



Instituto Superior de Engenharia de Lisboa

Área Departamental de Engenharia Eletrónica e Telecomunicações e de
Computadores

SeaSpot

Infraestrutura para facilitar a comunicação com objetos no mar

Autores:

Paulo Rosa | 44873
Raul Santos | 44806
Tiago Pilaro | 46147

Orientadores:

José Simão
Nuno Cota

Projeto e Seminário
Semestre de Verão 2022-2023

Resumo

A conexão entre diferentes dispositivos, *softwares* e agregados de comunicação tem vindo a aumentar para se formarem cada vez mais ambientes que adotam o conceito de Internet das Coisas (*Internet of Things*, também conhecido como *IoT*). A motivação para este projeto foi a necessidade de usar dispositivos de comunicação facilmente acessíveis (como é o caso do telemóvel/*smartphone*) em ambientes onde a conectividade de rede móvel é fraca ou inexistente, como é o caso da navegação costeira e zonas de pesca, onde existe necessidade de reportar a localização das embarcações ou enviar/receber pequenas mensagens para sistemas em terra. Este projeto realiza-se neste contexto, tendo como objetivo realizar um sistema de comunicação recorrendo a diferentes tecnologias de redes, nomeadamente *Bluetooth Low Energy (BLE)*, *LoRaWan* e *TCP/IP*.

Foi desenvolvida uma aplicação para um dispositivo móvel que comunica via *BLE* com dispositivos que suportam comunicação via *LoRa* para acederem a uma plataforma *LoRaWAN* designada de *TTN* (*The Things Network*). Esta servirá como ponto central para acesso a outros sistemas ou dispositivos. Para além disto, foi criada uma aplicação *web* que permite consultar o historial de mensagens enviadas e o perfil completo das propriedades dos dispositivos que enviam as mensagens.

Em suma, o produto final é composto por uma aplicação para dispositivos móveis que comunica com um dispositivo *LoRa* com o objetivo de enviar e receber mensagens através da infraestrutura *LoRaWAN*, bem como uma aplicação *web* capaz de receber e enviar mensagens de e para os dispositivos, como por exemplo a sua localização.

Abstract

The connection between different devices, software and aggregates of communications have been increasing to adopt an Internet Of Things (IoT) environment. The motivation for this project was the necessity to use devices of easier access (like mobile phones/smartphones) in environments where the mobile signal is weak or non-existent, similar to what happens in the coast and fishing areas, where there's a necessity of reporting the location of ships and send small messages to earth. This project aims at this context, with the goal of building a communication system involving multiple network technologies, like Bluetooth Low Energy (BLE), LoRaWan and TCP/IP

We built an application for a smartphone that communicates via BLE to devices that support LoRa, in order to access a LoRaWAN infrastructure, specifically TTN (The Things Network). This will serve as a central point of access to other systems or devices. On top of this, a web application was created which allows the consultation of the history of messages sent and the complete profile of the properties held by the devices that send the messages.

In short, the final product consists of an application for mobile devices which communicate with a LoRa device with the goal of sending and receiving messages through the LoRaWAN infrastructure, as well as making a web application capable of receiving and sending messages from and to devices, like their location per example.

Agradecimentos

Gostaríamos de agradecer aos nossos orientadores Professor José Simão e o Professor Nuno Cota por esta oportunidade para realizar um bom projeto de final de curso e por toda a ajuda, acessibilidade, exigência e simpatia.

Também queremos agradecer ao Professor Fernando Sousa, realçando o seu excelente trabalho em ajudar os alunos a concluírem a cadeira de Projeto e Seminário.

Por fim, enquanto autores, gostaríamos de salientar que foi um projeto muito enriquecedor que proporcionou diversos desafios que ajudaram a desenvolver aptidões e competências tanto na área de *software*, como de *hardware*.

Índice

1. Introdução.....	1
1.1 Motivação.....	1
1.2 Objetivos e análise.....	1
2. Trabalhos relacionados.....	2
2.1 Cayenne Low Power Payload.....	2
2.1.1 Comparações/Diferenças (Cayenne/SeaSpot).....	2
2.2 Rastreamento de localização via Lora e GPS.....	3
2.2.1 Comparações/Diferenças (Indie/SeaSpot).....	3
2.3 Conceitos.....	4
3. Arquitetura.....	5
3.1 Funcionalidades.....	5
3.2 Escolha das tecnologias e metodologias.....	6
3.3 Diagrama de execução (App Android - TTGO).....	6
4. Utilização das redes LoRaWAN e BLE.....	8
4.1 Protocolo de rede LoRaWAN.....	8
4.1.1 Visão geral do protocolo de rede LoRaWAN.....	8
4.1.2 Envio de mensagens.....	9
4.1.3 Vantagens e aplicabilidades da rede LoRaWAN.....	10
4.2 The Things Network (TTN).....	11
4.2.1 Descrição e importância da TTN.....	11
4.2.2 Arquitetura da TTN.....	11
4.3 Bluetooth Low Energy.....	12
4.3.1 Visão geral.....	12
4.3.2 Ligações e funcionalidades BLE.....	12
4.3.2 Operating States e Roles.....	13
4.3.3 Funcionalidades do Generic Access Profile (GAP).....	13
4.3.4 Funcionalidades do Generic Attribute Profile (GATT).....	14
5. Desenvolvimento.....	16
5.1 Programação do TTGO T-Beam.....	16
5.1.1 Visão geral.....	16
5.1.2 Programação do dispositivo.....	16
5.1.3 Programação das mensagens.....	18
5.1.3 Envio de bytes.....	19
5.1.4 Recepção de bytes.....	19
5.1.5 Obtenção do sinal GPS.....	20
5.2 Programação da aplicação Android.....	21
5.2.1 Comunicação entre a aplicação Android e o T-Beam.....	21
5.2.2 Problemas e limitações da biblioteca BLE para Android.....	23
5.3 Programação da aplicação web.....	24
5.3.1 Arquitetura da API.....	24
5.3.2 Estrutura do servidor.....	25

5.3.2 Modelo de dados.....	26
5.3.4 Obtenção do mapa.....	28
5.3.5 Interface cliente.....	28
5.4 Demonstração e testes numa situação real.....	30
6. Conclusão.....	31
6.1 Críticas construtivas.....	31
6.2 Pontos fortes.....	31
Referências.....	33

Índice de figuras

Figura 1 - Componentes da arquitetura SeaSpot.....	5
Figura 2 - Diagrama de Execução.....	7
Figura 3 - Dispositivo a enviar uplink para uma plataforma de rede LoRaWAN.....	8
Figura 4 - Plataforma de rede LoraWAN a enviar downlink para um dispositivo.....	8
Figura 5 - Diagrama do consumo de energia entre classes de dispositivos.....	9
Figura 6 - Protocolo da receção de downlink de dispositivos de classe A.....	10
Figura 7 - Comunicação entre dispositivo e plataforma de rede LoRaWAN.....	10
Figura 8 - Pilha do protocolo BLE.....	12
Figura 9 - Estados dos roles [23].....	13
Figura 10 - Estrutura do GATT server profile [26].....	14
Figura 11 - Dispositivo LILYGO® TTGO T-Beam V1.1 ESP32.....	16
Figura 12 - Permissões e ligação de Bluetooth e Localização.....	21
Figura 13 - Conexão, leitura e escrita de valores na app.....	22
Figura 14 - Caminhos da API.....	25
Figura 15 - Estrutura da WebApp.....	26
Figura 17 - Objeto Device.....	28
Figura 18 - Página da lista de mensagens.....	29
Figura 19 - Página de uma mensagem.....	30
Figura 20 - Página de um dispositivo.....	30
Figura 21 - TTGO e telemóvel com a app conectada ao TTGO.....	31

Índice de tabelas

Tabela 1 - Bitmap de características.....	18
---	----

1. Introdução

O projeto denominado *SeaSpot* visa facilitar a comunicação e a gestão de objetos no mar. Estes objetos podem ser, por exemplo, bóias com uma certa utilidade para pescadores e cientistas que estudam o mar ou outras entidades.

Logo, através de um dispositivo móvel, pretende-se facilitar a localização e a consulta de dados de outros dispositivos ou sensores presentes em bóias.

Dado que é comum haver pouca ou nenhuma rede no alto mar, vai ser utilizada uma tecnologia de redes designada *LoRa* [1], que tem capacidades de cobertura superiores às redes tradicionais. A bordo da embarcação será necessário um dispositivo que nos permite comunicar através do protocolo de rede *LoRaWAN* [2] com a plataforma de rede *The Things Network (TTN)*, que é global e livre. A comunicação com este dispositivo será feita com *Bluetooth Low Energy (BLE)* [3] usando um telemóvel.

1.1 Motivação

Com o desenvolvimento da tecnologia, existe cada vez mais a capacidade de transporte de dados provenientes de sensores ou de outros dispositivos.

A tecnologia designada *LoRa* tem-se destacado como uma das mais populares para a comunicação de dados devido à sua eficiência e facilidade de uso. Além de ser uma aplicação prática e inovadora desta tecnologia, este projeto vai inovar no sentido de permitir a integração entre dispositivos que no presente não conseguem comunicar através da rede *LoRa*, nomeadamente dispositivos móveis. A tecnologia *BLE* servirá de meio de comunicação entre estes sistemas.

A solução pode ter um impacto social positivo ao melhorar a comunicação e a segurança das embarcações de pesca em regiões remotas, além de possibilitar a coleta de dados que possam ser usados para fins científicos e de conservação da natureza. Desta forma, este projeto apresenta-se como uma oportunidade única de aprendizagem e de contribuição para a sociedade.

1.2 Objetivos e análise

Sendo estabelecido os termos e conceitos básicos do projeto, listamos abaixo um resumo das tarefas a cumprir.

- Permitir a comunicação entre um dispositivo móvel e um dispositivo que utilize o protocolo de comunicação *LoRaWAN*.
- A comunicação entre telemóvel e o dispositivo deverá ser feita com *Bluetooth Low Energy (BLE)*.
- Os dados trocados deverão ser informativos e úteis para diversas aplicações, como por exemplo, a localização.
- Adquirir conhecimentos de *LoRa*, *BLE*, e desenvolver uma aplicação *Android*
- Desenvolver uma aplicação *Web* para a gestão e visualização de dados.

2. Trabalhos relacionados

2.1 Cayenne Low Power Payload

A *Cayenne* [4] trata-se de uma plataforma *IoT* (*Internet Of Things*) desenvolvida pela *myDevices*. A ideia desta plataforma é possibilitar aos utilizadores ligarem os seus dispositivos a uma rede *LoRaWAN* disponível, e utilizarem também um dispositivo que se ligue à *Cayenne*, de forma a que se possam obter informações de determinada zona através de sensores que fornecem dados como temperatura, localização e humidade.

Para funcionar com a *Cayenne*, é necessário realizar um *Log In* na plataforma, e escolher o tipo de dispositivo que se irá utilizar para o projeto (existe a possibilidade de utilizar *Raspberry Pi* ou *Arduino*). É também possível usar a própria *API* da *Cayenne*: *CayenneAPI*). No entanto, para que a utilização seja semelhante à do projeto *SeaSpot*, tem de ser usado *LoRa*.

