolvimento do processo de programar o Slave MCU com diferentes linguagens de programação, teve como principal objetivo, encontrar um modo de o Master receber os binários da API e os carregar diretamente na memória flash externa do Slave. Para isso, a primeira etapa consistiu em entender como o Master poderia aceder à memória flash do Slave. Após analisar a documentação do ESP32 [76], [77], verificou-se que o ESP32 possui quatro interfaces SPI, cada uma associada a determinados pinos (Tabela 22):

Tabela 22 - Interfaces SPI fornecidas pelo ESP32

Interfaces SPI	MOSI	MISO	CLK	CS
SPI0/1	GPIO 08	GPIO 07	GPIO 06	GPIO 11
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 05
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

A interface SPI0 e SPI1 (ou SPI0/1) são utilizadas internamente para aceder à memória flash externa do dispositivo. Ambas as interfaces partilham os mesmos sinais de barramento SPI, existindo um controlador para determinar qual delas poderá aceder ao

barramento. Por outro lado, a SPI2 e SPI3 são interfaces de uso geral, por vezes designados como HSPI e VSPI, respetivamente. Estas duas interfaces estão abertas aos utilizadores, possuindo sinais de barramento independentes com os mesmos nomes atribuídos a cada uma das interfaces.

Depois de analisadas as interfaces SPI, procedeu-se à realização de alguns testes. O primeiro consistiu em verificar se o Master MCU, utilizando um dos seus barramentos de uso geral (VSPI ou HSPI), conseguiria escrever e ler nos endereços da memória flash externa do Slave através do barramento SPI0/1 (conforme o ilustrado na Figura 73). Contudo, antes mesmo de ser executado, surgiram algumas incertezas sobre a viabilidade deste teste. Isto pelo facto de ocorrer pelo barramento SPI0/1 a transferência de dados entre os periféricos internos[16] do microcontrolador. O uso deste barramento para fins gerais, além de não ser recomendando, pode até pôr em risco o funcionamento consistente do microcontrolador. Apesar disso, para validar em termos práticos as considerações mencionadas anteriormente, o teste foi realizado. O resultado foi tendencialmente coerente com o previsto: o Master não enfrentou dificuldades significativas ao ler os endereços do Slave, porém, na escrita o resultado foi o oposto, observando-se na maioria das vezes dificuldades para a conclusão bem-sucedida do processo.

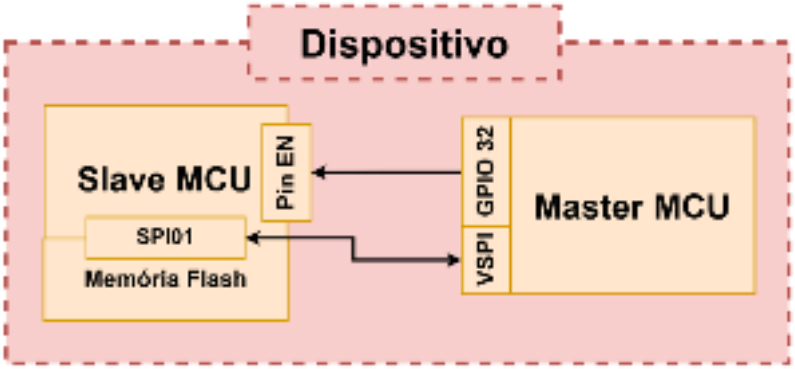


Figura 73 - Disposição dos elementos em blocos utilizados no teste

Na figura anterior, através da porta GPIO 23 do Master MCU, o pino EN do Slave MCU é configurado com um nível lógico zero, de modo evitar conflitos de acesso na sua memória flash. Durante a inicialização do Slave MCU, se o pino EN permanecer com um nível lógico baixo, obrigará o Slave entra no modo "*Download Boot*". Neste modo, o microcontrolador aguarda apenas pela receção de um novo *firmware* através da interface SPI0/1, sem iniciar o carregamento de *firmware* armazenado na memória. Assim, enquanto

[16] A memória flash externa do microcontrolador é considerada um periférico interno.

o Slave estiver neste modo, a memória flash fica livre para ser acedida pelo master, sem que haja risco de conflitos de acesso.

Ao passar para o segundo teste, o objetivo permaneceu o mesmo do primeiro. No entanto, desta vez tentou-se aceder diretamente aos pinos da memória do Slave, em vez de utilizar o barramento SPI0/1. Para tal, foi necessário aceder ao respetivo integrado da memória, presente no microcontrolador ESP32. No entanto, surgiu uma dificuldade: por norma, estas placas de desenvolvimento têm um escudo metálico soldado para proteger o módulo contra danos físicos, reduzir as interferências eletromagnéticas externas e ajudar a dissipar o calor gerado pelo próprio (Figura 74).

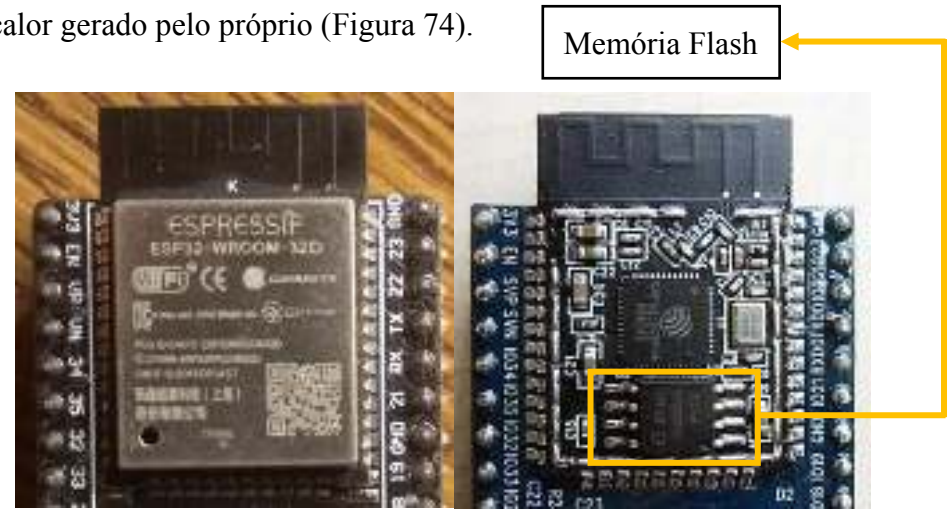


Figura 74 - Módulo ESP32 com escudo metálico (à esquerda) e sem escudo (à direita) com a memória assinalada [78]

De acordo com relatos de alguns utilizadores, constata-se que a remoção do escudo metálico não é uma tarefa simples, havendo riscos de dessoldar os componentes eletrónicos do módulo durante o processo de remoção. Por esse motivo, efetuou-se uma pesquisa por outras placas de desenvolvimento ESP32 sem escudo, encontrando-se uma alternativa fornecida pela SparkFun (Figura 75).

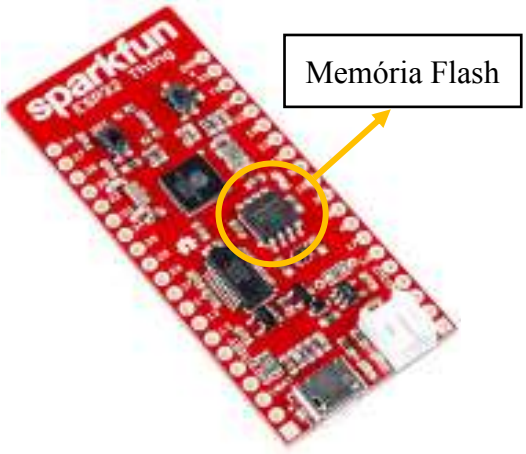


Figura 75 - Placa de desenvolvimento da SparkFun ESP32 Thing sem o escudo metálico com a memória assinalada [79]

Imediatamente após, foram estabelecidas as conexões entre o barramento VSPI do Master MCU e a memória flash do Slave MCU. Antes de se soldar os condutores nos terminais da memória, consultou-se o *datasheet* da memória flash (Winbond W25Q32JV) presente no Slave MCU. O esquema das conexões encontra-se representado na Figura 76.

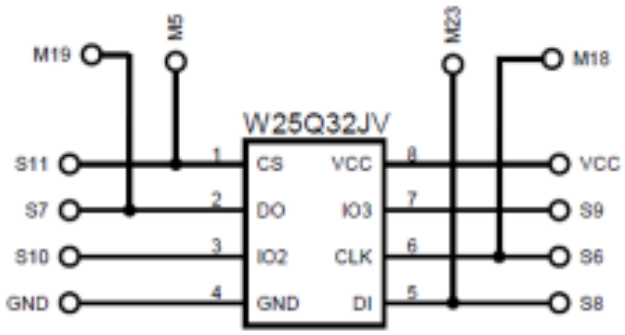


Figura 76 - Esquemático das ligações à memória entre Master (M) e Slave (S) - Primeira versão

Ao contrário do primeiro, este teste foi bem-sucedido, uma vez que foi possível que o Master acedesse à memória do Slave enquanto este permanecia no modo "*Download Boot*". Contudo, durante o teste verificou-se que mesmo depois de o processo de escrita/leitura terminar, o Master permanecia conectado ao barramento da memória flash do Slave. Para que o Slave recuperasse o acesso à sua memória, era necessário reiniciar o Master ou colocá-lo no modo "*Download Boot*". Após algumas tentativas fracassadas para reverter esta situação via software, chegou-se à conclusão de que a forma mais adequada de resolver esta questão, seria abrir eletronicamente o circuito de acesso entre o Master e a memória externa do Slave. Para isso, foram utilizados quatro transístores, um para cada canal do barramento SPI, permitindo que, através deles, o Master bloqueasse o seu próprio acesso à memória do Slave sempre que fosse necessário (Figura 77).

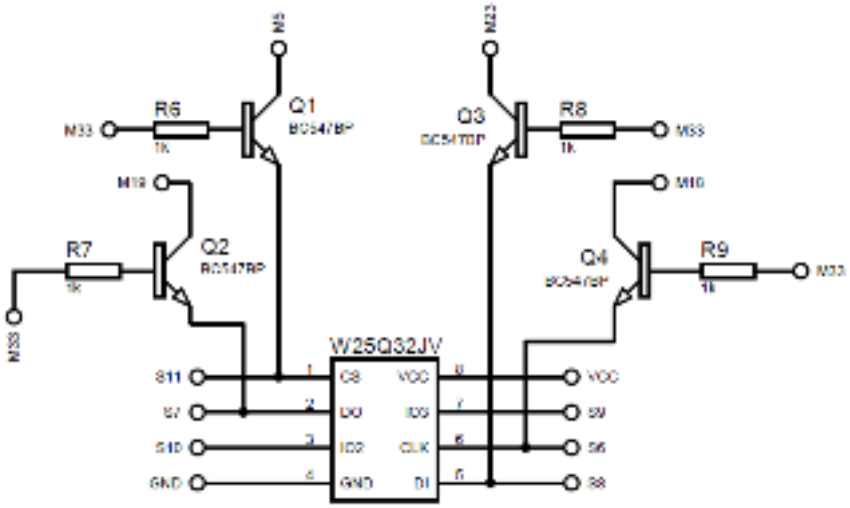


Figura 77 - Esquemático das ligações à memória entre Master (M) e Slave (S) - Segunda versão

Sempre que o Master precisar de efetuar alguma operação na memória do Slave, além de colocar o Slave no modo "*Download Boot*", é necessário que o Master defina o nível lógico alto (1) na sua porta GPIO33 para ativar os transístores que lhe permitirão o acesso à referida memória.

4.2.4.2 Programação "Over The Air"

Um dos objetivos deste trabalho, consiste em fornecer a possibilidade de realizar atualizações sem fios ao *firmware* do Slave. Para se alcançar este objetivo, uma das opções disponíveis é através da existente funcionalidade "*Over The Air*" (OTA), que possibilita a atualização remota de *firmware* dos dispositivos, sem a necessidade de conectá-los fisicamente a um computador ou um equipamento similar. A utilização do ecossistema disponibilizado pelo Arduino possibilita o uso da biblioteca "ArduinoOTA". Esta biblioteca permite uma rápida implementação da tecnologia OTA no microcontrolador, podendo o utilizador efetuar vários tipos de configurações, tais como selecionar a partição da memória onde os binários de *firmware* serão armazenados.

Após os avanços apresentados no subcapítulo anterior, o objetivo desta fase consistiu em garantir que o Master recebesse e carregasse os ficheiros de *firmware* na memória do Slave. Na Figura 78 é apresentado o diagrama que demonstra o processo discutido neste parágrafo.

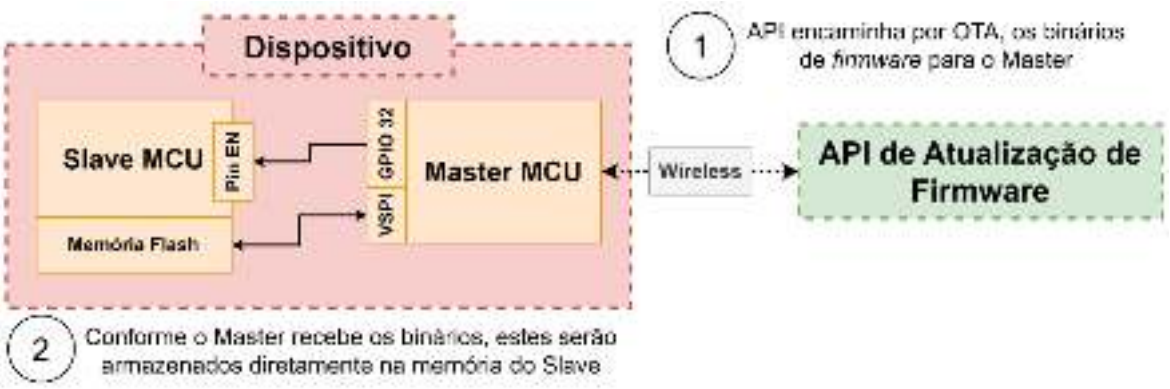


Figura 78 - Diagrama de atualização do *firmware* do Slave MCU (Primeira Proposta)

Apesar das várias tentativas, não foi possível concretizar a proposta descrita no parágrafo anterior. No decorrer dos testes, verificou-se que a biblioteca "ArduinoOTA" permite que o microcontrolador carregue os ficheiros recebidos na sua própria memória *flash*. No entanto, o mesmo não se aplica quando se trata de carregar os ficheiros diretamente numa memória externa, como é o caso da memória *flash* do Slave para com o Master. Logo, para alcançar o objetivo de o Master carregar os ficheiros de *firmware* no Slave, foi

necessário fazer uma alteração ao plano inicial: em vez do Master carregar os ficheiros de *firmware* diretamente na memória do Slave, deverá primeiramente armazená-los na própria memória e só depois deverão ser transferidos para a memória do Slave (Figura 79).

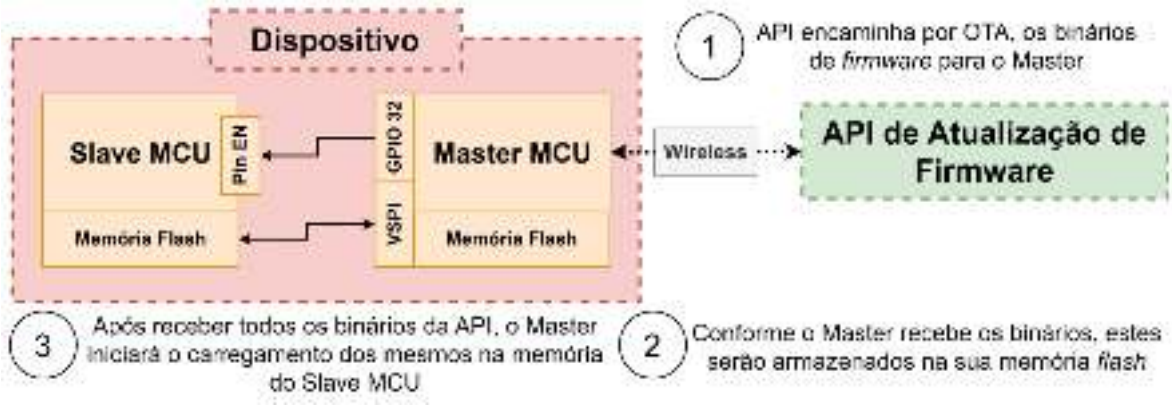


Figura 79 - Diagrama de atualização do *firmware* do Slave MCU (Segunda Proposta)

Uma vez que os ficheiros devem ser armazenados pelo Master numa partição de memória destinada para esse efeito, foi necessário analisar a tabela de partições utilizada por padrão no ESP32 (Tabela 23), com o propósito de verificar a viabilidade desta proposta. Imediatamente após a Tabela 23, encontra-se uma breve descrição de cada partição.

Tabela 23 - Tabela de partições utilizada por padrão pelo ESP32

Nome	Tipo	Subtipo	Offset	Tamanho
nvs	data	nvs	0x9000	0x5000
otadata	data	ota	0xe000	0x2000
app0	app	ota_0	0x10000	0x140000
app1	app	ota_1	0x150000	0x140000
spiffs	data	spiffs	0x290000	0x170000

1. NVS (*Non-Volatile Storage*): é uma partição de armazenamento não-volátil que contém dados persistentes do sistema, como configurações do Wi-Fi, informações de rede, etc.
2. OTA Data: contém informações relacionadas com as atualizações de *firmware* que foram realizadas no dispositivo por OTA. Esta partição é utilizada pelo sistema para verificar se há alguma atualização de *firmware* disponível, armazenando as informações necessárias para realizar a atualização.

3. APP0: É a partição principal da aplicação, que contém o código executável do programa principal. O código presente nesta partição é executado quando o dispositivo é iniciado.
4. APP1: É a partição secundária da aplicação, que pode ser utilizada para armazenar uma cópia de segurança do código executável. Esta partição é útil no caso de ocorrer algum problema com a partição APP0, pois possibilita a recuperação do *firmware* anteriormente carregado.
5. SPIFFS (*SPI Flash File System*): É uma partição que tem a função de armazenar vários tipos de ficheiros, desde imagens, páginas HTML, etc. É semelhante a um sistema de arquivos num computador, sendo útil para aplicações que precisam de armazenar dados de forma persistente, como logs ou configurações.

Na Tabela 23, além do nome de cada partição, são descritos o tipo e subtipo de cada partição, que definem o formato da partição e o seu propósito, respetivamente. O offset de cada partição representa a posição de memória onde a partição se situa, indicando o primeiro endereço de memória dessa partição. Já o tamanho de cada partição é apresentado em hexadecimal e representa a capacidade da partição em bytes.

Após a análise das partições, concluiu-se que a partição mais adequada para armazenar os ficheiros de *firmware* do Slave seria a partição spiffs do Master MCU. A justificação prende-se com o facto de a partição spiffs ter a capacidade de armazenamento de 0x170000 bytes (equivalentes a 1,4375 MB em decimal), enquanto a app0, responsável por armazenar o programa do utilizador, possui um tamanho de 0x140000 bytes (equivalentes a 1,25 MB). Considerando que o programa não poderá exceder os 1,25 MB e possuindo a memória spiffs 1,4375 MB, conclui-se que a memória disponível na partição spiffs é mais do que suficiente para guardar os binários a serem carregados no Slave MCU.

Tal como já foi referido no capítulo 4.2.2, uma vez que a partição spiffs do Master está a ser utilizada para armazenar configurações internas do microcontrolador, foi necessário dividi-la em duas partições: uma destinada a esse tipo de configurações e outra para armazenar os ficheiros de *firmware* a carregar-se na memória flash do Slave. Ao criar uma partição, devem ser tidos em conta alguns fatores, com destaque para os seguintes [80]:

1. O nome atribuído à partição deve ter no máximo 16 bytes de comprimento.

2. O tipo deve ter um valor entre 0 e 254, sendo normalmente utilizados o 0x00 (app) e 0x01 (data).
3. O subtipo deve ser apenas especificado para partições do tipo app e data. Os subtipos a serem atribuídos variam consoante o tipo da partição escolhido.
4. O valor definido como offset e tamanho devem ser múltiplos de 4 KB.

Nesse sentido, a partição spiffs apresentada na Tabela 23, foi substituído por duas novas partições: spiffs0 e spiffs1. A nova partição spiffs0 destina-se para as configurações do microcontrolador e a spiffs1 para armazenar os binários que serão carregados no Slave. Na Tabela 24 é apresentado o resultado da nova tabela de partições do Master MCU.

Tabela 24 - Nova tabela de partições do Master MCU

Nome	Tipo	Subtipo	Offset	Tamanho
nvs	data	nvs	0x9000	0x5000
otadata	data	ota	0xe000	0x2000
app0	app	ota_0	0x10000	0x130000
app1	app	ota_1	0x140000	0x130000
spiffs0	data	spiffs	0x270000	0x4000
spiffs1	data	spiffs	0x274000	0x18C000

A partição spiffs0 foi configurada com um tamanho de 0x4000 Bytes, o que equivale a 16 KB. Inicialmente, optou-se por um tamanho de 4 KB, uma vez que este é um valor mais do que suficiente. No entanto, durante os testes verificou-se que ao atribuir tamanhos de partição inferiores a 16 KB, ocorriam erros de execução no microcontrolador. Já na partição sppifs1 foi atribuído o restante espaço de 0x18C000 Bytes (1.547 MB).

Depois da restruturação da memória do Master MCU, o próximo e último passo consistiu em carregar os binários por OTA na partição spiffs1 da memória *flash* do Master. Desta forma, os ficheiros enviados pela API de Atualização de Firmware, seriam carregados pelo Master na partição app0 da memória *flash* do Slave (conforme o ilustrado na Figura 80).

