ESTGOH
Escola Superior de Tecnologia e Gestão
de Oliveira do Hospital
Instituto Politécnico de Coimbra

# Sistema de obtenção de dados e controlo remoto de contadores inteligentes

**Relatório apresentado com vista à obtenção do grau de Mestre no âmbito da realização do ciclo de estudos de Mestrado em Informática Aplicada**

**Autor:**
Diogo Novais

**Orientador:**
Professor Francisco Afonso

**Coorientador:**
Professor António Paulino

**Data:** Outubro de 2019

# Agradecimentos

Começo por agradecer aos meus orientadores, Professor Francisco Afonso e Professor António Paulino, pelo apoio, pelo acompanhamento ao longo do estágio e pela disponibilidade, e em particular ao Professor Francisco Afonso pela ajuda dada na revisão do relatório.

Agradeço ao meu supervisor de estágio, Jorge Saraiva, pelo apoio, por me receber na empresa, pela disponibilidade e por me ter ajudado a superar problemas que surgiram durante o desenvolvimento do projeto.

Agradeço aos meus colegas de trabalho, em particular aos colegas José Marques e Fábio Monteiro com os quais desenvolvi este projeto em conjunto, pelo espirito de equipa e pela disponibilidade, e aos colegas Marco Mota e Fábio Marques pelo apoio e pelo conhecimento transmitido.

Agradeço aos meus colegas de turma pela amizade e entreajuda, e em particular à colega Liliana Oliveira por me lembrar dos prazos e por não me deixar perder o foco nas tarefas.

Agradeço aos meus amigos pelo apoio e por me terem ajudado a descontrair durante os momentos mais difíceis.

Por fim agradeço em especial à minha família pelo apoio dado ao longo destes anos, e em particular aos meus pais, irmãos e avós por estarem presentes, pela ajuda e pela motivação.

# Resumo

Atualmente, ainda é comum que as empresas distribuidoras de água, eletricidade e gás utilizem contadores que necessitam que um funcionário se desloque ao local para obter leituras. A aquisição destas leituras presencialmente é uma tarefa demorada e com a possibilidade de falhar de várias maneiras, como por exemplo, se o contador estiver dentro de um prédio e no momento da leitura não houver ninguém para abrir a porta. Este relatório apresenta um sistema que foi desenvolvido durante o estágio na empresa TULAlabs e que permite ler, monitorizar, controlar e configurar contadores remotamente. Serão apresentados todos os componentes do sistema, tais como bases de dados, *web services*, *website* de administração, componentes de apoio á rede sem fios, *gateways* e os contadores inteligentes. Inicialmente o sistema foi desenvolvido com o objetivo de obter leituras de contadores de água, mais tarde foi generalizado para suportar qualquer tipo de contador que reporte dados numéricos. Adicionalmente o sistema foi generalizado para controlar qualquer tipo de contador. O sistema foi desenvolvido com sucesso e satisfaz quase todos os requisitos definidos pelo cliente. Além disto, diversas outras funcionalidades não inicialmente especificadas foram desenvolvidas.

## Palavras-chave

Contadores inteligentes, Internet das coisas, Máquina a máquina.

# Abstract

Currently it is still common for water, electricity, and gas public utilities to use meters which require that an employee goes to the site to acquire readings. The acquisition of this readings in person is a time consuming task which has the possibility to fail in many ways, for example if the meter is inside a building and at the reading moment there is no one to open the door. This report presents a system developed during the internship at TULAlabs which can read, monitor, control and configure meters remotely. All system components such as databases, web services, administration website, wireless network support components, gateways and smart meters will be presented. Initially the system was developed with the purpose of getting readings from water meters, later the system was generalized to support any kind of meter which reports numeric data. Additionally the system was generalized to control any kind of meter. This system was developed successfully and satisfies almost all requirements defined by the client. In addition, several other functionalities not initially specified were developed.

## Keywords

Smart meters, Internet of things, Machine to machine.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Siglas e Acrónimos

| | |
|---|---|
| **6LoWPAN** | IPv6 over Low-Power Wireless Personal Area Networks |
| **ACL** | Access Control List |
| **AES** | Advanced Encryption Standard |
| **AJAX** | Asynchronous JavaScript And Xml |
| **ALOHA** | Additive Links On-line Hawaii Area |
| **API** | Application Programming Interface |
| **ASP** | Active Server Pages |
| **BLE** | Bluetooth Low Energy |
| **BMS** | Battery Management System |
| **CA** | Certificate Authority |
| **CPU** | Central Processing Unit |
| **CRUD** | Create, Read, Update, Delete |
| **CSMA/CA** | Carrier-Sense Multiple Access with Collision Avoidance |
| **CSRF** | Cross-Site Request Forgery |
| **CSS** | Cascading Style Sheets |
| **DNS** | Domain Name System |
| **EAV** | Entity-Attribute-Value |
| **EF** | Entity Framework |
| **EUI** | Extended Unique Identifier |
| **FTP** | File Transfer Protocol |
| **GPIO** | General-Purpose Input/Output |
| **GUID** | Globally Unique IDentifier |
| **HTML** | HyperText Markup Language |
| **HTTP** | HiperText Transfer Protocol |
| **HTTPS** | HiperText Transfer Protocol Secure |
| **IDE** | Integrated Development Environment |
| **IIS** | Internet Information Services |

| | |
|---|---|
| **IP** | Internet Protocol |
| **IoT** | Internet of Things |
| **JSON** | JavaScript Object Notation |
| **LoRa** | Long Range |
| **LoRaWAN** | Long Range Wide Area Network |
| **M2M** | Machine to Machine |
| **MAC** | Medium Access Control |
| **MQTT** | Message Queuing Telemetry Transport |
| **MSB** | Most Significant Bit |
| **MVC** | Model View Controller |
| **ORM** | Object-Relational Mapping |
| **PAT** | Port Address Translation |
| **PNG** | Portable Network Graphics |
| **POCO** | Plain Old Clr Object |
| **QoS** | Quality of Service |
| **RAM** | Random-Access Memory |
| **RDP** | Remote Desktop Protocol |
| **REST** | REpresentational State Transfer |
| **RPC** | Remote Procedure Call |
| **RSA** | Rivest–Shamir–Adleman |
| **RSSI** | Received Signal Strength Indicator |
| **SCADA** | Supervisory Control And Data Acquisition |
| **SFTP** | SSH File Transfer Protocol |
| **SGBD** | Sistema Gestor de Bases de Dados |
| **SHA** | Secure Hash Algorithm |
| **SP** | Stored Procedure |
| **SPI** | Serial Peripheral Interface |
| **SQL** | Structured Query Language |
| **SSH** | Secure SHell |

| | |
|---|---|
| **SVG** | Scalable Vector Graphics |
| **TCP** | Transmission Control Protocol |
| **TLS** | Transport Layer Security |
| **UDP** | User Datagram Protocol |
| **URL** | Uniform Resource Locator |
| **VPN** | Virtual Private Network |
| **WCF** | Windows Communication Foundation |
| **XML** | eXtensible Markup Language |

# 1. Introdução

A empresa onde realizei o estágio foi a TULAlabs, esta empresa foca-se em desenvolver soluções tecnológicas personalizadas para diversas áreas, tais como telecomunicações, segurança, automação, mobilidade, sistemas inteligentes de transporte e industria. Esta empresa desenvolve, configura, instala e efetua manutenção de várias soluções de *hardware* e *software*, enquadrando-se no plano de estudos do mestrado em informática aplicada, na medida em que a maioria das disciplinas estão relacionadas com o desenvolvimento de *software* e instalação e configuração de redes de computadores.

## 1.1. Motivação

A medição do consumo de água presencialmente é uma tarefa que requer não só a deslocação de funcionários mas também acesso aos contadores, que por vezes estão instalados no interior de edifícios, muros, etc., estando inacessíveis caso não esteja ninguém presente para permitir o acesso a esses contadores no momento em que o funcionário tenta obter a leitura. Em caso de incumprimento do pagamento da fatura dentro dos prazos estipulados, a empresa distribuidora de água tem que enviar um funcionário para interromper o fornecimento de água enquanto o pagamento não for regularizado e quando o pagamento for regularizado a empresa tem que enviar um funcionário para restaurar o fornecimento de água. Por estes e outros motivos o processo de medição do consumo de água e gestão do fornecimento presencialmente pode tornar-se dispendioso e as operações podem tornar-se lentas tanto para a empresa como para o cliente.

A utilização de contadores “inteligentes” torna o processo de medição de consumo de água mais eficiente e económico para a empresa que distribui água, sendo possível obter remotamente medições do consumo de água em intervalos de tempo mais pequenos do que um mês, sem depender de acesso físico ao contador, permite que o fornecimento de água seja interrompido e restaurado remotamente quando necessário e torna a fraude das leituras de mais difícil. Os contadores inteligentes também têm vantagens para o cliente. Este pode receber uma faturação mais detalhada de modo a poder tirar conclusões sobre o consumo de água e alterar hábitos de consumo para reduzir os custos, acaba com as faturas estimadas que raramente correspondem ao consumo real e protege o cliente, evitando que este “abra a porta” a ladrões que se fazem passar por funcionários da empresa distribuidora de água.

Este projeto foi desenvolvido pela TULAlabs como solução personalizada para um cliente, o cliente em questão é uma empresa distribuidora de água. Quando o projeto foi iniciado a empresa possuía contadores de água digitais instalados em alguns dos seus clientes, estes contadores tornavam a obtenção da leitura mais rápida, comparando com o método em que o funcionário copiava manualmente o valor mostrado no contador, e registavam alguns tipos de alertas, tais como a deteção de campos magnéticos fortes (possível tentativa de bloquear o contador) ou de fluxo de água reverso (possível tentativa de parar ou diminuir o consumo registado pelo contador). Estes contadores armazenam leituras de consumo de água e alertas ativados na memória interna, desde a última vez em que um funcionário visitou o contador e descarregou a memória para o seu terminal móvel, mas têm um limite do número de leituras e alarmes que podem ser

armazenados na memória interna e necessitam que o funcionário de desloque aos locais onde os contadores estão instalados para descarregar a memória interna.

## 1.2. Objetivos

Este projeto consiste no desenvolvimento de contadores inteligentes que permitem efetuar operações remotamente, tais como obter leituras do consumo de água, interromper e resumir o fornecimento de água quando necessário, emitir alertas entre outras funcionalidades que serão apresentadas nos próximos capítulos. Os contadores inteligentes comunicam sem fios com *gateways* colocados em locais que permitem cobrir grandes áreas geográficas, os *gateways* comunicam com servidores que gerem a rede sem fios e esses servidores por sua vez comunicam com o *software* desenvolvido. O *software* desenvolvido permite armazenar leituras, alertas, estados das válvulas, estados das baterias, entre outros tipos de mensagens transmitidas pelos contadores numa base de dados, controlar as válvulas dos contadores, configurar os contadores e os *gateways* entre outras operações que serão apresentadas nos próximos capítulos. Um dos requisitos era que o sistema suportasse contadores que reportam leituras em quatro níveis de precisão: litros, dezenas de litros, centenas de litros e milhares de litros (ou metros cúbicos). Para implementar este requisito foi dada a opção ao utilizador de configurar o tipo de dados que o sistema deve esperar do contador, mais tarde reparei que se permitisse configurar mais do que um sensor por contador o sistema poderia suportar leituras de outros tipos de contadores, tais como contadores elétricos, generalizando este conceito atualmente o sistema permite obter, processar, armazenar e consultar dados numéricos temporais de qualquer tipo de dispositivo. Esta flexibilidade foi expandida aos comandos dos contadores, sendo possível configurar novos comandos para todos os tipos de dispositivos, e para os alertas, sendo possível configurar novos alertas para todos os tipos de dispositivos.

## 1.3. Organização

Este relatório está organizado em doze capítulos, de seguida será apresentada uma breve descrição de cada capítulo:

- O capítulo 2 apresenta outros sistemas semelhantes e mostra as diferenças entre esses sistemas e o sistema apresentado neste relatório.
- O capítulo 3 apresenta o plano de tarefas do projeto e a as tarefas desse plano que foram realizadas.
- O capítulo 4 apresenta os objetivos que se pretende atingir com a realização do projeto, o que foi feito para atingir esses objetivos, qual foi a minha contribuição para a realização do projeto e os métodos aplicados para realizar tarefas recorrentes.
- O capítulo 5 apresenta os componentes que compõem o sistema e o *software* e *hardware* utilizado para implementar o sistema.
- O capítulo 6 apresenta os componentes de *hardware* utilizados para implementar o sistema em mais detalhe, as suas funções e as suas especificações.
- O capítulo 7 apresenta os componentes de *software* utilizados para implementar o sistema, quais as suas funções, o modo como alguns desses componentes foram desenvolvidos e configurados e o modo como estes componentes comunicam entre si.

- O capítulo 8 apresenta os protocolos que foram utilizados para efetuar a comunicação entre os componentes de *software*, com especial atenção ao protocolo de rádio que foi desenvolvido de raiz, e alguns exemplos de mensagens.
- O capítulo 9 apresenta as principais funcionalidades do sistema que foram implementadas, sendo essas funcionalidades as que foram definidas pelo cliente e as que foram adicionadas de modo facilitar a gestão do sistema ou expandir a sua utilidade.
- O capítulo 10 apresenta as medidas que foram tomadas para tornar a comunicação entre os vários componentes do sistema segura e as medidas que foram tomadas parar tornar os próprios sistemas seguros.
- O capítulo 11 apresenta os testes que foram efetuados com o objetivo de encontrar e resolver problemas e garantir o funcionamento contínuo do sistema.
- O capítulo 12 apresenta conclusões com relação aos pontos fortes e fracos do sistema desenvolvido e ideias que podem ser desenvolvidas futuramente para melhorar o sistema.

# 2. Estado da arte

Neste capítulo serão apresentadas algumas das soluções estudadas durante o desenvolvimento deste projeto, que permitem obter de dados e controlar contadores inteligentes remotamente. Como o cliente pretendia um sistema de medição de consumo de água residencial e controlo remoto dos contadores, este capítulo foca-se nos contadores de água inteligentes disponibilizados pelas empresas que desenvolveram as soluções e as operações que podem ser efetuadas remotamente nesses contadores.

Para cada solução serão apresentados os pontos fortes, ou seja as principais funcionalidades vantajosas que não são comuns neste tipo de soluções, e as principais limitações ou características que fazem com que a solução não seja totalmente compatível com os requisitos do cliente. Para cada empresa serão apresentados os contadores, as redes de comunicação e as plataformas de gestão e monitorização.

## 2.1. Diehl

A Diehl[1] é uma empresa tecnológica alemã que se foca nas áreas de aviação, defesa, metalúrgica e medição. Dentro da área de medição esta empresa possui soluções que permitem gerir e medir o consumo de água, eletricidade, gás e energia térmica.

A solução[2] de medição do consumo de água da Diehl utiliza contadores que comunicam por M-Bus (Meter-Bus), IrDA (infravermelhos) e ZVEI (M-Bus sobre uma interface ótica) para acelerar a aquisição de leituras presencial. Para obter leituras remotamente a Diehl utiliza IZAR RADIO (comunicação por rádio proprietária), LoRaWAN e Narrowband IoT (comunicação de banda estreita para a internet das coisas).

Esta solução inclui uma plataforma *online* de gestão e monitorização de contadores inteligentes chamada “IZAR PLUS”. Esta plataforma pode ser utilizada por empresas distribuidoras para efetuar a gestão e monitorização do consumo de água, eletricidade, gás, energia elétrica e energia térmica, a partir de um *web browser*. Também possui uma aplicação de gestão e monitorização de contadores inteligentes chamada “IZAR@NET”. Esta aplicação pode ser instalada em computadores da empresa cliente, tendo quase todas as funcionalidades da plataforma *online*, sendo acedida também através de um *web browser*.

A Diehl fornece vários tipos de contadores de água, de entre os quais contadores inteligentes que possuem uma bateria de lítio para alimentação e transmitem as leituras em intervalos periódicos. É possível obter leituras dos contadores inteligentes através de uma rede sem fios fixa, em que existem *gateways* que recebem as leituras dos contadores e carregam-nas periodicamente para uma das plataformas através de FTP ou SFTP. Também é possível obter leituras em movimento através de um dispositivo móvel, em que os funcionários se deslocam a pé ou num veículo até junto dos contadores, transportando um módulo recetor de dados que recolhe leituras dos contadores ao passar perto destes. Se a recolha for feita num veículo, este pode deslocar-se a uma velocidade máxima de 50KM/h enquanto o dispositivo móvel recolhe leituras.

**Pontos fortes**

Esta solução tem disponíveis vários tipos de contadores para diferentes aplicações e casos de uso, desde contadores residenciais, contadores industriais e contadores próprios para redes de distribuição de água. O sistema apresentado neste relatório fornece apenas um tipo de contador residencial, no entanto poderiam ser desenvolvidos mais tipos de contadores tendo por base o microcontrolador, sensores, módulo de rádio e *software* do contador desenvolvido.

**Limitações**

A rede sem fios fixa é unidirecional (os contadores transmitem, os *gateways* recebem), portanto não é possível enviar comandos para os contadores (tais como comandos de controlo da válvula) ou alterar configurações remotamente. O sistema apresentado neste relatório também obtém leituras através de uma rede sem fios fixa, sendo que essa rede também permite enviar comandos e alterar configurações dos contadores remotamente.

Esta solução não inclui válvulas de corte de fornecimento de água com controlo remoto para clientes residenciais, quer pela rede móvel como pela rede sem fios fixa. O sistema apresentado neste relatório contém contadores de água para clientes residenciais, com válvula incorporada.

Os alertas dos contadores estão principalmente relacionados com consumo de água. Por exemplo existem alertas de fugas de água, excesso de fluxo, insuficiência de fluxo e fluxo reverso, existindo também um alerta de bateria fraca e um alerta de perda de ligação de rádio. Alguns alertas que eram requisito do cliente não são suportados por esta solução, como por exemplo, alertas relacionados com tentativa de fraude. Os contadores do sistema apresentado neste relatório também emitem estes alertas (com a exceção dos alertas de excesso de fluxo, insuficiência de fluxo e perda de ligação de rádio). Adicionalmente emitem alertas de tentativas de fraude (tais como deteção de campo magnético forte), alto consumo, zero consumo, erro de memória e data-hora inválida.

## 2.2. Kamstrup

A Kamstrup[3] é uma empresa tecnológica alemã que se foca na área de medição de consumos, particularmente na medição de consumo de energia elétrica, água, aquecimento e arrefecimento.

Para obter leituras remotamente a solução[4] de medição do consumo de água da Kamstrup utiliza contadores que comunicam por M-Bus com fios, M-Bus sem fios e Sigfox (tecnologia proprietária de comunicação sem fios de banda estreita). Existe um contador especial que pode ser personalizado para vários casos de uso e que pode comunicar por M-Bus sem fios, Zigbee, GPRS, 3G entre outras tecnologias de rádio. Além dos contadores possui também uma válvula de corte de fornecimento água que pode ser controlada remotamente através de M-Bus sem fios.

Esta solução inclui uma plataforma de gestão e monitorização dos contadores chamada “READy Manager” que, suporta até 100000 contadores, segundo o anunciado e pode ser instalada em computadores da empresa cliente.

As leituras podem ser obtidas em movimento, através de um dispositivo chamado "READy Converter" que comunica com os contadores por M-Bus sem fios. Este dispositivo liga-se por Bluetooth a um telemóvel com sistema operativo Android que possua a aplicação "READy App". Por sua vez o telemóvel liga-se a um computador com o *software* "READy Manager" através de uma rede IP, como a Internet. As leituras também podem ser obtidas por uma rede sem fios fixa. Neste caso os contadores transmitem periodicamente a leitura atual para os *gateways* através de M-Bus sem fios e por sua vez os *gateways* comunicam a cada hora a ultima leitura recebida de cada contador para o software "READy Manager" através de uma rede IP.

**Pontos fortes**

Permite partilhar redes sem fios fixas com outras empresas distribuidoras de água, eletricidade, aquecimento ou arrefecimento, reduzindo os custos iniciais de instalação da rede e os custos de manutenção. A Kamstrup afirma que esta solução cumpre o regulamento geral de proteção de dados e que cada empresa só consegue aceder aos seus dados. O sistema apresentado neste relatório suporta "multitenancy", sendo possível partilhar não só as redes sem fios fixas como também a própria plataforma de suporte, gestão e monitorização com mais do que uma empresa ou departamento.

Suporta contadores de outras marcas desde que estes comuniquem por M-Bus com fios. A Kamstrup fornece um dispositivo chamado "M-Bus Master" que serve como ponto central (comunica com um computador com READy Manager diretamente) ou como repetidor da rede cablada M-Bus. Segundo a Kamstrup este dispositivo pode suportar até 1250 contadores em cada sistema M-Bus. O sistema apresentado neste relatório suporta contadores de outras marcas desde que seja desenvolvido um *gateway* que traduza as mensagens trocadas com os contadores em pedidos para os *web services*.

**Limitações**

A rede sem fios fixa é unidirecional, portanto não é possível enviar comandos para os contadores nem alterar configurações remotamente. A Kamstrup possui alguns modelos de contador que suportam comunicação bidirecional, sendo possível alterar configurações dos contadores sem fios, no entanto só é possível configurar contadores deslocando-se fisicamente até junto do contador (a Kamstrup afirma que é possível alterar configurações a partir da rua). O sistema apresentado neste relatório obtém todas as leituras através de uma rede sem fios fixa, não existindo um dispositivo que comunique diretamente com os contadores. No entanto é possível aceder ao *website* de administração através da Internet e alterar configurações no local, de modo a verificar se estas tiveram efeito.

A válvula de interrupção do fornecimento de água é um dispositivo separado. O sistema apresentado neste relatório contém contadores de água com válvula incorporada. Como o cliente deste sistema pretendia ter controlo remoto sobre o fornecimento de água de todos os seus clientes, incorporar a válvula em cada contador torna-se mais barato do que adquirir um contador e válvula separados. Adicionalmente a manutenção torna-se mais fácil pois existe apenas uma bateria para carregar ou trocar e não é necessário associar cada contador e cada válvula a cada cliente.

## 2.3. Itron

A Itron[5] é uma empresa tecnológica norte-americana que se foca na área de medição de consumos, particularmente na medição de consumo de energia elétrica, energia térmica, gás e água.

Para obter leituras remotamente a solução[6] de medição do consumo de água da Itron utiliza contadores que possuem uma bateria de lítio e comunicam sem fios pelo protocolo RADIAN, LoRaWAN, M-Bus sem fios e Sigfox.

A Itron fornece uma plataforma *online* de gestão e monitorização chamada "Temetra". Esta plataforma permite carregar leituras dos contadores recolhidas pelos dispositivos móveis de recolha de leituras, inclui um portal para os clientes da empresa distribuidora de água poderem consultar os seus consumos e pode ser integrada com sistemas de faturação. Também fornece uma plataforma chamada "EverBlu", que permite gerir e monitorizar contadores inteligentes ligados por uma rede sem fios fixa, e uma plataforma chamada "AnyQuest", que permite descarregar leituras dos contadores obtidas por dispositivos móveis de recolha de leituras, sendo que ambas as plataformas podem ser instaladas em computadores da empresa cliente.

As leituras podem ser obtidas em movimento, a pé ou com um veículo, utilizando um dispositivo chamado "Itron Mobile Radio" que se liga aos contadores através de uma tecnologia de rádio proprietária chamada "SRead". Este dispositivo liga-se a um telemóvel, tablet ou computador portátil com sistema operativo Android ou Windows através de Bluetooth, permitindo obter leituras e enviar comandos para os contadores. A Itron também fornece um computador portátil chamado "MC3" que possui o *hardware* necessário para comunicar com os contadores e o *software* de aquisição de leituras e controlo dos contadores, permitindo obter leituras e controlar os contadores com apenas um dispositivo. As leituras também podem ser obtidas por uma rede sem fios fixa, neste caso os contadores transmitem a leitura atual periodicamente ou quando solicitado, para os *gateways* e os *gateways* encaminham as leituras para a plataforma "EverBlu" através de uma rede IP.

**Pontos fortes**

A rede sem fios fixa é bidirecional, sendo possível solicitar leituras, enviar comandos de rede, atualizar o *firmware* e atualizar a data-hora dos contadores. Embora este sistema permita enviar um número limitado de comandos, estes comandos são importantes para a gestão remota dos contadores. O sistema apresentado neste relatório também suporta comunicação bidirecional com os contadores, todos os comandos e configurações possíveis podem ser transmitidos através da rede sem fios fixa.

Esta solução inclui dispositivos que podem ser ligados a qualquer tipo de contador que emita pulsos. Estes dispositivos convertem os pulsos dos contadores em leituras e envia-as para a plataforma de aquisição de leituras, servindo como intermediário entre contadores antigos ou não suportados e a plataforma de leituras. O sistema apresentado neste relatório não inclui dispositivos que servem de intermediário entre contadores que emitem pulsos e a plataforma de aquisição de leituras. Tal dispositivo poderia ser desenvolvido tendo por base o *hardware* e *software* do contador desenvolvido e, em vez de utilizar o sensor de fluxo de água, usar um sensor de pulsos em que o volume de água que cada pulso representa seria configurável.

**Limitações**

Esta solução não inclui válvulas de corte de fornecimento de água com controlo remoto, quer pela rede móvel como pela rede sem fios fixa. O sistema apresentado neste relatório contém contadores de água com válvula incorporada.

As baterias de alguns dispositivos, nomeadamente sensores de pulsos e repetidores de sinal, não são substituíveis. Segundo a Itron, as baterias destes dispositivos devem durar entre doze a vinte anos, no entanto além do custo dos novos dispositivos de substituição é um desperdício deitar fora dispositivos que só necessitam de uma bateria nova. O sistema apresentado neste relatório contém contadores de água cuja bateria pode ser recarregada ou substituída quando necessário.

A potência de transmissão dos contadores é relativamente baixa (menos de 10 mW). Embora os contadores utilizem frequências relativamente baixas (p. ex.: 433 MHz) e um tipo de modulação cuja desmodulação é resistente a interferências (Frequency-shift Keying), o alcance da transmissão será reduzido. Isto não é problema para casos em que as leituras são obtidas com dispositivos móveis, pois normalmente estes passam perto dos contadores, mas pode ser um problema para casos em que as leituras são obtidas por uma rede sem fios fixa, já que os *gateways* fixos vão ter mais dificuldade em receber as transmissões dos contadores. Dependendo da região e frequência utilizada, os contadores incluídos no sistema apresentado neste relatório podem transmitir com uma potência máxima de cerca de 125 mW, tornando a aquisição de mensagens dos contadores mais fácil.

## 2.4. Resumo

De seguida são resumidamente apresentadas as principais diferenças entre as soluções apresentadas e a solução desenvolvida.

As principais funcionalidades da solução desenvolvida que não estão presentes nas soluções apresentadas são:

- A rede sem fios fixa é bidirecional, sendo possível receber e enviar mensagens para contadores inteligentes (Diehl e Kamstrup).
- Todos os tipos de alertas requeridos pelo cliente são suportados, por exemplo alertas de tentativa de fraude (Diehl).
- A solução inclui válvula de corte do fornecimento de água com controlo remoto para clientes residenciais (Diehl e Itron).
- Os contadores inteligentes residenciais incluem válvula de corte do fornecimento de água com controlo remoto (Kamstrup).
- As baterias de todos os dispositivos são substituíveis (Itron).
- A potência de transmissão dos contadores é razoável, tornando a receção das mensagens pelos *gateways* fixos mais fácil (Itron).

As principais funcionalidades das soluções apresentadas que não estão presentes na solução desenvolvida são:

- Aquisição de leituras utilizando dispositivos móveis, criando uma “rede sem fios móvel” (todas as soluções).
- Variedade de contadores para diferentes casos de uso (Diehl).

- Tem suporte a contadores de outras marcas através de M-Bus com fios (Kamstrup).
- A solução inclui um dispositivo que converte pulsos dos contadores em mensagens de leitura compatíveis com o sistema (Itron).

As principais semelhanças entre a solução desenvolvida e as soluções apresentadas são apresentadas resumidamente:

- Aquisição de leituras de contadores remotos automaticamente através de uma rede sem fios fixa (todas as soluções).
- A solução inclui contadores de água inteligentes (todas as soluções).
- Os contadores comunicam com a rede sem fios fixa por LoRaWAN (Diehl e Itron).
- A solução inclui uma válvula de corte do fornecimento de água para clientes residenciais (Kamstrup).
- A solução inclui uma plataforma de gestão e monitorização de contadores inteligentes (todas as soluções).
- Permite partilhar a rede sem fios fixa com outras empresas (Kamstrup).

# 3. Plano

Neste capítulo é apresentado o plano do projeto. O plano do projeto é dividido em fases e as fases são divididas em tarefas. As tarefas apresentadas são de alto nível. Por exemplo, existe uma tarefa "Configuração remota dos contadores" que é subdividida nas seguintes tarefas:

- Desenvolver uma página no *website* de administração que permita alterar configurações dos contadores.
- Especificar e implementar métodos de configuração de contadores nos *web services*.
- Especificar e implementar mensagens de configuração no protocolo de rádio.
- Implementar a alteração de configurações remotamente no *software* dos contadores.

Incluir todas as subtarefas no plano tornaria o plano demasiado extenso e desnecessariamente complicado. As subtarefas foram definidas pela equipa de desenvolvimento num documento à parte, e praticamente só foram consultadas pelos elementos da própria equipa de desenvolvimento para servir de guia e medir o progresso das tarefas do projeto. Quando todas as subtarefas que compõem uma tarefa são concluídas a tarefa era marcada como concluída no plano.

Será apresentado o plano definido no início do projeto, os imprevistos que surgiram durante o desenvolvimento do projeto e as alterações ao plano que resultaram desses imprevistos.

## 3.1. Plano Inicial

Nesta secção é apresentado o plano inicial. Este plano contém as fases e as tarefas de alto nível do projeto e os respetivos prazos. Na fase de análise foram identificados os requisitos e tomadas decisões que, se fossem tomadas mais tarde, poderiam interromper o desenvolvimento do projeto (a chamada "paralisia por análise"). Na fase de *design* foi feito um protótipo de alta-fidelidade do *website* de administração, este protótipo era composto por páginas *web* que implementavam as funcionalidades mais importantes para o cliente, mostrando dados simulados. Durante a fase de desenvolvimento foi realizada a maior parte do trabalho, este trabalho consistiu principalmente no desenvolvimento de *software* e na integração de componentes de *hardware*. A fase de testes aconteceu em simultâneo com a fase de desenvolvimento, cada funcionalidade desenvolvida era testada assim que o seu desenvolvimento era concluído permitindo detetar e corrigir erros o mais cedo possível. Adicionalmente foram feitos alguns testes de utilização do sistema após terminar o desenvolvimento. Por fim, na fase de demonstração o sistema desenvolvido foi demonstrado ao cliente e foram feitos alguns ajustes finais. A Figura 3-1 mostra o diagrama de Gantt do plano inicial.

| ID | WBS | Task Name | Duration | Start | Finish |
|---|---|---|---|---|---|
| 1 | **1** | **Análise** | **16 days** | **Mon 10/10/16** | **Mon 31/10/16** |
| 2 | 1.1 | Levantamento e análise de requisitos | 5 days | Mon 10/10/16 | Fri 14/10/16 |
| 3 | 1.2 | Reunião com o cliente | 1 day | Mon 17/10/16 | Mon 17/10/16 |
| 4 | 2.2 | Seleção do hardware dos contadores | 10 days | Tue 18/10/16 | Mon 31/10/16 |
| 5 | 2.3 | Seleção de gateway e tecnologia de comunicação sem fios | 10 days | Tue 18/10/16 | Mon 31/10/16 |
| 6 | 2.4 | Seleção de protocolos de comunicação | 5 days | Tue 18/10/16 | Mon 24/10/16 |
| 7 | **2** | **Design** | **15 days** | **Wed 02/11/16** | **Tue 22/11/16** |
| 8 | 2.1 | Protótipo do website | 15 days | Wed 02/11/16 | Tue 22/11/16 |
| 9 | **3** | **Desenvolvimento** | **120 days** | **Wed 23/11/16** | **Tue 16/05/17** |
| 10 | 3.1 | Feedback do cliente | 120 days | Wed 23/11/16 | Tue 16/05/17 |
| 11 | 3.2 | Base de dados | 80 days | Wed 23/11/16 | Thu 16/03/17 |
| 12 | 3.3 | Website | 120 days | Wed 23/11/16 | Tue 16/05/17 |
| 13 | 3.4 | Web service | 40 days | Fri 17/03/17 | Tue 16/05/17 |
| 14 | 3.5 | Contadores | 80 days | Wed 23/11/16 | Thu 16/03/17 |
| 15 | 3.6 | Protocolo de comunicação com os contadores | 65 days | Fri 27/01/17 | Tue 02/05/17 |
| 16 | 3.7 | Software dos gateways | 40 days | Fri 27/01/17 | Thu 23/03/17 |
| 17 | 3.8 | Aquisição, transmissão e armazenamento de leituras | 30 days | Mon 28/11/16 | Tue 10/01/17 |
| 18 | 3.9 | Configuração remota dos contadores | 15 days | Fri 16/12/16 | Thu 05/01/17 |
| 19 | 3.10 | Controlo remoto dos contadores | 15 days | Fri 06/01/17 | Thu 26/01/17 |
| 20 | 3.11 | Emissão e armazenamento de alertas | 15 days | Fri 27/01/17 | Thu 16/02/17 |
| 21 | 3.12 | Agendamento de ações | 20 days | Fri 17/02/17 | Thu 16/03/17 |
| 22 | **4** | **Testes** | **140 days** | **Fri 02/12/16** | **Thu 22/06/17** |
| 23 | 4.1 | Utilização do de funcionalidades do sistema a partir do website | 140 days | Fri 02/12/16 | Thu 22/06/17 |
| 24 | 4.2 | Teste de resposta e carga do web service | 10 days | Wed 17/05/17 | Tue 30/05/17 |
| 25 | 4.3 | Cobertura do sinal de rádio | 5 days | Wed 26/04/17 | Wed 03/05/17 |
| 26 | 4.4 | Testes de medição de consumo de água e de interrupção do fornecimento | 2 days | Thu 04/05/17 | Fri 05/05/17 |
| 27 | **5** | **Demonstração** | **5 days** | **Fri 23/06/17** | **Thu 29/06/17** |
| 28 | 5.1 | Apresentação ao cliente | 1 day | Fri 23/06/17 | Fri 23/06/17 |
| 29 | 5.2 | Pequenos ajustes/personalização | 5 days | Fri 23/06/17 | Thu 29/06/17 |

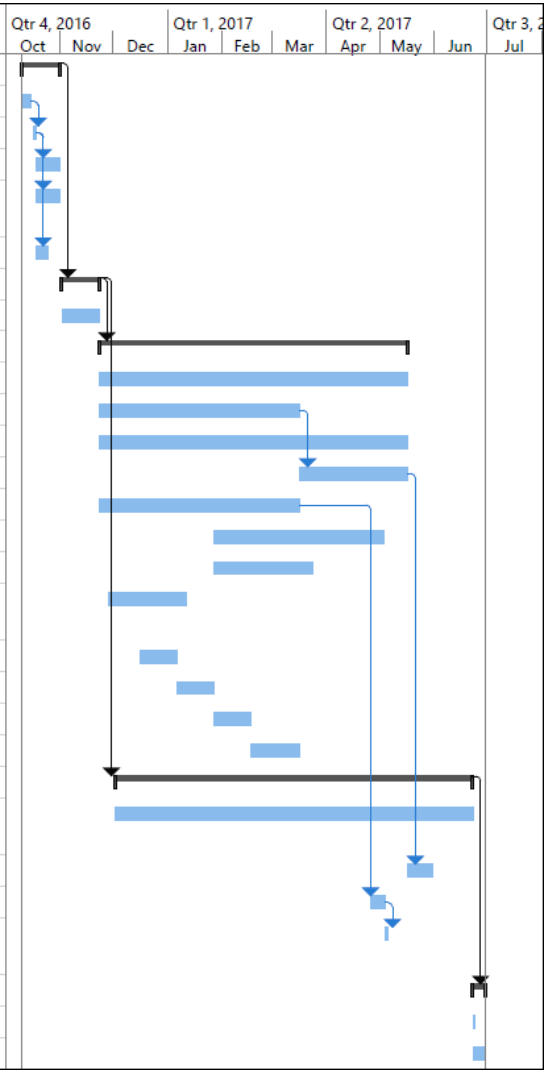



**Figura 3-1 - Plano inicial**

## 3.2. Plano Final

Nesta secção é apresentado o plano final, este plano mostra as tarefas executadas, o tempo que cada tarefa realmente demorou e os imprevistos que causaram alterações ao plano inicial.

Inicialmente foi definido que eu iria desenvolver o *website* de administração, a base de dados e o *web service* no IIS. O *web service* no IIS seria utilizado pelos outros elementos da equipa de desenvolvimento para que o *software* de *gateway* desenvolvido por eles pudesse comunicar com o *software* de servidor desenvolvido por mim, servindo como uma interface entre os *gateways* e o servidor. O tempo necessário para o desenvolvimento do contador de água inteligente foi subestimado, o que se traduziu em atrasos no plano inicial, pelo que eu passei a ficar responsável pelo desenvolvimento do *software* dos *gateways* e o resto da equipa focou-se no desenvolvimento do contador de água inteligente.

Este aumento de responsabilidades fez com que a prioridade de algumas funcionalidades que não faziam parte dos requisitos do cliente, mas que seriam mais-valias para facilitar a administração de grandes números de contadores, fosse reduzida. O trabalho adicional traduziu-se em várias funcionalidades de baixa prioridade parcialmente implementadas. No entanto, os requisitos mais importantes para o cliente foram implementados.

Quando comecei a desenvolver o *software* dos *gateways* demorei algum tempo a aprender a trabalhar com as plataformas Node.js e Node-RED, pois estas possuem bibliotecas, funções, estruturas de dados, etc. específicas. Posteriormente outra alteração importante ao projeto foi a decisão de não utilizar o *gateway* da Multitech, sendo desenvolvido um *gateway* para o projeto. Adaptei o código desenvolvido em Node-RED para o *gateway* da Multitech, para comunicar com o LoRa App Server através de MQTT em vez de comunicar diretamente com os contadores através de LoRa e criei uma máquina virtual para alojar o Node-RED. Este processo de adaptação do código demorou algum tempo, pois a versão de Node-RED incluída no *gateway* da Multitech era muito antiga e consequentemente o código desenvolvido não era compatível com a última versão do Node-RED, disponível na altura em que o código foi adaptado.

A Figura 3-2 mostra o diagrama de Gantt do plano final, as tarefas foram concluídas dentro dos prazos previstos, com a exceção do desenvolvimento dos contadores e do *software* dos *gateways*. Adicionalmente após a data de conclusão do projeto foram desenvolvidas funcionalidades que facilitam a administração do sistema.

| ID | WBS | Task Name | Duration | Start | Finish |
|---|---|---|---|---|---|
| 1 | **1** | **Análise** | **16 days** | **Mon 10/10/16** | **Mon 31/10/16** |
| 2 | 1.1 | Levantamento e análise de requisitos | 5 days | Mon 10/10/16 | Fri 14/10/16 |
| 3 | 1.2 | Reunião com o cliente | 1 day | Mon 17/10/16 | Mon 17/10/16 |
| 4 | 2.2 | Seleção do hardware dos contadores | 10 days | Tue 18/10/16 | Mon 31/10/16 |
| 5 | 2.3 | Seleção de gateway e tecnologia de comunicação sem fios | 10 days | Tue 18/10/16 | Mon 31/10/16 |
| 6 | 2.4 | Seleção de protocolos de comunicação | 5 days | Tue 18/10/16 | Mon 24/10/16 |
| 7 | **2** | **Design** | **15 days** | **Wed 02/11/16** | **Tue 22/11/16** |
| 8 | 2.1 | Protótipo do website | 15 days | Wed 02/11/16 | Tue 22/11/16 |
| 9 | **3** | **Desenvolvimento** | **120 days** | **Wed 23/11/16** | **Tue 16/05/17** |
| 10 | 3.1 | Feedback do cliente | 120 days | Wed 23/11/16 | Tue 16/05/17 |
| 11 | 3.2 | Base de dados | 80 days | Wed 23/11/16 | Thu 16/03/17 |
| 12 | 3.3 | Website | 120 days | Wed 23/11/16 | Tue 16/05/17 |
| 13 | 3.4 | Web service | 40 days | Fri 17/03/17 | Tue 16/05/17 |
| 14 | 3.5 | Contadores | 120 days | Wed 23/11/16 | Tue 16/05/17 |
| 15 | 3.6 | Protocolo de comunicação com os contadores | 65 days | Fri 27/01/17 | Tue 02/05/17 |
| 16 | 3.7 | Software dos gateways | 70 days | Fri 27/01/17 | Tue 09/05/17 |
| 17 | 3.8 | Aquisição, transmissão e armazenamento de leituras | 30 days | Fri 27/01/17 | Thu 09/03/17 |
| 18 | 3.9 | Configuração remota dos contadores | 15 days | Fri 27/01/17 | Thu 16/02/17 |
| 19 | 3.10 | Controlo remoto dos contadores | 15 days | Fri 17/02/17 | Thu 09/03/17 |
| 20 | 3.11 | Emissão e armazenamento de alertas | 15 days | Fri 10/03/17 | Thu 30/03/17 |
| 21 | 3.12 | Agendamento de ações | 20 days | Fri 31/03/17 | Tue 02/05/17 |
| 22 | **4** | **Testes** | **140 days** | **Fri 02/12/16** | **Thu 22/06/17** |
| 23 | 4.1 | Utilização do de funcionalidades do sistema a partir do website | 140 days | Fri 02/12/16 | Thu 22/06/17 |
| 24 | 4.2 | Teste de resposta e carga do web service | 10 days | Wed 17/05/17 | Tue 30/05/17 |
| 25 | 4.3 | Cobertura do sinal de rádio | 5 days | Wed 17/05/17 | Tue 23/05/17 |
| 26 | 4.4 | Testes de medição de consumo de água e de interrupção do fornecimento | 2 days | Wed 24/05/17 | Thu 25/05/17 |
| 27 | **5** | **Demonstração** | **5 days** | **Fri 23/06/17** | **Thu 29/06/17** |
| 28 | 5.1 | Apresentação ao cliente | 1 day | Fri 23/06/17 | Fri 23/06/17 |
| 29 | 5.2 | Pequenos ajustes/personalização | 5 days | Fri 23/06/17 | Thu 29/06/17 |
| 30 | **6** | **Trabalho adicional** | **25 days** | **Fri 30/06/17** | **Thu 03/08/17** |
| 31 | 6.1 | Funcionalidades de auxilio à administração do sistema e resolução de problemas | 25 days | Fri 30/06/17 | Thu 03/08/17 |

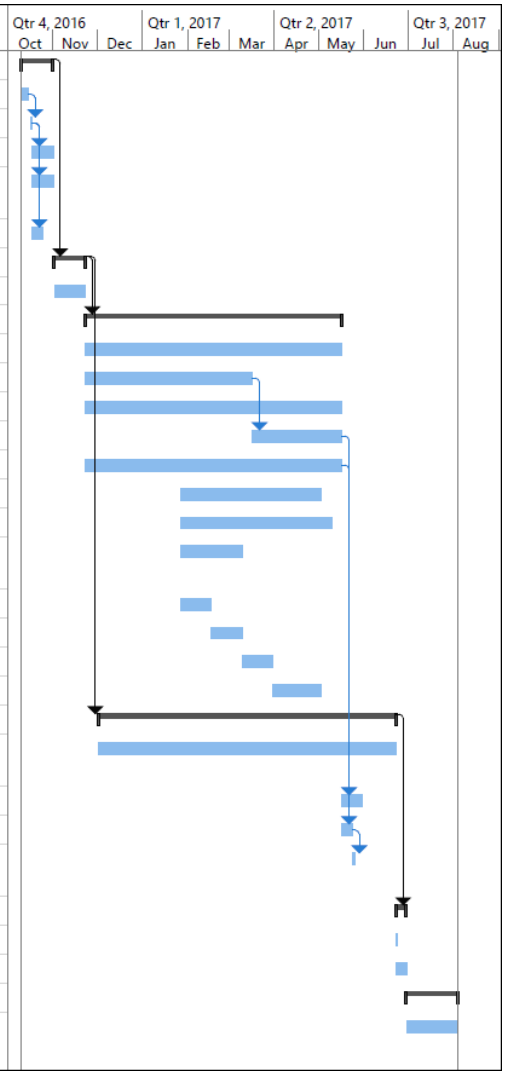



**Figura 3-2 - Plano final**

# 4. Objetivos e Metodologias

Neste capítulo serão apresentados os objetivos do estágio e do projeto, o trabalho que foi realizado para atingir esses objetivos e uma vista geral do projeto. Como existiu uma equipa de desenvolvimento será apresentada a minha contribuição para o projeto e contribuição dos meus colegas. As principais metodologias de trabalho também serão apresentadas em detalhe neste capítulo.

## 4.1.1. Objetivos

Os objetivos do estágio são principalmente pôr em prática os conhecimentos adquiridos ao longo do percurso académico, ter a experiencia de trabalhar em equipa num meio profissional, aprender novos métodos de trabalho e aprender a trabalhar com tecnologias utilizadas no mercado de trabalho, com a assistência do monitor de estágio, resultando na aquisição de novas competências e desenvolvimento do sentido de responsabilidade.

O objetivo principal deste projeto é desenvolver um sistema que permita automatizar ou facilitar as principais operações que os funcionários das empresas distribuidoras de água efetuam nas instalações dos clientes, tais como medir o consumo de água ou interromper e restaurar o fornecimento de água. Para atingir esse objetivo foram desenvolvidos contadores "inteligentes" que possuem um microcontrolador, um conjunto de sensores e atuadores, comunicação sem fios e alimentação por bateria, estes contadores são colocados nas instalações dos clientes, substituindo os contadores instalados nessas instalações. Adicionalmente foram implementadas funcionalidades adicionais que tornam o sistema mais flexível permitindo, por exemplo, suportar outros tipos de contadores ou ajudar a resolver problemas com o sistema, sendo que estas funcionalidades não fazem parte dos requisitos definidos pelo cliente.

Os contadores de água comunicam periodicamente a leitura do consumo de água, possuem uma válvula que pode cortar o fluxo de água, possuem sensores para detetar tentativas de falsificar leituras e possuem sensores para detetar problemas com o contador. Os contadores comunicam com o sistema através de *gateways* colocados em pontos que permitem o máximo de cobertura possível, de modo a poderem cobrir as áreas em que os contadores são instalados e que o alcance de alguns *gateways* se sobreponha de modo a obter redundância na comunicação com os contadores.

Os *gateways* ligam-se à Internet através de uma rede celular de telecomunicações de modo a conseguirem comunicar com os servidores de suporte da rede LoRa, tendo como objetivos principais encaminhar pacotes de e para os contadores e serem geridos pelos servidores de modo a formar uma única rede com múltiplos pontos de acesso.

Existem dois tipos de servidores, os servidores de suporte à rede Lora e os servidores que servem o *software* desenvolvido. Os servidores de suporte à rede Lora gerem a rede LoRa e servem de intermediário entre a rede LoRa e a aplicação desenvolvida para este projeto, através de um *software* desenvolvido para esse efeito. Os servidores que servem o *software* desenvolvido servem os vários componentes de *software* que implementam os requisitos do lado do servidor, de entre os quais um *software* que comunica com os servidores de suporte à rede LoRa e que codifica/descodifica as mensagens dos contadores, *web services* que permitem efetuar operações na base de dados e enviar/receber mensagens de/para os contadores, um *website* que é a interface de utilizador e permite visualizar leituras dos contadores, controlar os contadores,

configurar contadores entre outras operações e *software* comum que é utilizado pelos vários componentes de *software* da aplicação para executar operações em comum e que permite coordenar a comunicação entre os vários componentes. O sistema utiliza bases de dados para fins de persistência dos dados da aplicação, permitindo armazenar e consultar dados quando necessário. A Figura 4-1 mostra o diagrama da vista geral do projeto.

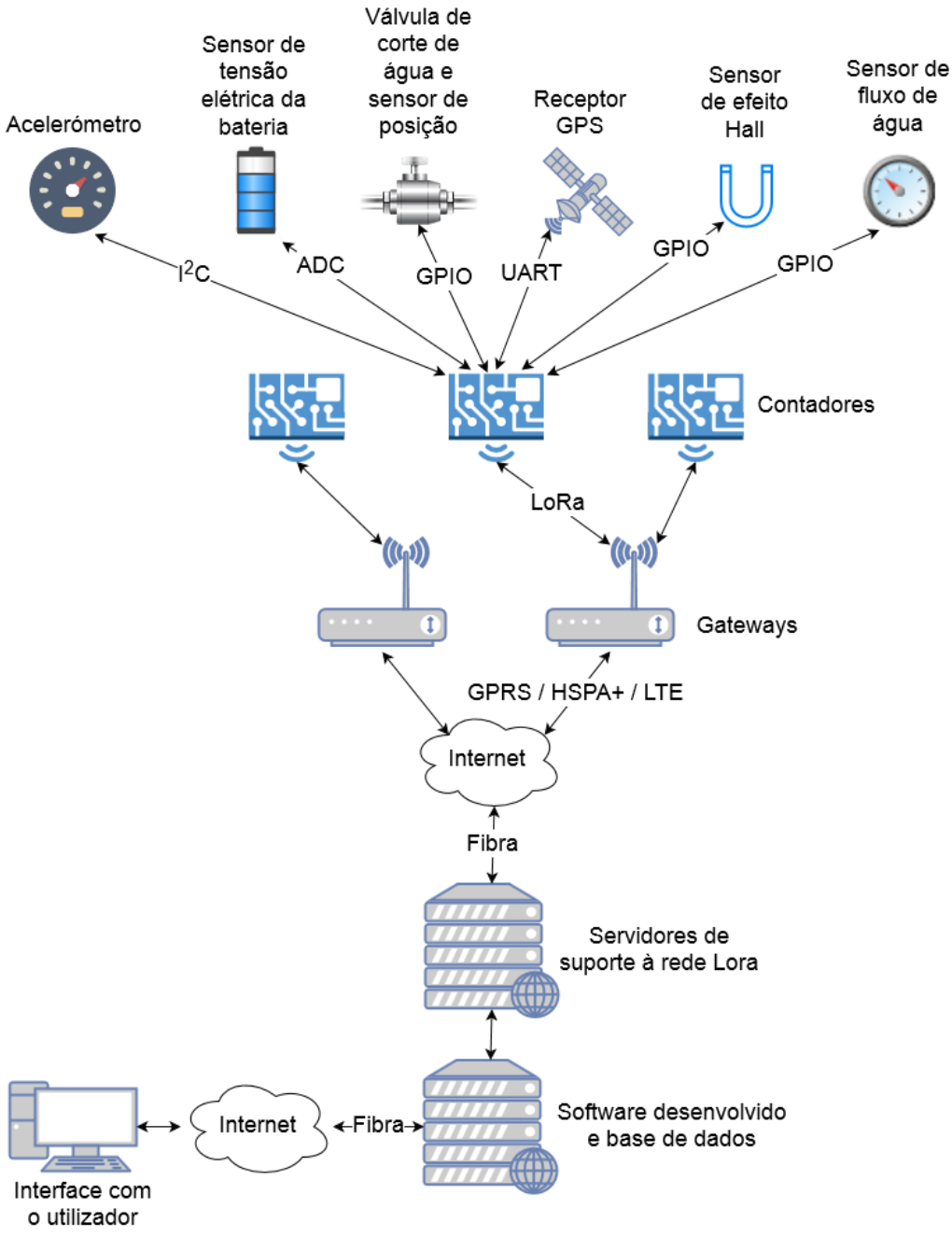



**Figura 4-1 - Vista geral do projeto**

### 4.1.2. Contribuições para o projeto

A equipa de desenvolvimento deste projeto foi composta por mim e mais dois colegas. Fui responsável pelo desenvolvimento do *software* dos servidores, enquanto os meus

colegas foram responsáveis pelo desenvolvimento de *hardware* e do respetivo *software* de suporte.

Executei as atividades:

- Projeto e implementação da base de dados.
- Desenvolvimento de um *website* de administração.
- Especificação do *web service* para consultar a base de dados e implementação desse *web service* no IIS.
- Especificação do *web service* de *gateway*.
- Implementação do *web service* de *gateway* e do protocolo de comunicação via rádio no Node-RED.
- Integração do *software* implementado no Node-RED com o LoRa App Server.

Os meus colegas executaram as atividades:

- Integração dos sensores, atuadores e módulo de rádio LoRa com o microcontrolador dos contadores de água.
- Desenvolvimento do *software* para o microcontrolador dos contadores de água.
- Configuração da LoRa Gateway Bridge.
- Integração do módulo de rádio LoRa com o *gateway*.

Em conjunto executamos as atividades:

- Especificação do protocolo de comunicação via rádio com os contadores.
- Configuração do servidor de rede LoRa (LoRa Server).
- Configuração do servidor de aplicação LoRa (LoRa App Server).
- Configuração do mediador MQTT (Eclipse Mosquitto).
- Realização de testes nos contadores, tais como cobertura de sinal, transmissão e receção de mensagens, controlo da válvula, alteração de configurações, receção de leituras, alertas, entre outros.

A Figura 4-2 mostra um diagrama das contribuições para o projeto.

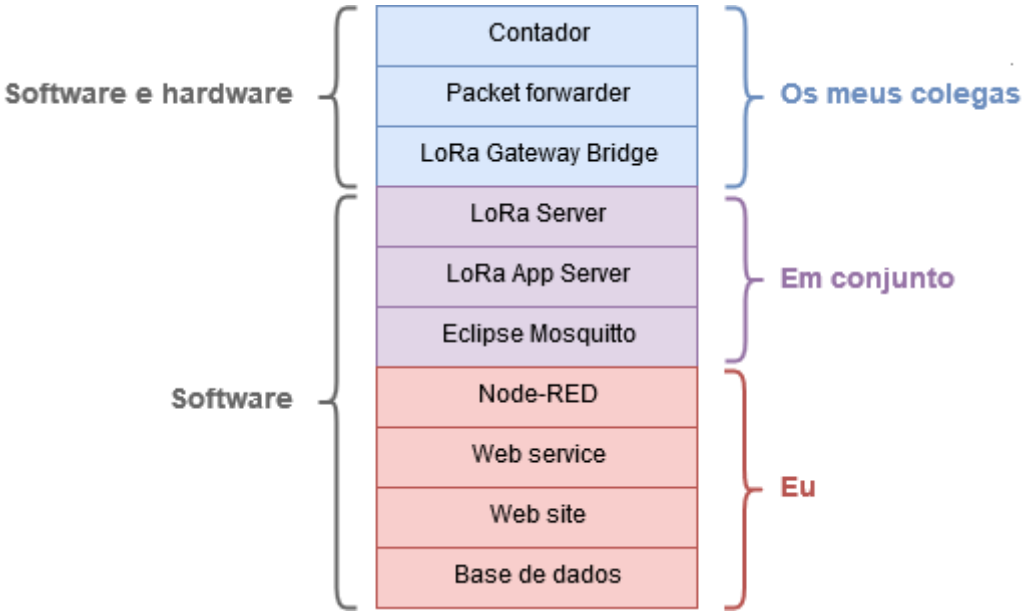



**Figura 4-2 – Contribuições para o projeto**

### 4.1.3. Metodologia de desenvolvimento de *software*

Foi seguido um método de desenvolvimento de *software* com algumas características semelhantes com o método ágil. A equipa focava-se em trabalhar num requisito de cada vez, testava o código desse requisito e quando este passasse os testes era integrado no sistema. No entanto não eram feitas reuniões diárias (*daily scrum*) com os restantes elementos da equipa de desenvolvimento. Os problemas encontrados durante o desenvolvimento de um requisito eram discutidos normalmente apenas entre os elementos que estavam a desenvolver o requisito. Só eram feitas reuniões com todos os elementos quando era necessário. Adicionalmente não existia o papel de "*scrum master*" na equipa de desenvolvimento.

A última versão do sistema estava sempre acessível a partir da Internet e era atualizada no final da semana, de modo a que o cliente pudesse aceder e verificar se os novos requisitos implementados correspondiam ao que pretendia. Se o cliente aprovasse os novos requisitos implementados, esses requisitos eram considerados concluídos, caso contrário as alterações que o cliente pretendia efetuar eram anotadas junto do código fonte do respetivo requisito, num comentário iniciado pela expressão "TODO". Quando um comentário no código fonte é iniciado pela expressão "TODO" o IDE Visual Studio adiciona automaticamente o comentário a uma lista de tarefas, posteriormente é possível ir para a zona do código fonte onde o comentário se encontra clicando na tarefa relacionada. Os requisitos que necessitavam de ser alterados a pedido do cliente eram mantidos nas próximas atualizações semanais do sistema, na interface do utilizador era adicionada a palavra "beta" junto ao título da página e hiperligações que levassem à página desse requisito. Quando as alterações pedidas pelo cliente fossem implementadas e o cliente aprovasse os novos requisitos, iniciava-se o desenvolvimento do próximo requisito, de acordo com a prioridade atribuída pelo cliente. Em qualquer altura o cliente podia remover requisitos ou alterar a sua prioridade, também podia modificar os requisitos atuais ou adicionar novos requisitos, mas nesse caso era feita uma reunião com os elementos da equipa de desenvolvimento para averiguar se era possível implementar os requisitos dentro do tempo disponível.

A documentação de código fonte C# é feita com recurso a "XML Documentation Comments", que são comentários no formato XML precedidos de três barras (///), inseridos no código fonte antes da declaração de componentes tais como classes, métodos, propriedades, interfaces ou enumerações. Esses comentários possuem uma descrição dos componentes de *software* e podem incluir exemplos de utilização do componente. No caso de métodos, funções e interfaces podem também descrever os parâmetros que devem ser passados e os valores retornados. Ao referenciar componentes documentados enquanto se escreve código no IDE, são apresentadas automaticamente as descrições definidas nos comentários de documentação XML, acelerando o desenvolvimento, pois a documentação é apresentada automaticamente quando é mais necessária. Adicionalmente é possível compilar todos os comentários de documentação XML em formatos que facilitam leitura com a aplicação Doxygen, esta aplicação permite gerar a documentação em HTML, LaTeX, PDF, RTF, entre outros formatos e gerar diagramas de classes.

A documentação das bases de dados é feita com a aplicação "Database Note Taker", esta aplicação liga-se a uma base de dados existente, carrega objetos da base de dados (tabelas, colunas, esquemas, stored procedures, etc.) da base de dados, permite que o utilizador descreva os objetos da base de dados e gera a documentação automaticamente

com base nas descrições fornecidas pelo utilizador. Posteriormente, se forem feitas alterações à base de dados é possível ligar à base de dados, sincronizar os objetos, descrever os novos objetos e gerar documentação atualizada, sendo que a aplicação guarda o histórico das alterações à base de dados.

# 5. Arquitetura

Neste capítulo serão apresentados os vários componentes que compõem o sistema desenvolvido para este projeto e como esses componentes interagem com outros componentes e com os utilizadores. Na seção da arquitetura lógica serão apresentados os componentes conceptuais do projeto sem especificar implementações. Na seção da arquitetura física serão apresentados os vários componentes de *software* e *hardware* utilizados para implementar o sistema.

## 5.1. Arquitetura lógica

Do ponto de vista lógico o projeto pode ser dividido em cinco partes; os contadores, os *gateways*, os servidores de suporte à rede LoRa, os servidores de suporte ao *software* desenvolvido e a interface com o utilizador. Os contadores são responsáveis por adquirir e transmitir dados, receber e executar comandos e receber e alterar configurações. Os *gateways* fazem parte da especificação LoRaWAN e têm como principal função servir de intermediários entre uma rede LoRa e uma rede IP que pode ser, por exemplo, a Internet ou uma rede local. O servidor de rede LoRa faz parte da especificação LoRaWAN e tem como principal função agregar um conjunto de *gateways* num ponto central, de modo a formar uma rede LoRa com múltiplos pontos de acesso para cobrir uma área geográfica maior. Os servidores de aplicação LoRa fazem parte da especificação LoRaWAN e têm como principais funções facilitar a integração de aplicações com a rede LoRa e permitir partilhar uma rede LoRa com múltiplas aplicações distintas. A "aplicação" deste sistema é composta por todos os componentes de *software* desenvolvidos, tem como principais funções receber e enviar mensagens de e para os contadores através do servidor de aplicação LoRa, gerir contadores, *gateways* e utilizadores, permitir visualizar leituras, alertas, resultados de operações dos contadores e entradas no *log*, diagnosticar erros e resolver problemas. A interface de utilizador é servida pela aplicação, fornece um meio a partir do qual os utilizadores podem efetuar operações e visualizar resultados, podendo ser acedida por mais do que um utilizador em simultâneo. A base de dados é utilizada pela aplicação para armazenar dados a longo prazo, permitindo que a aplicação seja reiniciada sem perder dados e que os dados estejam disponíveis para serem consultados quando necessário. A Figura 5-1 mostra o diagrama da arquitetura lógica.