É também apresentada a possibilidade de o utilizador se ligar a uma rede *LoRaWAN*, bem como os dispositivos modelos que podem ser utilizados. Os que são aconselhados para utilização, de modo a que a implementação fique semelhante ao projeto, são: *The Things Network* como rede *LoRaWAN* a ser utilizada, e *Cayenne LPP* como dispositivo modelo. Caso o dispositivo ainda não esteja registado na *The Things Stack*, tal terá que ser feito, de modo a que sejam fornecidos dados como *Address* e *Keys*.

Assim que os passos anteriores estejam concluídos, será possível ver o dispositivo num mapa, bem como as informações que estão a ser recolhidas através dos sensores, sendo possível observar dados que já tenham sido obtidos (ou seja, é mantido um historial).

2.1.1 Comparações/Diferenças (*Cayenne/SeaSpot*)

Apesar de que o que foi exposto como exemplo ser um breve resumo do funcionamento da plataforma *Cayenne*, bem como é que esta pode ser configurada de modo a que a sua utilização fique semelhante ao projeto *SeaSpot*, é possível ver semelhanças nos modos de funcionamento.

Tal como utilizado no projeto *SeaSpot*, a *Cayenne* utiliza uma rede *LoRaWAN*, mais especificamente, a *The Things Network* (*TTN*), por se tratar de uma rede de fácil utilização e livre acesso. Quem tenha acesso à *The Things Network* (*TTN*) fica também habilitado à utilização da *The Things Stack*, onde se pode registrar dispositivos que serão utilizados para realizar pedidos a esta mesma rede. Outro ponto semelhante ao projeto é a utilização de um dispositivo *Low Power Payload* (*LPP*) que é um formato de envio de dados compatível com o *LoRaWAN*.

No entanto, existem diferenças no modo de utilização da plataforma *Cayenne* e no como esta funciona comparativamente ao projeto. Essas diferenças são observáveis no tipo de dispositivo onde as leituras irão ser observadas, bem como os dispositivos que realizam esses mesmos pedidos. A *Cayenne* é uma plataforma que existe apenas no formato *Desktop*, estando o utilizador limitado quanto à sua utilização (dados apenas observáveis num computador), enquanto que no presente projeto, a ideia é os pedidos poderem ser feitos e observados num telemóvel.

2.2 Rastreamento de localização via *Lora* e *GPS*

O projeto [5] que será referido agora tem como base uma implementação amadora e autónoma, não tendo um nome associado (pelo que no decorrer desta secção, será referido como: *Indie TTGO GPS Tracker*).

O *Indie* trata-se de um projeto onde é feito uma ligação entre 3 dispositivos, sendo esses: um telemóvel, um *TTGO T-Beam*, e um *TTGO LoRa*. A ideia base do projeto é poder realizar tanto *Live Tracking* como *GeoTracking/ GeoFencing*, ou seja, poder obter dados de caráter geográfico de um determinado dispositivo que realiza essas leituras, ou até mesmo de uma determinada área geográfica.

A ligação dos dispositivos é feita da seguinte forma: um determinado telemóvel está ligado, de forma direta (*Wired Connection*), a um dispositivo *TTGO LoRa*; desta forma, a ligação com o dispositivo *TTGO T-Beam* é feita através do *TTGO LoRa*.

2.2.1 Comparações/Diferenças (*Indie/SeaSpot*)

Uma vez que não existem muitas informações sobre este projeto amador (exceto o modo de programação e configuração dos elementos do mesmo), é apenas possível fazer referência aos tipos de tecnologia utilizados, bem como o modo de emparelhamento.

Tal como utilizado no projeto *SeaSpot*, o projeto *Indie* recorre a um telemóvel *Android* para realizar os seus pedidos. No entanto, o modo de comunicação que é utilizado para que este faça os ditos pedidos para o *TTGO T-Beam*, é diferente: enquanto que no projeto *SeaSpot* a ligação entre telemóvel e *TTGO T-Beam* se realiza por *BLE*, no projeto *Indie* é utilizado um dispositivo Adafruit Feather (que suporta *LoRa*) que é conectado via *USB-C* do telemóvel para *USB Micro C* do dispositivo.

Relativamente ao modo de como os pedidos são enviados e recebidos, não está explícito o modo de como estes são efetuados, ou tratados.

2.3 Conceitos

Como foi possível observar pelos exemplos referidos anteriormente, até à data em que este documento foi escrito, não existe uma implementação que visa resolver o problema a que está proposta a resolução. No entanto, é de notar que existem projetos e aplicações que realizam parte do problema, ainda que de forma parcial, como a ligação de dispositivos a sensores de modo a que os seus dados possam ser observados num *desktop* (*Cayenne*), ou a utilização de um telemóvel para a comunicação com um dispositivo *TTGO T-Beam* usando uma abordagem diferente (*Indie*).

Assim, o projeto procura dinamizar o conceito que é a obtenção de dados de sensores, de uma forma mais simples, recorrendo não só a algo que um utilizador comum possui diariamente (o seu próprio telemóvel), como tendo uma forma de consultar os dados de uma forma mais intuitiva

3. Arquitetura

A arquitetura do projeto pode ser ilustrada na Figura 1. Através de um telemóvel *Android*, haverá leitura e escrita de dados que o *TTGO* disponibiliza via *BLE*. Dados estes que são configurados pelo programador do *TTGO* ou que são obtidos da *TTN* (na aplicação real do projeto, também se poderiam adicionar outros sensores que disponibilizavam valores).

Relativamente à aplicação *web*, ela apresentará mensagens enviadas à aplicação *TTN* via o *TTGO* e de outros dispositivos que estejam registados na aplicação *LoRaWAN*. Estas mensagens são guardadas e ilustradas na página *web*, acompanhadas com um mapa da localização dos dispositivos que as enviam.

Para que o projeto seja simulado tendo em conta condições reais, as mensagens enviadas pelos outros dispositivos *LoRa* podem ser simuladas via *TTN*.

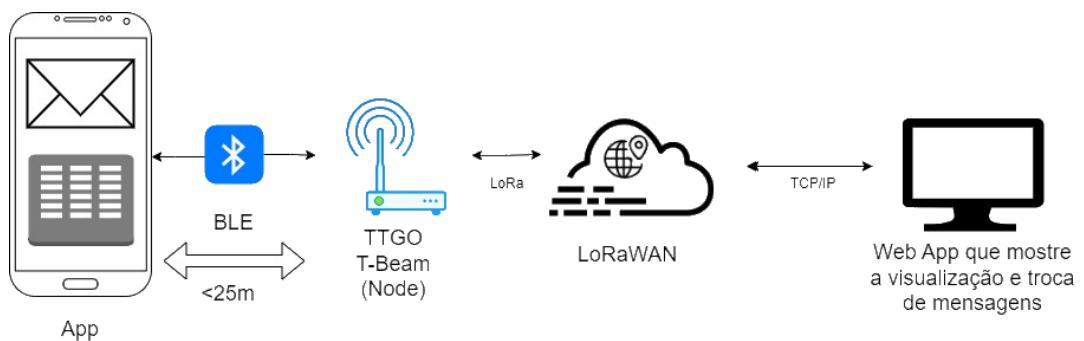


Figura 1 - Componentes da arquitetura *SeaSpot*

3.1 Funcionalidades

No final, o projeto deverá ter as seguintes capacidades:

- A app *Android* lê e escreve valores específicos do *TTGO*.
- O *TTGO* recebe mensagens da aplicação *LoRaWAN*
- O *TTGO* envia escritas para a *web app*
- A *web app* guarda essas mensagens (e metadados) e disponibiliza-as
- A *web app* permite apagar mensagens da base de dados (por serem antigas)

3.2 Escolha das tecnologias e metodologias

Para a realização deste projeto, será necessário um conjunto de *software* e *hardware*. O componente de *hardware* a ser utilizado é um *TTGO T-Beam* [6] do fornecedor *LILYGO®*, programado na linguagem *MicroPython* [7]. A plataforma de rede *LoRaWAN* a ser utilizada é a *The Things Network*.

Para o desenvolvimento da aplicação *web* será utilizado a plataforma *NodeJS* com recurso à *framework Express* [8], uma estrutura de aplicativo da Web de *back-end* para criar *APIs RESTful*, e *Handlebars* [9] para gerar o *HTML* no *front-end*. A base de dados será *Elasticsearch* [10]. O editor de escolha será o *Visual Studio Code* (VSC) [11].

Para a aplicação móvel será necessário o *Android Studio* [12] e será usada a linguagem *Kotlin*.

Um objetivo extra e para o enriquecimento do trabalho é a habilidade de mostrar um mapa que mostre a localização de dispositivos aos quais podemos comunicar. Para tal, será usado o *Leaflet* [13]. Esta biblioteca não só é *opensource* como também é muito leve, rápida e fácil de usar.

3.3 Diagrama de execução (*App Android - TTGO*)

No contexto de arquitetura do projeto, é importante ter em conta (no que toca a uma ajuda visual) como irá proceder a comunicação entre o dispositivo móvel (telemóvel) e o *TTGO*.

O telemóvel irá estabelecer um emparelhamento inicial com o *TTGO*, de forma a que comece a existir um canal bidirecional exclusivo entre os dois dispositivos. Assim que esse canal estiver estabelecido, o telemóvel pode efetuar pedidos. Na imagem 2, o que está demonstrado é a alteração de características presentes no telemóvel, no que toca ao dispositivo. Com base em *triggers*, assim que uma característica é atualizada (neste caso, é feito um pedido de escrita), é realizado um *Uplink* para *TTN* com essa atualização.

É importante ter a noção de *scheduling* em conta, onde são efetuados *downlinks* a partir da *TTN* para a obtenção dos pedidos de atualização das características (tal serve para o caso de existirem múltiplos *TTGO's*, e estes poderem ler características entre si).