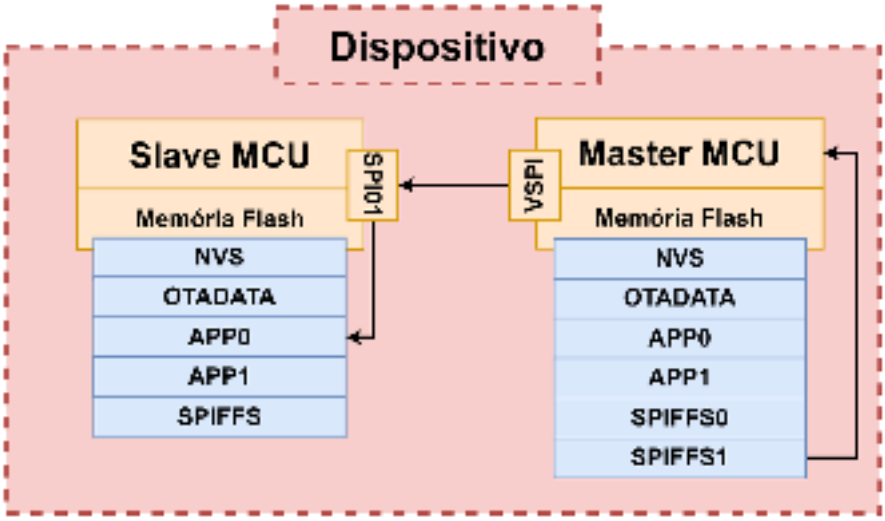


Figura 80 - Carregamento dos binários de *firmware* no Slave MCU através do Master MCU

Durante os testes, foi possível constatar que a integração entre os dois microcontroladores foi bem-sucedida. Através do barramento VSPI, o master conseguiu comunicar com a memória *flash* do Slave, carregando os respetivos binários enviados pelo utilizador. Importante salientar, que enquanto este processo decorre, a porta EN do Slave é colocada no nível lógico baixo, impedindo-o de iniciar e de se conectar à sua própria memória *flash*. Assim, o acesso à memória fica livre para ser programado pelo Master. Após o carregamento, a porta EN volta ao seu nível lógico padrão (nível alto), permitindo ao Slave voltar ao seu funcionamento padrão. Ao regressar à sua configuração padrão, o Slave verificará a existência de um *firmware* na partição APP0 e procederá automaticamente ao seu carregamento e execução.

4.2.5 Montagem e Considerações Finais do Dispositivo

Ao longo do desenvolvimento do sistema IoT, houve uma particular preocupação em atingir os objetivos relacionados com o dispositivo. Esta preocupação surgiu tanto devido ao valor que as concretizações destes objetivos acrescentariam ao projeto, como também ao impacto significativo de que o cumprimento – ou não – destes objetivos provocariam no resultado do projeto. Por exemplo, se no decorrer dos testes, se confirmasse que não era possível programar o dispositivo com várias linguagens de programação, não teria sido necessário desenvolver e integrar as ferramentas relacionadas com o carregamento de *firmware* na plataforma. Assim, para que a concretização deste projeto fosse sucedida, tornou-se evidente a relevância de alcançar os três objetivos mencionados no início da secção 4.2.

Dito isto, e após o desenvolvimento do dispositivo, pretende-se proceder-se a uma retrospetiva, de forma a verificar se o trabalho desenvolvido cumpriu ou não cada um dos três objetivos mencionados. Analisando cada um desses três objetivos, conclui-se que:

1. O objetivo de disponibilizar múltiplas tecnologias de comunicação no dispositivo foi alcançado com sucesso. Foram utilizados dois microcontroladores ESP32, que já incluem as tecnologias de Wi-Fi, Bluetooth., I^2C, SPI e UART. Além disso, foi agregada uma tecnologia de longo alcance no Slave MCU – conforme apresentado no capítulo 4.2.1 – com objetivo de disponibilizar um dispositivo com um leque de tecnologias de comunicação ainda mais diversificado, permitindo ao utilizador desenvolver a sua solução IoT de acordo com as circunstâncias em que se insere.
2. A comunicação bidirecional sem fios também foi um dos objetivos alcançados. O dispositivo é capaz de se comunicar com a interface do utilizador através de Wi-Fi, como é apresentado no capítulo 4.2.3.
3. Uma das principais funcionalidades a implementar-se era a programação do dispositivo com diferentes linguagens. Este objetivo foi concluído, podendo o utilizador programar o Slave MCU utilizando quatro linguagens de programação, conforme apresentado no capítulo 4.2.4.

De modo a fornecer uma visão geral de todos os componentes apresentados e abordados, a Figura 81 apresenta o diagrama de blocos do dispositivo.

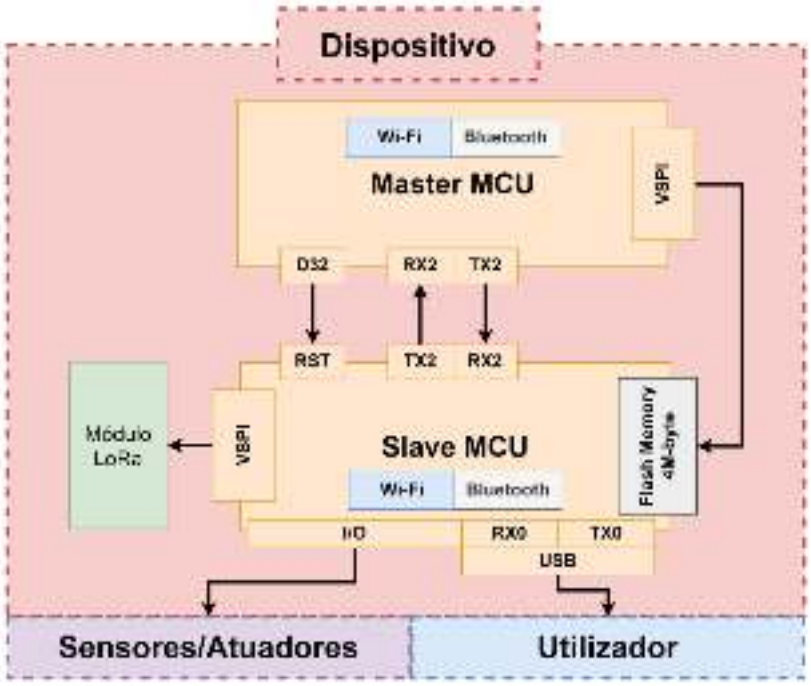


Figura 81 - Diagrama em blocos do dispositivo

Os fluxogramas do código executado pelo dispositivo são apresentados no Apêndice B. Além disso, no Apêndice C, encontra-se o esquema elétrico do dispositivo que foi montado e testado numa *breadboard* (Figura 82), com o propósito de verificar o seu funcionamento durante o processo de desenvolvimento.

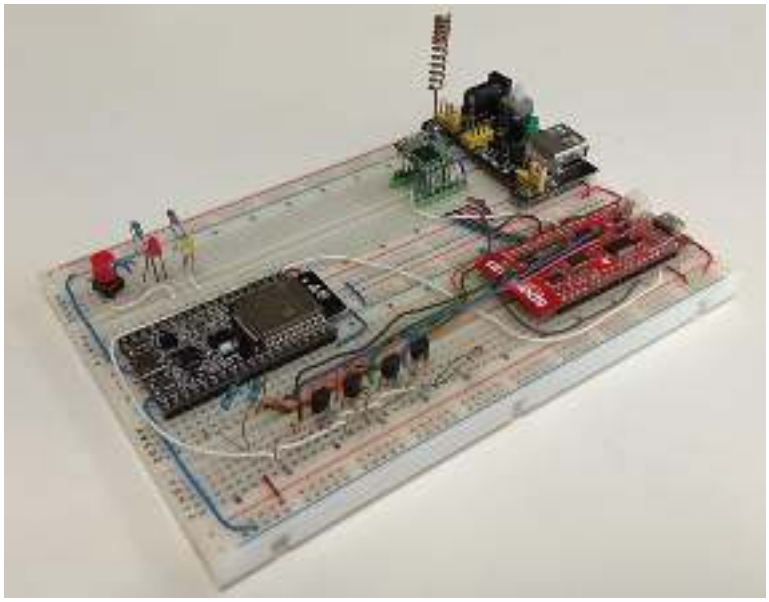


Figura 82 - Montagem do dispositivo protótipo numa *breadboard*

Após os testes e verificações realizados nas secções do subcapítulo 4.2, procedeu-se à melhoria da apresentação e robustez do dispositivo protótipo apresentado na figura anterior. O objetivo foi não apenas aprimorar a aparência, mas também aproximá-lo do aspeto de um produto final. Nesse sentido, procedeu-se à soldadura dos componentes do circuito da *breadboard* (Figura 82) para uma *protoboard*. No entanto, para a conceção desta placa de prototipagem, foi necessário ter em conta alguns aspetos que não foram considerados na montagem do circuito anterior:

1. Dado que o circuito montado na *breadboard* não tinha como objetivo abordar questões relacionadas à alimentação, optou-se por utilizar simplesmente um módulo de alimentação para *breadboard* (localizado no canto superior direito da Figura 82). Neste novo circuito, foram incluídos novos componentes que possibilitam uma alimentação adequada e controlada do circuito.
2. Uma vez que este dispositivo IoT tem como objetivo proporcionar condições de mobilidade, permitindo o seu funcionamento em área remotas, é necessário incluir uma bateria.

Considerando os dois aspetos anteriormente mencionados, para além dos componentes presente no circuito da *breadboard*, foi necessário adicionar novos componentes à *protoboard* desenvolvida. Dentre os componentes adicionados, encontram-se um interruptor de dois estados, um módulo carregador micro USB para baterias de lítio, um conversor DC-DC *Step Up/Step Down*, uma bateria, botões de pressão e um pente de pinos de acesso às GPIOs do Slave.

Na Figura 83, apresenta-se o resultado da *protoboard* desenvolvida, com os principais componentes assinalados e legenda abaixo da mesma.

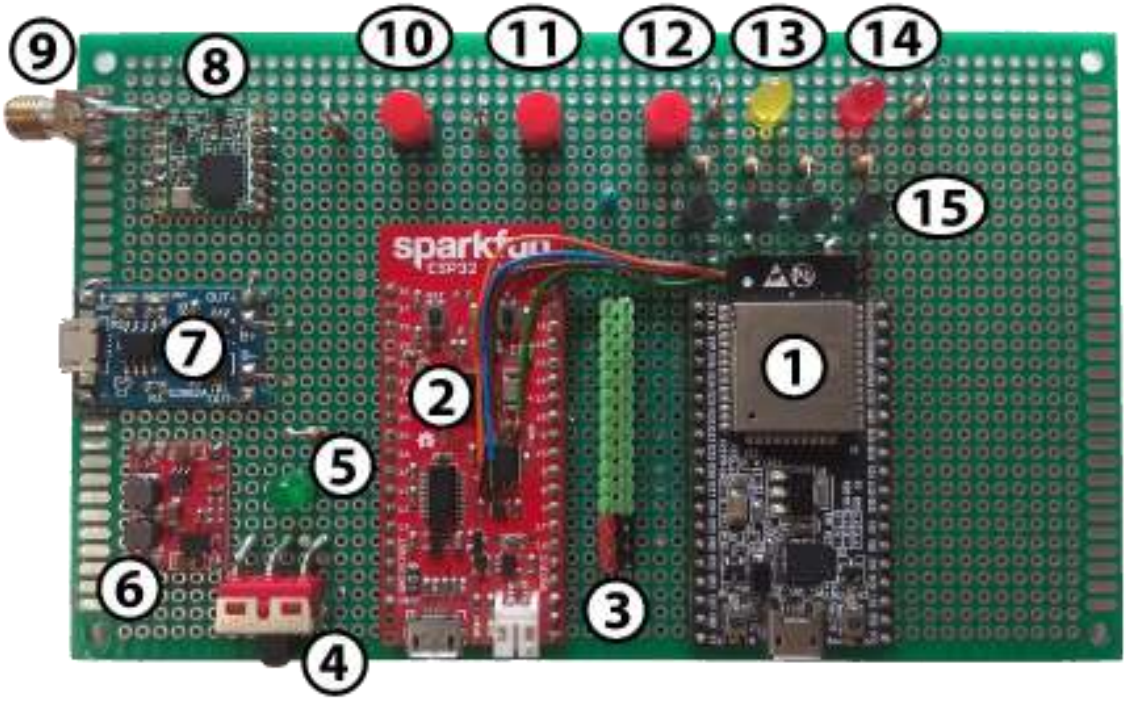


Figura 83 - *Protoboard* desenvolvida com os principais componentes legendados

Legenda:

1. **Master MCU**: microcontrolador responsável pela conectividade do dispositivo com a plataforma;
2. **Slave MCU**: microcontrolador responsável por executar a solução projetada pelo utilizador;
3. **GPIOs do Dispositivo**: pente de 30 pinos que permite ao utilizador o acesso à alimentação (3,3V e GND) e às GPIOs disponíveis do dispositivo, mais especificamente do Slave MCU. Na Figura 84, encontra-se uma descrição mais detalhada sobre cada um dos pinos do pente;

4. **Interruptor do Dispositivo**: um interruptor de dois estados que permite ligar ou desligar o dispositivo;
5. **LED de Estado**: LED que indica se o dispositivo está ligado (LED aceso) ou desligado (LED apagado);
6. **Conversor SO9-DC**: é um conversor DC-DC *step-up/step-down* (*buck-boost*) que, através de uma tensão de entrada entre 3V e 15V, fornece uma tensão de saída de 3,3V e uma corrente de até 0,6A. Este tipo de conversor DC/DC permite que a tensão de saída seja maior, menor ou igual à tensão de entrada, possibilitando uma eficiência de conversão superior a 75%;
7. **Carregador TP4056**: trata-se de um carregador linear de 5V projetado especificamente para baterias de Li-Ion (iões de lítio) de célula única, com uma corrente máxima de carregamento de 1A. O circuito integrado DW01 assegura a proteção da bateria contra sobrecorrente, sobrecarga e descargas excessivas. O TP4056 interrompe automaticamente o processo de carregamento assim que a tensão atingir os 4,2V;
8. **Módulo LoRa**: módulo SX1276 responsável pelas comunicações de longo alcance do dispositivo;
9. **Conector da Antena**: componente utilizado para conectar a antena ao módulo LoRa;
10. **Botão de Reset**: botão de pressão responsável por reiniciar o dispositivo;
11. **Botão de Boot**: botão de pressão responsável por colocar o Slave MCU no modo "*Download Boot*". Este é o modo que o microcontrolador deve ser colocado antes de ser programado.
12. **Botão de Restauro**: botão de pressão responsável por restaurar as configurações de fábrica do dispositivo. Ao pressioná-lo durante 5 segundos, os parâmetros da rede Wi-Fi configurada (SSID e senha) serão removidos. Após este processo, o utilizador poderá definir uma nova rede Wi-Fi para o dispositivo se conectar (conforme abordado no subcapítulo 4.2.2);
13. **LED de Aviso**: LED amarelo que, dependendo da frequência de intermitência luminosa, informa o utilizador sobre o tipo de ocorrência que está a decorrer (discriminadas no Apêndice D). Todas as ocorrências assinaladas por este LED podem ser superadas pelo utilizador.

14. **LED de Erro**: LED vermelho que, com base na frequência de intermitência luminosa, informa o utilizador sobre o tipo de ocorrência que está a decorrer (discriminadas no Apêndice D). Ao contrário do LED de Aviso, as ocorrências sinalizadas por este LED são consideradas críticas e requerem a intervenção do suporte técnico.

15. **Chaves Eletrónicas de Acesso à Memória do Slave MCU**: quatro transístores controlados pelo Master MCU, com a responsabilidade de permitir que o Master MCU aceda à memória flash do Slave MCU. Este tópico é abordado no final do subcapítulo 4.2.4.1.

Ainda situado por baixo da *protoboard*, haverá uma bateria de Li-Ion de 3,7V e com capacidade de 2400mAh. Esta bateria estará conectada eletricamente ao módulo TP4056, que fornecerá energia ao resto do circuito.

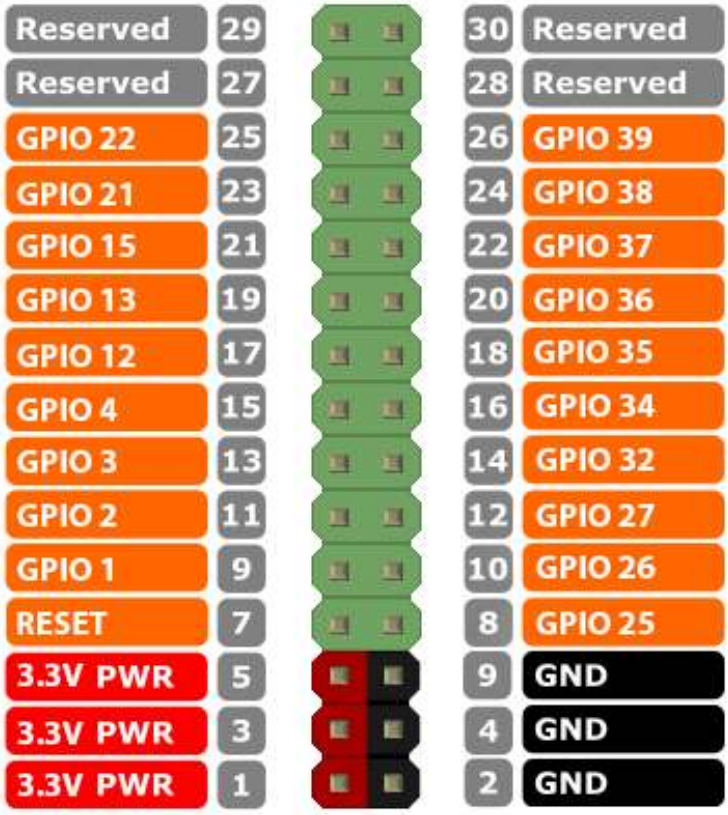


Figura 84 - GPIOs do dispositivo (Slave MCU)

Com o intuito de aliar a elegância à durabilidade do dispositivo, procedeu-se à conceção e impressão 3D de um chassi. Esta estrutura de polímero de ácido polilático (PLA) é constituída por uma base e uma tampa, tal como é demonstrado na Figura 85. A base é responsável por acomodar a *protoboard* e a bateria, garantindo uma maior estabilidade e segurança durante o uso do dispositivo. A tampa, por sua vez, é fixada sobre a base,

proporcionando uma proteção adicional aos componentes internos contra poeiras e possíveis danos externos.

Figura 85 - Base (parte inferior) e tampa (parte superior) do chassi do dispositivo

Durante o processo de design, foram definidas algumas aberturas nas laterais da base para facilitar o acesso a determinados componentes do dispositivo. Com esse propósito, possibilitou-se o acesso conveniente às portas de comunicação UART do Master e Slave MCU (n.º 1 e 2 da Figura 83), ao interruptor do dispositivo (n.º 4), à porta USB do carregador TP4056 (n.º 7) e ao conector da antena do módulo LoRa (n.º 9). Similarmente, na tampa, foram também incluídas vias de acesso às GPIOs do dispositivo (n.º 3 da Figura 83), aos botões de Reset, Boot e Restauro (n.º 10, 11 e 12), e a possibilidade de visualização dos LEDs de Estado, Aviso e Erro (n.º 5, 13 e 14). Posteriormente, para a identificação dos acessos do dispositivo mencionados anteriormente, procedeu-se à gravação a laser de códigos de identificação na tampa do chassi, conforme ilustrado na Figura 86.

Figura 86 - Tampa do dispositivo com os respetivos códigos de identificação gravados a laser

Uma vez concluída a tampa, prosseguiu-se com a montagem do dispositivo. Para tal, procedeu-se à inserção da bateria na base do chassi, como também a conexão da mesma ao respetivo conector da *protoboard* (Figura 87).

Figura 87 - Conexão da bateria ao conector da *protoboard*

Antes de inserir a *protoboard* na base do chassi, recorreu-se uma vez mais à impressora 3D para projetar uma pequena peça que se ajustasse no interruptor localizado na *protoboard*, com o objetivo de tornar mais conveniente para o utilizador o acionamento do interruptor do dispositivo (Figura 88). Esta peça, designada como adaptador de interruptor, para além de melhorar a estética do dispositivo, tem como objetivo tornar mais conveniente o acesso ao seu interruptor.

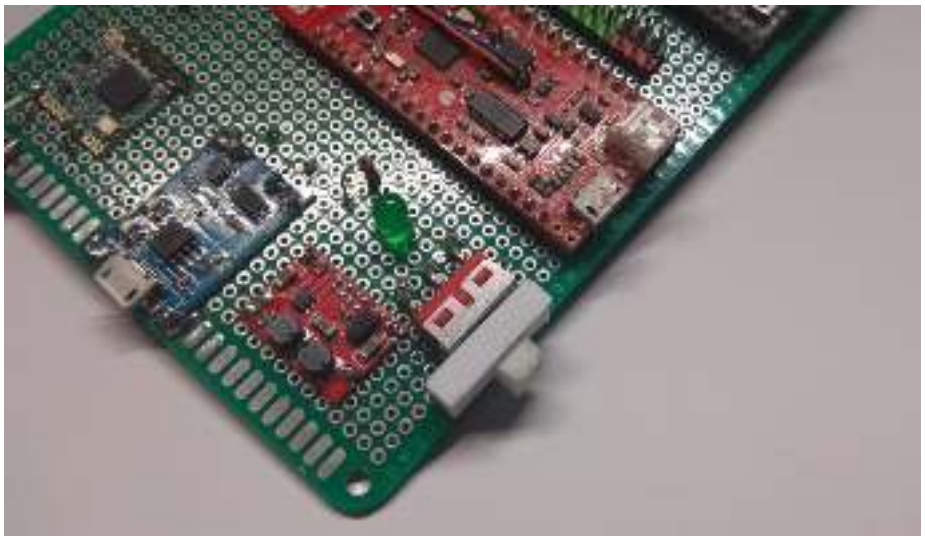


Figura 88 - Adaptador de interruptor inserido no interruptor da *protoboard*

Para fixar a *protoboard* à base do chassi, aparafusou-se dois parafusos nas duas das quatro roscas disponíveis e situadas nos cantos da base (Figura 89).

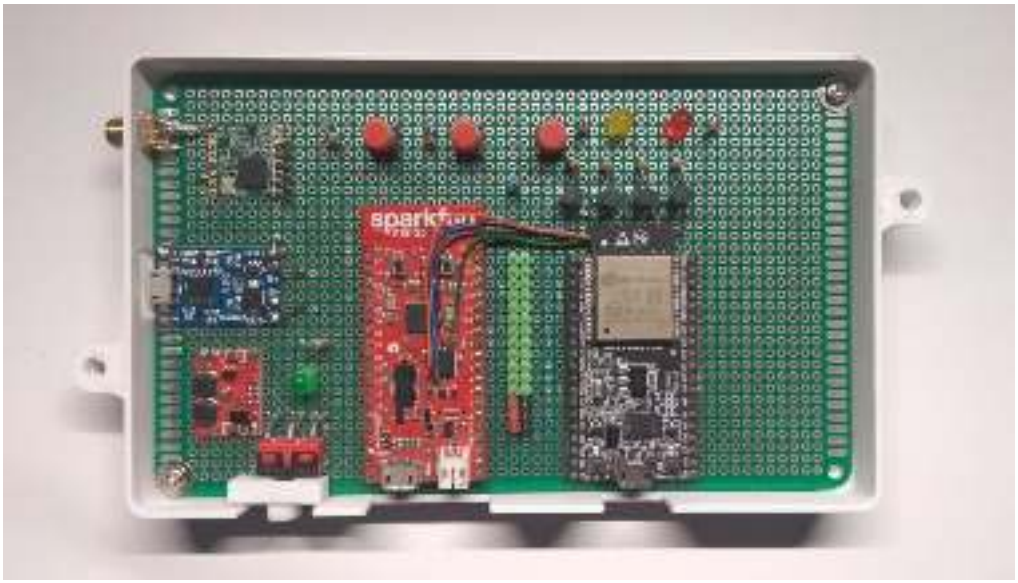


Figura 89 - Inserção da *protoboard* e bateria (instalada por debaixo da *protoboard*) na base do chassi

Posteriormente, fixou-se a tampa do chassi à base através de dois parafusos localizados nas laterais do dispositivo (Figura 90).