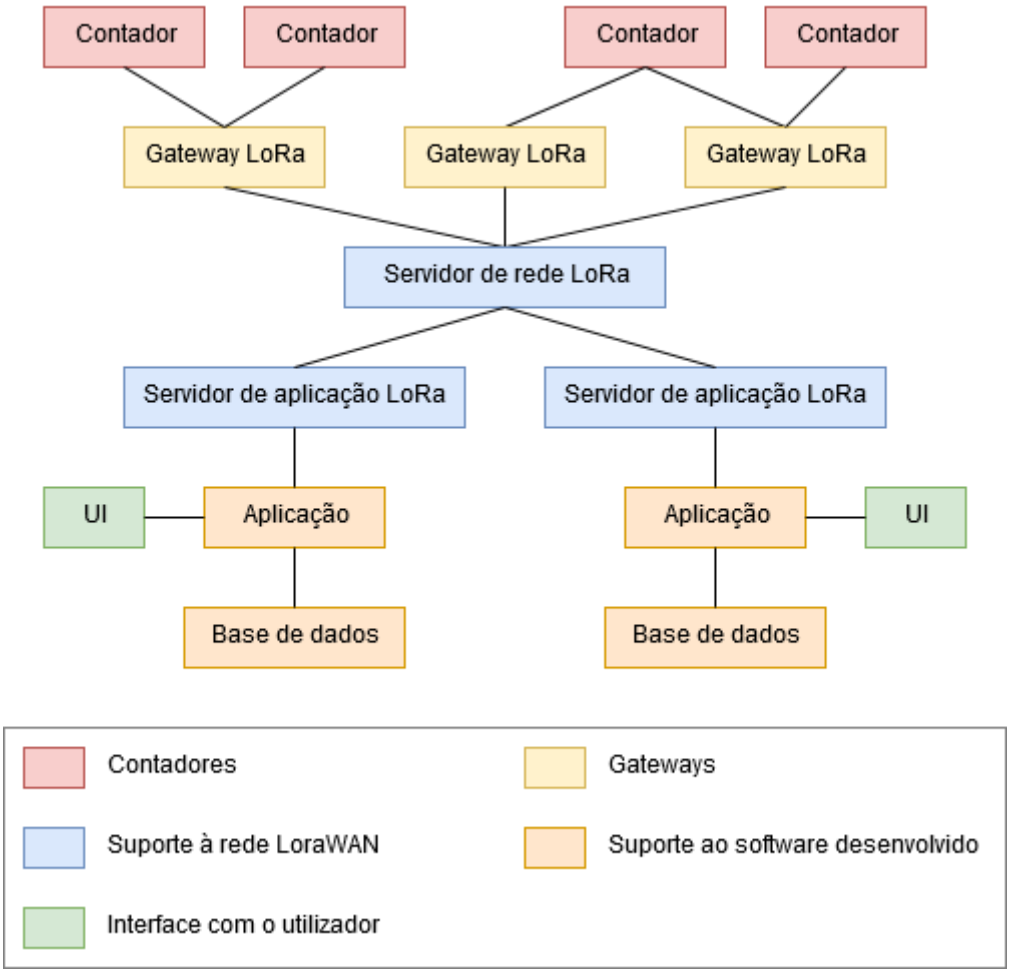



**Figura 5-1 - Arquitetura lógica**

Neste diagrama existem quatro contadores, dois desses contadores (mais à esquerda) estão dentro do alcance de apenas um *gateway*, um terceiro contador está dentro do alcance de dois *gateways* e um quarto contador (mais à direita) está dentro do alcance de apenas um *gateway*. Existe também um servidor de rede LoRa que serve de suporte a toda a rede, dois servidores de aplicação LoRa que suportam aplicações distintas que partilham a mesma rede LoRa para comunicar com os contadores, duas aplicações distintas e respetivas bases de dados e interfaces de utilizador.

## 5.2. Arquitetura física

Neste projeto existem vários componentes implementados na forma de *software* e protocolos que permitem a comunicação entre esses componentes. De seguida são apresentados os componentes de *software* concretos que foram utilizados neste projeto.

- O *software* do contador transmite leituras periodicamente ou quando solicitado, envia alertas sobre tentativas de fraude ou problemas com o microcontrolador, recebe e executa comandos e recebe e aplica novas configurações. Comunica com o "packet forwarder" através de LoRaWAN.
- O "packet forwarder" converte mensagens dos contadores, recebidas pelo *chipset* de rádio LoRa, em mensagens no formato JSON e mensagens no

formato JSON em mensagens para os contadores, enviadas pelo *chipset* de rádio LoRa. Comunica com os contadores através do protocolo LoRaWAN e com o "LoRa Gateway Bridge" através de UDP.

- A "LoRa Gateway Bridge" converte mensagens do "LoRa Server" para o formato JSON compatível com o "packet forwarder" e converte mensagens no formato JSON do "packet forwarder" em mensagens para o "LoRa Server". Comunica com o "LoRa Server" através do protocolo MQTT e com o "packet forwarder" através do protocolo UDP.
- O "Eclipse Mosquitto" atua como mediador MQTT entre a "LoRa Gateway Bridge" e o "LoRa Server".
- O "LoRa Server" encaminha mensagens dos contadores recebidas pela "LoRa Gateway Bridge" para o "LoRa App Server" e vice-versa, encripta as mensagens da aplicação com uma chave de rede de 128 bits, gere o acesso à rede LoRaWAN na segunda camada do modelo OSI (MAC), remove mensagens duplicadas quando mais do que um *gateway* recebe a mesma mensagem de um contador, agenda o envio de mensagens para os contadores e decide qual é o melhor *gateway* para enviar mensagens para cada contador. Comunica com a "LoRa Gateway Bridge" através do protocolo MQTT e com o "Lora App Server" através do protocolo gRPC.
- O "LoRa App Server" encaminha mensagens dos contadores recebidas do "LoRa Server" para o "Node-RED" e vice-versa, encripta as mensagens da aplicação com uma chave de aplicação de 128 bits, gere os *gateways* que fazem parte da rede LoRa, gere os contadores associados à aplicação e trata dos pedidos de associação dos contadores com a rede LoRa. Comunica com o "LoRa Server" através do protocolo gRPC e com o "Node-RED" através do protocolo MQTT.
- Mais uma vez o "Eclipse Mosquitto" atua como mediador MQTT, neste caso entre o "LoRa App Server" e o "Node-RED".
- O "Node-RED" converte os comandos do *website* em mensagens que seguem o protocolo de rádio binário e envia essas mensagens para os contadores através do "Lora App Server" e, de forma inversa, converte mensagens dos contadores que estão no formato do protocolo de rádio em pedidos para o *web service* no IIS seguindo a especificação do *web service*. Comunica com o "LoRa App Server" através do protocolo MQTT e com o IIS através do protocolo HTTP.
- O IIS serve um dos *web services* e o *website*, o *website* permite administrar o sistema, o *web service* recebe e processa pedidos do Node-RED e fornece configurações ao Node-RED. Comunica com o "Node-RED" através do protocolo HTTP, com o "*Web browser*" através do protocolo HTTP e com o SQL Server através do protocolo TCP.
- O "SQL Server" armazena dados a longo prazo e disponibiliza esses dados quando necessário. Comunica com o IIS através do protocolo TCP.

Esta é uma vista geral dos componentes de *software*, no capítulo 7 estes componentes são apresentados com mais detalhe. O packet forwarder, LoRa Gateway Bridge, LoRa Server, Eclipse Mosquitto, LoRa App Server e o Node-RED são gratuitos e possuem código fonte aberto.

A Figura 5-2 mostra o diagrama de componentes de *software*, é apresentado um componente por tipo de modo a não tornar o diagrama confuso, também são mostradas

as ligações entre os componentes especificando os respetivos protocolos de comunicação.

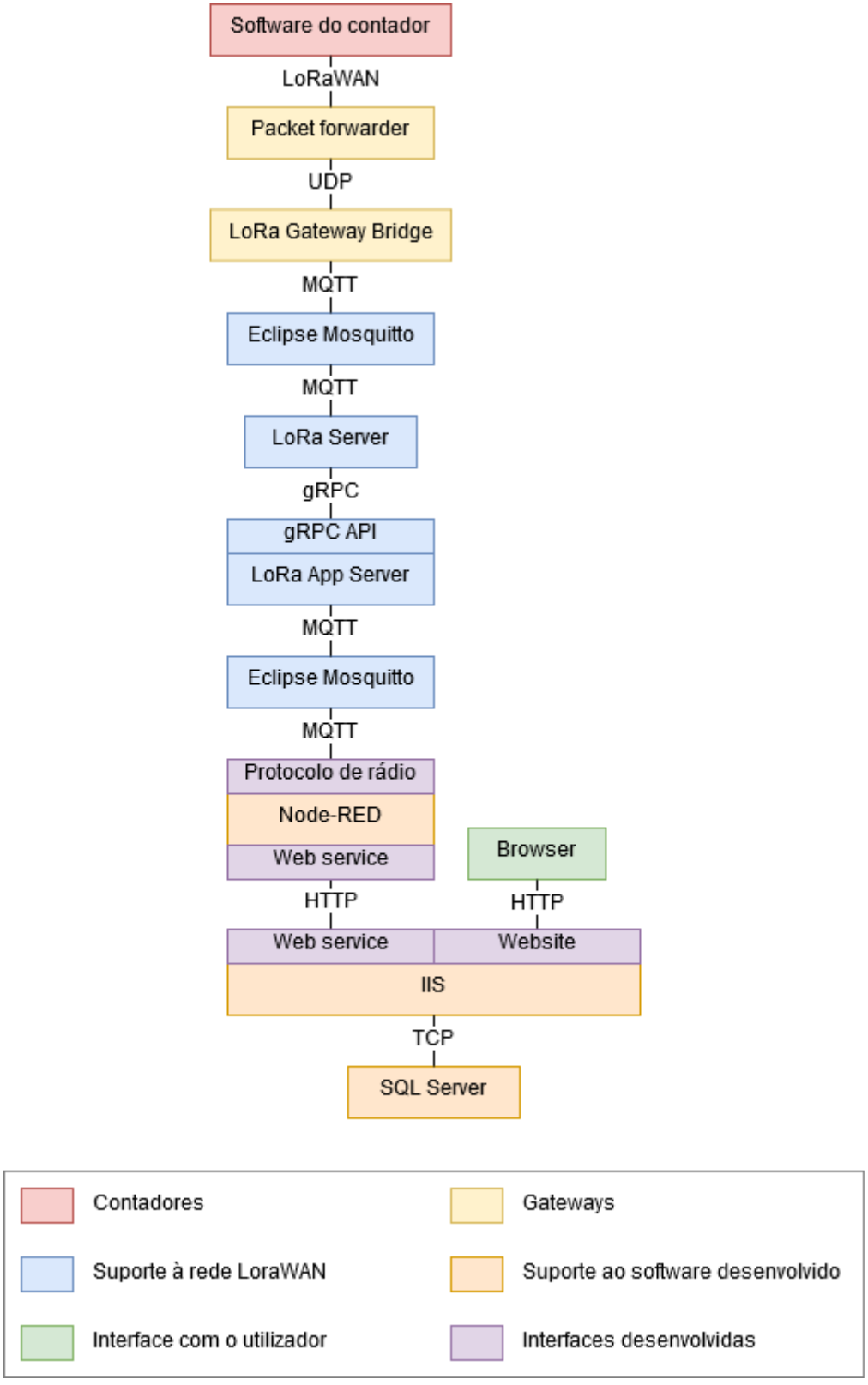



**Figura 5-2 – Diagrama de componentes de *software***

O projeto é composto por vários componentes de *hardware* onde o *software* é executado. De seguida são apresentados os componentes de *hardware* que foram

utilizados neste projeto, o *software* que executa em cada componente de *hardware* e as tecnologias de rede utilizadas para transmitir dados.

- Os contadores executam uma aplicação que foi desenvolvida com recurso a ".Net Micro Framework", esta plataforma fornece funcionalidades semelhantes a um sistema operativo pois fornece uma camada de abstração de *hardware*, suporta *drivers* de dispositivos, fornece um escalonador de *threads* de modo a suportar *multithreading*, faz gestão de memória, entre outros. Comunica com os *gateways* através de LoRa.
- Os *gateways* executam a LoRa Gateway Bridge e o packet forwarder e possuem o sistema operativo Linux. Comunicam com os contadores através de LoRa e com o servidor do LoRa Server através da Internet (ligação GPRS, HSPA+ ou LTE dependendo das condições de sinal da rede móvel).
- Existe uma máquina virtual dedicada para o LoRa Server e uma instancia do Eclipse Mosquitto, esta máquina possui o sistema operativo Linux. O Eclipse Mosquitto instalado nesta máquina serve como mediador MQTT entre os *gateways* e o LoRa Server. Comunica com os *gateways* através da Internet (ligação Ethernet) e com a máquina que possui o LoRa App Server através de Ethernet.
- Existe uma máquina virtual dedicada para o LoRa App Server e uma instancia do Eclipse Mosquitto (diferente da instancia instalada com o LoRa Server), esta máquina possui o sistema operativo Linux. O Eclipse Mosquitto instalado nesta máquina serve como mediador MQTT entre o LoRa App Server e o Node-RED. Comunica com a máquina que possui o LoRa Server através de Ethernet e com a máquina que possui o Node-RED também através de Ethernet.
- Existe uma máquina virtual dedicada para o Node-RED, esta máquina possui o sistema operativo Linux. Comunica com a máquina que possui o LoRa App Server através de Ethernet e com a máquina que possui o IIS também através de Ethernet.
- Existe uma máquina virtual dedicada para o IIS e para o SQL Server, esta máquina possui o sistema operativo Windows. Comunica com a máquina que possui o Node-RED através de Ethernet e com a máquina que possui o *web browser* através da Internet (ligação Ethernet).
- O *web browser* utilizado para aceder ao *website* de administração não depende de nenhum sistema operativo, pode ser utilizado qualquer sistema operativo desde que seja utilizado um *web browser* recente, caso seja utilizado um *web browser* antigo algumas funcionalidades podem não estar disponíveis. Comunica com a máquina que possui o IIS através da Internet (ligação Wi-Fi ou Ethernet).

.

A Figura 5-3 mostra os componentes de *hardware* utilizados neste projeto, cada componente no diagrama representa uma máquina distinta, o nome do componente de *hardware* e do respetivo sistema operativo encontra-se na parte de baixo marcado com uma cor, o nome dos componentes de *software* que executam nesse *hardware* encontram-se na parte de cima a cinzento.

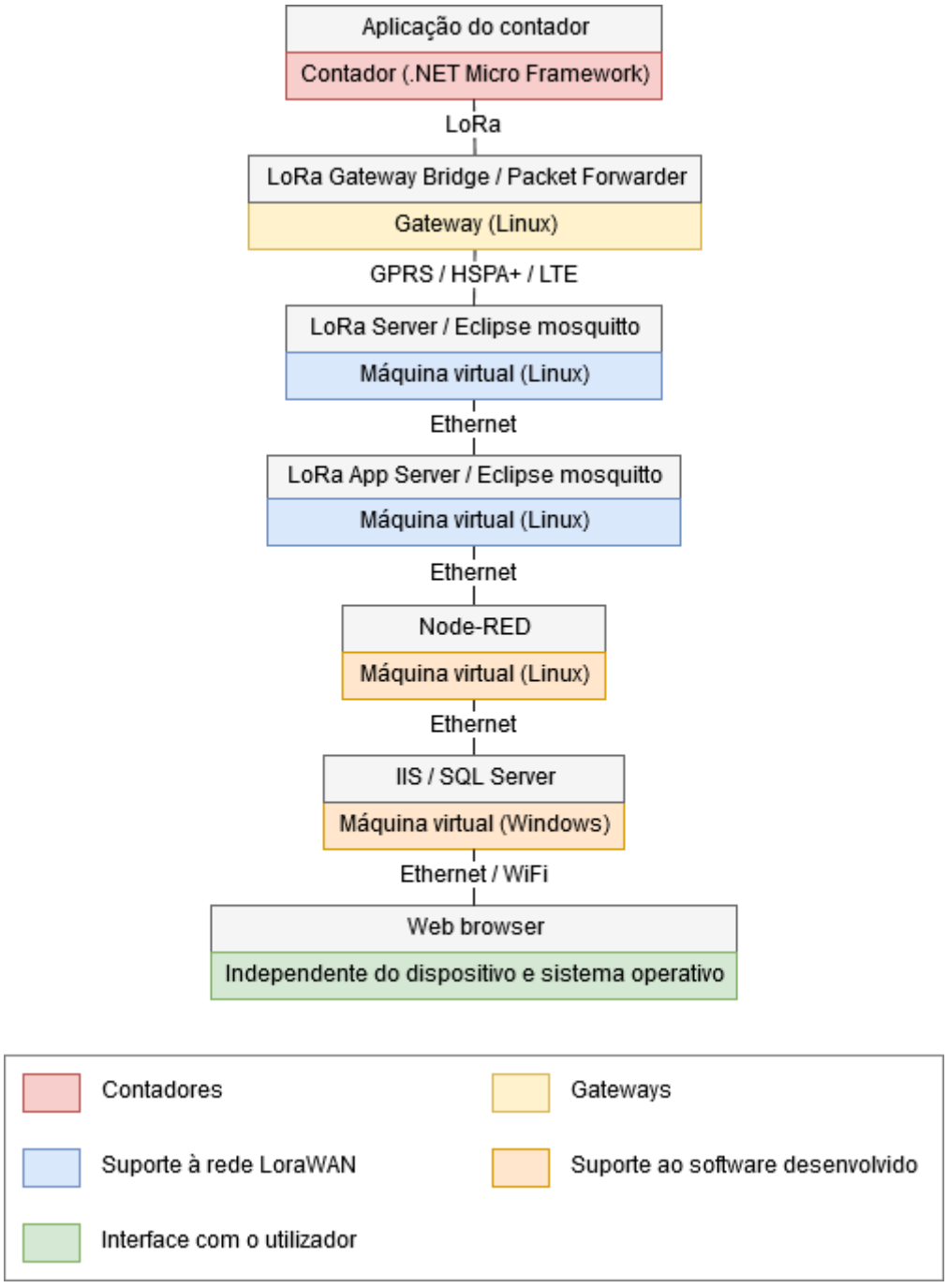



**Figura 5-3 – Diagrama de componentes de *hardware***

Os contadores foram desenvolvidos pelos meus colegas, na versão final os *gateways* utilizados foram desenvolvidos pela TULAlabs e as máquinas virtuais são fornecidas pelo serviço Amazon EC2. A tecnologia LoRa é necessária para comunicar com os contadores atuais pois estes não possuem outros meios de comunicação remota, para comunicar com os contadores através de outras tecnologias de redes sem fios (p. ex.: Zigbee) seria necessário adicionar suporte para essa tecnologia nos contadores e nos *gateways*, entre os restantes componentes de *hardware* é possível utilizar outras tecnologias de rede desde que essas tecnologias suportem o encapsulamento do protocolo IP. A única restrição em relação ao tipo de arquitetura de CPU é relativa ao IIS, este era um requisito e só é suportado em Windows na microarquitectura x86 e x86-64, é possível executar os restantes componentes noutra arquitetura de CPU, por

exemplo ARMv8-A. Esta é uma vista geral dos componentes de *hardware* e a sua relação com os componentes de *software*, no capítulo 6 estes componentes são apresentados com mais detalhe.

# 6. Hardware

Neste capítulo vão ser apresentados os componentes de *hardware* utilizados neste projeto. Serão apresentados os componentes que compõem cada dispositivo de *hardware*, os meios de comunicação que cada dispositivo possui e, caso se aplique, as suas especificações. Alguns componentes têm um dispositivo físico dedicado, como é o caso dos contadores e *gateways*, outros podem partilhar o mesmo dispositivo físico, como é o caso das máquinas virtuais.

## 6.1. Contadores

Os contadores de água são os "*end devices*" deste projeto. São compostos por um conjunto de atuadores e sensores ligados a um microcontrolador que comunica sem fios através da tecnologia LoRa. Possuem um atuador que é uma válvula eletromecânica. Como sensores possuem sensor de fluxo de água, sensor de campo magnético, sensor da posição da válvula, sensor de tensão da bateria e acelerómetro. Podem também possuir um recetor de GPS mas este será instalado apenas em contadores de teste, devido ao custo relativamente elevado. A Figura 6-1 mostra uma foto de um dos contadores de água.

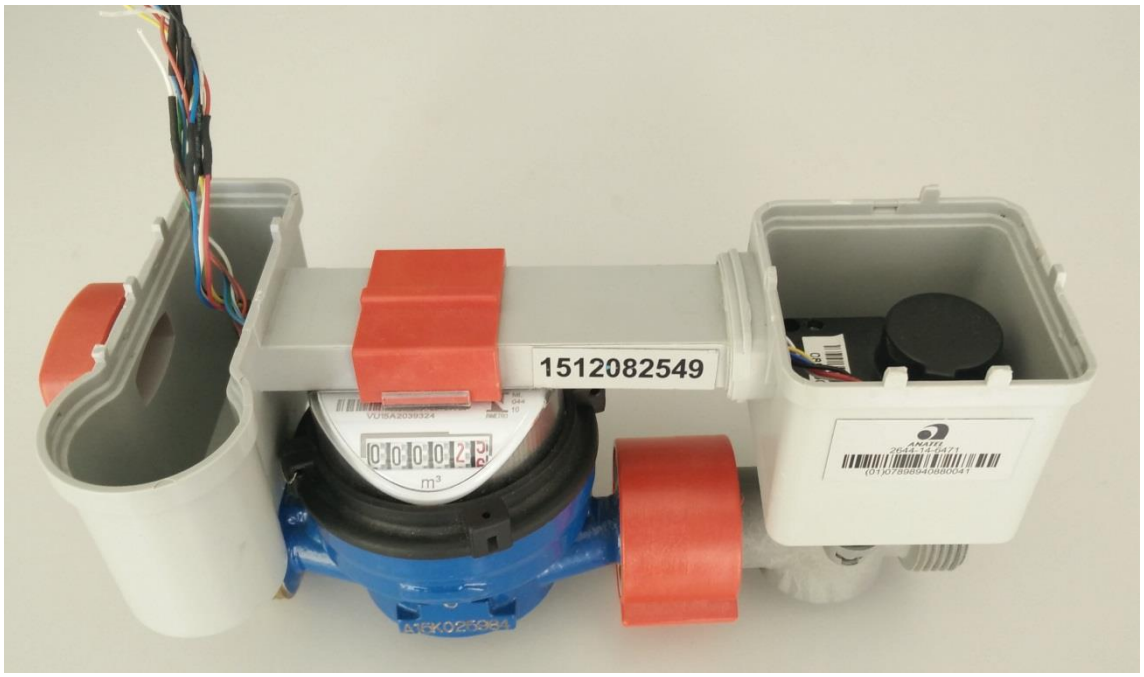



**Figura 6-1 - Contador de água**

A válvula eletromecânica serve para interromper o fluxo de água, o sensor de fluxo de água efetua a leitura do volume de água consumido e deteta se o fluxo está no sentido contrário. O sensor de campo magnético deteta tentativas de fraude das leituras do consumo de água com recurso a ímanes ou eletroímanes. O sensor de posição da válvula está incorporado na própria válvula eletromecânica e serve para detetar se esta está aberta, fechada ou numa posição intermediária. O sensor de tensão de bateria permite estimar a energia restante na bateria, pois a taxa média de descarga é baixa e portanto o valor da tensão é pouco afetado pelo consumo de energia dos diversos componentes (existem planos para incorporar futuramente um sistema gestor de bateria, ou BMS, de

modo a obter melhores estimativas da capacidade restante da bateria, mas a versão final do contador não possui um sistema gestor de bateria). O acelerómetro deteta tentativas de mover o contador do local onde foi instalado. O recetor GPS serve para efetuar o mapeamento de sinal no terreno. O microcontrolador recebe uma mensagem de pedido de mapeamento de sinal, mede as condições de receção dessa mensagem e envia uma mensagem de resposta que contém as condições do sinal, as coordenadas e a altitude do local onde a medição foi feita. O microcontrolador possui um módulo de rádio LoRa através do qual comunica com *gateways* LoRa, os *gateways* por sua vez encaminham as mensagens do microcontrolador para o LoRa Server, sendo este o canal de comunicação com o resto do sistema, pois o microcontrolador não tem acesso direto a uma rede IP.

O microcontrolador do contador é alimentado por uma bateria de lítio, funcionando constantemente desde que é instalado. O sistema foi desenvolvido com esta limitação como prioridade, reduzindo o consumo de energia sempre que possível, a capacidade da bateria foi então calculada para durar cerca de cinco anos em condições de utilização normais, sendo que a utilização excessiva da válvula eletromecânica ou comunicação excessiva reduzem o tempo de vida útil da bateria. Quando a bateria tem pouca energia restante o microcontrolador envia uma mensagem de alerta, adicionalmente é possível obter o valor da tensão da bateria enviando um comando a partir do *website* de administração. A Figura 6-2 mostra uma foto da placa de circuito impresso do contador desenvolvido pela TULAlabs, onde se pode ver o microcontrolador. A Figura 6-3 mostra uma foto do outro lado da mesma placa de circuito impresso onde se pode ver o módulo de rádio LoRa.

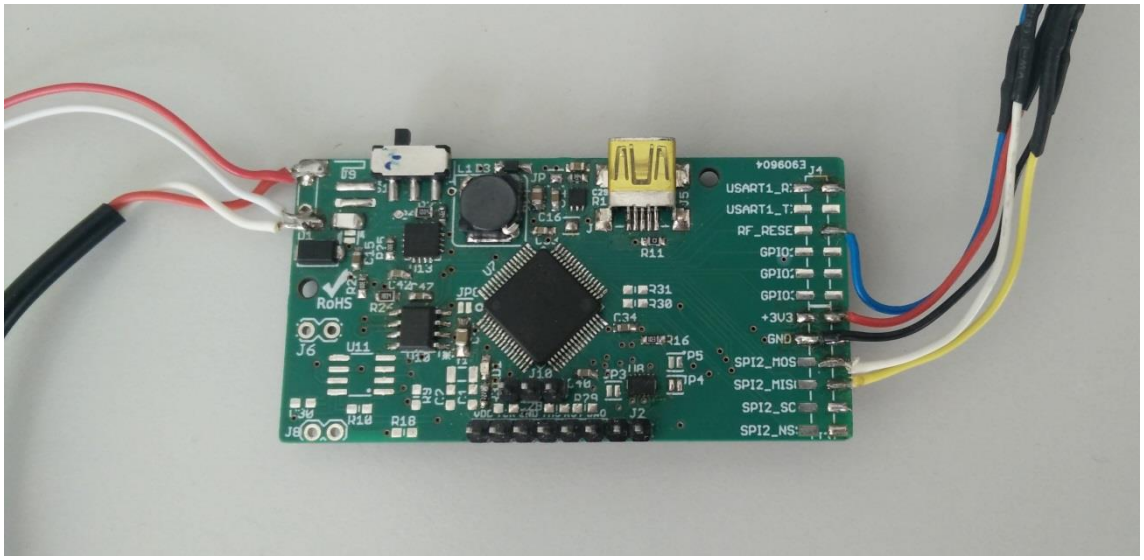



**Figura 6-2 – Placa de circuito impresso do contador**

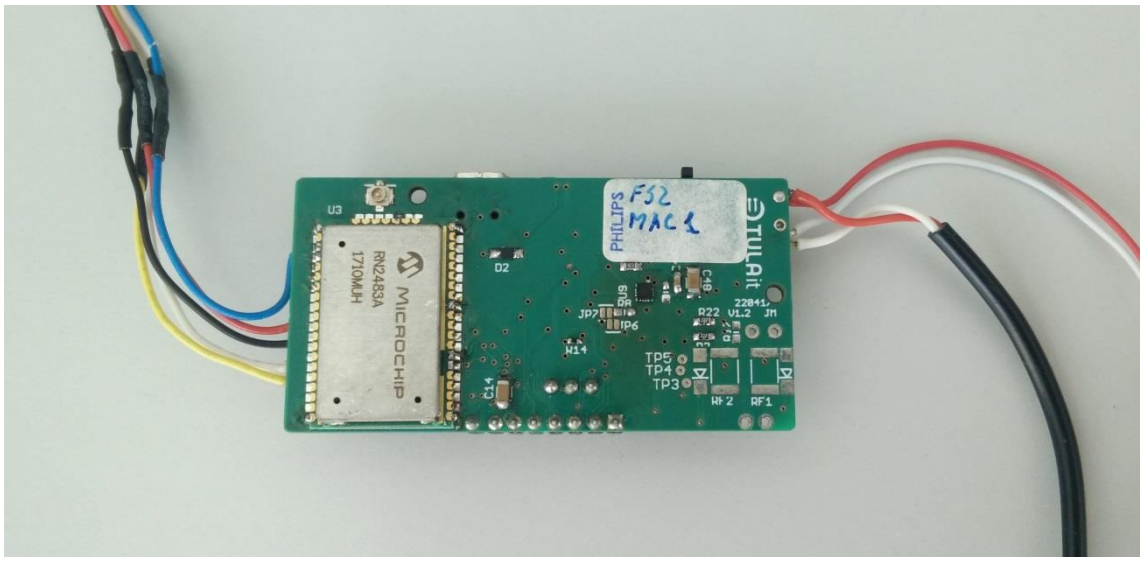



**Figura 6-3 - Outro lado da placa de circuito impresso do contador**

# 6.2. Gateways

O *gateway* utilizado inicialmente foi o MultiConnect Conduit da MultiTech[7]. Segundo o fabricante, este *gateway* serve para interligar várias tecnologias de redes sem fios e de redes cabladas. Este *gateway* é programável e permite que sejam instaladas diversas placas de entrada/saída do fabricante, como por exemplo placas RS-232, GPIO e Ethernet. Uma dessas placas, a "mCard LoRaWAN", fornece uma interface de rádio LoRa que é utilizada para comunicar com os contadores, segundo o fabricante pode suportar milhares de dispositivos que utilizem o módulo de rádio LoRa Multiconnect mDot.

Em relação ao software incorpora o packet forwarder, um Network Server (parte da especificação LoRaWAN) proprietário, um Application Server (parte da especificação LoRaWAN) proprietário e o Node-RED. Não incorpora uma Gateway Bridge pois não há necessidade de encriptar ou transmitir as mensagens pela rede, já que todos servidores LoRaWAN se encontram no *gateway*. O sistema operativo que utiliza é uma versão personalizada de Linux desenvolvida pela Multitech, chamada "mLinux", que se baseia em OpenEmbedded-Core, é de código fonte aberto e serve de base para alguns dos produtos da Multitech. A Figura 6-4 mostra uma foto do *gateway* MultiConnect Conduit.


**Figura 6-4 - Gateway MultiConnect Conduit (imagem reproduzida de [8])**

Pode ser programado de duas formas: compilando uma imagem personalizada do sistema "mLinux", contendo a aplicação desenvolvida e gravando essa imagem na memória *flash* interna ou desenvolvendo a aplicação no Node-RED, que já vem incluído na imagem de fábrica. A opção de desenvolvimento escolhida foi o Node-RED, pois permite testar e resolver problemas rapidamente, não sendo necessário compilar e gravar novas imagens do sistema de cada vez que é feita alguma alteração ao código, adicionalmente todo o *software* pode ser configurado em apenas uma interface. Estas características foram importantes, pois reduziram o tempo necessário para desenvolver o *software* do *gateway* e permitiram que o sistema fosse testado e demonstrado ao cliente dentro dos prazos. A imagem de fábrica do *gateway* possui uma interface *web* onde é possível configurar o sistema Linux, as placas de entrada/saída e todo o *software* incluído, tal como o Network Server ou Application Server. As especificações de *hardware* do *gateway* MultiConnect Conduit encontram-se na Tabela 6-1

| | |
|---|---|
| **CPU** | 400 MHz 32-bit ARM9, 16KB cache de dados, 16KB cache de instruções |
| **RAM** | 256 MB DDR |
| **Armazenamento** | 256 MB memória flash, cartão micro SD |
| **Conectividade** | LoRa, 3G, Wi-Fi, Bluetooth, Ethernet, Porta série |
| **Frequências LoRa suportadas** | 868 MHz (EU), 915 MHz (US), 923 MHz (AS) |
| **Rede celular** | HSPA+, GPRS |
| **Wi-Fi** | 802.11 a/b/n/g 2.4 Ghz, 5 Ghz |
| **Bluetooth** | BLE 4.1 |
| **Ethernet** | 10/100 Mbps |
| **Porta série** | RS-232, RS-485 |
| **USB** | 2 x USB A, 1 x micro USB (debug) |
| **Tensão elétrica** | 9V a 32V DC |
| **Temperatura tolerada em funcionamento** | -30°C a +70°C |
| **Humidade relativa tolerada** | 20% a 90% |
| **Material da caixa** | Metal |

**Tabela 6-1 – Especificações de *hardware* do *gateway* MultiConnect Conduit**

A meio do desenvolvimento do *software* do *gateway*, foi decidido não utilizar o *gateway* da Multitech. Esta decisão foi tomada depois de testar o *gateway* da Multitech em ambiente urbano e chegar à conclusão que seriam necessários demasiados *gateways* para cobrir áreas significativas, o que tornava custo do projeto demasiado elevado. Adicionalmente foram feitos testes de carga no *gateway* da Multitech que revelaram que o *hardware* não era adequado para suportar o Node-RED. Assim, foi desenvolvido um novo *gateway* pelos meus colegas que utiliza o *software* LoRa Gateway Bridge do projeto LoRa Server[9]. O número de funções do novo *gateway* foi reduzido, libertando recursos tais como tempo de processamento e memória, para que esses recursos possam ser utilizados no encaminhamento de mensagens e na execução de outras tarefas necessárias ao projeto. A Figura 6-5 mostra as funções do *gateway* da Multitech no projeto e as funções do novo *gateway* que o substituiu.

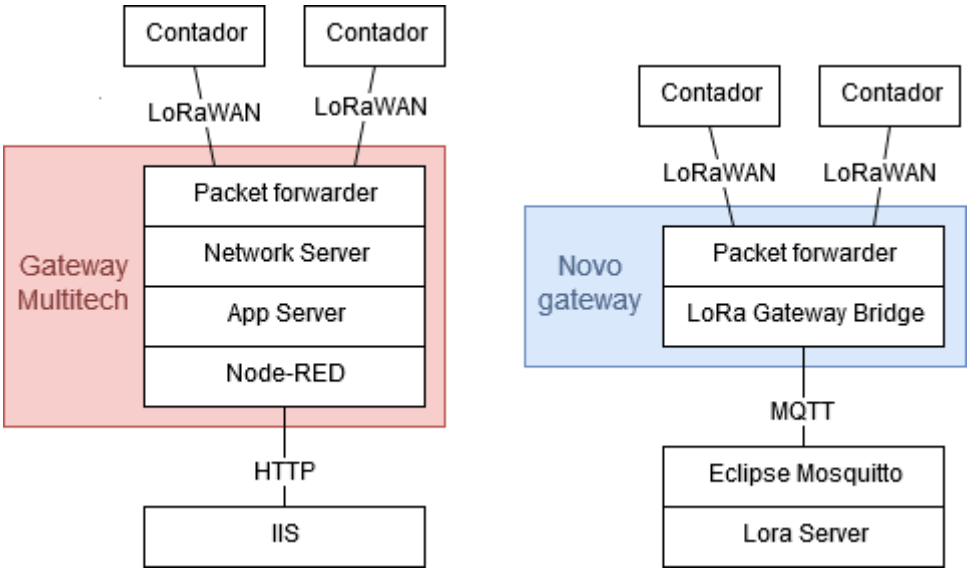



**Figura 6-5 - Funções do *gateway* da Multitech e do novo *gateway***

O *software* de suporte à rede LoRaWAN (Network Server e App Server) e o Node-RED passaram para máquinas virtuais dedicadas alojadas no serviço Amazon EC2. Neste novo cenário foi utilizado o Network Server "LoRa Server" e o App Server "LoRa App Server" do projeto LoRa Server[9], o sistema desenvolvido no Node-RED para o *gateway* da Multitech foi adaptado para passar a servir de intermediário entre o LoRa App Server e o *web service* no IIS (encaminhando mensagens enviadas pelos contadores para o IIS) e também passou a servir de intermediário entre o *website* e o LoRa App Server (encaminhando mensagens do IIS para os contadores).

O *hardware* do novo *gateway* começou por basear-se no Raspberry Pi 2 Model B. A Figura 6-6 mostra uma foto do Raspberry Pi 2 Model B.

**Figura 6-6 - Raspberry Pi 2 Model B (imagem reproduzida de [10])**

Foram feitos testes de carga e foi determinado que o novo *gateway* tinha uma performance superior e que conseguia tempos de resposta mais baixos do que o *gateway* da MultiTech. Estes resultados devem-se principalmente ao facto de o novo *gateway* não ter tantas funções e de o CPU e memória RAM serem mais rápidos. As

especificações, mais relevantes para o projeto, do novo *gateway* baseado no Raspberry Pi 2 Model B encontram-se na Tabela 6-2.

| | |
|---|---|
| **CPU** | 900MHz 32-bit quad-core ARM Cortex-A7 |
| **RAM** | 1 GB DDR2 |
| **Armazenamento** | 4 GB cartão micro SD |
| **Conectividade** | LoRa (chipset externo), Ethernet, UART, GPIO |
| **Frequências LoRa suportadas** | 433MHz (EU), 868(EU) MHz |
| **Ethernet** | 10/100 Mbps |
| **Porta série** | UART |
| **USB** | 4 x USB A, 1 x micro USB (alimentação) |
| **Tensão elétrica** | 5V DC |
| **Material da caixa** | Alumínio |

**Tabela 6-2 – Especificações do *gateway* baseado no Raspberry Pi 2 Model B**

Havia planos para desenvolver um *gateway* que não necessitasse do Raspberry Pi. Tive a informação de que depois de terminar o estágio o *hardware* do novo *gateway* foi substituído, desta vez por um *gateway* desenvolvido pela TULAlabs e que se encontra em produção em diversos projetos, sendo adaptado o sistema do novo *gateway* para funcionar no *gateway* desenvolvido pela TULAlabs.

## 6.3. Servidores

Os sistemas em produção estão instalados em máquinas virtuais alojadas no serviço Amazon EC2. Durante o desenvolvimento do sistema foram criadas várias máquinas virtuais no *hypervisor* VirtualBox, de modo a simular o ambiente de produção. Todas as máquinas virtuais utilizam a distribuição Debian do sistema operativo Linux, com a exceção da máquina que corre o IIS e o SQL Server que utiliza o sistema operativo Windows 10. A Figura 6-7 mostra a lista de todas as máquinas virtuais no VirtualBox.


**Figura 6-7 - Máquinas virtuais no VirtualBox**

Todas as máquinas virtuais possuem uma interface de rede do tipo “Host-Only”, este tipo de interface comunica com um *switch* implementado internamente em *software* pelo VirtualBox, não tendo acesso a redes externas. Existe uma máquina virtual chamada “Router” que executa um servidor DHCP, um servidor DNS e faz o encaminhamento de pacotes das outras máquinas para a Internet, esta máquina possui a interface “Host-Only” para comunicar com as outras máquinas e uma interface

"Bridged" que expõe uma interface real no computador de desenvolvimento para a máquina virtual "Router" e que permite ligar à rede local e à Internet. O computador de desenvolvimento possui uma interface "Host-Only", de modo a ser possível administrar diretamente as máquinas virtuais sem ser necessário expor, por exemplo, o servidor SSH de cada máquina Linux para a rede externa, através de "IP masquerading" (PAT) ou uma VPN. A interface "Host-Only" do computador de desenvolvimento tem apenas configurado o endereço IP e a máscara de rede, de modo a que o sistema operativo não tente utilizar a interface "Host-Only" para aceder a redes externas. A Figura 6-5 mostra um esquema da rede configurada no computador de desenvolvimento.

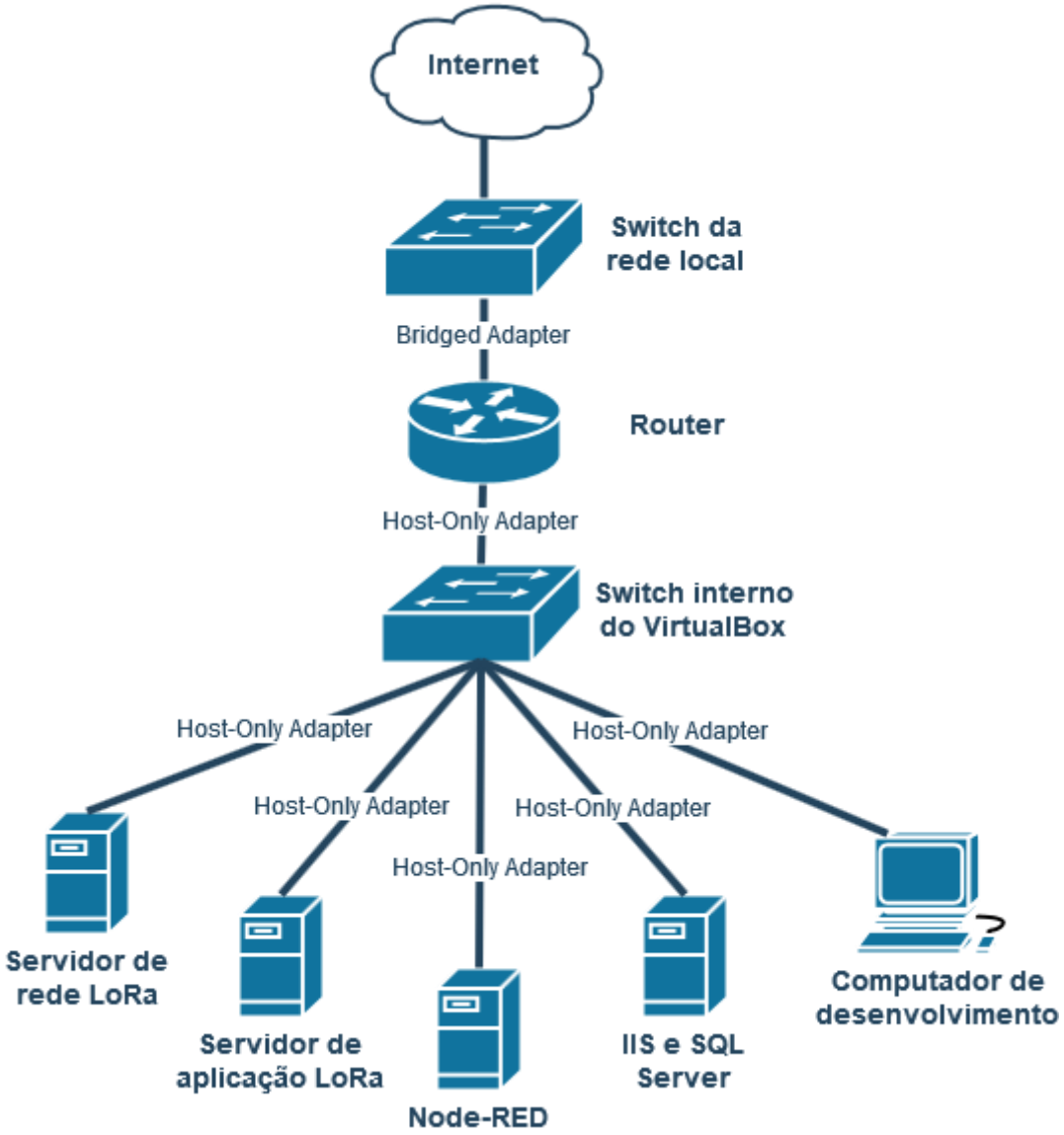



**Figura 6-8 – Esquema da rede configurada no computador de desenvolvimento**

Sendo utilizadas máquinas virtuais é possível definir algumas especificações de *hardware* alterando apenas as configurações, desde que o computador "hospedeiro" suporte essas especificações. A Tabela 6-3 mostra as especificações de *hardware* das máquinas virtuais de desenvolvimento criadas no VirtualBox.

| Nome | S.O. | N.º CPUs | RAM | Armazenamento | Conectividade |
|---|---|---|---|---|---|
| Router | Debian GNU/Linux | 1 | 64 MB | 512 MB | Host-Only Adapter 1000 Mbps |
| | | | | | Bridged Adapter 1000 Mbps |
| Servidor de rede LoRa | Debian GNU/Linux | 1 | 512 MB | 8 GB | Host-Only Adapter 1000 Mbps |
| Servidor de aplicação LoRa | Debian GNU/Linux | 1 | 512 MB | 8 GB | Host-Only Adapter 1000 Mbps |
| Node-RED | Debian GNU/Linux | 1 | 512 MB | 8 GB | Host-Only Adapter 1000 Mbps |
| IIS e SQL Server | Windows 10 Professional | 2 | 4096 MB | 60 GB | Host-Only Adapter 1000 Mbps |

**Tabela 6-3 – Especificações de hardware das máquinas virtuais no VirtualBox**

O acesso à máquina virtual que tem a função de *router*, à máquina que possui o servidor de rede LoRa, à máquina que possui o servidor de aplicação LoRa e à máquina que possui o Node-RED é feito através do protocolo SSH, o servidor SSH utilizado foi o OpenSSH (openssh-server) e os clientes SSH utilizados para administrar essas máquinas virtuais foram o OpenSSH (openssh-client) e o PuTTY. O acesso à máquina que serve o IIS e o SQL Server é feito através do protocolo RDP, o servidor RDP utilizado foi o servidor RDP incluído no Windows, o cliente RDP utilizado foi o cliente RDP incluído no Windows e o Remmina.

# 7. Software

Neste capítulo vão ser apresentados os vários componentes de *software* que compõem este projeto. Será apresentado o *software* que executa nos contadores, o *software* que executa nos *gateways*, o *software* de suporte à rede LoRa e o *software* que serve de suporte ao sistema e que fornece uma interface de administração. O *software* pode encontrar-se em *hardware* dedicado, como é o caso dos contadores, ou em *hardware* partilhado, como é o caso do IIS e SQL Server.

## 7.1. Contadores

Os contadores possuem um microcontrolador que executa *software* que tem várias responsabilidades, sendo as principais:

- Medir o consumo de água e enviar a leitura periodicamente e quando solicitado.
- Abrir ou fechar a válvula de controlo do fluxo de água quando solicitado.
- Detetar a posição da válvula de controlo do fluxo de água.
- Medir periodicamente a tensão da bateria e enviar um alerta caso esteja demasiado baixa.
- Medir a tensão da bateria periodicamente e quando solicitado.
- Colocar os sensores/atuadores em modo de baixo consumo quando não estão a ser utilizados e restaurar o funcionamento normal quando necessário.
- Gerir a fila de pedidos e respostas de comandos.
- Emitir alertas tais como “fuga de água”, “fluxo reverso” ou “bateria fraca”.

A válvula é fechada automaticamente quando é detetado um consumo constante durante um longo intervalo de tempo e o contador envia um alerta de fuga de água para o sistema, o sistema assume que existe uma fuga de água e que o cliente não se encontra em casa, pois caso estivesse em casa o consumo provavelmente iria oscilar. A válvula pode ser fechada ou aberta remotamente por um comando enviado pelo *website* de administração. Adicionalmente alguns contadores permitem fazer mapeamento das condições do sinal LoRa, com recurso a informações do módulo de rádio LoRa e do módulo recetor de GPS.

## 7.2. Packet Forwarder

O Packet Forwarder é um *software* que faz a interface com o *chipset* LoRa dos *gateways* e fornece uma interface mais simples para comunicar com os contadores por LoRa. Este *software* executa nos *gateways* e comunica com o *chipset* LoRa através de uma interface SPI, as mensagens têm o formato JSON e são enviadas/recebidas pelo protocolo UDP, abstraindo assim o protocolo e os comandos específicos do *chipset* do rádio LoRa.

## 7.3. LoRa Gateway Bridge

A LoRa Gateway Bridge é um *software* que faz a ligação entre o Packet Forwarder e o Lora Server. Este software é instalado normalmente nos *gateways* e é instalado na forma de serviço, comunicando com o Packet Forwarder que normalmente se encontra no

mesmo *gateway* e com o LoRa Server que normalmente se encontra num servidor externo.

A LoRa Gateway Bridge recebe mensagens do packet forwarder através do protocolo de transporte UDP, cria uma nova mensagem JSON compatível com o LoRa Server a partir da mensagem JSON recebida do packet forwarder e envia a nova mensagem para o LoRa Server através do protocolo MQTT. De forma inversa quando o LoRa Gateway Bridge recebe mensagens do LoRa Server através do protocolo MQTT, cria uma nova mensagem JSON compatível com o packet forwarder a partir da mensagem JSON recebida do LoRa Server e envia a nova mensagem para o packet forwarder através de UDP. O packet forwarder não pode comunicar diretamente com o LoRa Server, pois este envia e recebe dados dos contadores pelo protocolo MQTT enquanto que o packet forwarder utiliza o protocolo UDP e embora as mensagens de ambos utilizem o formato JSON, a especificação do protocolo e a estrutura das mensagens é diferente, portanto é sempre necessário algum software intermediário que converta as mensagens JSON no formato apropriado.

O packet forwarder comunica através de UDP, o que pode ser aceitável para redes locais seguras e robustas, mas não é viável para comunicação pela Internet, em que não há controlo sobre por onde os dados passam e não há garantia de que os dados cheguem ao destinatário. Adicionalmente o Packet Forwarder não fornece qualquer tipo de autenticação ou encriptação dos dados, o que permitiria a alguém mal-intencionado intercetar mensagens enviadas e recebidas pelos contadores e enviar comandos para os contadores. Estes problemas são solucionados executando uma instância da LoRa Gateway Bridge em cada *gateway*, configurado de modo a que os dados enviados e recebidos da Internet sejam protegidos por TLS e que o acesso seja autenticado por utilizador e palavra-passe. Caso múltiplos *gateways* se encontrem numa rede segura, é possível ter apenas uma instância do LoRa Gateway Bridge num servidor e ligar os packet forwarders dos *gateways* a essa instância, reduzindo a carga nos *gateways.*

## 7.4. Eclipse Mosquitto

O Eclipse Mosquitto é uma implementação gratuita e de código fonte aberto de um mediador MQTT, atuando como intermediário entre clientes MQTT. Fornece um meio centralizado onde os clientes podem criar tópicos, os tópicos atuam como canais *multicast* para onde é possível enviar mensagens, outros clientes podem subscrever a tópicos passando a receber todas as mensagens que são enviadas para esses tópicos. O Eclipse Mosquitto controla o acesso através de utilizador e palavra-passe e limita o acesso a tópicos a partir de ACLs, sendo possível utilizar a mesma instância para transmitir dados de mais do que um *tenant*, em que cada *tenant* tem apenas acesso aos seus tópicos. Neste sistema o Eclipse Mosquitto tem como principal função fornecer um canal de comunicação sobre uma rede IP entre alguns componentes de suporte à rede LoRa. Os detalhes do protocolo MQTT são apresentados na seção 8.5.

Em relação à configuração do Eclipse Mosquitto, a instância que serve de intermediário entre o Node-RED e o LoRa App Server foi configurada para permitir apenas uma mensagem no processo de envio para cada cliente MQTT. Este modo limita a performance, pois não é possível trocar mensagens em paralelo entre o Node-RED e o LoRa App Server, mas garante que as mensagens são entregues aos clientes MQTT na mesma ordem em que foram recebidas. Por exemplo, se for enviado um comando para

abrir a válvula do contador e de seguida for enviado outro comando para fechar a válvula, é esperado que a válvula fique fechada, pois a ultima mensagem enviada para o contador será a de fecho da válvula, se fosse permitido entregar as mensagens em paralelo, o estado final da válvula dependeria de uma *race condition* entre as duas mensagens de comando da válvula. Todas as instâncias do Eclipse Mosquitto foram configuradas para armazenar um total de 10000 mensagens por cliente MQTT, esta fila de mensagens é necessária pois se a capacidade de processamento do Eclipse Mosquitto ou a capacidade da rede for excedida durante um pequeno intervalo de tempo, novas mensagens recebidas no Eclipse Mosquitto podem ser armazenadas e enviadas mais tarde.

## 7.5. LoRa Server

O LoRa Server é uma implementação gratuita e de código fonte aberto de servidor de rede LoRaWAN, tem como principais funções o controlar o acesso ao meio (segunda camada do modelo OSI) em interfaces LoRa, agendar transmissões dos *gateways* para os dispositivos e desduplicar mensagens, ignorando mensagens repetidas recebidas por mais do que um *gateway*.

O LoRa Server depende de software adicional para funcionar, nomeadamente o SGBD PostgreSQL para armazenamento de dados permanentes, Redis para armazenamento de dados temporários e um mediador MQTT, sendo utilizado o mediador MQTT Mosquitto. Todo o software adicional é gratuito e de código aberto e foi instalado na mesma máquina virtual em que o LoRa Server foi instalado, que possui o sistema operativo Linux.

A configuração do LoRa Server é feita num ficheiro de configuração, onde se pode configurar os parâmetros de ligação à base de dados no PostgreSQL, os parâmetros de ligação à base de dados Redis, os parâmetros de ligação ao mediador MQTT, os tópicos MQTT utilizados para comunicar com o LoRa Gateway Bridge, parâmetros da rede tais como o identificador de rede ou os canais de rádio, parâmetros que configuram a comunicação com os dispositivos das várias classes, os parâmetros e intervalos do agendador de transmissões dos *gateways*, o tipo de estatísticas a recolher e a sua granularidade, entre outros. Algumas destas configurações e outras que não se encontram no ficheiro de configuração podem ser alteradas a partir da interface *web* do LoRa App Server.

## 7.6. LoRa App Server

O LoRa App Server é uma implementação gratuita e de código fonte aberto de servidor de aplicação LoRaWAN, tem como principais funções gerir as configurações de dispositivos e *gateways* LoRaWAN, gerir o processo de associação dos dispositivos LoRaWAN com a rede LoRa e encriptar/desencriptar as mensagens de/para os contadores.

Fornece várias APIs a partir das quais é possível integrar com outros sistemas, de entre as quais MQTT, gRPC, REST e serviços de “cloud computing” (p. ex.: Google Cloud Platform Pub/Sub). Também é possível integrar com bases de dados (p. ex.: PostgreSQL), neste caso as mensagens são armazenadas em vez de serem encaminhadas. Foi escolhida a integração MQTT, pois a integração REST não é tão

eficiente como a integração MQTT. O protocolo MQTT é binário enquanto o protocolo HTTP é de texto. Adicionalmente na data em que foi feita a migração do sistema do *gateway* no Node-RED, o Node-RED não possuía nós de entrada/saída gRPC, como ainda não estava familiarizado com o processo de desenvolvimento de novos nós para o Node-RED, decidi utilizar a integração mais apropriada que o Node-RED suportava, ou seja MQTT.

O LoRa App Server depende de software adicional para funcionar, coincidentemente requer o mesmo software que o LoRa Server, ou seja PostgreSQL para armazenamento de dados permanentes, Redis para armazenamento de dados temporários e um mediador MQTT, sendo também utilizado o mediador MQTT Mosquitto. O software adicional é gratuito e de código aberto e foi instalado na mesma máquina virtual em que o LoRa App Server foi instalado, que possui o sistema operativo Linux.

Possui uma interface *web* a partir da qual é possível configurar os *gateways*, dispositivos e aplicações (grupo de dispositivos do mesmo tipo). Nesta interface é possível definir parâmetros tais como o EUI dos dispositivos e *gateways*, o EUI e chave da aplicação. A Figura 7-1 mostra a página de configuração de dispositivos LoRaWAN na interface *web* do LoRa App Server.

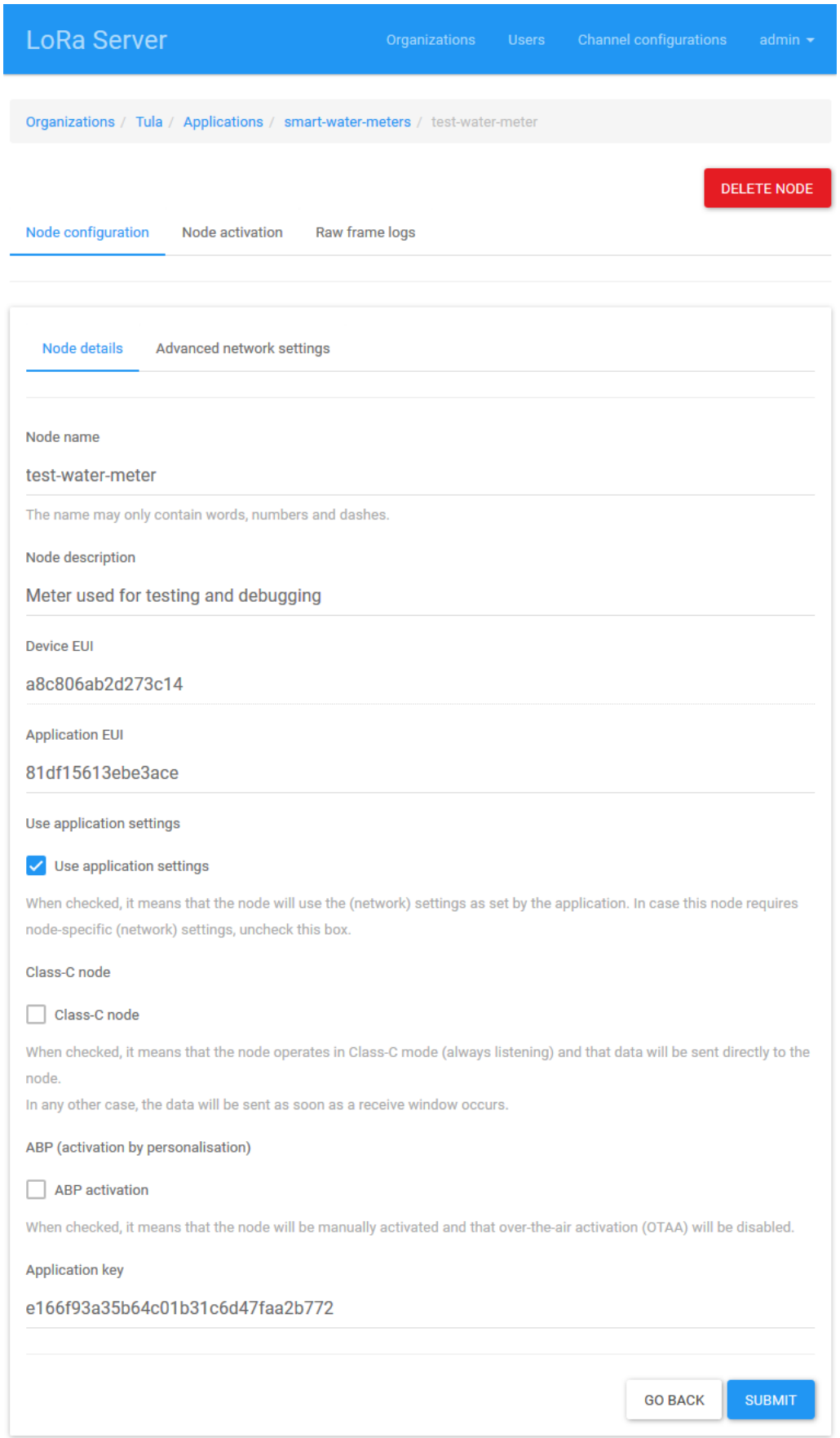



**Figura 7-1 - Configuração de dispositivos LoRaWAN no LoRa App Server**

Na interface *web* é possível configurar perfis de dispositivos, em que cada perfil define parâmetros de configuração de um tipo de dispositivo no Network Server, tais como a classe do dispositivo (A, B, C), o método de associação com a rede (OTAA ou ABP), os intervalos de transmissão e receção (RX1, RX2 window), potência isotrópica radiada equivalente máxima (Maximum EIRP), entre outros. Dispositivos que são configurados de forma semelhante podem ser associados a um perfil de dispositivo e consequentemente todas as configurações desses dispositivos passam a ser definidas pelo perfil. Caso seja necessário alterar configurações específicas individualmente por dispositivo, pode-se sobrepor as configurações do perfil na página de configuração do dispositivo.

Também é possível configurar *gateways* e perfis de *gateway* de forma semelhante aos dispositivos. Os *gateways* que são configurados de forma semelhante podem ser associados a perfis de *gateway*, de modo a que as suas configurações passem a ser definidas pelo perfil. Dentre as configurações dos *gateways* que é possível alterar na interface *web* destaca-se o EUI do *gateway*, os canais de rádio e o Network Server através do qual se liga. Adicionalmente a interface *web* mostra estatísticas sobre as *frames* enviadas e recebidas por hora, dia, semana e mês pelos *gateways*, as coordenadas e altitudes dos *gateways* e as suas localizações no mapa.

Uma instância de LoRa App Server pode-se ligar a vários LoRa Servers, os parâmetros de ligação com os LoRa Servers, tais como o IP, porto e certificados usados para TLS, podem ser configurados a partir da interface *web* do LoRa App Server, facilitando a gestão dos LoRa Servers utilizados.

É possível monitorizar eventos e mensagens encaminhadas de/para os dispositivos em tempo real na interface *web* do LoRa App Server, de modo a permitir verificar se o sistema está a funcionar normalmente ou a facilitar uma eventual resolução de problemas. O registo de eventos pode ser visualizado por dispositivo, o registo de mensagens pode ser visualizado por dispositivo e por *gateway*, permitindo monitorizar todas as mensagens encaminhadas por um determinado *gateway*.