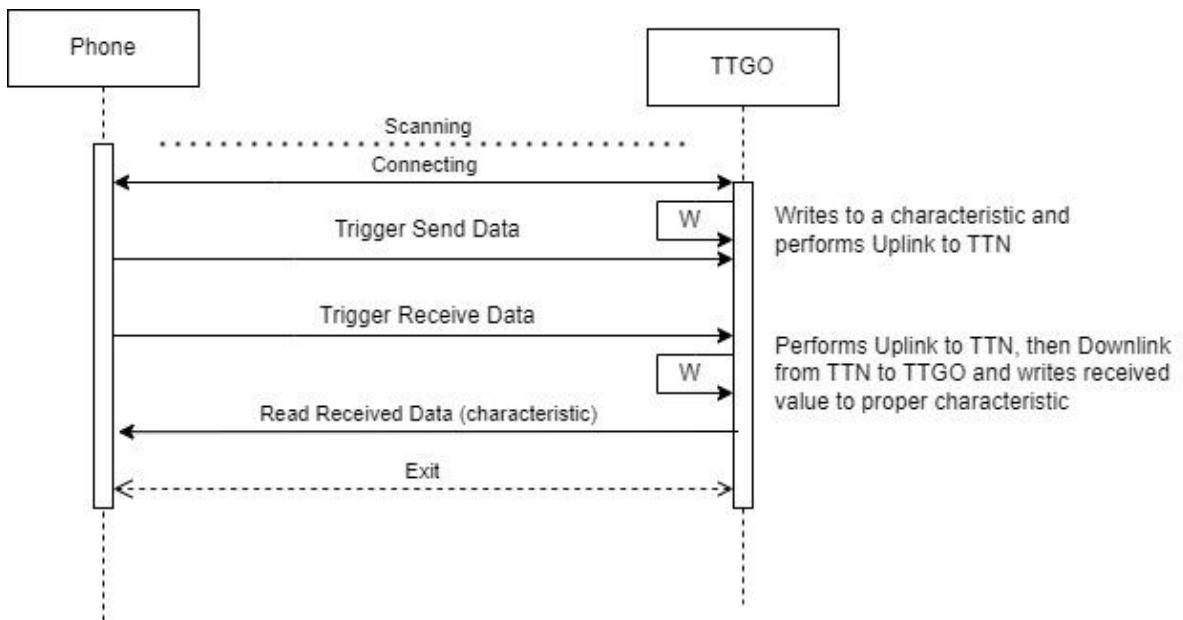


Figura 2 - Diagrama de Execução

4. Utilização das redes LoRaWAN e BLE

Neste capítulo, é explicado como o protocolo de rede *LoRaWAN* e *BLE* funcionam e como estas tecnologias vão ser usadas no projeto. Visto que têm capacidades e especificações substancialmente diferentes de outras plataformas de comunicação sem fios, neste capítulo é feito a organização e explicação da matéria para uma boa compreensão do projeto. A especificação da *LoRaWAN* pode ser consultada aqui [\[43\]](#).

4.1 Protocolo de rede LoRaWAN

4.1.1 Visão geral do protocolo de rede LoRaWAN

Uma rede baseada em *LoRaWAN* é composta por dispositivos (*devices*), *gateways*, um servidor de rede (*network server*) e uma aplicação servidora (*application server*). Os dispositivos enviam mensagens para *gateways* (*uplink*), e os *gateways* encaminham-nas para o servidor de rede, para finalmente chegar à aplicação servidora. Além disso, o servidor de rede pode enviar mensagens através de um *gateway* para um ou mais dispositivos (*downlinks*). Isto é exemplificado nas duas seguintes imagens:

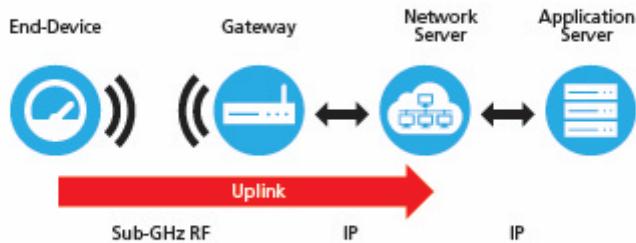


Figura 3 - Dispositivo a enviar uplink para uma plataforma de rede *LoRaWAN* (retirado de [\[14\]](#))

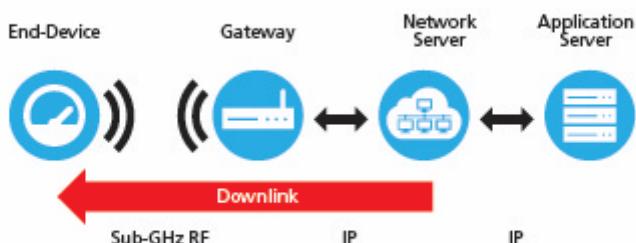


Figura 4 - Plataforma de rede *LoRaWAN* a enviar downlink para um dispositivo (retirado de [\[14\]](#))

Os dispositivos que suportam a rede *LoRaWAN* podem ser classificados em três classes: Classe A, Classe B e Classe C. A classe dos dispositivos é uma forma

de indicar o nível de energia usado e a gestão das janelas de receção (bem como a frequência de envios de *uplinks* e *downlinks*) [15]. Quanto mais eficiente, maior o tempo de vida da bateria. O modo de uso do TTGO disponibilizado será do tipo A, como se verá mais à frente.

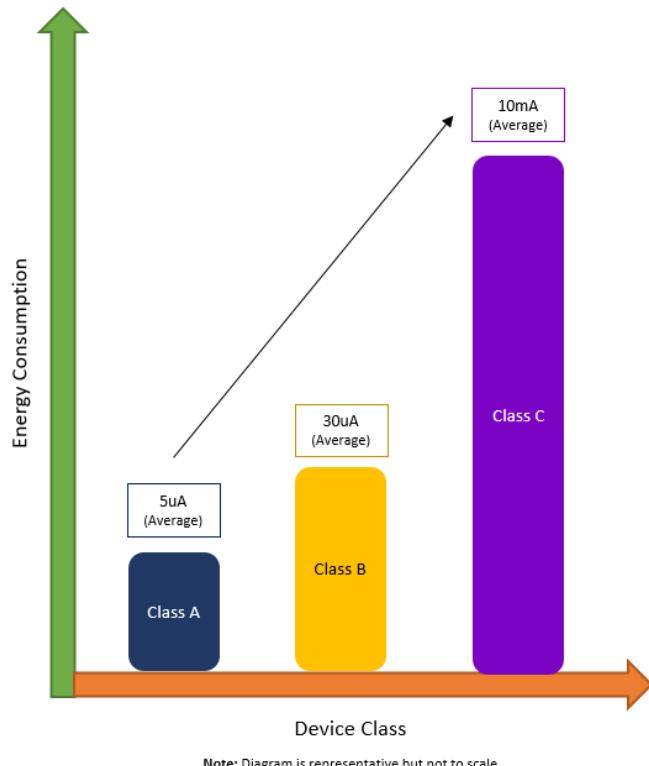


Figura 5 - Diagrama do consumo de energia entre classes de dispositivos
(retirado de [14])

4.1.2 Envio de mensagens

As mensagens que são agendadas para *downlink* do servidor de rede são colocadas em filas de espera até à próxima vez que uma mensagem de *uplink* for recebida do dispositivo e uma janela de recepção for aberta para efetuar um *downlink*. Este desenho é específico para aplicativos que exigem comunicação de *downlink* em resposta a um *uplink*, ou que podem agendar *downlinks* com antecedência com requisitos de latência bastante flexíveis.

Os dispositivos suportam comunicação bidirecional entre um dispositivo e um *gateway*. As mensagens de *uplink* (do dispositivo para o servidor) podem ser enviadas a qualquer momento. Para receber mensagens via *downlink*, existem duas janelas de receção em horários especificados (1 segundo e 2 segundos por *default*) após uma transmissão de *uplink* (*TX*). Se o servidor não responder em nenhuma dessas janelas de receção (*RX*), a próxima oportunidade será após a próxima transmissão de *uplink* do dispositivo. O servidor pode responder na primeira janela de receção (*RX1*), ou na segunda janela de receção (*RX2*), mas não usa ambas as janelas.

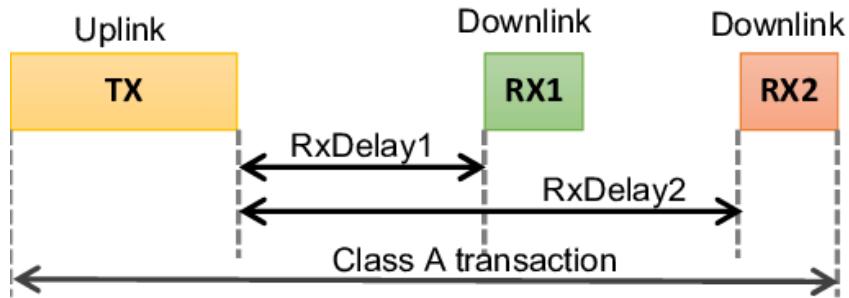


Figura 6 - Protocolo da receção de *downlink* de dispositivos de classe A
(retirado de [16])

4.1.3 Vantagens e aplicabilidades da rede *LoRaWAN*

A principal diferença entre a abordagem *LoRaWAN* e outras abordagens de comunicação a longas distâncias é que os dispositivos são emparelhados com a própria rede e não estão vinculados exclusivamente a um único *gateway*. Ao invés disso, os dispositivos transmitem os sinais para todos os *gateways* dentro do alcance. Cada um dos *gateways* receptores passa os dados para o servidor de rede e, em seguida, o servidor de rede elimina a duplicação da mensagem e envia uma única versão para a aplicação servidora. Abaixo estão listadas as vantagens da arquitetura:

- *Gateways* podem ser adicionados em múltiplos sítios e têm alcances razoáveis [17]
- A entrega de mensagens é robusta, visto que vários *gateways* recebem o mesmo pacote de dados durante cada *uplink*. Isso é chamado de diversidade espacial de *uplink*.
- Não há necessidade de planejar frequências diferentes para cada *gateway* ou realojar frequências quando o número de *gateways* mudar. Todos os *gateways* estão constantemente a ouvir todas as frequências da rede.
- É usada pouca energia.

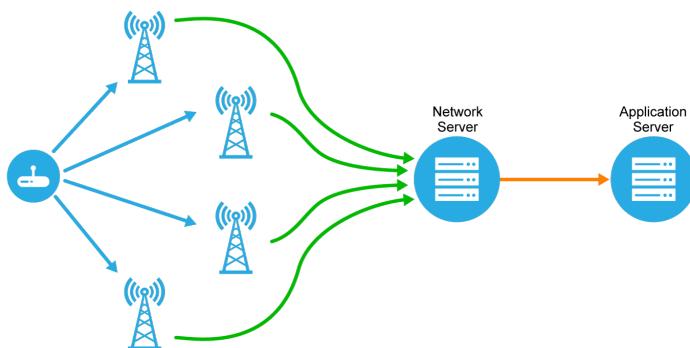


Figura 7 - Comunicação entre dispositivo e plataforma de rede LoRaWAN
(retirado de [14])

4.2 The Things Network (TTN)

4.2.1 Descrição e importância da TTN

A *The Things Network (TTN)* é uma plataforma de rede global, aberta e gratuita com o intuito de suportar a comunicação de dispositivos *IoT* por meio do protocolo de rede *LoRaWAN* [18].

A *TTN* baseia-se em servidores *cloud* que conecta dispositivos e *gateways LoRaWAN* no mundo. Tem a capacidade de autenticação de dispositivos, gestão de dados e integração com aplicativos e serviços externos.

A utilização da *TTN* é especialmente relevante para projetos de *IoT* que exigem uma cobertura ampla e de alcance global. Com a *TTN*, os programadores podem aproveitar uma rede estabelecida e colaborativa, evitando a necessidade de construir e manter sua própria plataforma. Além disso, a *TTN* oferece suporte a diversas aplicações e casos de uso, desde gestão ambiental, agricultura inteligente e até cidades inteligentes.

4.2.2 Arquitetura da TTN

A arquitetura da *TTN* oferece uma plataforma escalável e flexível para a comunicação de dispositivos através do protocolo de rede *Low Power Wide Area Networking LoRaWAN*.