Figura 90 - Tampa do chassi aparafusada na base

Após a conexão da antena no respetivo conector, é finalmente apresentado na Figura 91 o resultado do dispositivo.

Figura 91 - Resultado do dispositivo

5 Conclusões e Trabalho Futuro

A presente dissertação centrou-se no estudo e desenvolvimento de uma proposta de sistema IoT, que providencie recursos atualmente não disponíveis no mercado das plataformas/sistemas IoT. A disponibilização destes recursos visa sobretudo, simplificar o processo de conectividade entre o dispositivo e a plataforma, visto que este ainda é um aspeto pouco explorado neste tipo de infraestruturas. O objetivo é simplificar a experiência de desenvolvimento das soluções IoT, inclusive para aqueles com menos experiência na área, de forma a incentivá-los a criar as suas próprias soluções.

A principal vantagem do sistema proposto, reside na disponibilização dos principais componentes de hardware e software para auxiliar no desenvolvimento da solução pretendida. Para além disso, é fornecida uma plataforma composta por APIs e uma IU, assim como um dispositivo PnP, preparado para interagir com a plataforma. Para esse propósito, são disponibilizados recursos como múltiplas tecnologias de comunicação equipadas num único dispositivo, comunicação bidirecional entre o dispositivo e a IU, e a possibilidade de programar o dispositivo com diferentes linguagens de programação. A implementação destes recursos permite ao utilizador concentrar-se exclusivamente no desenvolvimento da solução a implementar, sem ter de se preocupar com as tarefas normalmente exigidas por outras plataformas, tais como a integração do seu dispositivo com a plataforma utilizada.

Destaca-se ainda a mais-valia de o dispositivo ser composto por dois microcontroladores, um dos quais – o Slave MCU – é dedicado exclusivamente ao uso do utilizador, enquanto o outro – o Master MCU – atua como um *gateway* entre o Slave MCU e a plataforma. Deste modo, o Master MCU assume integralmente o processo de conectividade com a plataforma, permitindo que o utilizador se concentre exclusivamente no desenvolvimento da sua solução.

Considera-se que os objetivos propostos inicialmente foram essencialmente alcançados. No entanto, durante o desenvolvimento desta dissertação, foram identificadas algumas possíveis melhorias, das quais se destacam:

1. Implementação do FreeRTOS no Master MCU

   O FreeRTOS é um sistema operativo de tempo real (RTOS - *Real Time Operating System*), sem custos de licença, destinado a microcontroladores compatíveis de baixo custo e potência, tal como o ESP32. A utilização deste sistema operativo apresenta várias vantagens, entre as quais se destacam: o FreeRTOS apresenta uma maior

eficiência no que diz respeito ao consumo de recursos, como de memória e processamento, quando comparado com o núcleo utilizado nesta dissertação (núcleo do Arduino para ESP32); é composto por uma grande comunidade de programadores que disponibilizam vários tipos de recursos, tais como documentação, exemplos de código e fóruns de discussão; permite dividir o programa em várias tarefas independentes e simultâneas (*multitasking*), que podem ser executadas em paralelo. Embora ainda não tenha sido testado, uma das possibilidades para reduzir o dispositivo desenvolvido para apenas um microcontrolador, consistiria na implementação do FreeRTOS. Para tal, o plano passa por dividir o armazenamento e processamento do microcontrolador ESP32 em dois blocos distintos: um para a conectividade e outro para a solução do utilizador. Cada um dos blocos ficaria com endereços de armazenamento independentes e previamente definidos, ocorrendo o seu processamento simultaneamente.

2. Reforçar a segurança do sistema IoT

   O propósito geral desta dissertação relaciona-se com o aumento da usabilidade do sistema IoT, não se concentrando em questões relacionadas com a segurança. Contudo, como qualquer infraestrutura, a segurança é um dos pilares para que um sistema IoT seja bem-sucedido. Numa perspetiva futura, é essencial fornecer a privacidade e integridade aos dados que circulam pelo sistema desenvolvido. Para isso, é fundamental a utilização de algum protocolo de segurança, como TLS, que permita cifrar as informações transmitidas pelos protocolos HTTP e WebSocket. No caso dos dispositivos e APIs que utilizam o protocolo CoAP, pode-se usar o protocolo de segurança DTLS (*Datagram Transport Layer Security*). O DTLS é uma implementação do protocolo TLS simplificada e compatível com comunicações UDP. Podem ainda ser utilizados algoritmos de criptografia, como o AES (*Advanced Encryption Standard*) para a cifragem de dados críticos (informações pessoais do utilizador, senhas, chaves de criptografia, etc.).

3. Utilização de um proxy reverso no acesso às APIs

   Quando se opta por uma abordagem de implementação de microsserviços, as APIs são divididas em múltiplos componentes autónomos, cada qual com o seu próprio endereço de IP. Esta abordagem apresenta vários benefícios já discutidos anteriormente. No entanto, também coloca alguns desafios. Ao designar-se um

endereço de IP a cada uma das APIs, os componentes pertencentes ao respetivo sistema IoT precisam de conhecer quais endereços de IP foram atribuídos às APIs. Por si só, este facto aumenta a complexidade de implementação do sistema, uma vez que exige que esta informação seja registada em cada um dos componentes do sistema. A situação agrava-se no caso de ser necessário alterar os endereços de IP das APIs, visto que nessa hipotética situação, será necessário atualizar os endereços de IP armazenados em cada componente do sistema. A implementação de um proxy reverso pode simplificar a manutenção do sistema, ficando este responsável por encaminhar as solicitações exteriores para a correspondente API. Assim, cada componente do sistema, apenas necessitará de guardar um único endereço de IP, o do proxy reverso. Além disso, esta abordagem é vantajosa em termos de segurança, uma vez que o cliente apenas terá acesso direto ao proxy reverso, mantendo os restantes componentes menos expostos. Um exemplo de um proxy reverso que pode ser utilizado nesta situação é o Nginx.

Existem outras melhorias que poderiam aprimorar a experiência do utilizador, tais como a configuração de alertas personalizados. Este tipo de alertas poderia ser configurado pelo utilizador para, por exemplo, emitir uma notificação sempre que uma porta ou janela é aberta. Além desta, seria benéfico incluir na interface do utilizador uma indicação dos dispositivos online, bem como a capacidade de verificar o histórico de mensagens trocadas entre o utilizador e os seus dispositivos. Por fim, seria aliciante implementar um ambiente de programação em blocos na IU, de modo a facilitar aos utilizadores a programação dos seus dispositivos.

Referências Bibliográficas

[1] M. Castells, «The impact of the internet on society: A Global Perspective», *Open Mind*, p. 25, 2014.

[2] P. Turkama, «Building Inclusive Digital Societies Through the Use of Open Source Technologies», *Lect. Notes Inf. Syst. Organ.*, vol. 30, pp. 35–50, 2019, doi: 10.1007/978-3-030-10737-6_3.

[3] R. Khan, S. U. Khan, R. Zaheer, e S. Khan, «Future internet: The internet of things architecture, possible applications and key challenges», *Proc. - 10th Int. Conf. Front. Inf. Technol. FIT 2012*, pp. 257–260, 2012, doi: 10.1109/FIT.2012.53.

[4] K. K. Goyal, A. Garg, A. Rastogi, e S. Singhal, «A Literature Survey on Internet of Things (IoT)», *Int. J. Adv. Manuf. Technol.*, vol. 9, n. 6, pp. 3663–3668, Dez. 2018.

[5] S. Madakam, R. Ramaswamy, e S. Tripathi, «Internet of Things (IoT): A Literature Review», *J. Comput. Commun.*, vol. 03, n. 05, pp. 164–173, 2015, doi: 10.4236/jcc.2015.35021.

[6] K. Asthon, «That ' Internet of Things ' Thing», *RFiD J.*, p. 4986, 2009, [Em linha]. Disponível em: www.itrco.jp/libraries/RFIDjournal-That Internet of Things Thing.pdf

[7] M. Gunturi, H. D. Kotha, e M. Srinivasa Reddy, «Overview of the Internet of things», *Int. Telecommun. Union*, vol. 10, n. 9, pp. 659–665, 2018.

[8] J. Gubbi, R. Buyya, S. Marusic, e M. Palaniswami, «Internet of Things (IoT): A vision, architectural elements, and future directions», *Futur. Gener. Comput. Syst.*, vol. 29, n. 7, pp. 1645–1660, Set. 2013, doi: 10.1016/j.future.2013.01.010.

[9] D. Evans, «The Internet of Things: How the Next Evolution of the Internet Is Changing Everything», *CISCO White Paper*, n. April. CISCO, 2011.

[10] K. L. Lueth, «State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time», *IOT Analytics Research*, 2020. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/

(acedido 15 de Julho de 2023).

[11] S. Sinha, «State of IoT 2023: Number of connected IoT devices growing 16% to 16.7 billion globally», *IoT Analytics Research*, 2023. https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ (acedido 15 de Julho de 2023).

[12] J. Guth *et al.*, «A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences», 2018, pp. 81–101. doi: 10.1007/978-981-10-5861-5_4.

[13] Mordorintelligence, «Global Smart Homes Market - Growth, Trends, Covid-19 Impact, And Forecasts (2021 - 2026)», *Mordorintelligence*, 2021. https://www.mordorintelligence.com/industry-reports/global-smart-homes-market-industry (acedido 15 de Julho de 2023).

[14] Mordorintelligence, «Smart Cities Market - Growth, Trends, Covid-19 Impact, And Forecasts (2021 - 2026)», *Mordorintelligence*, 2021. https://www.mordorintelligence.com/industry-reports/smart-cities-market (acedido 15 de Julho de 2023).

[15] B. Vogel, Y. Dong, B. Emruli, P. Davidsson, e R. Spalazzese, «What is an open IoT platform? Insights from a systematic mapping study», *Future Internet*, vol. 12, n. 4. 2020. doi: 10.3390/FI12040073.

[16] A. Tamboli, *Build Your Own IoT Platform*. Berkeley, CA: Apress, 2019. doi: 10.1007/978-1-4842-4498-2.

[17] P. Wegner, «IoT Platform Companies Landscape 2021/2022: Market consolidation has started», *IOT Analytics Research*, 2021. https://iot-analytics.com/iot-platform-companies-landscape/ (acedido 15 de Julho de 2023).

[18] J. Guth, U. Breitenbucher, M. Falkenthal, F. Leymann, e L. Reinfurt, «Comparison of IoT platform architectures: A field study based on a reference architecture», *2016 Cloudification Internet Things, CIoT 2016*, 2017, doi: 10.1109/CIOT.2016.7872918.

[19] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, e M. Ayyash, «Internet of

Things: A Survey on Enabling Technologies, Protocols, and Applications», *IEEE Commun. Surv. Tutorials*, vol. 17, n. 4, pp. 2347–2376, 2015, doi: 10.1109/COMST.2015.2444095.

[20] K. J. Singh e D. S. Kapoor, «Create Your Own Internet of Things: A survey of IoT platforms.», *IEEE Consum. Electron. Mag.*, vol. 6, n. 2, pp. 57–68, Abr. 2017, doi: 10.1109/MCE.2016.2640718.

[21] X. Wei, L. Lin, N. Meng, W. Tan, S.-C. Kong, e Kinshuk, «The effectiveness of partial pair programming on elementary school students' Computational Thinking skills and self-efficacy», *Comput. Educ.*, vol. 160, n. July 2020, p. 104023, Jan. 2021, doi: 10.1016/j.compedu.2020.104023.

[22] Y. V. Vazquez e S. E. Barbosa, «Recycling of mixed plastic waste from electrical and electronic equipment. Added value by compatibilization», *Waste Manag.*, vol. 53, pp. 196–203, Jul. 2016, doi: 10.1016/j.wasman.2016.04.022.

[23] Global E-Waste Monitor, «Global e-waste surging: up 21 per cent in 5 years», 2020. https://www.itu.int/en/mediacentre/Pages/pr10-2020-global-ewaste-monitor.aspx (acedido 15 de Julho de 2023).

[24] Parlamento Europeu, «Parlamento Europeu quer dar "direito à reparação" aos consumidores da UE», *Europarl.europa.eu*, 2020. https://www.europarl.europa.eu/news/pt/press-room/20201120IPR92118/parlamento-europeu-quer-dar-direito-a-reparacao-aos-consumidores-da-ue (acedido 15 de Julho de 2023).

[25] A. Bhatia, Z. Yusuf, D. Ritter, e N. Hunke, «Who Will Win the IoT Platform Wars?», *BCG Perspect.*, p. 6, 2017, [Em linha]. Disponível em: https://image-src.bcg.com/Images/BCG-Who-Will-Win-the-IoT-Platform-Wars-June-2017_2_tcm58-162424.pdf

[26] C. McClelland, «What is an IoT Platform?», *Leverege*, 2017. https://www.leverege.com/blogpost/what-is-an-iot-platform (acedido 15 de Julho de 2023).

[27] P. Pires, F. Delicato, T. V. Batista, T. Avila, E. Cavalcante, e M. Pitanga,

«Plataformas para a Internet das Coisas», em *Minicursos do XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, 2015, pp. 110–169.

[28] P. Sethi e S. R. Sarangi, «Internet of Things: Architectures, Protocols, and Applications», *J. Electr. Comput. Eng.*, vol. 2017, pp. 1–25, 2017, doi: 10.1155/2017/9324035.

[29] N. M. Kumar e P. K. Mallick, «The Internet of Things: Insights into the building blocks, component interactions, and architecture layers», *Procedia Comput. Sci.*, vol. 132, pp. 109–117, 2018, doi: 10.1016/j.procs.2018.05.170.

[30] M. Burhan, R. Rehman, B. Khan, e B.-S. Kim, «IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey», *Sensors*, vol. 18, n. 9, p. 2796, Ago. 2018, doi: 10.3390/s18092796.

[31] K. Dickerson, R. García-Castro, P. Kostelnik, e M. Paralič, «Standards for the IoT», em *IoT Platforms, Use Cases, Privacy, and Business Models*, Cham: Springer International Publishing, 2021, pp. 125–147. doi: 10.1007/978-3-030-45316-9_6.

[32] A. Mynzhasova *et al.*, «Drivers, standards and platforms for the IoT: Towards a digital VICINITY», em *2017 Intelligent Systems Conference (IntelliSys)*, IEEE, Set. 2017, pp. 170–176. doi: 10.1109/IntelliSys.2017.8324287.

[33] M. Pasha e S. M. W. Shah, «Framework for E-Health Systems in IoT-Based Environments», *Wirel. Commun. Mob. Comput.*, vol. 2018, pp. 1–11, Jun. 2018, doi: 10.1155/2018/6183732.

[34] T. Kramp, R. van Kranenburg, e S. Lange, *Enabling Things to Talk*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-40403-0.

[35] P. Fremantle, «A Reference Architecture for The Internet of Things», 2015.

[36] European Commission, «Internet of Things Architecture - IoT-A», *cordis.europa.eu*, 2010. https://cordis.europa.eu/project/id/257521 (acedido 15 de Julho de 2023).

[37] M. Bauer *et al.*, «IoT Reference Architecture», em *Enabling Things to Talk*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 163–211. doi: 10.1007/978-3-642-40403-0_8.

[38] D. Díaz López *et al.*, «Developing Secure IoT Services: A Security-Oriented Review of IoT Platforms», *Symmetry (Basel).*, vol. 10, n. 12, p. 669, Nov. 2018, doi: 10.3390/sym10120669.

[39] AWS IoT Core, «AWS IoT Core Developer Guide», 2022.

[40] Microsoft, «Microsoft Azure IoT», 2022. https://docs.microsoft.com/pt-pt/azure/iot-fundamentals/

[41] Arduino, «Arduino Cloud IoT», *docs.arduino.cc*, 2022. https://docs.arduino.cc/cloud/iot-cloud (acedido 15 de Julho de 2023).

[42] M. Noura, M. Atiquzzaman, e M. Gaedke, «Interoperability in Internet of Things: Taxonomies and Open Challenges», *Mob. Networks Appl.*, vol. 24, n. 3, pp. 796–809, Jun. 2019, doi: 10.1007/s11036-018-1089-9.

[43] V. Seoane, C. Garcia-Rubio, F. Almenares, e C. Campo, «Performance evaluation of CoAP and MQTT with security support for IoT environments», *Comput. Networks*, vol. 197, n. July, p. 108338, Out. 2021, doi: 10.1016/j.comnet.2021.108338.

[44] S. R. Jan, F. Khan, F. Ullah, N. Azim, e M. Tahir, «Using Coap Protocol for Resource Observation in Iot», *Int. J. Emerg. Technol. Comput. Sci. Electron. ISSN 0976-1353 Vol. 21 Issue 2 – April 2016.*, vol. 21, n. 2, pp. 11–14, 2016.

[45] Z. Shelby, ARM, K. Hartke, e C. Bormann, «The Constrained Application Protocol (CoAP)», 2014.

[46] B. H. Çorak, F. Y. Okay, M. Güzel, Ş. Murt, e S. Ozdemir, «Comparative Analysis of IoT Communication Protocols», *2018 Int. Symp. Networks, Comput. Commun. ISNCC 2018*, 2018, doi: 10.1109/ISNCC.2018.8530963.

[47] S. Hamdani e H. Sbeyti, «A comparative study of COAP and MQTT communication protocols», *7th Int. Symp. Digit. Forensics Secur. ISDFS 2019*, pp. 1–5, 2019, doi: 10.1109/ISDFS.2019.8757486.

[48] D. Thangavel, X. Ma, A. Valera, H. X. Tan, e C. K. Y. Tan, «Performance evaluation of MQTT and CoAP via a common middleware», *IEEE ISSNIP 2014 - 2014 IEEE 9th Int. Conf. Intell. Sensors, Sens. Networks Inf. Process. Conf. Proc.*, n. November

2015, 2014, doi: 10.1109/ISSNIP.2014.6827678.

[49] A. Thantharate, C. Beard, e P. Kankariya, «CoAP and MQTT Based Models to Deliver Software and Security Updates to IoT Devices over the Air», em *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, Jul. 2019, pp. 1065–1070. doi: 10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00183.

[50] A. Burange, H. Misalkar, e U. Nikam, *Security in MQTT and CoAP Protocols of IOT's application layer*, vol. 839. Springer Singapore, 2019. doi: 10.1007/978-981-13-2372-0_24.

[51] A. M. B e M. Petri, *Future Access Enablers for Ubiquitous and Intelligent Infrastructures*, vol. 159. em Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 159. Cham: Springer International Publishing, 2015. doi: 10.1007/978-3-319-27072-2.

[52] A. Zourmand, A. L. Kun Hing, C. Wai Hung, e M. AbdulRehman, «Internet of Things (IoT) using LoRa technology», em *2019 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS)*, IEEE, Jun. 2019, pp. 324–330. doi: 10.1109/I2CACIS.2019.8825008.

[53] D. Zorbas, G. Z. Papadopoulos, P. Maille, N. Montavont, e C. Douligeris, «Improving LoRa Network Capacity Using Multiple Spreading Factor Configurations», em *2018 25th International Conference on Telecommunications (ICT)*, IEEE, Jun. 2018, pp. 516–520. doi: 10.1109/ICT.2018.8464901.

[54] Semtech Corporation, «SX1276/77/78/79», *Datasheet*, n. March. 2015.

[55] B. Yu, L. Yang, e C.-C. Chong, «Optimized Differential GFSK Demodulator», *IEEE Trans. Commun.*, vol. 59, n. 6, pp. 1497–1501, Jun. 2011, doi: 10.1109/TCOMM.2011.041111.100010A.

[56] Tmatlantic, «FSK - frequency shift keying», *Tmatlantic*. https://www.tmatlantic.com/encyclopedia/index.php?ELEMENT_ID=10422 (acedido 15 de Julho de 2023).

[57] Near Communications, «GFSK modulation», *Near Communications*. https://sites.google.com/site/nearcommunications/gfsk-modulation (acedido 5 de Novembro de 2022).

[58] A. Kumbhar, «Overview of ISM Bands and Software-Defined Radio Experimentation», *Wirel. Pers. Commun.*, vol. 97, n. 3, pp. 3743–3756, Dez. 2017, doi: 10.1007/s11277-017-4696-z.

[59] J. Burns, S. Kirtay, e P. Marks, «Future use of Licence Exempt Radio Spectrum», *UK Spectr. Policy Forum*, vol. 44, n. July, p. 72, 2015, [Em linha]. Disponível em: https://plumconsulting.co.uk/wpdm-package/plum-july-2015-future-use-of-licence-exempt-radio-spectrum-pdf/

[60] ANACOM, «Isenção de Licença de Estação», 2023.

[61] Thethingsnetwork, «Frequency Plans by Country», *thethingsnetwork.org*. https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/ (acedido 15 de Julho de 2023).

[62] European Conference of Postal and Telecommunications Administrations, «ERC Recommendation 70-03», n. February. pp. 1–92, 2022. [Em linha]. Disponível em: https://docdb.cept.org/download/3700

[63] Espressif Systems, «ESP32 Series Datasheet», *Espressif Systems*. pp. 1–68, 2022. [Em linha]. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[64] Espressif, «ESP32 Series - ESP Product Selector», *products.espressif.com/*, 2022. https://products.espressif.com/#/product-selector?names=&filter=%7B%22Products%22:[%22SoC%22],%22Series%22:[%22ESP32%22]%7D (acedido 15 de Julho de 2023).

[65] Electronupdate, «Espressif ESP32 Teardown», *Electronupdate*, 2018. http://electronupdate.blogspot.com/2018/08/espressif-esp32-teardown.html (acedido 15 de Julho de 2023).

[66] Espressif, «ESP Product Selector», *products.espressif.com/*, 2022.

https://products.espressif.com/#/product-selector?names=&filter=%7B%22Products%22%3A%5B%22Module%22%5D,%22Series%22%3A%5B%22ESP32%22,%22ESP32-S2%22,%22ESP32-C3%28including ESP8685%29%22,%22ESP32-C2%28including ESP8684%29%22,%22ESP32-S3%22%5D%7D (acedido 15 de Julho de 2023).