## 7.7. Node-RED

A função do sistema implementado em Node-RED é essencialmente fazer a interface entre o *web service* e *website* no IIS com o LoRa App Server, convertendo mensagens dos contadores em pedidos HTTP para o *web service* no IIS e convertendo pedidos HTTP do *website* em mensagens para os contadores. As mensagens de/para os contadores encontram-se no Anexo A: Protocolo de rádio. Os pedidos e respostas do *web service* implementado no Node-RED encontram-se no Anexo C: Especificação do *web service* de *gateway*. Os pedidos/respostas do *web service* no IIS encontram-se no Anexo D: Especificação do *web service* no IIS.

Segundo o site do Node-RED "O Node-RED é uma ferramenta de programação para ligar dispositivos de *hardware*, APIs e serviços *online* de formas novas e interessantes."[11]. As principais vantagens do Node-RED são a facilidade com que se integra com sistemas externos, o desenvolvimento rápido de *software* e a possibilidade de executar o código em qualquer plataforma em que o Node-RED é suportado. Segue o paradigma de programação orientada a fluxo de dados em que os dados podem entrar no sistema de várias formas, são processados em vários "nós" e podem sair do sistema também de várias formas. Os "nós" são grupos de instruções que implementam uma

determinada função e que podem ser ligados a outros nós. Alguns nós vêm incluídos com a instalação do Node-RED, outros podem ser descarregados com o gestor de pacotes "npm" a partir do repositório do Node-RED. Existem nós de entrada de dados, nós de saída de dados e nós intermediários. Os nós de entrada recebem dados a partir da rede, ficheiros, etc., os nós de saída enviam dados para a rede, ficheiros, etc. e os nós intermediários permitem ler e manipular dados recebidos de outros nós e enviar o resultado para outros nós. Como exemplo um nó recebe dados em binário de um sensor de temperatura através de $I^2C$ (nó de entrada), envia os dados binários para um nó que normaliza para uma escala térmica (por exemplo graus Celsius), o nó que normaliza os dados (nó intermediário) envia os dados normalizados para outro nó que envia dados por TCP (nó de saída) e os dados normalizados são enviados para outro dispositivo pelo protocolo TCP, em o dispositivo de destino espera receber valores de temperatura na escala de graus Celsius pelo protocolo TCP.

### 7.7.1. Relação com o Node.js

O Node-RED executa como uma aplicação Node.js. Segundo o site do Node.js "O Node.js é uma plataforma de execução assíncrona de JavaScript que tem por base o motor de execução de JavaScript do Google Chrome (chamado V8) e utiliza um modelo de entrada e saída orientado a eventos, sem bloqueio."[12] O código desenvolvido em Node-RED é escrito na linguagem de programação JavaScript, podendo utilizar os módulos instalados no Node.js. Internamente os nós do Node-RED são módulos do Node.js, que contém as instruções a executar e configurações para o Node-RED. A Figura 7-2 mostra as camadas de abstração desde o *hardware* até ao sistema desenvolvido em Node-RED.

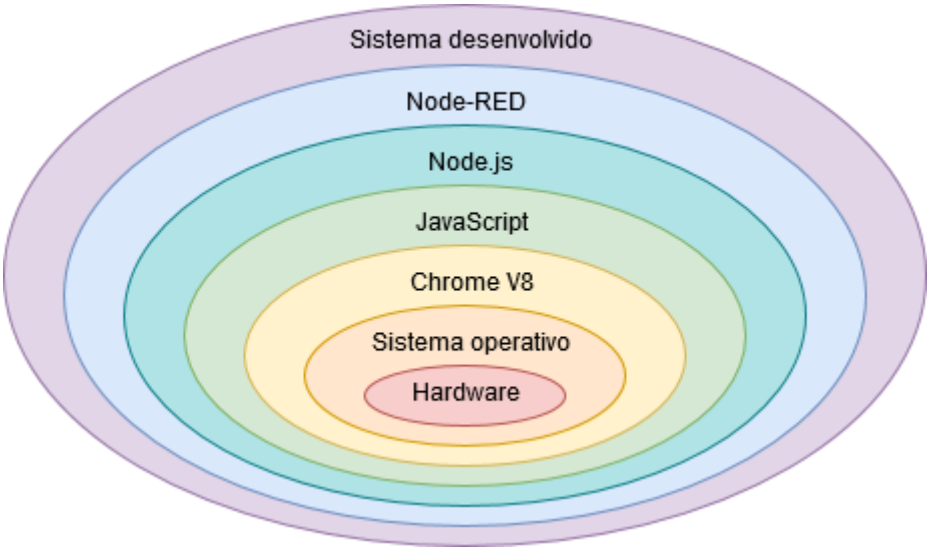



**Figura 7-2 - Camadas de abstração do sistema desenvolvido em Node-RED**

### 7.7.2. Opção pelo Node-RED

O principal motivo pelo qual foi utilizado o Node-RED para fazer a interface com o LoRa Application Server foi o facto de o *gateway* de desenvolvimento inicial ser o MultiConnect Conduit da MultiTech, o software para este *gateway* foi desenvolvido no Node-RED, pois a alternativa era desenvolver o próprio software com suporte à arquitetura do processador do *gateway*, compilar uma imagem com Linux e o software

desenvolvido embebido e gravá-la em memória flash do *gateway*, o que era um processo demorado. Quando foi tomada a decisão de desenvolver o próprio *gateway*, seguindo uma arquitetura mais próxima da especificação LoRaWAN em que os *gateways* não possuem LoRa Network Server, LoRa App Server nem Node-RED, o código desenvolvido em Node-RED foi adaptado para executar num servidor dedicado, de modo a poupar o tempo que seria necessário para implementar as suas funcionalidades e integrar o *website* e o *web service* no IIS com o LoRa App Server (ver seção 12.3).

### 7.7.3. Programação

O Node-RED inclui um IDE na forma de aplicação *web*, sendo necessário apenas um *web browser* para desenvolver *software* para o Node-RED. À esquerda no IDE do Node-RED existe uma secção que mostra os nós agrupados por categoria, o que facilita a sua pesquisa. À direita existe uma secção que mostra informações sobre o nó selecionado, mostra avisos e erros emitidos pelos nós e mostra também as variáveis guardadas no contexto do nó, no contexto do fluxo e no contexto global. No centro encontra-se a área em que é possível inserir, configurar, ligar, e remover instâncias dos nós instalados. A Figura 7-3 mostra o IDE do Node-RED. No centro são mostrados alguns dos nós que geram dados aleatório para testar o *web service* no IIS e o nó selecionado é responsável por deserializar as mensagens MQTT do LoRa App Server.

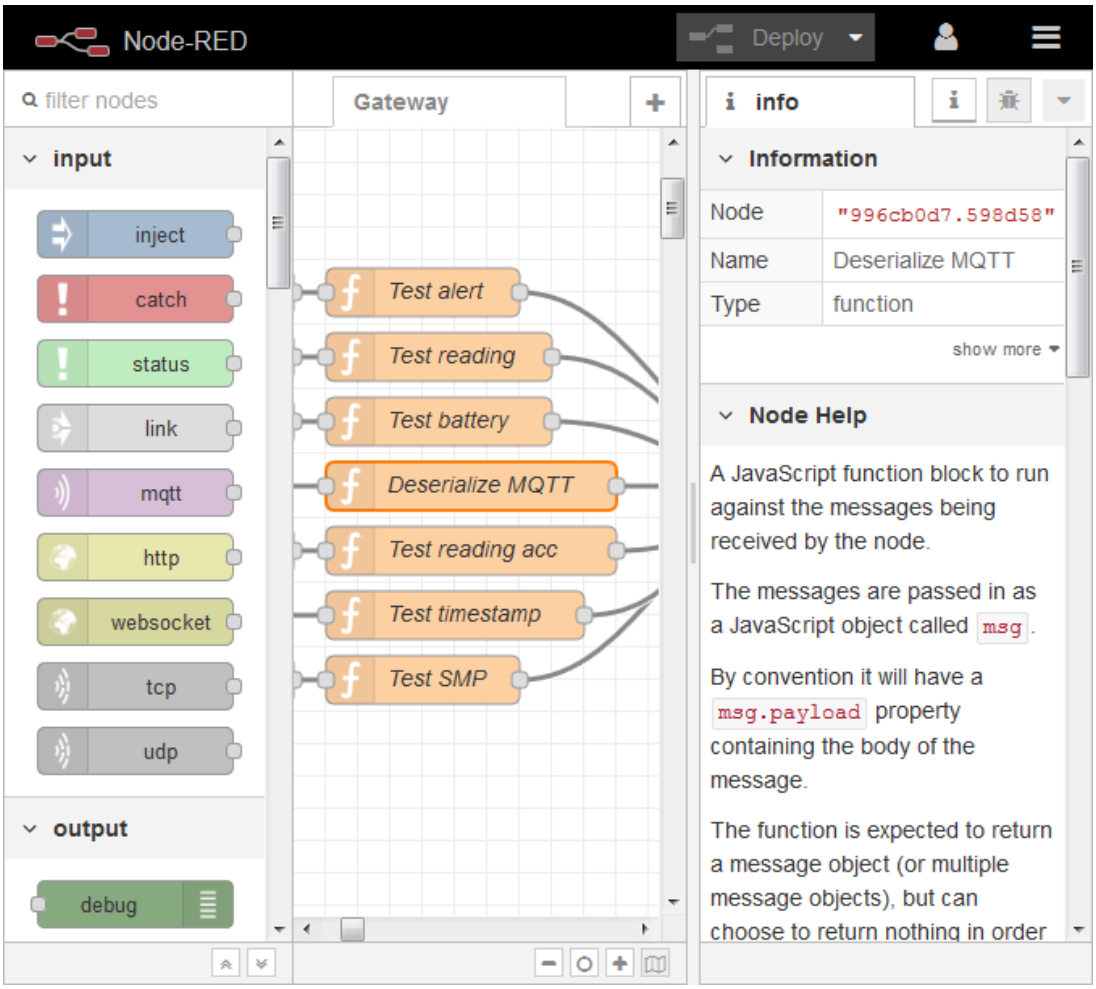



**Figura 7-3 - IDE do Node-RED**

A Figura 7-4 mostra um exemplo simples de um nó de entrada ligado a um nó de saída, a ligação entre os nós chama-se "fio" e ao conjunto de nós ligados por fios dá-se o nome de "fluxo" na nomenclatura do Node-RED, esta configuração em particular faz com que todas as mensagens MQTT da aplicação 1 sejam mostradas no separador "*debug*" do IDE:


**Figura 7-4 - Ligação entre o nó de entrada MQTT e saída debug**

A partir deste conceito de fluxo de dados é possível combinar nós e criar sistemas mais complexos com funções personalizadas, em que um nó pode encaminhar a sua saída de mensagens para múltiplos nós, pode ter mais do que uma saída de mensagens e pode também receber mensagens de vários nós.

Os nós mais importantes neste sistema são os nós laranja representados com a letra "f" seguida de um nome, pois são estes nós que permitem definir funções. O sistema poderia ser implementado por completo num único nó de função, no entanto a separação dos processos em funções mais pequenas torna o sistema mais fácil de entender e facilita os testes de *software*, a deteção de problemas e a respetiva correção. A Figura 7-5 mostra um nó de função, o fio à esquerda representa a entrada de mensagens na função, o fio à direita representa a saída de mensagens da função.

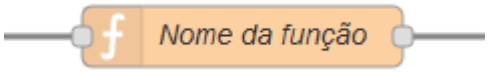



**Figura 7-5 – Nó de função**

O código dos nós de função é escrito na linguagem de programação JavaScript e têm acesso a todos os módulos incluídos com o Node.js (por exemplo o módulo "buffer"). A Figura 7-6 mostra o código de um nó de função que descodifica o código do alerta de uma mensagem de alerta do contador.

```
if(msg.payload.length !== 2) {
  node.error("Invalid alert message size");
  return null;
}
// Alert code is specified by the 6 LSB from the second byte
var alertCode = msg.payload[1] & 0x3F;
msg.payload = alertCode;
return msg;
```

**Figura 7-6 - Código que descodifica mensagens de alerta dos contadores**

O Node-RED é uma plataforma relativamente recente, a primeira versão publicada no GitHub foi em 2013 e desde então o seu desenvolvimento tem sido rápido, com uma nova versão quase todos os meses. As novas versões corrigem *bugs* e adicionam funcionalidades úteis mas, por vezes, incluíam alterações que faziam com que o sistema desenvolvido no Node-RED deixasse de funcionar, sendo necessário várias horas para tornar o código compatível com a nova versão. Paralelamente o Node.js, que serve de base ao Node-RED, também teve um desenvolvimento rápido, o que por vezes obrigou a atualizar ambos pois as novas versões de Node-RED já não eram compatíveis com a versão de Node.js instalada, dificultando a atualização do sistema.

Quando foi feita a migração do código do *gateway* da Multitech para uma máquina virtual dedicado, as alterações foram significativas, pois o Node-RED incluído no *gateway* da Multitech ainda estava na versão 0.11.1 (publicada em Julho de 2015), enquanto que a ultima versão publicada no GitHub era a 0.16.2 (publicada em Janeiro de 2017). Adicionalmente o Node.js incluído no *gateway* da Multitech ainda estava na versão a 0.10.40 (publicada em Julho de 2015), enquanto que a última versão publicada no GitHub era a 7.6.0 (publicada em Fevereiro de 2017). Apesar de gastar algum tempo a atualizar o sistema, valeu quase sempre a pena pois os *bugs* que por vezes causavam falhas em algumas operações foram resolvidos e algumas funções personalizadas foram simplificadas ou removidas, porque partes ou o todo das funcionalidades implementadas por essas funções passaram a ser também implementadas pelo Node-RED, então essas partes foram substituídas pela implementação do Node-RED, passando a responsabilidade de manter esse código para a equipa de desenvolvimento do Node-RED.

### 7.7.4. Pedidos para o web service no IIS

As mensagens dos contadores são encaminhadas pelo LoRa App Server para o Node-RED através do protocolo MQTT. O Node-RED descodifica o protocolo de rádio, gera um objeto JavaScript de acordo com a especificação do *web service* no IIS, serializa o objeto JavaScript no formato JSON e faz um pedido HTTP para o *web service*, encapsulando o texto JSON no corpo pedido. A Figura 7-7 mostra parte da arquitetura do sistema em que o Node-RED se insere, onde se pode ver a função de encaminhamento de mensagem do Node-RED. As setas representam o fluxo de mensagens dos contadores para o *web service* no IIS.

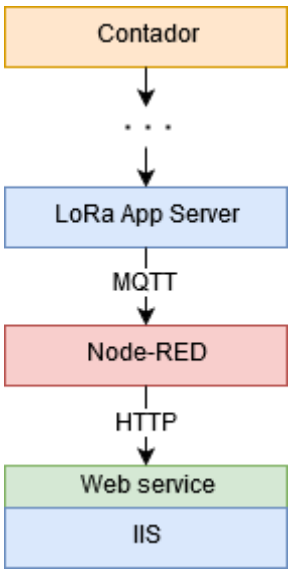



**Figura 7-7 - Encaminhamento de mensagens dos contadores pelo Node-RED**

O processo de encaminhar mensagens dos contadores para o *web service* no IIS é composto pelos seguintes passos:

1. Receber a mensagem do contador por MQTT e deserializá-la.
2. Obter dados do contador que enviou a mensagem.
3. Descodificar o cabeçalho da mensagem.
4. Descodificar a mensagem.

5. Gerar e enviar um pedido HTTP para o *web service* para o devido caminho.
6. Guardar a mensagem caso o pedido não seja recebido pelo *web service* ou caso a resposta indique que a operação falhou.

As mensagens dos contadores encaminhadas pelo LoRa App Server chegam ao sistema do Node-RED através do protocolo MQTT. A Figura 7-8 mostra o nó de entrada MQTT que foi configurado para ligar ao mediador de mensagens MQTT Eclipse Mosquitto, o LoRa App Server também está ligado a este mediador e sempre que recebe uma mensagem de um contador encaminhada pelo LoRa Network Server, publica-a no respetivo tópico MQTT de receção de mensagens desse contador. O Node-RED recebe mensagens de todos os contadores da aplicação LoRa especificada porque utiliza um *wildcard* em vez de especificar o EUI do contador no caminho do tópico (application/1/node/+/rx).


**Figura 7-8 - Nó de entrada MQTT subscrito ao tópico de mensagens dos contadores**

O protocolo MQTT transporta dados binários sem se importar com o formato destes, o que obriga a que o LoRa App Server serialize os dados a enviar por MQTT e que o Node-RED deserialize os dados a receber por MQTT. A Figura 7-9 mostra o nó que deserializa as mensagens MQTT do LoRa App Server.


**Figura 7-9 - Nó que deserializa as mensagens MQTT do LoRa App Server**

A Figura 7-10 mostra o código que deserializa os dados recebidos do LoRa App Server por MQTT O primeiro bloco de código deserializa os dados recebidos no formato JSON, o segundo bloco de código descodifica a mensagem do contador de Base64 para um *buffer* de *bytes*, o terceiro bloco de código gera uma nova mensagem com os parâmetros mais relevantes recebidos do LoRa App Server, dentre os quais a mensagem do contador e o EUI do contador.

```
var appServerMsg = null;
try {
  appServerMsg = JSON.parse(msg.payload);
} catch(error) {
  node.error(`Error parsing JSON over MQTT: ${error}`);
  return null;
}
if(appServerMsg === null) {
  node.error("JSON over MQTT was not parsed");
  return null;
}
```

```
var meterPayload = Buffer.from(appServerMsg.data, "base64");
```

```
var loRaMsg = {
  payload: meterPayload,
  eui: appServerMsg.devEUI,
  ...
};
return loRaMsg;
```

**Figura 7-10 - Código que deserializa as mensagens MQTT do LoRa App Server**

De modo a gerar os pedidos para o *web service* é necessário saber qual o GUID do contador que enviou a mensagem, pois o sistema identifica os contadores a partir do GUID, de forma a abstrair-se do *hardware* que gerou a mensagem, permitindo, por exemplo, substituir *hardware* com falhas alterando apenas o EUI associado ao contador. A Figura 7-11 mostra o nó que obtém dados dos contadores quando são recebidas mensagens destes.


**Figura 7-11 - Nó que obtém dados dos contadores**

A Figura 7-12 mostra o código que obtém os dados dos contadores a partir do EUI. O primeiro bloco de código procura o contador na lista de contadores associados presente no contexto global do sistema, comparando o EUI do contador que enviou a mensagem os EUIs dos contadores associados. O segundo bloco de código verifica se o contador está associado e se não estiver interrompe o fluxo de dados atual. O terceiro bloco de código guarda os dados dos contadores numa nova propriedade da mensagem, de modo a que os próximos nós do fluxo possam obter informações do contador através desta propriedade.

```
var meters = global.get("meters");
if(typeof meters === "undefined") {
  node.error(`No meter data. EUI="${msg.eui}"`);
  return null;
}
var meter = null;
for(var i in meters) {
  if(meters[i].eui == msg.eui) {
    meter = meters[i];
    break;
  }
}
```

```
if(meter === null) {
  node.error(`Meter is not associated. EUI="${msg.eui}"`);
  return null;
}
```

```
msg.meterData = meter;
return msg;
```

**Figura 7-12 - Código que obtém os dados dos contadores através do EUI**

As mensagens dos contadores possuem um cabeçalho que especifica o tipo de mensagem que se trata. Esta informação é necessária para que o sistema que vai descodificar a mensagem saiba como a deve descodificar. Adicionalmente todas as mensagens contêm o estado atual da válvula (aberta, fechada, etc.), de modo a que seja possível saber o último estado da válvula sem ter que enviar um comando e esperar pela resposta. A Figura 7-13 mostra o nó que descodifica o cabeçalho das mensagens dos contadores e um nó "*switch*" que encaminha os vários tipos de mensagens para a função descodificadora apropriada.

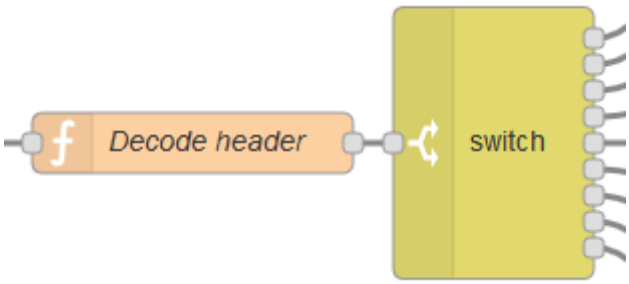



**Figura 7-13 - Nó que descodifica o cabeçalho das mensagens dos contadores**

A Figura 7-14 mostra o código que descodifica o cabeçalho das mensagens dos contadores. O primeiro bloco obtém e valida o tipo de mensagem. O segundo bloco obtém o estado da válvula a partir da lista de estados da válvula dos contadores, verifica se o estado se alterou, e caso tenha alterado altera o estado guardado para o novo estado juntamente com a data/hora atual. O terceiro bloco de código cria uma nova entrada na lista de estados da válvula dos contadores, se não existir uma entrada para o contador atual. O quarto bloco de código guarda as alterações à lista de estados da válvula dos contadores caso tenham sido feitas alterações.

```
var msgType = msg.payload[0] & 0x0F;
if(msgType < 0 || msgType > 5){
    node.error("Invalid message type");
    return null;
}
msg.msgType = msgType;
```

```
var valveStatus = (msg.payload[0] & 0x30) >> 4;
var meterValves = global.get("meterValves");
var meterFound = false;
var valveStatusChanged = false;
for(var i in meterValves) {
    if(meterValves[i].eui === msg.eui) {
        if(meterValves[i].status !== valveStatus) {
            meterValves[i].status = valveStatus;
            meterValves[i].lastUpdate = new Date();
            valveStatusChanged = true;
        }
        meterFound = true;
        break;
    }
}
```

```
if(meterFound === false) {
    meterValves.push({
        eui: msg.eui,
        status: valveStatus,
        lastUpdate: new Date()
    });
    valveStatusChanged = true;
}
```

```
if(valveStatusChanged === true) {
    global.set("meterValves", meterValves);
}
return msg;
```

**Figura 7-14 - Código que descodifica o cabeçalho das mensagens dos contadores**

Para exemplificar a descodificação de um tipo específico de mensagem, vai ser mostrado o processo de descodificação das mensagens de leitura dos contadores. As mensagens que reportam leituras dos contadores são descodificadas de acordo com o protocolo de rádio (Anexo A: Protocolo de rádio), o valor da leitura e a data e hora em a leitura foi feita são extraídas a partir do corpo da mensagem com recurso a lógica binária e operações matemáticas. A Figura 7-15 mostra o nó que descodifica as mensagens de leitura dos contadores.

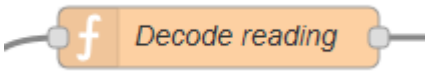



**Figura 7-15 - Nó que descodifica as mensagens de leitura dos contadores**

A Figura 7-16 mostra o código que descodifica as mensagens de leitura de contadores de água. O primeiro bloco de código descodifica a leitura do contador. O segundo bloco de código obtém os dez bits mais significativos e os vinte e quatro bits menos significativos da data/hora de aquisição da leitura, desloca os dez bits mais significativos vinte e quatro vezes para a esquerda e soma o resultado com os vinte e quatro bits mais significativos de modo a obter a data/hora de aquisição da leitura. Este bloco de código é um *workaround* relativamente à limitação da linguagem JavaScript

em que os operandos em lógica binária são sempre convertidos para números inteiros de 32 bits com sinal (salvo em algumas operações), multiplicar os dez bits mais significativos por $2^{24}$ (16777216) produz o mesmo resultado que deslocá-los vinte e quatro vezes para a esquerda. O terceiro bloco de código guarda a leitura e a data/hora de aquisição da leitura no *payload* na mensagem, abstraindo o próximo nó do protocolo de rádio.

```
if(msg.payload.length !== 9) {
  node.error("Invalid reading message size");
  return null;
}
```

```
var reading =
    (msg.payload[1] << 22) |
    (msg.payload[2] << 14) |
    (msg.payload[3] << 6)  |
    (msg.payload[4] >>> 2);
```

```
var unixTime10Msb =
    (msg.payload[4] & 0x03) << 8 |
    msg.payload[5];
var unixTime24Lsb =
    (msg.payload[6] << 16) |
    (msg.payload[7] << 8)  |
    msg.payload[8];
var leftShift24 = unixTime10Msb * 16777216;
var dataAcquiredAt = leftShift24 + unixTime24Lsb;
```

```
msg.payload = {
    sensorReadings: [reading],
    dataAcquiredAt: dataAcquiredAt
};
return msg;
```

**Figura 7-16 - Código que descodifica mensagens de leitura dos contadores de água**

De modo a obter os nomes dos sensores que um determinado contador possui, o modelo do contador é obtido a partir da lista de modelos de contador. A Figura 7-17 mostra o nó que obtém os dados do modelo dos contadores.

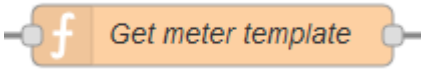



**Figura 7-17 - Nó que obtém os dados do modelo dos contadores**

A Figura 7-18 mostra o código que obtém os dados do modelo de contador. O primeiro bloco de código tenta obter a lista de modelos de contador a partir do contexto global do sistema. O segundo bloco de código procura o modelo do contador que enviou a mensagem. O terceiro bloco de código guarda os dados do modelo de contador numa nova propriedade da mensagem, de modo a que os próximos nós do fluxo possam obter informações do modelo do contador através desta propriedade.

```
var templates = global.get("meterTemplates");
if(typeof templates === "undefined") {
  node.error("Could not get global meter templates");
  return null;
}
```

```
var template = null;
for(var i in templates) {
    if(templates[i].id === msg.meterData.templateId) {
        template = templates[i];
        break;
    }
}
if(template === null) {
    node.error(`Could not get meter template. Id="${msg.meterData.templateId}"`);
    return null;
}
```

```
msg.meterTemplate = template;
return msg;
```

**Figura 7-18 - Código que obtém os dados do modelo de contador**

É gerado um pedido para o *web service* a partir dos dados descodificados da mensagem, dos dados do contador e dos dados do modelo do contador. A Figura 7-19 mostra o nó que gera os pedidos de leitura dos contadores para o *web service* no IIS.

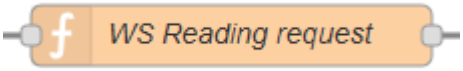



**Figura 7-19 - Nó que gera os pedidos de leitura para o web service no IIS**

A Figura 7-20 mostra o código que gera os pedidos de leitura dos contadores para *o web service* no IIS. O primeiro bloco de código tenta obter a configuração do sistema. O segundo bloco de código serializa a data para o *standard* ISO-8601. O terceiro bloco de código gera o objeto a enviar, contendo o GUID do contador, uma lista dos sensores, uma lista de datas e uma lista de leituras (mais detalhes no Anexo D: Especificação do *web service* no IIS). O *web service* está preparado para receber múltiplas leituras de um contador num pedido de inserção de leituras, de modo a tornar a comunicação mais eficiente. Como é esperado que os contadores de água só efetuem leituras uma vez por dia, este sistema está configurado para enviar imediatamente a leitura assim que a recebe. O quarto bloco de código gera o URL do pedido a partir da configuração do sistema, define o método do pedido, define os cabeçalhos do pedido a partir da configuração do sistema, coloca o objeto gerado no *payload* da mensagem de modo a ser enviado no pedido e cria uma nova propriedade na mensagem em que guarda uma cópia do mesmo objeto, para caso o pedido não seja bem-sucedido a leitura possa ser armazenada para ser enviada mais tarde (o *payload* da mensagem definido nesta função vai ser substituído pela resposta do *web service*).

```
var config = global.get("configuration");
if(typeof config === "undefined") {
    node.error("Could not get global configuration");
    return null;
}
```

```
var dataAcquiredAt =
    new Date(msg.payload.dataAcquiredAt * 1000).toISOString();
```

```
var payload = {
    meterId: msg.meterData.guid,
    sensors: msg.meterTemplate.sensors,
    dataAcquiredAt: [dataAcquiredAt],
    readings: [msg.payload.sensorReadings]
};
```

```
msg.url =
`${config.wsProtocol}://${config.wsHost}:${config.wsPort}/API/Readings/InsertIndexe
d`;
msg.method = "POST";
msg.headers = {
    "Host": config.wsHost,
    "Content-Type": "application/json",
    "Authorization": config.wsAuthHeader
};
msg.payload = payload;
msg.reading = payload;
return msg;
```

**Figura 7-20 - Código que gera o pedido para o web service**

O pedido gerado é enviado para o *web service* no IIS com recurso ao nó de pedido HTTP. Este nó recebe os parâmetros do pedido, tais como o URL ou o método, através da mensagem que recebe na entrada, gera um pedido com os parâmetros especificados, envia o pedido, e devolve a resposta numa nova mensagem na saída. A Figura 7-21 mostra o nó que faz o pedido para o *web service* e devolve e resposta.


**Figura 7-21 - Nó que faz o pedido para o web service e devolve a resposta**

Se ocorrer algum erro na transmissão do pedido/resposta ou se a resposta do *web service* possuir um código que não indique que a operação foi bem-sucedida a leitura é guardada para ser enviada mais tarde. Existe uma fila de envio por cada tipo de mensagem, neste caso as mensagens de leituras dos contadores seriam armazenadas numa fila específica para leituras dos contadores. Mais tarde o sistema desenvolvido em Node-RED tenta enviar as mensagens pendentes nas filas de envio. Se uma fila de envio for recebida com sucesso pelo *web service* no IIS essa fila é limpa no Node-RED, de modo a não enviar as mesmas mensagens mais do que uma vez. Não vai ser apresentado o código que realiza esta operação por ser extenso, como alternativa será mostrado um resumo na forma de fluxograma. A Figura 7-22 mostra o fluxograma do algoritmo que guarda mensagens dos contadores caso não seja possível envia-las para o *web service*. A Figura 7-23 mostra o algoritmo que tenta enviar todas as mensagens pendentes armazenadas nas filas de envio.

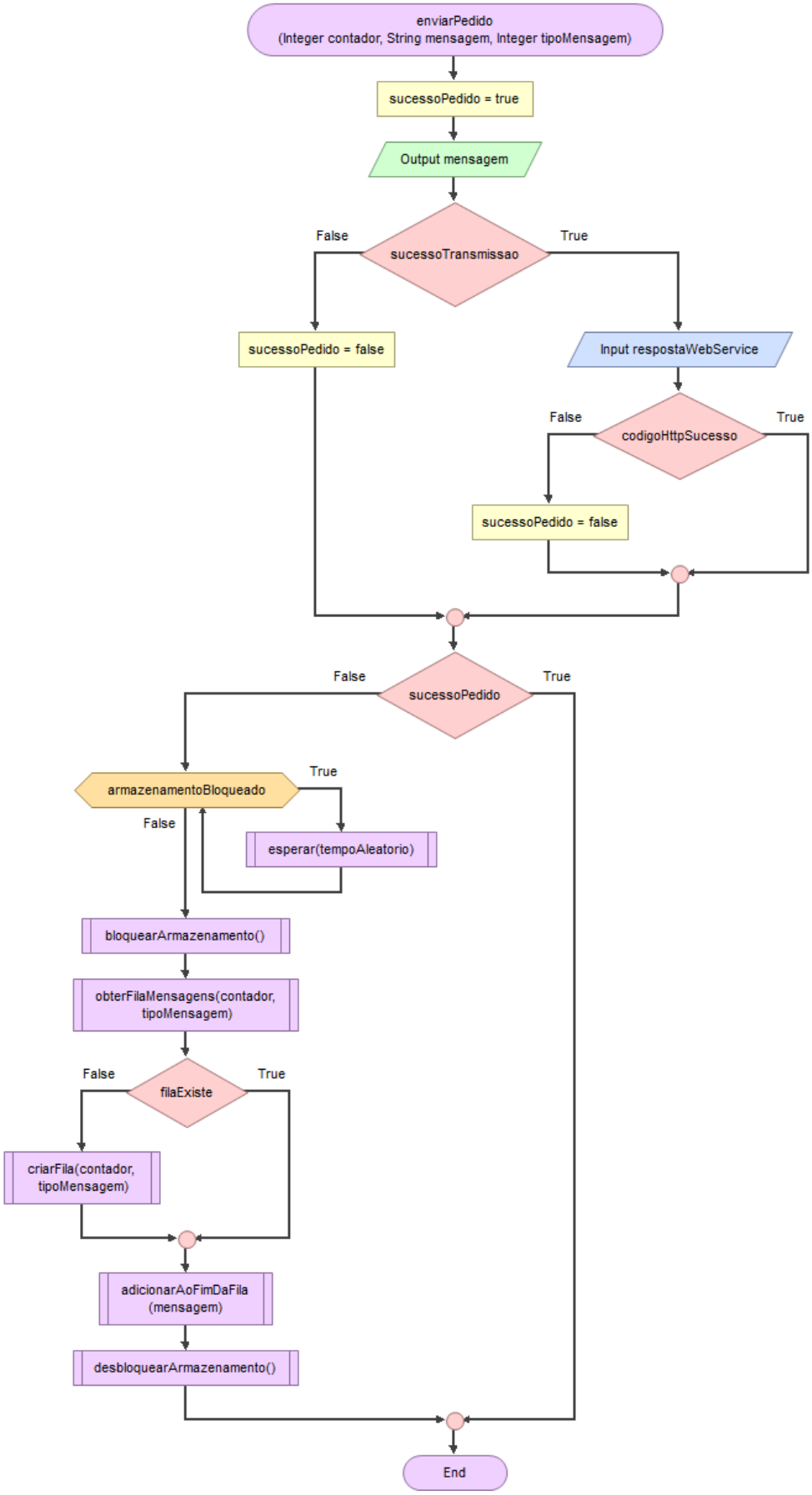



**Figura 7-22 - Algoritmo que guarda mensagens caso falhe o envio**

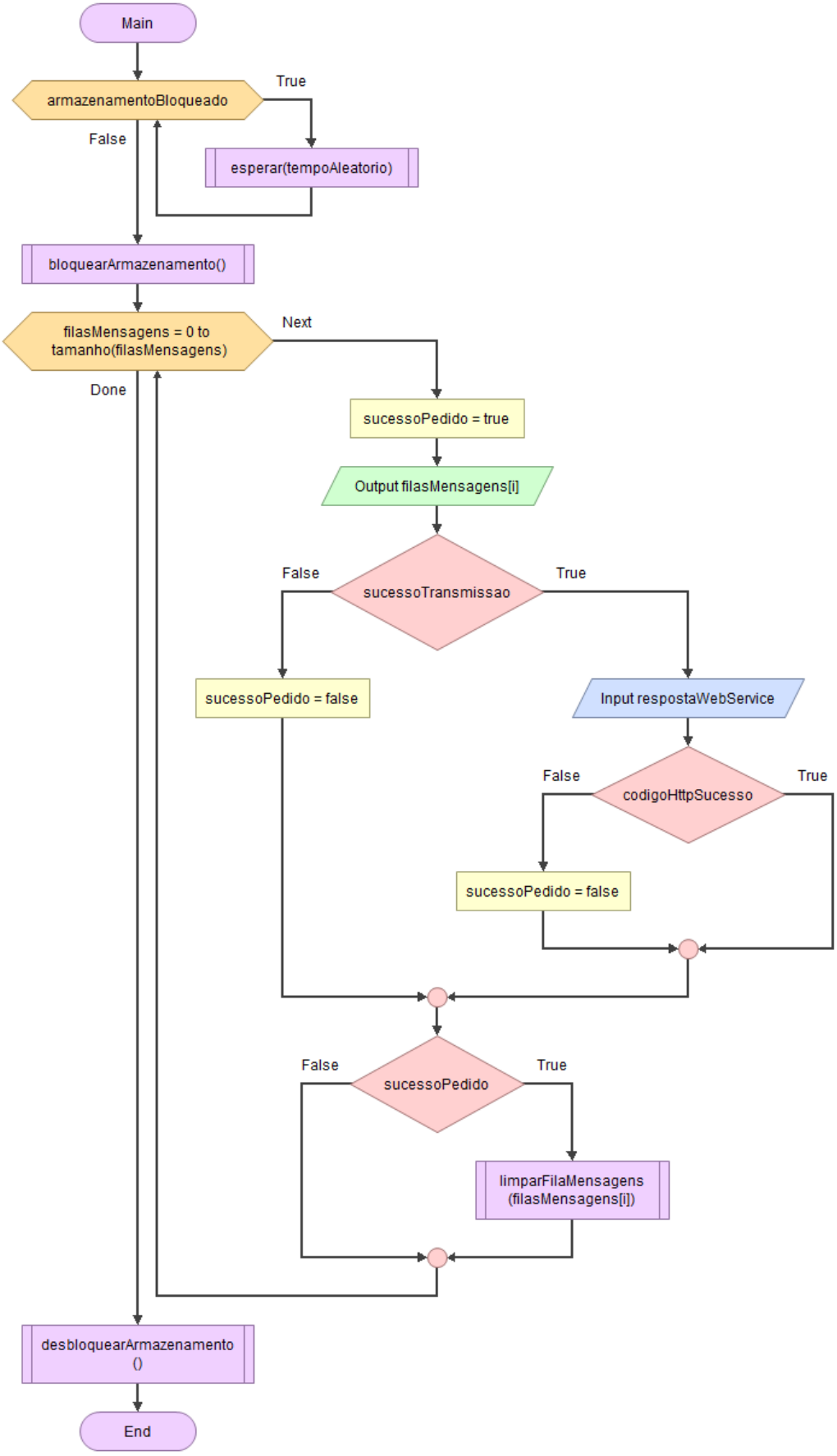



**Figura 7-23 - Algoritmo que envia as mensagens pendentes nas filas de envio**

### 7.7.5. Implementação de Web service em Node-RED

De modo a que seja possível receber novas configurações para o sistema do Node-RED e encaminhar comandos para os contadores enviados a partir do *website*, foi desenvolvido um *web service* no Node-RED. A Figura 7-24 mostra a função principal do *web service* no Node-RED, que é encaminhar mensagens para os contadores, no diagrama de componentes de *software* onde o Node-RED se insere, as setas representam o fluxo de mensagens do *website* no IIS para os contadores.

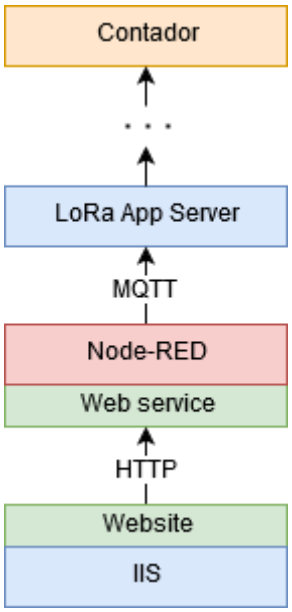



**Figura 7-24 – Função principal do *web service* no Node-RED**

O processo de encaminhar comandos e configurações do *website* para os contadores é composto pelos seguintes passos:

1. Receber o pedido HTTP do *website*.
2. Autenticar o pedido HTTP.
3. Obter dados do contador a controlar ou configurar.
4. Gerar a mensagem a enviar para o contador.
5. Enviar a mensagem do contador por MQTT e enviar resposta ao pedido HTTP.

Foram utilizados os nós que implementam um *web service* e que já vêm incluídos na coleção de nós *standard* do Node-RED, estes nós são implementados com a *framework web* "Express.js". Também foram criadas funções personalizadas para implementar o *web service*, estas funções podem dar resposta aos pedidos HTTP com um código numérico e respetiva descrição do que aconteceu, quer o pedido seja executado com sucesso, quer ocorra um erro. Exemplos de respostas do *web service*: "code: 200, message: Close valve command sent", "code: 404, message: Meter valve status is unknown".

Foi adicionado um nó de entrada HTTP para cada caminho (p. ex.: /api/meter/valve-control) e respetivo método HTTP (p. ex.: POST). Estes nós deserializam o pedido HTTP e facilitam o acesso aos seus componentes, tais como os cabeçalhos, cookies, parâmetros e corpo. A Figura 7-25 mostra alguns dos nós de entrada do *web* service.

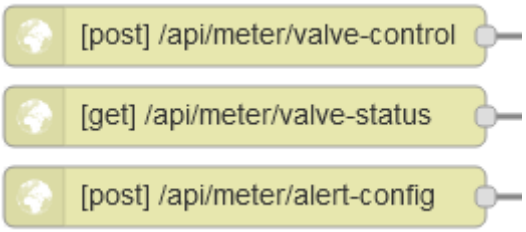



**Figura 7-25 - Nós HTTP de entrada**

Todos os pedidos são autenticados com recurso a uma função personalizada, pois os nós de entrada HTTP não suportam qualquer tipo de autenticação. Se a autenticação for bem-sucedida o pedido é aceite, se a autenticação falhar o *web service* devolve uma resposta que indica que a autenticação falhou. A Figura 7-26 mostra o nó que autentica os pedidos e o nó de saída HTTP chamado caso a autenticação falhe.

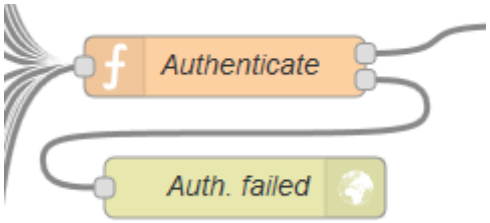



**Figura 7-26 - Nós de autenticação dos pedidos**

Foi utilizado o esquema de autenticação "Basic"[13] do protocolo HTTP, na implementação atual o "user-id" é o GUID do *gateway* e a "password" é a palavra-passe do *web service* no *gateway* (diferente da palavra-passe utilizada para efetuar pedidos para o *web service* no IIS). A Figura 7-27 mostra o código que autentica os pedidos recebidos pelo *web service* no Node-RED. No primeiro bloco de código tenta-se obter o cabeçalho HTTP de autenticação. No segundo bloco de código tenta-se obter a configuração do sistema. Caso não seja possível o erro é registado no *log* e é enviada uma mensagem para o cliente HTTP a informar que ocorreu um erro interno. No terceiro bloco de código verifica-se se o valor do cabeçalho de autenticação do pedido é igual ao valor do cabeçalho de autenticação da configuração global, ou seja verifica-se se o pedido está autorizado a aceder ao *web service* com as credenciais fornecidas.

```
if(typeof msg.req.headers["authorization"] === "undefined") {
  node.error(`Missing authorization header. URL=${msg.req.url}`);
  msg.payload = { code: 401, message: "Missing authorization header" };
  return [null, msg];
}
var authHeader = msg.req.headers["authorization"];
```

```
var config = global.get("configuration");
if(typeof config === "undefined") {
  node.error("Could not get global configuration");
  msg.payload = { code: 500, message: "Internal server error" };
  return [null, msg];
}
```

```
if(authHeader !== config.gwAuthHeader) {
  node.error(`Authentication failed. URL=${msg.req.url}`);
  msg.payload = { code: 401, message: "Authentication failed" };
  return [null, msg];
}
return [msg, null];
```

**Figura 7-27 - Código que autentica os pedidos para o *web service* no Node-RED**

O valor do cabeçalho de autenticação esperado para autenticar o pedido (ver seção 10.6.2) é gerado no arranque do sistema ou quando a configuração é alterada, sendo armazenado na configuração global de modo a acelerar o processo de autenticação. A Figura 7-28 mostra o código que gera o valor do cabeçalho de autenticação e guarda-o na configuração global.

```
...
var gwAuthString = `${ config.gwId }:${ config.gwPassword }`;
var gwBase64AuthString = Buffer.from(gwAuthString, "utf8").toString("base64");
config.gwAuthHeader = `Basic ${gwBase64AuthString}`;
global.set("configuration", config);
...
```

**Figura 7-28 - Código que gera o valor do cabeçalho de autenticação**

Os pedidos que controlam ou configuram contadores requerem informações sobre o contador. Este nó verifica se o contador está associado ao sistema e obtém os dados deste, a partir da lista de contadores associados. A Figura 7-29 mostra o nó que obtém dados dos contadores associados e o nó de saída HTTP chamado caso o contador não esteja associado.

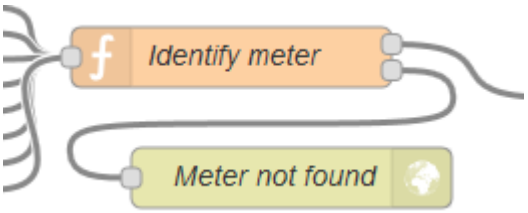



**Figura 7-29 - Nós que obtém dados dos contadores**

Uma instância do sistema do Node-RED só tem informações sobre os contadores que lhe foram associados, não é possível enviar mensagens para contadores que não estão associados com a instância. A Figura 7-30 mostra o código que verifica se o contador está associado ao sistema e obtém os respetivos dados. O primeiro bloco de código tenta

obter o GUID do contador a partir do URL ou corpo do pedido. O segundo bloco de código tenta obter a lista de contadores associados e os dados contador a controlar/configurar. O terceiro bloco de código verifica se os dados do contador incluem o respetivo EUI. O quarto bloco de código guarda os dados do contador numa nova propriedade (meterData) da mensagem, de modo a serem utilizados posteriormente pelos próximos nós do *web service*.

```
var meterId = null;
if(typeof msg.req.query.meterId !== "undefined") {
  meterId = msg.req.query.meterId;
} else if(typeof msg.req.body.meterId !== "undefined"){
  meterId = msg.req.body.meterId;
}
if(meterId === null) {
  node.error(`Missing meter id. URL="${msg.req.url}"`);
  msg.payload = { code: 400, message: "Missing meter id" };
  return [null, msg];
}
```

1

```
var meters = global.get("meters");
if(typeof meters === "undefined") {
  ...
  return [null, msg];
}
var meter = null;
for(var i in meters) {
  if(meters[i].guid == meterId) {
    meter = meters[i];
    break;
  }
}
if(meter === null) {
  ...
  return [null, msg];
}
```

2

```
if(meter.eui === null) {
    ...
    return [null, msg];
}
```

3

```
msg.meterData = meter;
return [msg, null];
```

4

**Figura 7-30 - Código que obtém os dados dos contadores a partir do GUID**

Cada nó de entrada HTTP possui um nó que processa o pedido e transmite o resultado para um nó de saída A Figura 7-31 mostra o nó que processa pedidos de configuração dos alertas dos contadores. Este nó, tal como todos os nós que processam pedidos relacionados com contadores, possuem duas saídas: uma para o nó HTTP de saída que responde ao pedido e outra que envia o comando em binário para o contador.

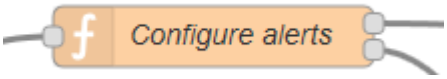



**Figura 7-31 - Nó que configura alertas dos contadores**

A Figura 7-32 mostra o código do nó que gera a mensagem de configuração dos alertas dos contadores, especificada no protocolo de rádio (Anexo A: Protocolo de rádio). O

primeiro bloco de código gera a mensagem a enviar para o contador que é composta por dois bytes de acordo com o protocolo de rádio, junta o EUI do contador para onde a mensagem deve ser enviada e escreve o cabeçalho da mensagem no primeiro byte (offset 0). O segundo bloco de código converte a configuração dos alertas num mapa de bits, em que cada bit define se um determinado alerta deve ser ativado (1) ou desativado (0), e de seguida escreve o mapa de bits de alertas no segundo byte da mensagem (offset 1). O terceiro bloco de código gera a resposta para o pedido HTTP.

```
var loRa = {
    payload: Buffer.alloc(2),
    eui: msg.meterData.eui
};
loRa.payload.writeUInt8(0x06, 0);
```

```
var alerts = msg.payload;
var alertsBitmap = 0x00;
if(alerts.reverseFlux === true)     alertsBitmap |= 0x80;
if(alerts.magneticFraud === true)   alertsBitmap |= 0x40;
if(alerts.highConsumption === true) alertsBitmap |= 0x20;
if(alerts.zeroConsumption === true) alertsBitmap |= 0x10;
if(alerts.lowBattery === true)      alertsBitmap |= 0x08;
if(alerts.memoryError === true)     alertsBitmap |= 0x04;
if(alerts.wrongDateTime === true)   alertsBitmap |= 0x02;
loRa.payload.writeUInt8(alertsBitmap, 1);
```

```
msg.statusCode = 200;
msg.payload = { message: "Alert configuration sent" };
return [msg, loRa];
```

**Figura 7-32 - Código que gera a mensagem de configuração de alertas**

Por fim é enviada uma resposta para o cliente HTTP e a mensagem para o contador é enviada para o LoRa App Server através do protocolo MQTT. A Figura 7-33 mostra o nó de saída HTTP do web service e o nó de saída MQTT que liga a um mediador MQTT.

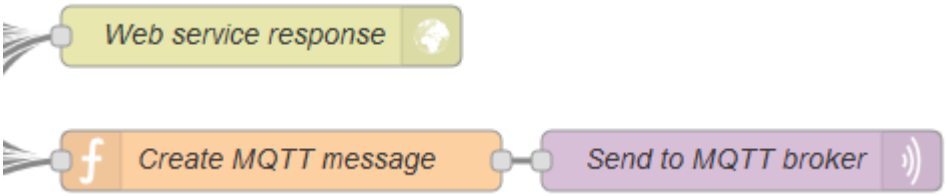



**Figura 7-33 - Nó de saída HTTP do web service e nó de saída MQTT**

A Figura 7-34 mostra o código que gera a mensagem MQTT para o LoRa App Server que é enviada através do mediador MQTT "Eclipse Mosquitto". O primeiro bloco de código tenta obter a configuração do sistema, caso não seja possível o erro é registado no *log*. O segundo bloco de código gera a mensagem a enviar para o LoRa App Server, é gerado texto aleatório que serve para identificar a mensagem, é definido que a receção da mensagem deve ser confirmada (*acknowledge*), o porto (semelhante a um porto TCP ou UDP) é configurado para ser igual ao identificador da aplicação LoRa desde que esteja dentro do intervalo válido e a mensagem do protocolo de rádio é codificada em Base64. O terceiro bloco de código gera a mensagem a enviar para o mediador MQTT, é definido o tópico da mensagem com o identificador da aplicação LoRa e o EUI do contador, é definido o nível dois de qualidade do serviço na entrega da mensagem (o

nível dois garante a entrega da mensagem apenas uma vez) e é definido que a última mensagem deste tópico não deve ser guardada no mediador MQTT. Ainda no terceiro bloco de código a mensagem para o LoRa App Server é serializada no formato JSON e encapsulada na mensagem MQTT (propriedade "payload" da variável "msgToMqttBroker"), pois o protocolo MQTT apenas transporta dados binários e a estrutura desses dados é responsabilidade da aplicação que os envia.

```
var config = global.get("configuration");
if(typeof config === "undefined") {
    node.error("Could not get global configuration");
    return null;
}
```

```
var msgToAppServer = {
    reference: Math.random().toString(36).substr(2, 8),
    confirmed: true,
    fPort: (config.appId > 0 && config.appId < 224 ? config.appId : 1),
    data: msg.payload.toString("base64")
};
```

```
var msgToMqttBroker = {
    topic: `application/${config.appId}/node/${msg.eui}/tx`,
    qos: 2,
    retain: false,
    payload: JSON.stringify(msgToAppServer)
};
return msgToMqttBroker;
```

**Figura 7-34 - Código que gera a mensagem MQTT para o LoRa App Server**

Nota: alguns nós que encaminham as mensagens do *web service* foram omitidos de modo a simplificar a apresentação.

# 7.8. IIS - Website

De modo a permitir configurar remotamente os contadores e os *gateways*, consultar leituras e notificações, configurar e resolver problemas com o sistema, entre outras funcionalidades, foi desenvolvido um *website* que permite administrar o sistema de forma centralizada. Foi tomada a decisão de desenvolver um *website* em vez de aplicações nativas, pois um *website* pode ser acedido a partir de qualquer máquina que possua um *web browser*, no futuro se for necessário administrar o sistema a partir de outros dispositivos (p. ex.: *tablet*), estes dispositivos podem ser suportados com menos esforço do que se fosse necessário desenvolver uma aplicação especifica para o dispositivo. Adicionalmente para lançar novas versões do *software* basta atualizar o servidor e todos os clientes passam a usar a ultima versão, não sendo necessário atualizar as aplicações instaladas em todas as máquinas de administração. O servidor *web* em que o *website* é alojado é o IIS, pois era um requisito. O IIS só é suportado no sistema operativo Microsoft Windows, o que por sua vez torna a utilização do Microsoft Windows um requisito.

## 7.8.1. Compatibilidade com browsers

Como a decisão de desenvolver um *website* foi baseada na possibilidade de maximizar o número de máquinas a partir das quais é possível administrar o sistema, tornou-se então

importante assegurar que as funcionalidades principais do *website* executam sem problemas no maior número de *browsers* possível.

As funções principais do *website* são implementadas com elementos HTML básicos, tais como hiperligações (elemento <a>) ou formulários (elemento <form>) e não necessitam de JavaScript, portanto podem ser utilizadas em praticamente todos os browsers, incluindo browsers antigos ou sem suporte. Se o *browser* utilizado para aceder ao *website* suportar JavaScript então o código JavaScript é utilizado em ocasiões onde faz sentido, permitindo reduzir o tempo de carregamento de páginas e aceder a funcionalidades adicionais, tais como visualizar leituras em gráficos interativos.

A Figura 7-35 mostra a página de configuração de contadores no *web browser* Internet Explorer 11, no sistema operativo Windows 7.

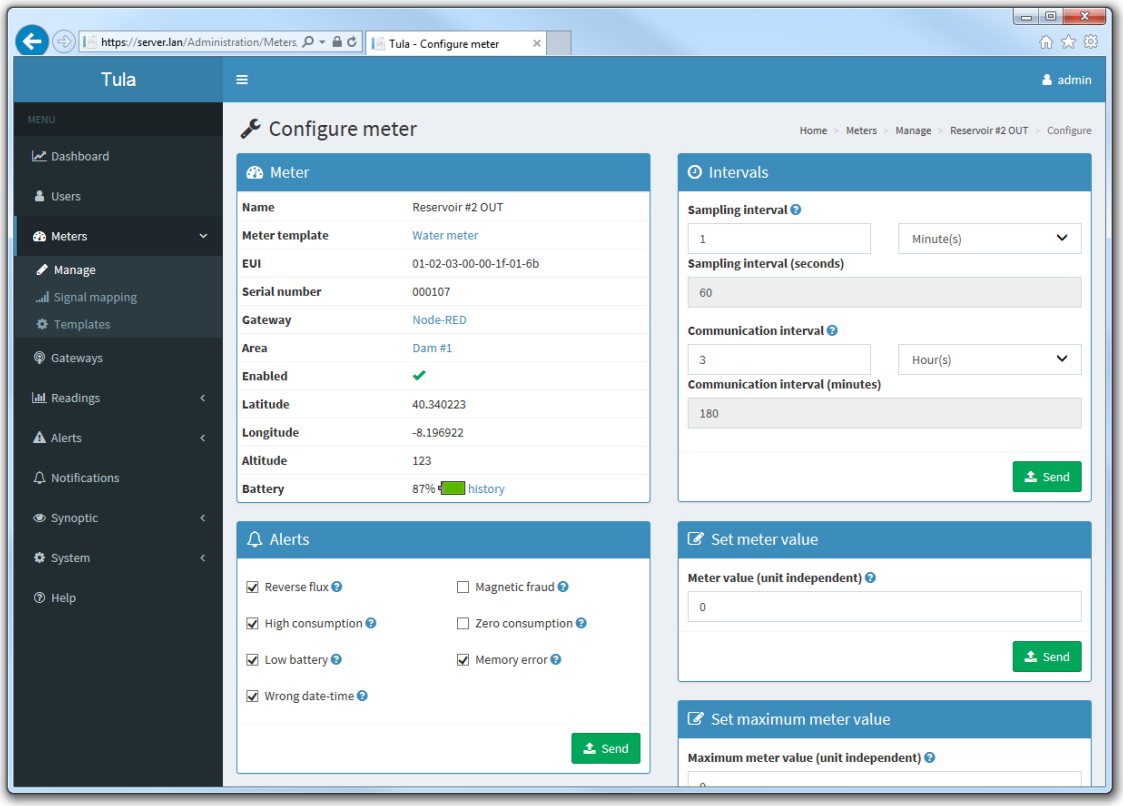



**Figura 7-35 - Configuração de contadores no Internet Explorer**

A Figura 7-36 mostra a página de visualização de leituras de contadores no *web browser* Internet Explorer Mobile 11.0, no sistema operativo Windows Phone 8.1.

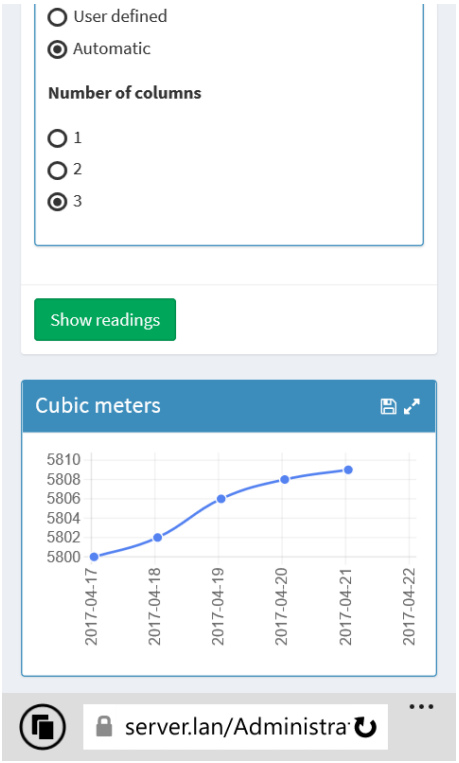



**Figura 7-36 - Visualização de leituras no Internet Explorer Mobile**

A Figura 7-37 mostra a página de gestão de contadores no browser Links 2.8, no sistema operativo Linux 4.9, através de SSH.

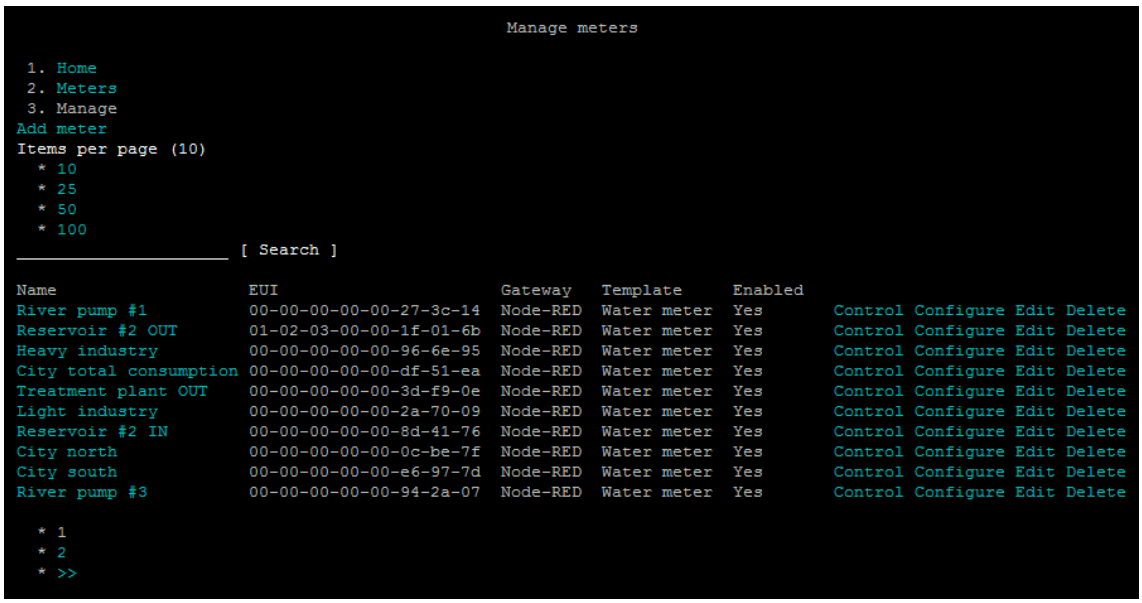



**Figura 7-37 - Gestão de contadores no Links**

A maior parte do código foi desenvolvida em JavaScript (ECMAScript 2016), mas em certos casos em que a forma de realizar uma operação depende da implementação JavaScript ou API JavaScript incluída no *web browser*, foi utilizada a biblioteca "jQuery" para suportar o maior número de *browsers* possível. Seria possível desenvolver funções distintas para cada *web browser* (e as suas versões), mas devido à falta de tempo a utilização da biblioteca jQuery foi mais viável.

### 7.8.2. ASP.NET MVC

De acordo com o requisito de desenvolver o *website* na linguagem de programação C# com a *framework* ".NET Framework", foi escolhida a *framework* ASP.NET para desenvolver o site. A *framework* ASP.NET é uma extensão de .NET Framework com o objetivo de ajudar a desenvolver *websites* ou aplicações para a *web*. Entre os vários padrões para desenvolver um *website* em ASP.NET, escolhi o padrão de desenvolvimento de software *model-view-controller* através da *framework* ASP.NET MVC que tem como base ASP.NET. Foi escolhido o padrão MVC de modo a facilitar as alterações sucessivas ao código, separar as responsabilidades e reutilizar facilmente partes do código.