Device (Dispositivo) - Um dispositivo que inclui um *LoRa Modem*, envie dados para *gateways* habilitados com o protocolo de rede *LoRaWAN*, tem um identificador globalmente exclusivo *DevEUI* e um identificador exclusivo de rede *DevAddr*. No dispositivo é possível escolher entre os seguintes modos de ativação, por *Over The Air Activation (OTAA)* e/ou *Activation By Personalization (ABP)*. Foi escolhido *ABP* por motivos de simplicidade, mais concretamente, para que o endereço do dispositivo seja fixo e sempre identificável [19]. Visto que não está previsto que o *TTGO* navegue de um *cluster* [20] de uma área (continental) para outra, não existe uma necessidade plausível em usar *OTAA*. No entanto, não está descartada a sua utilização no futuro, como foi recomendado.

Gateway - Os *gateways* são os pontos de acesso ao protocolo de rede *LoRaWAN*. São responsáveis por receber as mensagens dos dispositivos e encaminhar para o servidor de rede da *TTN*. Os *gateways* comunicam-se com os dispositivos utilizando o protocolo *LoRaWAN*.

Network Server (Servidor de rede) - O servidor de rede, também conhecido como sistema de *back-end*, é o componente central da arquitetura da *TTN*. Este é responsável pela gestão, registo e autenticação dos dispositivos, controla o acesso à rede, encaminha as mensagens entre os dispositivos e as aplicações e aplica as

políticas de segurança. O servidor de rede também coordena os *gateways* para garantir a receção correta das mensagens dos dispositivos.

Application Server (Aplicação servidora) - A aplicação servidora gera a camada de aplicação da *LoRaWAN*, incluindo o processamento das mensagens recebidas dos dispositivos e pelo envio das mensagens de volta para os dispositivos. Também permite integrações de forma a que as mensagens sejam enviadas para outros sistemas, por exemplo via *HTTP* usando *webhooks*, como se verá mais à frente.

Consola da TTN - A Consola da *TTN* é a interface *web* fornecida pela plataforma. Com ela, os utilizadores podem gerir as aplicações, visualizar informações sobre os dispositivos, configurar *gateways*, analisar o tráfego de dados e realizar outras tarefas administrativas relacionadas à rede *TTN*.

4.3 Bluetooth Low Energy

4.3.1 Visão geral

Bluetooth Low Energy (BLE) é um protocolo de comunicação que se destaca pelo consumo muito baixo de energia. Oferece ligações rápidas, procedimentos eficientes de conexão e utiliza pacotes muito curtos para transmissão de dados. O *BLE* segue um *design* assimétrico para periféricos, reutiliza algumas características do *Bluetooth Clássico* e utiliza uma arquitetura de rádio [21]. Foi usado o livro *Getting Started with Bluetooth Low Energy* [44] como referência para esta matéria.

4.3.2 Ligações e funcionalidades *BLE*

É importante perceber como funciona uma conexão *BLE*, quais as funções desempenhadas pelos dispositivos envolvidos e como é que os dados são transferidos de um dispositivo para o outro. Na figura abaixo ilustra como o protocolo *BLE* se divide.

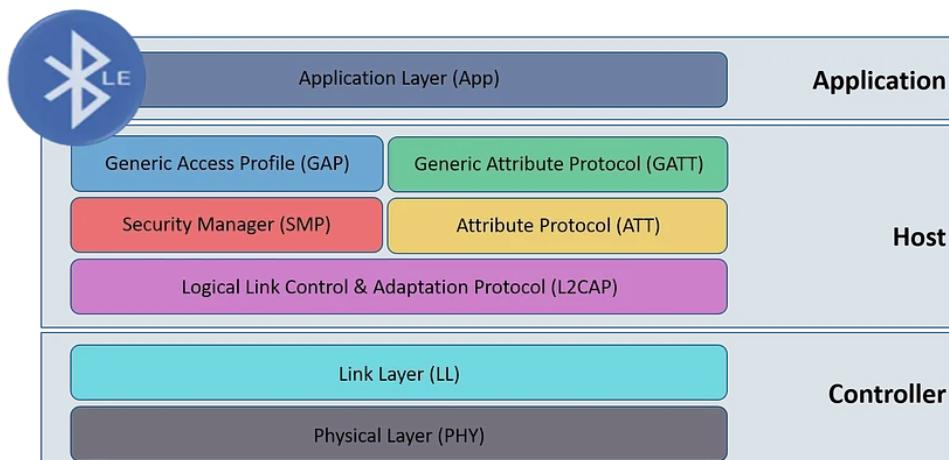


Figura 8 - Pilha do protocolo BLE
(retirado de [22])

4.3.2 Operating States e Roles

Numa aplicação *BLE* existem *Operating States* para classificar os intervenientes e *Roles* para classificar ações. Os estados são:

- *Standby* - Refere-se a um estado no *BLE* em que um dispositivo não está a transmitir nem a receber pacotes.
- *Advertising* - É uma funcionalidade do *BLE* onde um dispositivo realiza *broadcasts advertisement* em canais de *advertising*. Estes *advertisement* contém informação sobre os dispositivos e os serviços disponíveis .
- *Scanning* - É o processo executado por um dispositivo *BLE* para procurar *advertisers* realizando *scan* em canais de *advertising*.
- *Initiating* - É o processo em que um dispositivo inicia a ligação com um *advertiser*. Assim que o dispositivo que realiza o scan encontrar o dispositivo de *advertising* de interesse, pode inicializar o pedido de ligação.

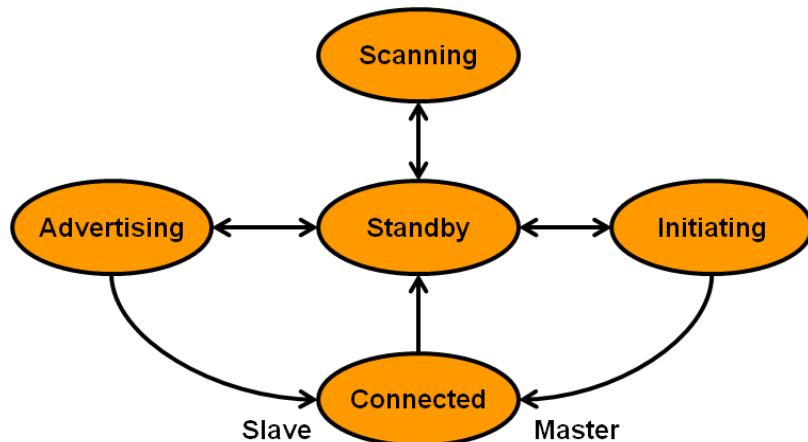


Figura 9 - Estados dos *roles* [23]

Os dispositivos podem ter os seguintes papéis:

- *Master* - o dispositivo *BLE* que inicia uma solicitação de ligação a um dispositivo *advertising*.
- *Slave* - o dispositivo *BLE* que aceita uma solicitação de ligação após *advertising*.

4.3.3 Funcionalidades do *Generic Access Profile (GAP)*

Para que dispositivos *BLE* transmitam dados entre si, deve ser formado um canal de comunicação. A forma como esse canal é formado e mantido é da responsabilidade do *GAP*.

O *GAP* estabelece que para dois dispositivos se conectarem e comunicarem, um deve assumir o papel de *Central (Master)* e o outro deve assumir o papel de *Peripheral (Slave)*. O *Central* é o dispositivo que recebe pedidos de outros (e decide se e como podem ser feitos) e expõe informação e funcionalidades a quem se conecta a ele, enquanto que o *Peripheral* é o dispositivo que se conecta ao *Central*.

4.3.4 Funcionalidades do *Generic Attribute Profile (GATT)*

O *Generic Attribute Profile (GATT)* no *BLE* estabelece a estrutura dos dados disponibilizados para quem se conecta à aplicação *BLE*. O GATT pode conter vários “serviços”, com várias “características” e estas podem ter vários “descritores”. Estes 3 componentes são referidos como “atributos”. Estes termos só são usados para classificar e organizar informação de acordo com a especificação *BLE* [24]. Este documento referido especifica quais os *Universally Unique Identifier (UUID)* que se devem usar de modo a identificar uma ampla variedade de propriedades, objetos ou funções no contexto de dispositivos *IoT* ou microcontroladores, de forma a haver interoperabilidade entre dispositivos e uma boa forma de interpretar valores (embora seja permitido ao programador customizar ou se desviar da especificação [25]).

Este perfil utiliza o *Attribute Protocol (ATT)* como um mecanismo de transporte para organizar dados em atributos (conjunto de bytes). O TTGO concretiza este modelo de dados na forma de um *GATT server*, cujo telemóvel se vai conectar.

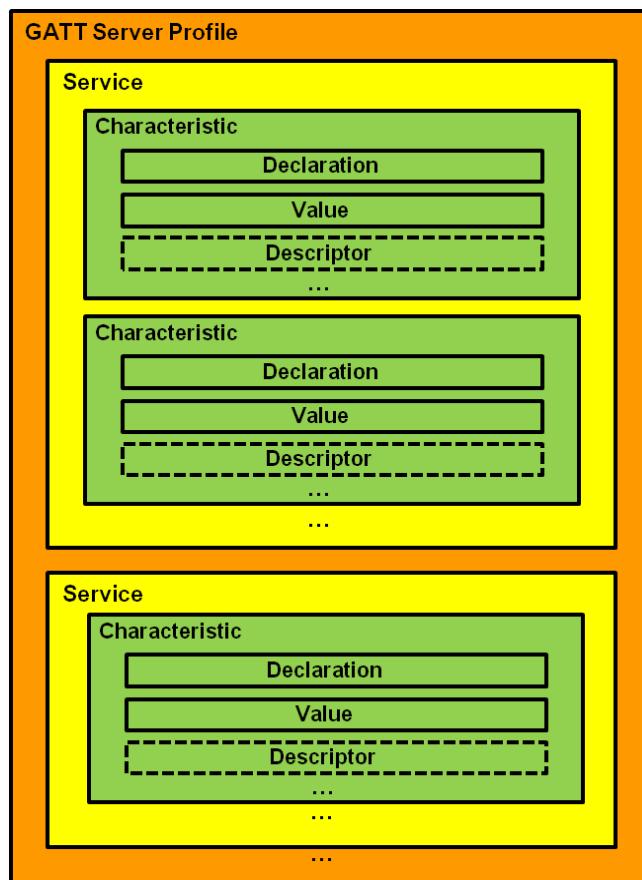


Figura 10 - Estrutura do GATT server profile [26]

O *GATT server* está associado às funções de dispositivo *Link Layer Slave* e *GAP Peripheral*. Contém os dados a serem monitorados, organizados como um Banco de Dados de Atributos. O servidor recebe as solicitações do cliente e envia de volta respostas. Atua como um fornecedor de dados.

O GATT *client* está associado às funções do dispositivo *Link Layer Master* e *GAP Central*. Ele requer a presença e as características dos atributos em um servidor. O cliente envia solicitações ao servidor e recebe respostas. Atua como consumidor de dados e serviços.

A funcionalidade GATT de um dispositivo é logicamente separada da função *master/slave*. As funções de *master/slave* controlam como a conexão de rádio *BLE* é gerida e as funções de *client/server* são ditadas pelo armazenamento e fluxo de dados.