[67] Espressif, «Development Boards», *espressif.com*, 2023. https://www.espressif.com/en/products/devkits (acedido 15 de Julho de 2023).

[68] Arduino, «What is Arduino?», *arduino.cc*, 2018. https://www.arduino.cc/en/Guide/Introduction (acedido 15 de Julho de 2023).

[69] Espressif, «Official IoT Development Framework», *espressif.com*. https://www.espressif.com/en/products/sdks/esp-idf (acedido 15 de Julho de 2023).

[70] Espruino, «Espruino», *Espruino.com*. https://www.espruino.com/ (acedido 15 de Julho de 2023).

[71] MicroPython, «MicroPython», *micropython.org*. https://micropython.org/ (acedido 15 de Julho de 2023).

[72] P. R. Gomes *et al.*, «Development and assessment of an IoT system for monitoring air and soil quality in the agricultural sector», em *ECOS 2023 - 36th lnternational Conference on Efficiency, Cost, Optimization, Simulation and Environmental lmpact of Energy Systems ULPGC*, 2023, pp. 1321–1332.

[73] HC-12, «Hc-12 Wireless Rf Uart Communication Module V2.4 User Manual». p. 10, 2016. [Em linha]. Disponível em: www.hc01.com

[74] Semtech, «SEMTECH SX1276», *semtech.com*, 2023. https://www.semtech.com/products/wireless-rf/lora-connect/sx1276#inventory (acedido 15 de Julho de 2023).

[75] Semtech, «SEMTECH SX1278», *semtech.com*, 2023. https://www.semtech.com/products/wireless-rf/lora-connect/sx1278#inventory (acedido 15 de Julho de 2023).

[76] Espressif Systems, «ESP32 Pin List», *Espressif Systems*. 2016. [Em linha].

Disponível em: https://gzhls.at/blob/ldb/1/b/3/9/ce458f9742b549ee683216c160d28abea52e.pdf

[77] Espressif, «SPI Master Driver», *docs.espressif.com*, 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/spi_master.html (acedido 15 de Julho de 2023).

[78] Brian Krent, «Espressif ESP-WROOM-32 Wi-Fi & Bluetooth Module», *wikimedia.org*, 2017. https://commons.wikimedia.org/wiki/File:Espressif_ESP-WROOM-32_Wi-Fi_%26_Bluetooth_Module.jpg (acedido 15 de Julho de 2023).

[79] SparkFun, «SparkFun ESP32 Thing», *sparkfun.com*, 2023. https://www.sparkfun.com/products/13907 (acedido 15 de Julho de 2023).

[80] Espressif, «Partition Tables», *docs.espressif.com*, 2023. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/partition-tables.html (acedido 15 de Julho de 2023).

[81] B. Rojas, «Toward Next-Generation Access Networking Technologies in Industrial / Enterprise Internet of Things IoT Vision and Ecosystem», *IDC*, n. October, 2015.