Existem outras alternativas que satisfazem o requisito de utilizar a linguagem C# e a *framework* .NET para desenvolvimento do *website*, tais como a *framework* ASP.NET Web Forms e ASP.NET Web Pages. A *framework* ASP.NET Web Forms facilita e acelera o desenvolvimento de páginas *web*, pois fornece um conjunto de componentes que podem ser inseridos numa página web com "drag-and-drop" e segue um padrão de programação orientado a eventos, em que a *framework* deteta eventos que acontecem no *browser* e executa funções do lado do servidor que tratam desses eventos de forma automática, no entanto esta *framework* não faz a separação de responsabilidades, dificultando a reutilização de código e os testes unitários. A *framework* ASP.NET Web Pages é mais simples pois não obriga a seguir um padrão ou estrutura de código, é semelhante à *framework* ASP.NET MVC mas não segue o padrão MVC, no entanto esta flexibilidade requer cuidados adicionais ao desenvolver grandes projetos, pois a estruturação do código passa a ser responsabilidade do programador.

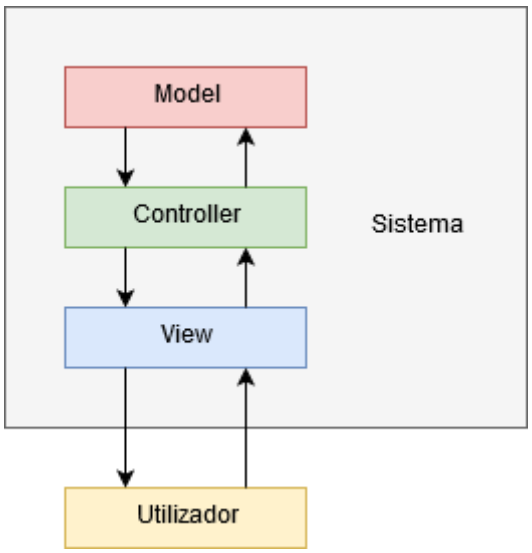



**Figura 7-38 - Padrão de desenvolvimento de *software* MVC**

Na *framework* ASP.NET MVC os *models* são implementados com a linguagem de programação C# com classes POCO (Plain Old Clr Object), o propósito destas classes é fornecer uma estrutura de dados que abstrai a forma como os dados são armazenados, facilitando a passagem de dados para os *controllers* e os testes unitários. *Os controllers* são implementados com a linguagem de programação *C#* com classes que derivam da classe "Controller". A passagem de dados entre os *controllers* e as *views* é feita através de classes POCO C# chamadas *view models*, de modo a serem distinguidas das classes que implementam os *models*. As *views* são codificadas com a sintaxe de programação

ASP.NET Razor, esta sintaxe permite escrever código HTML com código C# embebido, o código HTML serve como modelo da página e o código C# gera código HTML de forma dinâmica, em que o seu conteúdo depende dos dados passados pelo *controller*. O utilizador interage com a *view* através do *web browser*, enviando pedidos e recebendo respostas através deste.

### 7.8.3. Mapa do *website*

O *website* está organizado de forma hierárquica de modo a simplificar a navegação, adicionalmente o website possui *breadcrumbs* que orientam o utilizador, mostrando o caminho desde a página inicial até à página em que encontra. A Figura 7-39 mostra o mapa simplificado das páginas do *website*.

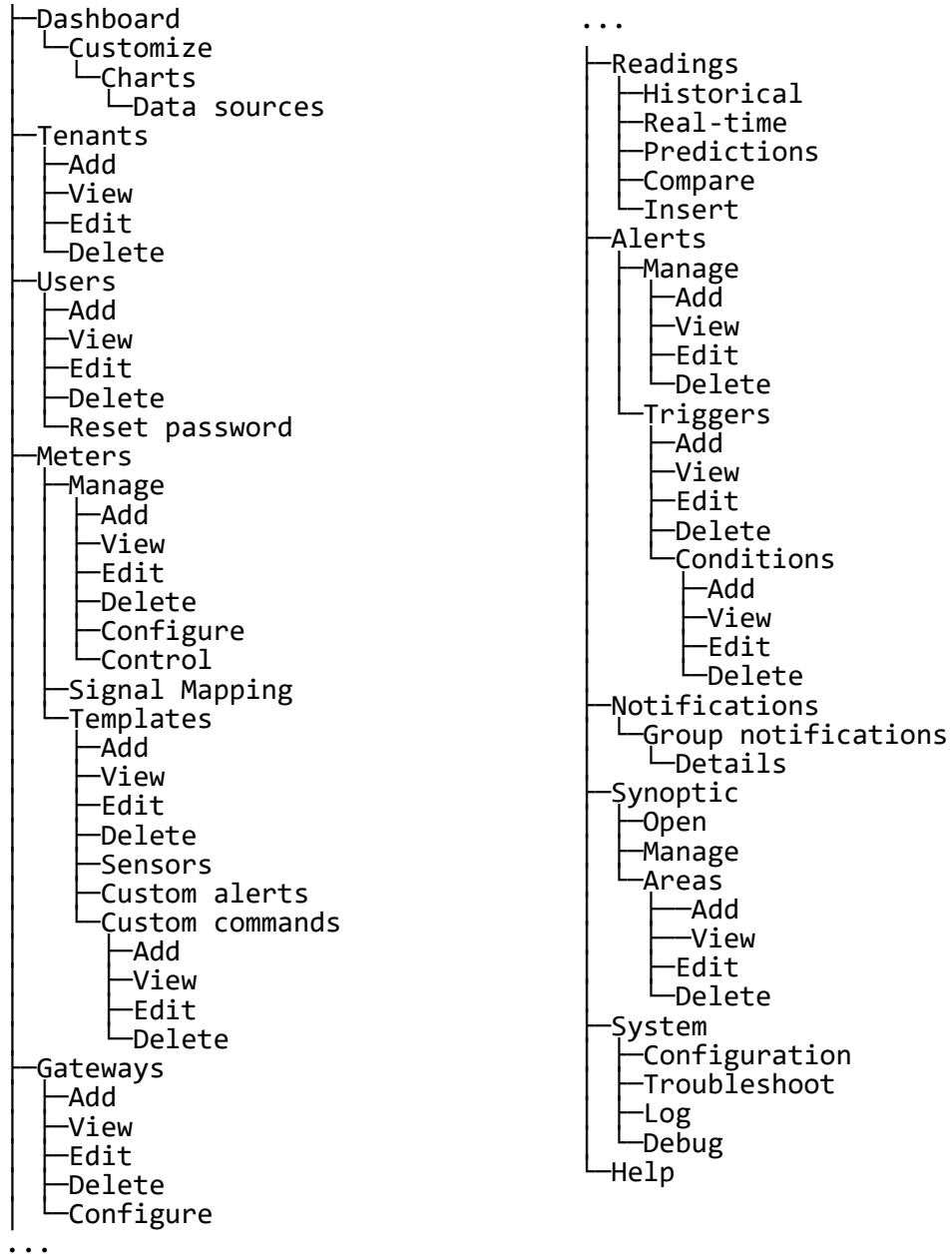



**Figura 7-39 - Mapa do *website***

O mapa não mostra todos os caminhos possíveis entre as páginas, por exemplo, é possível aceder à página de detalhes do modelo de contador (Meters > Templates > View) a partir da página de detalhes do contador (Meters > Manage > View), mas não existe ligação entre estas páginas no mapa. Para representar todos os caminhos possíveis seria necessário utilizar um grafo em vez de uma árvore, mas tal representação seria confusa e não adicionaria informação relevante.

### 7.8.4. Programação

Os *models* são implementados na forma de classes que podem possuir propriedades simples, propriedades que representam relações e anotações. Através da Entity Framework, as classes são representadas na base de dados na forma de tabelas, as propriedades simples são representadas na forma de colunas da tabela e as propriedades de navegação representam relações entre tabelas, tais como "muitos para um" e "um para muitos".

A Figura 7-40 mostra um exemplo de *model*, neste caso o *model* dos contadores. O primeiro bloco de código marcado mostra a chave primária, que possui a anotação "Key". O segundo bloco de código marcado mostra algumas propriedades simples, todas estas propriedades contêm a anotação "DisplayName" que define um nome "amigável" que pode ser utilizado como nome da propriedade numa *view*. A anotação "Required" define que propriedade tem que ter um valor, caso contrário a validação do *model* falha e que a respetiva coluna da na base de dados não pode ser nula, a anotação "MaxLength" define o tamanho máximo da propriedade e respetiva coluna da tabela na base de dados. O terceiro bloco de código marcado mostra um exemplo de propriedades que representam uma relação "muitos para um", neste caso podem existir muitos contadores de um determinado modelo. A propriedade "MeterTemplateId" guarda a chave forasteira e a propriedade "MeterTemplate" pode ser usada para referenciar e aceder ao modelo de contador associado. O quarto bloco de código marcado mostra um exemplo de relação "um para muitos", neste caso um contador pode ter muitas sessões de mapeamento de sinal, a propriedade "SignalMappingSessions" pode ser usada para referenciar e aceder às sessões de mapeamento de sinal associadas.

```
public class Meter
{
    [Key]
    public Guid Id { get; set; }

    [Required]
    [DisplayName("Name")]
    [MaxLength(50, ErrorMessage = "Maximum 50 characters")]
    public string Name { get; set; }
    [DisplayName("Enabled")]
    public bool Enabled { get; set; }
    [MaxLength(23, ErrorMessage = "Maximum 23 characters")]
    [DisplayName("EUI")]
    public string Eui { get; set; }
    [DisplayName("Latitude")]
    public decimal? Latitude { get; set; }
    [DisplayName("Longitude")]
    public decimal? Longitude { get; set; }
    [DisplayName("Altitude")]
    public short? Altitude { get; set; }

    [DisplayName("Meter template")]
    public int MeterTemplateId { get; set; }
    public virtual MeterTemplate MeterTemplate { get; set; }

    public virtual ICollection<SignalMappingSession>
      SignalMappingSessions { get; set; }
    ...
}
```

**Figura 7-40 - Exemplo de *model***

As *views* são codificadas com sintaxe de programação ASP.NET Razor. Primeiro é escrito o código HTML de modo a criar um protótipo de alta-fidelidade e depois os elementos ou valores estáticos são substituídos por código C# que é embebido no código HTML de modo a tornar a página dinâmica. As *views* contêm código C# relativamente simples, de modo a seguir o princípio de separação de responsabilidades e facilitar futuras alterações ao código HTML. Como exemplo as *views* não contêm código que faz consultas na base de dados ou que altera de alguma forma os dados do *model*, embora tais operações sejam possíveis. Todos os dados que as *views* necessitam são obtidos e passados pelas *actions* dos *controllers* e, de forma semelhante, todos os dados necessários para as *actions* dos *controllers* são obtidos e passados pelas *views*. A Figura 7-41 mostra um exemplo de *view* no *web browser*, esta *view* permite adicionar novos contadores ao sistema, os menus não são mostrados pois não fazem parte desta *view* e deste modo é possível mostrar a imagem da *view* com mais detalhe.

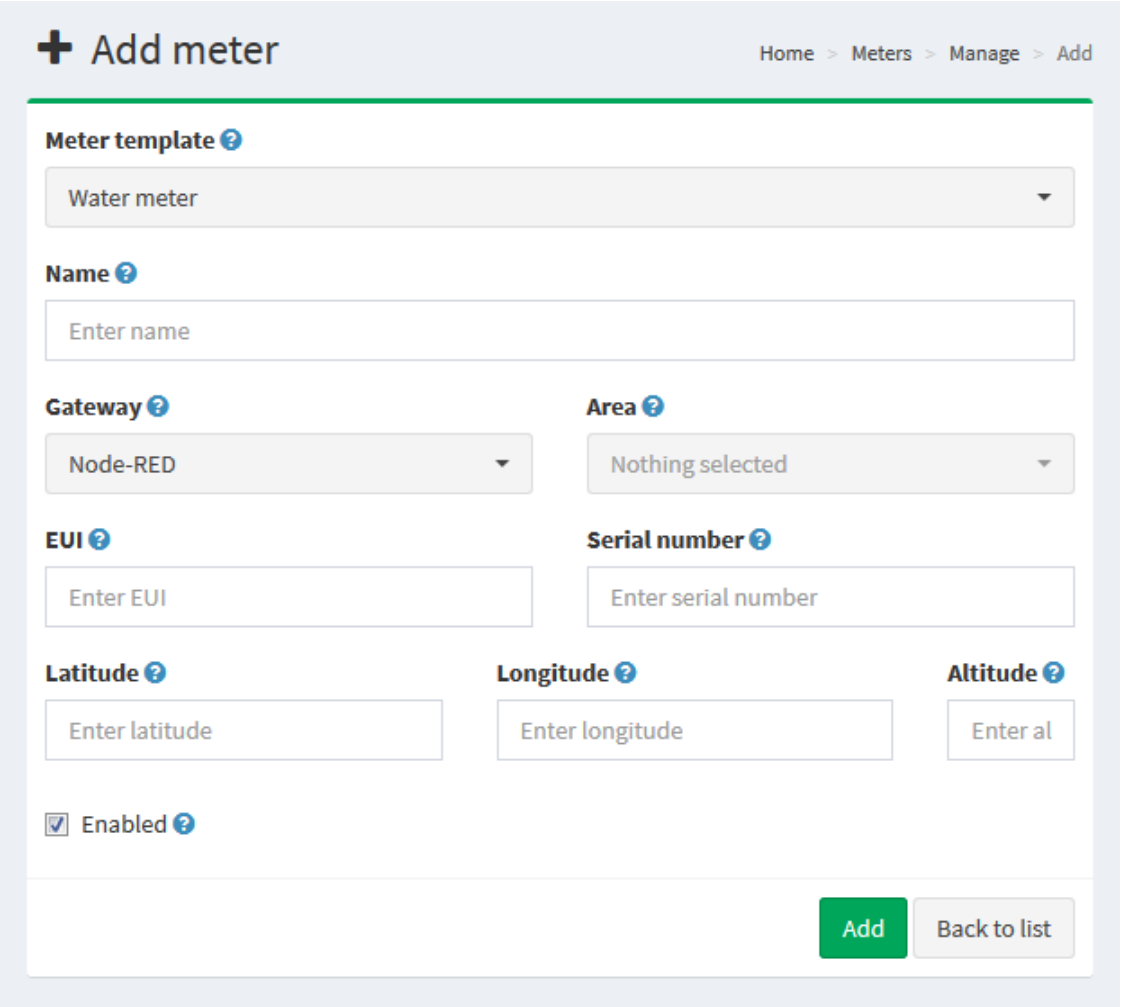



**Figura 7-41 - *View* que permite adicionar novos contadores**

A Figura 7-42 mostra o código fonte da *view* que permite adicionar novos contadores ao sistema. O primeiro bloco de código marcado define qual o *model* (ou *view model*) que é utilizado para enviar e receber dados para a respetiva *action* do *controller*. O segundo bloco de código define algumas configurações que são utilizadas pela página de *layout* (página modelo que serve de base para outras páginas), neste caso o ícone da página, o título e a descrição, entre outras configurações. O terceiro bloco de código mostra uma mensagem de erro caso ocorra algum erro. O quarto bloco de código insere um valor no formulário utilizado para evitar que *sites* externos façam pedidos para a *action* desta *view* (ataques CSRF). O quinto bloco de código gera uma lista a partir da qual o utilizador pode escolher o modelo de contador e mostra uma mensagem de erro caso o modelo escolhido seja inválido (por exemplo se não existir nenhum modelo de contador na base de dados). O sexto bloco de código mostra um exemplo de reutilização de código, o formulário que permite adicionar contadores é semelhante ao formulário que permite editar contadores, portanto as partes em comum foram movidas para uma *view* em comum (*partial view*) de modo a evitar duplicação de código e facilitar alterações futuras. O sétimo bloco de código mostra um exemplo de utilização de secções da página de *layout*, em que o código definido dentro destas secções vai ser transportado para partes especificas na *view* de *layout*, neste caso o código da secção “Styles” vai ser transportado para dentro do elemento “<head>” na página de *layout* e o código da secção “Scripts” vai ser transportado para o final da página, antes de fechar o elemento “<body>”.

```
@model Website.ViewModels.Meters.MeterCreateViewModel                                    1
```

```
@{                                                                                       2
  ViewBag.TitleIcon = "plus";
  ViewBag.Title = "Add meter";
  ViewBag.PageDescription = "";
  const string errorMessage = "ErrorMessage";
}
<div class="row">
  <div class="col-md-10 col-lg-9">
    @if (TempData.ContainsKey(errorMessage))                                             3
    {
      <div class="callout callout-danger">
        <p>@TempData[errorMessage].ToString()</p>
      </div>
    }
    <div class="box box-success">
      @using (Html.BeginForm())
      {
        <div class="box-body">
          @Html.AntiForgeryToken()                                                       4
          @Html.ValidationSummary(true, null, new { @class = "text-danger" })
          <div class="form-group">
            @Html.LabelFor(model => model.MeterTemplateId,                               5
              new { @class = "control-label" })
            @Html.DropDownListFor(model => model.MeterTemplateId,
              Model.ViewBag.MeterTemplates,
              new {
                @class = "selectpicker form-control",
                data_live_search = "true"
              })
            @Html.ValidationMessageFor(model => model.MeterTemplateId,
              null, new { @class = "text-danger" })
          </div>
          @Html.Partial("_MeterCreateEditPartial", Model)                                6
        </div>
        <div class="box-footer clearfix">
          <div class="pull-right">
            <input class="btn btn-success" type="submit" value="Add" />
            <a class="btn btn-default" href="@Url.Action("Index")">Back to list</a>
          </div>
        </div>
      }
    </div>
  </div>
</div>
@section Styles                                                                          7
{
    @Styles.Render("~/bundles/css/bootstrap-select")
}
@section Scripts
{
    @Scripts.Render("~/bundles/bootstrap-select")
    <script src="~/Scripts/meters/create-edit.js"></script>
}
```

**Figura 7-42 – Exemplo de *view***

Os *controllers* são compostos por vários métodos denominados *actions*, cada *action* recebe, processa e devolve a resposta para um pedido HTTP específico. Quando uma

*action* apenas consulta e devolve dados do *model* é, normalmente, simples. Quando uma action faz alterações ao *model* (e portanto à base de dados) torna-se mais complexa, pois existem vários passos que antecedem e sucedem a operação concreta. Uma *action* que faz alterações ao *model* é composta por várias secções de código, sendo os mais comuns:

1. Deserialização (*unmarshalling*) do *model* ou *view model*.
2. Obtenção de dados auxiliares para a *view*.
3. Limpeza, validação e formatação do *model*.
4. Obtenção dos dados necessários para a operação.
5. Realização da operação.
6. Reversão da operação em caso de falha.
7. Passagem de dados e controlo para a respetiva *view* ou redirecção para outra *action*.

A Figura 7-43 mostra um exemplo de *view model*, este *view model* é utilizado para passar dados entre a *action* "Create" do *controller* "Meters" e a respetiva *view*.

```
public class MeterCreateViewModel : MeterCreateEditBase
{
  public MeterCreateViewBag ViewBag { get; set; }
  [Required]
  [DisplayName("Meter template")]
  public int? MeterTemplateId { get; set; }
}
```

**Figura 7-43 - Exemplo de *view model***

Este *view model* possui apenas uma propriedade, "MeterTemplateId", pois deriva de outra classe chamada "MeterCreateEditBase" que tal como o nome sugere é a classe base dos *view models* das *actions* "Create" e "Edit". Foi utilizada a herança neste caso pois existem propriedades em comum entre as actions "Create" e "Edit", deste modo é possível criar uma *partial view* com os elementos de formulário em comum que pode ser reutilizada, evitando a duplicação de código das *views*. A Figura 7-44 mostra a classe base do *view model* "MeterCreateViewModel", em que se pode ver as restantes propriedades que também são passadas entre a *action* "Create" e a respetiva *view*, o bloco de código marcado mostra um exemplo de utilização notar a anotação "Range" para limitar o intervalo de valores que podem ser utilizados para representar a latitude e longitude em graus decimais.

```
public class MeterCreateEditBase
{
  public MeterCreateEditCommonViewBag CommonViewBag { get; set; }

  [Required]
  [DisplayName("Name")]
  [MaxLength(50, ErrorMessage = "Maximum 50 characters")]
  public string Name { get; set; }
  [DisplayName("Serial Number")]
  public string SerialNumber { get; set; }
  [DisplayName("Enabled")]
  public bool Enabled { get; set; }
  [DisplayName("EUI")]
  public string Eui { get; set; }
  [DisplayName("Latitude")]
  [Range(-90, 90, ErrorMessage = "Latitude is out of range (-90 to 90)")]
  public decimal? Latitude { get; set; }
  [DisplayName("Longitude")]
  [Range(-180, 180, ErrorMessage = "Longitude is out of range (-180 to 180)")]
  public decimal? Longitude { get; set; }
  [DisplayName("Altitude")]
  public short? Altitude { get; set; }
  [DisplayName("Gateway")]
  public Guid? GatewayId { get; set; }
  [DisplayName("Area")]
  public int? AreaId { get; set; }
}
```

**Figura 7-44 - Classe base do *view model* "MeterCreateViewModel"**

A Figura 7-45 mostra um exemplo de definição de um dos *controllers*, neste caso chamado "MetersController", e de uma das suas *actions*, "Create", o *controller* possui a anotação "Authorize" que indica que o utilizador deve estar autenticado para aceder ao *controller* e a anotação "SessionExpiredActionFilter" que é uma anotação personalizada que redireciona o utilizador para outra página, caso não esteja autenticado (neste caso redireciona para a página de login).

A *action* possui a anotação "HttpPost" que indica que só processa pedidos com o verbo "POST", a anotação "ValidateAntiForgeryToken" verifica se o valor do *cookie* "__RequestVerificationToken" é igual ao valor enviado no formulário, evitando que outros sites externos possam fazer pedidos para esta *action* deste site (ataques CSRF), sendo que este mecanismo de proteção só se encontra nas *actions* que fazem algum tipo de alterações a *models*. Por fim, a anotação "Authorize" indica que o utilizador deve ter pelo menos um dos papéis especificados na lista para poder aceder à *action*, neste caso o utilizador tem que pertencer ao grupo "SuperAdmin" ou "Admin" para poder aceder a esta *action*.

O *view model* "MeterCreateViewModel" é adicionado como parâmetro "mc" da *action* "Create" para indicar à *framework* que quando receber pedidos HTTP da *view*, deve tentar deserializar o pedido para o *view model*, neste caso como a *action* trata do verbo "POST", os dados vêm no corpo do pedido.

```
[Authorize]
[SessionExpiredActionFilter(RedirectTo = "login")]
public class MetersController : Controller
{
  ...
  [HttpPost]
  [ValidateAntiForgeryToken]
  [Authorize(Roles = "SuperAdmin,Admin")]
  public async Task<ActionResult> Create(MeterCreateViewModel mc)
  {
    ...
  }
  ...
}
```

**Figura 7-45 – Exemplo de definição da *action* e deserialização do *view model***

A Figura 7-46 mostra um exemplo de obtenção de dados auxiliares para a *view*. Estes dados são utilizados para criar uma interface de utilizador mais amigável, por exemplo em vez de pedir ao utilizador que insira o identificador do *gateway* a que pretende associar um contador, a *view* mostra uma lista de *gateways* e o utilizador pode simplesmente selecionar um dos *gateways* da lista. Estes dados são preenchidos logo no início da *action*, pois caso ocorra algum erro que possa ser corrigido pelo utilizador a partir do *website*, vão ser necessários para mostrar a mesma *view* com o erro. De notar que é adicionado um registo no *log* caso ocorra uma exceção (neste caso é mostrada a exceção base "Exception" para reduzir o tamanho do código apresentado) quando um erro não pode ser corrigido pelo utilizador a partir do *website*, adicionalmente é mostrada uma mensagem de erro ao utilizador que indica que ocorreu um erro interno.

```
public async Task<ActionResult> Create(MeterCreateViewModel mc)
{
  ...
  try
  {
    mc.ViewBag = FillCreateViewBag(tenantId);
    mc.CommonViewBag = FillCreateEditCommonViewBag(tenantId);
  }
  catch (Exception e)
  {
    await LogCore.AddEntry(new LogEntry
    {
      Level = LogLevel.Error,
      CreatedAt = DateTimeOffset.Now,
      SourceName = WebSiteConstants.LogSourceName,
      Method = $"{ControllerFqn}.{actionName}",
      Message = $"{WebSiteLogMessages.ViewBagError}: {e.Message}",
      ExtraInfo = string.Format("MeterName=\"{0}\" MeterTemplateId={1} " +
        "MeterEUI=\"{2}\" MeterGatewayId={3}",
        mc.Name, mc.MeterTemplateId, mc.Eui, mc.GatewayId),
      TenantId = tenantId,
      System = false
    });
    TempData[errorMessage] = WebSiteMessages.ViewBagError;
    return View(mc);
  }
  ...
}
```

**Figura 7-46 - Exemplo de obtenção de dados auxiliares para a *view***

Os dados do *view model* são limpos, validados e armazenados num formato padrão. A Figura 7-47 mostra parte do código que trata de limpar, validar e formatar os dados do *view model*. No primeiro bloco de código marcado são limpos os caracteres normalmente utilizados para separar bytes do EUI e caracteres invisíveis que possam ser introduzidos acidentalmente. No segundo bloco o EUI é validado e caso seja inválido é adicionada uma mensagem de erro ao *view model*. No terceiro e quarto bloco o EUI é formatado dependendo do número de bits. No quinto bloco é verificado se o *view model* é valido e caso não seja a *view* é mostrada de novo com os respetivos erros.

```
public async Task<ActionResult> Create(MeterCreateViewModel mc)
{
  ...
  string meterEui = null;
  if (!string.IsNullOrEmpty(mc.Eui))
  {
    string eui = mc.Eui
      .Replace("-", "")
      .Replace(":", "")
      .Replace(" ", "")
      .Replace("\t", "");

    bool euiIsEui48 = new Regex("^[a-fA-F0-9]{12}$").IsMatch(eui);
    bool euiIsEui64 = new Regex("^[a-fA-F0-9]{16}$").IsMatch(eui);
    if (!euiIsEui48 && !euiIsEui64)
    {
      ModelState.AddModelError("Eui", WebSiteMessages.MeterInvalidEui);
    }
    if (euiIsEui48)
    {
      meterEui = string.Format("{0}-{1}-{2}-{3}-{4}-{5}",
        eui.Substring(0, 2), eui.Substring(2, 2),
        eui.Substring(4, 2), eui.Substring(6, 2),
        eui.Substring(8, 2), eui.Substring(10, 2));
    }
    else if (euiIsEui64)
    {
      meterEui = string.Format("{0}-{1}-{2}-{3}-{4}-{5}-{6}-{7}",
        eui.Substring(0, 2), eui.Substring(2, 2),
        eui.Substring(4, 2), eui.Substring(6, 2),
        eui.Substring(8, 2), eui.Substring(10, 2),
        eui.Substring(12, 2), eui.Substring(14, 2));
    }
  }
  ...
  if (!ModelState.IsValid)
  {
    TempData[errorMessage] = WebSiteMessages.PleaseCorrectMarkedItems;
    return View(mc);
  }
  ...
}
```

**Figura 7-47 – Exemplo de limpeza, validação e formatação do *view model***

Os dados necessários para a realização da operação são obtidos a partir das respetivas fontes de dados. A Figura 7-48 mostra parte do código que trata de obter os dados necessários para a operação. O primeiro bloco de código obtém os dados do modelo de contador a partir da base de dados. O segundo bloco de código verifica se o modelo de contador foi encontrado e caso não tenha sido encontrado mostra a *view* de novo com uma mensagem de erro. O terceiro bloco de código verifica se o modelo de contador

possui sensores e caso não possua sensores mostra a *view* de novo com uma mensagem de erro.

```
public async Task<ActionResult> Create(MeterCreateViewModel mc)
{
  ...
  MeterTemplate meterTemplate = null;
  try
  {
    meterTemplate = await Db.MeterTemplates
      .Where(t =>
        t.Id == mc.MeterTemplateId &&
        t.TenantId == tenantId)
      .Include(t => t.MeterSensors)
      .FirstOrDefaultAsync();
  }
  catch (Exception e) {
    await LogCore.AddEntry(new LogEntry
    {
      ...
    });
  }
  if (meterTemplate == null)
  {
    TempData[errorMessage] = WebSiteMessages.MeterTemplateNotFound;
    return View(mc);
  }

  if (meterTemplate.MeterSensors.Count == 0)
  {
    TempData[errorMessage] = WebSiteMessages.MeterTemplateHasNoSensors;
    return View(mc);
  }
  ...
}
```

**Figura 7-48 – Exemplo de obtenção de dados para executar a operação**

Quando estão reunidas todas as condições para a executar a operação, os vários passos que compõem a operação são executados. A Figura 7-49 mostra um exemplo de código que executa uma operação, neste caso adicionar um contador. O primeiro bloco de código marcado cria um objeto que armazena informações do novo contador, adiciona esse objeto à coleção de contadores, tenta guardar as alterações na base de dados, se não conseguir guardar o contador na base de dados ativa um marcador (rollback) que indica a operação falhou e que as alterações feitas até ao momento devem ser revertidas, adiciona um registo no *log* com detalhes sobre o sucedido e gera uma mensagem de erro que será mostrada na *view* ao utilizador. O segundo bloco de código verifica se o primeiro bloco de código falhou, caso não tenha falhado tenta criar a tabela de leituras do contador na base de dados de leituras. Se o segundo bloco de código falhar o procedimento é igual, ativar o marcador de reversão das alterações, adicionar um registo no *log* e mostrar o erro ao utilizador. As restantes suboperações comportam-se de forma semelhante.

```
public async Task<ActionResult> Create(MeterCreateViewModel mc)
{
  ...
  bool rollback = false;
  int operation = 0;
  Meter meter = new Meter
  {
    Id = GetUnusedMeterGuid(),
    MeterTemplateId = mc.MeterTemplateId.GetValueOrDefault(),
    Name = mc.Name,
    GatewayId = mc.GatewayId,
    Eui = meterEui,
    ...
    TenantId = tenantId,
  };
  Db.Meters.Add(meter);
  try
  {
    await Db.SaveChangesAsync();
  }
  catch (Exception e)
  {
    rollback = true;
    await LogCore.AddEntry(new LogEntry
    {
      ...
    });
    TempData[errorMessage] = WebSiteMessages.MeterInsertDbError;
  }
  operation++;
```

```
  if (!rollback)
  {
    ...
    OperationResult createDbTable =
      MeterCore.CreateDbReadingsTable(meter.Id, sensors);
    if (!createDbTable.OperationCompleted)
    {
      rollback = true;
      ...
      TempData[errorMessage] = createDbTable.Message;
    }
    operation++;
  }
```

```
  if (!rollback)
  {
    ...
  }
  ...
}
```

**Figura 7-49 - Exemplo de execução de uma operação**

Caso ocorra algum erro ao realizar a operação as alterações feitas até ao momento devem ser revertidas, de modo a evitar afetar a integridade e consistência dos dados na base de dados e retornar o sistema ao estado anterior. A Figura 7-50 mostra um exemplo de código que reverte as operações realizadas até ao momento. O primeiro bloco de código marcado remove o contador se este foi adicionado. O segundo bloco de código marcado remove a tabela de leituras do contador na base de dados de leituras. Caso alguma operação de reversão falhe é ativado um marcador que indica que a essa operação de reversão falhou. O terceiro bloco de código marcado adiciona um registo

no *log* com a listagem das operações de reversão que falharam, gera uma mensagem de erro que será mostrada na *view* ao utilizador e passa o controlo para a *view*.

```
public async Task<ActionResult> Create(MeterCreateViewModel mc)
{
  ...
  if (rollback)
  {
    bool rollbackFailed = false;
    List<string> rollbackErrors = new List<string>();
    if (operation > 0)
    {
      Db.Meters.Remove(meter);
      try
      {
        await Db.SaveChangesAsync();
      }
      catch (Exception)
      {
        rollbackFailed = true;
        rollbackErrors.Add(WebSiteLogMessages.MeterRemove);
      }
    }

    if (operation > 1)
    {
      OperationResult deleteDbTable = MeterCore.DeleteDbReadingsTable(meter.Id);
      if (!deleteDbTable.OperationCompleted)
      {
        rollbackFailed = true;
        rollbackErrors.Add(WebSiteLogMessages.MeterDeleteReadingsTable);
      }
    }
    ...
    if (rollbackFailed)
    {
      string failedOperations = string.Join("; ", rollbackErrors);
      await LogCore.AddEntry(new LogEntry
      {
        ...
        Message = string.Format("{0}; {1}", WebSiteLogMessages.MeterAddError,
          WebSiteLogMessages.RollbackFailed),
        ExtraInfo = string.Format("FailedOperations=\"{0}\" MeterName=\"{1}\" " +
          "MeterTemplateId={2} MeterTemplateName=\"{3}\" GatewayName=\"{4}\"",
          failedOperations, meter.Name, meterTemplate.Id, meterTemplate.Name,
          gateway.Name),
        ...
      });
      TempData[errorMessage] += $"; {WebSiteMessages.RollbackFailed}";
    }
    return View(mc);
  }
  ...
}
```

**Figura 7-50 - Exemplo de reversão de uma operação**

Por fim, se a operação foi bem-sucedida, o controlo é passado para a respetiva *view* e o código HTML gerado pela *view* é enviada para o *web browser*. Alternativamente o controlo pode ser passado para outra *action*, e essa *action* por sua vez pode passar o controlo para a sua *view* ou para outra *action*. A Figura 7-51 mostra um exemplo de passagem de controlo para outra *action*, neste caso o controlo vai ser passado para a

*action* "Index" que faz parte do mesmo *controller* que a *action* "Create". O primeiro bloco de código marcado redireciona para *action* "Create" e o segundo bloco de código marcado é a definição dessa *action* dentro do mesmo *controller*.

```
[Authorize]
[SessionExpiredActionFilter(RedirectTo = "login")]
public class MetersController : Controller
{
  ...
  [HttpPost]
  [ValidateAntiForgeryToken]
  [Authorize(Roles = "SuperAdmin,Admin")]
  public async Task<ActionResult> Create(MeterCreateViewModel mc)
  {
    ...
    return RedirectToAction("Index");
    ...
  }
  ...
  [Authorize(Roles = "SuperAdmin,Admin,ReadOnly")]
  public async Task<ActionResult> Index(int? page, int? items, string search,
    string orderBy, string sortingOrder)
  {
    ...
  }
  ...
}
```

**Figura 7-51 - Exemplo de passagem de controlo para outra *action***

Existem outros tópicos interessantes a desenvolver, tais como as classes de configuração do *website*, classes de configuração de rotas para os URLs, action filters, classes de *hub* do SignalR ou os bundles de CSS e JavaScript, mas como esta secção ficaria muito extensa resolvi focar-me apenas na implementação de uma página *web* utilizando o padrão MVC.

## 7.9. IIS - Web Service

A função do *web service* no IIS é fornecer uma interface para receber e processar pedidos dos *gateways*, permitindo que estes possam consultar ou armazenar dados na base de dados. O *web service* no IIS é utilizado pelos *gateways* para obter configurações durante o arranque e para receber mensagens dos contadores, tais como leituras, alertas ou o estado atual da bateria. Do ponto de vista do *web service* no IIS, um *gateway* é qualquer componente que serve de intermediário entre o *web service* no IIS e os contadores e que pode enviar pedidos HTTP e processar as respostas. Inicialmente os *gateways* eram *gateways* físicos (MultiConnect Conduit da MultiTech) que comunicavam diretamente com os contadores, mas na implementação atual os *gateways* são instâncias do Node-RED que comunicam com os contadores através do LoRa App Server e outros componentes de *software* e *hardware* (ver seção 6.2). Os motivos principais para desenvolver um *web service* são a possibilidade de interagir facilmente com outros sistemas, pois o protocolo HTTP é aberto, gratuito e comum, e a possibilidade de simplificar a configuração dos *gateways*, pois a maioria das *firewalls* permite tráfego de saída para os portos utilizados (80 para HTTP e 443 para HTTPS). A Figura 7-52 mostra um diagrama simples da função do *web service* no IIS.

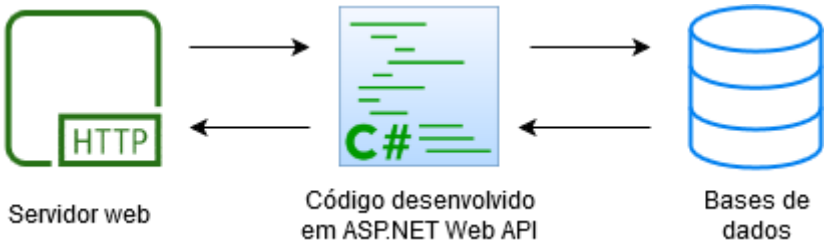



**Figura 7-52 - Função do *web service* no IIS**

### 7.9.1. Programação

O *web service* foi implementado com recurso à *framework* "ASP.NET Web API" que facilita a implementação de *web services* e tem como base a *framework* .NET Framework. De acordo com o requisito de desenvolver código do lado do servidor com a linguagem de programação C# foram consideradas outras opções, umas mais complexas (p. ex.: WCF) mas que continham demasiadas funcionalidades que não eram necessárias, tornando o *web service* desnecessariamente mais complexo e outras mais simples (p. ex.: ASP:NET Web Service) mas que necessitam de código e configurações adicionais para funcionar em conjunto com o *website*. A *framework* ASP.NET Web API fornece mecanismos para receber pedidos HTTP, processar o URL dos pedidos e através de regras de encaminhamento chamar métodos que processam o pedido e geram uma resposta.

Os componentes mais importantes da *framework* ASP.NET Web API são os métodos que processam os pedidos HTTP, que na nomenclatura do padrão MVC são denominados *actions*. As *actions* são agrupadas em *controllers*, sendo estes *controllers* parte do padrão de arquitetura MVC. Existe uma *action* por cada pedido definido na especificação do *web service* (Anexo D: Especificação do *web service* no IIS). Uma *action* é composta por várias secções de código que têm uma determinada responsabilidade, salvo algumas exceções as *actions* contêm normalmente as seguintes secções:

1. Deserialização (*unmarshalling*) dos parâmetros do pedido para o *model*.
2. Validação do *model*.
3. Obtenção dos dados necessários da base de dados.
4. Validação adicional.
5. Processamento do pedido e geração da resposta.

Para demonstrar estas secções será utilizada a *action* "Insert" do *controller* "Alerts". A função desta action é receber alertas dos contadores, emitir o alerta para o *website* de administração e guardar o alerta na base de dados. A Figura 7-53 mostra um exemplo de deserialização os parâmetros do pedido. Neste caso do pedido de inserção de alertas, esta *action* recebe os parâmetros no corpo do pedido HTTP no formato JSON, a *framework* ASP.NET WebAPI deserializa estes parâmetros e coloca automaticamente os valores dos parâmetros no *model* especificado, que neste caso é o "AlertJson".

```
public class AlertsController : ApiController
{
  ...
  [HttpPost]
  public async Task<IHttpActionResult> Insert(AlertJson alertJson)
  {
    ...
  }
  ...
}
```

**Figura 7-53 – Exemplo de deserialização dos parâmetros de um pedido HTTP**

De seguida o *model* é validado com recurso aos mecanismos da *framework* ASP.NET MVC. A Figura 7-54 mostra o bloco de código destacado que verifica se o *model* é valido. Caso a validação falhe adiciona um registo no *log* e gera uma resposta para o cliente que indica os motivos pelos quais a validação falhou.

```
public class AlertsController : ApiController
{
  ...
  [HttpPost]
  public async Task<IHttpActionResult> Insert(AlertJson alertJson)
  {
    ...
    if (!ModelState.IsValid)
    {
      List<ValidationError> validationErrors =
        WebApiHelper.FormatModelState(ModelState);
      string alertJsonDateTime = alertJson.dateTime.HasValue ?
        alertJson.dateTime.Value.ToString("u") : string.Empty;
      await LogCore.AddEntry(new LogEntry
      {
        Level = LogLevel.Error,
        CreatedAt = DateTimeOffset.Now,
        SourceName = WebApiConstants.LogSourceName,
        Method = $"{ControllerFqn}.{actionName}",
        Message = $"{WebApiLogMessages.ModelNotValid}: " +
          string.Join("; ", validationErrors.Select(v => v.Error)),
        ExtraInfo = string.Format("meterId={0} code={1} dateTime=\"{2}\"",
           alertJson.meterId, alertJson.code, alertJsonDateTime),
        TenantId = auth.TenantId,
        System = false
      });
      return Content(HttpStatusCode.BadRequest, validationErrors);
    }
    ...
  }
  ...
}
```

**Figura 7-54 – Exemplo de validação dos dados do pedido**

A validação é feita através de regras de validação definidas no *model*. Deste modo é possível reutilizar as regras de validação em outras partes do sistema, caso necessário. A Figura 7-55 mostra o *model* onde os dados JSON do pedido são colocados depois de serem deserializados, as propriedades “meterId”, “code” e “dateTime” são obrigatórias pois têm a *data annotation* “[Required]”. Adicionalmente o método “Validate” permite

fazer outros tipos de validação não suportadas pelas *data annotations*, neste caso apenas verifica se a data do alerta antecede a data de inicio do calendário Unix.

```
public class AlertJson : IValidatableObject
{
  [Required]
  public Guid? meterId { get; set; }
  [Required]
  public int? code { get; set; }
  [Required]
  public DateTimeOffset? dateTime { get; set; }

  public IEnumerable<ValidationResult> Validate(
    ValidationContext validationContext)
  {
    DateTimeOffset minimumDate =
      new DateTimeOffset(1970, 1, 1, 0, 0, 0, TimeSpan.Zero);
    if (DateTimeOffset.Compare(dateTime.GetValueOrDefault(), minimumDate) < 0)
    {
      yield return new ValidationResult(
        $"Minimum supported date is {minimumDate.ToString("u")}",
        new[] { "dateTime" });
    }
  }
}
```

**Figura 7-55 - *Model* com regras de validação**

Os dados necessários ao processamento do pedido são obtidos a partir da base de dados, e em seguida são efetuadas validações adicionais. A Figura 7-56 mostra um exemplo de bloco de código que obtém um alerta da base de dados. De notar que se não existir um alerta de sistema com o código especificado ou um alerta pertencente ao *tenant* atual com o código especificado é dada a resposta “alerta não encontrado” ao cliente, adicionalmente se ocorrer algum erro ao consultar a base de dados esse erro é registado no *log* do sistema.

```
public class AlertsController : ApiController
{
  ...
  [HttpPost]
  public async Task<IHttpActionResult> Insert(AlertJson alertJson)
  {
    ...
    Alert alert = null;
    try
    {
      alert = await Db.Alerts
        .Where(a =>
          a.Code == alertJson.code &&
          (a.TenantId == auth.TenantId || a.System == true))
        .FirstOrDefaultAsync();
    }
    catch (Exception e)
    {
      await LogCore.AddEntry(new LogEntry
      {
        ...
      });
      ...
    }
    if (alert == null)
    {
      await LogCore.AddEntry(new LogEntry
      {
        ...
      });
      ...
      return Content(HttpStatusCode.NotFound, WebApiResponses.AlertNotFound);
    }
    ...
  }
  ...
}
```

**Figura 7-56 – Exemplo de obtenção de dados da base de dados**

Após obter e validar todos os dados necessários, o pedido é processado. A Figura 7-57 mostra um exemplo de bloco de código que cria uma nova notificação, adiciona a notificação à base de dados e publica a notificação em tempo real para todos os utilizadores com sessão iniciada que pertencem ao *tenant* do contador que emitiu o alerta. Por fim, é enviada uma resposta ao cliente que confirma que a operação foi executada com sucesso, no corpo da resposta é enviado um objeto JSON com uma mensagem de confirmação, neste caso a resposta tem o código HTTP 201, que significa que o recurso (notificação de alerta) foi criado.

```
public class AlertsController : ApiController
{
  ...
  [HttpPost]
  public async Task<IHttpActionResult> Insert(AlertJson alertJson)
  {
    ...
    var notification = new Notification
    {
      Date = alertJson.dateTime.GetValueOrDefault(),
      SourceId = alert.Id,
      SourceName = alert.Name,
      Title = $"{meter.Name} - {alert.Name}",
      Message = $"Meter {meter.Name} has sent alert {alert.Name}",
      Type = (int)NotificationType.Alert,
      Viewed = false,
      TenantId = meter.TenantId
    };
    Db.Notifications.Add(notification);
    try
    {
      await Db.SaveChangesAsync();
    }
    catch (Exception e)
    {
      await LogCore.AddEntry(new LogEntry
      {
        ...
      });
      return Content(HttpStatusCode.InternalServerError,
        WebApiResponses.AlertInsertDbError);
    }
    NotificationsHub.Publish(new NotificationRealTime
    {
      Id = notification.Id,
      Title = notification.Title,
      Message = notification.Message,
      Type = notification.Type,
      TenantId = auth.TenantId
    });
    return Content(HttpStatusCode.Created, WebApiResponses.AlertInserted);
  }
  ...
}
```

**Figura 7-57 - Exemplo de processamento de pedido**

## 7.9.2. Comunicação com o *website*

Inicialmente o *web service* servia apenas como um meio de aceder de forma controlada à base de dados, de modo a ser possível, por exemplo, inserir leituras dos contadores ou obter configurações dos *gateways*. Mais tarde decidi que seria útil ver informações em tempo real no *website* assim que estas fossem recebidas pelo *web service* (p. ex.: alertas), sendo necessário desenvolver um mecanismo de comunicação entre o *web* service e o *website*.

O *web service* era implementado de forma isolada do *website*, ou seja, ambos os sistemas eram executados como duas aplicações distintas no IIS. Esta situação causava problemas quando um dos sistemas era atualizado no servidor e o outro não era, por exemplo, quando novas versões tornavam a interface entre estes sistemas incompatível.

Para prevenir esta situação o *web service* e o *website* foram integrados no mesmo projeto, deste modo são sempre atualizados em simultâneo e o *web service* pode enviar informações em tempo real para o *website* de administração sem recorrer a interfaces externas.

# 7.10. SQL Server

O SQL Server da Microsoft foi o SGBD utilizado neste projeto pois era um requisito, no entanto poderia ter sido utilizada outra base de dados desde que fornecesse o respetivo conector. No SQL Server existem duas bases de dados: uma para armazenar leituras dos contadores e outra para armazenar todos os dados restantes.

## 7.10.1. Base de dados principal

A base de dados principal armazena toda a informação do sistema à exceção das leituras que são armazenadas numa base de dados separada. Todos os utilizadores de uma instância partilham esta base de dados, o que reduz a carga tanto a nível de memória como em tempo de processador. São criados grupos de utilizadores denominados "tenants" que têm acesso apenas a parte dos dados armazenados nesta base de dados, o que permite que mais do que um departamento, filial ou empresa possa partilhar esta base de dados, esta funcionalidade denominada "multitenancy" é explicada em mais detalhe na seção 9.4.1.

A Tabela 7-1 explica resumidamente a função das tabelas da base de dados principal.

| Tabela | Descrição |
|---|---|
| __MigrationHistory | Faz parte da Entity Framework. Contém informações sobre as versões anteriores e a versão atual da base de dados. |
| Alerts | Contém informações sobre os alertas de sistema e os alertas personalizados, tais como o código que os identifica, o nome e a descrição que é apresentada ao utilizador. |
| Areas | Contém informações sobre as áreas onde pode existir um ou mais contadores. Serve de base para a estrutura do sinóptico. |
| AspNetRoles | Faz parte do sistema ASP.NET Identity. Contém informações sobre os "papéis" que podem ser atribuídos aos utilizadores. Este é o mecanismo de autorização utilizado no projeto. |
| AspNetUserClaims | Faz parte do sistema ASP.NET Identity. Suporte a um mecanismo de autorização alternativo baseado em "direitos". Não é utilizado neste projeto. |
| AspNetUserLogins | Faz parte do sistema ASP.NET Identity. Guarda informações sobre logins através de sistemas de terceiros (p. ex.: Google, Facebook). Não é utilizado neste projeto. |

| | |
|---|---|
| AspNetUserRoles | Faz parte do sistema ASP.NET Identity. Associa um ou mais utilizadores a um ou mais "papéis". |
| AspNetUsers | Faz parte do sistema ASP.NET Identity. Contém informações sobre os utilizadores que se podem autenticar no sistema. |
| BatteryStatus | Contém o histórico do estado da bateria dos contadores. |
| CustomCommandFields | Contém informações sobre os campos que compõem um comando personalizado. |
| CustomCommands | Contém informações sobre os comandos personalizados que podem ser adicionados a um modelo de contador. |
| DashboardCharts | Contém informações sobre os gráficos personalizados que são mostrados nos dashboards. |
| DashboardChartSources | Contém informações sobre fontes de dados que são mostrados nos gráficos do dashboard. |
| Dashboards | Contém informações sobre os dashboards que são apresentados na página inicial no *website* de administração e quais os gráficos que são apresentados. |
| FloorPlans | Contém informações sobre a planta de uma determinada área. Estes dados são utilizados pelo sinóptico para mostrar a planta das áreas que possuem planta. |
| Gateways | Contém informações e configurações dos *gateways* através dos quais os contadores comunicam com o sistema. |
| LogRecords | Contém registos de eventos importantes, tais como falhas na execução de operações. |
| Meters | Contém informações e configurações dos contadores que efetuam leituras, emitem alertas, executam comandos, etc. |
| MeterSensors | Contém informações e configurações dos sensores que compõem os modelos de contador. |
| MeterTemplateAlerts | Associa múltiplos alertas a múltiplos modelos de contador. |
| MeterTemplates | Contém informações sobre os modelos de contadores, cada modelo de contador tem pelo menos um sensor associado. Opcionalmente associa também comandos personalizados, alertas personalizados e acionadores (*triggers*). |
| Notifications | Contém notificações, tais como mensagens do sistema, o registo (*log*) dos contadores, alertas emitidos pelos contadores e as ocasiões em que os contadores ativaram acionadores (*triggers*). |

| | |
|---|---|
| SignalMappingPoints | Contém informações sobre medições das condições do sinal dos contadores, que são efetuadas para testar a cobertura do sinal. |
| SignalMappingSessions | Agrupa medições de sinal dos contadores em sessões. |
| SvgShapes | Contém informações sobre formas vetoriais no formato SVG que podem ser usadas para representar uma área na planta. Estes dados são utilizados pelo sinóptico para delimitar e poder interagir com áreas. |
| Tenants | Agrupa os utilizadores em conjuntos que têm acesso aos mesmos dados. |
| TriggerConditions | Contem informações sobre as condições em que um determinado *trigger* deve ativar-se. |
| Triggers | Contém informações sobre os *triggers* associados a um modelo de contador que emitem notificações quando são acionados. |

**Tabela 7-1 - Função das tabelas da base de dados principal**

O modelo ER da base de dados e uma descrição mais detalhada encontram-se no Anexo B: Especificação da base de dados principal.

### 7.10.2. Entity framework

O acesso à base de dados principal é feito com recurso à Entity Framework que é uma *framework* do tipo ORM (Object-Relational Mapping). Permite trabalhar com dados relacionais utilizando objetos de domínio diretamente, reduzindo o código necessário para aceder à base de dados.

Fornece três modos diferentes de fazer o mapeamento entre a base de dados e os objetos, dependendo da parte desta relação que o programador pretende desenvolver. Um dos modos, *model first*, permite criar um modelo entidade-relacionamento com recurso a uma ferramenta própria (Entity Framework Designer) e a partir desse modelo criar a estrutura da base de dados e as classes que representam as tabelas e as suas relações. Outro modo, *database first*, permite que se crie a base de dados, tabelas e relações da forma que se achar mais conveniente. Posteriormente este modo faz engenharia reversa do modelo da base de dados e a partir desse modelo cria as classes que representam as tabelas e as suas relações. O último modo é o *code first*, este foi o modo utilizado neste projeto.

### 7.10.3. Code First

O modo *code first* permite criar as classes, estabelecer relações entre elas e posteriormente gerar a estrutura da base de dados. Na base de dados classes são representadas por tabelas, as propriedades dessas classes por colunas das tabelas e as relações entre classes com relações entre tabelas. Para configurar as colunas das tabelas são utilizadas *Data Annotations* que permitem especificar, por exemplo, se uma coluna é chave primária. A Figura 7-58 mostra parte de uma classe desenvolvida para o modo *code first*.

```
public class Meter
{
  [Key]
  public Guid Id { get; set; }
  [Required]
  [DisplayName("Name")]
  [MaxLength(50, ErrorMessage = "Maximum 50 characters")]
  public string Name { get; set; }
  [DisplayName("Enabled")]
  public bool Enabled { get; set; }
  [MaxLength(23, ErrorMessage = "Maximum 23 characters")]
  [DisplayName("EUI")]
  public string Eui { get; set; }
  [DisplayName("Latitude")]
  public decimal? Latitude { get; set; }
  [DisplayName("Longitude")]
  public decimal? Longitude { get; set; }
  [DisplayName("Altitude")]
  public short? Altitude { get; set; }
  [DisplayName("Meter template")]
  public int MeterTemplateId { get; set; }
  public virtual MeterTemplate MeterTemplate { get; set; }
  ...
}
```

**Figura 7-58 - Exemplo de classe utilizada pelo *Code First***

A classe representa um contador, a chave primária "Id" é definida com a anotação "[Key]", a restrição de obriga a que o valor não seja nulo é definida com a anotação "[Required]". O tipo de dados das colunas da tabela são definidos pelo tipo de dados das propriedades da classe, por exemplo a propriedade "Name" é do tipo *string* e será representado na base de dados com o tipo de dados mais adequado, neste caso VARCHAR. A anotação "[MaxLength]" define o tamanho máximo da propriedade, que neste caso é 50 caracteres. Ainda é possível definir uma mensagem de erro por omissão que aparece na interface gráfica caso o utilizador exceda o tamanho máximo da propriedade, no caso da propriedade "Name" aparece a mensagem "Maximum 50 characters".

### 7.10.4. Migrações e controlo de versões

Ao longo do desenvolvimento do projeto as classes do domínio foram sendo alteradas, e classes, propriedades e relações foram criados, modificados ou removidos. Em vez de criar a estrutura da base de dados principal novamente e perder dados existentes, foram utilizadas as migrações *code first* para atualizar a base de dados. As migrações *code first*, depois de configuradas, criam a estrutura da base de dados a partir das classes que fazem parte da classe de contexto da base de dados. Qualquer alteração posterior às classes de domínio pode ser refletida na estrutura da base de dados criando mais migrações. A Figura 7-59 mostra a propriedade "Title" da classe "DashboardChart", a Figura 7-60 mostra a mesma propriedade e classe com uma anotação adicional "[Required]" de modo a não permitir que a propriedade "Title" seja nula.

```
public class DashboardChart
{
    ...
    [StringLength(40)]
    public string Title { get; set; }
    ...
}
```

**Figura 7-59 – Anotações do atributo "Title"**

```
public class DashboardChart
{
    ...
    [Required]
    [StringLength(40)]
    public string Title { get; set; }
    ...
}
```

**Figura 7-60 – Alteração das anotações do atributo "Title"**

Um passo intermédio na criação da estrutura da base de dados é a criação do modelo ER que fica armazenado na tabela "__MigrationHistory" criada pelo *code first*. Posteriormente pode-se fazer alterações no código e criar uma nova migração, que gera um novo modelo ER e compara com o último modelo ER guardado na base de dados. A Figura 7-61 mostra o comando que gera uma nova migração.

```
PM> Add-Migration ChartTitleNotNull
Scaffolding migration 'ChartTitleNotNull'.
The Designer Code for this migration file includes a snapshot of your
current Code First model. This snapshot is used to calculate the changes
to your model when you scaffold the next migration. If you make additional
 changes to your model that you want to include in this migration, then
you can re-scaffold it by running 'Add-Migration ChartTitleNotNull' again.
PM>
```

**Figura 7-61 - Migração do modelo *Code First***

A partir desta migração pode-se gerar as instruções SQL que atualizam a base de dados com recurso ao *code first*, de forma a suportar o novo modelo. A qualquer altura pode-se migrar uma base de dados existente para uma versão mais recente ou mais antiga, sendo este o método de controlo de versões da base de dados. A Figura 7-63 mostra as instruções SQL geradas para atualizar a base de dados para a nova versão e a Figura 7-63 mostra as instruções SQL geradas para reverter a base dados para a versão anterior, ambas as instruções foram geradas pela migração *code first* criada na Figura 7-61.

```
ALTER TABLE [dbo].[Charts] ALTER COLUMN [Title] [nvarchar](40) NOT NULL
```

**Figura 7-62 – Instruções SQL para migrar a base de dados para a nova versão**

```
ALTER TABLE [dbo].[Charts] ALTER COLUMN [Title] [nvarchar](40) NULL
```

**Figura 7-63 – Instruções SQL para reverter a base de dados para a versão anterior**

O script SQL gerado pelas migrações *code first* é executado primeiro na base de dados local, as alterações são testadas e só depois é copiado para o servidor de demonstração em que é executado quando for conveniente (normalmente quando ninguém esta a fazer testes no terreno com *hardware* ou a demonstrar o sistema para o cliente).

### 7.10.5. Base de dados de leituras

A base de dados de leituras armazena as leituras reportadas pelos contadores. O motivo pelo qual foi criada uma base de dados separada deve-se ao facto das consultas relacionadas com leituras serem efetuadas com *prepared statements* e *stored procedures* escritos manualmente, enquanto que a base de dados principal é criada, atualizada e consultada sempre através da Entity Framework. Deste modo assegura-se as migrações *code first* não interferem com as tabelas de leituras, *prepared statements* ou *stored procedures* e que os *prepared statements* ou *stored procedures* não interferem com as migrações do *code first*.

A Tabela 7-2 explica resumidamente a função das tabelas da base de dados de leituras de contadores.

| Tabela | Descrição |
|---|---|
| aggregated_meter_readings | Guarda leituras dos contadores, agregadas por vários intervalos de tempo e por várias funções de agregação, acelerando a consulta de leituras agregadas, pois não é necessário agregar leituras em cada consulta. |
| stored_procedure_log | Guarda registos de eventos importantes que acontecerem durante a execução de stored procedures. É útil para resolver problemas relacionados com stored procedures. |
| meter_[ meter-id ] | Tabela de leituras de um contador. Existe uma por contador, o nome da tabela é composta pelo prefixo "meter_" seguido do identificador único do contador (GUID). Exemplo: "meter_6b154a64-581f-47a7-962f-8ce406b1fa35". |

**Tabela 7-2 - Função das tabelas da base de dados de leituras**

Esta base de dados encontra-se na mesma instância do SQL Server de desenvolvimento, na mesma máquina, pois esta máquina consegue processar o volume de leituras atual dos contadores. Embora seja também uma base de dados relacional, contrasta com a base de dados principal onde se fazem principalmente operações CRUD. Esta base de dados é utilizada principalmente para inserir novos registos ou consultar registos existentes, tendo portanto um perfil de utilização diferente. Sabe-se à partida que o número de leituras vai estar constantemente a aumentar, assim separar dados de leitura facilita a administração do sistema, por exemplo, caso seja necessário migrar as leituras para outra máquina com mais capacidade os utilizadores continuam com acesso ao *website* de administração, deixando apenas de ver as leituras durante o processo.

Quando projeto é compilado não se sabe à partida a quantidade de sensores que os contadores podem possuir ou quais os tipos de dados que esses sensores reportam. O

modelo que foi utilizado inicialmente para resolver este problema foi o Entity–Attribute–Value (EAV)[14]. Este modelo define três tabelas, uma para a entidade (neste caso os contadores), outra para os atributos (neste caso os sensores) e uma última para os valores (neste caso as leituras). Este modelo é ineficiente pois requer acesso a 3 tabelas antes de se poder inserir novos dados, sendo a inserção a operação mais frequente das leituras este modelo não é o mais indicado. A consulta dos dados também torna-se mais lenta pois todas as leituras de todos os sensores de todos os contadores encontram-se na mesma tabela, sendo necessário restringir a consulta apenas a leituras de um determinado contador e ordenar os resultados. A Figura 7-64 mostra um diagrama de entidade-relacionamento simplificado do modelo EAV, este modelo guarda apenas valores do tipo "decimal" com dez algarismos no total e duas casas decimais (tabela "valores", coluna "valor").