5. Desenvolvimento

5.1 Programação do TTGO T-Beam

5.1.1 Visão geral

O dispositivo responsável pela comunicação com a plataforma de rede *LoRaWAN* é o *TTGO T-BEAM*. É um dispositivo fabricado pela empresa *LILYGO*, construído em torno do microprocessador *ESP32*. O *TTGO T-BEAM* conta com todas as funcionalidades disponíveis do *ESP32*, *WiFi*, *BLE* (*Bluetooth Low Energy*), *LoRa*, *GPS*, etc.

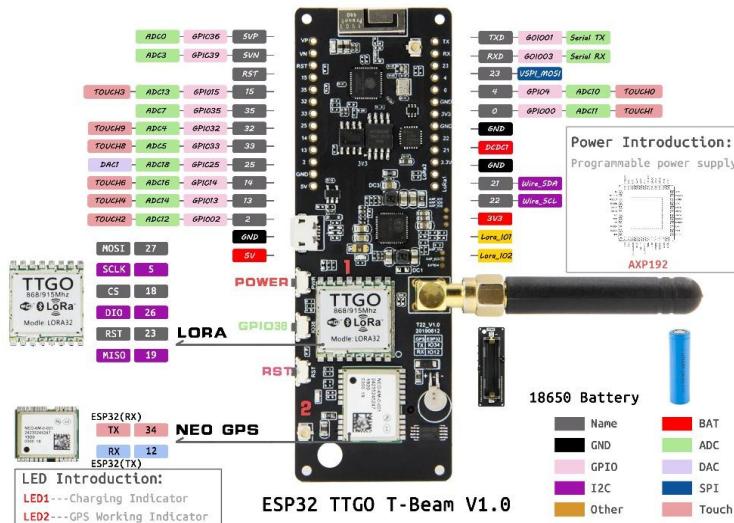


Figura 11 - Dispositivo *LILYGO® TTGO T-Beam V1.1 ESP32*
(retirado de [\[14\]](#))

5.1.2 Programação do dispositivo

Com o ambiente de desenvolvimento configurado e com o registo de uma aplicação na *The Things Network*, é realizada a programação do dispositivo *TTGO*. Com o recurso às bibliotecas fornecidas pela *Pycom*, permite programar o dispositivo para estabelecer a ligação com a rede *LoRaWAN* e estabelecer uma ligação através do *BLE* com a aplicação móvel.

O *TTGO* vai ter fundamentalmente:

- Funções de *uplink* e *downlink*
- Servidor *GATT*, e *callbacks* específicos para cada uma das características.

Para habilitar as funcionalidades de *LoRa* e *BLE* no *TTGO*, é necessário adicionar as bibliotecas adequadas ao ambiente de desenvolvimento. As seguintes bibliotecas são necessárias:

Biblioteca *LoRa*: A biblioteca *LoRa* ("from network import LoRa") fornece as funções e métodos necessários para a comunicação de dados usando a tecnologia *LoRa*. Na *LoRa* existem dois métodos de ligação que podem ser facilmente configuráveis: a *LoRaWAN ABP* (*Activation By Personalization*) e a *LoRaWAN OTAA* (*Over The Air Activation*).

- *ABP*, significa que as chaves criptográficas fornecidas pela *TTN* (*dev_addr*, *nwk_swkey*, *app_swkey*) são configuradas *hardcoded* (manualmente) no dispositivo e podem enviar dados para o *gateway* sem a necessidade de um procedimento de "*handshake*" para a troca de chaves (como é feito no método de conexão *OTAA*).
- *OTAA*, envia uma solicitação de *Join* para o *LoRaWAN Gateway* através das chaves (*dev_eui*, *app_eui*, *app_key*) fornecidas pela *TTN*. Se as chaves estiverem corretas, o *gateway* responderá com uma mensagem de aceitação de *join* e, a partir desse ponto, o dispositivo poderá enviar e receber dados de/para o *gateway*. Se as chaves estiverem incorretas, nenhuma resposta será recebida.

O método *lora.join(activation, auth, [timeout=None, dr=None])* permite estabelecer uma ligação com a rede *LoRaWAN*.

Biblioteca *socket*: A biblioteca *socket* ("import socket") fornece as funções necessárias para criar *LoRa raw sockets*. Um *socket* representa a interface de comunicação com o módulo de rádio *LoRa* do dispositivo. Esta interface familiar e conveniente permite enviar e receber dados, sem a necessidade de lidar diretamente com a linguagem de baixo nível do rádio.

Alguns dos métodos comuns disponíveis na classe de *socket LoRa* do *Pycom MicroPython*:

- *socket.send(bytes)*: Envia dados por *LoRa*.
- *socket.recv(bufsize)*: Recebe dados por *LoRa*.
- *socket.setblocking(flag)*: Define se as operações de envio e receção de dados são bloqueantes ou não bloqueantes.
- *socket.setsockopt(level, optname, value)*: Define as opções de configuração do *socket*.

Biblioteca *BLE*: A biblioteca *BLE* ("from network import Bluetooth") está desenhada para se conectar e comunicar entre dispositivos (em particular plataformas móveis). Na comunicação por *BLE*, o dispositivo *TTGO T-Beam* vai atuar como servidor para fornecer leituras e escritas de características que existem em cada serviço para o telemóvel. O telemóvel, por sua vez, atua como cliente e conecta-se ao *TTGO* para receber esses serviços.

Alguns dos métodos comuns disponíveis na classe de *Bluetooth* do *Pycom MicroPython* são:

Bluetooth:

- `bluetooth.start_scan(timeout)`
- `bluetooth.connect(mac_addr, [timeout=None])`
- `bluetooth.callback([trigger=None, handler=None, arg=None])`
- `bluetooth.events()`
- `bluetooth.set_advertisement([name=None, manufacturer_data=None, service_data=None, service_uuid=None])`
- `bluetooth.advertise([Enable])`
- `bluetooth.service(uuid, [isprimary=True, nbr_chars=1, start=True])`

Service: Os serviços são usados para categorizar os dados em blocos de dados designados de características. Um serviço pode ter várias características e cada serviço possui um *ID* numérico exclusivo (*UUID*).

- `service.characteristic(uuid, *, permissions, properties, value)`
- `service.characteristics()`

Characteristics: encapsula um único dado.

- `characteristic.callback(trigger=None, handler=None, arg=None)`

Biblioteca `_thread`: Esta biblioteca oferece primitivas de baixo nível para trabalhar com múltiplas *threads*.

- `_thread.start_new_thread(function, args[, kwargs]):` Inicia uma nova *thread* e devolve o identificador. A *thread* executa a função com a lista de argumentos `args`.

5.1.3 Programação das mensagens

Na *TTN*, as mensagens transmitidas têm uma secção que se chama *MAC (Media Access Control) payload* que podem ser interpretadas como os dados que são enviados nas mensagens *downlink* e *uplink* por parte do utilizador [\[27\]](#).

O *MAC payload* divide-se em duas partes importantes que contêm os dados que “realmente” se pretende obter por parte dos utilizadores [\[28\]](#):

- *Frame Payload* (ou apenas *payload*)
- *Frame Port* (também referido como *Fport*)

O *payload* terá os dados associados a uma característica. O *Fport* é usado como o identificador hexadecimal para o *bitmap*, para identificar a que característica o *payload* se refere [\[29\]](#). O seu valor, pode-se encontrar no intervalo [1, 223] [\[30\]](#). Na seguinte tabela, será possível observar o *bitmap* que associa cada característica a um *byte*, segundo o nome da variável criada no código para o *TTGO*.

Serviço_Caracterísica	Valor em hexadecimal
ID_USERDATA_STRING	0x3
ID_BATTERY_LEVEL	0x4
ID_LOCATION_LATITUDE	0x5
ID_LOCATION_LONGITUDE	0x6
ID_PHONE_ID	0x7
ID_BROADCAST_STRING	0x8
ID_LOCATION	0x9

Tabela 1 - Bitmap de características

A razão pela qual se começa a numeração por 0x3 é porque:

- O *Fport* 0x0 é inválido e só aparece quando um *uplink* ou *downlink* falhou
- O *Fport* é 0x1 quando é simulado um *uplink* ou *downlink* usando o *website* da *TTN*
- O *Fport* por *default* é igual a 0x2 quando é feito um *uplink* via o *TTGO*

Para se fazer *downlink*, é preciso um *uplink* como mencionado anteriormente, e esse *uplink* estará reservado para um dos *Fport* em cima consoante o campo que se pretende enviar.

5.1.3 Envio de bytes

Segundo as capacidades e termos de utilização da *TTN*, pode-se dizer que, por norma, pode-se enviar 51 bytes por *payload*, mas deve-se enviar o mínimo possível [\[31\]](#). Estes limites podem variar pela taxa de dados e país [\[32\]](#) [\[33\]](#).

5.1.4 Re却ão de bytes

Para se fazer um *downlink*, como se viu no capítulo anterior sobre a *LoRaWAN*, é necessário um *uplink* [\[34\]](#). Nota: entre se fazer *schedule* de um *downlink*, e ao *TTGO* fazer *downlink*, pode ser preciso esperar 5 a 10 segundos (segundo testes efetuados) para que o pedido de *downlink* não retorne vazio, nem sempre há garantia que corra bem [\[35\]](#).

5.1.5 Obtenção do sinal GPS

No dispositivo TTGO está integrado o módulo *NEO-6M GPS*, que permite obter informações de localização. O módulo *GPS NEO-6M* utiliza a tecnologia *GPS* para obter informações precisas de localização em tempo real. Para a obtenção das coordenadas geográficas é necessário ter em conta vários fatores. É essencial que o dispositivo esteja num espaço aberto para conseguir obter sinal de satélites suficientes, e assim determinar a sua posição. Perto do módulo *GPS NEO-6M* há um *LED* que indica o seu estado. Caso não esteja intermitente, significa que está à procura de satélites, e a piscar a cada 1s indica que a posição fixa foi encontrada (o módulo pode ver satélites suficientes). Para a programação do dispositivo na obtenção do sinal *GPS* foi utilizado código *MicroPython* e *open-source* pelo engenheiro Nuno Cruz.

5.2 Programação da aplicação Android

Em primeiro lugar, para uma boa realização da aplicação móvel, foram estabelecidos certos objetivos básicos:

- O desenho da *User-Interface (UI)* da aplicação *android*, deve ser: fácil de usar, apresentável e comunicativo para os seus utilizadores. No âmbito da programação, o código deve ser: fácil de interpretar, estruturado e bem documentado.
- Para a escolha do método de criação de *UI's*, é utilizado *Jetpack Compose* [36], que é uma *framework* recente realizada pela Google que simplifica e acelera o desenvolvimento da *UI* no *Android*.
- Os meios de teste da aplicação não foram via um emulador. Foi preciso usar um telemóvel real para que fosse possível a conexão com o *TTGO*.