Apêndice A – Fluxogramas das APIs

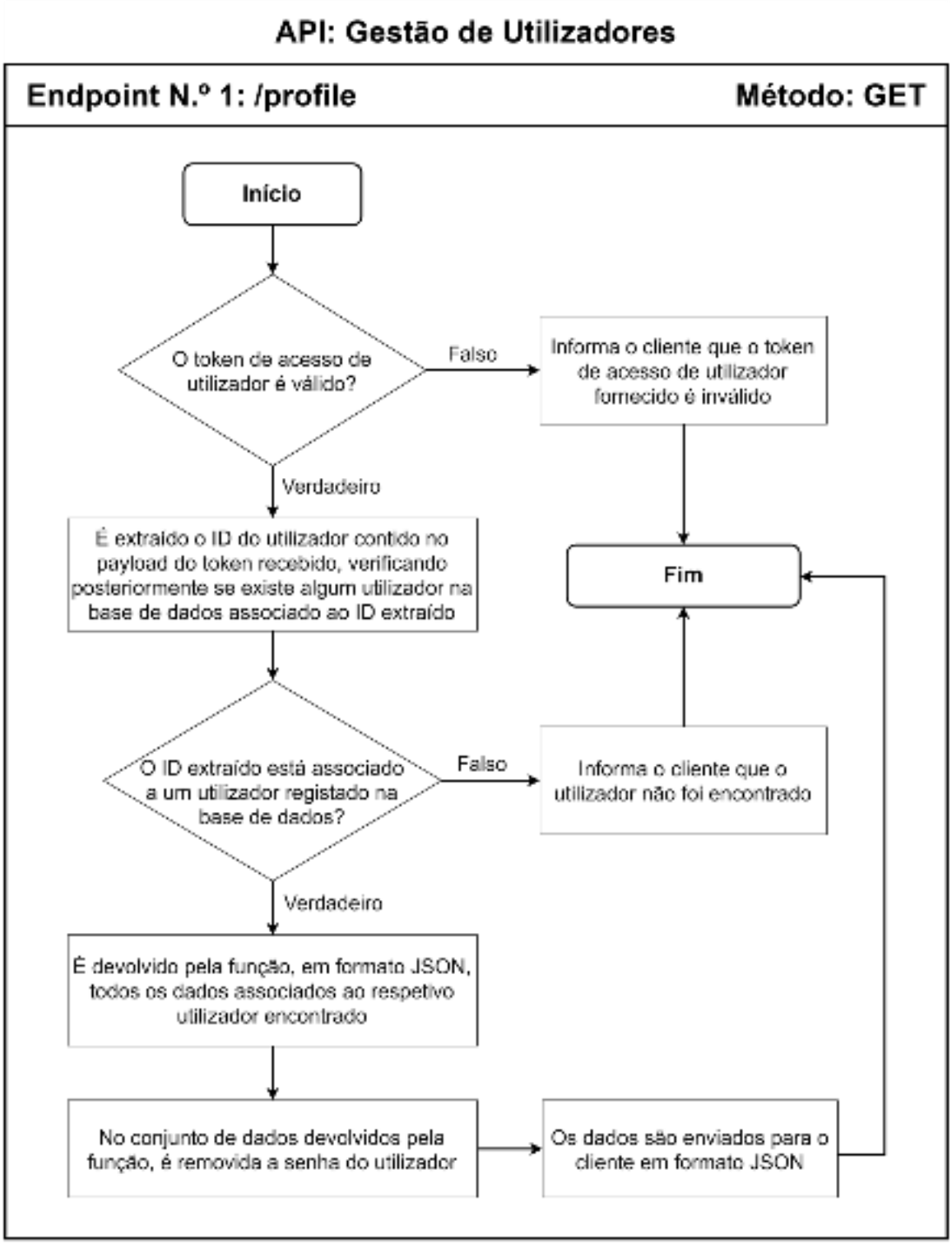


Figura 92 - Fluxograma do Endpoint N.º 1 da API de Gestão de Utilizadores

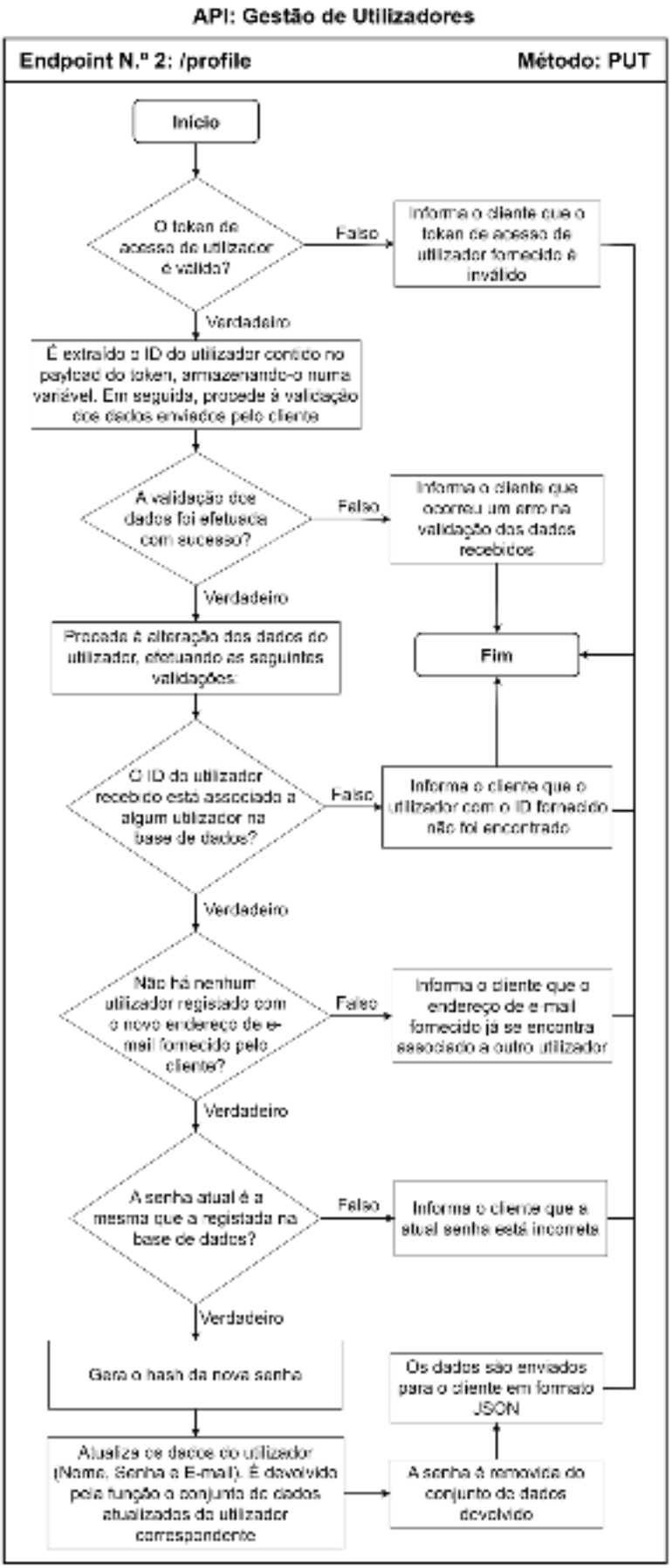


Figura 93 - Fluxograma do Endpoint N.º 2 da API de Gestão de Utilizadores

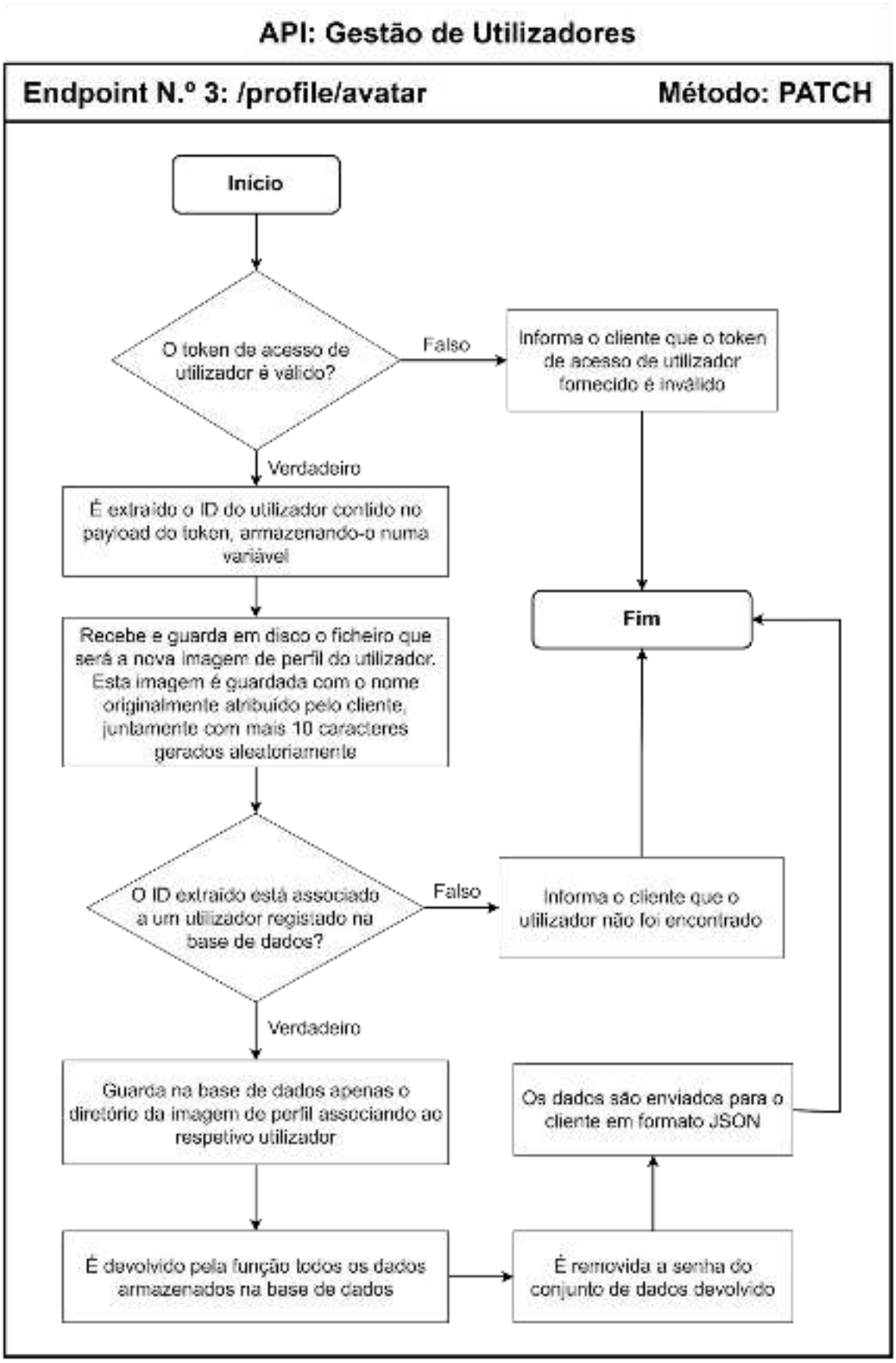


Figura 94 - Fluxograma do Endpoint N.º 3 da API de Gestão de Utilizadores

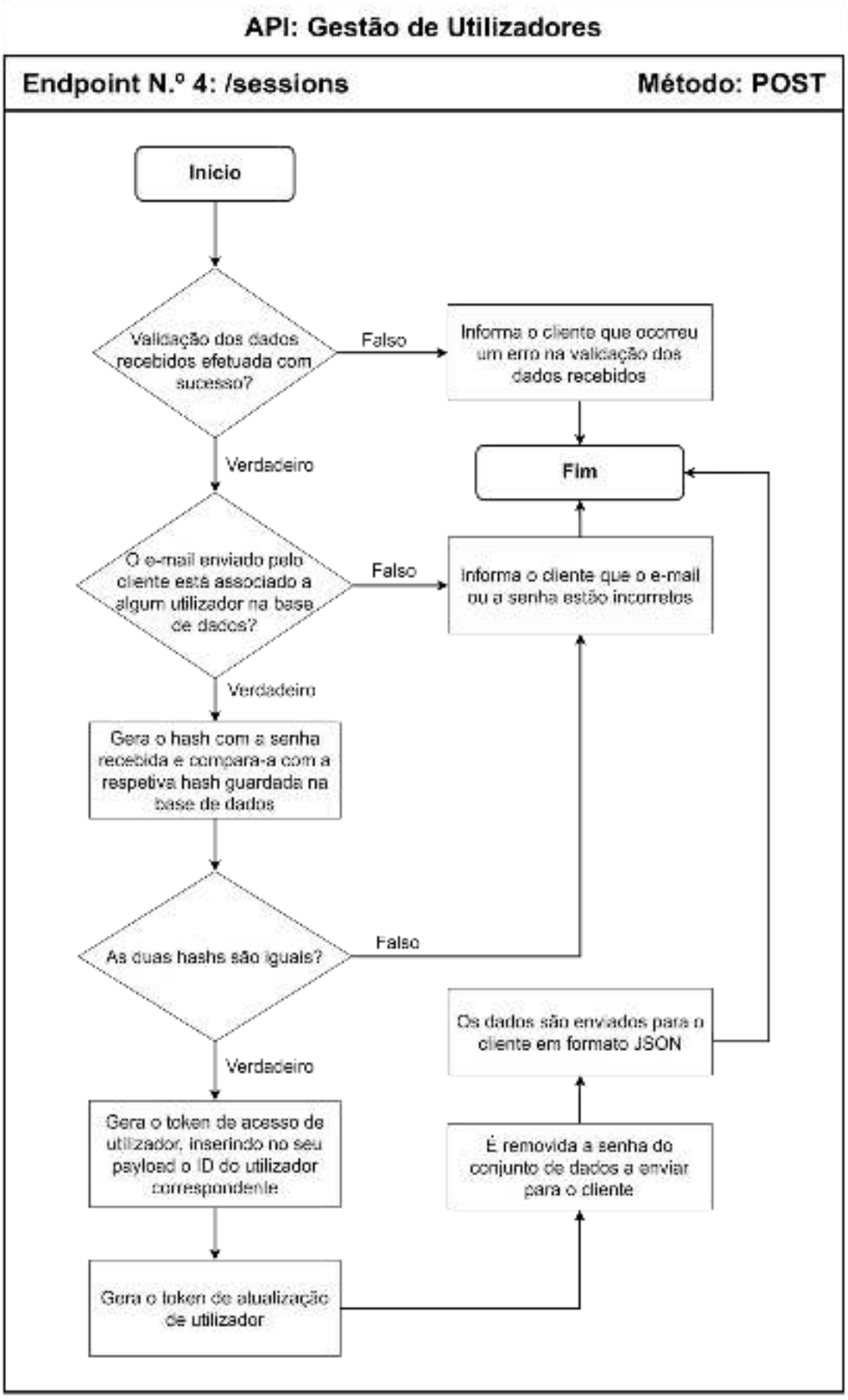


Figura 95 - Fluxograma do Endpoint N.º 4 da API de Gestão de Utilizadores

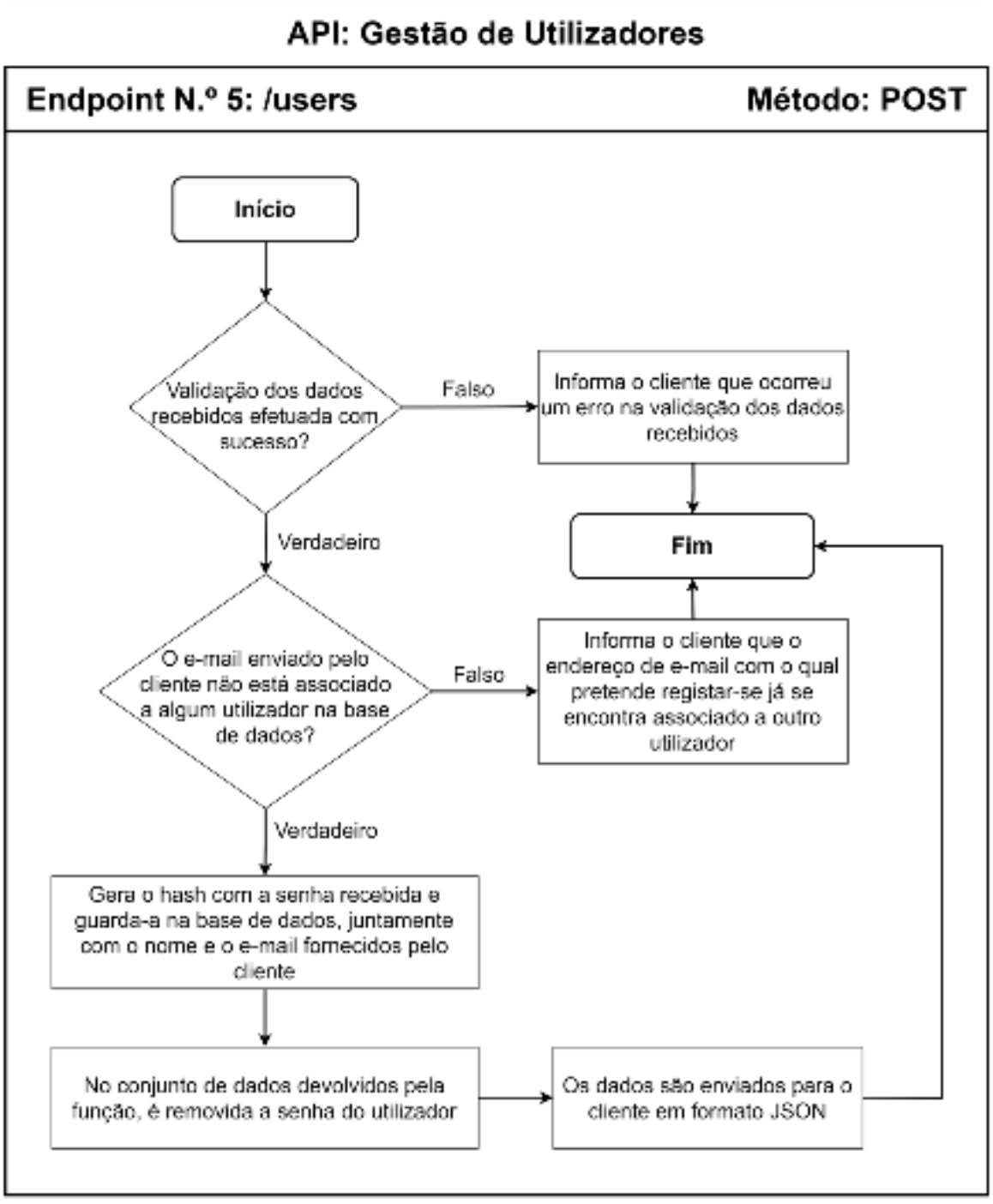


Figura 96 - Fluxograma do Endpoint N.º 5 da API de Gestão de Utilizadores

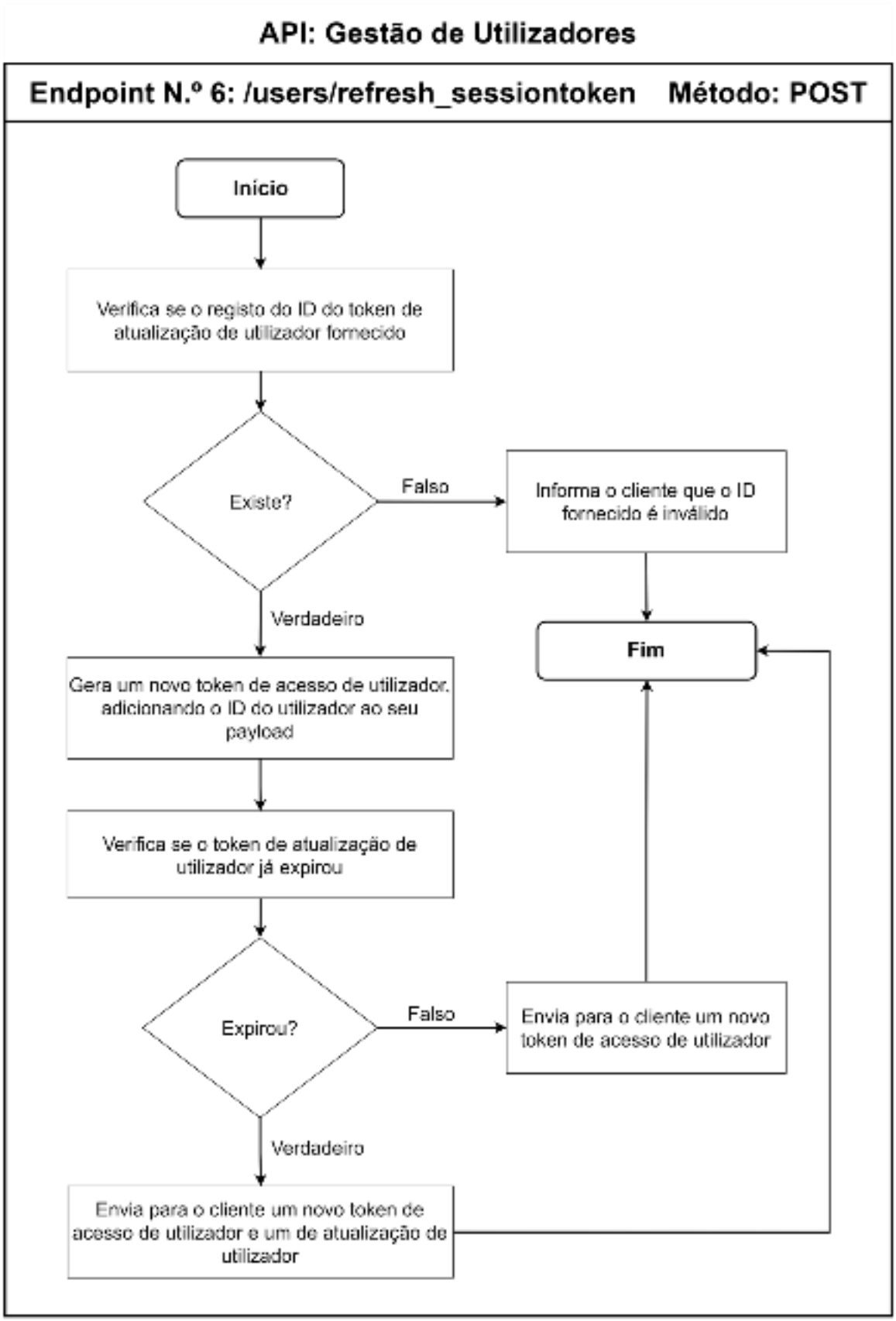


Figura 97 - Fluxograma do Endpoint N.º 6 da API de Gestão de Utilizadores

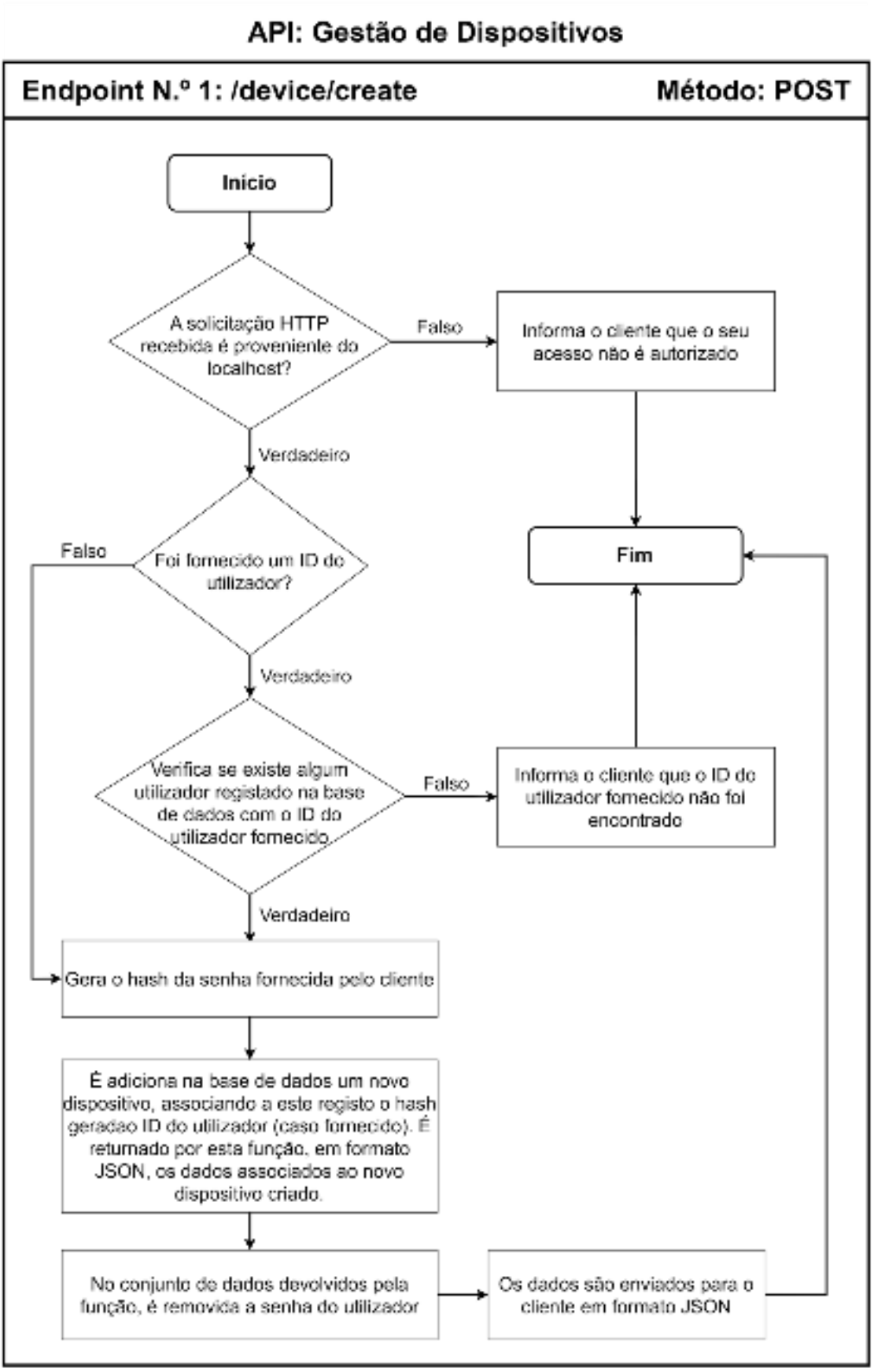


Figura 98 - Fluxograma do Endpoint N.º 1 da API de Gestão de Dispositivos

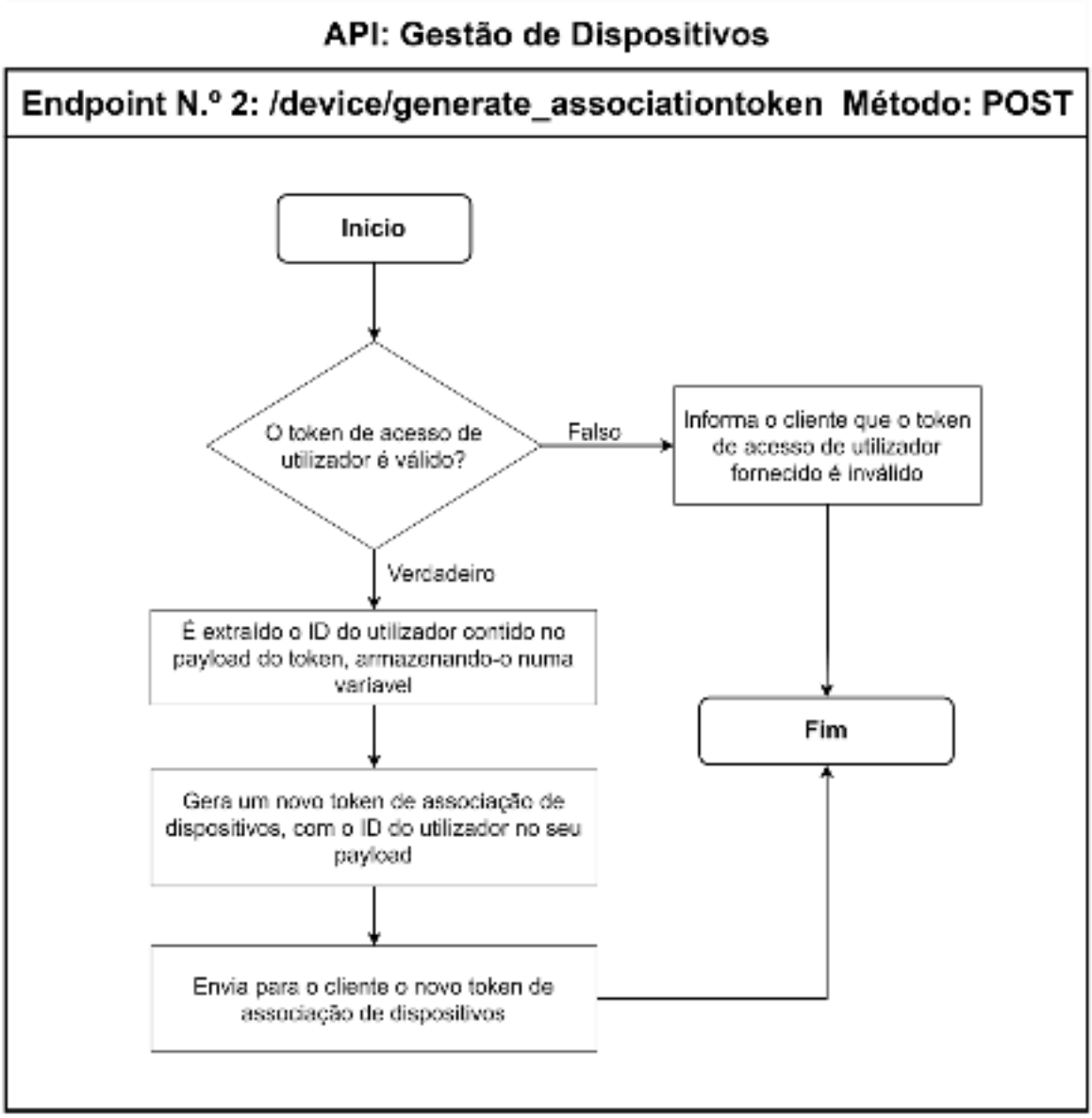


Figura 99 - Fluxograma do Endpoint N.º 2 da API de Gestão de Dispositivos

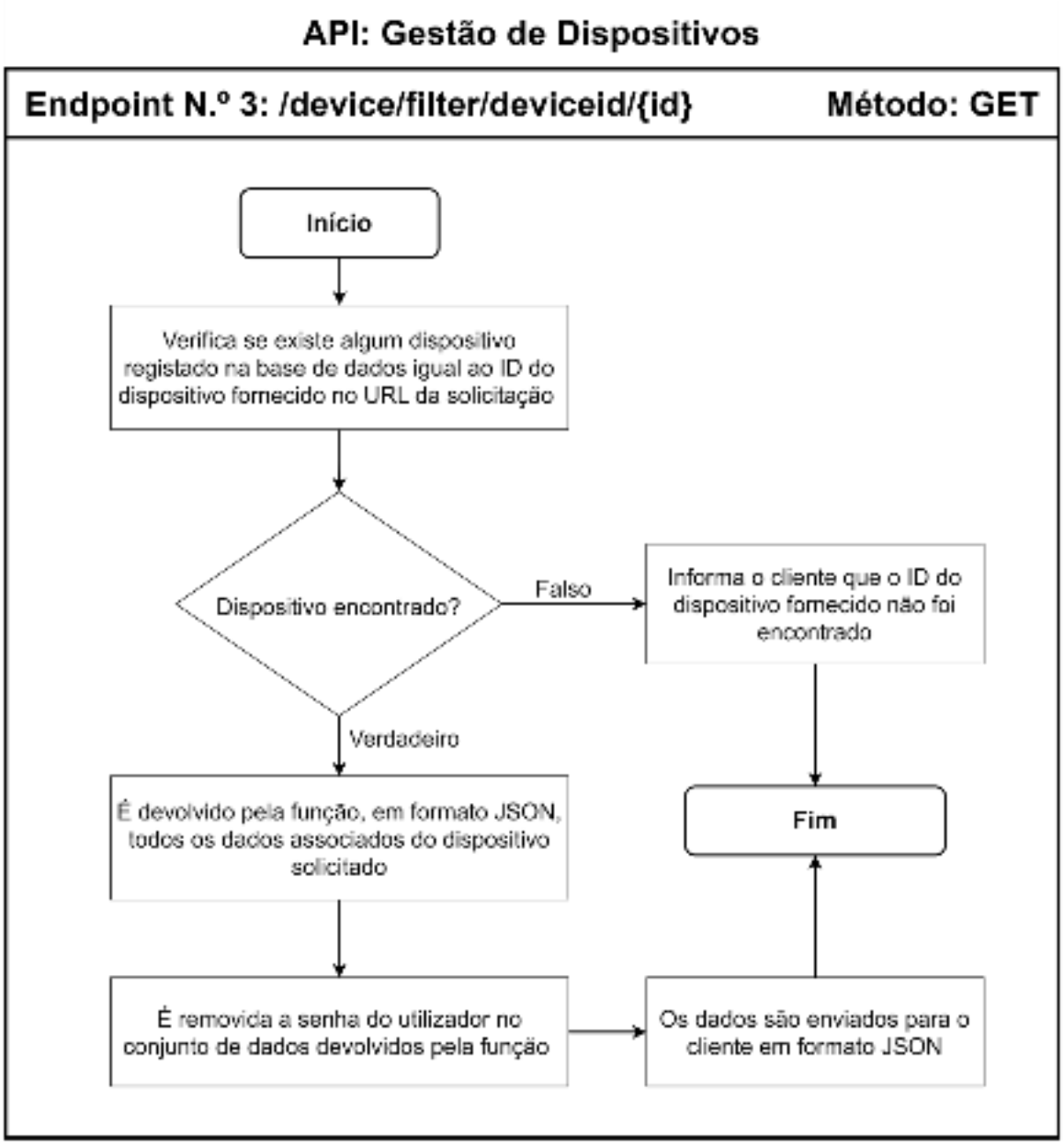


Figura 100 - Fluxograma do Endpoint N.º 3 da API de Gestão de Dispositivos

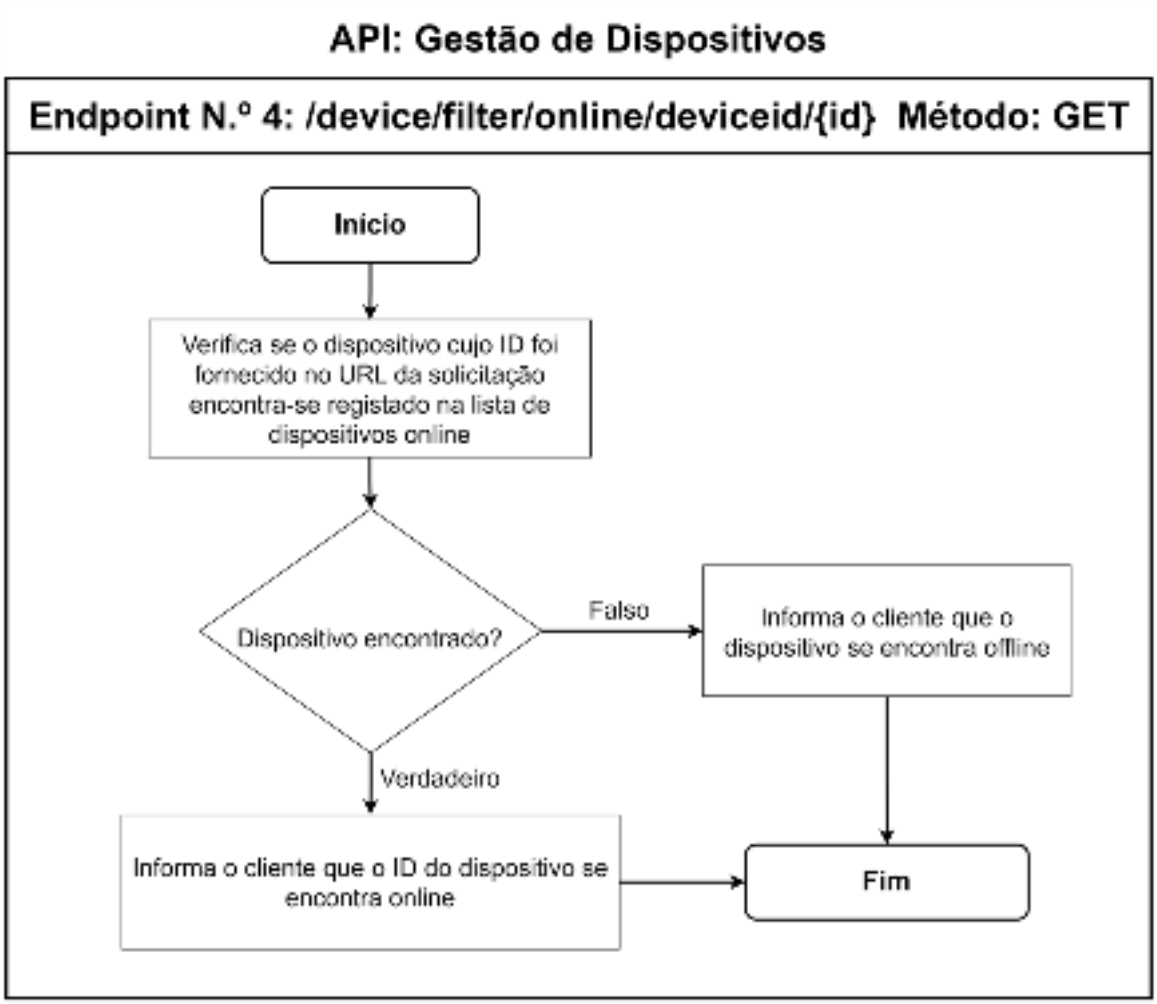


Figura 101 - Fluxograma do Endpoint N.º 4 da API de Gestão de Dispositivos

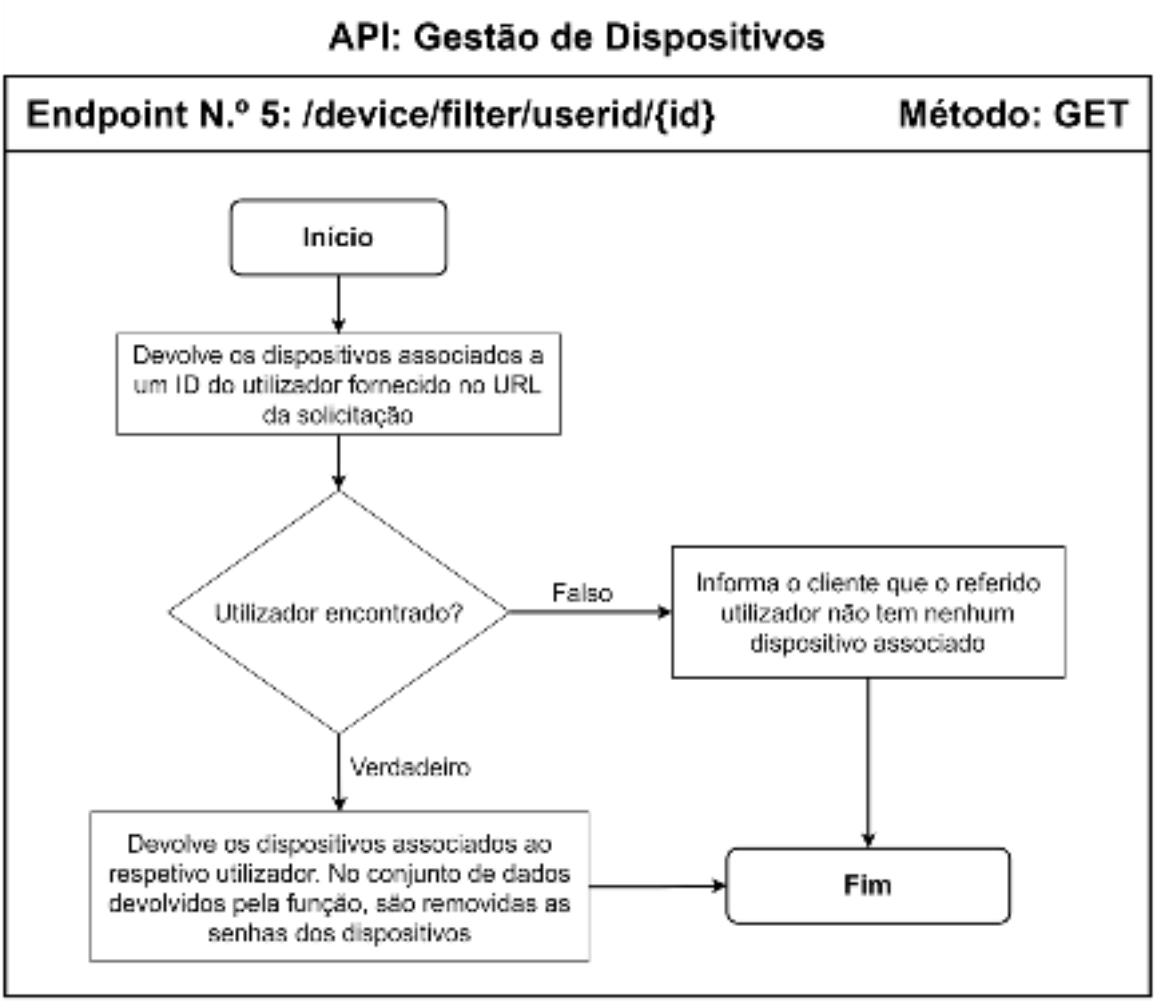


Figura 102 - Fluxograma do Endpoint N.º 5 da API de Gestão de Dispositivos

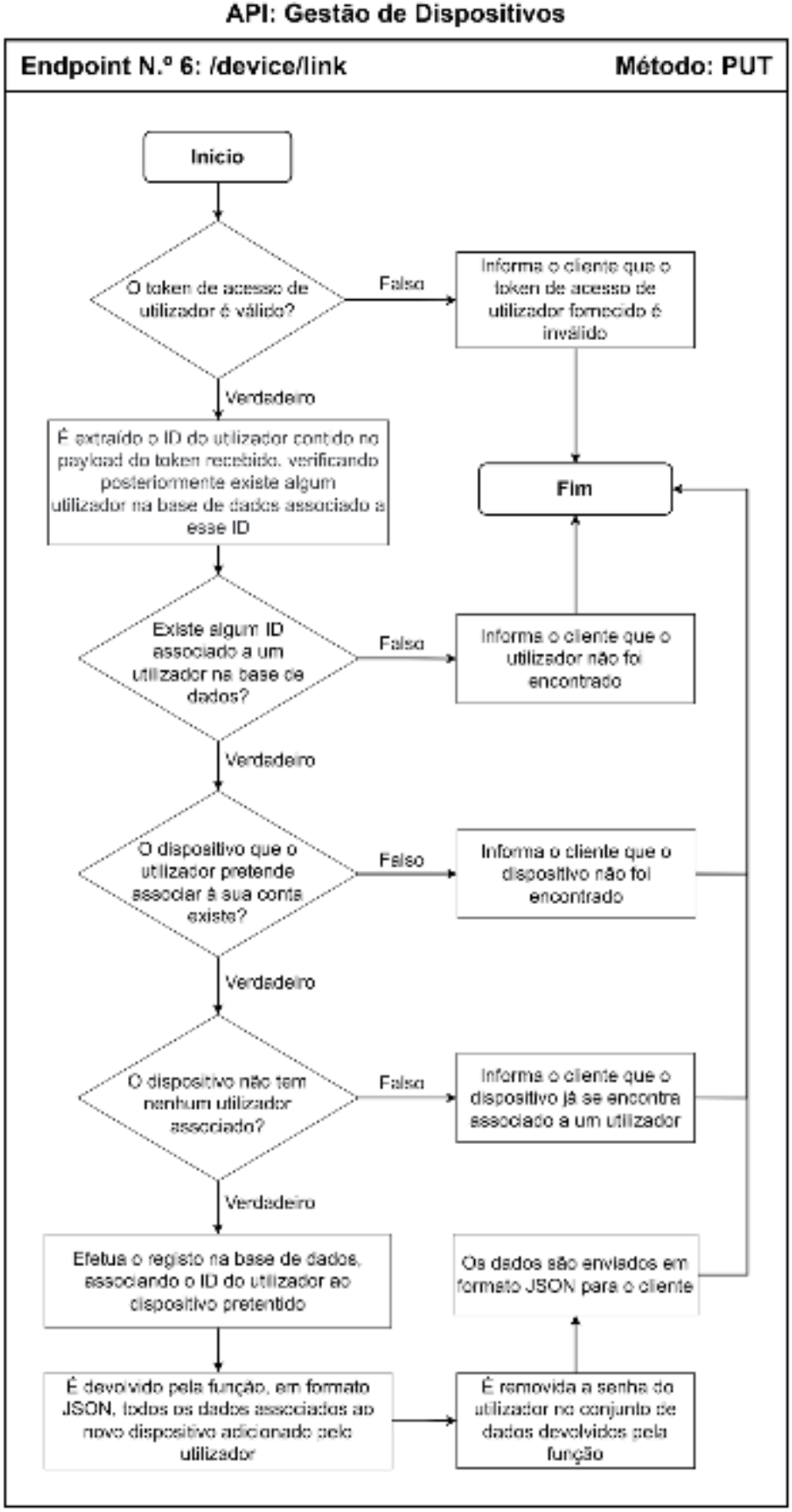


Figura 103 - Fluxograma do Endpoint N.º 6 da API de Gestão de Dispositivos

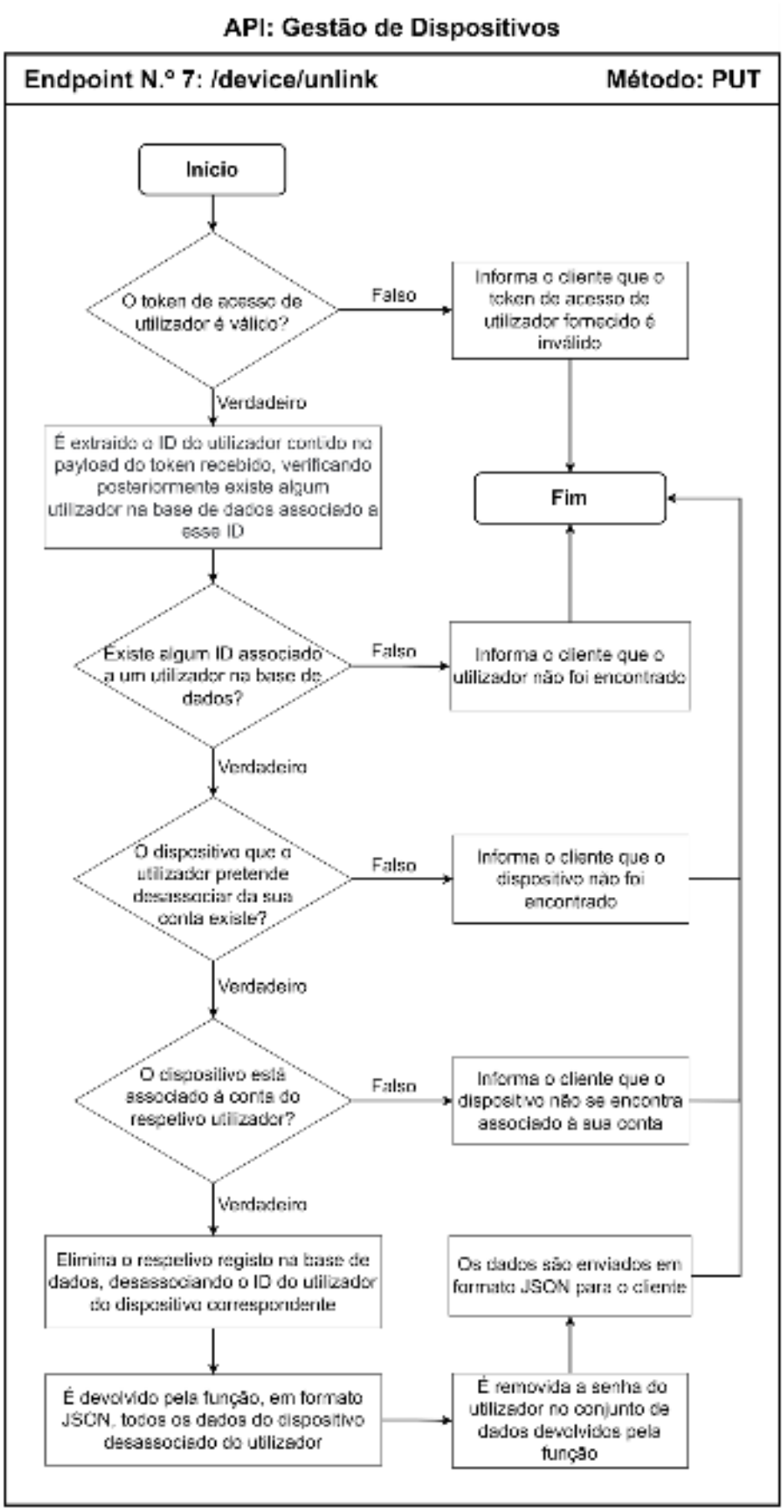