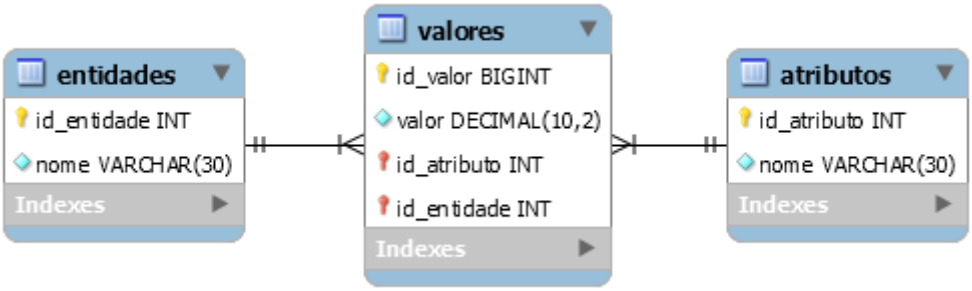



**Figura 7-64 – Diagrama ER simplificado do modelo EAV**

O tipo de dados frequentemente utilizado para armazenar leituras de sensores é o "decimal" na nomenclatura do SQL Server, que guarda valores numéricos na base decimal em vez da base binária (como é o caso do tipo de dados "float"), portanto não guarda uma aproximação binária de valores decimais. As colunas do tipo de dados decimal são configuradas pelo parâmetro "precision" que define o numero máximo de algarismos a armazenar e o parâmetro "scale" que define a posição da virgula no número. O espaço utilizado por cada valor decimal depende apenas da precisão (mais algarismos necessitam de mais espaço), podendo ocupar entre 5 a 17 bytes na base de dados[15]. No modelo EAV todas as leituras encontravam-se na mesma coluna de uma tabela, não sendo possível escolher precisões ou escalas diferentes para cada sensor, o que obrigava a que se utilizasse precisões e escalas elevadas para poder abranger todo o tipo de sensores. A Figura 7-65 mostra um exemplo de desperdício de espaço em que os espaços para algarismos a cinzento não vão ser utilizados.

| Unidade | Algarismos necessários | Algarismos utilizados | Parte inteira | | | | | | | | Parte decimal | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Volume ($m^3$) | 8 | 11 | | | 1 | 8 | 6 | 3 | 4 | 2 | 8 | 7 | |
| Volume (L) | 8 | 11 | 2 | 7 | 4 | 6 | 5 | 7 | 9 | 2 | | | |
| Temperatura (°C) | 5 | 11 | | | | | | | 2 | 5 | 7 | 4 | 5 |
| | | | Precision | | | | | | | | | | |
| | | | | | | | | | | | Scale | | |

**Figura 7-65 - Desperdício de espaço com sensores diferentes**

Neste exemplo o volume em metros cúbicos poderia ser guardado Cuma coluna decimal com os parametros [precision = 8] e [scale = 2]; o volume em litros com [precision = 8] e [scale = 0]; e a temperatura em graus Celsius poderia ser guardada com [precision = 5] e [scale = 3].

A solução encontrada foi criar uma tabela por contador quando o contador é adicionado ao sistema, existindo portanto uma tabela por contador. O nome da tabela é formado pela união da palavra "Meter" e o GUID do contador, sendo portanto único na base de dados. Existem colunas comuns a todas as tabelas de leituras e colunas que dependem do modelo do respetivo contador. A Figura 7-66 mostra a estrutura de uma tabela de leituras de um contador de água, as colunas "id" e "data_acquired_at" existem em todas as tabelas, a coluna "m3" foi criada com base no modelo do contador.

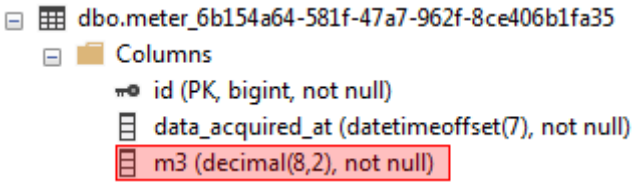



**Figura 7-66 – Estrutura de uma tabela de leituras**

A vantagem desta solução é que permite que o sistema possa armazenar leituras de vários tipos de contadores de água que podem reportar as leituras em litros, dezenas de litros, centenas de litros, milhares de litros ou outras unidades que requerem escalas e precisões específicas sem que seja necessário alterar o código do sistema. Também permite armazenar dados outros dispositivos, tais como estações meteorológicas, que possuem vários sensores com escalas e precisões distintas.

Inicialmente era criada uma tabela por cada modelo de contador, sendo que cada leitura possuía também uma coluna com o identificador do contador. Após um incidente em que um bug no sistema do *gateway* fez com que fossem inseridas milhões de leituras repetidas de um contador, tornando a consulta das leituras de qualquer contador do mesmo modelo demasiado lenta, foi decidido guardar as leituras de cada contador em tabelas separadas. Esta decisão teve ainda outra vantagem; a consulta leituras na base de dados requer menos uma restrição na cláusula WHERE para mostrar leituras de apenas um contador, como mostrado no exemplo abaixo:

```
WHERE meter_id = '6b154a64-581f-47a7-962f-8ce406b1fa35'
```

As tabelas de leituras de contadores têm uma chave primária (id) do tipo inteiro com 64 bits, que é sequencial e incrementada automaticamente.

Existe uma coluna que define o instante em que a leitura foi efetuada (data_acquired_at) do tipo data-hora, com precisão de 100 nanosegundos e informação sobre o fuso horário, permitindo que a data-hora possa ser interpretada corretamente fora da zona horária onde foi obtida. Inicialmente ponderei utilizar a coluna de data-hora como chave primária já que em funcionamento normal vai ser feita apenas uma leitura num determinado instante no tempo, mas por experiencia durante os testes cheguei à conclusão que não era boa ideia pois caso o contador reportasse a mesma data errada consecutivamente (p. ex.: 1970-01-01 00:00:00) as leituras seriam rejeitadas por restrição de chave primária. É preferível guardar leituras com datas erradas do que

rejeitá-las, pois caso o intervalo de amostra do contador se mantenha constante, as datas podem ser corrigidas posteriormente de acordo com a ordem de inserção.

Cada sensor do contador possui a sua respetiva coluna numérica, os sensores são definidos pelo modelo de contador escolhido quando o contador foi criado. O nome, a precisão e a escala destas colunas são definidos pelo utilizador ao criar o modelo do contador. Caso a coluna seja configurada com o tipo de dados decimal, a precisão pode ser no máximo 38 e a escala deve ser menor ou igual à precisão. A Figura 7-67 mostra uma tabela que guarda dados de uma estação meteorológica, demonstrando que é possível guardar informações de mais do que um sensor e que o sistema pode ser utilizado para guardar leituras de outros tipos de contadores, além de contadores de água.

| id | data_acquired_at | temp... | humidity | dewpoint | pressure | wind_vel... | wind_dir... | rain |
|---|---|---|---|---|---|---|---|---|
| 2119 | 2017-07-01 09:07... | 20.27 | 65.00 | 13.49 | 940.05 | 0.00 | 143.00 | 0.00 |
| 2120 | 2017-07-01 09:37... | 21.99 | 58.00 | 13.38 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2121 | 2017-07-01 09:52... | 22.88 | 57.00 | 14.00 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2122 | 2017-07-01 10:22... | 22.22 | 56.00 | 13.00 | 940.05 | 0.00 | 143.00 | 0.00 |
| 2123 | 2017-07-01 10:37... | 24.49 | 53.00 | 14.27 | 940.39 | 0.00 | 143.00 | 0.00 |

**Figura 7-67 – Leituras de uma estação meteorológica**

Ao eliminar um contador a sua tabela de leituras é também eliminada, mas antes de confirmar a eliminação é possível exportar todas as leituras do contador.

### 7.10.6. Stored procedures

O acesso à base de dados de leituras é feito sem recurso a Entity Framework. Quando é utilizada a EF, sabe-se no momento de compilação todos os nomes das tabelas, colunas e as suas relações, pois estes são derivados dos objetos de domínio. Em contraste não se sabe à partida quantos tipos de contadores vão ser adicionados à plataforma, nem quais os seus sensores.

Uma das vantagens da EF é que esta gera consultas parametrizadas possibilitando a utilização da cache do plano de execução de consultas, o que acelera bastante as consultas recorrentes das quais fazem parte as inserções de dados. Isto é importante porque o SQL Server reutiliza planos de execução em cache apenas se as consultas forem exatamente iguais, por exemplo alterar um valor na clausula WHERE ou adicionar um espaço em branco a uma consulta não parametrizada gera um *hash* diferente, impossibilitando a reutilização do plano de execução.

Como não se pode recorrer à EF tem-se como alternativa os *prepared statements*, que são modelos de consultas parametrizados. Os *prepared statements* possibilitam a reutilização do plano de execução das consultas, pois os valores são parametrizados e portanto se forem alterados não geram um plano de execução diferente. Neste caso os *prepared statements* teriam que ser construídos sempre que fosse necessário inserir novos dados, pois mais uma vez não se sabe à partida os nomes das tabelas ou colunas.

A solução escolhida foi utilizar *stored procedures*, estes não dependem da cache para acelerar consultas pois o plano de execução é compilado e armazenado permanente na base de dados, sendo reutilizado sempre que o *stored procedure* é executado. Quando um novo contador é adicionado é criado automaticamente um *stored procedure* que

permite a inserção de dados na respetiva tabela. A Figura 7-68 mostra um exemplo de instruções SQL geradas automaticamente pelo sistema, para criar um *stored procedure* de inserção de leituras de um contador de água.

```
CREATE PROCEDURE [dbo].[meter_6b154a64-581f-47a7-962f-8ce406b1fa35_insert]
  @data_acquired_at datetimeoffset(7),
  @m3 decimal(8, 2)
AS
BEGIN
  SET NOCOUNT ON
  INSERT INTO [dbo].[meter_6b154a64-581f-47a7-962f-8ce406b1fa35]
    ([data_acquired_at], [m3])
  VALUES
    (@data_acquired_at, @m3);
END;
```

**Figura 7-68 – Instruções SQL que criam um stored procedure de inserção de leituras**

A Figura 7-69 mostra a utilização do *stored procedure* de inserção de leituras criado na Figura 7-68 de modo a inserir novas leituras na base de dados.

```
public OperationResult InsertReadings(AssociativeReadings aReadings,
  IEnumerable<InsertSensor> sensors)
{
  ...
  string storedProcedureName = ReadingsDb.MeterDao
    .GetInsertSpName(aReadings.MeterId);
  transaction = connection
    .BeginTransaction(IsolationLevel.ReadCommitted,"InsertReadings");
  SqlCommand command = new SqlCommand(storedProcedureName, connection, transaction);
  command.CommandType = CommandType.StoredProcedure;
  foreach (AssociativeReading r in aReadings.Readings)
  {
    command.Parameters.Clear();
    command.Parameters.Add(new SqlParameter("@data_acquired_at", r. DataAcquiredAt));
    foreach (AssociativeSensor s in r.Sensors)
    {
      command.Parameters.Add(new SqlParameter("@" + s.Name, s.Value));
    }
    command.ExecuteNonQuery();
  }
  transaction.Commit();
  ...
}
```

**Figura 7-69 - Inserção de leituras utilizando um stored procedure**

Caso o *stored procedure* de inserção de leituras seja apagado ou restringida a permissão de acesso por lapso, é gerado um *prepared statement* automaticamente para que as leituras possam ser guardadas. A Figura 7-70 mostra como é gerado um *prepared statement* automaticamente para guardar leituras.

```
private OperationResult InsertReadingsPreparedStatement(
  AssociativeReadings aReadings, IEnumerable<InsertSensor> meterSensors,
  SqlConnection connection)
{
                                          ...
  transaction = connection.BeginTransaction(IsolationLevel.ReadCommitted,
    "InsertAssociativeReadingsPs");
  SqlCommand command = new SqlCommand(null, connection, transaction);
  command.CommandType = CommandType.Text;
  foreach (AssociativeReading r in aReadings.Readings)
  {
    StringBuilder sql = new StringBuilder(120);
    sql.Append("INSERT INTO ");
    sql.Append(ReadingsDb.MeterDao.GetReadingsTableName(aReadings.MeterId));
    sql.Append("\n  ([data_acquired_at]");
    // Columns
    foreach (AssociativeSensor s in r.Sensors) {
      sql.Append(", [" + s.Name + "]");
    }
    // Values
    sql.Append(")\nVALUES\n  (@data_acquired_at");
    foreach (AssociativeSensor s in r.Sensors)
    {
      sql.Append(", @" + s.Name);
    }
    sql.Append(");");
    command.CommandText = sql.ToString();
    // Parameters
    SqlParameter dateTimeParam = new SqlParameter("@data_acquired_at",
      SqlDbType.DateTimeOffset, 7);
    dateTimeParam.Value = r.DateTime;
    command.Parameters.Clear();
    command.Parameters.Add(dateTimeParam);
    foreach (AssociativeSensor s in r.Sensors)
    {
      InsertSensor sensor = meterSensors
        .FirstOrDefault(ss => ss.ColumnName.Equals(s.Name));
      SqlParameter sensorParam = new SqlParameter();
      sensorParam.ParameterName = "@" + s.Name;
      switch (sensor.DataType)
      {
        case SensorDataType.Decimal:
          sensorParam.SqlDbType = SqlDbType.Decimal;
          sensorParam.Precision = Convert
            .ToByte(sensor.IntegerDigits + sensor.DecimalPlaces);
          sensorParam.Scale = sensor.DecimalPlaces;
          sensorParam.Value = s.Value;
          break;
                                          ...
      }
      command.Parameters.Add(sensorParam);
    }
    command.Prepare();
    command.ExecuteNonQuery();
  }
  transaction.Commit();
                                          ...
}
```

**Figura 7-70 – Inserção de leituras com utilizando um prepared statement**

A geração de um *prepared statement* para cada inserção de leituras de um contador torna a operação de inserção mais lenta, pois a geração do *prepared statement* e

pesquisa pelo plano de execução na cache demora mais tempo do que simplesmente passar os valores para o *stored procedure* de inserção de leituras. Além disso quando a inserção de leituras é feita com recurso ao *prepared statements*, a reutilização do plano de consulta depende do tamanho da cache do SQL Server (quando a cache está cheia as os planos mais antigos são eliminados) e do numero de consultas distintas ou "ad hoc" (quando são feitas muitas consultas distintas a cache enche mais rápido). Esta redução de performance mantém-se até que o *stored procedure* seja restaurado com recurso às ferramentas de *troubleshooting* incluídas no sistema.

Ao eliminar um contador do sistema o *stored procedure* de inserção de leituras é também eliminado, pois a tabela de leituras do contador é eliminada e portanto o *stored procedure* não pode ser executado.

### 7.10.7. Agrupamento de leituras

O *website* de administração permite que os utilizadores definam qualquer intervalo de tempo para visualizar leituras de qualquer contador. Se, por exemplo, um contador com apenas um sensor efetuar uma leitura a cada minuto, durante uma semana terá efetuado um total de 10080 leituras (60*24*7), o carregamento deste número de leituras torna a apresentação dos gráficos lenta, além disso a visualização dos gráficos torna-se difícil pois devido à falta de espaço os pontos começam a ficar sobrepostos. A solução encontrada para este problema foi permitir agrupar os dados em intervalos pré-definidos (hora, dia, semana, mês, ano) com algumas funções simples (média, máximo, mínimo, etc.), tornando a visualização dos gráficos mais fácil e a sua apresentação quase instantânea. No entanto agrupar um número elevado de leituras no servidor pode tornar-se lento, por exemplo o valor médio de um sensor ao longo de uma década agrupado por ano gera um gráfico com apenas dez pontos, mas tal como no exemplo anterior se esse sensor efetuar uma leitura por minuto, ao fim de dez anos terá efetuado 5259600 leituras (60*24*365,25*10).

Para poder obter rapidamente leituras agrupadas da base de dados foi criado um sistema em que as leituras de todos os sensores de todos os contadores são agrupadas por todas as funções de agrupamento disponíveis, em todos os intervalos de tempo permitidos, em segundo plano. A Figura 7-71 mostra um esquema visual do agrupamento de leituras dos contadores.

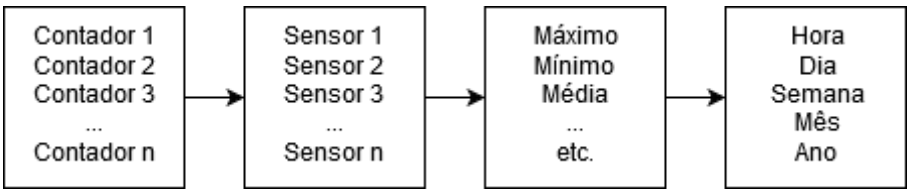



**Figura 7-71 – Esquema de agrupamento de leituras**

De modo a acelerar este processo periódico, quando um contador é adicionado, é também gerado automaticamente um SP que trata de agregar as leituras do novo contador; é necessário gerar um SP por contador devido ao facto de cada contador possuir a sua própria tabela de leituras. É gerado também um SP "agregador" que trata de executar os *stored procedures* de todos os contadores, este SP é atualizado sempre que é adicionado ou removido um contador. A ação de agregar dados dos contadores é desencadeada através de tarefas agendadas. São adicionadas tarefas agendadas ao sistema operativo que executam a cada hora, dia, semana, mês e ano, cuja função é

executar o SP agregador no respetivo intervalo de tempo. A Figura 7-72 mostra um diagrama de chamada dos *stored procedures* de agregação de leituras.

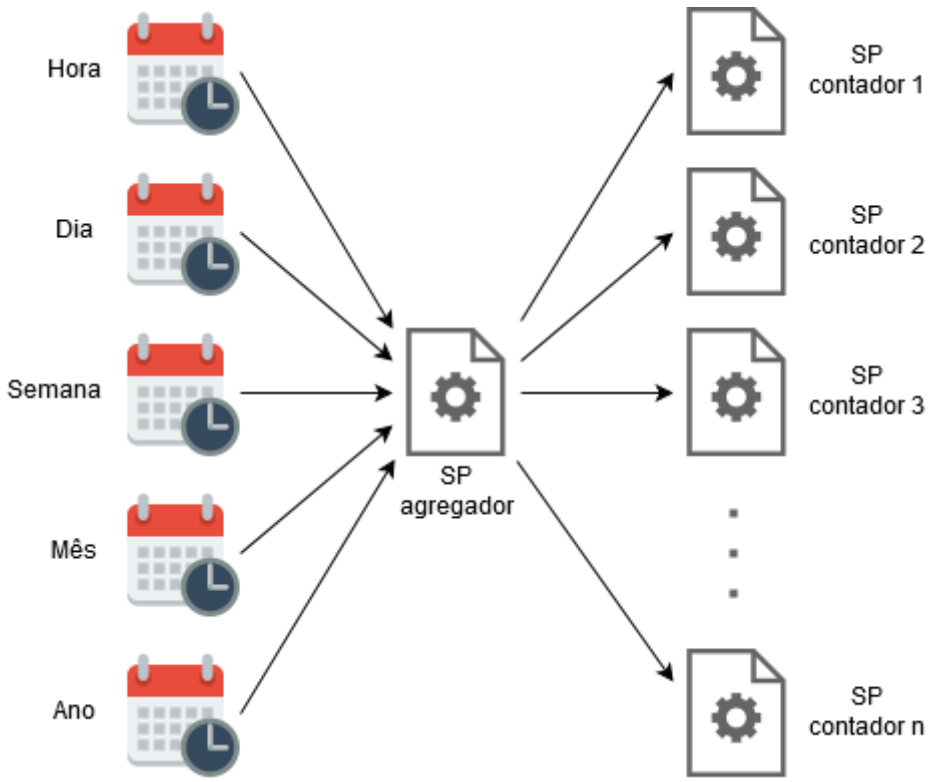



**Figura 7-72 - Diagrama de chamada dos stored procedures de agregação de leituras**

A Figura 7-73 mostra as tarefas que agregam leituras periodicamente.

| Name | Status | Triggers |
|---|---|---|
| AggregateDailyReadings | Ready | At 00:00 every day |
| AggregateHourlyReadings | Ready | At 00:00 every day - After triggered, repeat every 1 hour for a duration of 1 day. |
| AggregateMonthlyReadings | Ready | At 00:00 on day 1 of January, February, March, April, May, June, July, August, September, October, November, December, starting 01/01/2017 |
| AggregateWeeklyReadings | Ready | At 00:00 every Monday of every week, starting 01/01/2017 |
| AggregateYearlyReadings | Ready | At 00:00 on day 1 of January, starting 01/01/2017 |

**Figura 7-73 - Tarefas periódicas de agregação de leituras**

Cada tarefa executa o SP agregador, passando um valor numérico que sinaliza qual o intervalo de leituras a agrupar (p. ex.: hora, dia), o comando executado pela tarefa de agregação diária de leituras é o seguinte:

```
sqlcmd -S srv-01\MSSQLSERVER -d "MeterReadings" -Q "EXECUTE
[dbo].[readings_aggregator] @interval = 2"
```

Os comandos executados pelas outras tarefas são semelhantes é exceção do valor do parâmetro “interval”, por exemplo para agregar leituras mensalmente o valor passado é “4” em vez de “2”.

A partir do intervalo passado o SP de agregador gera uma data de início e uma data de fim que definem o intervalo tempo concreto em que as leituras devem ser agregadas. Caso, por exemplo, a tarefa de agregação horária de leituras seja executada, o SP agregador cria uma nova data com o ano, mês, dia e hora atuais, e definindo horas, minutos, segundos e milissegundos para zero, esta data passa a ser a data de fim do

intervalo, para criar a data de inicio do intervalo é subtraída uma hora à data de fim do intervalo. Os *stored procedures* de agregação de leituras dos contadores são então executados um a um, passando como parâmetros qual o intervalo de tempo a calcular, a data de início do intervalo e a data de fim do intervalo. A Figura 7-74 mostra as instruções SQL que criam o SP agregador, foram omitidas as partes que definem as datas de inicio e fim do intervalo "dia", "semana" e "mês" por serem muito semelhantes à definição do intervalo "hora" e "ano", além disso são mostrados apenas três chamadas aos *stored procedures* de agregação de leituras dos contadores por serem iguais.

```
CREATE PROCEDURE [dbo].[readings_aggregator]
  @interval int
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @date_time_now datetimeoffset(7) = SYSDATETIMEOFFSET();
  DECLARE @date_time_start datetimeoffset(7);
  DECLARE @date_time_end datetimeoffset(7);
  ...
  -- hour
  IF @interval = 1
  BEGIN
    SET @date_time_end = DATETIMEOFFSETFROMPARTS(YEAR(@date_time_now),
      MONTH(@date_time_now), DAY(@date_time_now),
      DATEPART(hour, @date_time_now), 0, 0, 0, 0, 0, 7);
    SET @date_time_start = DATEADD(hour, -1, @date_time_end);
  END;
  ...
  -- year
  IF @interval = 5
  BEGIN
    SET @date_time_end = DATETIMEOFFSETFROMPARTS (YEAR(@date_time_now),
      1, 1, 0, 0, 0, 0, 0, 0, 7);
    SET @date_time_start = DATEADD(year, -1, @date_time_end);
  END;
  -- execute each meter stored procedure
  EXEC [dbo].[meter_5a380883-a1dd-4cf5-98d3-1962ff49b6c6_aggregate]
    @interval, @date_time_start, @date_time_end;
  EXEC [dbo].[meter_1204953e-f48b-40b5-80b6-5125ec20b574_aggregate]
    @interval, @date_time_start, @date_time_end;
  EXEC [dbo].[meter_6b154a64-581f-47a7-962f-8ce406b1fa35_aggregate]
    @interval, @date_time_start, @date_time_end;
  ...
END;
```

**Figura 7-74 - Stored procedure agregador de leituras**

A Figura 7-75 mostra o SP de um dos contadores que é chamado pelo SP agregador, este SP trata de agregar as leituras do contador com o identificador "6b154a64-581f-47a7-962f-8ce406b1fa35".

```
CREATE PROCEDURE [dbo].[meter_6b154a64-581f-47a7-962f-8ce406b1fa35_aggregate]
  @interval int,
  @date_time_start datetimeoffset(7),
  @date_time_end datetimeoffset(7)
AS
BEGIN
  SET NOCOUNT ON;
  DECLARE @meter_id uniqueidentifier = '6b154a64-581f-47a7-962f-8ce406b1fa35';
  -- query
  WITH readings AS (
    SELECT *
    FROM [dbo].[meter_6b154a64-581f-47a7-962f-8ce406b1fa35]
    WHERE
      [data_acquired_at] >= @date_time_start AND
      [data_acquired_at] < @date_time_end
  )
  -- insert
  INSERT INTO [dbo].[aggregated_meter_readings]
    ([computed_at], [aggregate_function], [group_by_interval], [value],
    [meter_id], [meter_sensor])
  SELECT @date_time_start, 3, @interval, MAX([readings].[m3]), @meter_id, 'm3'
  FROM readings
  UNION ALL
  SELECT @date_time_start, 4, @interval, AVG([readings].[m3]), @meter_id, 'm3'
  FROM readings
  UNION ALL
  SELECT @date_time_start, 5, @interval, STDEVP([readings].[m3]), @meter_id, 'm3'
  FROM readings
  ...
END;
```

**Figura 7-75 - Stored procedure agregador de leituras de um contador**

Neste SP apenas são agregados dados de apenas um sensor (m3), pois trata-se de um contador de água que possui apenas um sensor. As leituras são agregadas por valor máximo, média e desvio padrão pois este contador comunica as leituras do consumo de água de forma incremental, ou seja, cada nova leitura reporta o consumo total desde que o contador foi instalado; não faz sentido agregar por exemplo por valor mínimo ou somar as leituras. A Figura 7-76 mostra um gráfico de exemplo de leituras de um contador de água com um sensor (m3) que reporta sempre o consumo total, onde se pode ver que, como exemplo, somar estes valores não faria sentido.

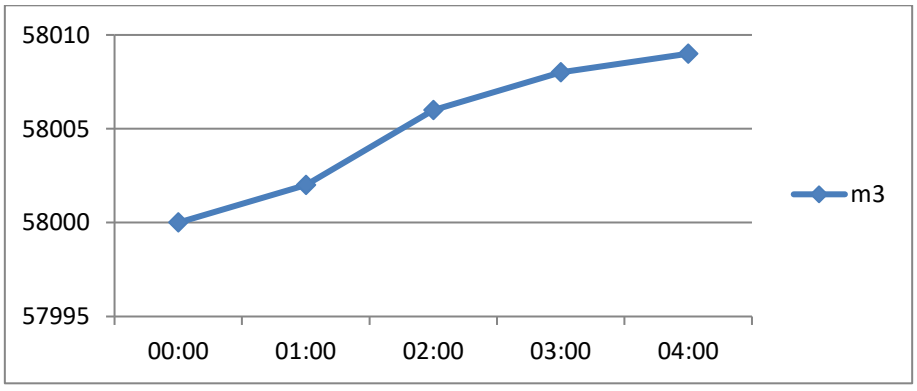



**Figura 7-76 - Exemplo de leituras de um sensor de volume de água**

Outros sensores reportam leituras com valores absolutos, portanto faz sentido agregar com outras funções de agrupamento. A Figura 7-77 mostra um gráfico de exemplo de leituras de um sensor de precipitação de uma estação meteorológica, como este sensor reporta a precipitação em um determinado intervalo, faz sentido agregar leituras pelo valor mínimo ou somar as leituras para obter respetivamente a precipitação mínima e acumulada num determinado intervalo de tempo.

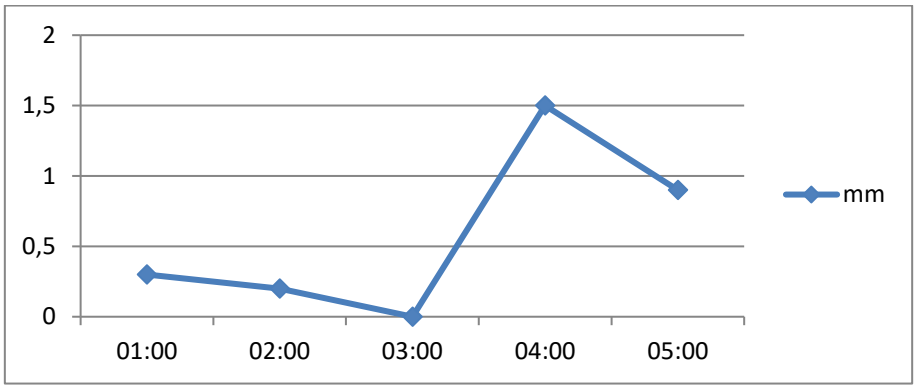



**Figura 7-77 - Exemplo de leituras de um sensor de precipitação**

O tipo de leituras que os sensores reportam pode ser configurado ao adicionar um novo modelo de contador.

# 8. Comunicação e protocolos

Este capítulo foca-se na comunicação entre os vários componentes de *software*, abordando também tecnologias de rede da camada física do modelo OSI.

## 8.1. LoRaWAN

LoraWAN é uma tecnologia de comunicação digital por rádio que que foi desenvolvida para a "Internet das coisas" e possui algumas características que a tornam ideal para este projeto, tais como baixo consumo de energia, longo alcance e mecanismos de segurança que protegem as transmissões. "LoRaWAN" refere-se à camada de acesso ao meio (segunda camada do modelo OSI), "LoRa" refere-se à camada física de rádio (primeira camada do modelo OSI). Foram consideradas várias tecnologias de comunicação para comunicar com os contadores. A tecnologia que se adequava melhor ao projeto era a LoRaWAN. De seguida são apresentadas algumas das tecnologias que foram consideradas para este projeto:

- Wi-Fi – Esta tecnologia tem a vantagem de ser amplamente utilizada, existindo portanto vários tipos módulos de rádio Wi-Fi prontos a serem incorporados em sistemas embebidos. Tornaria o projeto mais simples, pois os contadores poderiam comunicar diretamente com uma rede IP e, portanto, com a aplicação através de equipamento de rede comum. No entanto o alcance da rede é reduzido em ambientes urbanos e a comunicação consome demasiada energia para que o contador possa ser alimentado apenas por uma bateria. A banda de 2.4GHz está atualmente saturada em ambientes urbanos com vários pontos de acesso Wi-Fi, entre outras fontes de interferência, obrigando a utilizar a banda de 5GHz que tem um alcance mais reduzido do que a banda de 2.4GHz.
- Bluetooth – Tal como o Wi-Fi, esta tecnologia é amplamente utilizada e existem módulos prontos a serem incorporados em sistemas embebidos. Existe uma variante de baixo consumo desta tecnologia (BLE) que seria indicada para este projeto, combinando BLE com a tecnologia 6LoWPAN os contadores poderiam comunicar diretamente com a aplicação através de uma rede IP. No entanto utiliza a banda de 2.4GHz que como já foi referido não é ideal devido à interferência. O alcance é ainda mais reduzido do que o Wi-Fi, o que obriga a utilizar mais *gateways* e consequentemente aumenta o custo do projeto.
- Zigbee – Outra tecnologia considerada mas que não foi utilizada, esta tecnologia é semelhante à tecnologia Bluetooth mas funciona com uma topologia de rede em malha em que cada nó na rede também atua como repetidor, permitindo aumentar o alcance da rede. Esta tecnologia não foi selecionada porque tem um curto alcance entre nós e a rede é suportada pelos próprios nós (neste caso contadores de água), reduzindo a fiabilidade da rede e o tempo de vida útil das baterias.

- Rede móvel 3G/4G – O custo de ligar cada contador através de uma rede celular é demasiado elevado, pois não só o custo módulo do *modem* é relativamente elevado, como existem custos recorrentes associados à utilização de uma rede móvel de terceiros. Adicionalmente uma rede celular de terceiros é uma "caixa negra" para os seus utilizadores, no sentido em que todo o caminho percorrido pelos pacotes entre o terminal móvel e a Internet está dependente da operadora, não sendo possível que o cliente (neste caso empresa distribuidora de água) efetue qualquer tipo de *troubleshooting* ou reparações na rede caso ocorram problemas.

De realçar que a tecnologia Wi-Fi, Bluetooth e Zigbee suportam a topologia de rede em malha que pode aumentar o alcance da rede, utilizando nós (contadores de água) como repetidores. No entanto seria necessário adicionar vários nós fixos alimentados com energia da rede, com um sistema de alimentação ininterrupta e os nós fixos deveriam estar dentro do alcance de múltiplos contadores para aumentar a capacidade da rede, tornar a rede tolerante a falhas e aumentar o alcance da rede, o que aumentaria o custo do projeto.

A única tecnologia de comunicação, até à data, que verificava todos os requisitos foi a LoRaWAN. Tem longo alcance o que reduz significativamente o número de *gateways* necessário (2-5Km em ambiente urbano, 15Km em ambiente rural), tem um modo de consumo muito reduzido (Classe A), suporta um grande número de dispositivos por *gateway* (cerca 7800 por interface de rádio) e é possível ter controlo completo sobre a rede. Estas vantagens sobrepõem-se às suas limitações, tais como possuir uma taxa de transmissão baixa, não suportar o protocolo IP ou necessitar de vários servidores de suporte. A Figura 8-1 mostra um diagrama da arquitetura de rede LoRaWAN.

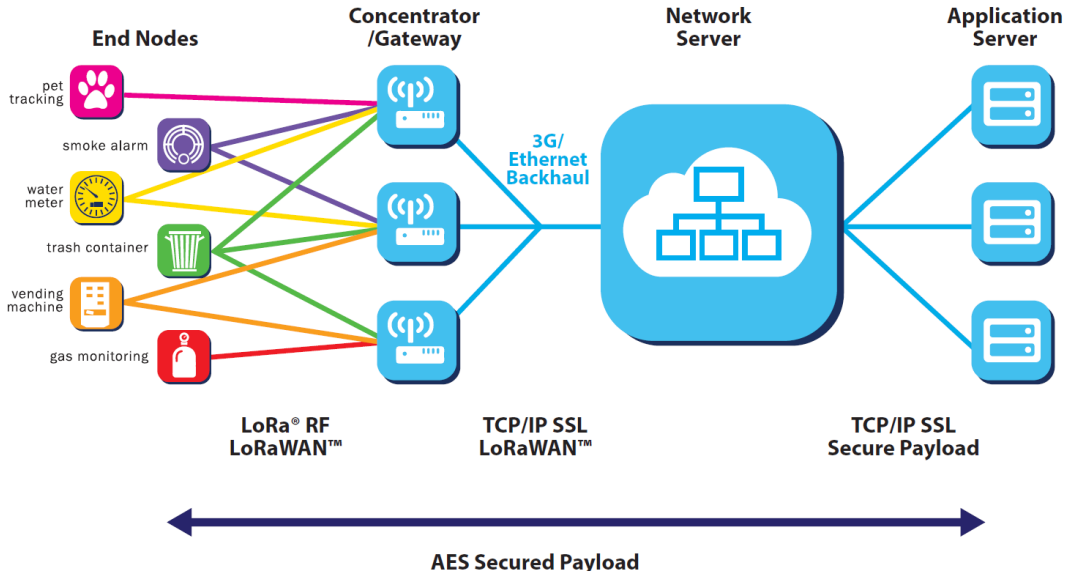



**Figura 8-1 - Arquitetura LoRaWAN (reproduzido de [16])**

## 8.2. *Website*

O protocolo HTTP é utilizado para servir o *website*, é utilizada a segunda versão deste protocolo, chamada HTTP/2, mas caso o *web browser* não suporte esta versão o *website* é servido pelo protocolo HTTP 1.1, perdendo os benefícios do protocolo HTTP/2, tais

como a multiplexagem da ligação TCP, que consiste em utilizar a mesma ligação para efetuar múltiplos pedidos em paralelo reduzindo a latência e a utilização de recursos, ou a compressão de cabeçalhos, que reduz o tamanho dos pedidos e respostas reduzindo a utilização de largura de banda da rede. O *website* possui ficheiros JavaScript e CSS comuns a todas as páginas. Adicionalmente algumas páginas possuem e JavaScript e CSS específico. Caso fosse utilizada a configuração por omissão do *website* o *web browser* efetuaria um pedido HTTP por cada ficheiro, o que não é muito eficiente pois cada pedido e resposta HTTP possui um cabeçalho que tem que ser processado e transmitido e, caso seja utilizado o protocolo HTTP 1.1 a transmissão de cada ficheiro ocupa uma ligação TCP entre o *web browser* e o servidor *web*, o que consome recursos em ambas as partes e pode levar a congestionamento dos pedidos do *web browser* caso seja excedido o número máximo de ligações. Para tornar a transferência de ficheiros JavaScript e CSS mais eficientes o tamanho destes ficheiros foi reduzido ao mínimo através da técnica de "minification", que remove espaços em branco dos ficheiros, remove quebras de linha, simplifica nome de funções, variáveis, etc. ao menor número possível de caracteres, entre outros. Adicionalmente, os ficheiros que são utilizados em conjunto pelo menos uma vez numa página *web* foram agrupados através da técnica de "bundling", esta técnica combina múltiplos ficheiros do mesmo tipo (p. ex.: JavaScript) num único ficheiro, o que reduz o número de pedidos HTTP que o *web browser* tem que fazer e que o servidor *web* tem que responder, já que é feito um pedido por cada grupo de ficheiros em vez de um pedido por cada ficheiro.

A técnica AJAX é utilizada em várias páginas para poder carregar dados, descarregar dados ou executar operações no servidor *web* de forma assíncrona, atualizando apenas uma parte da página *web* em vez de carregar uma nova página completa. Ao utilizar menos dados do que uma página *web* completa, esta técnica faz com que o conteúdo seja descarregado ou carregado mais rápido pelo *web browser*, reduz a carga no servidor e reduz a utilização da rede.

Segundo a especificação do protocolo HTTP apenas o cliente (p. ex.: *web browser*) pode enviar pedidos para o servidor e o servidor não pode enviar pedidos para o cliente. Uma das formas de contornar esta limitação é através de *polling*, em que o cliente envia pedidos periodicamente para o servidor, o servidor responde se tem novos dados e caso tenha novos dados envia-os para o cliente na resposta. Inicialmente foi utilizado *polling* através da técnica AJAX, de modo a poder atualizar partes das páginas *web* sem interromper o trabalho do utilizador. Mais tarde foi utilizada a biblioteca SignalR para este efeito, esta biblioteca permite receber e enviar dados para o *web browser* de forma assíncrona, permitindo que o servidor *web* possa chamar funções JavaScript e passar dados para estas no *web browser*, através de um mecanismo de RPC. Esta biblioteca utiliza o protocolo WebSocket para enviar e receber dados para o *web browser*, no entanto se o *web browser* ou os *proxies* HTTP intermediários não suportarem WebSockets são utilizadas técnicas alternativas, tais como "long polling" ou "forever frame", que são menos eficientes do que WebSockets mas são compatíveis com a maioria dos *web browsers* e *proxies* HTTP.

Para transmitir dados para o *web browser* foi utilizado o padrão *publish-subscribe* em SignalR, os *web browsers* subscrevem aos *hubs* necessários do SignalR e a partir desse momento quando forem publicadas novas mensagens pelo servidor *web* nesses *hubs*, todos os clientes que os subscreveram vão receber essas mensagens. Os *hubs* em SignalR formam uma API que permite que o servidor *web* possa chamar funções JavaScript nos *web browsers*. Como exemplo existe um *hub* que permite subscrever a

leituras em tempo real dos contadores, subscrevendo a um determinado contador nesse *hub*, quando novas leituras desse contador chegam ao *web service* no IIS são também encaminhadas para todos os *web browsers* que subscreveram às leituras em tempo real do contador. Alguns *hubs* funcionam em modo *broadcast*, em que todas as mensagens publicadas no *hub* são transmitidas para todos os *web browsers* que estão ligados ao servidor *web*, um exemplo de *hub* em modo *broadcast* é o *hub* de alertas, todos os *web browsers* ligados ao servidor *web* recebem os alertas em tempo real sem necessitarem de subscrever previamente ao *hub* de alertas.

## 8.3. *Web Services*

Utilizando um protocolo comum é possível transmitir dados através da Internet facilmente, pois a maioria das *firewalls* está configurada para permitir tráfego HTTP e HTTPS. Simultaneamente um *web service* fornece uma interface através do qual outras aplicações podem interagir com a aplicação, ou seja uma API. Sendo uma interface, a implementação do cliente e a implementação do servidor são independentes e o único requisito para que a comunicação é que haja um acordo no formato das mensagens enviadas e recebidas.

O *web service* no IIS pode ser servido através do protocolo HTTP/2 se for utilizado o IIS incluído com o Windows 10. O IIS requer o driver em modo *kernel* "http.sys". Este driver implementa um servidor *web* em *system space*, ou seja no espaço de memória virtual do *kernel* Windows, e serve de intermediário entre o IIS e as interfaces de rede por motivos de segurança e escalabilidade[17]. O protocolo HTTP/2 só é suportado a partir da versão do http.sys incluída no Windows 10. O *web service* no Node-RED também pode ser servido através do protocolo HTTP/2, pois o *web server* que é utilizado para implementar o *web service* (Express.js) tem suporte para HTTP/2.

A especificação do *web service* implementado no Node-RED encontra-se no Anexo C: Especificação do *web service* de *gateway*. A especificação do *web service* no IIS encontra-se no Anexo D: Especificação do *web service* no IIS.

## 8.4. Protocolo de rádio

O protocolo de rádio foi definido em conjunto com um dos colegas responsáveis pelo desenvolvimento do *hardware* e *software* dos contadores. Durante o desenvolvimento, à medida que os requisitos foram sendo desenvolvidos e que foi necessário enviar ou receber dados dos contadores, foram definidas as mensagens necessárias e adicionadas ao protocolo de rádio. Do lado dos servidores o componente de *software* que implementa o protocolo de rádio é o Node-RED, este é responsável por receber e descodificar mensagens dos contadores e codificar e enviar mensagens para os contadores, os contadores implementam o protocolo de rádio de forma inversa, descodificando mensagens codificadas pelo Node-RED e codificando mensagens antes de as enviar para o Node-RED.

O protocolo de rádio é binário, é utilizado o número mínimo de bits para transmitir as informações necessárias, pois a ligação de rádio tem uma largura de banda pequena (entre 125 e 250kHz) e consequentemente a taxa de transmissão é muito baixa (entre 250 e 11000bps). Na Europa, dependendo da faixa de frequências utilizada, um dispositivo pode ter *duty cycle* (percentagem de tempo de transmissão) entre 0,1% e

10% [18] o que se traduz numa velocidade média entre 0,25bps e 11bps em faixas que só permitem 0,1% de *duty cycle*. Outra limitação resultante dos requisitos é os contadores serem alimentados exclusivamente por bateria, sendo necessário reduzir o tempo de transmissão e receção de modo a aumentar a vida útil da bateria.

### 8.4.1. Classes de dispositivos

Existem três classes (A, B e C) de dispositivos na especificação LoRaWAN 1.0.1 [19]. Cada classe define um compromisso entre o consumo de energia e latência na comunicação. A classe "C" é a que consome mais energia. Dispositivos desta classe estão constantemente ativos à espera de novas mensagens do *gateway* LoRa, no entanto tem a latência mais baixa das três classes. A classe "B" é a mais equilibrada. Tem um consumo de energia reduzido pois dispositivos desta classe não estão constantemente ativos à espera de novas mensagens, pois existem intervalos periódicos sincronizados com o *gateway* LoRa em que o dispositivo pode receber novas mensagens, podendo estar em modo de baixo consumo de energia entre esses intervalos. A classe "A" é a que consome menos energia, os dispositivos só ficam à espera de novas mensagens depois de transmitirem uma mensagem, deste modo após receber uma mensagem do dispositivo o *gateway* LoRa tem um pequeno intervalo de tempo para enviar mensagens pendentes para o dispositivo. A classe escolhida para os dispositivos deste projeto (contadores) foi a classe "A" pois diminui o consumo de energia da bateria, de modo a que não é necessário utilizar uma bateria de grandes dimensões para obter a vida útil necessária.

### 8.4.2. Mensagens assíncronas

As limitações da classe "A" de dispositivos LoRa foram mitigadas com recurso a mensagens assíncronas. Contudo, é esperado que os pedidos de comandos e configurações enviados para os contadores sejam respondidos eventualmente no futuro. Dependendo da configuração do contador as respostas podem ser enviadas alguns minutos, horas ou dias após o envio dos pedidos, pois o pedido só é recebido pelo contador após o contador enviar uma mensagem. Como as leituras de consumo são as únicas mensagens enviadas periodicamente, pode-se esperar que as mensagens pendentes sejam entregues quando as leituras são enviadas. No entanto o envio de outro tipo de mensagens (p. ex.: alertas) também faz com que as mensagens pendentes sejam entregues.

O contador tem um tempo limitado de transmissão (*duty cycle*) para a rede LoRa. Se houver respostas na fila de transmissão do contador é possível que uma nova resposta seja colocada na fila de transmissão e tenha que aguardar até chegar a sua vez. A Figura 8-2 mostra um diagrama de sequência de um pedido para o contador e respetiva resposta. O diagrama está simplificado pois assume que todas as transmissões são bem-sucedidas e que não existem pedidos na fila de envio do *gateway* ou na fila de receção do contador. Adicionalmente não são mostradas todas as mensagens de confirmação de receção.

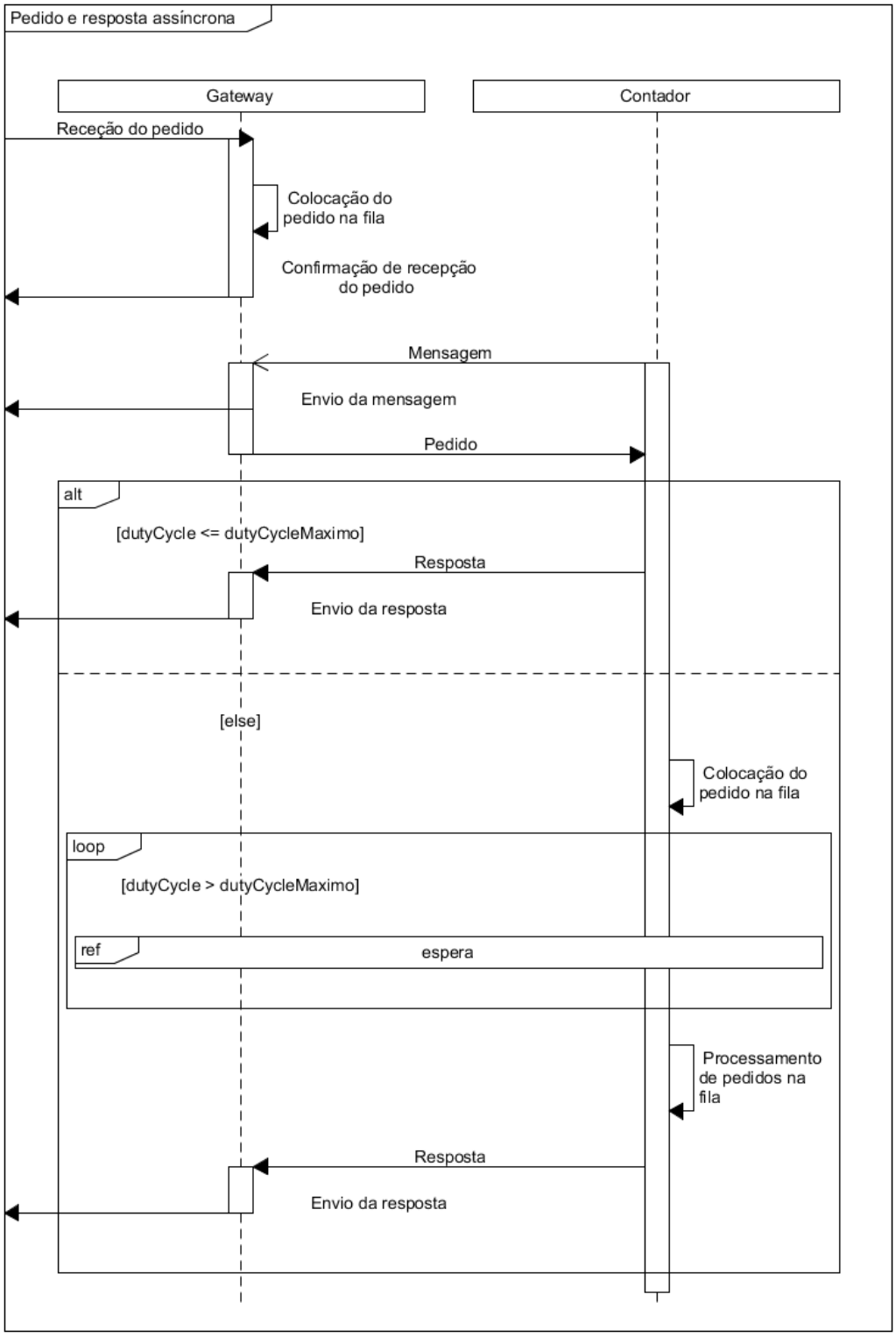



**Figura 8-2 - Pedido e resposta assíncrona do protocolo de rádio**

### 8.4.3. Mensagens recebidas dos contadores

Os contadores enviam mensagens periodicamente para reportar o consumo de água até ao momento e enviam mensagens durante o arranque para obter a data e hora atual, enviam mensagens de alerta, entre outras. Essas mensagens são transmitidas por rádio

para os *gateways* e são encaminhadas pelos servidores de suporte à rede LoRa até ao Node-RED onde são interpretadas e processadas. Dependendo do tipo de mensagem esta pode ser convertida num pedido HTTP para o *web service* no IIS (exemplo: leitura de consumo de água), ou pode ser gerada uma mensagem de resposta que é enviada para o contador (exemplo: contador solicita a data/hora atual e é devolvida uma mensagem com a data/hora atual).

As mensagens que são recebidas dos contadores são formadas por um cabeçalho seguido do *payload* (conteúdo). O cabeçalho é composto por oito bits e define o tipo de mensagem (quatro bits), o estado atual da válvula (dois bits) e possui espaço que atualmente não está a ser utilizado (dois bits). O estado da válvula de cada contador é guardado no Node-RED, sendo atualizado a cada nova mensagem recebida de cada contador. Deste modo é possível obter imediatamente o último estado da válvula reportado pelo contador sem necessidade de enviar uma mensagem de pedido e esperar pela resposta. A Figura 8-3 mostra a estrutura do cabeçalho das mensagens recebidas dos contadores, cada coluna representa um bit.

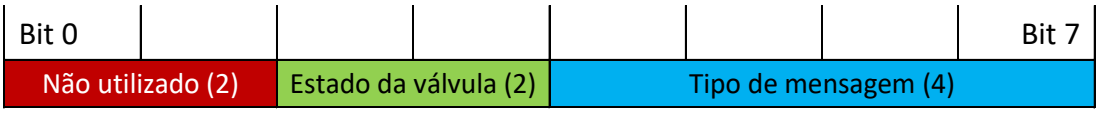



**Figura 8-3 - Cabeçalho das mensagens recebidas dos contadores**

O campo "Tipo de mensagem" é um número inteiro de quatro bits sem sinal, que define a estrutura do resto da mensagem. A partir deste valor podem-se definir até dezasseis mensagens distintas que vão ser processadas de forma diferente. Por exemplo, para enviar uma leitura de consumo de água o tipo de mensagem é definido para "1", deste modo quando a mensagem chega ao Node-RED este sabe que deve construir um número inteiro a partir dos próximos trinta bits e que este número representa o consumo de água em litros, de igual modo os próximos trinta e quatro bits representam a data e hora em que a leitura foi obtida. A Figura 8-4 mostra a estrutura de uma mensagem de leitura, as colunas representam bits e as linhas representam bytes.

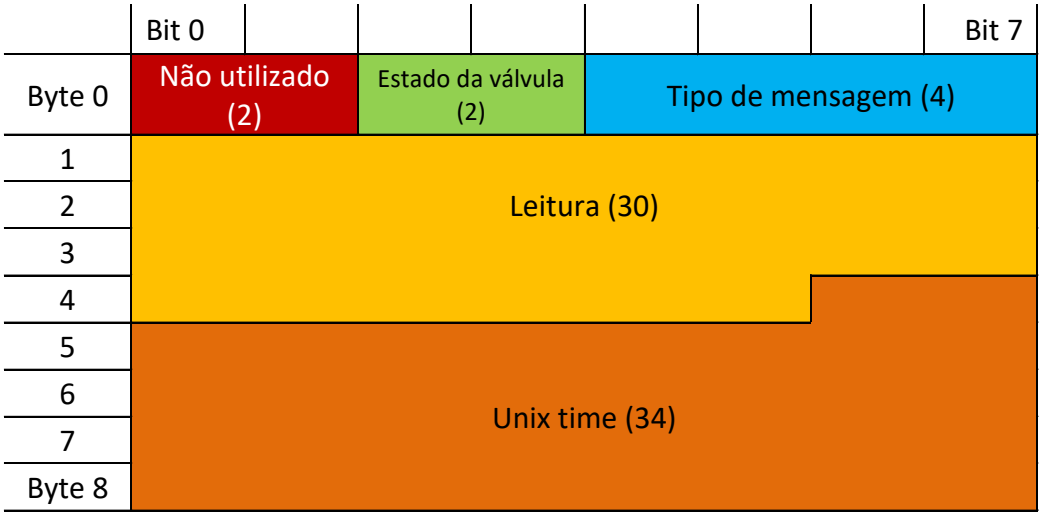



**Figura 8-4 - Estrutura de uma mensagem de leitura**

O conteúdo desta mensagem poderia ser, por exemplo, o seguinte conjunto de bytes; [0x01, 0x00, 0x00, 0xEA, 0x60, 0x58, 0xDE, 0xED, 0x80].

Analisando o primeiro byte (0x01 em hexadecimal ou 00000001 em binário) que representa o cabeçalho pode-se deduzir:

00000001 - O tipo da mensagem é “1” o que significa que é uma leitura de consumo de água.

00000001 - O estado da válvula é “0” o que significa que está fechada.

Estas deduções foram feitas com base na especificação do protocolo de rádio, a especificação contém informações sobre como identificar e descodificar os vários tipos de mensagens recebidas dos contadores e encontra-se no Anexo A: Protocolo de rádio. A Figura 8-5 mostra parte do código no Node-RED que extraí o tipo de mensagem e o estado da válvula do cabeçalho da mensagem, o código completo é mostrado na seção 7.7.3.

```
...
var msgType = msg.payload[0] & 0x0F;
...
var valveStatus = (msg.payload[0] & 0x30) >> 4;
...
```

**Figura 8-5 – Código que descodifica o tipo de mensagem e estado da válvula**

Como a mensagem é do tipo “leitura de consumo de água”, a partir dos próximos trinta bits pode-se gerar um número inteiro de trinta bits sem sinal que representa o consumo de água em litros:

0x00, 0x00, 0xEA, 0x60 = 00000000 00000000 11101010 01100000, convertendo para decimal deduz-se que o consumo de água lido foi de 15000 litros.

A partir dos restantes trinta e quatro bits pode-se gerar um número inteiro de trinta e quatro bits sem sinal que representa a diferença em segundos entre momento em que a leitura do consumo de água foi efetuada e a data 1970-01-01 00:00:00:

0x60, 0x58, 0xDE, 0xED, 0x80 = 01100000 01011000 11011110 11101101 10000000, somando este número de segundos à data 1970-01-01 00:00:00 pode-se deduzir que a leitura foi efetuada em 2017-04-01 às 00:00:00.

A Figura 8-6 mostra parte do código no Node-RED que extrai o consumo de água e a data/hora em que a leitura foi efetuada. O primeiro bloco de código extrai o consumo de água. O segundo bloco de código extrai a data/hora em que a leitura do consumo de água foi efetuada. O código completo é mostrado na secção 7.7.3.

```
...
var reading =
    (msg.payload[1] << 22) |
    (msg.payload[2] << 14) |
    (msg.payload[3] << 6)  |
    (msg.payload[4] >>> 2);
```

```
var unixTime10Msb =
    (msg.payload[4] & 0x03) << 8 |
    msg.payload[5];
var unixTime24Lsb =
    (msg.payload[6] << 16) |
    (msg.payload[7] << 8)  |
    msg.payload[8];
var leftShift24 = unixTime10Msb * 16777216;
var dataAcquiredAt = leftShift24 + unixTime24Lsb;
...
```

**Figura 8-6 – Código que descodifica as mensagens de leitura do consumo de água**

### 8.4.4. Mensagens enviadas para os contadores

A partir da interface do utilizador (website) é possível alterar configurações e enviar vários comandos para os contadores, adicionalmente algumas mensagens são enviadas automaticamente para os contadores como resposta a pedidos dos contadores, como é o caso do pedido de data/hora atual. As mensagens de configuração, controlo e reposta são transmitidas por rádio para os contadores, sendo estes descodificadas e executadas por estes.

As mensagens que são enviadas para os contadores são formadas por um cabeçalho seguido do *payload*. O cabeçalho é composto por oito bits e atualmente apenas quatro bits são utilizados para indicar qual o tipo de mensagem que se segue, de forma semelhante às mensagens recebidas dos contadores. A Figura 8-7 mostra a estrutura do cabeçalho das mensagens enviadas para os contadores, cada coluna representa um bit.

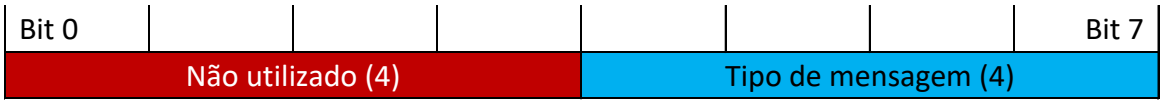



**Figura 8-7 - Cabeçalho das mensagens enviadas para os contadores**

A Figura 8-8 mostra um exemplo de mensagem enviada para os contadores, que neste caso é uma mensagem de controlo da válvula dos contadores, as colunas representam bits e as linhas representam bytes.

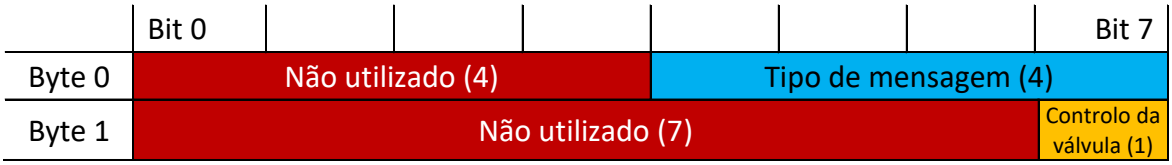



**Figura 8-8 - Estrutura de uma mensagem de controlo da válvula**

O conteúdo de uma mensagem de controlo da válvula poderia ser, por exemplo, o seguinte conjunto de bytes; [0x04, 0x01].

Analisando o primeiro byte (0x04 em hexadecimal ou 00000100 em binário) que representa o cabeçalho pode-se deduzir:

00000100 – O tipo de mensagem é “4” o que significa que é uma mensagem de controlo de válvula.

O contador irá utilizar esta informação para determinar como processar o resto da mensagem. A partir do próximo byte pode-se obter o bit que determina se o contador deve abrir ou fechar a válvula:

00000001 – O bit é “1” o que significa que o contador deve abrir a válvula.

A especificação do protocolo de rádio (Anexo A: Protocolo de rádio) mostra como codificar os vários tipos de mensagens enviadas para os contadores. A Figura 8-9 mostra parte do código no Node-RED que gera a mensagem de controlo da válvula que vai ser enviada para o contador.

```
...
var loRaMsg = {
    payload: Buffer.alloc(2),
    eui: msg.meterData.eui
};
loRaMsg.payload.writeUInt8(0x04, 0);
if(closeValve === true) {
    loRaMsg.payload.writeUInt8(0x00, 1);
    ...
} else if(closeValve === false) {
    loRaMsg.payload.writeUInt8(0x01, 1);
    ...
}
...
```

**Figura 8-9 – Bloco de código que gera a mensagem de controlo da válvula**

# 8.5. MQTT

Segundo o documento de especificação[20] o "MQTT é um protocolo publish-subscribe com arquitetura cliente-servidor que foi desenhado para ser leve, aberto, simples e fácil de implementar. Estas características fazem que seja ideal para várias situações, incluindo ambientes com limitações tais como comunicação máquina-para-máquina (M2M) e Internet das coisas (IoT), em que é necessário utilizar quantidades de código reduzidas e a largura de banda de rede é preciosa".

O LoRa App Server pode ser integrado com outros sistemas a partir de três APIs; gRPC, REST e MQTT. Escolhi a API MQTT para integrar o Node-RED com o LoRa App Server porque é um protocolo mais eficiente do que REST (HTTP) e porque não tive tempo para aprender a utilizar e testar o protocolo gRPC. Adicionalmente o protocolo MQTT é utilizado na comunicação entre a LoRa Gateway Bridge e o LoRa Server, pois este é o único protocolo suportado para este efeito.

## 8.5.1. Padrão publish-subscribe

O protocolo MQTT segue o padrão publish-subscribe. Depois de se ligar a um mediador MQTT, o cliente MQTT deve subscrever a tópicos para poder enviar e receber mensagens. Para subscrever a tópicos o cliente tem que enviar uma mensagem de subscrição para o mediador MQTT contendo uma lista de todos os tópicos que pretende subscrever. Após receber a mensagem de subscrição dos tópicos, o mediador MQTT responde ao cliente com uma mensagem que confirma, ou não, a subscrição a cada tópico solicitado. O cliente pode a qualquer altura enviar uma mensagem com uma lista de tópicos que pretende cancelar a subscrição, deste modo pode deixar de receber mensagens desses tópicos sem ter que terminar e iniciar uma nova ligação.

Quando um cliente envia (publica) uma mensagem para um determinado tópico, todos os clientes que subscreveram esse tópico recebem a mensagem assim que possível. Deste modo os clientes que pretendem receber mensagens de um determinado tópico não precisam de contactar periodicamente o mediador para receber novas mensagens, reduzindo a sobrecarga no mediador e na rede. O cliente pode definir um conjunto de opções que controlam como o mediador deve tratar a mensagem, essas opções são abordadas na seção 8.5.6.

Os clientes MQTT deste projeto são a LoRa Gateway Bridge, o LoRa Server, o LoRa App Server e o Node-RED, estes ligam-se a um dos mediadores MQTT para poderem criar, publicar e subscrever a tópicos em comum, de modo a comunicarem entre si através do protocolo MQTT.