5.2.1 Comunicação entre a aplicação Android e o T-Beam

Para comunicar com o *T-Beam*, o telemóvel precisa de ligar o *Bluetooth* e a sua localização. Quando a aplicação é instalada pela primeira vez, é pedido a permissão de acesso ao *GPS* e em determinados telemóveis é ainda pedido pela permissão de “Conexão com dispositivos próximos”, que representa a permissão de acesso ao *Bluetooth*. Quando é mostrado que o *Bluetooth* está ligado, na verdade as funcionalidades de *BLE* também estarão ligadas. A razão pela qual é necessário a localização tem a ver com o facto de que é necessário para se poder comunicar via *BLE*.

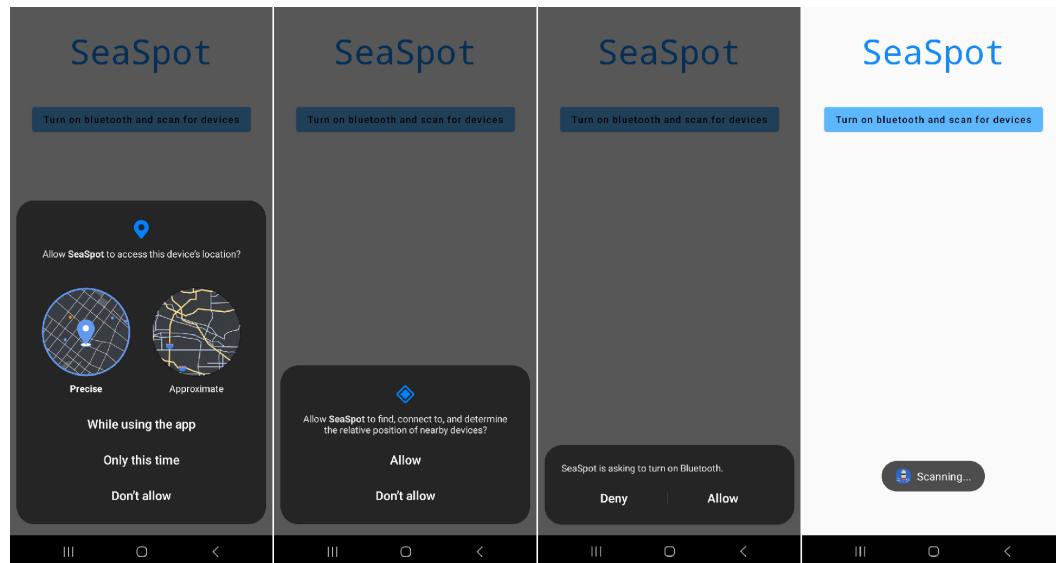


Figura 12 - Permissões e ligação de Bluetooth e Localização

Na página (*activity*) principal da *app*, existe um botão para ser feito um *scan* dos dispositivos *BLE* encontrados. Caso o *Bluetooth* ou localização estejam desligados, são mostrados meios próprios para se ligar diretamente estes dois componentes. Feito isto, será mostrado, após 5 segundos, todos os dispositivos encontrados na forma de uma lista de botões. No topo, aparecerá os dispositivos que tenham um nome, visto que é provável que sejam os mais relevantes.

Ao se carregar no botão é iniciada uma ligação ao servidor GATT. Se foi realizada com sucesso e se foram encontrados serviços, avança-se para o segundo ecrã, que listará os serviços na forma de *Card (Views)* e terá um botão no topo para o utilizador se desconectar. Ao se carregar num serviço, abrir-se-á um *Card (Views)* de características dentro do serviço. Se não forem editáveis, aparecerá só o valor da característica, caso contrário, o seu valor estará dentro de uma caixa de texto editável e um botão para aplicar a mudança.

Para ser feita uma tentativa de atualizar um dos dados disponibilizados pelo servidor GATT, foi criado um serviço especial designado de *ObjectTransfer*. Este serviço contém uma característica que está associado a um botão *Refresh/Atualizar*, o utilizador ao premir este botão irá permitir receber um *downlink* da TTN para o TTGO. Para atualizar o valor de uma característica, deve-se fechar e abrir o *Card* associado ao serviço. Ao atualizar este valor irá conter o *port* associado ao serviço onde foi recebido o *payload* contido no *downlink*.

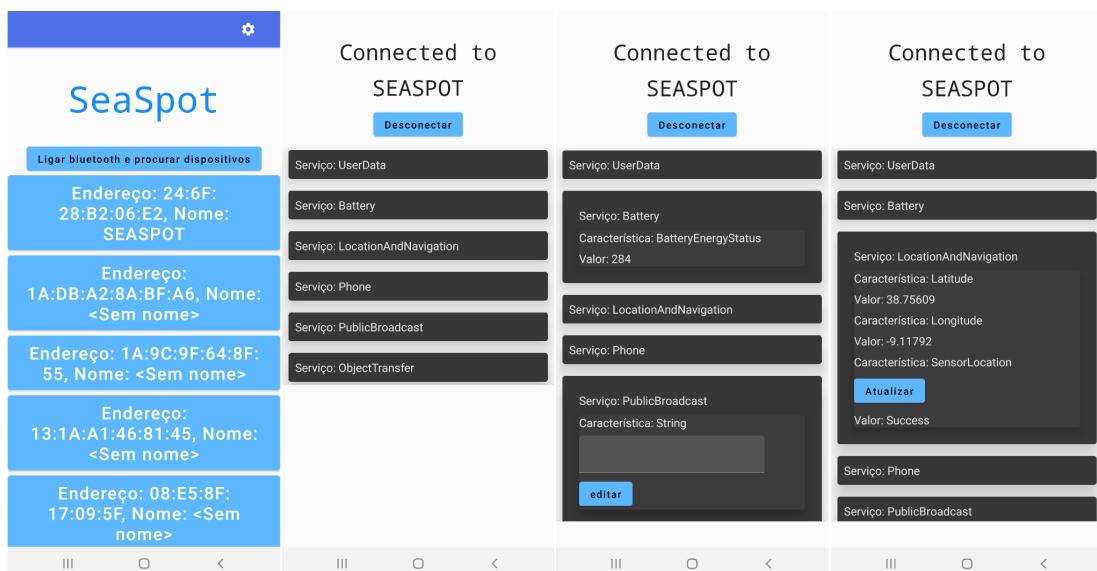


Figura 13 - Conexão, leitura e escrita de valores na *app*

5.2.2 Problemas e limitações da biblioteca BLE para Android

Entre a comunidade de programadores de *Android*, tem havido alguns problemas ou dificuldades com o uso da biblioteca *android.bluetooth.ble* [37]. É possível ter como exemplo de, por vezes, o *callback onServicesDiscovered(...)* indicar que a lista de serviços está vazia [38] [39]. Segundo os testes realizados, pode acontecer entre 15% a 40% das vezes. Por vezes, o ritmo da falha parece variar entre telemóveis. Uma solução proposta entre a comunidade de programadores de *Android* é criar um *delay* de 1 segundo entre a transição do estado *BONDED* (estabelecida entre os dispositivos) e a chamada à função *discoverServices* [51] [52], o que resultou. Outra explicação para este fenómeno seria a qualidade do *hardware* ou do *firmware* instalado ou a possibilidade de que em ambientes com muitos dispositivos *BLE*, possam ocorrer interferências.

Efetuar tarefas como ler a lista de características de um serviço não são diretamente possíveis com a biblioteca *Android*. Para tal, foi preciso criar *threads* (para não bloquear a *thread* principal) que chamam métodos auxiliares e bloqueantes (que foram criados para o efeito). Estes métodos fazem uso de *locks* e condições (da biblioteca *java.util.concurrent.locks*) para ser possível chamar os *callbacks* um a um, que são executadas noutras *threads* criadas pela instância do objeto *BluetoothGattCallback*.

Deste modo, foi criada uma solução em que as *threads* criadas pela aplicação comunicassem com as *threads* da biblioteca *BLE* do android, de forma a haver uma simples máquina de estados que permitisse a leitura correta e sincronizada de listas de características.

Para além disto, também foi necessário decidir que métodos usar, caso o nível da *API* do *Android* do telemóvel usado seja igual ou superior a 33 [40], que foi devidamente tratado avaliando o nível da *API* de *Android* do telemóvel cliente.

5.3 Programação da aplicação web

A aplicação *web* vai servir como uma alternativa mais prática e mais informativa que a aplicação móvel. Ao contrário da *app* móvel e da aplicação *TTN*, vai armazenar as mensagens numa base de dados.

5.3.1 Arquitetura da API

A *API* envolve 3 aspectos principais:

- O *webhook* que envia dados da *TTN* para a *API*
- Obtenção e eliminação de mensagens
- Consulta dos dados ou propriedades dos dispositivos registados

O *webhook* é configurado na página da aplicação, na *tab integrations* → *Webhooks*. É indicado o *URL* ao qual a *TTN* vai efetuar um pedido *HTTP* do tipo *POST*, com diversas informações no *body* e *headers* (que é enviado pela aplicação *TTN*). Na criação do *webhook* é preciso especificar uma *API key* da *TTN*(esta *API key* foi gerada anteriormente pela *TTN*) e os tipos de eventos que vão desencadear esta chamada, sendo neste caso, quando há um *uplink* para a aplicação.

Por exemplo, para o campo *BaseURL* é indicado “<https://X/api/uplink>”. Em que “X” é, por exemplo, um endereço público na internet, seja um endereço *ngrok* [41] ou uma aplicação que foi *deployed* via um hospedeiro de *websites* numa *cloud* (como o *Heroku* ou *Azure*). Os *uplinks* cujo *Fport* diferentes daquele da tabela 1 são ignorados, logo essas mensagens não são guardadas na base de dados.

Do lado da *API*, é verificado se o cliente que accede a este *endpoint* têm o header “x-downlink-apikey” igual ao gerado na *TTN* e usado pelo *webhook*. Com a intenção de evitar clientes de adicionarem mensagens sem autenticação nem autorização.

Na figura abaixo, é mostrado os caminhos da *API* desenvolvida. Foi utilizado o *Swagger Editor* para gerar a documentação.

The screenshot shows the Swagger Editor interface with the following sections:

- Uplink**:
 - POST /uplink**: A path that only TTN can call, via its webhook. It stores the uplink message in our DB.
- Messages**:
 - GET /messages**: Obtains message feed, accepts query params
 - DELETE /messages/all/{id}**: Deletes all messages of a specific device
 - GET /messages/{id}**: Obtains a specific message
 - DELETE /messages/{id}**: Deletes a specific message
- Devices**:
 - GET /devices/{id}**: Obtains information about a specific device

Figura 14 - Caminhos da API

Relativamente aos pedidos que a *API* suporta, é possível obter várias mensagens (suporta paginação e filtrar por característica) ou eliminar todas as mensagens e obter ou eliminar uma mensagem específica. E é possível consultar os dados de um dispositivo específico, que mostra as propriedades com os valores mais recentes para cada propriedade.

De forma a que a *API* desenvolvida guarde corretamente as mensagens enviadas para o *path* dos *uplinks*, é usado o valor do *f_port* (*Frame Port*) que vem no *body* para identificar a que característica a mensagem se refere.