Figura 104 - Fluxograma do Endpoint N.º 7 da API de Gestão de Dispositivos

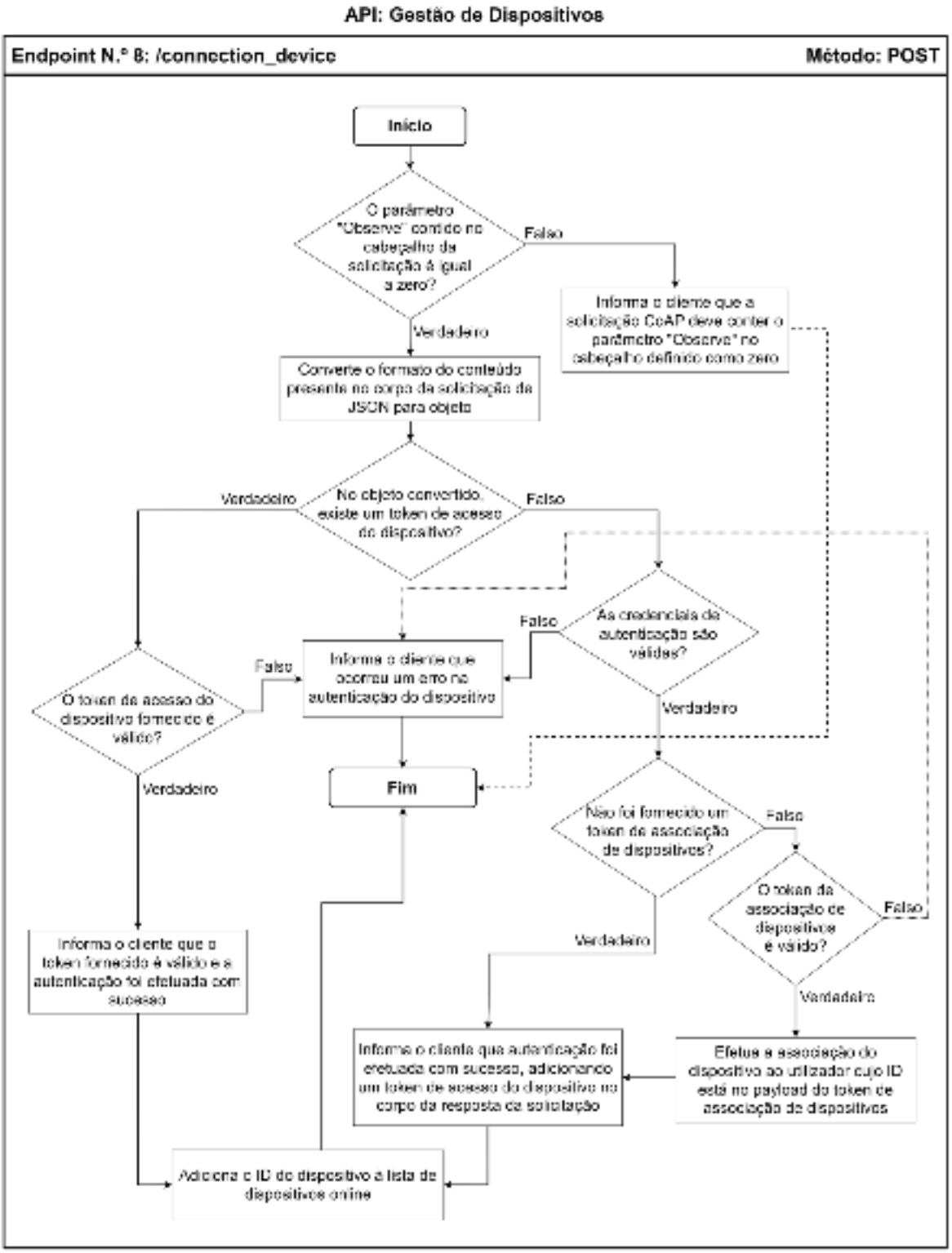


Figura 105 - Fluxograma do Endpoint N.º 8 da API de Gestão de Dispositivos

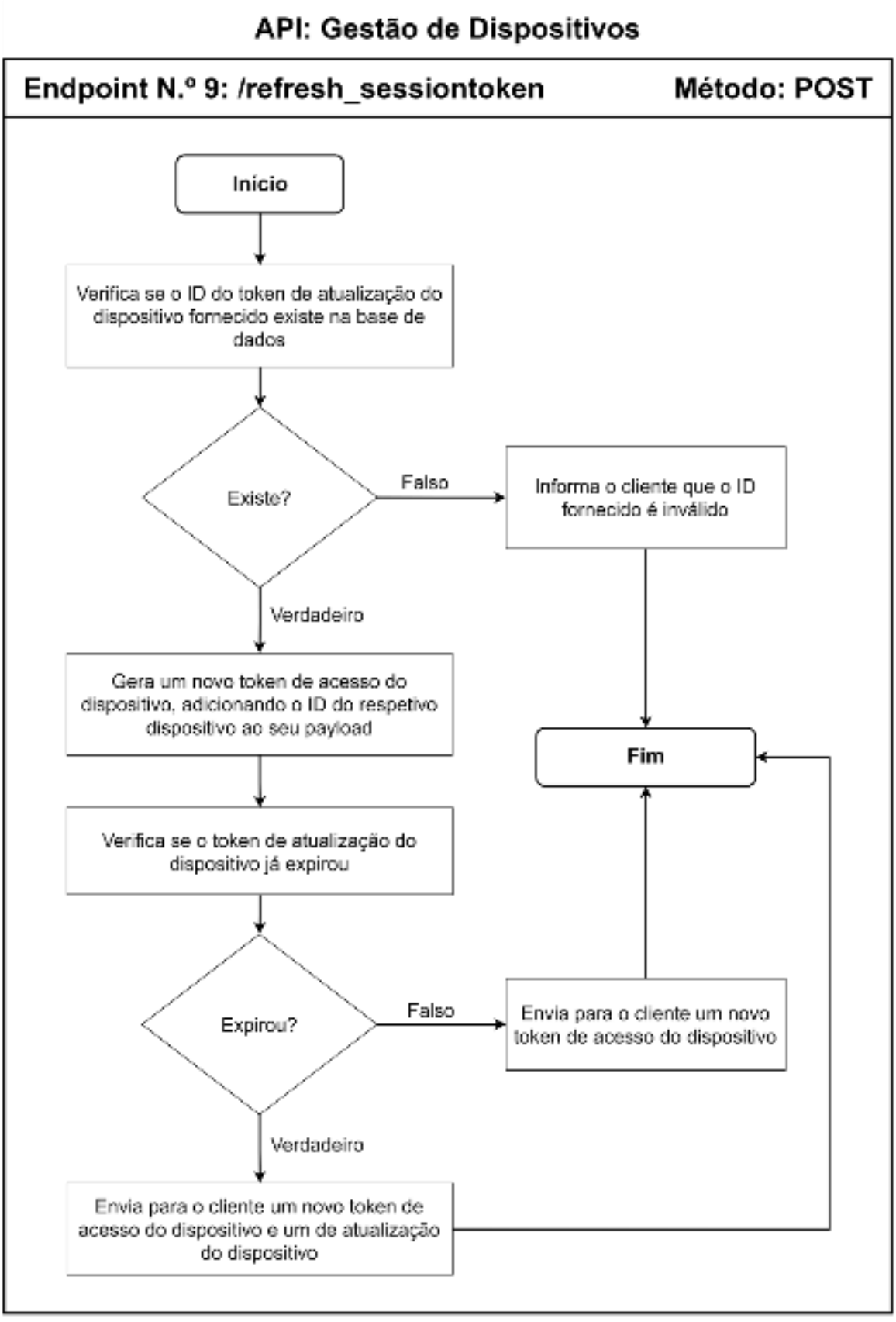


Figura 106 - Fluxograma do Endpoint N.º 9 da API de Gestão de Dispositivos

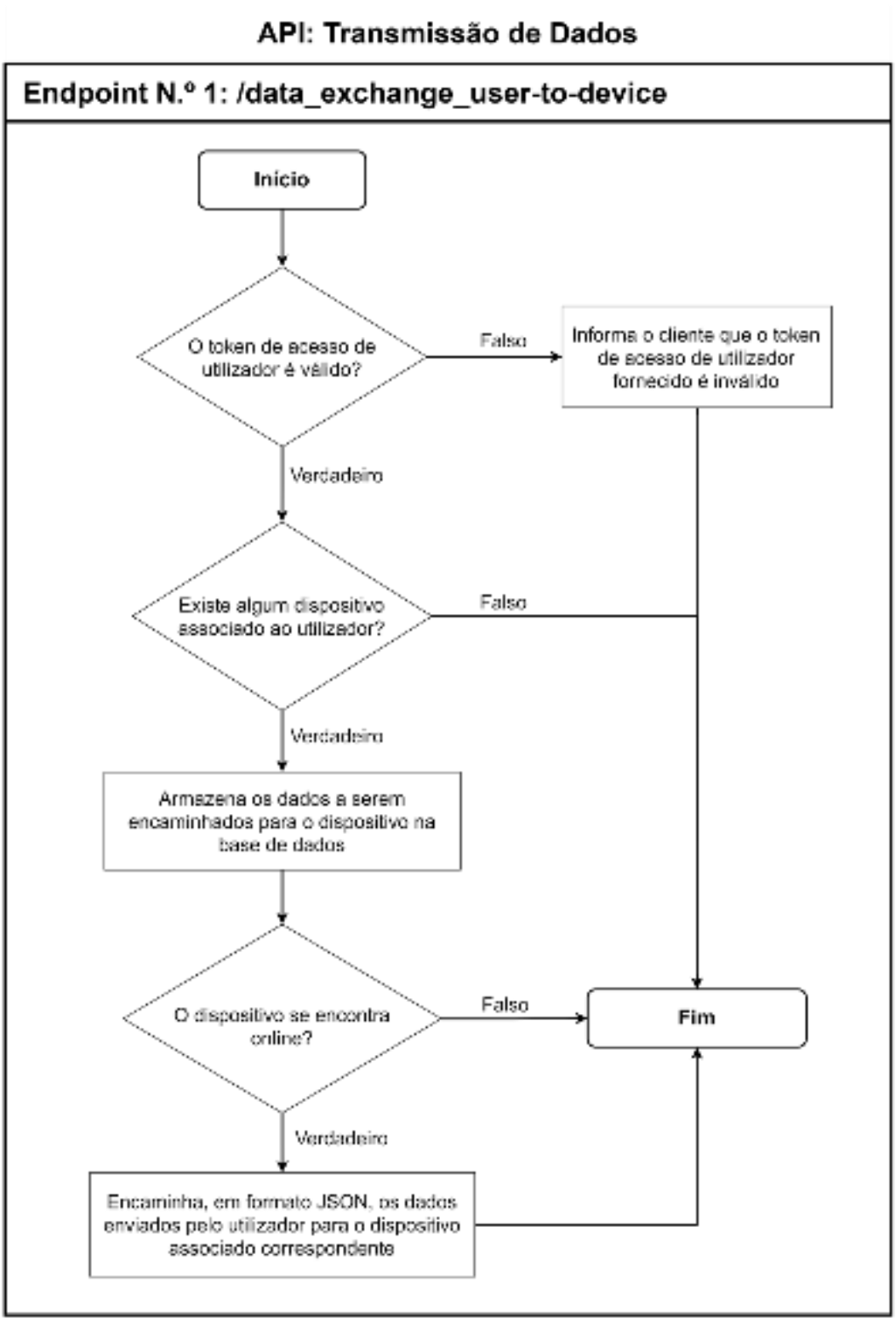


Figura 107 - Fluxograma do Endpoint N.º 1 da API de Transmissão de Dados

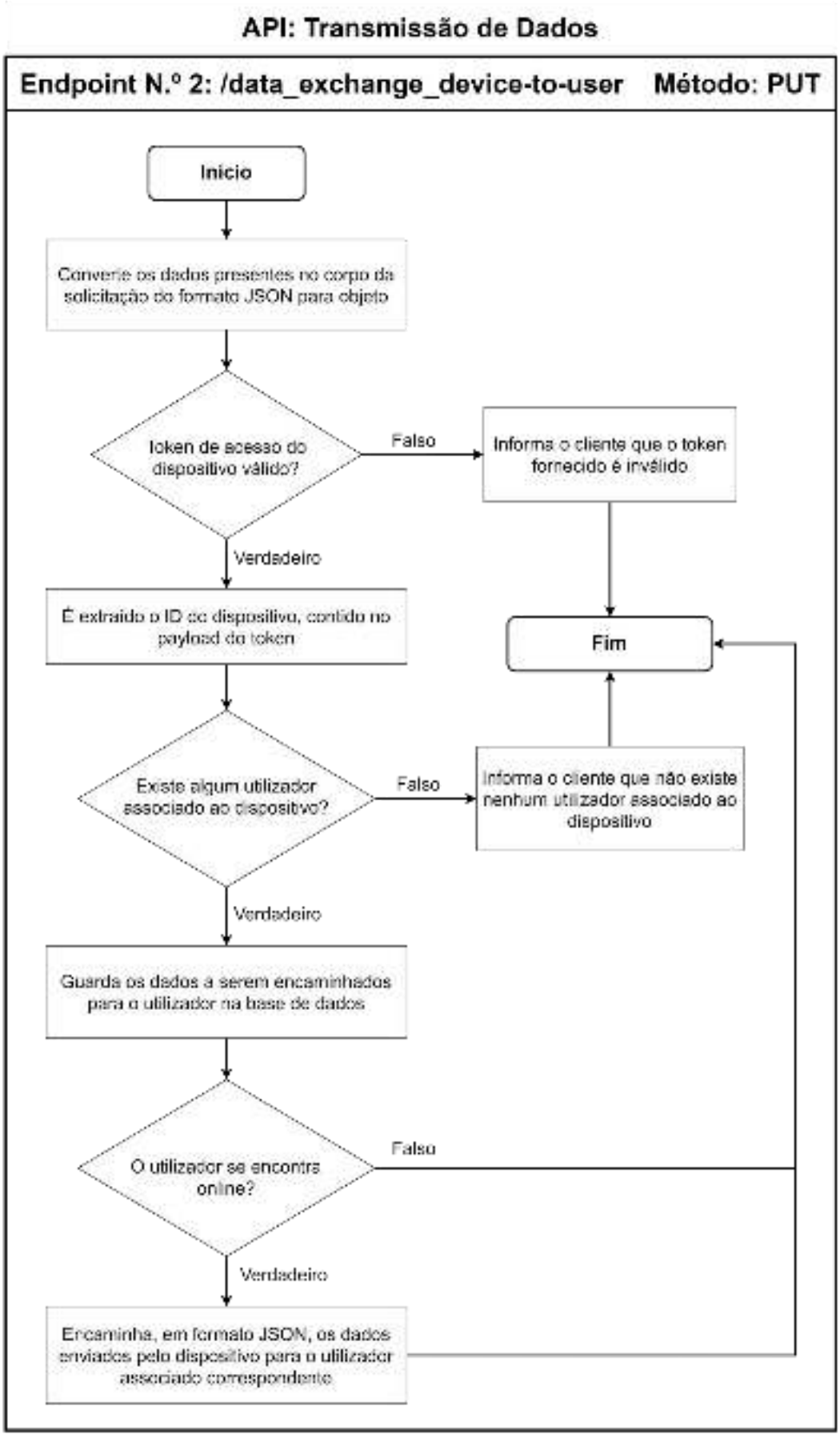


Figura 108 - Fluxograma do Endpoint N.º 2 da API de Transmissão de Dados

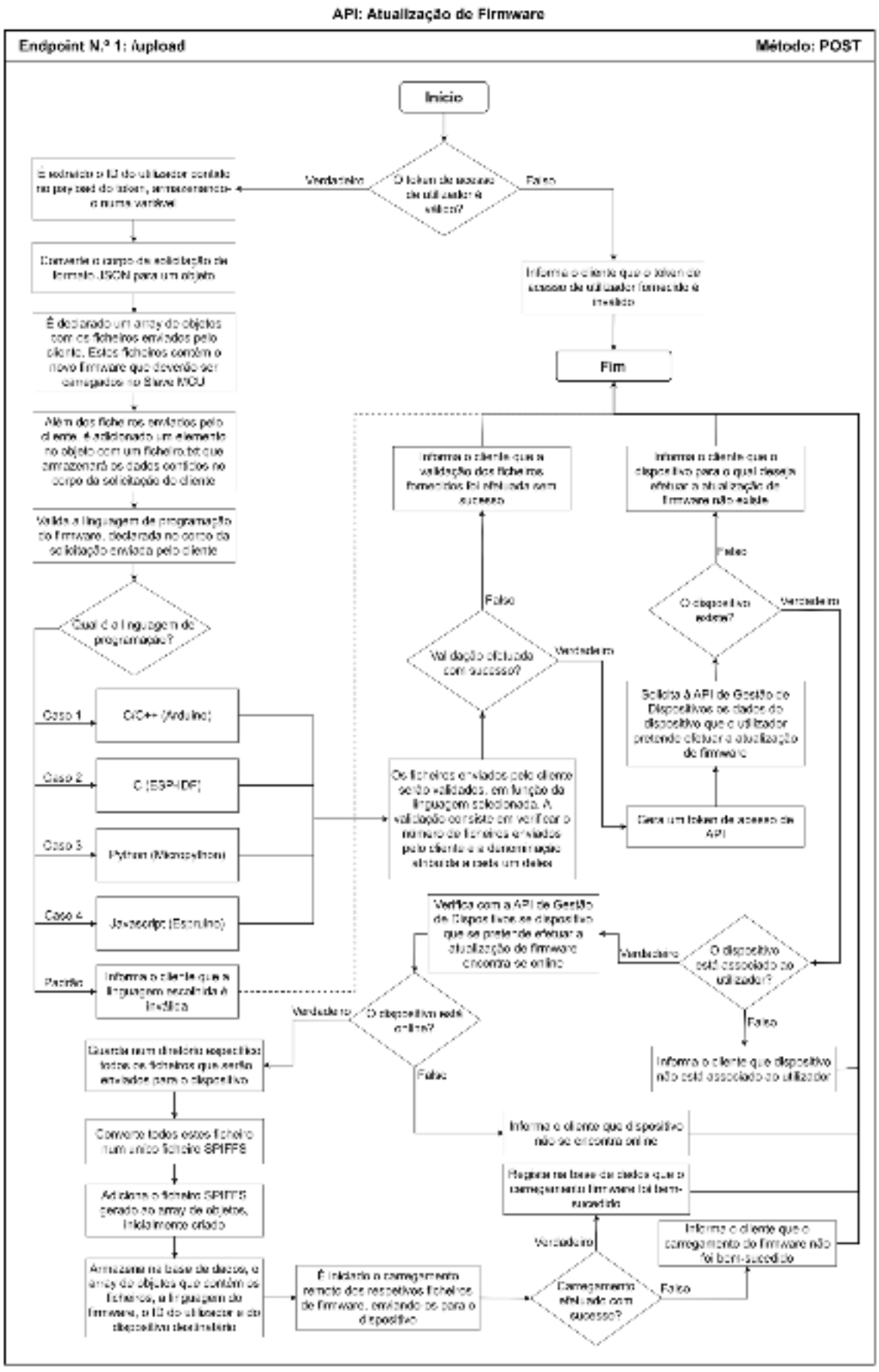


Figura 109 - Fluxograma do Endpoint N.º 1 da API de Atualização de Firmware

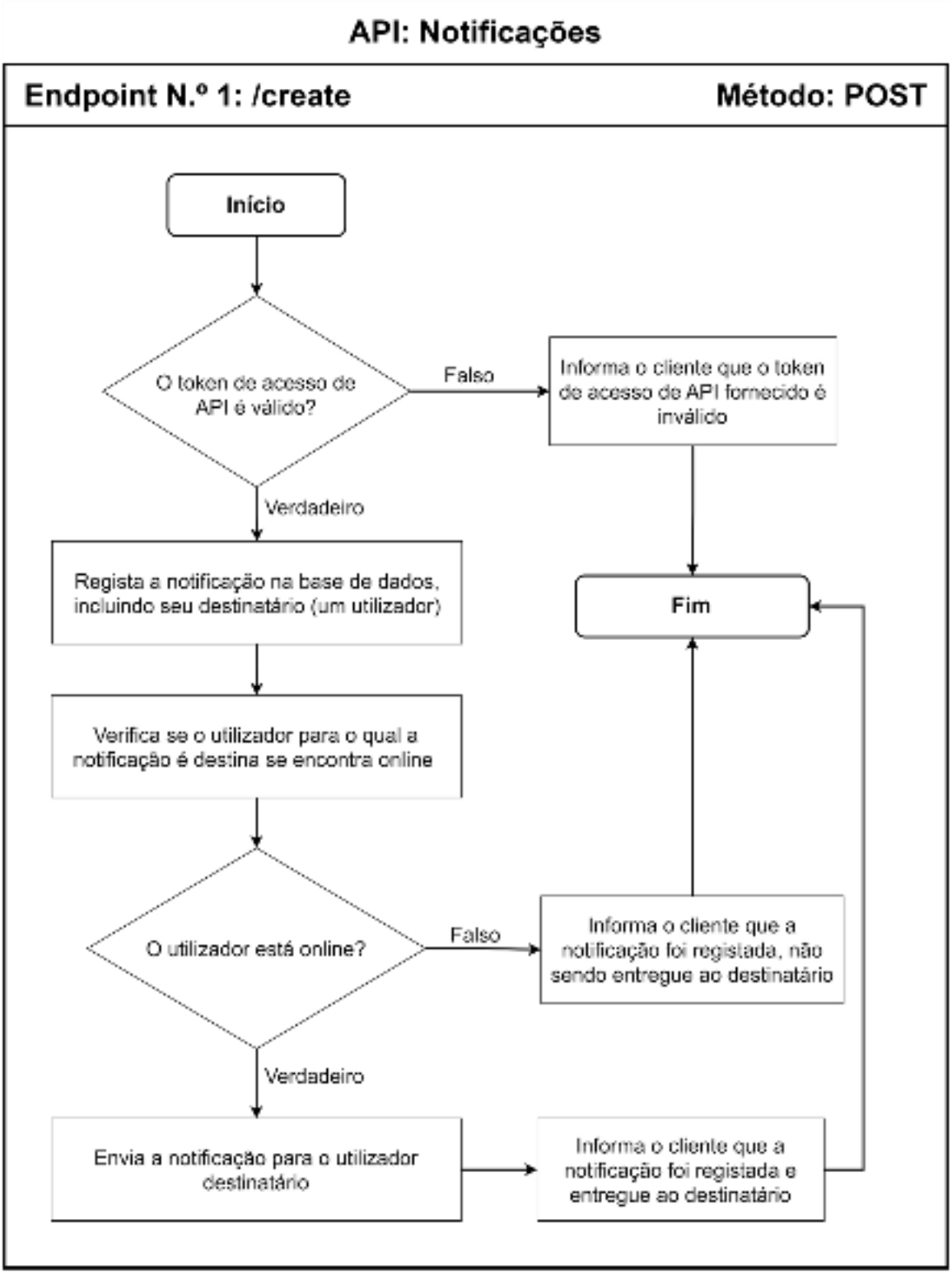


Figura 110 - Fluxograma do Endpoint N.º 1 da API de Notificações

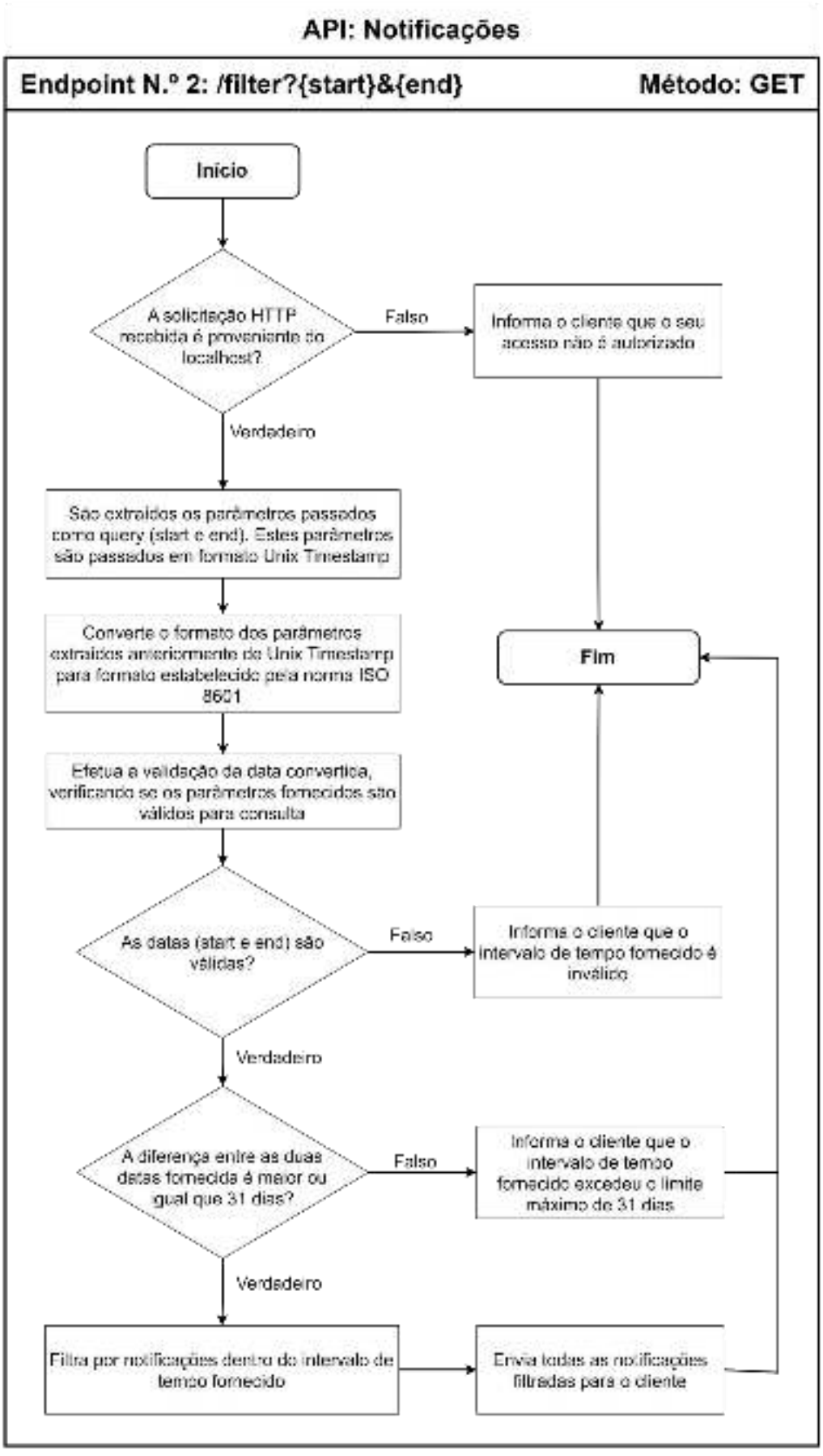


Figura 111 - Fluxograma do Endpoint N.º 2 da API de Notificações

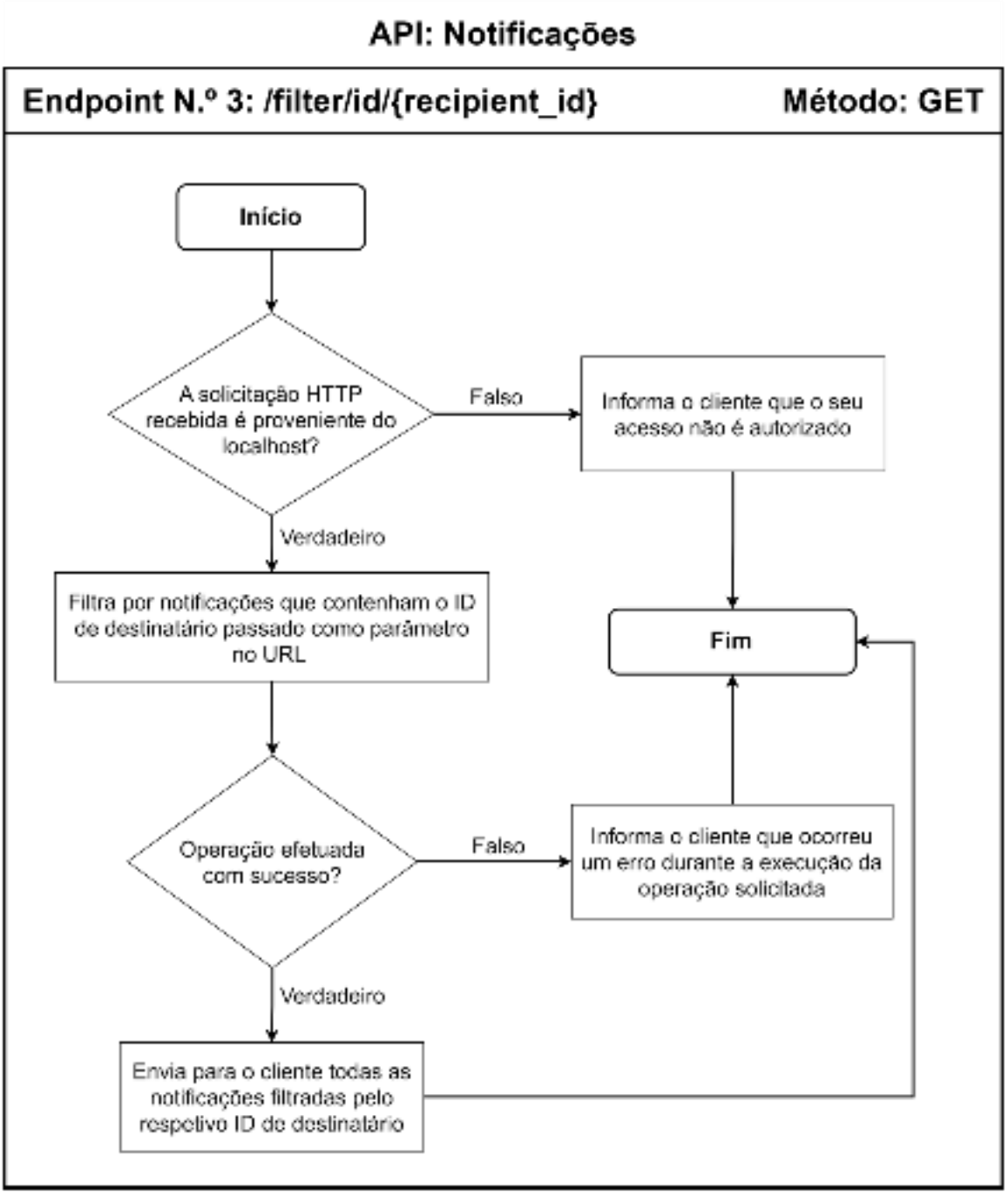


Figura 112 - Fluxograma do Endpoint N.º 3 da API de Notificações

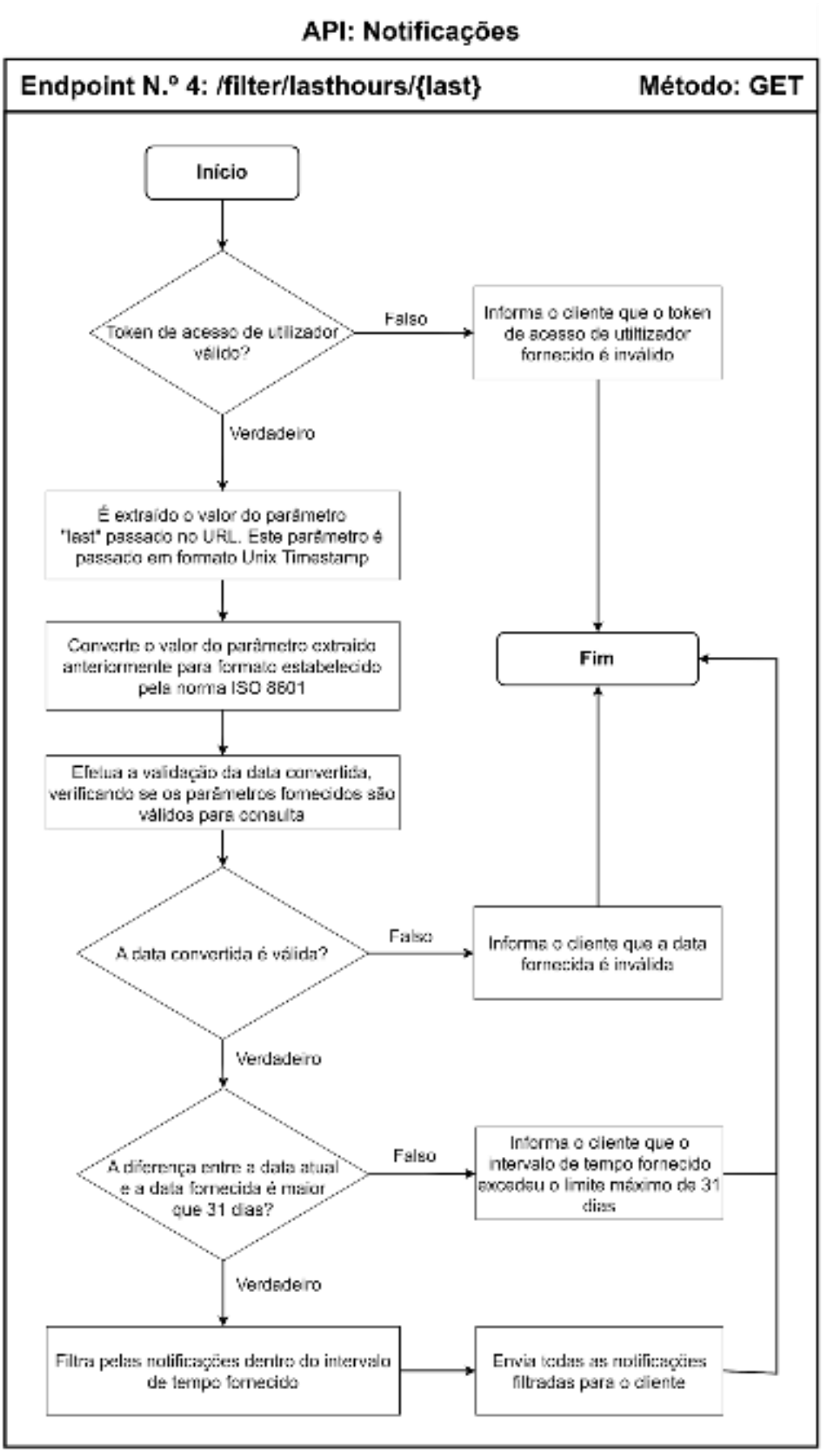


Figura 113 - Fluxograma do Endpoint N.º 4 da API de Notificações

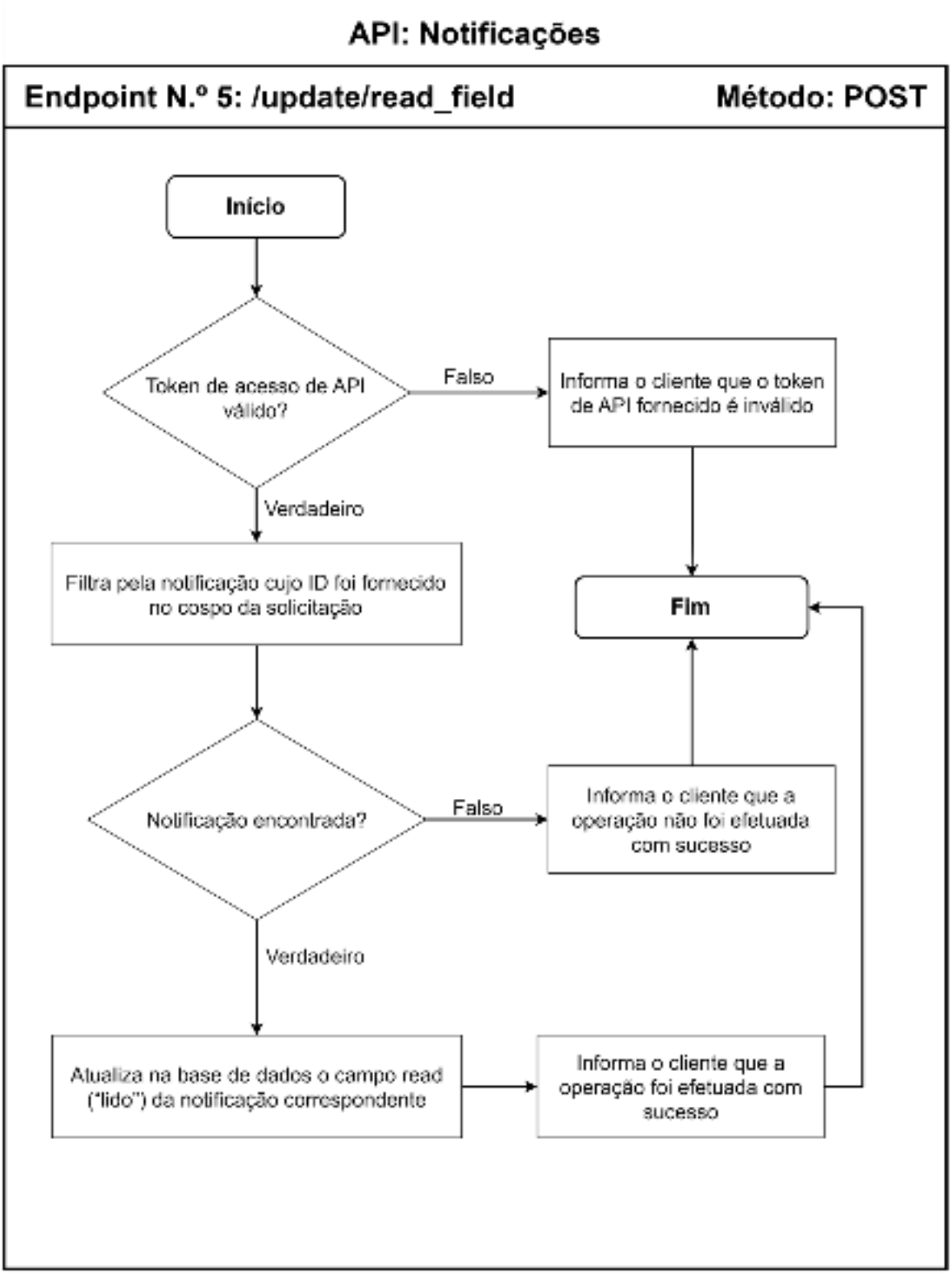


Figura 114 - Fluxograma do Endpoint N.º 5 da API de Notificações

Apêndice B – Fluxograma do Dispositivo

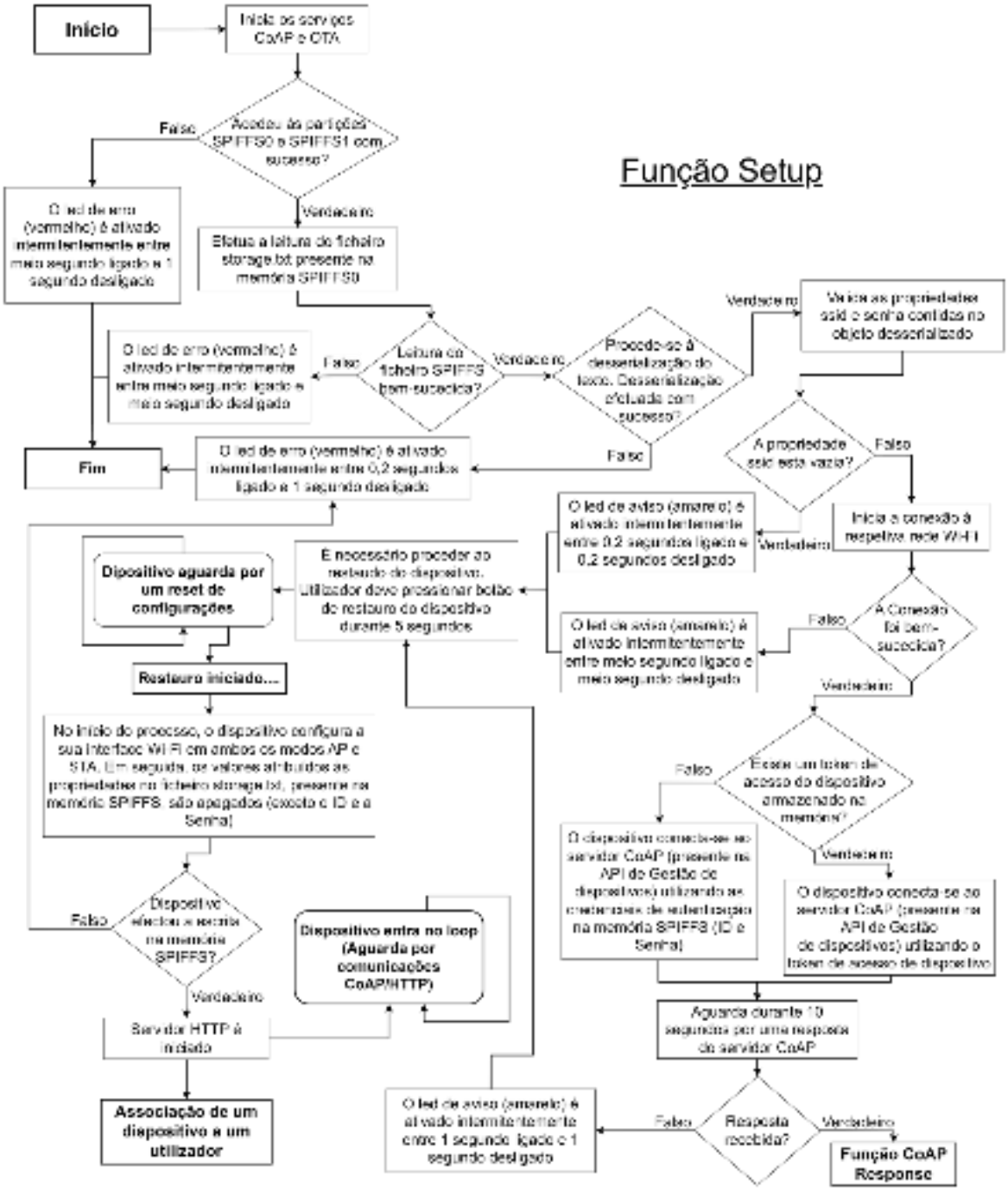


Figura 115 - Fluxograma da função Setup do firmware do dispositivo

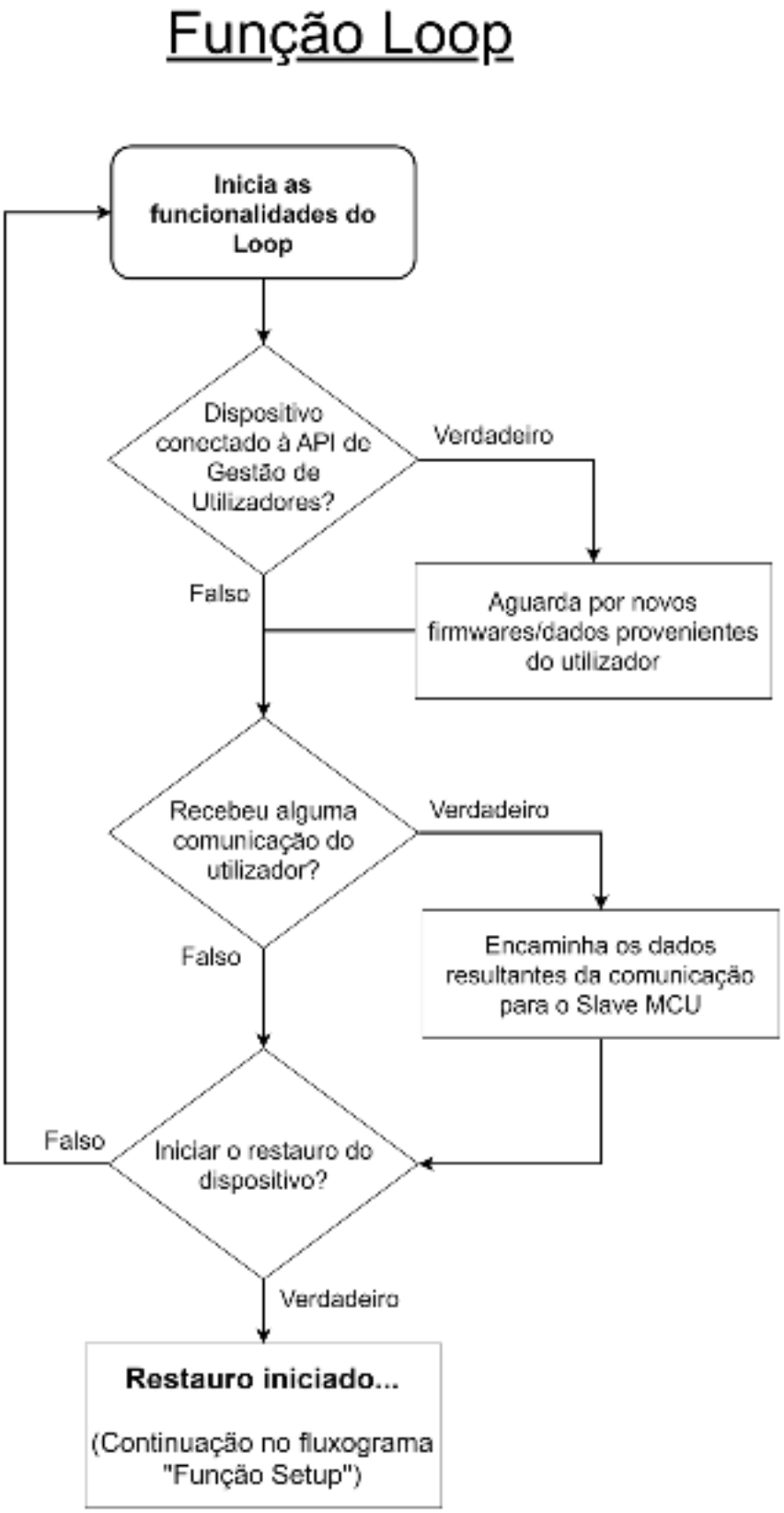


Figura 116 - Fluxograma da função Loop do firmware do dispositivo

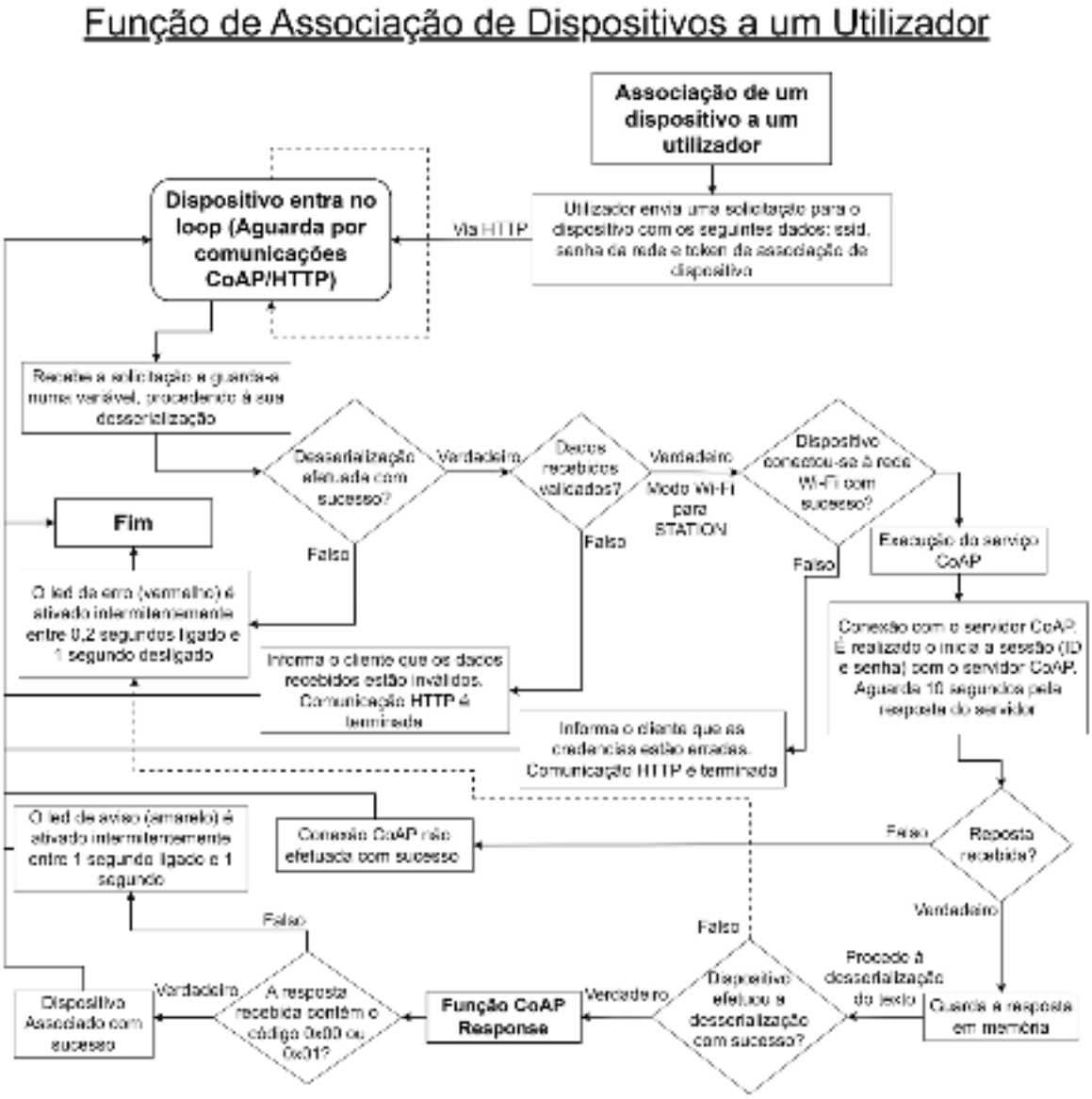


Figura 117 - Fluxograma de associação do dispositivo a um utilizador