### 8.5.2. Qualidade do serviço

A qualidade do serviço (QoS) define o balanço entre a performance e a consistência na entrega das mensagens. O nível de qualidade do serviço zero não garante a entrega da mensagem, o nível um garante que a mensagem é entregue pelo menos uma vez podendo ser entregue a mesma mensagem mais do que uma vez (duplicada) e o nível dois garante que a mensagem é entregue apenas uma vez. Foi utilizado o nível dois de qualidade do serviço, de modo a garantir que, por exemplo, as leituras recebidas dos contadores e os comandos enviados para os contadores cheguem ao destinatário e sejam entregues apenas uma vez.

### 8.5.3. Tópicos

Os tópicos do protocolo MQTT são grupos de comunicação que podem ser criados pelos clientes. Posteriormente outros clientes podem subscrever a esses tópicos, passando a fazer parte do grupo. Quando uma mensagem é enviada para um determinado tópico, todos os clientes que subscreveram esse tópico vão receber a mensagem. Os tópicos podem ter vários níveis separados por uma barra (/) e suportam caracteres "*wildcard*" que permitem que os clientes subscrevam a todos os tópicos que correspondam à regra. O caractere "+" representa um nível com qualquer nome e o caractere "#" representa qualquer número de níveis com qualquer nome.

Os tópicos são criados e utilizados pelo LoRa App Server para enviar e receber mensagens dos contadores. Foi criado um tópico para enviar mensagens e outro para receber por cada contador de cada aplicação. O LoRa App Server também publica mensagens de estado dos contadores, mensagens de erro dos contadores e mensagens de confirmação de receção de mensagens (*acknowledgement*), sendo possível configurar o "caminho" para esses tópicos. Os tópicos são criados e utilizados automaticamente para a comunicação entre a LoRa Gateway Bridge e o LoRa Server.

Como exemplo o tópico "application/1/node/00-04-a3-0b-00-1a-ea-5a/tx" é utilizado para transmitir mensagens para um dispositivo específico, que possui o endereço EUI "00-04-a3-0b-00-1a-ea-5a" e que faz parte da aplicação "1", mas se se pretendesse transmitir para todos os dispositivos da aplicação "1" poderia ser utilizado o tópico "application/1/node/+/tx". O tópico "application/#" poderia ser utilizado para receber todas as mensagens de todas as aplicações configuradas no LoRa App Server, incluindo aquelas que não são relacionadas com os contadores.

### 8.5.4. Sessões persistentes

Existem dois tipos de sessões que um cliente pode estabelecer com um mediador MQTT; persistentes e não persistentes. As sessões não persistentes funcionam como uma transmissão de TV em direto; se o cliente não estiver ligado no momento em que as mensagens são enviadas, vai perder essas mensagens. As sessões persistentes funcionam como um servidor de *email*; se o cliente não estiver ligado no momento em que as mensagens são enviadas, estas são guardadas no mediador MQTT, na sessão do

cliente e da próxima vez que o cliente se ligar o mediador envia todas as mensagens que o cliente ainda não recebeu, sendo apenas eliminadas da sessão do cliente quando o cliente confirmar a receção. Esta funcionalidade está dependente do nível de qualidade do serviço, as mensagens com nível de qualidade do serviço zero não são armazenadas na sessão, apenas as mensagens com o nível de qualidade do serviço um ou dois são armazenadas, pois estes níveis garantem a entrega das mensagens.

### 8.5.5. Encapsulamento do protocolo de rádio

O protocolo MQTT é um protocolo de transporte de dados binários, independente do tipo ou formato dos dados. É portanto possível enviar dados no formato JSON, este é o formato utilizado pela API MQTT do LoRa App Server. As mensagens do protocolo de rádio são enviadas do Node-RED para o LoRa App Server através do protocolo MQTT, sendo previamente codificadas em Base64, encapsuladas numa mensagem para o LoRa App Server, que por sua vez é convertida no formato JSON e encapsulada noutra mensagem para o servidor MQTT, que também é convertida no formato JSON e por fim é enviada. A Figura 8-10 mostra o diagrama do processo de encapsulamento das mensagens enviadas para os contadores.

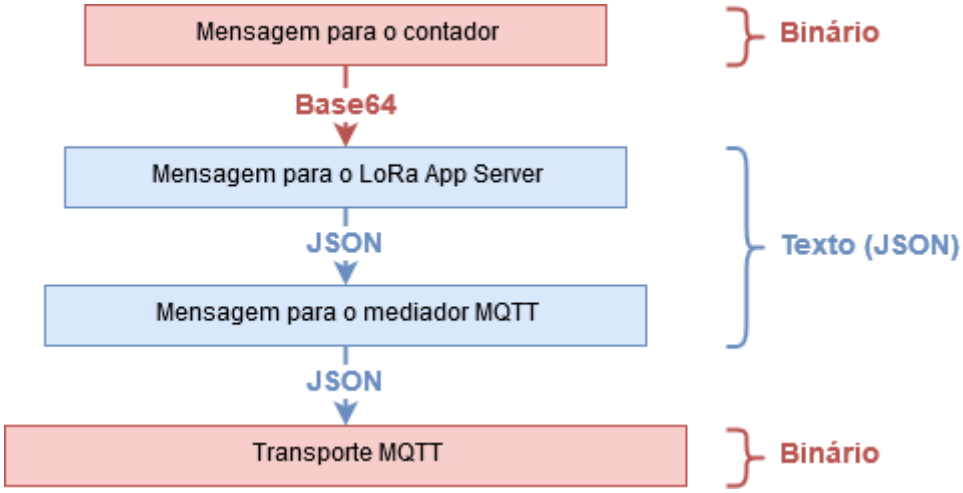



**Figura 8-10 - Processo de encapsulamento de mensagens para os contadores**

Este processo pode ser ineficiente mas é necessário para que o mediador MQTT receba e publique a mensagem no tópico de envio de mensagens para o contador e que o LoRa App Server encaminhe a mensagem corretamente. São utilizados apenas as opções mínimas necessárias nos objetos JSON, de modo a reduzir a carga na rede.

### 8.5.6. Formato das mensagens

Para que o Node-RED possa comunicar com o LoRa App Server, as mensagens devem seguir um formato específico. As mensagens enviadas para o LoRa App Server são compostas por uma referência, definição da necessidade de confirmar a receção da mensagem, porto LoRa e os dados a enviar.

A propriedade “reference” serve para identificar a mensagem caso a confirmação de receção da mensagem seja solicitada. Neste campo são enviados oito caracteres alfanuméricos aleatórios de modo a que a probabilidade de existirem duas mensagens por confirmar do mesmo dispositivo, com a mesma referência, na mesma aplicação seja muito reduzida.

A propriedade “confirmed” define se o dispositivo LoRa deve confirmar a receção das mensagens recebidas. As confirmações podem ser recebidas e encaminhadas pelo Node-RED, mas atualmente só são utilizadas para certificar que os comandos são recebidos pelos contadores; quando não há confirmação de uma mensagem essa mensagem é retransmitida.

A propriedade “fPort” define o porto LoRa, semelhante a um porto TCP ou UDP. É um valor inteiro sem sinal de oito bits segundo a especificação LoRa e portanto pode tomar valores de “0” a “255”. É utilizado o número da aplicação como porto, desde que esse número esteja entre “1” e “223” (inclusive), caso contrário utiliza-se o porto “1”. O porto “0” é utilizado para enviar comandos MAC, o porto “224” é utilizado para testes da camada MAC, os portos “225” a “255” estão reservados para extensões futuras do protocolo.

A propriedade “data” contém os dados a enviar para os dispositivos, estes dados são encapsulados num objeto do tipo “Buffer” do Node.js de modo a facilitar a codificação para Base64, para serem enviados no formato JSON. A Figura 8-11 mostra um exemplo de mensagem no formato JSON enviada do Node-RED para o LoRa App Server.

```
{
  "reference": "4t2wpt16",
  "confirmed": true,
  "fPort": 1,
  "data": "BAA="
}
```

**Figura 8-11 - Mensagem enviada do Node-RED para o LoRa App Server**

As mensagens enviadas do Node-RED para o LoRa App Server são encapsuladas em mensagens MQTT. As mensagens MQTT são compostas por um tópico, definição do nível de qualidade do serviço, definição da retenção da mensagem e os dados a enviar.

A propriedade “topic” define o tópico MQTT onde a mensagem vai ser publicada, existe um tópico por dispositivo e por aplicação para enviar mensagens para os dispositivos LoRa.

A propriedade “qos” define o nível de qualidade do serviço MQTT, neste projeto é utilizado o nível dois para garantir que mensagens enviadas e recebidas dos contadores são entregues apenas uma vez.

A propriedade “retain” faz com que seja guardada uma cópia da última mensagem do tópico especificado, quando é estabelecida uma ligação ao mediador MQTT essa mensagem é transmitida automaticamente. Esta opção não é ativada neste projeto, pois se a ligação entre o LoRa App Server e o mediador MQTT ou o Node-RED e mediador MQTT fosse interrompida e reestabelecida, a ultima mensagem enviada em cada tópico seria enviada de novo, este não é o comportamento pretendido pois as últimas mensagens enviadas e recebidas dos contadores seriam repetidas e a rede seria utilizada desnecessariamente.

A propriedade “payload” contém os dados a publicar no mediador MQTT, de modo a serem recebidos pelo LoRa App Server, estas mensagens são serializadas no formato JSON, de acordo com a API MQTT do LoRa App Server.

A Figura 8-12 mostra uma mensagem de exemplo enviada do Node-RED para o servidor MQTT, esta mensagem é um comando para o contador com EUI “00-04-a3-0b-00-1a-ea-5a” abrir a válvula de fluxo de água.


```
{
  "topic": "application/1/node/00-04-a3-0b-00-1a-ea-5a/tx",
  "qos": 2,
  "retain": false,
  "payload":
"{\"reference\":\"4t2wpt16\",\"confirmed\":true,\"fPort\":1,\"data\":\"BAA=\"}"
}
```


**Figura 8-12 - Mensagem enviada o Node-RED para o mediador MQTT**

Quando uma mensagem de um dispositivo LoRa é recebida no LoRa App Server este publica o conteúdo da mensagem no tópico MQTT de receção do respetivo dispositivo. As mensagens dos dispositivos publicadas pelo LoRa App Server são compostas pelo identificador e nome da aplicação LoRa, o EUI e o nome do dispositivo, o estado da bateria do dispositivo, o endereço MAC e nome dos *gateways* que receberam a mensagem, posição atual dos *gateways*, data e hora em que a mensagem foi recebida pelos *gateways*, informações sobre o modo de transmissão da mensagem, vários parâmetros de diagnóstico da transmissão e a mensagem recebida do dispositivo LoRa.

A maior parte dos valores das propriedades da mensagem MQTT, tais como o nome da aplicação ou nome do dispositivo, são obtidos a partir da configuração do LoRa App Server, outros valores, tais como o RSSI, são obtidos a partir das condições de transmissão entre os dispositivos LoRa e o *gateway*.

A propriedade “deviceStatusBattery” guarda um valor inteiro sem sinal de oito bits que indica o estado da bateria, esta propriedade suporta 254 valores, entre “1” e “254” (o valor “0” representa alimentação por fonte externa, o valor “255” representa um erro ao obter o estado da bateria), o que não é suficiente para transmitir a tensão da bateria em milivolts, como especificado nos requisitos, portanto foi definida uma mensagem para esse efeito.

A propriedade “data” contém os dados recebidos dos dispositivos LoRa codificados em Base64, sendo necessário descodifica-los para binário antes de tentar descodificar o protocolo de rádio.

A Figura 8-13 mostra uma mensagem de exemplo enviada do LoRa App Server para o Node-RED, pelo protocolo MQTT, esta mensagem contém o estado da bateria do contador com EUI “00-04-a3-0b-00-1a-ea-5a”, em que a tensão é 3700 milivolts, a capacidade total 7400 miliwatts-hora e a capacidade atual 4800 milliwatts-hora.

```
{
  "applicationID": "1",
  "applicationName": "water-meters",
  "deviceName": "water-meter-1",
  "devEUI": "0004a30b001aea5a",
  "deviceStatusBattery": 0,
  "deviceStatusMargin": 8,
  "rxInfo": [
    {
      "mac": "1000000000000001",
      "name": "test-gateway",
      "latitude": 0,
      "longitude": 0,
      "altitude": 0,
      "time": "2017-07-03T15:32:39.432334478Z",
      "rssi": -87,
      "loRaSNR": 9
    }
  ],
  "txInfo": {
    "frequency": 868300000,
    "dataRate": {
      "modulation": "LORA",
      "bandwidth": 250,
      "spreadFactor": 7
    },
      "adr": false,
      "codeRate": "4/5"
  },
  "fCnt": 594,
  "fPort": 1,
  "data": " Ag50ASwAHOg="
}
```


**Figura 8-13 – Exemplo de mensagem do Lora App Server recebida pelo Node-RED**

# 9. Funcionalidades do sistema

Neste capítulo serão apresentadas funcionalidades principais do sistema, ou seja o que o que os utilizadores podem fazer com o sistema. Algumas dessas funcionalidades implementam requisitos do cliente, outras, embora não sejam requisitos, foram implementadas de modo a facilitar a administração do sistema e a resolução de problemas. A Tabela 9-1 mostra as funcionalidades principais do sistema, especificando quais as funcionalidades que foram requeridas pelo cliente.

| **Funcionalidade** | **Requisito do cliente** |
|---|---|
| Flexibilidade | |
| Adquirir, processar e armazenar leituras de outros contadores | Não |
| Controlar outros tipos de contadores | Não |
| Alertas personalizados | Não |
| Configurar *gateways* remotamente | Não |
| Configurar contadores remotamente | Sim |
| *Dashboard* configurável | Não |
| Registo | |
| Histórico de leituras | Sim |
| Notificações | Sim |
| Registo do sistema | Não |
| Resolução de problemas | |
| Deteção e correção de problemas | Não |
| Mapeamento do sinal LoRa | Não |
| Leituras em tempo real | Não |
| Notificações em tempo real | Não |
| Escalabilidade | |
| Multitenancy | Não |
| Rede LoRa | Sim |
| Servidores | Não |
| Exportação de dados | |
| Exportar leituras | Sim |
| Exportar dados de mapeamento de sinal | Não |
| Exportar outros dados | Não |

**Tabela 9-1 - Funcionalidades requeridas pelo cliente**

# 9.1. Flexibilidade

O sistema pode ser facilmente personalizado pelos utilizadores de modo a solucionar novos problemas, pois foi desenvolvido a pensar sempre em casos de uso genéricos de modo a poder adaptar-se a novas situações.

## 9.1.1. Adquirir, processar e armazenar leituras de outros contadores

A aquisição, processamento e armazenamento de leituras não está limitada a contadores de água. O sistema suporta qualquer tipo de contador que produza dados numéricos temporais. Isto é possível pois o sistema permite adicionar modelos de contador que definem quais os sensores que um determinado contador possui, os nomes, tipos de dados e grandezas físicas que esses sensores medem e também configurações dos gráficos que podem ser gerados através das leituras adquiridas pelos sensores.

O SGBD SQL Server suporta no máximo 4096 colunas por tabela[21], como cada contador possui uma tabela na base de dados de leituras e as leituras de cada sensor são guardadas em colunas distintas nas respetivas tabelas de leituras, o sistema atual suporta no máximo 4096 sensores por contador, outros SGBDs, tais como PostgreSQL, Oracle Database ou MySQL, possuem limites semelhantes ou inferiores.

Deste modo é possível suportar contadores desenvolvidos futuramente sem alterar o código fonte do sistema, quer estes sejam novas versões incrementais de contadores existentes ou novos contadores desenvolvidos de raiz. Para suportar novos contadores basta adicionar o respetivo modelo de contador e basear os novos contadores nesse modelo. Na seção 7.10.5 é detalhada a forma como os dados dos contadores são armazenados.

A Figura 9-1 mostra os detalhes de um modelo de contadores, neste exemplo é mostrado um modelo de uma estação meteorológica que possui sete sensores. Na figura também pode-se ver as configurações dos sensores, um resumo dos comandos personalizados do modelo, um resumo dos alertas personalizados associados ao modelo e os contadores que têm como base este modelo de contador.

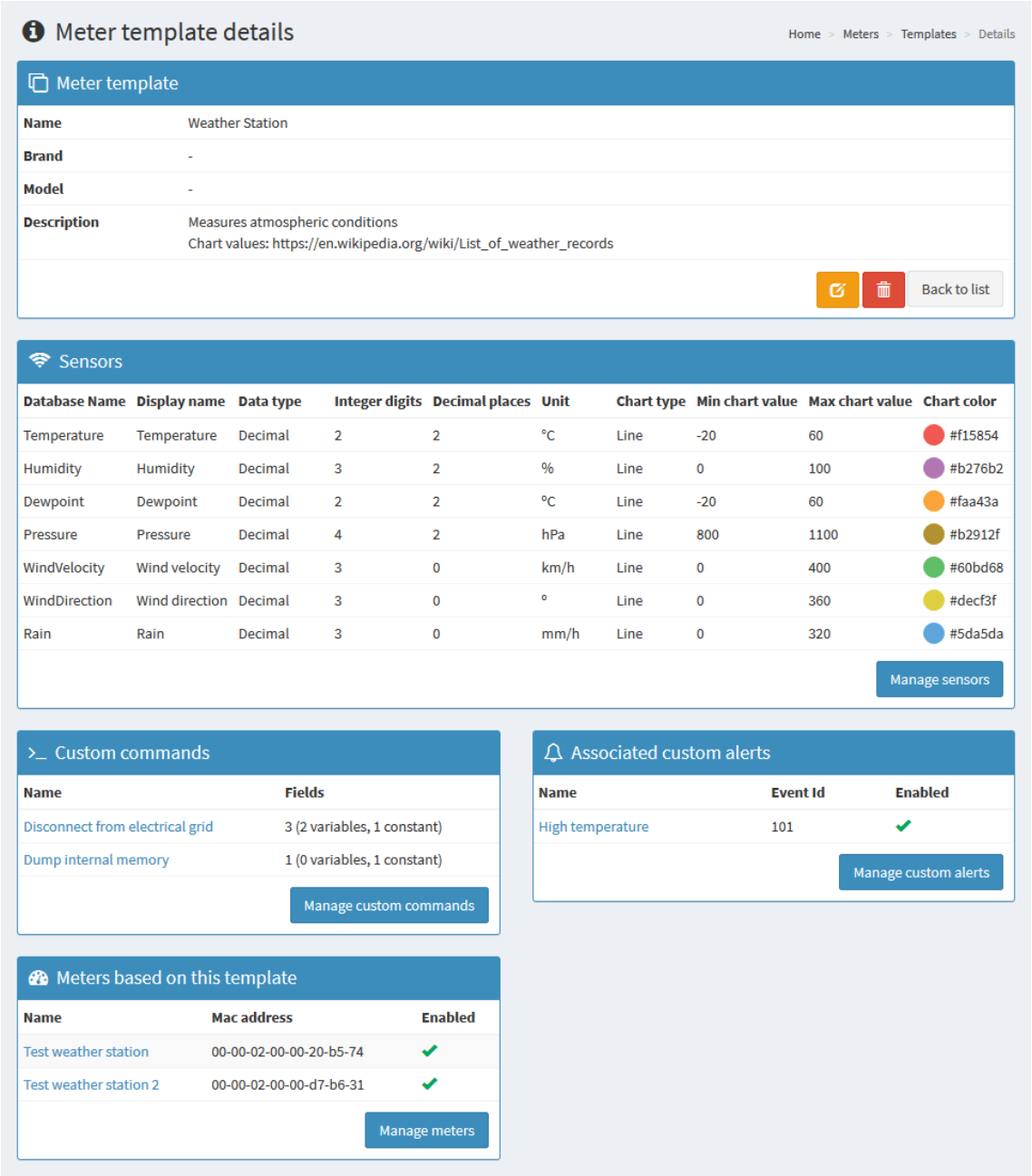



**Figura 9-1 – Detalhes de um modelo de dispositivo**

## 9.1.2. Controlar outros tipos de contadores

É possível adicionar comandos a modelos de contador, permitindo enviar novos comandos para os contadores. No *website* de administração é possível definir os nomes, tipos de dados e valores de constantes e variáveis que compõem um comando personalizado.

Como exemplo foi criado um modelo de contador para hipotéticos contadores de água industriais. Estes contadores seriam alimentados normalmente pela rede elétrica, mas possuiriam várias baterias que poderiam ser utilizadas para alimentação em caso de falhas de energia. Foi adicionado um comando personalizado ao modelo de contador industrial que permite desligar remotamente contadores da rede elétrica, o que poderia ser útil durante trovoadas e outros períodos de instabilidade de rede elétrica.

A Figura 9-2 mostra os detalhes deste comando personalizado. São mostrados os campos que compõem o comando numa tabela e a estrutura da mensagem. O campo "Not used" não é utilizado, tal como o nome indica, servindo apenas para reservar espaço que não é utilizado na mensagem. O campo "Message type" define o tipo de mensagem, da mesma forma que as mensagens para os contadores de água atuais (ver seção 8.4.4). O campo "Duration (minutes)" define por quantos minutos o contador deve permanecer desligado da rede elétrica, ligando-se novamente à rede elétrica no fim desse período. O campo "Use backup power" define se o contador deve utilizar reservas de energia enquanto estiver desligado da rede elétrica, de modo a permanecer funcional durante esse período. O campo "Backup battery number" define qual a bateria a utilizar para alimentar o contador,  se for ativada a opção de utilizar reservas de energia.

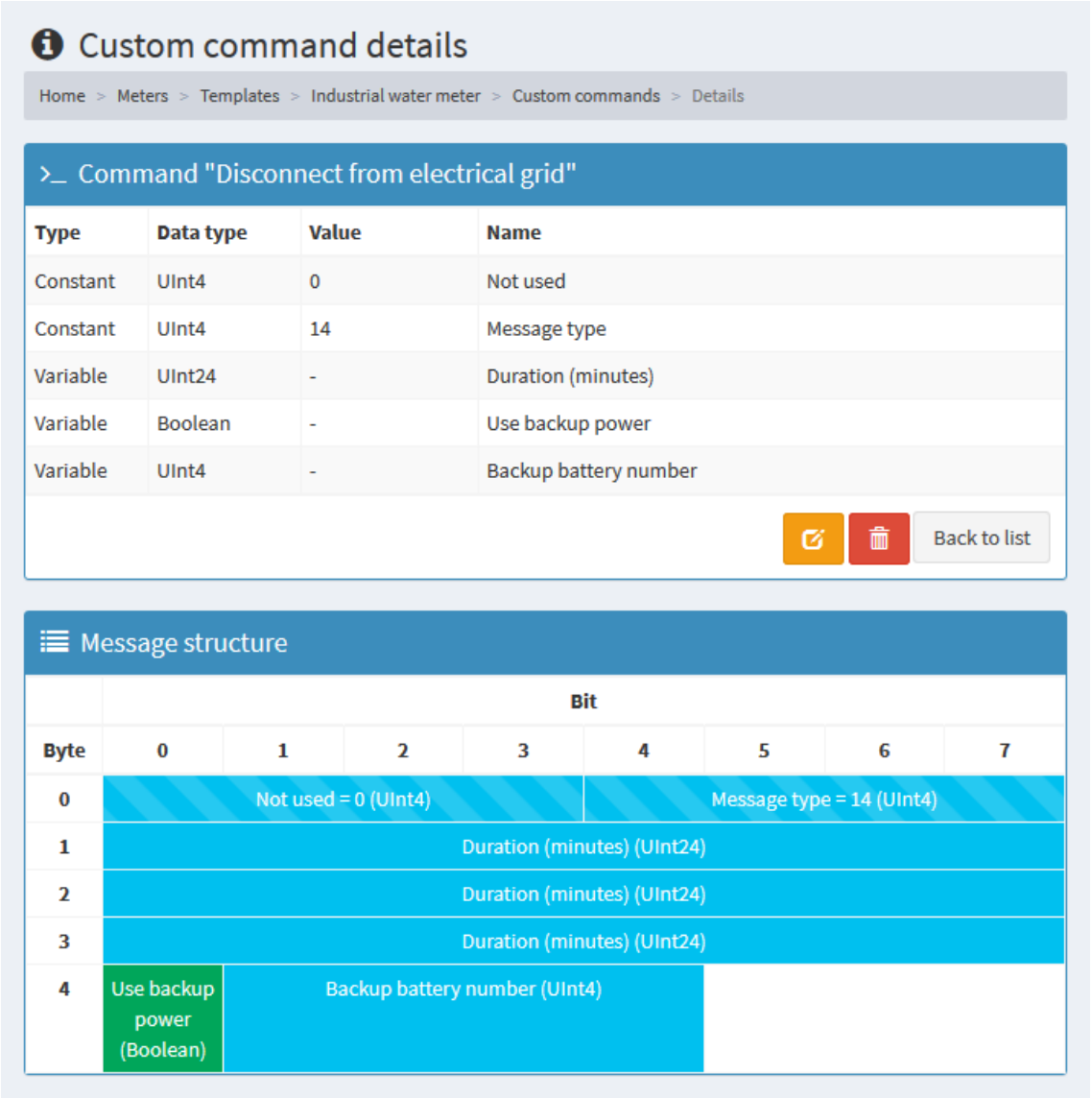



**Figura 9-2 - Comando personalizado que desliga o contador da rede elétrica**

Todos os contadores que sejam baseados neste modelo de contador passam a ter a opção de enviar este comando a partir do *website* de administração. A Figura 9-3 mostra a página de controlo de um contador de água industrial de exemplo. Na secção de comandos personalizados é possível selecionar o comando que foi adicionado anteriormente ao modelo de contador, dependendo do tipo de dados que foi configurado

são adicionados controlos apropriados na página para introduzir os valores das variáveis. Neste caso os campos "Duration (minutes)" e "Backup battery number" podem ser introduzidos em caixas que aceitam valores numéricos e o campo "Use backup power" pode ser ativado ou desativado numa caixa de verificação. De notar que não são solicitados valores para as constantes, pois os valores das constantes são definidos nas configurações do comando personalizado e só podem ser alterados na página de edição de comandos personalizados.

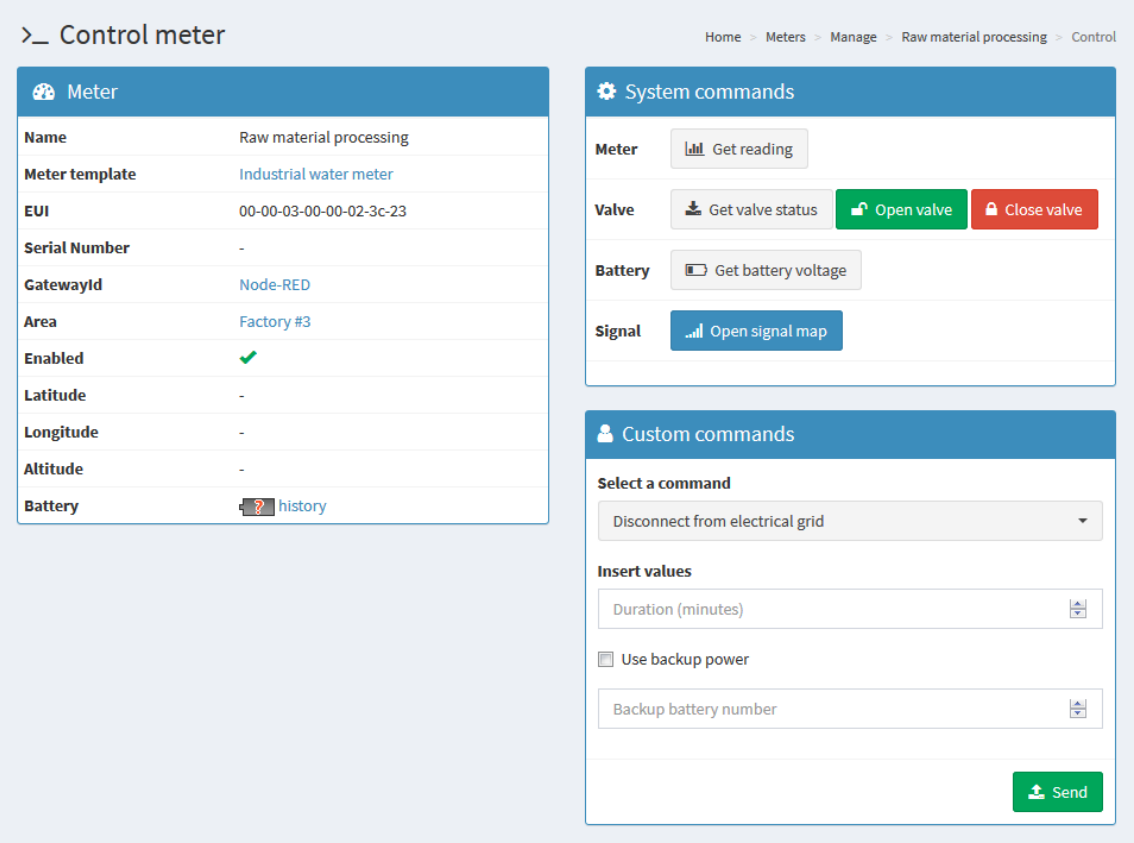



**Figura 9-3 - Comando personalizado que desliga o contador da rede elétrica**

### 9.1.3. Alertas personalizados

Além dos alertas pré-definidos para o caso de uso dos contadores, podem ser adicionados novos alertas personalizados. Estes novos alertas podem ser associados a modelos de contador e, a partir desse momento, os contadores baseados nesses modelos de contador podem emitir os novos alertas. Como exemplo, se no futuro for decidido adicionar um sensor de temperatura aos contadores de água para detetar se a temperatura ambiente pode afetar o tempo de vida útil da bateria (temperatura demasiado baixa ou elevada), e se o microcontrolador dos contadores for programado para enviar um alerta caso a temperatura ambiente não seja favorável, basta adicionar um novo alerta personalizado no sistema e associá-lo ao modelo de contador. A Figura 9-4 mostra a lista de alertas suportados pelo sistema. No final da lista existe um alerta personalizado de temperatura elevada.

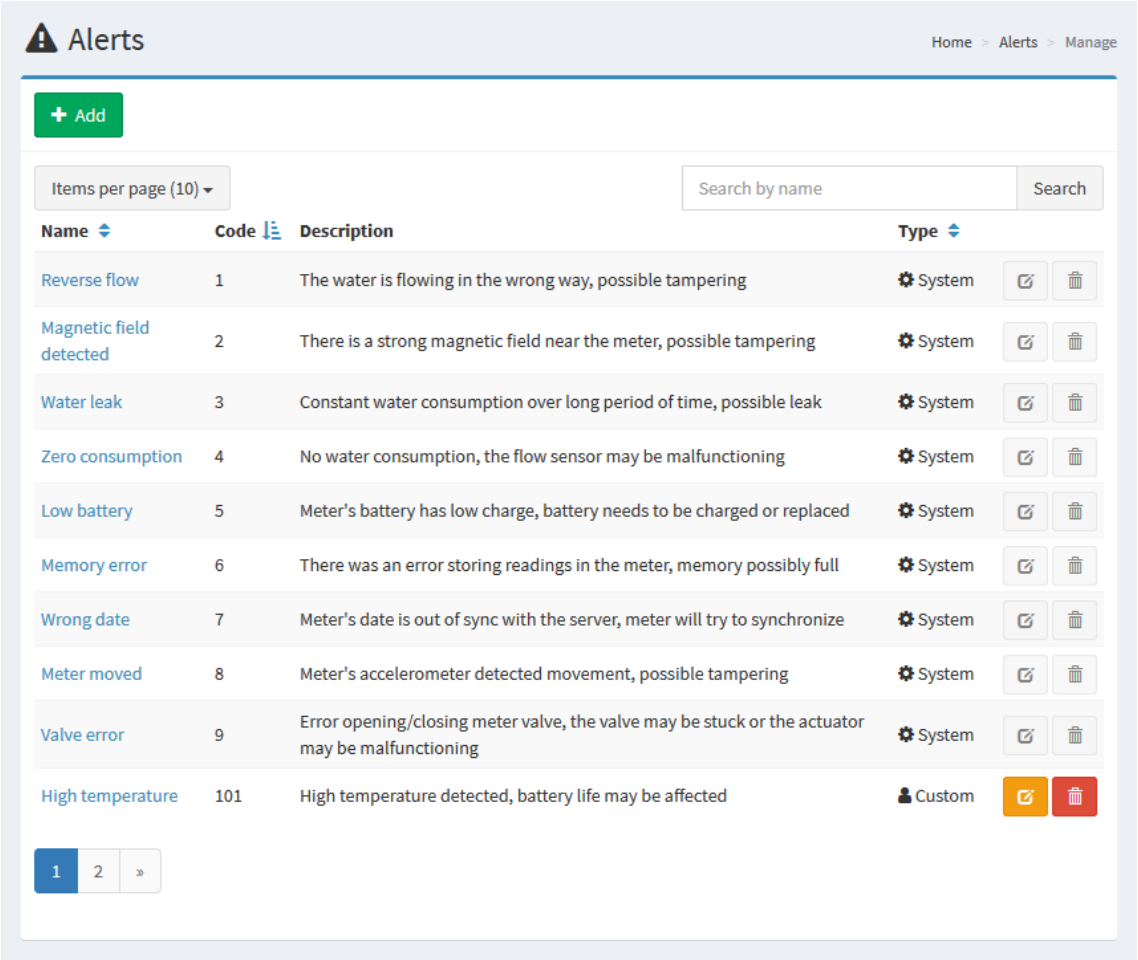



**Figura 9-4 – Lista de alertas com alerta personalizado de temperatura elevada**

Para adicionar um alerta personalizado basta introduzir o nome, código numérico e uma breve descrição. O código numérico é enviado pelos contadores quando estes emitem um alerta e é utilizado pelo sistema para identificar o tipo de alerta, mostrar o alerta no *website* de administração e registar o alerta nas notificações. Depois de adicionar alertas personalizados é possível associá-los a modelos de contadores, de modo a que os contadores baseados nesses modelos possam emitir os novos alertas (ver Figura 9-1).

### 9.1.4. Configurar *gateways* remotamente

Os *gateways* podem ser configurados remotamente a partir do *website* de administração. Do ponto de vista do sistema um "*gateway*" é qualquer dispositivo que satisfaça todas as seguintes condições:

- Serve de intermediário entre a máquina virtual com o IIS e os contadores.
- Consegue interagir corretamente com o *web service* no IIS.
- Possui um *web service* de acordo com a especificação do *web service* de *gateway* (Anexo C: Especificação do *web service* de *gateway*) que se encontra acessível a partir da máquina com o IIS.
- Está registado e autorizado no *website* de administração.

No sistema atual os *gateways* são implementados pelo sistema desenvolvido no Node-RED.

Para que os *gateways* possam utilizar o *web service* no IIS é necessário configurar os parâmetros de ligação e a palavra-passe do web service. De forma semelhante, para que o *website* de administração possa utilizar o *web service* dos *gateways* é necessário que o *gateway* possua uma palavra-passe para o seu próprio *web service* (o identificador de *gateway* é utilizado como nome de utilizador para autenticar em ambos os *web services*).

No *website* de administração é possível alterar as configurações dos *gateways*. Quando estas configurações são submetidas pelo *web browser* e validadas pelo código do lado do servidor do *website*, o código do lado do servidor do *website* efetua um pedido para o *web service* do *gateway* com as novas configurações (o código de servidor do *website* de administração atua como *proxy* entre o *web browser* e o *gateway*). Se a resposta do pedido de alteração da configuração do *gateway* indicar que as novas configurações do *gateway* foram aceites e aplicadas, o código do *website* guarda as novas configurações na base de dados e mostra uma mensagem de sucesso.

A Figura 9-5 mostra a página de configuração remota de um *gateway*. Nesta página é possível configurar os parâmetros de ligação ao *web service* no IIS, a palavra-passe utilizada para autenticar no *web service* no IIS, a palavra-passe de *gateway* que o *website* deve utilizar para se autenticar no *web service* do *gateway*, o identificador da aplicação LoRa a que o *gateway* está associado e ativar ou desativar o modo de resolução de problemas (que faz com que o *gateway* forneça informações mais detalhadas nos ficheiros de *log* e nas respostas do próprio *web service*).

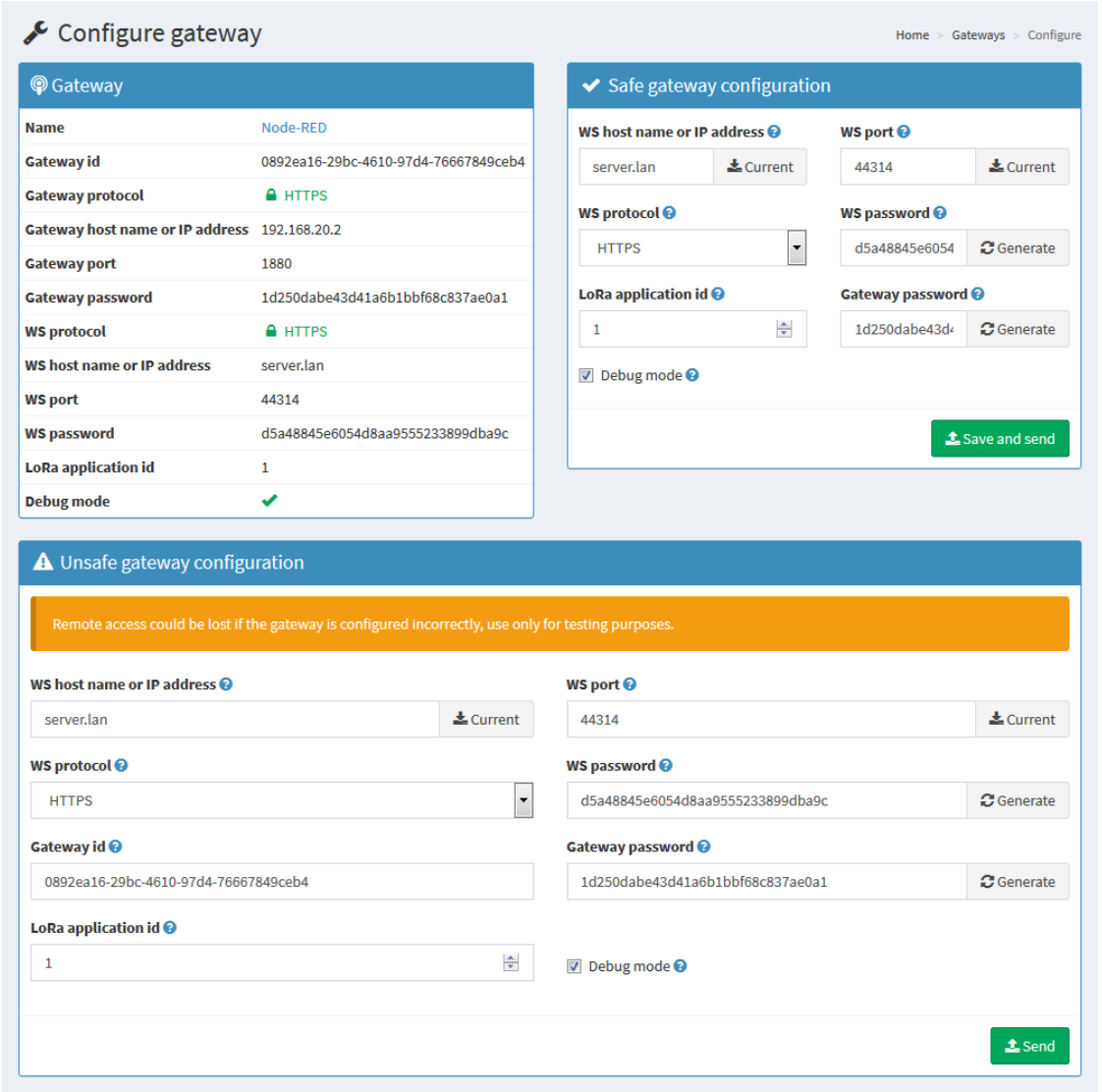



**Figura 9-5 - Configuração remota de um *gateway***

A configuração segura certifica-se de que as alterações aos diversos parâmetros não resultam na perda do acesso remoto ao *gateway*, validando as novas configurações e guardando-as na base de dados caso o *gateway* aplique as novas configurações e responda com uma mensagem de sucesso. A configuração é considerada segura pois as configurações armazenadas na base de dados são mantidas em sincronia com as configurações do *gateway*, não sendo possível chegar a um estado em que, por exemplo, a palavre-passe de *gateway* guardada na base de dados é diferente da palavra-passe de *gateway* guardada no *gateway*. Adicionalmente não é permitido alterar parâmetros que podem fazer com que o acesso remoto seja perdido se estes forem alterados sem medidas de precaução (p. ex.: identificador do *gateway*).

A configuração insegura é uma forma de poder alterar todos os parâmetros de configuração remotamente para fins de teste, valida apenas os tipos de dados dos parâmetros, envia a configuração para o *gateway* e mostra a resposta que o *gateway* deu ao pedido de alteração da configuração. Caso a configuração do *gateway* seja alterada a configuração na base de dados não é atualizada, adicionalmente a configuração insegura permite alterar o identificador do *gateway*. Estas características fazem com que seja possível perder o acesso ao *web service* do *gateway* se o identificador de *gateway* ou a palavra-passe do *gateway* forem alterados para valores inválidos (não existentes na base de dados) e que o *gateway* perca o acesso ao *web service* no IIS se for alterado algum parâmetro do *web service* no IIS para valores inválidos (nome DNS invalido, porto inválido, etc.), sendo necessário aceder ao *gateway* a partir de outro meio (p. ex.: SSH) para poder restaurar a comunicação dos *web services*.

### 9.1.5. Configurar contadores remotamente

Os contadores podem ser configurados remotamente a partir do *website* de administração. É possível configurar o intervalo de amostragem das leituras, o intervalo de comunicação das leituras, ativar ou desativar alertas, definir o valor de leitura do contador, definir o valor de leitura a que o contador deve fechar a válvula e agendar a abertura ou fecho da válvula.

Quando um contador recebe mensagens de configuração verifica se são válidas e se forem válidas altera a configuração e responde com uma mensagem que contém o código de sucesso (normalmente zero); se não forem válidas não altera a configuração e responde com uma mensagem que contém o código do erro ocorrido.

A Figura 9-6 mostra a página de configuração de um contador, cada secção (p. ex.: Set meter value) está associada a um tipo de mensagem do protocolo de rádio.

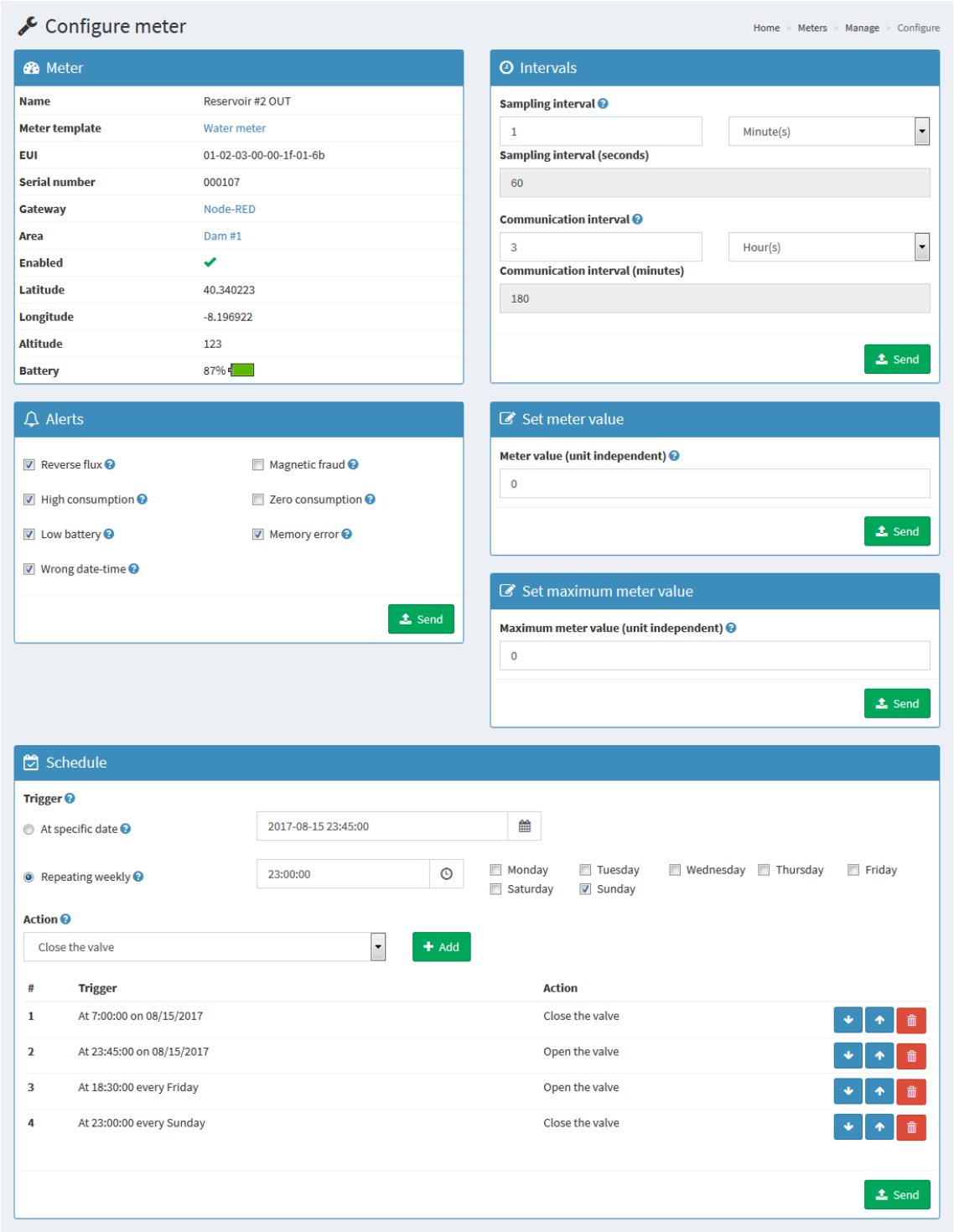



**Figura 9-6 - Configuração remota de um contador**

### 9.1.6. *Dashboard* configurável

O *dashboard* pode ser configurado para mostrar as leituras que o utilizador pretender ver, organizadas da forma que preferir. O dashboard serve de resumo do sistema, mostrando normalmente leituras dos contadores mais importantes (no caso dos contadores de água, possivelmente contadores em estações de tratamento, reservatórios, etc.). Podem ser criados separadores aos quais podem ser adicionados gráficos de vários tipos e indicadores que mostram leituras em tempo quase real. As leituras apresentadas

nos gráficos podem vir de contadores e sensores diferentes, permitindo combinar leituras de várias origens no mesmo gráfico e detetar eventuais correlações visualmente.

A Figura 9-7 mostra uma lista de separadores do *dashboard*, nesta configuração existem quatro separadores. O primeiro separador mostra leituras da rede de distribuição de água, o segundo mostra estatísticas da rede, tais como o *ping*, a taxa de transferência de dados e a perda de pacotes, o terceiro mostra a utilização de recursos do sistema, tais como a percentagem de tempo de processador utilizada, a memória RAM utilizada e a percentagem de espaço livre no disco rígido e o quarto mostra as leituras de contadores de energia elétrica que medem o consumo das máquinas que compõem o sistema.

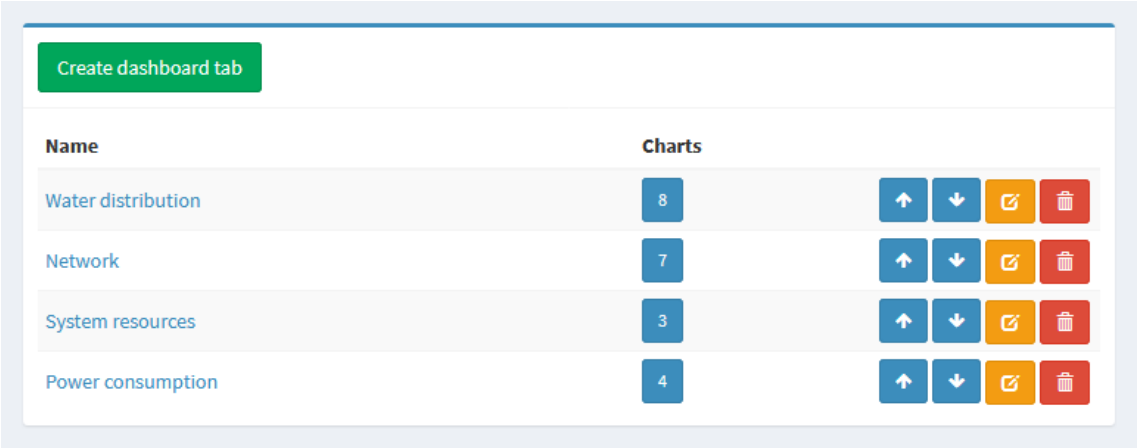



**Figura 9-7 - Lista de separadores do *dashboard***

A Figura 9-8 mostra a lista de gráficos configurados no separador “Water distribution”. Existem gráficos para hipotéticos contadores instalados em várias áreas, tais como o centro de uma cidade, a zona industrial de uma cidade, estações de tratamento ou reservatórios de água.

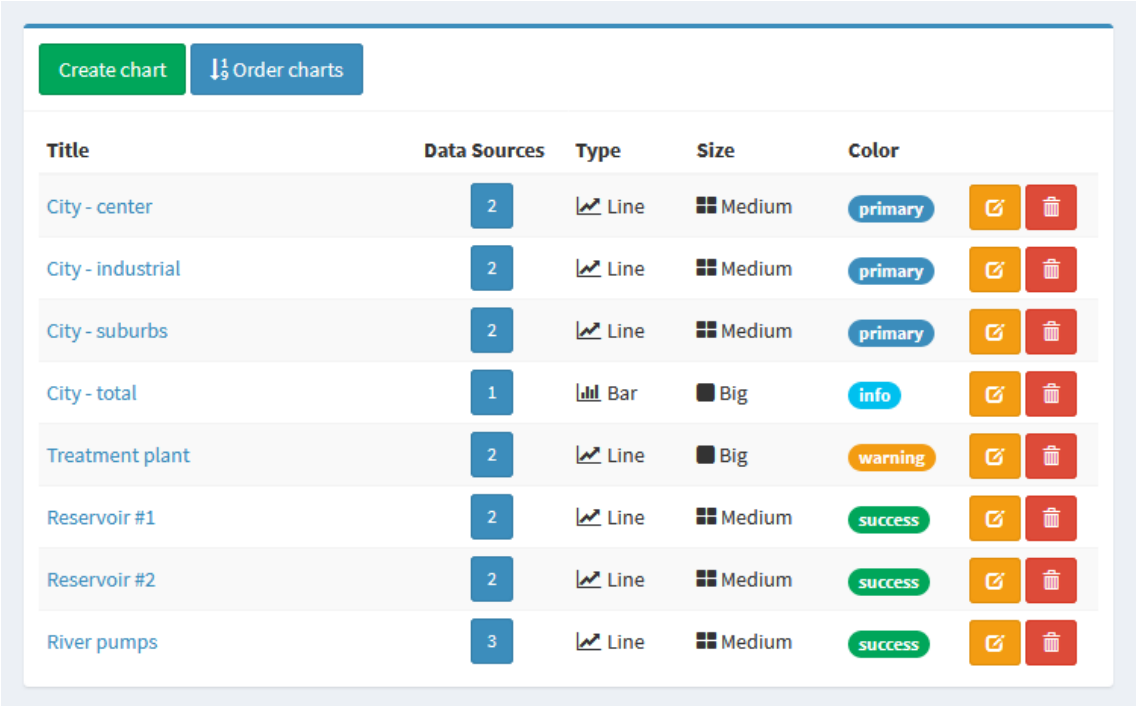



**Figura 9-8 - Lista dos gráficos de um separador**

A Figura 9-9 mostra uma lista de fontes do gráfico "City - center". Os dados deste gráfico são obtidos a partir de dois hipotéticos contadores de água na rede de distribuição que medem o consumo da parte norte e sul da cidade. O sensor de ambas as fontes de dados é o mesmo (volume de água medido em metros cúbicos) e portanto foram atribuídas cores diferentes a cada fonte de dados para se distinguirem melhor no gráfico.

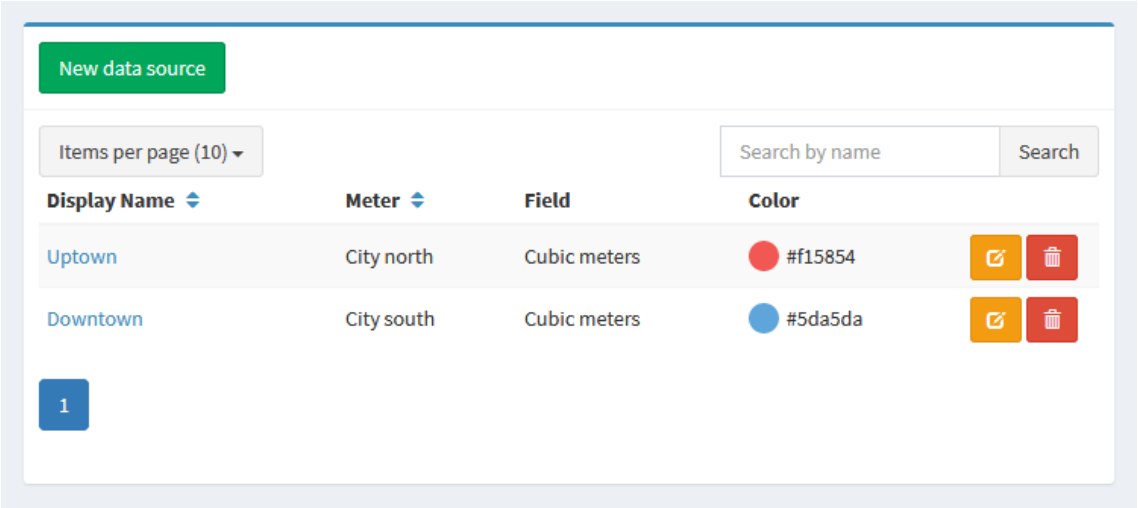



**Figura 9-9 - Lista das fontes de um gráfico**

# 9.2. Registo

As bases de dados desenvolvidas além de guardarem configurações do sistema também guardam registos históricos de eventos. Nesta seção serão apresentados os tipos de dados históricos que são armazenados.

## 9.2.1. Histórico de leituras

Na página do histórico de leituras é possível construir uma consulta à base de dados e visualizar os dados na forma de gráficos e tabelas e adicionalmente, é possível exportar dados das leituras. A Figura 9-10 mostra o histórico de leituras de um contador. Neste caso o contador é uma estação meteorológica. Na secção "Meter" o utilizador pode escolher o contador que se pretende visualizar leituras. Para facilitar a procura do contador numa grande lista de contadores é possível filtrar os contadores por área e por *gateway*, limitando o número de contadores apresentados na lista. Na secção "Sensors" o utilizador pode escolher quais os sensores dos contador que pretende ver as leituras, sendo dada a opção de marcar ou desmarcar todos os sensores. Na secção "Options" o utilizador pode escolher o intervalo de tempo dentro do qual pretende ver as leituras, existem intervalos de tempo pré-definidos que podem ser selecionados para facilitar a consulta, tais como "Ontem" ou "Últimos três meses". Também é possível agrupar leituras por intervalos de tempo utilizando funções de agrupamento, de modo a ser possível ver, por exemplo, médias diárias das leituras de um contador. Nas secções "Chart options" e "Table options" é possível configurar a apresentação dos gráficos e tabelas. Depois de submeter a consulta e se esta devolver leituras, pode ser apresentado um gráfico por cada sensor selecionado. O tipo de gráfico e algumas configurações, tais como a escala ou a cor depende do modelo do contador. Também pode ser apresentada

uma tabela que contém todas as leituras dos sensores selecionados, a partir desta tabela é possível exportar as leituras para o formato XLS, XLSX ou CSV.

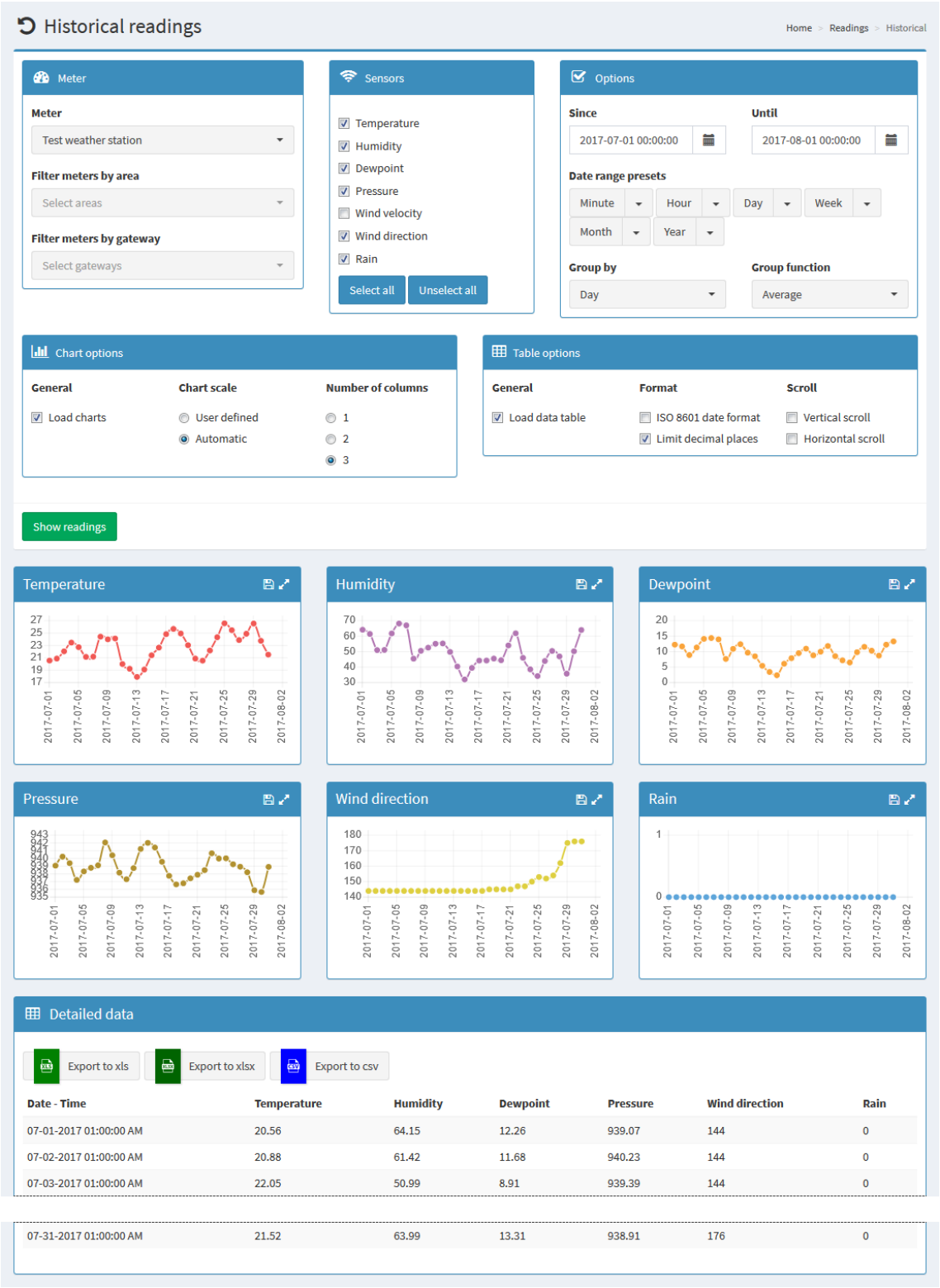



**Figura 9-10 - Histórico de leituras de um contador**

### 9.2.2. Notificações

Quando ocorrem eventos importantes são emitidas notificações. Essas notificações são mostradas a todos os utilizadores do *tenant* a que a notificação diz respeito e armazenadas na base de dados, à exceção das notificações de sistema que são mostradas a todos os utilizadores. A Figura 9-11 mostra os vários tipos de notificações que podem ser visualizadas a partir do *website* de administração. As notificações do tipo "trigger" são emitidas quando algum *trigger* é ativado. As notificações do tipo "alert" são emitidas quando algum contador emite um alerta. As notificações do tipo "meter log" são emitidas quando algum contador responde a comandos ou alterações de configuração. Por fim, as notificações do tipo "system" são emitidas quando acontece algum evento no sistema que é do interesse de todos os utilizadores de todos os *tenants*.