A *API* da *TTN* também suporta o agendamento de *downlinks* via *HTTP* [42]. Tal é feito, assim que é feito um *uplink*. Isto vai permitir que qualquer dispositivo *LoRa* envolvido na arquitetura, possa fazer *downlink* de mensagens enviadas por outros dispositivos (o próprio que enviou, se quiser, também o pode fazer). Mas esta funcionalidade não está atualmente ligada porque só faz sentido estar se tivermos vários *TTGO*'s. Em relação à forma como adicionamos a mensagem à fila, escolhemos fazer *replace* ao invés de *push* para que o utilizador do *TTGO* não tenha que fazer múltiplos *downlinks* para ter a mensagem mais recente. Logo, assim que o *TTGO* faz um *downlink*, lê sempre a mensagem mais recente.

5.3.2 Estrutura do servidor

Relativamente à forma como foi organizado o servidor, houve uma abordagem de injeção de implementações (ou dependências), em que as camadas de mais alto nível do código chamam camadas inferiores (na forma de funções). Também foram adotadas normas e nomes intuitivos para se compreender facilmente a estrutura. Na seguinte figura é ilustrada a estrutura das diretórias do código.

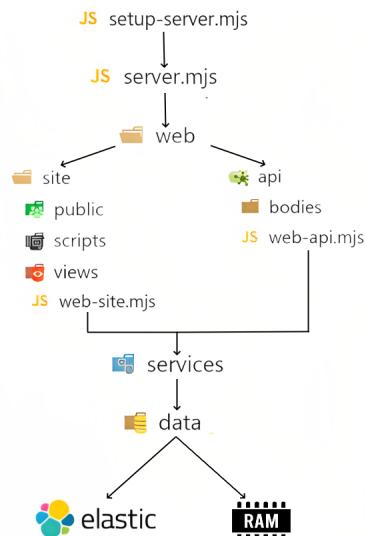


Figura 15 - Estrutura da *WebApp*

Começando da parte superior da figura, o ficheiro “*setup-server.mjs*” configura as definições gerais de inicialização do servidor, mais concretamente o

porto em que o servidor vai usar, e a configuração se o repositório de dados é em memória ou via *Elastic Search*.

Na pasta “web”, existem 2 subpastas: “site” e “api”. A pasta “api” contém os *handlers* para os pedidos que sejam feitos à *API*, a definição dos *bodies* e outros parâmetros e funções que interpretam e validam os pedidos feitos. Na pasta “site”, encontram-se as rotas válidas para os clientes que visitam o *Website* via *browser* assim como a declaração das vistas que são processadas do lado do servidor (*server-side rendered HTML*) e os *scripts* cliente. Ambas as rotas anteriores usam os serviços, que servem como um segundo filtro relativamente à lógica da leitura e escrita de dados. Para atender aos variados requisitos, existe uma camada de acesso a dados que tem em conta a configuração mencionada anteriormente. Essa abordagem permitiu lidar com as especificidades de cada requisito de forma adequada, garantindo um projeto bem estruturado e funcional. A existência de uma implementação da gestão dos dados em memória justifica-se no contexto de testes iniciais (uma vez que foi mais fácil de usar e programar) e também porque fortalece a compreensão da gestão dos dados.

5.3.2 Modelo de dados

A base de dados guarda duas entidades: Mensagens e Dispositivos. Nas seguintes imagens está ilustrado este modelo de dados com alguns valores como exemplo.

```
{
  "id": "33e9fbbd-5fa4-42ab-abd6-6fb5ffa415a9",
  "messageObj": {
    "applicationId": "ttgo-test-g10",
    "endDeviceId": "eui-70b3d57ed005bfb0",
    "deviceAddress": "260B893E",
    "location": {
      "latitude": {
        "value": 38.7565362672383,
        "id": {
          "code": 5,
          "inString": "Latitude"
        }
      },
      "longitude": {
        "value": -9.11603538108787,
        "id": {
          "code": 6,
          "inString": "Longitude"
        }
      }
    },
    "serviceCharacteristic": {
      "code": 8,
      "inString": "Broadcast string"
    },
    "payload": "61 62 63",
    "receivedAt": "2023-05-28T19:27:27.931Z"
  }
}
```

Figura 16 - Objeto *Message*

```
{  
    "id": "eui-70b3d57ed005bfb0",  
    "deviceObj": {  
        "applicationId": "ttgo-test-g10",  
        "deviceAdress": "260B893E",  
        "location": {  
            "latitude": {  
                "value": 38.7562409625148,  
                "id": {  
                    "code": 5,  
                    "inString": "Latitude"  
                }  
            },  
            "longitude": {  
                "value": -9.117382657933856,  
                "id": {  
                    "code": 6,  
                    "inString": "Longitude"  
                }  
            }  
        },  
        "name": {  
            "value": "TTGO ESP32",  
            "id": {  
                "code": 3,  
                "inString": "Userdata string"  
            }  
        },  
        "batteryEnergy": {  
            "value": 248,  
            "id": {  
                "code": 4,  
                "inString": "Battery energy"  
            }  
        },  
        "phone": {  
            "value": "+351 960 000 000",  
            "id": {  
                "code": 7,  
                "inString": "Phone"  
            }  
        },  
        "string": {  
            "value": "Olá SOS",  
            "id": {  
                "code": 8,  
                "inString": "Broadcast string"  
            }  
        },  
        "latestUpdate": "2023-06-02T19:27:27.931Z"  
    }  
}
```

Figura 17 - Objeto Device

5.3.4 Obtenção do mapa

Para mostrar um mapa no *website*, foi usada a biblioteca *open source* Leaflet [13]. A biblioteca é carregada uma única vez para o *browser* do cliente na primeira vez que este acede à página, na forma de ficheiros *Javascript* (.js) e *Cascading Style Sheets* (.css). Sempre que o cliente navega no mapa, são feitos pedidos *GET* que contém partes do mapa em imagens .png que são usadas para formar o mapa.

Nas páginas onde se pretende ter um mapa, é inserido um elemento *HTML* <div> com o ID “map” e um *style* para posicionar o mapa na página. De seguida a página irá executar um *script* que chama a função *showMap()* que pertence ao par de *scripts* enviados do servidor para o cliente (que se encontra na pasta /web/site/scripts/), passando como parâmetro a latitude e a longitude. Este *script* usufrui da biblioteca *Leaflet*, que é também carregada anteriormente para o *browser* do cliente.

5.3.5 Interface cliente

Na barra superior da página, pode-se carregar na *tab* Messages para se navegar para a página que lista as mensagens, ordenadas da mais recentes para menos recentes. Ao se carregar numa mensagem, o utilizador navega para a página específica da mensagem, visualizando o seu conteúdo. Ao se carregar no identificador do dispositivo, navega-se para a página específica do dispositivo. É também possível eliminar todas as mensagens ou mensagens específicas.

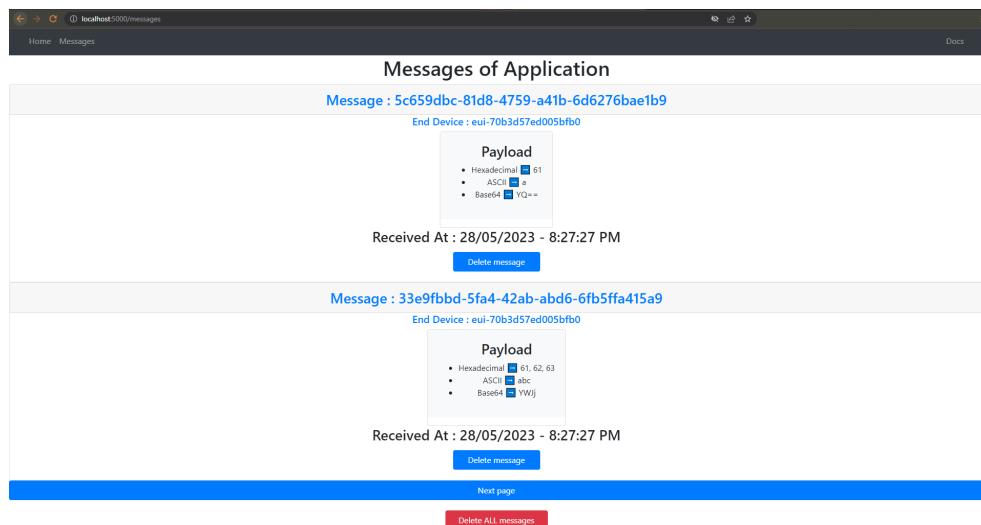


Figura 18 - Página da lista de mensagens

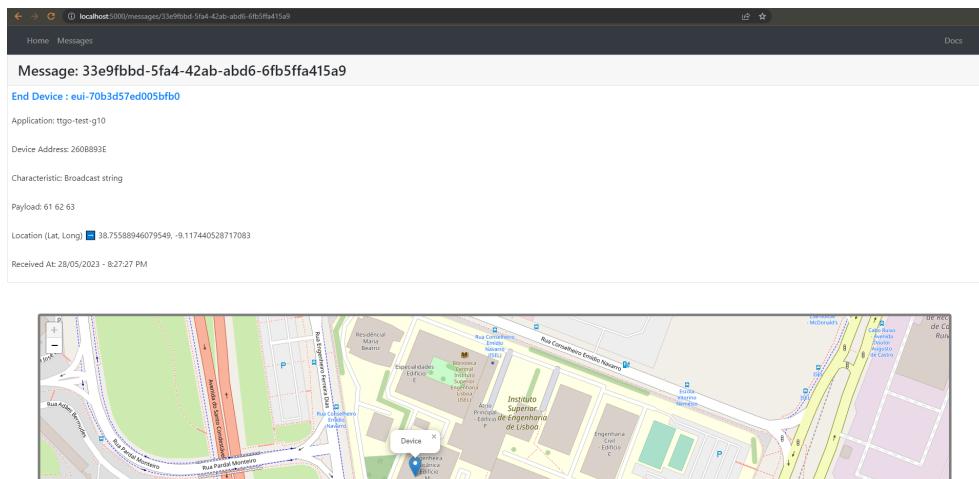


Figura 19 - Página de uma mensagem

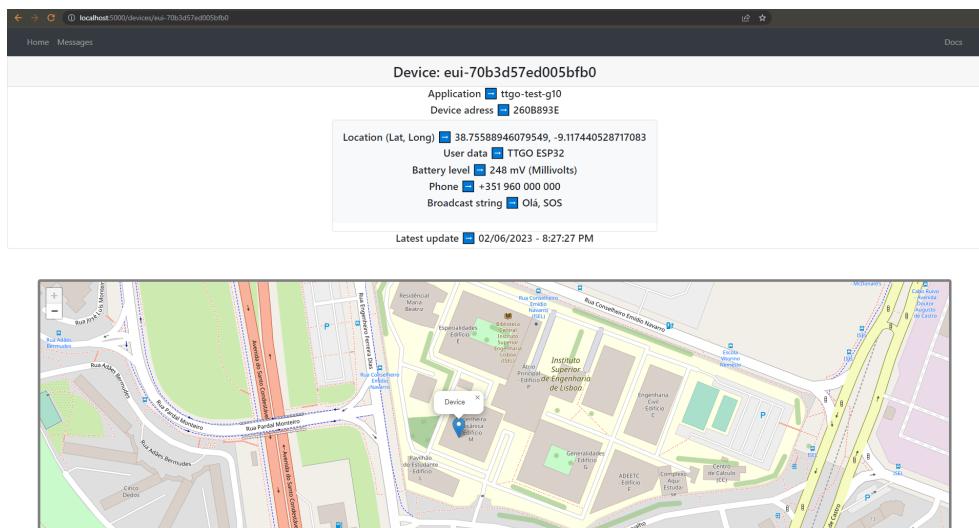


Figura 20 - Página de um dispositivo

5.4 Demonstração e testes numa situação real

Abaixo é mostrado o *TTGO* e um *Samsung Galaxy A12*. Também testou-se com um *Galaxy A50*.