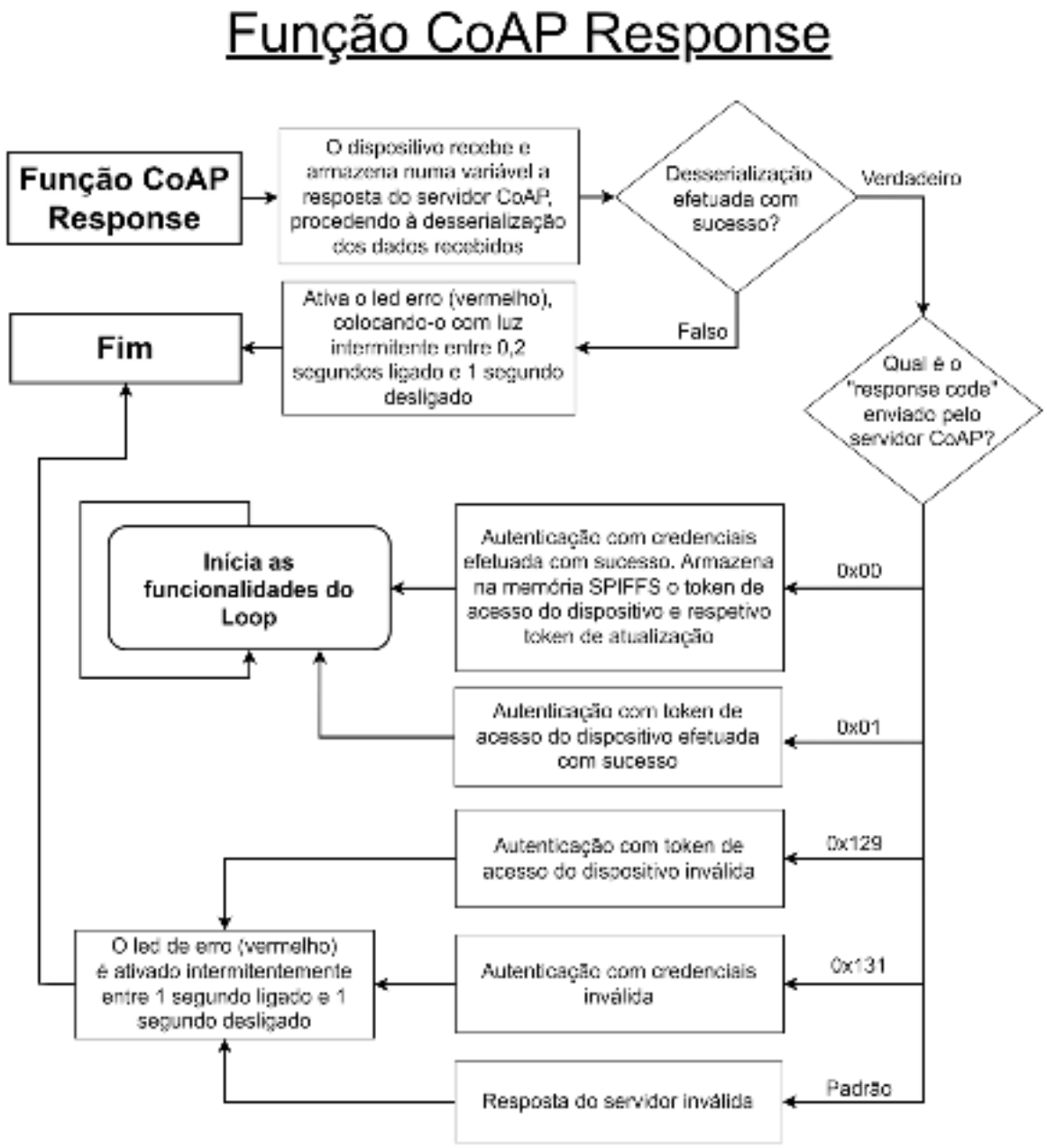


Figura 118 - Fluxograma do tratamento das respostas às solicitações CoAP provenientes das APIs

Apêndice C – Esquema Elétrico do Dispositivo

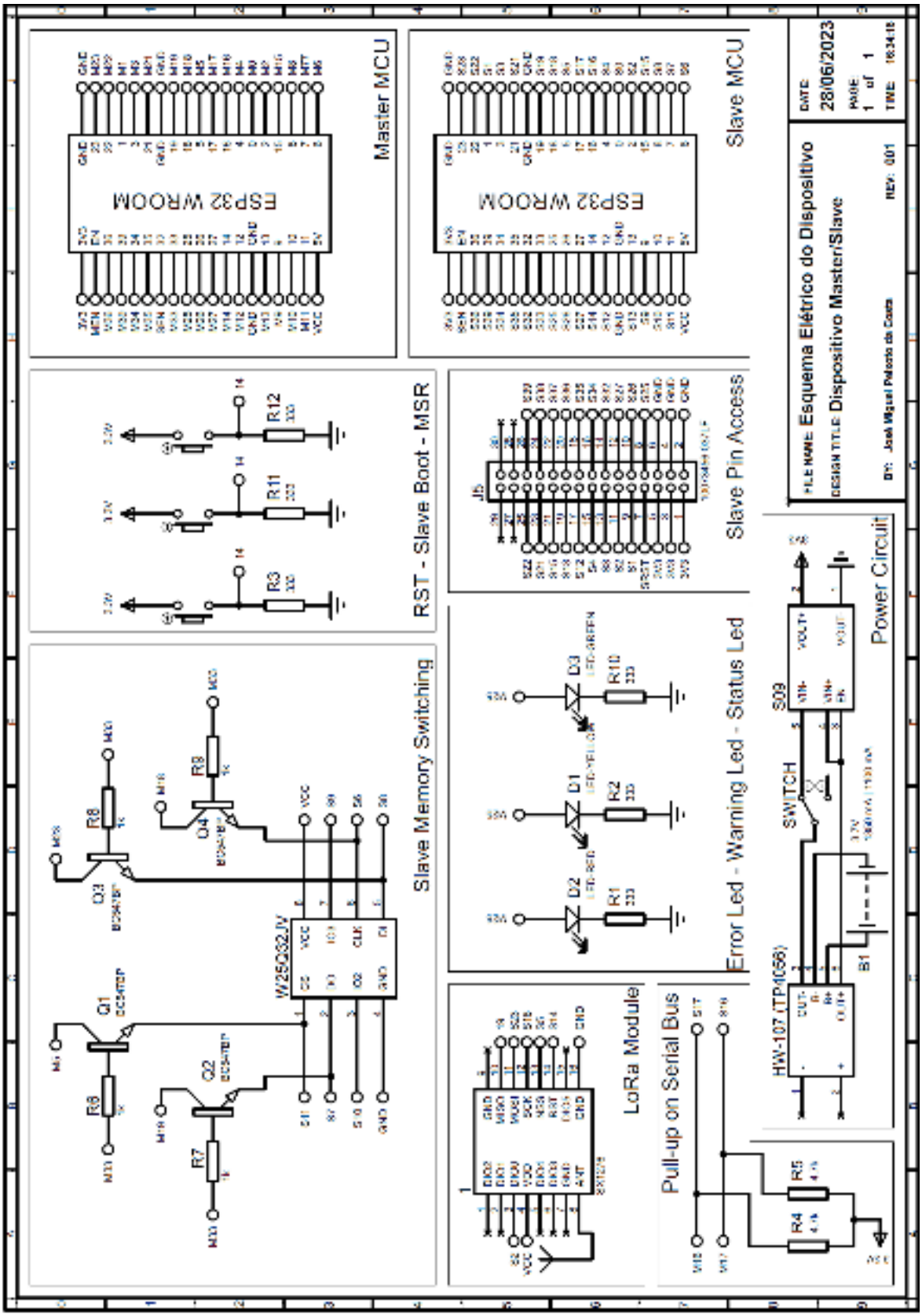


Figura 119 - Esquema Elétrico do Dispositivo

Apêndice D – Sinais Luminosos do Dispositivo

Tabela 25 - Descrição dos sinais luminosos emitidos pelo LED de Aviso

Tempo (s)		
Ligado	**Desligado**	**Significado**
1000	1000	Não foi possível estabelecer uma ligação ao servidor CoAP.
500	500	Não foi possível estabelecer uma ligação à rede Wi-Fi. Para configurar uma nova rede, o restauro deve ser dispositivo.
200	200	Não existem redes Wi-Fi guardadas na memória interna. O dispositivo deve ser restaurado.

Tabela 26 - Descrição dos sinais luminosos emitidos pelo LED de Erro

Tempo (s)		
Ligado	**Desligado**	**Significado**
1000	1000	Palavra-passe de autenticação incorreta.
500	1000	Ocorreu um erro durante a inicialização do SPIFFS.
200	1000	Ocorreu um erro durante a desserialização do objeto.
500	500	Ocorreu um erro ao ler o ficheiro SPIFFS.
200	200	Ocorreu um erro ao escrever no ficheiro SPIFFS.

Anexo A – Código Exemplo 1

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

```
#include "DHT.h"
```

```
#define IOT_HUB_NAME "IOT HUB NAME"
#define DEVICE_NAME "DEVICE NAME"
#define SAS_TOKEN "SAS TOKEN"

// WiFi settings
const char* ssid = "WifiName";
const char* password = "WifiPassword";´

const char* root_ca = \
                      "-----BEGIN CERTIFICATE-----\n" \