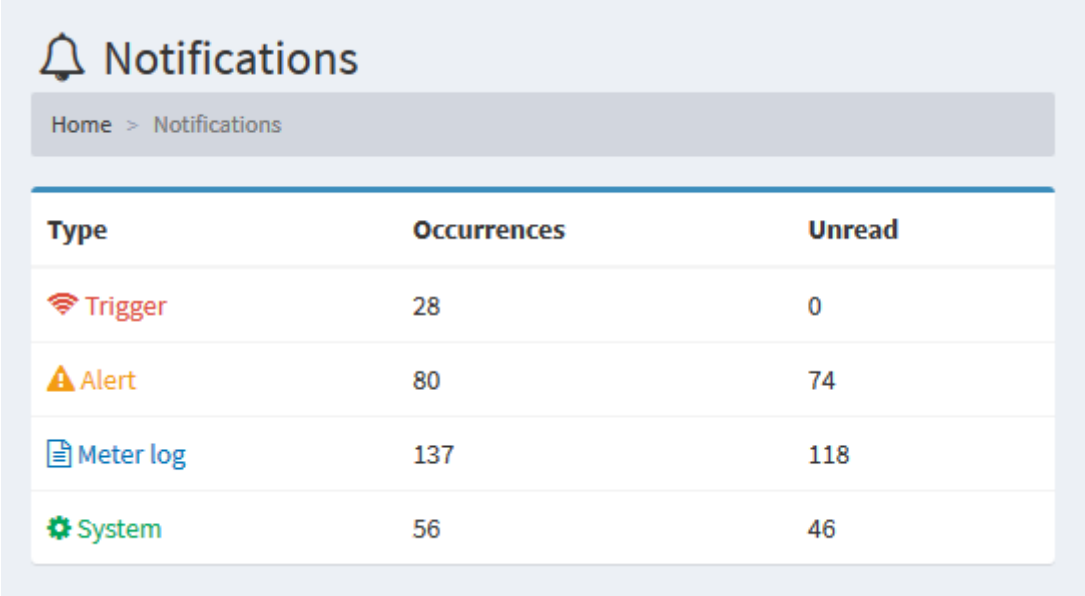



**Figura 9-11 - Tipos de notificações**

A Figura 9-12 mostra um exemplo de notificação, a notificação de registo do contador; mais especificamente uma mensagem de resposta do contador que reporta o estado da bateria. A notificação possui um título que descreve brevemente o que aconteceu, uma mensagem que explica detalhadamente o que aconteceu, a data em que a notificação foi emitida, o tipo de notificação, o subtipo de notificação (caso se aplique), o nome atual da fonte que emitiu a notificação (que é uma hiperligação para a fonte caso ainda exista) e o nome original da fonte da notificação quando a notificação foi emitida, servindo de referência caso o nome da fonte seja alterado entretanto ou a fonte seja apagada.

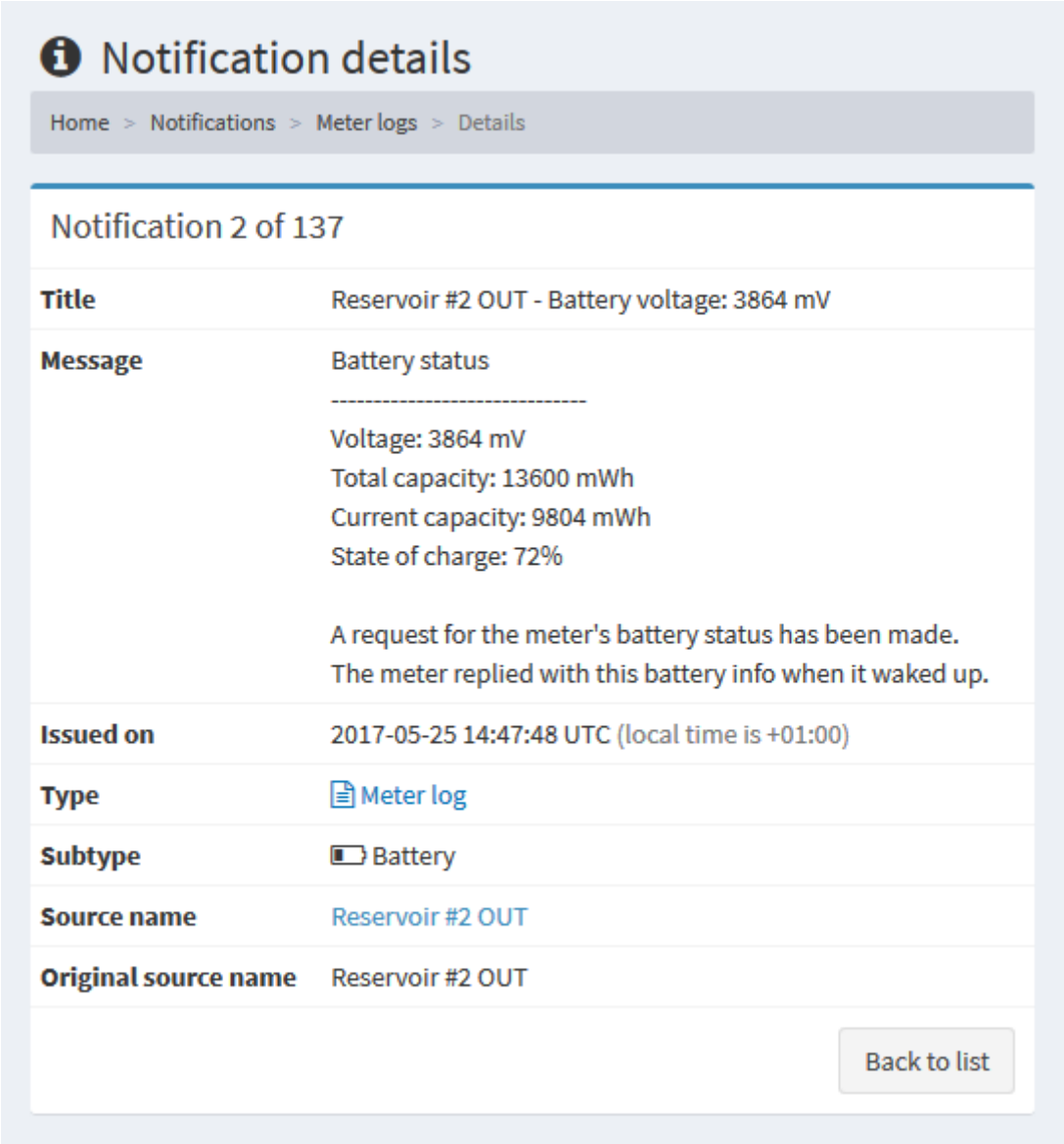



**Figura 9-12 - Exemplo de notificação**

### 9.2.3. Registo do sistema

O registo do sistema regista mensagens de depuração (*debug*), mensagens de informação, mensagens de aviso, mensagens de erro e mensagens de falha crítica. As mensagens de *debug* servem para ajudar o utilizador na resolução de problemas (p. ex.: repostas do *web service* no IIS) e têm que ser ativadas previamente na configuração do sistema para serem registadas. As mensagens de informação informam que alguma operação importante foi executada (p. ex.: execução agendada de *stored procedures*). As mensagens de aviso informam que ocorreu algum problema durante a execução de uma operação, mas que a operação foi executada com métodos alternativos (p. ex.: não foi possível inserir leituras com o *stored procedure* de inserção de leituras do contador, foi gerado um *prepared statement* para concluir a operação). As mensagens de erro informam o motivo pelo qual alguma operação não foi concluída (p. ex.: não foi possível guardar leituras de um contador na base de dados porque o contador especificado não existe na base de dados ou está desativado). As mensagens de falha crítica informam que o sistema não se encontra operacional. Existe um script

"*watchdog*" na máquina que possui o Node-RED que tenta obter periodicamente uma página do *website* e efetuar um pedido para o *web service* no IIS. Se não conseguir aceder a essa página ou efetuar o pedido para o *web service* guarda o evento e mais tarde tenta enviá-lo para o *web service* no IIS, sendo registado como uma falha crítica. Existem registos que pertencem a um determinado *tenant* e só estão visíveis a utilizadores desse *tenant* (p. ex.: erro – não foi possível adicionar um contador) e registos que pertencem ao sistema e que podem ser vistos por todos os utilizadores (p. ex.: falha crítica – o *website* esteve inacessível às 14:17 do dia 30/06/2017). A Figura 9-13 mostra o registo do sistema, nesta figura podem-se ver algumas mensagens de erro do *website* e do *web service*.

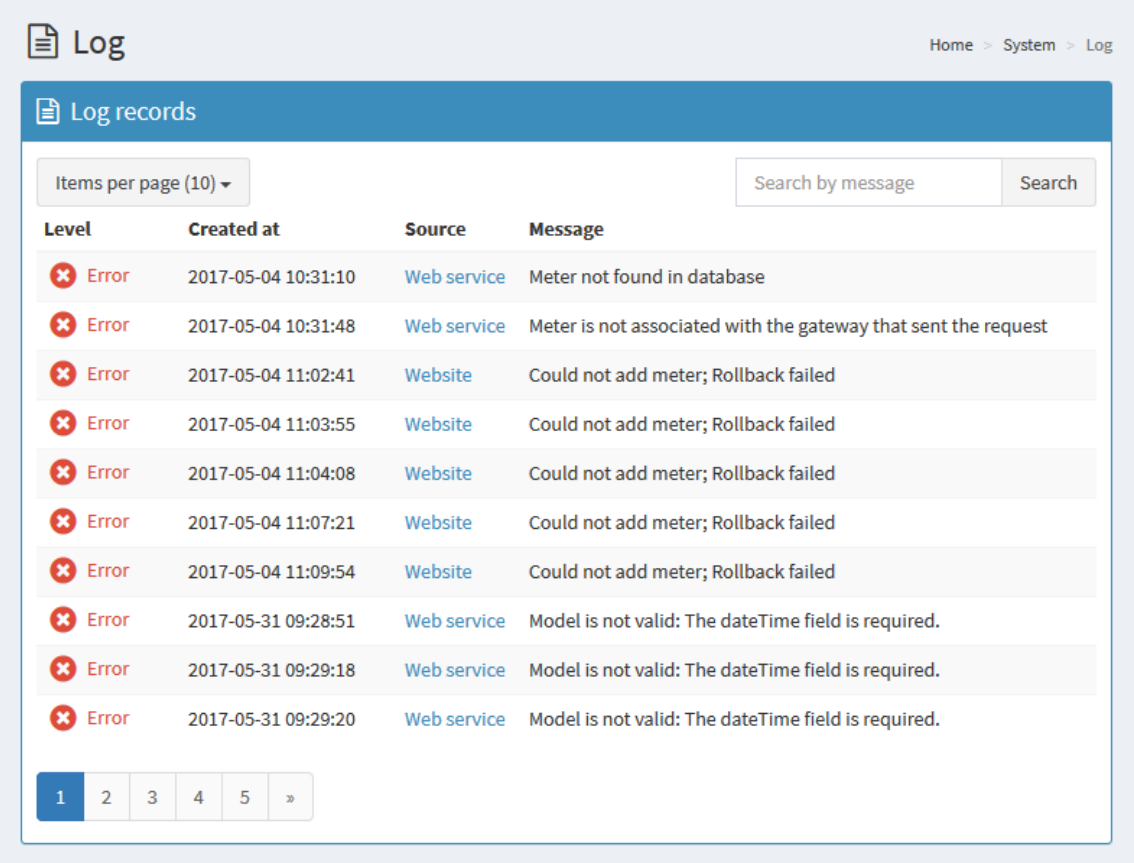



**Figura 9-13 - Registo de mensagens do sistema**

# 9.3. Resolução de problemas

A partir do *website* de administração é possível resolver os problemas mais comuns que podem ocorrer com o sistema, quer sejam relacionados com a base de dados, *gateways* ou contadores.

## 9.3.1. Deteção e correção de problemas

Como os objetos da base de dados de leituras são adicionados e removidos dinamicamente, quando o utilizador adiciona ou remove contadores, não se sabe à partida quais os objetos que a base de dados tem, pois estes podem variar com a utilização. Torna-se portanto necessário certificar que esses objetos adicionados e removidos dinamicamente existem na base de dados. Caso deixem de existir, as

operações que envolvem esses objetos começam a gerar erros no registo do sistema. A Figura 9-14 mostra a página de deteção e correção de problemas do sistema. Esta página permite detetar e restaurar objetos da base de dados em falta. Adicionalmente permite corrigir problemas com o agendamento de tarefas de agregação de leituras e agregar leituras manualmente caso estas tarefas não tenham sido executadas no tempo devido, a funcionalidade de cada secção é descrita na figura.

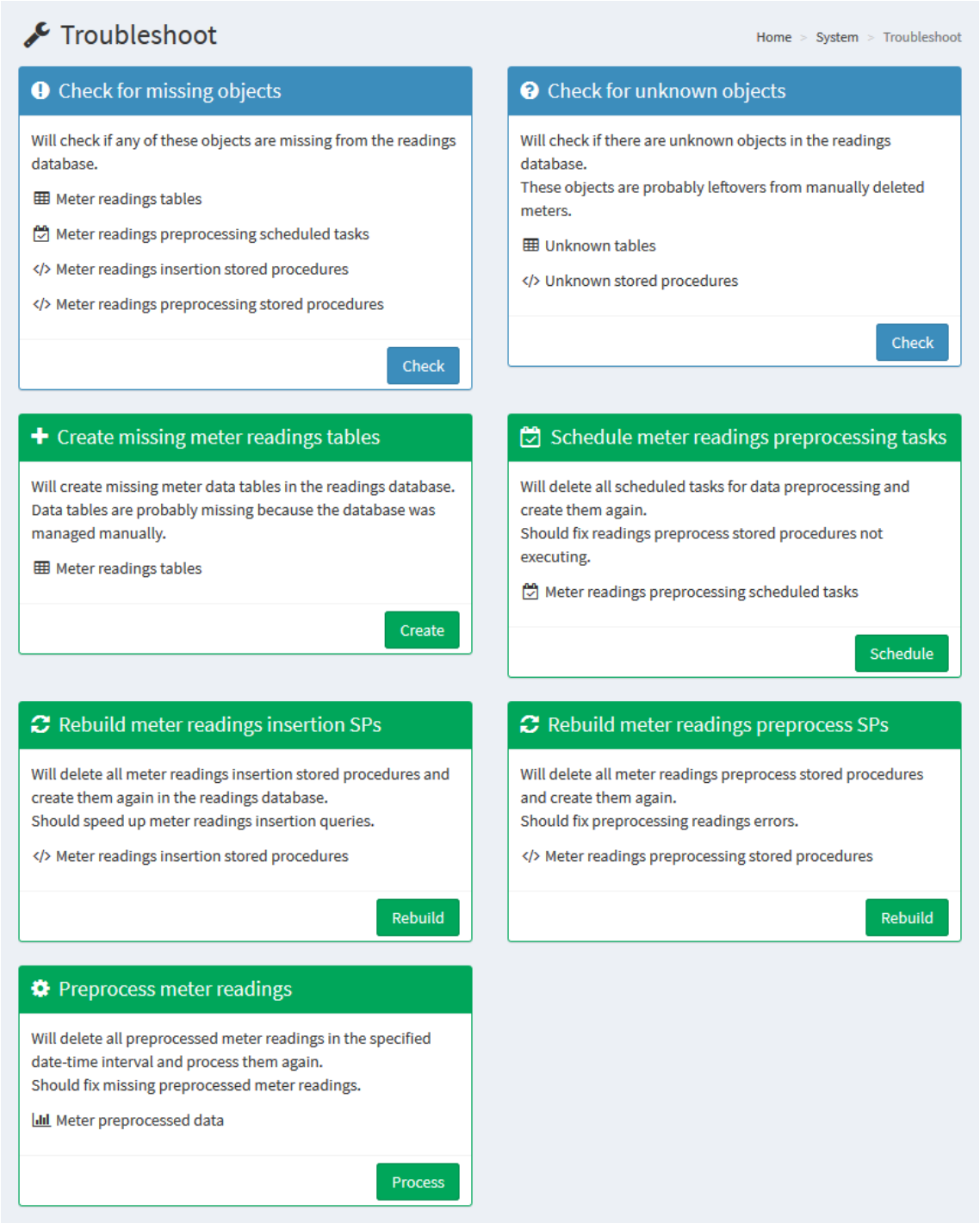



**Figura 9-14 – Deteção e correção de problemas do sistema**

### 9.3.2. Mapeamento do sinal LoRa

Foram desenvolvidos contadores pelos meus colegas que possuem apenas microcontrolador, bateria, módulo de rádio LoRa e módulo recetor de GPS. Estes contadores servem para mapear a cobertura do sinal da rede LoRa, de modo a perceber se existem áreas sem cobertura de sinal ou áreas que são cobertas apenas por um *gateway*. Os contadores podem ser transportados facilmente por pessoas ou podem ser instalados em automóveis e fazem parte da classe C de dispositivos LoRa, estando portanto sempre ativos e prontos para receber e responder a mensagens, não sendo necessário qualquer tipo de manuseamento durante uma sessão de mapeamento de sinal.

A partir do *website* de administração é possível criar, ativar, desativar, renomear e apagar sessões de mapeamento de sinal dos contadores, efetuar um pedido de informações das condições atuais de receção e envio do sinal LoRa do contador, limitar o intervalo de tempo em que devem ser mostradas ou exportadas as informações das condições do sinal LoRa e apagar informações das condições do sinal LoRa da sessão, que se encontrem dentro de um intervalo de tempo especificado.

Quando o contador recebe mensagens de solicitação das condições do sinal LoRa, guarda as informações das condições de receção dessa mensagem enviada pelo *gateway* e tenta obter informações sobre a localização atual através do módulo recetor de GPS. Se conseguir obter a localização atual envia uma mensagem com as informações da localização atual e as informações de receção da mensagem LoRa. Se não conseguir obter a localização atual descarta as informações de receção da mensagem LoRa e envia uma mensagem com o código de erro que indica qual o problema que ocorreu (p. ex.: o recetor de GPS não detetou sinais de satélites GPS).

Quando o Node-RED recebe uma mensagem de resposta de mapeamento de sinal, obtém informações das condições de receção da mensagem do contador a partir da mensagem MQTT recebida do LoRa App Server. A mensagem MQTT do LoRa App Server inclui os dados do contador e informações das condições de receção da mensagem LoRa (ver Figura 8-13). O Node-RED efetua um pedido HTTP para o *web service* no IIS, em que incluí informações sobre as condições de receção da mensagem recebida pelo contador, informações sobre as condições de receção da mensagem recebida pelo *gateway* que reportou o valor de RSSI mais elevado e informações sobre a localização atual do contador.

Estas informações são guardadas na sessão de mapeamento de sinal ativa do contador, caso o contador possua uma sessão ativa. Se o contador não possuir uma sessão ativa, as informações são descartadas e é adicionada uma mensagem de erro ao registo do sistema.

A Figura 9-15 mostra a página de mapeamento de sinal dos contadores com uma sessão aberta. Nesta página é possível visualizar um mapa com marcadores em que cada marcador é colocado na localização reportada pelo contador. A cor do marcador depende da média da força do sinal recebido pelo contador e pelo *gateway*. Vermelho e laranja significam sinal fraco, amarelo e verde-claro significam sinal médio, verde-escuro e azul significam sinal forte. Também é possível ver todas as mensagens de pedido de mapeamento de sinal e respetivas mensagens de resposta agrupadas numa tabela, onde se pode ver se a mensagem foi transmitida do *gateway* para o contador (pedido) e do contador para o *gateway* (resposta), a data e hora em que a mensagem foi recebida pelo *gateway*, a força do sinal LoRa, a relação entre o sinal e o ruído, o fator de espalhamento espectral, taxa de código, largura de banda, frequência, canal de rádio, latitude, longitude e altitude.

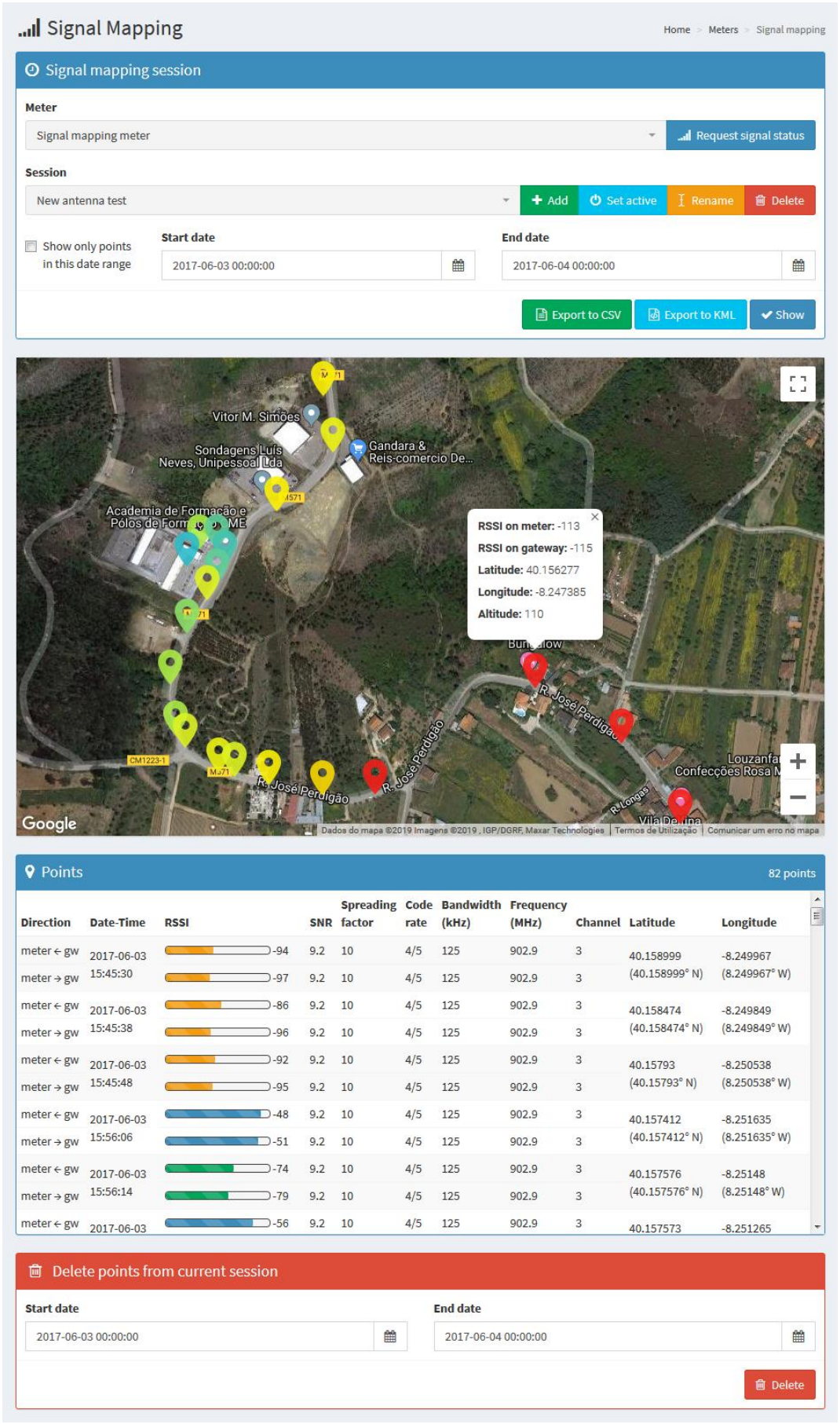



**Figura 9-15 – Sessão de mapeamento de sinal com um contador de testes**

### 9.3.3. Leituras em tempo real

No *website* de administração é possível visualizar as leituras assim que estas são recebidas pelo *web service* no IIS. Esta funcionalidade é útil para testar se os contadores estão a comunicar leituras corretamente e se o *web service* está a receber leituras válidas.

A Figura 9-16 mostra a página de leituras em tempo real. Neste caso são apresentadas as leituras em tempo real de um contador de água de testes. As leituras de todos os sensores são apresentadas na forma de gráfico e na forma de tabela. É possível receber, guardar e mostrar um número infinito de leituras no *website* (limitado pela memória do computador), ou limitar o número de leituras a guardar e mostrar no *website*, sendo que as leituras mais antigas são eliminadas criando uma "janela deslizante" de leituras ao longo do tempo.

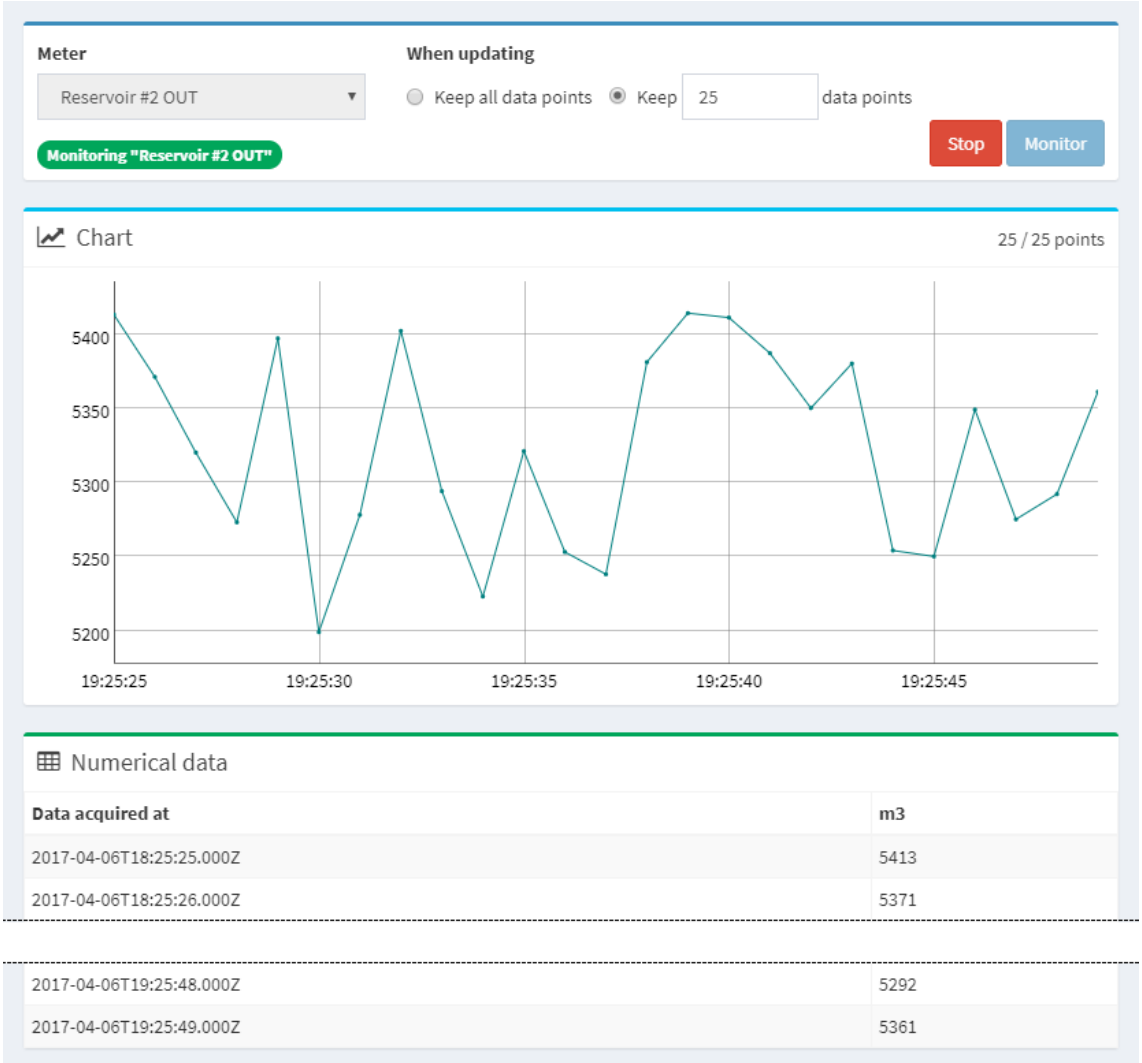



**Figura 9-16 - Leituras em tempo real**

### 9.3.4. Notificações em tempo real

As notificações são mostradas no *website* na forma de pequenas janelas sobre a página atual, assim que são recebidas pelo *web service* no IIS. Desta forma um utilizador que

tenha sessão iniciada no *website* de administração vai ser notificado imediatamente de eventuais problemas com os contadores ou com o sistema. A Figura 9-17 mostra uma janela de notificação de alerta de um contador.

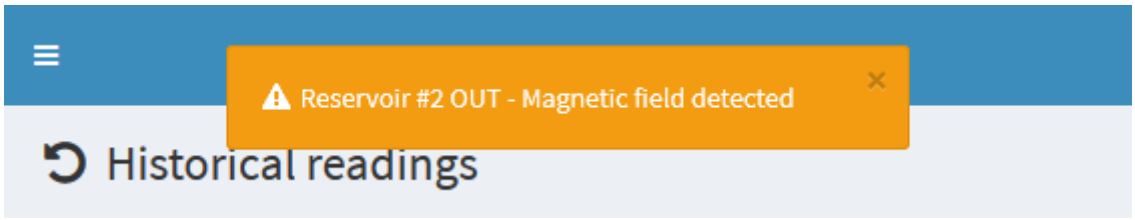



**Figura 9-17 - Alerta em tempo real**

Quando se faz um pedido de comando ou configuração para um contador é pouco provável que este responda imediatamente, pois os contadores passam a maior parte do tempo em modo de poupança de energia. Neste modo o módulo de rádio LoRa está desligado e portanto não pode receber mensagens. O contador só fica á espera de mensagens pendentes durante um pequeno intervalo de tempo depois de ligar o módulo de rádio LoRa e transmitir pelo menos uma mensagem. Portanto a resposta a um pedido efetuado pela rede LoRa é assíncrona. As mensagens assíncronas de resposta dos contadores também geram notificações. Se um utilizador efetuar um pedido para um contador vai ser notificado assim que o contador responder. A Figura 9-18 mostra uma janela de notificação de resposta a um pedido de relatório do estado da bateria.

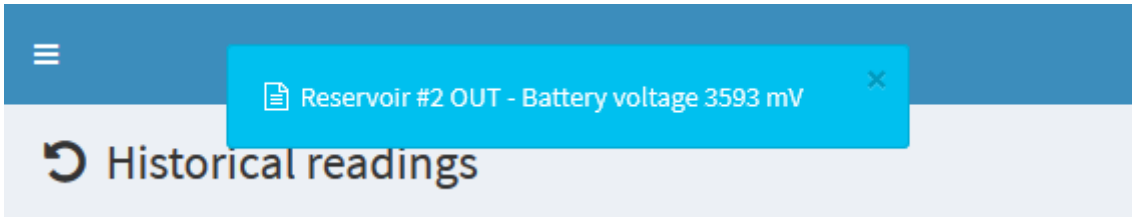



**Figura 9-18 - Informações do contador em tempo real**

As janelas de notificação aparecem sobre o cabeçalho da página atual. Portanto não impedem a utilização do *website* de administração. No entanto podem ser fechadas a qualquer altura clicando no ”X” no canto superior direito e também desaparecem automaticamente passado algum tempo se o utilizador não interagir com elas. Mesmo assim, se um utilizador não quiser ser incomodado, por exemplo, se estiver a receber muitas notificações simultaneamente, pode a qualquer momento desativar as notificações em tempo real nas preferências do seu perfil.

## 9.4. Escalabilidade

A escalabilidade deste sistema permite que a sua capacidade seja ajustada à carga, de modo a que o sistema se mantenha rápido e funcional em períodos de carga elevada. O sistema pode escalar verticalmente, adicionando mais recursos computacionais, e horizontalmente em alguns componentes, adicionando mais componentes do mesmo tipo.

### 9.4.1. Multitenancy

Executar múltiplas instâncias separadas de bases de dados, *website*, *web service* no IIS e Node-RED pode não ser eficiente se essas instâncias servirem um número reduzido de contadores, *gateways* ou utilizadores, pois cada componente tem uma utilização base fixa de recursos. Nesses casos é mais eficiente suportar vários *tenants* (clientes) com apenas uma instância de cada componente, sendo que os utilizadores e *gateways* de um determinado *tenant* têm apenas acesso aos dados do *tenant* a que pertencem. Deste modo, o investimento inicial e os custos operacionais dos sistemas de vários *tenants* podem ser reduzidos. Do ponto de vista dos utilizadores o sistema atua como se fosse composto por instâncias dedicadas dos vários componentes, os utilizadores não conseguem determinar se encontram-se a partilhar uma instância com outros *tenants*.

A falha de uma instância partilhada por utilizadores de vários *tenants* faz com que todos esses *tenants* fiquem sem acesso ao sistema. Portanto antes de decidir suportar vários *tenants* com apenas uma instância de cada componente, é necessário estimar se a capacidade do sistema partilhado é suficiente para todos os *tenants* (p. ex.: capacidade de armazenamento ou processamento) e avaliar previamente se um *tenant* pode ser um risco para os restantes *tenants* que partilham a instância (p. ex.: se o *tenant* necessita de expor o sistema à Internet enquanto que os outros *tenants* utilizam apenas redes privadas).

A decisão de suportar múltiplos *tenants* com apenas uma instância de cada componente afetou a estrutura da base de dados, o desenvolvimento do *website* de administração e a especificação do *web service* no IIS, pois antes de realizar operações no *website* ou no *web service* é necessário verificar se o utilizador ou *gateway* está a tentar aceder a dados do seu *tenant*, sendo que cada registo na base de dados deve ser associado a um determinado *tenant* de forma direta ou indireta. A Figura 9-19 mostra a lista de tenants configurados no sistema, esta página só está acessível a utilizadores com nível de permissão “super administrator”, a partir desta página é possível alterar os nomes dos *tenants*, os respetivos nomes dos *websites* de administração e gerir todos os utilizadores de cada *tenant*.

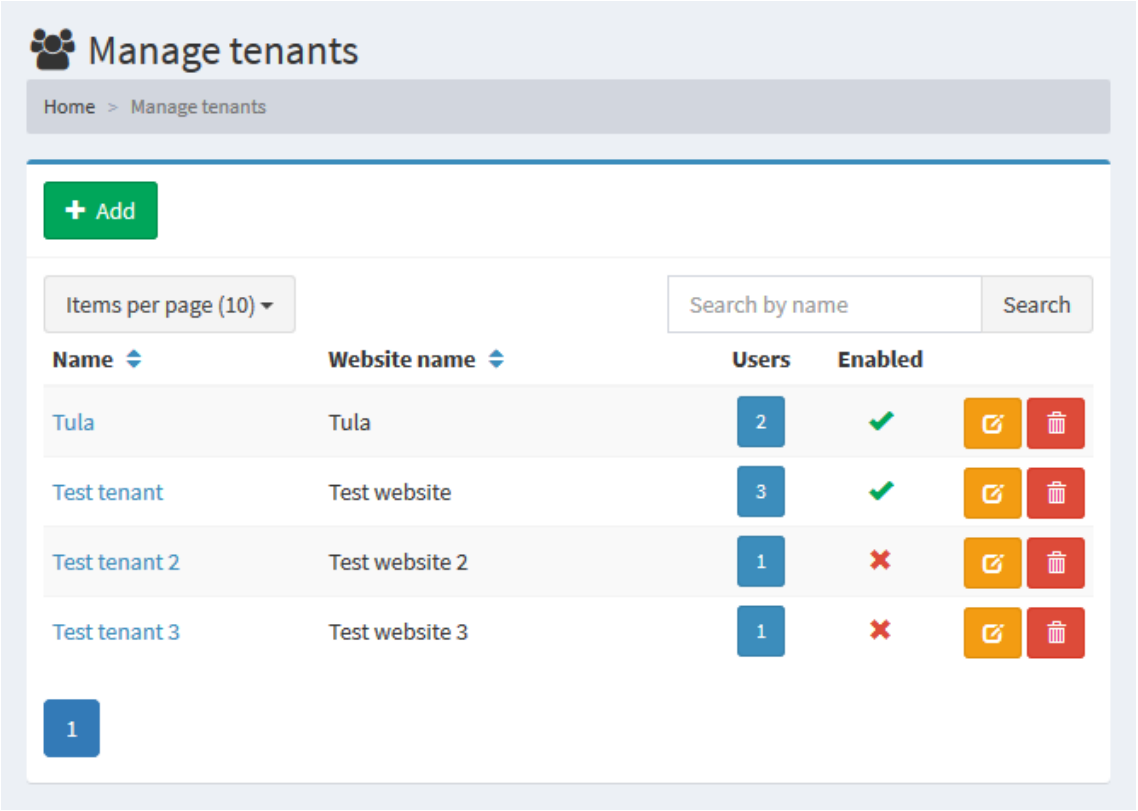



**Figura 9-19 - Lista de tenants configurados**

## 9.4.2. Rede LoRa

Os *gateways* LoRa são dispostos de forma a cobrir a maior área possível, ficando dentro do alcance de todos os contadores instalados e permitindo que a cobertura de vários *gateways* se sobreponha, de modo obter redundância na comunicação com os contadores. A Figura 9-20 mostra um exemplo de sobreposição da cobertura de sinal dos *gateways*, a cobertura do *gateway* azul e do *gateway* vermelho está sobreposta, dentro da área de sobreposição ambos os *gateways* podem receber mensagens dos contadores e ambos podem transmitir mensagens para os contadores, embora apenas um deles será selecionado pelo servidor de rede LoRa para transmitir mensagens para um determinado contador, com base nas condições do sinal de mensagens anteriores.

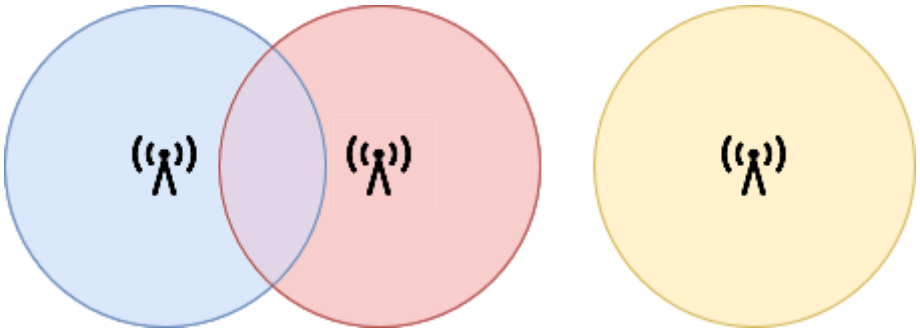


**Figura 9-20 - Exemplo de sobreposição da cobertura de sinal dos *gateways***

Depois de planear e instalar os contadores e *gateways* LoRa nos locais mais adequados, pode ser necessário instalar contadores em novos clientes que não se encontram dentro do alcance da rede LoRa. Devido à especificação LoRaWAN, para aumentar a cobertura

de uma rede LoRa basta instalar mais *gateways* LoRa em locais estratégicos, pois se estes forem coordenados pelo mesmo LoRa Server não vão interferir uns com os outros. Adicionalmente a ligação ao LoRa Server é feita através de outra tecnologia de rede que não utiliza a mesma parte do espetro radielétrico utilizado pela rede LoRa.

### 9.4.3. Servidores

Embora seja possível fazer várias combinações de componentes de *hardware* e *software*, a configuração atual (ver Figura 5-3) foi escolhida porque permite separar responsabilidades e adicionar facilmente capacidade a pontos de congestionamento (*bottleneck*) que possam eventualmente surgir. Como exemplo, se o LoRa Server ficar sobrecarregado com mensagens de vários dispositivos LoRa, pode-se escalar o LoRa Server verticalmente aumentando a capacidade de processamento, memória ou entrada/saída da máquina em que o LoRa Server está instalado.

Como o LoRa Server, LoRa App Server e Node-RED foram instalados em máquinas virtuais diferentes, é possível aumentar a capacidade destes componentes de software copiando as respetivas máquinas virtuais e configurando-as para trabalhar em paralelo com as máquinas atuais, pois uma instancia do LoRa App Server pode ligar-se a mais do que uma instancia do LoRa Server e é possível adicionar vários *gateways* (instâncias do Node-RED) no *website* de administração, sendo possível escalar horizontalmente o LoRa Server, LoRa App Server e Node-RED. Adicionalmente as instâncias do Eclipse Mosquitto e as instâncias do PostgreSQL podem ser executadas em máquinas virtuais separadas, se necessário.

Infelizmente as bases de dados, o *website* e o *web service* no IIS não foram projetados para escalar horizontalmente, pois não existe um mecanismo de coordenação entre estes componentes. No entanto devido à utilização esperada do sistema, em que os contadores comunicam leituras diariamente, em princípio uma máquina com hardware mediano pode suportar a taxa atual de inserção de leituras (ver capítulo 11). Adicionalmente contadores que se encontrem dentro da cobertura de sinal de outros contadores (problema do nó exposto), e contadores que se encontrem dentro da cobertura de sinal dos mesmos *gateways* mas que não se encontram dentro da cobertura de sinal de ambos (problema do nó oculto), não conseguem transmitir leituras em simultâneo no mesmo canal, pois uma rede LoRa utiliza o método ALOHA para gerir o acesso ao meio, o que significa que vai acontecer uma colisão, o *gateway* não vai aceitar as mensagens corrompidas e os contadores vão tentar transmitir as leituras de novo num canal aleatório, passado um intervalo de tempo aleatório. Mesmo que fosse utilizado o método de acesso ao meio CSMA/CA não seria possível que dois contadores transmitissem ao mesmo tempo no mesmo canal, pois o espectro radioelétrico é partilhado.

## 9.5. Exportação de dados

A partir do *website* de administração é possível exportar diversos tipos e dados apresentados, de modo a poder utilizar esses dados em outras aplicações ou em outras instâncias do sistema.

### 9.5.1. Exportar leituras

As leituras dos contadores podem ser exportadas a qualquer altura para ficheiros, de modo a serem utilizadas em outras aplicações ou sistemas. No histórico de leituras, depois de efetuar uma consulta selecionando os sensores do contador, o intervalo de tempo e opcionalmente agrupar as leituras, pode-se exportar as leituras da tabela. Cada leitura é exportada na forma de linha e cada sensor é exportado na forma de coluna, juntamente com a data e hora da aquisição da leitura.

Para demonstrar esta funcionalidade foi selecionado um intervalo de tempo na página de histórico de leituras, no dia 1 de julho de 2017, entre as dez e onze horas da manhã. Foram selecionados os sensores de temperatura, humidade e pressão atmosférica de uma estação meteorológica e não foi utilizada nenhuma função de agrupamento. Portanto as leituras exportadas são as mesmas que foram reportadas pela estação meteorológica. A Figura 9-21 mostra as leituras que foram selecionadas na base de dados com base na consulta.

Results | Messages

| | id | data_acquired_at | temperature | humidity | dewpoint | pressure | wind_velocity | wind_direction | rain |
|---|---|---|---|---|---|---|---|---|---|
| 2131 | 2131 | 2017-07-01 12:52:42.6800000 +00:00 | 25.88 | 44.00 | 12.72 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2132 | 2132 | 2017-07-01 13:07:39.2770000 +00:00 | 26.11 | 35.00 | 9.49 | 940.73 | 0.00 | 143.00 | 0.00 |
| 2133 | 2133 | 2017-07-01 13:22:40.1930000 +00:00 | 26.61 | 36.00 | 10.27 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2134 | 2134 | 2017-07-01 13:37:40.4230000 +00:00 | 27.77 | 33.00 | 10.11 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2135 | 2135 | 2017-07-01 13:52:40.3270000 +00:00 | 27.88 | 37.00 | 11.88 | 940.05 | 0.00 | 143.00 | 0.00 |
| 2136 | 2136 | 2017-07-01 14:07:40.0600000 +00:00 | 28.00 | 35.00 | 11.11 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2137 | 2137 | 2017-07-01 14:23:10.7270000 +00:00 | 29.27 | 31.00 | 10.50 | 940.05 | 0.00 | 143.00 | 0.00 |
| 2138 | 2138 | 2017-07-01 14:39:43.5430000 +00:00 | 28.72 | 32.00 | 10.38 | 940.39 | 0.00 | 143.00 | 0.00 |
| 2139 | 2139 | 2017-07-01 14:55:14.5370000 +00:00 | 27.88 | 32.00 | 9.72 | 940.05 | 0.00 | 143.00 | 0.00 |
| 2140 | 2140 | 2017-07-01 15:11:43.6870000 +00:00 | 28.88 | 32.00 | 10.61 | 940.05 | 0.00 | 143.00 | 0.00 |

**Figura 9-21 - Leituras selecionadas pela consulta na página de histórico de leituras**

Os dados da Figura 9-21 aparecem na tabela de leituras a seguir aos gráficos de leituras. No cabeçalho da tabela existem três botões que permitem exportar os dados para ficheiros (ver Figura 9-10). Os formatos suportados são folha de cálculo do Microsoft Excel (XLS e XLSX) e ficheiro de valores separados por vírgula (CSV). A Figura 9-22 mostra um exemplo de leituras exportadas no formato CSV e a Figura 9-23 mostra um exemplo de leituras exportadas no formato XLSX.

```
1  Data acquired at,Temperature,Humidity,Pressure
2  2017-07-01T13:07:39.277Z,26.11,35.00,940.73
3  2017-07-01T13:22:40.193Z,26.61,36.00,940.39
4  2017-07-01T13:37:40.423Z,27.77,33.00,940.39
5  2017-07-01T13:52:40.327Z,27.88,37.00,940.05
6  2017-07-01T14:07:40.060Z,28.00,35.00,940.39
7  2017-07-01T14:23:10.727Z,29.27,31.00,940.05
8  2017-07-01T14:39:43.543Z,28.72,32.00,940.39
9  2017-07-01T14:55:14.537Z,27.88,32.00,940.05
```

**Figura 9-22 - Leituras exportadas para um ficheiro no formato CSV**

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Data acquired at | Temperature | Humidity | Pressure |
| 2 | 2017-07-01T13:07:39.277Z | 26.11 | 35.00 | 940.73 |
| 3 | 2017-07-01T13:22:40.193Z | 26.61 | 36.00 | 940.39 |
| 4 | 2017-07-01T13:37:40.423Z | 27.77 | 33.00 | 940.39 |
| 5 | 2017-07-01T13:52:40.327Z | 27.88 | 37.00 | 940.05 |
| 6 | 2017-07-01T14:07:40.060Z | 28.00 | 35.00 | 940.39 |
| 7 | 2017-07-01T14:23:10.727Z | 29.27 | 31.00 | 940.05 |
| 8 | 2017-07-01T14:39:43.543Z | 28.72 | 32.00 | 940.39 |
| 9 | 2017-07-01T14:55:14.537Z | 27.88 | 32.00 | 940.05 |

**Figura 9-23 - Leituras exportadas para um ficheiro no formato XLSX**

Os gráficos das leituras dos contadores também podem ser exportados no formato de imagem PNG. No cabeçalho da secção do gráfico existe um ícone de disquete que permite exportar o gráfico tal como se apresenta. A Figura 9-24 mostra um gráfico de temperatura que foi exportado no formato de imagem PNG.

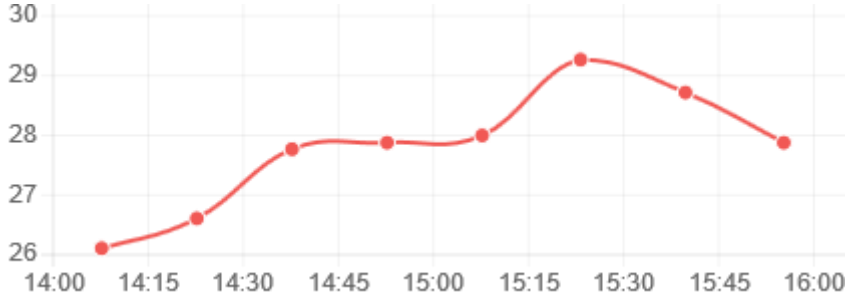



**Figura 9-24 - Gráfico de temperatura exportado no formato PNG**

## 9.5.2. Exportar dados de mapeamento de sinal

Os dados de mapeamento de sinal podem ser exportados no formato de valores separados por vírgula (CSV) ou no formato do Google Earth (KML), na página de mapeamento de sinal. Na secção de consulta existem os botões “Export to CSV” e “Export to KML”. A Figura 9-25 mostra uma sessão de mapeamento de sinal exportada no formato CSV e a Figura 9-26 mostra uma sessão de mapeamento de sinal exportada no formato KML.

```
1  Latitude,Longitude,Altitude,Timestamp,RSSI Meter <- Gateway,SNR Meter <- Gateway,Spreading Factor Meter
   <- Gateway,Code rate Meter <- Gateway ,Frequency Meter <- Gateway,Channel Meter <- Gateway,Bandwidth
   Meter <- Gateway,RSSI Meter -> Gateway,SNR Meter -> Gateway,Spreading Factor Meter -> Gateway,Code rate
   Meter -> Gateway ,Frequency Meter -> Gateway,Channel Meter -> Gateway,Bandwidth Meter -> Gateway
2  40.157573,-8.251265,140,2017-06-03 14:56:18,-56,9.20,10,4/5,902900,3,125,-54,9.20,10,4/5,902900,3,125
3  40.157576,-8.251480,140,2017-06-03 14:56:14,-74,9.20,10,4/5,902900,3,125,-79,9.20,10,4/5,902900,3,125
4  40.157412,-8.251635,140,2017-06-03 14:56:06,-48,9.20,10,4/5,902900,3,125,-51,9.20,10,4/5,902900,3,125
5  40.157930,-8.250538,145,2017-06-03 14:45:48,-92,9.20,10,4/5,902900,3,125,-95,9.20,10,4/5,902900,3,125
6  40.158474,-8.249849,150,2017-06-03 14:45:38,-86,9.20,10,4/5,902900,3,125,-96,9.20,10,4/5,902900,3,125
7  40.158999,-8.249967,152,2017-06-03 14:45:30,-94,9.20,10,4/5,902900,3,125,-97,9.20,10,4/5,902900,3,125
```

**Figura 9-25 - Sessão de mapeamento de sinal exportada no formato CSV**

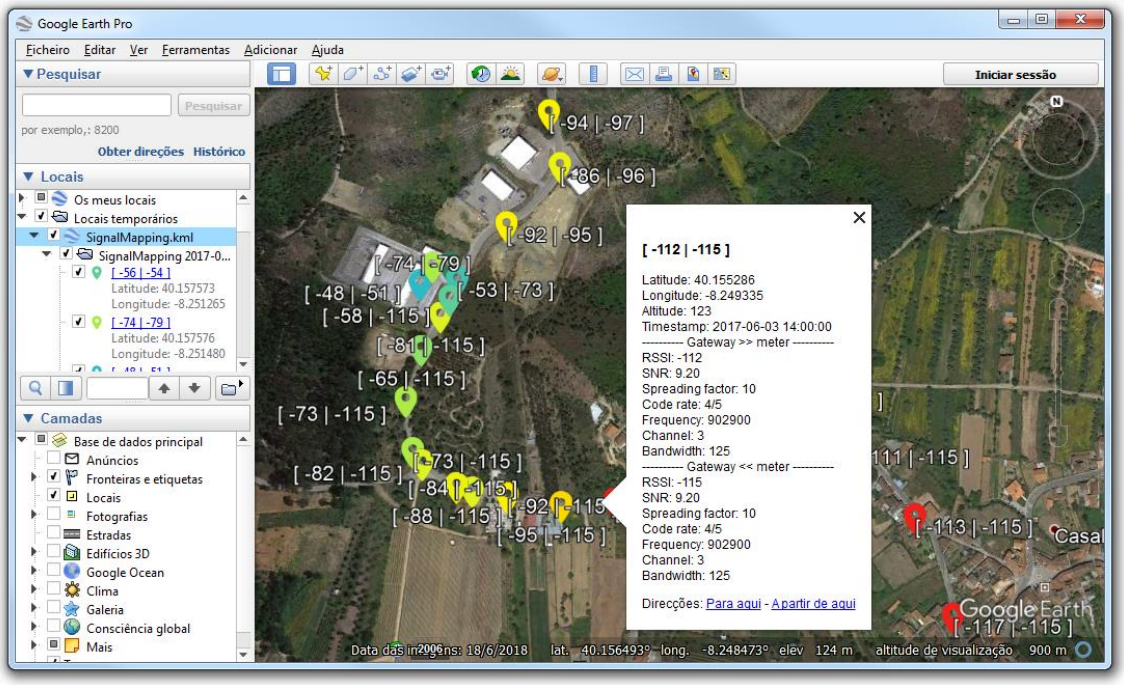



**Figura 9-26 - Sessão de mapeamento de sinal exportada no formato KML**

### 9.5.3. Exportar outros dados

Alguns dados da base de dados principal, tais como modelos de contador ou contadores, podem ser exportados no formato JSON. Mais tarde é possível importar esses dados na mesma instância do *website*, noutra instância do *website* ou noutras aplicações. Deste modo, os utilizadores podem transferir dados entre instâncias do *website* de administração, importar dados de aplicações externas, exportar dados para aplicações externas e efetuar cópias de segurança rudimentares para efeitos de testes. A Figura 9-27 mostra a listagem de modelos de contador, onde se pode ver os botões "Import" e "Export" que permitem importar e exportar dados respetivamente.

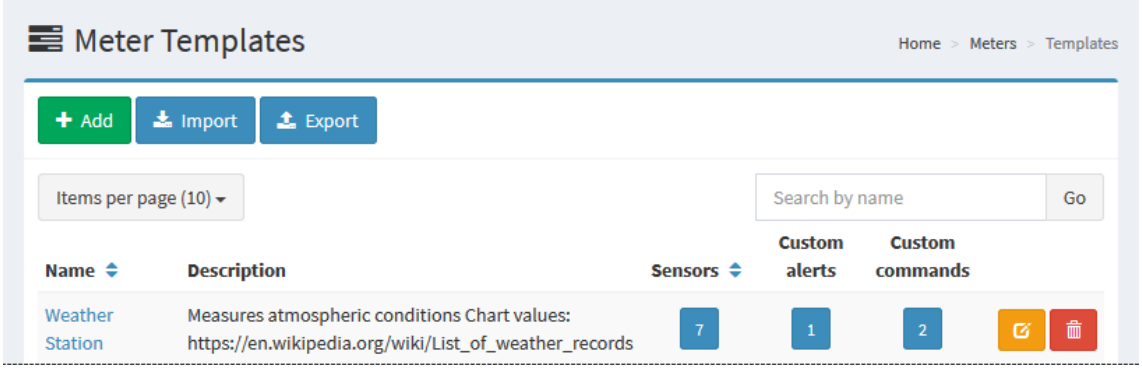



**Figura 9-27 - Botões de importação e exportação de dados dos modelos de contador**

A Figura 9-28 mostra os dados de um modelo de contador, neste caso o modelo de estação meteorológica apresentado na Figura 9-1, exportados no formato JSON. São exportados os dados do modelo, tais como o nome a apresentar, o nome da marca do contador, o nome do modelo do contador, descrição, sensores, comandos personalizados e alertas associados.

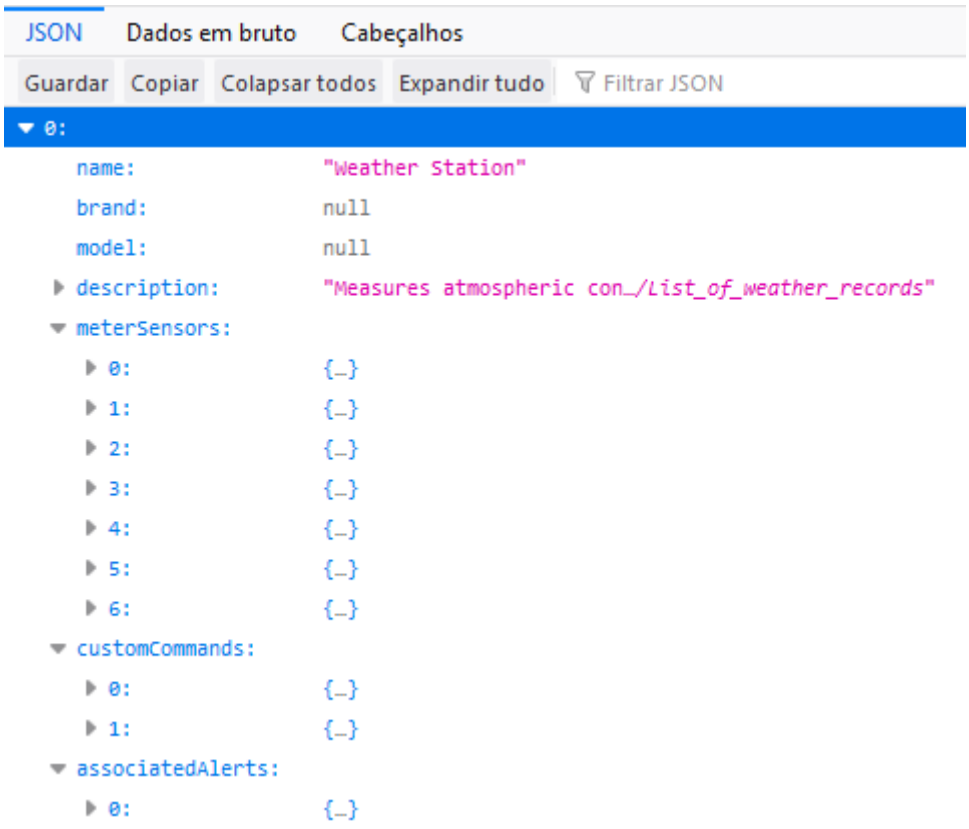



**Figura 9-28 - Modelo de contador exportado no formato JSON**

# 10. Segurança

Devido ao facto de que as comunicações entre os contadores e os *gateways* são feitas sem fios, por rádio, que as comunicações entre os *gateways* e o Eclipse Mosquitto são feitas pela Internet, e que as comunicações entre o *website* e os *web* browsers também são feitas pela Internet, a segurança das comunicações torna-se uma prioridade. Adicionalmente, cada componente de *software* e *hardware* deve ser protegido de modo a evitar o acesso não autorizado ao sistema. A segurança de um sistema assenta principalmente nos seguintes princípios:

- Autenticação - verificar se o acesso a um recurso está a ser feito realmente pela entidade que afirma ser.
- Autorização - verificar se a entidade tem permissão para aceder a um determinado recurso.
- Confidencialidade - proteger informações de acessos de entidades não autorizadas.
- Integridade - assegurar que as informações não foram alteradas durante a transmissão ou quando armazenadas num sistema.
- Disponibilidade – assegurar que um sistema é capaz disponibilizar informações quando solicitadas, mesmo durante uma eventual falha de um ou mais componentes de *hardware* ou *software*.
- Não repúdio – assegurar que o verdadeiro remetente de uma mensagem não pode negar ter enviado a mensagem.

## 10.1. LoRaWAN

A segurança de uma rede LoRa é algo que está incluído na especificação LoRaWAN. Todas as mensagens enviadas pela rede estão protegidas por duas chaves e existem mecanismos que protegem contra certo tipo de ataques, por exemplo, repetição de mensagens. É importante proteger a rede LoRa pois a área geográfica coberta por esta rede será significativa. Tomando como exemplo a rede LoRa da empresa distribuidora de água para a qual este projeto foi desenvolvido, poderá ser feito um ataque a partir de qualquer local em que a empresa possua clientes.

### 10.1.1. OTAA

O método de associação dos dispositivos com o servidor de rede LoRa utilizado foi o "Over-The-Air Activation" (OTAA). Este método define que devem ser trocadas mensagens de associação entre o dispositivo e o servidor de rede LoRa cada vez que a sessão é perdida (normalmente quando o dispositivo reinicia). A associação é feita entre o servidor de rede LoRa e os dispositivos de modo a permitindo o *roaming* entre vários *gateways* ou interfaces de rádio LoRa.

### 10.1.2. Mensagens de associação

Quando um dispositivo LoRa pretende associar-se a uma rede LoRa, envia uma mensagem a solicitar a associação à rede (join-request), segundo a especificação LoRaWAN deveria ser o servidor de rede a responder a esta mensagem, mas a documentação do LoRa App Server indica que nesta implementação o servidor de

aplicação é que trata do processo de associação e responde às mensagens de associação. A mensagem de associação é composta pelo identificador da aplicação (AppEUI), identificador do dispositivo (DevEUI) e um valor aleatório (DevNonce). O valor aleatório serve para evitar ataques em que a mensagem de associação é repetida (replay attack), com a intenção de bloquear a comunicação de dispositivos com a rede. O servidor (nesta implementação o LoRa App Server) mantém uma lista de valores utilizados anteriormente por cada dispositivo e ignora repetições.

Se o dispositivo estiver autorizado a aceder à rede, segundo a especificação LoRaWAN, o servidor de rede deveria responder com uma mensagem de confirmação (join-accept) mas, segundo a documentação do LoRa App Server, é o servidor de aplicação que responde a esta mensagem. Caso o dispositivo não esteja autorizado a associar-se à rede o LoRa App Server não responde ao pedido de associação. O campo mais importante da mensagem de confirmação é o valor aleatório de aplicação (AppNonce) enviado pelo servidor, a partir do qual o dispositivo consegue gerar uma chave de sessão de aplicação e uma chave de sessão de rede única por sessão. Também é enviado o identificador da rede (NetID), o endereço do dispositivo (DevAddr), a duração do intervalo de tempo entre a transmissão de mensagens pelos dispositivos e a abertura dos intervalos de receção de mensagens dos *gateways* (RxDelay) e uma lista de canais e frequências (CFList).