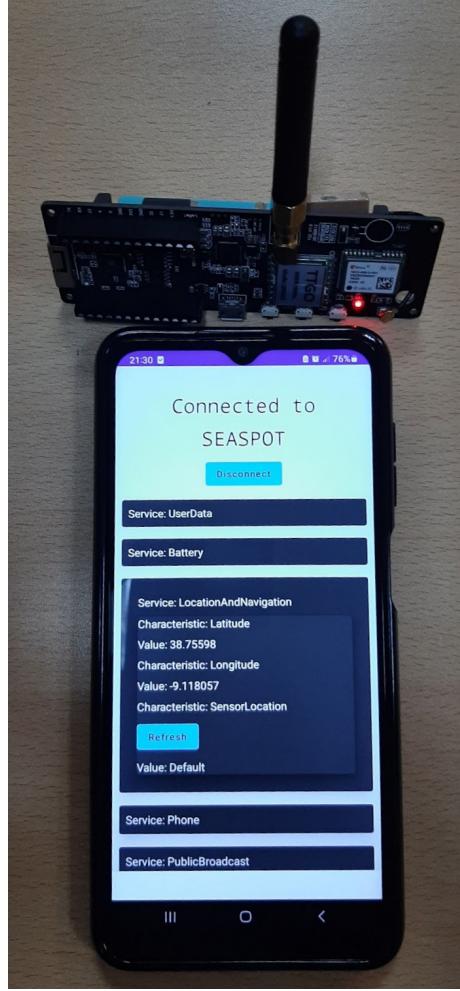


Figura 21 - *TTGO* e telemóvel com a *app* conectada ao *TTGO*

Relativamente à distância entre o *TTGO* e o telemóvel, foi possível detetar e efetuar uma ligação ao *TTGO* a uma distância entre 10 a 15 metros. Com piores condições, não foi possível estabelecer uma ligação ao *TTGO*.

Para testar os *webhooks*, foi utilizado a *tab de messaging* da aplicação, que tem uma interface que permite simular *uplinks* e fazer o agendamento de *downlink*. De forma a hospedar a *WebApp* foi utilizado o *ngrok*, que expõe o servidor que está no *localhost* utilizado durante o desenvolvimento e no porto escolhido (9000) num endereço público na *internet* que a aplicação *TTN* pode aceder.

É relevante notar que o *LED* azul (perto do centro) do *TTGO* só é ligado quando este está ligado a uma fonte de alimentação USB Micro C. O *LED* vermelho só estará ligado quando o GPS também estiver ligado (que se encontra perto do canto superior esquerdo na figura).

6. Conclusão

Neste trabalho, fomos capazes de desenvolver soluções de modo a concluir a ideia de projeto que nos foi proposta. À primeira vista, havia algumas dúvidas relativamente à concretização e ao resultado final pretendido, mas depois de investigar as capacidades e funcionalidades das tecnologias, conseguimos usufruir destas para ter um sistema funcional e útil. Ao longo do projeto enfrentamos uma variedade de requisitos com diferentes níveis de especificidade, o que nos proporcionou oportunidades e desafios únicos. A abertura desses requisitos permitiu uma maior flexibilidade na nossa abordagem, mas, ao mesmo tempo, exigiu que fossem tomadas decisões no desenho da arquitetura e protocolos. Essa dinâmica nos motivou a encontrar soluções criativas e planejadas para garantir o sucesso do projeto. Por exemplo, relativamente aos dados que podem ser transferidos e o sentido que fazem (via a especificação *GATT* e o *GATT server* criado), no seu aspeto visual e o facto de podermos mostrar num mapa a localização atual do dispositivo e a localização da origem das mensagens. Foram usadas diversas tecnologias, linguagens e bibliotecas. Umas usadas no curso *LEIC*, outras não, por exemplo a linguagem *MicroPython* e a utilização da infraestrutura *LoRaWAN*.

A maior dificuldade foi a programação do *TTGO*, porque houve muita experimentação, e foi preciso estar perto de um *gateway* (no *ISEL*, por exemplo) para trabalhar com o dispositivo. Também houve alguma dificuldade na programação da *app* na parte de utilizar a biblioteca *BLE* disponibilizada pelo *Android*.

Em suma, podemos dizer que conseguimos alcançar os objetivos deste projeto, e no caminho, superamos vários desafios e aprendemos muitas coisas novas. Mais uma vez, agradecemos aos orientadores, ao professor de Projeto e Seminário e a todos os professores que conhecemos e que nos ensinaram ao longo do curso.

6.1 Críticas construtivas

- Podia ter sido usada uma base de dados mais leve (em termos da RAM e espaço ocupado no disco), como o *PostgreSQL*.
- O código da aplicação *Android* podia estar melhor organizado.
- *API* e aplicação *web* podia ter mais funcionalidades.
- Deviam ser estabelecidos limites sobre as dimensões das mensagens e a forma de compressão. Tanto na base de dados, como nas características.

6.2 Pontos fortes

- Simplicidade das *user interfaces*, tanto no lado móvel como na aplicação *web*
- O código e a estrutura da aplicação *web* está muito boa. Foi evitado o *hardcoding* de strings e valores numéricos (exceto no essencial), foram usadas classes para representar objetos do domínio, há muitas funções utilitárias e dinâmicas. Também foi usado o JSDoc que documenta os tipos recebidos como parâmetro nas funções e classes. Isto evita erros durante a programação e leitura do código.
- Todas as decisões, referências estão devidamente explicadas. Seja por escrito ou por diagramas.
- Foi um projeto muito diversificado. Ou seja, aplicamos conhecimentos de 3 áreas de informática, designadamente nas áreas de programação de uma aplicação *full-stack*, uma aplicação móvel e a programação de um dispositivo *IoT* (ou de *hardware*).

Referências

Todos os *links* listados estão funcionais. Confirmado até à data 10-7-2023

- [1] <https://www.semtech.com/lora/what-is-lora>
- [2] <https://lora-alliance.org/about-lorawan/>
- [3] <https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-low-energy/>
- [4] <https://developers.mydevices.com/cayenne/lora/>
- [5] <https://www.instructables.com/Android-TTGO-T-Beam-GPS-Tracker/>
- [6] http://www.lilygo.cn/claprod_view.aspx?TypId=62&Id=1401&Fld=t28:62:28
- [7] <https://micropython.org/>
- [8] <https://expressjs.com/>
- [9] <https://handlebarsjs.com/>
- [10] <https://www.elastic.co/>
- [11] <https://code.visualstudio.com/>
- [12] <https://developer.android.com/studio>
- [13] <https://leafletjs.com/>
- [14] <https://lora-developers.semtech.com/documentation/tech-papers-and-guides lorawan-class-a-devices/>
- [15] <https://www.thethingsnetwork.org/docs/lorawan/classes/>,
<https://www.thethingsindustries.com/docs/devices/class-c/>
<https://www.thethingsindustries.com/docs/devices/class-b/>
- [16] https://researchgate.net/figure/LoRaWAN-Class-A-transaction-The-transaction-is-node-initiated-uplink-direction-the_fig2_334429577
- [17] <https://www.thethingsnetwork.org/docs/lorawan/limitations/>
- [18] <https://www.thethingsnetwork.org/>
- [19] <https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/>
- [20] <https://www.thethingsindustries.com/docs/reference/ttn/#clusters>
- [21] <https://www.bluetooth.com/learn-about-bluetooth/tech-overview/>
- [22] <https://pcng.medium.com/ble-protocol-stack-controller-2d2d5371deec>
- [23] <https://microchipdeveloper.com/wireless:ble-link-layer-roles-states>
- [24] <https://www.bluetooth.com/specifications/assigned-numbers/>
- [25] <https://stackoverflow.com/a/62345071>
- [26] <https://microchipdeveloper.com/wireless:ble-gatt-data-organization>

[27]

<https://www.thethingsnetwork.org/forum/t/why-ttn-call-the-user-payload-a-mac-payload/56071>

[28] <https://www.thethingsnetwork.org/docs/lorawan/message-types/#data-messages>

[29] <https://www.thethingsnetwork.org/forum/t/fport-correct-usage/33388>

[30]

<https://www.thethingsnetwork.org/docs/lorawan/message-types/#sending-mac-commands-and-application-specific-data-in-the-frmpayload-field>

[31]

<https://www.thethingsnetwork.org/docs/devices/bytes/#how-many-bytes-can-i-send>

[32]

<https://lora-developers.semtech.com/documentation/tech-papers-and-guides/the-book/packet-size-considerations/>

[33]

<https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/#eu863-870-maximum-payload-size>

[34]

<https://www.thethingsnetwork.org/forum/t/problem-with-sending-downlinks-before-the-first-uplink/60840/2>

[35]

<https://www.thethingsnetwork.org/forum/t/how-to-get-downlink-communication/41712/4>

[36] <https://developer.android.com/jetpack/compose>

[37]

<https://medium.com/@martijn.van.welie/making-android-ble-work-part-1-a736dcd53b02#:~:text=So%20why%20is%20BLE%20on%20Android%20so%20hard%3F>

[38] <https://issuetracker.google.com/issues/228984309>

[39] <https://stackoverflow.com/q/32363931>

[40]

[https://developer.android.com/reference/android/bluetooth/BluetoothGattCharacteristic#setValue\(byte\[\]\)](https://developer.android.com/reference/android/bluetooth/BluetoothGattCharacteristic#setValue(byte[]))

[41] <https://ngrok.com/>

[42]

<https://www.thethingsindustries.com/docs/integrations/webhooks/scheduling-downlinks/#scheduling-downlinks>

[43]

<https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification>

[44] Townsend, K., Cufí, C., Akiba, A., & Davidson, R. (2014). Getting Started with Bluetooth Low Energy. O'Reilly -

<https://www.oreilly.com/library/view/getting-started-with/9781491900550/#:~:text=Product%20information-,Table%20of%20contents,-Preface>

[45] <https://github.com/nunomcruz/ttgo-tbeam-gps-reset-python>

[46] <https://docs.pycom.io/firmwareapi/pycom/machine/uart/>

- [47] <https://github.com/inmcm/micropyGPS>
- [48] <http://aprs.gids.nl/nmea/>
- [49] <https://github.com/nunomcruz/ttgo-lorawan-probe>
- [50] <https://www.thethingsnetwork.org/docs/applications/ttnmapper/>
- [51] <https://stackoverflow.com/a/41526267>
- [52]
<https://medium.com/@martijn.van.welie/making-android-ble-work-part-2-47a3cdaade07#:~:text=to%20add%20a-,1000,-%E2%80%931500%20ms%20delay>