                      "....."                         \
                      "-----END CERTIFICATE-----\n";
WiFiClientSecure client;
```

```
#define DHTPIN 2
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

```
void sendRequest(String iothubName, String deviceName, String sasToken, String
message) {
  if ((WiFi.status() == WL_CONNECTED)) {

    HTTPClient http;
    String url = "https://" + iothubName + ".azure-devices.net/devices/" +
    deviceName + "/messages/events?api-version=2016-11-14";
    http.begin(url, root_ca);
    http.addHeader("Authorization", sasToken);
    http.addHeader("Content-Type", "application/json");
    int httpCode = http.POST(message);

    if (httpCode > 0) {
      String payload = http.getString();
      Serial.println(httpCode);
      Serial.println(payload);
    }

    else {
      Serial.println("Error on HTTP request");
    }
```

```
    http.end();
  }
}

void setup() {
  Serial.begin(115200);
  dht.begin();
  delay(100);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Print the IP address
  Serial.println(WiFi.localIP());
}

void loop() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t)) {
    Serial.println(F("Failed to read from DHT sensor!"));
    return;
  }
  Serial.print(F("Humidity: "));         Serial.println(h);
  Serial.print(F("Temperature: "));      Serial.println(t);

  sendRequest(IOT_HUB_NAME, DEVICE_NAME, SAS_TOKEN, "{\"humidity\":" + String(h)
  + ",\"temperature\":" + String(t) + "}");

  delay(10000);
}
```