### 10.1.3. Chave de aplicação

A chave de aplicação (AppKey) é uma chave AES de 128 bits que serve como base para gerar outras chaves utilizadas para encriptar as comunicações com os dispositivos. A partir desta chave e de um valor aleatório enviado pelo servidor de rede LoRa (na implementação atual é o LoRa App Server que gera o valor aleatório) é gerada a chave de sessão de aplicação (AppSKey) e a chave de sessão de rede (NwkSKey). A chave de aplicação deve ser configurada no servidor de aplicação LoRa (associando o EUI do dispositivo à chave de aplicação) e no dispositivo. Como a chave de aplicação é única por dispositivo, se um atacante obtiver a chave de aplicação de um dispositivo, não vai poder ver ou modificar mensagens trocadas com outros dispositivos da mesma aplicação.

### 10.1.4. Chave de sessão de aplicação

A chave de sessão de aplicação (AppSKey) é uma chave AES de 128 bits que é utilizada para encriptar as mensagens trocadas entre o servidor de aplicação LoRa e os dispositivos. Para gerar esta chave primeiro é concatenado um byte fixo com valor 0x02, com um valor aleatório enviado pelo servidor (AppNonce), com o identificador da rede (NetID) e com um valor aleatório enviado pelo dispositivo (DevNonce) e depois o resultado da concatenação é encriptado com o algoritmo AES utilizando a chave de aplicação. De seguida é apresentada a função que gera a chave de sessão de aplicação.

$$AppSKey = aes128\_encrypt(AppKey, 0x02 \mid AppNonce \mid NetID \mid DevNonce \mid pad_{16})$$

O valor "$pad_{16}$" é o resultado de uma função que adiciona zeros ao final dos dados até que o número de bits dos dados seja um múltiplo de 16.

A função da chave de sessão aplicação é evitar que operadores de uma rede LoRa consigam visualizar ou modificar mensagens trocadas entre a aplicação e os

dispositivos. Uma analogia a esta proteção extra seria uma rede Wi-Fi de um restaurante que é protegida por uma chave de rede (PSK) e que é utilizada por um cliente desse restaurante para aceder a um *website* pelo protocolo HTTPS. Embora o administrador da rede possa obter o tráfego entre o dispositivo do cliente e o ponto de acesso Wi-Fi, não vai poder visualizar ou modificar o tráfego para o *website*.

### 10.1.5. Chave de sessão de rede

A chave de sessão de rede (NwkSKey) é uma chave AES de 128 bits que é utilizada para encriptar e verificar as mensagens entre os dispositivos e o servidor de rede LoRa, fornecendo confidencialidade e integridade dos dados. É gerada de forma semelhante à chave de sessão de aplicação diferindo apenas num bit no byte fixo, na chave de sessão de aplicação o byte fixo tem o valor 0x02 enquanto que na chave de sessão de rede o byte fixo tem o valor 0x01, esta diferença faz com que a chave de sessão de rede gerada através do algoritmo AES seja diferente da chave de sessão de aplicação. De seguida é apresentada a função que gera a chave de sessão de rede.

NwkSKey = aes128_encrypt(AppKey, 0x01 | AppNonce | NetID | DevNonce | $pad_{16}$)

A função desta chave é proteger a rede LoRa de utilizadores não autorizados, evitando que possam ver ou modificar mensagens dos dispositivos ligados à rede.

## 10.2. Certificados CA

Um certificado CA é um certificado especial em que uma determinada máquina confia, a partir do qual são assinados outros certificados utilizados para proteger dados ou comunicações. A segurança das comunicações entre os servidores utilizados neste sistema depende principalmente da segurança das chaves privadas dos certificados CA. Se um atacante obtiver a chave privada do certificado de um determinado serviço (p. ex.: IIS) vai poder ver e modificar todo o tráfego desse serviço, já que vai poder desencriptar, visualizar/modificar e encriptar de novo todo o tráfego do serviço utilizando um novo certificado assinado pelo certificado CA comprometido.

De modo a evitar que uma única chave privada possa comprometer todas as comunicações dos servidores do sistema, foram geradas chaves privadas RSA de 2048 bits para cada tipo de máquina e a partir dessas chaves foram criados certificados CA para cada tipo de máquina. Foi então criado um certificado para o nome DNS de cada máquina, assinado com o certificado CA desse tipo de máquina. Todos os certificados criados (incluindo os certificados CA) foram assinados com o algoritmo SHA-2 de 256 bits. A Tabela 10-1 mostra os vários tipos de máquinas e os serviços que utilizam certificados para proteger as suas comunicações.

| Tipo de máquina | Número | Serviços |
|---|---|---|
| IIS | 1 | *Website* e *Web Service* |
| Node-RED | 2 | Antigo *software* dos gateways |
| LoRa App Server | 3 | Servidor de aplicação LoRa e Eclipse Mosquitto |
| LoRa Server | 4 | Servidor de rede LoRa e Eclipse Mosquitto |

**Tabela 10-1 - Tipos de máquinas e respetivos serviços que utilizam certificados**

Os certificados CA foram instalados máquinas do mesmo tipo, por exemplo o certificado CA para o tipo de máquinas "Node-RED" foi instalado na máquina virtual do Node-RED local e na máquina virtual do Node-RED de demonstração. De modo a que uma máquina possa validar a autenticidade dos certificados de serviços externos, também foram instalados certificados CA nas máquinas que utilizam esses serviços. A Figura 10-1 mostra um diagrama dos tipos de máquinas e respetivos certificados CA instalados, os números correspondem aos tipos de máquina da Tabela 10-1.

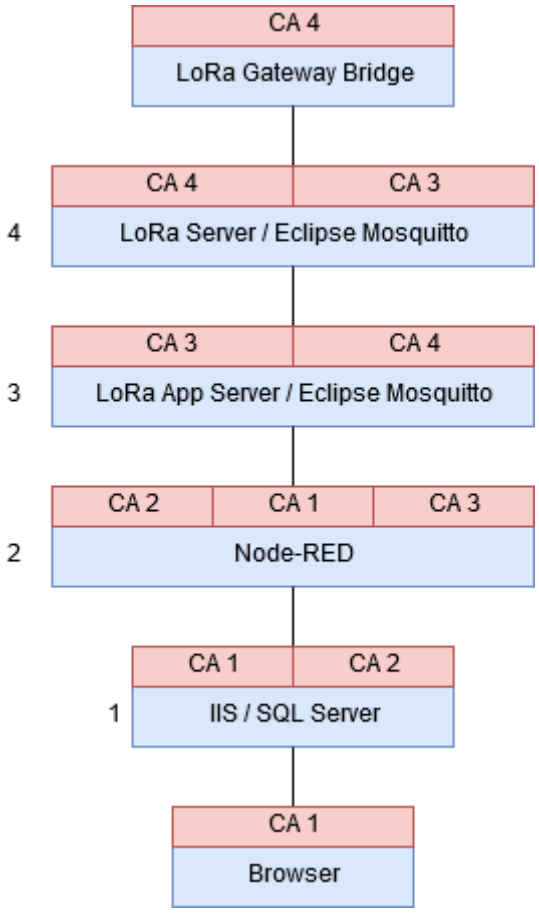



**Figura 10-1 - Distribuição dos certificados CA**

Em alternativa poderiam ser utilizados certificados auto-assinados, eliminando a necessidade de utilizar certificados CA, mas cada vez que um certificado fosse alterado (p. ex.: quando o domínio DNS fosse alterado) seria necessário configurar manualmente cada sistema que se liga a cada serviço para confiar nos novos certificados. Embora seja possível automatizar este processo, por exemplo através de *scripts*, a utilização de

certificados CA que são instalados nas máquinas que fornecem e utilizam os serviços, permite trocar o certificado desses serviços apenas nos servidores (p. ex.: quando é necessário mudar o nome DNS ou renovar certificados que expiraram dentro de pouco tempo), pois os clientes podem verificar a autenticidade dos novos certificados através dos certificados CA com que foram assinados. Adicionalmente, se for necessário adicionar máquinas de um determinado tipo para dividir a carga e eliminar *bottlenecks* (p. ex.: adicionar outra máquina para servir mais instâncias de Node-RED), basta criar um novo certificado assinado pelo certificado CA desse tipo de máquina que os clientes passam a confiar automaticamente nos serviços da nova máquina.

## 10.3. TLS

O protocolo TLS é um protocolo criptográfico que fornece formas de proteger as comunicações entre máquinas numa rede de computadores, encriptando o tráfego de aplicações. Este protocolo encripta todos os dados enviados pela rede por aplicações com um algoritmo criptográfico de chave simétrica, sendo que a chave utilizada para encriptar e desencriptar os dados é negociada entre os intervenientes através de um algoritmo que utiliza criptografia de chave assimétrica. A criptografia de chave assimétrica utiliza um par de chaves público/privada, em que um interveniente fornece a sua chave pública a todos os intervenientes que pretendam estabelecer uma ligação segura e a respetiva chave privada é mantida em segredo, na posse do interveniente. Este método é seguro pois a chave pública de cada interveniente serve apenas para encriptar os dados, para desencriptar os dados é necessário possuir a chave privada, portanto as chaves simétricas negociadas só são conhecidas pelos intervenientes, tornando os dados encriptados muito difíceis de desencriptar por um atacante.

O protocolo TLS é utilizado para proteger as comunicações entre todos os componentes de *software* deste projeto à exceção da comunicação entre os contadores e os *gateways*, que utilizam um mecanismo de segurança próprio da especificação LoRaWAN.

## 10.4. Eclipse Mosquitto

O Eclipse Mosquitto foi configurado para encriptar todo o tráfego MQTT com recurso ao protocolo TLS, deste modo todas as mensagens trocadas com os contadores não podem ser visualizadas ou modificadas por terceiros.

A autenticação é feita através de utilizador e palavra-passe, pois este é um dos métodos de autenticação do protocolo MQTT e o Node-RED só suporta este tipo de autenticação. Foi criado um utilizador por cada máquina que se liga em cada instancia do Eclipse Mosquitto. Por exemplo, na instância do Eclipse Mosquitto que serve de intermediário entre o Node-RED e o LoRa App Server foram criados os utilizadores “node-red” e “app-server” que são utilizados respetivamente por estes.

A autorização é feita através de ACLs que controlam o acesso aos tópicos criados no Eclipse Mosquitto. Por exemplo, na instância que serve de intermediário entre o Node-RED e o LoRa App Server foi criada uma ACL de escrita para o tópico “application/+/device/+/tx” para o utilizador “node-red”, pois este é o tópico por omissão que o Node-RED (ou outro cliente MQTT) deve utilizar para enviar mensagens para os contadores através do LoRa App Server. Cada utilizador/cliente do Eclipse Mosquitto só pode subescrever aos tópicos permitidos pela ACL.

## 10.5. LoRa App Server

A comunicação do LoRa App Server com o LoRa Server é feita através de gRPC. O LoRa App Server foi configurado para encapsular a API gRPC com o protocolo TLS, protegendo deste modo as mensagens trocadas com os contadores e as chamadas de procedimento remoto que são trocada com o LoRa Server.

O LoRa App Server requer software adicional para funcionar, nomeadamente os SGBD PostgreSQL e Redis, que embora se encontrem na mesma máquina virtual que o LoRa App Server e não estejam acessíveis pela rede, têm o acesso protegido por utilizador e palavra-passe, de modo a que se um atacante conseguir obter acesso não privilegiado à maquina, não consiga efetuar alterações às bases de dados do LoRa App Server. O Eclipse Mosquitto também foi instalado nesta máquina para servir de mediador entre o LoRa App Server e o Node-RED, embora o Eclipse Mosquitto se encontre na mesma máquina que o LoRa App Server a ligação entre estes é feita através do protocolo TLS, pois o Eclipse Mosquitto está configurado para proteger as suas comunicações por TLS para proteger a comunicação com o Node-RED que se encontra numa máquina diferente na rede.

## 10.6. Node-RED

O Node-RED foi configurado para servir o IDE e o *web service* através do protocolo HTTPS, deste modo tanto as mensagens trocadas com os contadores como a interface de desenvolvimento são protegidas, desta forma se o tráfego for intercetado entre o cliente e o servidor, não será possível ver ou modificar o seu conteúdo sem que o cliente e o servidor tomem conhecimento.

### 10.6.1. IDE

O acesso ao IDE do Node-RED é limitado através de contas de utilizador. A autenticação é feita através de um nome de utilizador e palavra-passe e a autorização é feita através de um campo de permissões. É possível restringir o acesso aos nós, aos fluxos e às configurações, no entanto foi criada apenas uma conta com acesso total de leitura e escrita e outra conta com acesso apenas de leitura. A Figura 10-2 mostra a interface de login apresentada quando um utilizador tenta aceder ao IDE do Node-RED.

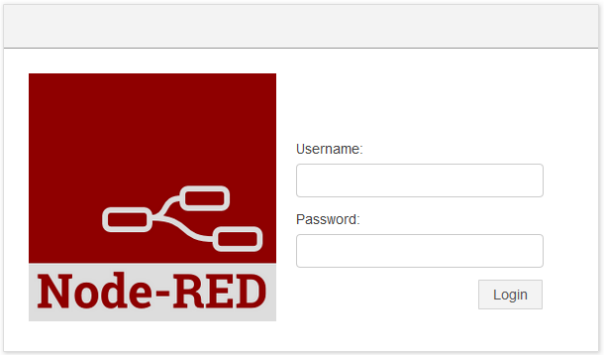



**Figura 10-2 - Login do IDE do Node-RED**

### 10.6.2. Web Service

Como o protocolo HTTP não possui estado (ou sessão) torna-se necessário provar a identidade em cada pedido. Foi utilizado o método de autenticação "Basic" do protocolo HTTP (RFC 7617)[13] que não necessita de *cookies* de sessão e outros mecanismos mais complexos, utilizando um método standard torna-se mais fácil integrar com outros sistemas.

Como os nós de entrada HTTP do Node-RED que implementam o *web service* não suportam qualquer tipo de autenticação, foi implementado o método de autenticação Basic no Node-RED. Cada instância do sistema desenvolvido em Node-RED possui um identificador único de *gateway* e uma palavra-passe de *web service* que são utilizados como utilizador e palavra-passe do método de autenticação Basic. O identificador único de *gateway* e a palavra-passe do *web service* podem ser alterados a partir do *website* de administração, adicionalmente o nó de função que implementa o método de autenticação Basic pode ser facilmente modificado no IDE do Node-RED de modo a implementar outros métodos de autenticação. A Figura 10-3 mostra como o valor que é enviado no cabeçalho de autenticação é gerado.

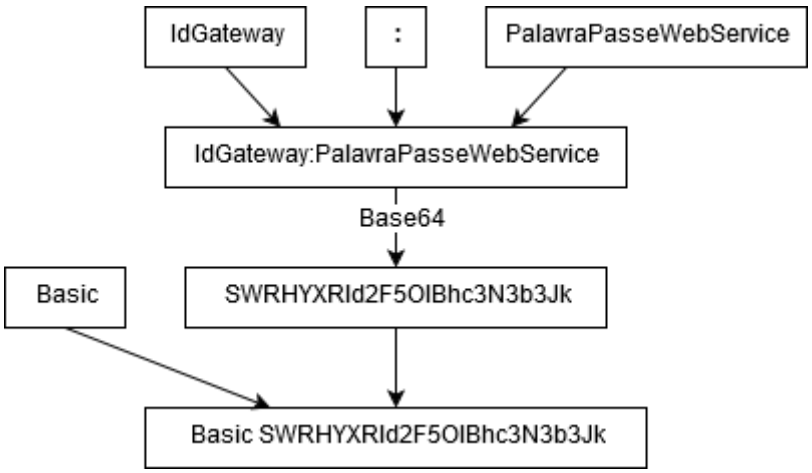



**Figura 10-3 - Construção do cabeçalho de autenticação HTTP**

Este valor (Basic SWRHYXRId2F5OlBhc3N3b3Jk) é então inserido no cabeçalho de autenticação. A Figura 10-4 mostra um pedido para o *web service* no Node-RED capturado com o analisador de pacotes Wireshark. Este pedido foi efetuado pelo protocolo HTTP de modo a ser possível visualizar facilmente o seu conteúdo, o cabeçalho de autenticação está destacado a vermelho.

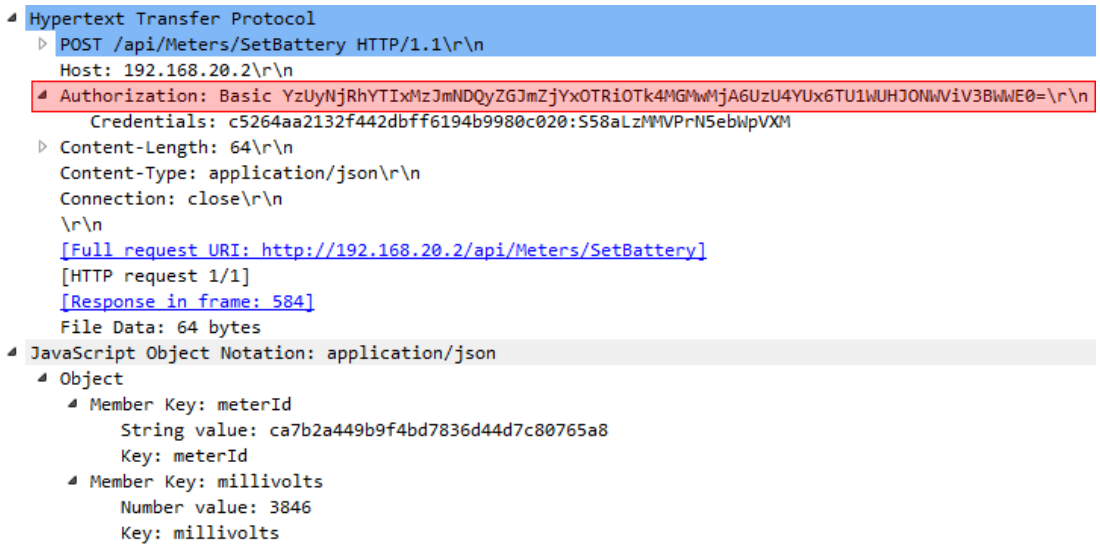



**Figura 10-4 - Pedido HTTP para o web service no Node-RED**

O nó de função que autentica pedidos HTTP para o *web service* no Node-RED executa os passos de construção do cabeçalho de autenticação em ordem inversa, de modo a obter o identificador único do *gateway* e a palavra-passe do *web* service e a verificar se estes correspondem aos valores configurados. O Wireshark realiza esta operação automaticamente, mostrando as credenciais no formato "Utilizador:Palavra-passe" no campo "Credentials". Existe uma palavra-passe diferente para cada instância do *web service* no Node-RED, garantindo que se um *gateway* for comprometido não seja possível controlar ou configurar todos os outros *gateways* e contadores do sistema.

# 10.7. IIS

O IIS foi configurado para servir o *web service* e o *website* através do protocolo HTTPS, de modo a proteger as mensagens dos contadores trocadas com o Node-RED e a o *website* administração, desta forma não é possível ver ou modificar o tráfego entre o Node-RED e o *web service* no IIS, pois os pedidos HTTPS vão falhar, ou ver ou modificar o tráfego entre o *web browser* e o *website* de administração, pois o *web browser* vai alertar o utilizador.

O certificado CA utilizado para assinar certificados de instâncias do IIS deve ser instalado nos sistemas operativos ou nos *browsers* das máquinas que se pretende utilizar para aceder ao *website* de administração, caso contrário será mostrada uma mensagem que avisa que a ligação é insegura. Ao instalar o certificado CA no cliente a mensagem que avisa que a ligação é insegura deixa de aparecer. É importante não habituar utilizadores do *website* a ignorar esta mensagem, pois caso a ligação entre o IIS e o *web browser* seja mesmo comprometida o utilizador vai ignorar a mensagem, iniciar sessão no *website* e utilizá-lo sem preocupações.

## 10.7.1. Web Service

A autenticação do *web service* no IIS é semelhante à autenticação do *web service* no Node-RED, é utilizado o esquema de autenticação Basic, como explicado na seção sobre a segurança do *web service* no Node-RED (10.6.2). O identificador único do *gateway* é também utilizado como nome de utilizador, mas é utilizada uma palavra-

passe diferente por cada *gateway* (instância do Node-RED), distinta da palavra-passe utilizada para autenticar com o *web service* no Node-RED. Deste modo, se um *gateway* for comprometido um atacante só pode obter configurações desse *gateway* e enviar leituras/alertas/etc. "em nome" dos contadores associados ao mesmo, não interferindo com outros contadores.

### 10.7.2. Website

A autenticação e autorização do *website* de administração são baseadas no sistema ASP.NET Identity. Este é o sistema de autenticação e autorização mais recente da Microsoft para *websites* baseados na framework ASP.NET.

A autenticação é feita com recurso a um par utilizador/palavra-passe, por ser um requisito, no entanto este sistema suporta também autenticação de dois fatores, outros tipos de autenticação (p. ex.: *tokens*) e pode ser integrado com outros sistemas de autenticação se necessário. As sessões expiram após quarenta minutos de inatividade, impedindo que os utilizadores se esqueçam de sessões iniciadas num determinado web *browser*. Consequentemente, sessões de tempo limitado reduzem a utilidade dos *cookies* de sessão caso sejam furtados por algum tipo de vírus que se encontre no sistema em que o utilizador iniciou a sessão, pois a sessão expira se não for carregada uma página a cada quarenta minutos.

A autorização foi implementada com três níveis de permissão de acesso aos dados do sistema. O nível mais baixo (*read only*) permite apenas acesso de leitura aos dados de um *tenant*, exceto os dados dos outros utilizadores, *tenants* e configurações. Um utilizador com este nível de permissão pode apenas ver as leituras dos contadores, configurações dos contadores, configurações dos *gateways*, alertas e notificações. A Tabela 10-2 mostra as permissões do nível "*read only*" em detalhe.

| | Ver/Listar | Adicionar | Modificar | Apagar | Controlar |
|---|---|---|---|---|---|
| Dashboard | | | | | |
| Utilizadores | | | | | |
| Tenants | | | | | |
| Contadores | | | | | |
| Gateways | | | | | |
| Leituras | | | | | |
| Alertas | | | | | |
| Notificações | | | | | |
| Configurações | | | | | |

**Tabela 10-2 - Permissões do nível de acesso "read only"**

O nível intermédio (*administrator*) permite acesso total aos dados do *tenant* a que o utilizador pertence, adicionalmente permite gerir utilizadores do próprio *tenant*, controlar e configurar contadores e *gateways* remotamente e alterar configurações do sistema. A Tabela 10-3 mostra as permissões do nível "*administrator*" em detalhe.

| | Ver/Listar | Adicionar | Modificar | Apagar | Controlar |
|---|---|---|---|---|---|
| Dashboard | | | | | |
| Utilizadores | | | | | |
| Tenants | | | | | |
| Contadores | | | | | |
| Gateways | | | | | |
| Leituras | | | | | |
| Alertas | | | | | |
| Notificações | | | | | |
| Configurações | | | | | |

**Tabela 10-3 - Permissões do nível de acesso "administrator"**

O nível mais alto (*super administrator*) permite gerir todos os dados de todos os *tenants*, adicionalmente este é o único nível de autorização que permite gerir os tenants do sistema. A Tabela 10-4 mostra as permissões do nível "super administrator" em detalhe.

| | Ver/Listar | Adicionar | Modificar | Apagar | Controlar |
|---|---|---|---|---|---|
| Dashboard | | | | | |
| Utilizadores | | | | | |
| Tenants | | | | | |
| Contadores | | | | | |
| Gateways | | | | | |
| Leituras | | | | | |
| Alertas | | | | | |
| Notificações | | | | | |
| Configurações | | | | | |

**Tabela 10-4 - Permissões do nível de acesso "super administrator"**

# 10.8. Bases de dados

O acesso às bases de dados está protegido com uma conta de utilizador que só tem acesso à base de dados principal e à base de dados de leitura. A conta de utilizador foi criada no próprio SQL Server. Este modo de autenticação é chamado "SQL Authentication". É mais simples do que o modo de autenticação alternativo "Windows Authentication" pois não é necessário criar uma conta de utilizador na máquina em que o SQL Server está alojado e configurar as permissões dessa conta para limitar o acesso ao sistema operativo.

As leituras de contadores são armazenadas numa base de dados dedicada. Ao armazenar as leituras numa base de dados separada é possível utilizar serviços de armazenamento de terceiros para guardar leituras de forma segura, pois além dos valores das leituras e dos identificadores dos contadores não existem dados sensíveis nesta base de dados, não sendo possível descobrir o que os contadores estão a medir ou a sua localização apenas com esta base de dados. Todos os outros dados da aplicação tais como logins, dados de localização, chaves de autenticação dos *web services*, detalhes dos contadores, *logs*, etc. são guardados na base de dados principal e como ocupam pouco espaço relativamente à base de dados de leituras, podem ser armazenados em servidores da própria empresa.

## 10.9. Sistemas operativos

O acesso aos sistemas operativos das máquinas virtuais que possuem serviços e dos *gateways* foi protegido com recurso a utilizador e palavra-passe. O acesso remoto por SSH dos *gateways* e das máquinas virtuais com o sistema operativo Linux e o acesso remoto por RDP do servidor com sistema operativo Windows só pode ser feito através de uma VPN, utilizando o *software* OpenVPN. As *firewalls* das máquinas virtuais e dos *gateways* foram configuradas para bloquear todo o tipo tráfego de entrada e saída à exceção do tráfego absolutamente necessário (p. ex.: DNS) e do tráfego dos vários serviços (p. ex.: LoRa App Server). Atualizações de segurança são aplicadas automaticamente nas máquinas virtuais. Atualizações dos sistemas operativos só são aplicadas alguns meses depois de serem publicadas de modo a evitar *bugs*, ter tempo para testar o *software* desenvolvido com a nova versão do sistema operativo e encontrar eventuais incompatibilidades.

## 10.10. Hardware

Os contadores de água estão protegidos com lacre, tal como os contadores mecânicos, de modo a detetar tentativas de abrir o contador. Adicionalmente o acelerómetro deteta tentativas de mover o contador e outros sensores detetam tentativas de fraude de leituras.

Os *gateways* também estão protegidos com lacre, estes serão instalados em locais com acesso restrito, normalmente no andar mais elevado ou no telhado de edifícios de modo a aumentar a linha de vista para aumentar a cobertura do sinal LoRa e diminuir o comprimento do cabo de antena (caso seja necessário) para diminuir perdas de sinal.

Os servidores estão protegidos pela Amazon Web Services. Segundo esta, os *data centers* foram construídos em locais com risco de catástrofes naturais reduzido, têm planos para reduzir o impacto de catástrofes naturais, restringem o acesso físico aos servidores, permitindo que os funcionários acedam apenas a áreas e servidores necessários para executar o seu trabalho, possuem sistema de videovigilância, equipa de segurança, deteção de intrusão, proteção contra incêndios, fugas de água, entre outros.[22]

As máquinas utilizadas para ligar ao *website* de administração devem possuir as atualizações de segurança mais recentes. As contas de utilizadores não devem possuir privilégios elevados e se for necessário guardar palavras-passe para aceder ao *website* deve ser utilizado um gestor de palavras-passe em vez do *web browser*.

# 11. Testes

Foi montada uma versão completa do sistema para testes, utilizando as máquinas virtuais apresentadas na seção 6.3, os *gateways* apresentados na seção 6.2, e os contadores apresentados na seção 6.1. Este capítulo apresenta os principais testes que foram efetuados de modo a certificar que o sistema funciona sem problemas.

## 11.1. Contadores

Foram feitos vários tipos de testes aos contadores. Os testes em que participei foram testes de codificação e descodificação de mensagens, alteração de configurações, execução de comandos e cobertura do sinal LoRa.

Os testes de codificação de mensagens pelos contadores consistiam em enviar mensagens a partir de um contador de testes, tentar descodificá-las no Node-RED e verificar se o tipo de mensagem e os valores obtidos correspondiam ao que o *software* do contador de testes pretendia enviar. De forma inversa, os testes de descodificação de mensagens pelos contadores consistiam em enviar mensagens a partir do Node-RED, tentar descodificá-las com o *software* do contador de testes e verificar se o tipo de mensagem e os valores obtidos correspondiam ao que o Node-RED pretendia enviar.

Os testes de alteração de configurações consistiam em efetuar alterações à configuração de um contador de testes a partir do *website* de administração, verificar se essas alterações foram aplicadas na memória do contador e se produziram o efeito pretendido.

Os testes de execução de comandos consistiam em executar comandos num contador de testes a partir do *website* de administração, verificar se o comando foi executado no contador e verificar a resposta do contador.

Os testes de cobertura de sinal consistiam em instalar um *gateway* LoRa com acesso à Internet através de rede móvel 4G num local a testar e conduzir um automóvel transportando o contador de mapeamento de sinal LoRa pelos arredores do *gateway* até que a comunicação por LoRa falhe, enquanto alguém no escritório efetuava pedidos de mapeamento de sinal constantemente. O condutor do automóvel efetuava uma chamada em alta voz para quem ficava no escritório, de modo a ser informado das condições do sinal e alterar o trajeto.

## 11.2. Web Services

Os testes dos *web services* foram feitos com o *software* Postman, sendo criada uma coleção de pedidos e respostas para o *web service* no IIS e outra para o *web service* no Node-RED.

A Figura 11-1 mostra um teste no Postman para o *web service* no IIS. Neste caso o Postman atua como um *gateway* que pretende obter informações sobre os contadores que lhe estão associados. Este teste verifica se o código HTTP da resposta indica que a operação foi bem-sucedida, se o tipo de conteúdo está corretamente definido, se o esquema JSON da resposta é válido, se a resposta inclui informações de pelo menos um contador e se o pedido foi respondido em menos de um segundo.

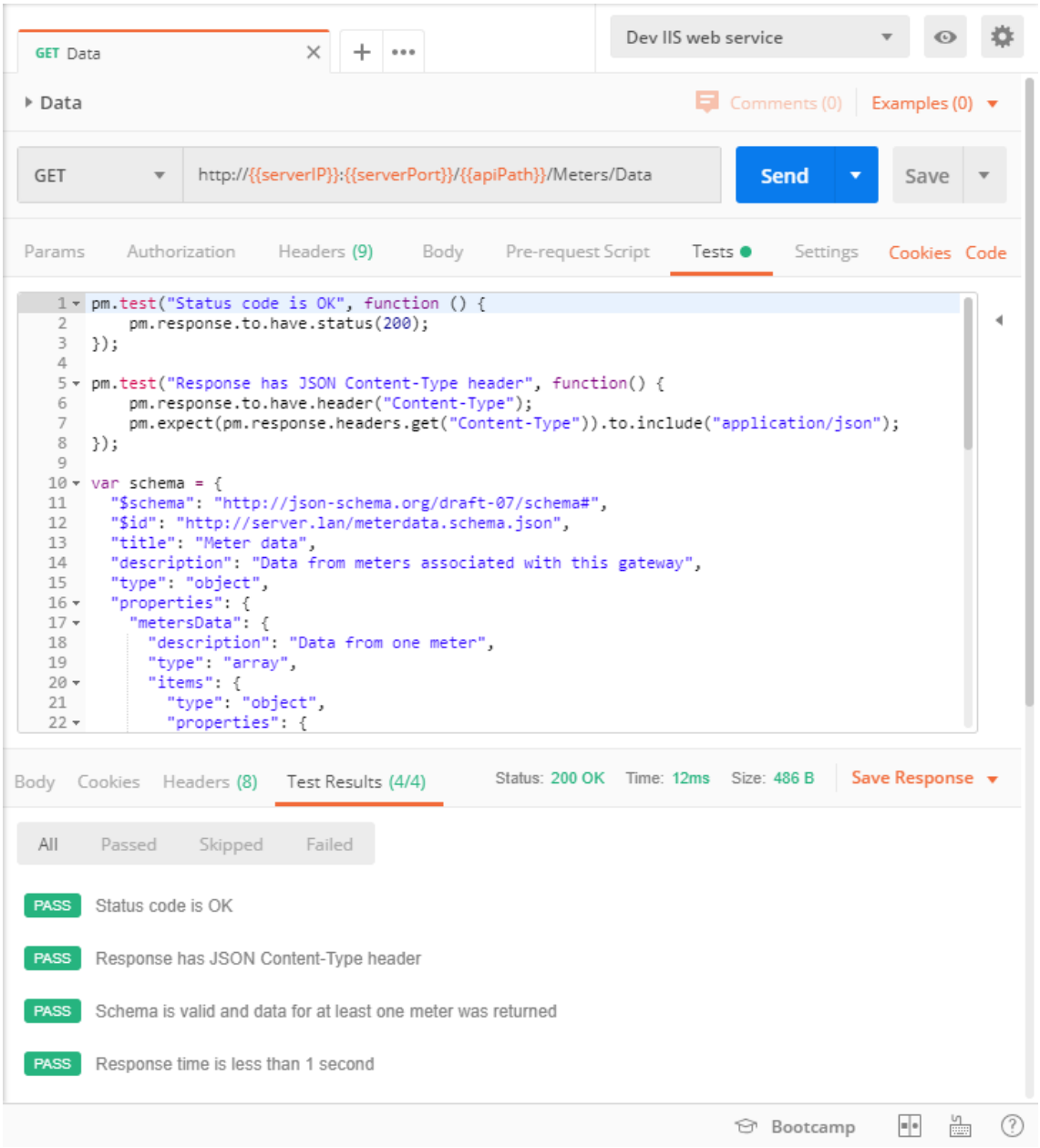
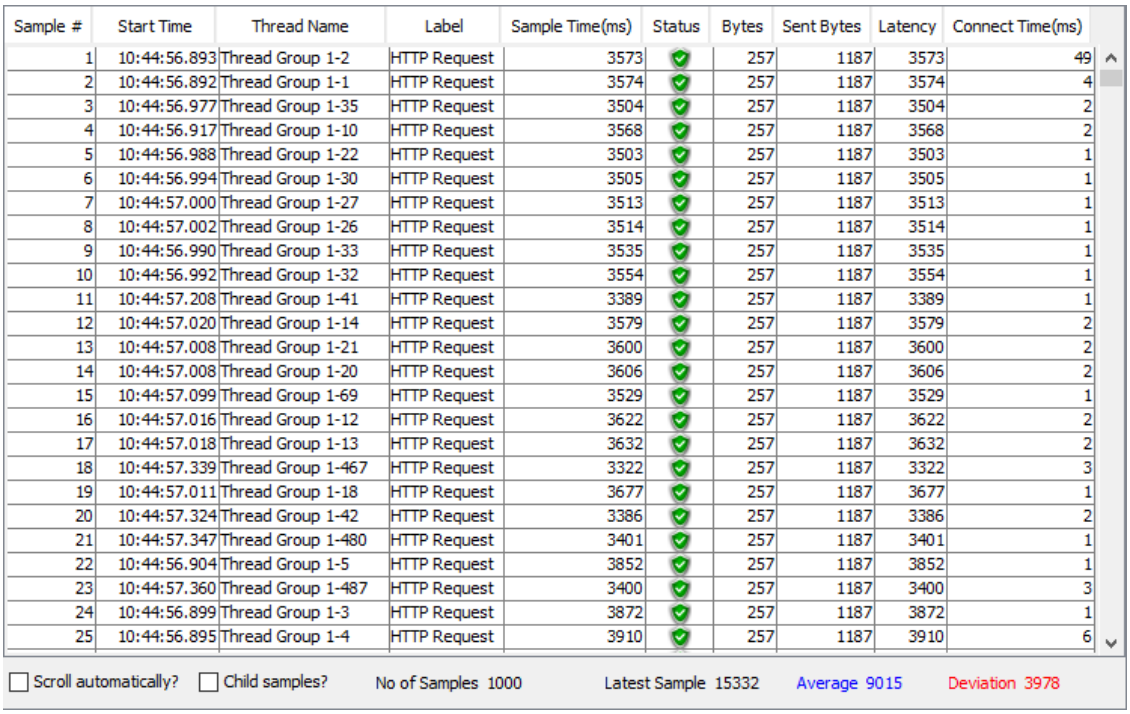



**Figura 11-1 - Teste para o *web service* no IIS**

Também foram efetuados testes de carga para determinar se as implementações atuais dos *web services* conseguem processar e responder a uma grande quantidade de pedidos feitos simultaneamente.

A Figura 11-2 mostra um exemplo teste de carga para o *web service* no IIS. A Figura 11-3 mostra as estatísticas desse teste. O teste consiste em enviar mil pedidos HTTP para o método que insere leituras de um determinado contador de forma associativa (ver Anexo D: Especificação do *web service* no IIS), em que cada pedido contém dez leituras de um contador. Este é um dos métodos mais "pesados" do *web service* pois possui o esquema JSON mais complexo, efetua várias consultas na base de dados principal e na base de dados de leituras antes de efetuar a operação e publica as novas leituras para os *web browsers* dos utilizadores que tenham subscrito às leituras em tempo real do contador.



| Sample # | Start Time | Thread Name | Label | Sample Time(ms) | Status | Bytes | Sent Bytes | Latency | Connect Time(ms) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 10:44:56.893 | Thread Group 1-2 | HTTP Request | 3573 | | 257 | 1187 | 3573 | 49 |
| 2 | 10:44:56.892 | Thread Group 1-1 | HTTP Request | 3574 | | 257 | 1187 | 3574 | 4 |
| 3 | 10:44:56.977 | Thread Group 1-35 | HTTP Request | 3504 | | 257 | 1187 | 3504 | 2 |
| 4 | 10:44:56.917 | Thread Group 1-10 | HTTP Request | 3568 | | 257 | 1187 | 3568 | 2 |
| 5 | 10:44:56.988 | Thread Group 1-22 | HTTP Request | 3503 | | 257 | 1187 | 3503 | 1 |
| 6 | 10:44:56.994 | Thread Group 1-30 | HTTP Request | 3505 | | 257 | 1187 | 3505 | 1 |
| 7 | 10:44:57.000 | Thread Group 1-27 | HTTP Request | 3513 | | 257 | 1187 | 3513 | 1 |
| 8 | 10:44:57.002 | Thread Group 1-26 | HTTP Request | 3514 | | 257 | 1187 | 3514 | 1 |
| 9 | 10:44:56.990 | Thread Group 1-33 | HTTP Request | 3535 | | 257 | 1187 | 3535 | 1 |
| 10 | 10:44:56.992 | Thread Group 1-32 | HTTP Request | 3554 | | 257 | 1187 | 3554 | 1 |
| 11 | 10:44:57.208 | Thread Group 1-41 | HTTP Request | 3389 | | 257 | 1187 | 3389 | 1 |
| 12 | 10:44:57.020 | Thread Group 1-14 | HTTP Request | 3579 | | 257 | 1187 | 3579 | 2 |
| 13 | 10:44:57.008 | Thread Group 1-21 | HTTP Request | 3600 | | 257 | 1187 | 3600 | 2 |
| 14 | 10:44:57.008 | Thread Group 1-20 | HTTP Request | 3606 | | 257 | 1187 | 3606 | 2 |
| 15 | 10:44:57.099 | Thread Group 1-69 | HTTP Request | 3529 | | 257 | 1187 | 3529 | 1 |
| 16 | 10:44:57.016 | Thread Group 1-12 | HTTP Request | 3622 | | 257 | 1187 | 3622 | 2 |
| 17 | 10:44:57.018 | Thread Group 1-13 | HTTP Request | 3632 | | 257 | 1187 | 3632 | 2 |
| 18 | 10:44:57.339 | Thread Group 1-467 | HTTP Request | 3322 | | 257 | 1187 | 3322 | 3 |
| 19 | 10:44:57.011 | Thread Group 1-18 | HTTP Request | 3677 | | 257 | 1187 | 3677 | 1 |
| 20 | 10:44:57.324 | Thread Group 1-42 | HTTP Request | 3386 | | 257 | 1187 | 3386 | 2 |
| 21 | 10:44:57.347 | Thread Group 1-480 | HTTP Request | 3401 | | 257 | 1187 | 3401 | 1 |
| 22 | 10:44:56.904 | Thread Group 1-5 | HTTP Request | 3852 | | 257 | 1187 | 3852 | 1 |
| 23 | 10:44:57.360 | Thread Group 1-487 | HTTP Request | 3400 | | 257 | 1187 | 3400 | 3 |
| 24 | 10:44:56.899 | Thread Group 1-3 | HTTP Request | 3872 | | 257 | 1187 | 3872 | 1 |
| 25 | 10:44:56.895 | Thread Group 1-4 | HTTP Request | 3910 | | 257 | 1187 | 3910 | 6 |



**Figura 11-2 - Um teste de carga do *web service***

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| HTTP Request | 1000 | 9015 | 3322 | 15399 | 3978,56 | 0,00% | 53,3/sec | 13,37 | 61,74 | 257,0 |
| TOTAL | 1000 | 9015 | 3322 | 15399 | 3978,56 | 0,00% | 53,3/sec | 13,37 | 61,74 | 257,0 |

**Figura 11-3 - Estatísticas do teste de carga do *web service***

Os testes apresentados foram feitos instalando o SQL Server e o IIS numa máquina real, instalando o *website*, o *web service* e as bases de dados nessa máquina e testando o *web service* com o Apache JMeter a partir da mesma máquina. A máquina utilizada para efetuar este teste foi o computador portátil Toshiba Satellite L40-18L, este possui um processador Intel Pentium T2330 a 1.6GHz com dois núcleos, RAM DDR2-533 de 3GB, SSD Intel 520 Series 120GB e sistema operativo Windows 10 de 64 bits. Por se tratar de uma máquina antiga, de baixa performance, que consegue processar mil pedidos pesados simultaneamente (uma situação improvável em ambiente de produção), com um total de dez mil leituras em menos de dezasseis segundos (um bom tempo), pode-se deduzir que máquinas mais recentes com melhores especificações de *hardware* conseguem executar este teste em menos tempo.

## 11.3. Website

Inicialmente o *website* foi testado com testes unitários aos controllers, pois estes testam a implementação das várias operações suportadas pelo *website* diretamente nos controllers, no entanto os testes dos *controllers* podem ser insuficientes quando os dados são trocados de forma errada entre as *actions* dos *controllers* e as *views*. Para exemplificar este problema um valor do tipo inteiro é inserido numa página *web* pela *view* como um número decimal, pois foi utilizado um método para converter o valor numérico em *string* de forma errada:

```
@{ int valor = 123; }
<input type="hidden" name="valor" value="@valor.ToString("N")" />
```

Quando a *view* é executada, o HTML gerado e enviado para o cliente é o seguinte:

```
<input type="hidden" name="valor" value="123.00">
```

Este é um exemplo de erro que acontece frequentemente porque o parâmetro "N" do método "ToString()" indica que o valor deve ser representado na forma numérica, mas a representação depende do tipo de dados original. Caso o valor seja do tipo inteiro são adicionadas duas casas decimais. Quando o *web browser* envia o valor, este pode ser ou não aceite pela *action*, dependendo do tipo de dados do *view model*. Se a propriedade "valor" do *view model* for do tipo *string* ou um dos tipos de números decimais, o valor é extraído do pedido HTTP pelo mecanismo de "*model binding*" fornecido pela *framework* e a *action* é executada. No entanto se a propriedade "valor" do *view model* for de um dos tipos de números inteiros, é lançado um erro de desserialização pois a conversão de um número decimal para um número inteiro pode causar perda de informação ao truncar a parte decimal.

Este problema passa facilmente despercebido pelos testes unitários dos controllers, sendo apenas detetado quando um utilizador tentar submeter o formulário num *web browser*. Adicionalmente como este valor é enviado num campo oculto um utilizador normal não vai conseguir contornar este problema, pois apenas recebe um erro que indica que um valor desconhecido é inválido.

Para resolver este problema foi utilizada a extensão Selenium IDE no *web browser* Firefox para gravar a utilização de páginas do *website* de administração, generalizar essas gravações e utilizar posteriormente a gravações para automatizar a utilização de páginas e testar funcionalidades. A Figura 11-4 mostra um teste feito às páginas de gestão dos contadores, no *website* de administração.

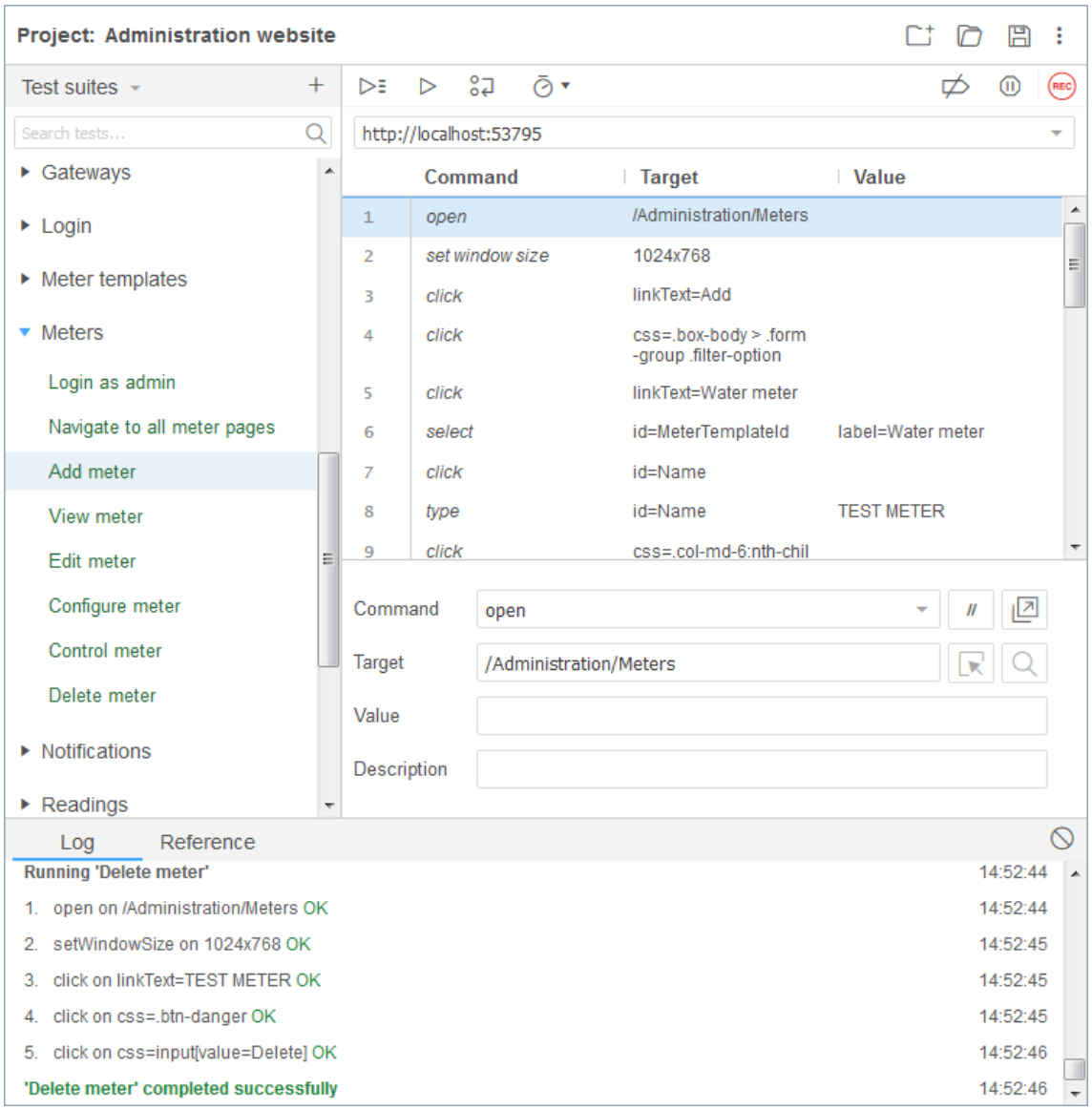



**Figura 11-4 - Teste da gestão de contadores a partir do *website***

Não foram efetuados testes de carga ao *website* pois não é esperado que seja acedido por muitos utilizadores em simultâneo, adicionalmente as otimizações feitas à forma como os dados são transferidos do servidor *web* para o *web browser* (ver seção 8.2) tornaram o carregamento de páginas quase instantâneo.

# 12. Conclusão e trabalho futuro

Neste capítulo será avaliado o estado atual do sistema, verificando se a solução entregue ao cliente satisfaz os seus requisitos. Serão apresentadas as limitações do sistema desenvolvido, ou seja os requisitos do cliente que não foram satisfeitos, e serão também apresentadas sugestões de trabalho futuro para melhorar o sistema.

Os requisitos mais importantes para o cliente foram satisfeitos e quase todos os restantes requisitos também foram satisfeitos, apenas um requisito não foi satisfeito e outro foi satisfeito parcialmente (ver seção 12.2). O cliente pode agora obter leituras de consumo de água, receber alertas de problemas e tentativas de fraude, controlar os contadores de água, configurar contadores e *gateways* e diagnosticar problemas com os contadores de água remotamente, a partir de um único *website*, como pretendia.

## 12.1. Pontos Fortes

**Flexibilidade**

As funcionalidades do sistema desenvolvido tornam-no indicado não só para o problema original, mas também para outros tipos de problemas semelhantes, o sistema foi desenvolvido com a intenção de ser o mais flexível possível, de modo a suportar outros tipos de contadores e casos de uso.

**Comunicação bidirecional universal**

Uma característica importante do sistema que não se encontra noutros sistemas é a comunicação bidirecional com todos os contadores a partir de uma rede fixa sem fios. Deste modo é possível controlar e configurar os contadores remotamente a partir de qualquer local e dispositivo com *web browser* e acesso à Internet. Adicionalmente, devido ao funcionamento da classe A de dispositivos LoRa, o tempo de vida útil da bateria não é reduzido significativamente.

**Suporte a *software* antigo**

O *website* de administração funciona em todos os *web browsers* atualmente mais utilizados, em versões antigas desses *web browsers* e até em *web browsers* descontinuados, tornando-se indicado para clientes empresariais em que as atualizações de *hardware* e *software* são normalmente raras devido aos custos associados.

**Independência da Internet**

O sistema pode ser instalado em máquinas do cliente e funcionar numa rede privada sem acesso à Internet, pois não depende de quaisquer plataformas de "*cloud computing*" porque todas as funcionalidades são implementadas localmente pelos diversos componentes de *software*. Esta característica torna o sistema ideal para casos em que o cliente já possui máquinas e rede própria, reduzindo potencialmente os custos de operação do sistema.

## 12.2. Limitações

Alguns requisitos do cliente não foram implementados ou foram implementados parcialmente. De seguida são apresentadas e justificadas as limitações do sistema desenvolvido.

**Repetidores de sinal**

O cliente pretendia que fossem desenvolvidos repetidores de sinal, mas como a especificação LoRaWAN atual não suporta repetidores de sinal estes não foram desenvolvidos. Foi recomendado instalar *gateways* LoRa mais próximos dos contadores, se possível, e caso os *gateways* LoRa não consigam obter uma boa ligação á rede móvel foi recomendado utilizá-los em conjunto com *gateways* 4G separados, também disponibilizados pela TULAlabs. Os *gateways* 4G são instalados em locais com bom sinal de rede móvel e os *gateways* LoRa ligam-se aos *gateways* 4G por cabo Ethernet ou Wi-Fi de modo a ligarem-se à Internet.

**Agendamento de comandos**

A implementação do agendamento de ações do contador está incompleta, devido à dificuldade em perceber o que o cliente pretendia realizar com esta funcionalidade. O cliente não especificou claramente como as ações deveriam ser agendadas, especificando apenas que deveria ser possível programar comandos para executar numa data e hora exata, durante um período de tempo e quando o consumo de água atingisse um determinado valor. No entanto nem todas as combinações de comandos com os métodos de agendamento solicitados fazem sentido (p. ex.: não faz sentido alterar a configuração de alertas quando o consumo de água atingir um determinado valor). Foram implementados apenas os casos de uso que o cliente utilizou como exemplos, ou seja, permitir que o cliente contrate um determinado volume de água e fechar a válvula quando o consumo de água atingir esse volume e permitir programar a válvula para abrir e fechar em determinada data (p. ex.: abrir a válvula dia 15-08-2017 às 23:45). Como este requisito não foi bem entendido, também foi implementado agendamento semanal do controlo da válvula (p. ex.: abrir a válvula às sextas-feiras às 18:30 e fechar a válvula aos domingos às 23:00). Este tipo de agendamento foi mantido visto que o trabalho já foi realizado e que poderá ser útil em outras situações.

## 12.3. Trabalho Futuro

Embora o sistema implemente quase todos os requisitos do cliente há sempre melhorias que podem ser feitas mais tarde, de modo a facilitar sua utilização e configuração. De seguida são apresentadas sugestões para melhorar do sistema.

**Comunicação entre o IIS e os *gateways* por WebSockets**

No sistema atual existem dois *web services*, um implementado no IIS e o outro no Node-RED, deste modo é possível enviar comandos para os contadores ou alterar configurações dos *gateways* de forma imediata, sem recurso a *polling*. No entanto torna-se necessário efetuar a configuração e manutenção de dois *web services* distintos e se for necessário efetuar alterações na API pode ser necessário alterar código fonte de cliente e servidor no IIS e no Node-RED, pois ambos atuam como cliente e servidor HTTP.

Poderia ser desenvolvida uma API com WebSockets como alternativa aos *web services*. Como a *framework* .NET tem suporte para WebSockets, o código dos *controllers* ASP.NET MVC poderia ser movido para uma camada comum de modo a reutilizar o código atual. Os WebSockets e o *web service* no IIS funcionariam apenas como meios de entrada e saída de dados. Na página de configuração dos *gateways*, no *website* de administração, seria dada a opção de comunicar com os *gateways* através de WebSockets ou através dos *web services*.

**Comunicação entre o IIS e o LoRa App Server por gRPC**

É possível desenvolver *software* em C# que comunica diretamente com o LoRa App Server através do sistema de chamada de procedimento remoto gRPC e o cliente gRPC desenvolvido em C# poderia ser incorporado com o *website* e com o *web service* no IIS. Deste modo seria possível dispensar o sistema desenvolvido no Node-RED e o Eclipse Mosquitto, pois o encaminhamento de mensagens dos contadores entre o IIS e o LoRa App Server seria feito de forma direta. No entanto o método de comunicação atual, através de dois *web services*, seria mantido de modo a permitir comunicar com outros *gateways* que não utilizem LoRaWAN para comunicar com contadores. Na página de configuração dos *gateways*, no *website* de administração, seria dada a opção de comunicar com os *gateways* através da API gRPC do LoRa App Server ou através dos *web services* (ou também WebSockets, caso sejam implementados), sendo solicitados parâmetros de configuração diferentes dependendo da opção escolhida.

**Integrar a gestão dos contadores e *gateways* com o LoRa App Server**

Atualmente é necessário configurar os contadores e *gateways* no *website* de administração e no LoRa App Server, no entanto o *website* de administração poderia comunicar com o LoRa App Server através da API gRPC, de modo a ser possível configurar os contadores e *gateways* em apenas um lugar. Integrando o website com o LoRa App Server, ao adicionar contadores ou *gateways* no *website* de administração estes seriam também adicionados automaticamente no LoRa App Server. Ao alterar configurações dos contadores ou *gateways* no *website* de administração essas configurações seriam também alteradas no LoRa App Server e ao apagar contadores ou *gateways* no *website* de administração estes seriam também apagados no LoRa App Server. Nas páginas do *website* de administração que permitem adicionar, editar, configurar e remover contadores e *gateways*, seria dada a opção de aplicar estas operações automaticamente no LoRa App Server através da API gRPC, se os contadores e *gateways* em questão utilizarem o LoRa App Server como intermediário.

***Website* e *web service* servidos a partir de outros servidores HTTP ou sistemas operativos**

De modo a suportar outros sistemas operativos seria possível adaptar o código de ASP.NET para ASP.NET Core, deste modo o *website* e o *web service* no IIS (e também a API WebSocket, caso seja implementada) poderiam ser servidos a partir de uma máquina com um sistema operativo que não fosse Windows, eliminando o requisito de utilizar o Windows para servir o *website* e o *web service*. A principal motivação para adaptar o código de ASP.NET para ASP.NET Core é o facto de que em várias plataformas de *cloud computing* (incluindo a plataforma Amazon EC2), máquinas virtuais com o sistema operativo Windows são normalmente mais caras do que máquinas virtuais com sistema operativo Linux, mesmo quando as especificações de *hardware* são iguais (possivelmente por causa da licença do Windows). Adicionalmente

o sistema operativo Linux necessita normalmente de menos recursos de *hardware* do que o sistema operativo Windows, reduzindo os custos ainda mais (ver Tabela 6-3).

ASP.NET Core é uma extensão da *framework* .NET Core, ambas são gratuitas, têm o código fonte aberto e são desenvolvidas pela Microsoft. Aplicações desenvolvidas com a *framework* .NET Core podem ser executadas em outros sistemas operativos, por exemplo Linux. Para servir o *website* desenvolvido com ASP.NET Core poderia ser utilizado o servidor *web* Kestrel, este também é gratuito, de código fonte aberto e desenvolvido pela Microsoft.

**Desenvolver o sinóptico**

O sinóptico foi uma tentativa de criar um mapa que representaria as áreas físicas em que os contadores estão instalados e que o utilizador poderia navegar, selecionar contadores e ver leituras, executar comandos ou alterar configurações diretamente a partir do sinóptico. Resumidamente o sinóptico forneceria uma interface de gestão semelhante a SCADA, mas mais simplificada. Inicialmente o sinóptico funcionava apenas com contadores de água, mais tarde foi generalizado para suportar outros tipos de contadores, mas como este trabalho ficou a meio atualmente o sinóptico não está funcional, permitindo apenas navegar nas áreas físicas e ver as quais os contadores que se encontram nessas áreas.

**Triggers**

Uma funcionalidade que foi desenvolvida parcialmente foi os *triggers*, estes permitem definir alertas que são emitidos quando certas condições são satisfeitas. Atualmente as condições suportadas são a comparação de valores de leituras. Como exemplo, é possível criar um trigger "Alto consumo" para um contador de água que emite um alerta quando o consumo do contador ultrapassar um certo valor, ou um trigger "Risco de incêndio elevado" para uma estação meteorológica quando a temperatura é superior a 30ºC e a humidade relativa inferior a 20%.

A intenção original dos *triggers* era implementar os alertas "Baixo consumo" e "Alto consumo" dos contadores de água no lado do servidor, permitindo que o utilizador configurasse quais os valores considerados "altos" e "baixos" de cada contador. O desenvolvimento parou quando o cliente esclareceu que não necessitava de configurar os valores de alto consumo por contador, a partir do *website* de administração, pois todos os contadores iam ser configurados com um valor fixo, e que o valor de baixo consumo seria zero.

Na implementação atual um trigger é uma série de condições ligadas pelo operador "e", ou seja o *trigger* é ativado se a condição "A" se verificar e a condição "B" se verificar e a condição "C" se verificar, e assim por diante. As condições de um trigger são compostas por comparações de leituras de qualquer contador pelos seus valores absolutos (valores recebidos no *web service*) ou valores relativos (diferença entre os valores recebidos no *web service* e os valores da última leitura). Uma ideia de melhoria seria permitir outros tipos de operadores lógicos, por exemplo "ou", "ou exclusivo" e "negação" e permitir encadear essas condições com parênteses, por exemplo "(A e B) ou C".

### Previsões

Uma ideia que não chegou a ser implementada foi a previsão de leituras e alertas com base em dados históricos. Foi pensado utilizar métodos estatísticos, tais como regressão linear simples, pois o volume de água consumido está sempre a aumentar e portanto uma linha seria uma aproximação, sem grande rigor, da tendência de consumo no futuro. Adicionalmente foi pensado utilizar regressão polinomial, pois capturaria melhor a variação das leituras tanto do consumo de água como de outros tipos. Também foi pensado utilizar *machine learning* para gerar modelos analíticos e a partir desses modelos obter melhores previsões de leituras futuras.

### Comparações de leituras

Mais uma ideia que não chegou a ser implementada foi a comparação de leituras. Esta permitiria comparar leituras de um contador em períodos homólogos e comparar leituras de vários contadores que se baseiam no mesmo modelo de contador permitindo, por exemplo, calcular diferenças de valores num determinado intervalo de tempo. Também seria possível comparar leituras de contadores de modelos diferentes num determinado intervalo de tempo, no entanto o utilizador teria que especificar regas de conversão para, por exemplo, comparar metros cúbicos com litros.

# Bibliografia


[1] "Diehl Group." [Online]. Disponível em: https://www.diehl.com/.
[2] "Diehl Metering - Solutions." [Online]. Disponível em: https://www.diehl.com/metering/en/portfolio/solutions/.
[3] "Kamstrup - Smart metering solutions." [Online]. Disponível em: https://www.kamstrup.com/.
[4] "Kamstrup water solutions - Smart water metering." [Online]. Disponível em: https://www.kamstrup.com/en-us/water-solutions.
[5] "Itron." [Online]. Disponível em: https://www.itron.com/.
[6] "Itron - Water." [Online]. Disponível em: https://www.itron.com/na/solutions/who-we-serve/water.
[7] "MultiConnect Conduit - Programmable Gateway for the Internet of Things (MTCDT Series)." [Online]. Disponível em: https://www.multitech.com/brands/multiconnect-conduit.
[8] "IoT Gateways, Routers and Modems - MultiTech." [Online]. Disponível em: https://www.multitech.com/products/gateways-routers-modems.
[9] "LoRa Server, open-source LoRaWAN network-server." [Online]. Disponível em: https://www.loraserver.io/.
[10] "Raspberry Pi 2 Model B – Raspberry Pi." [Online]. Disponível em: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.
[11] "Node-RED." [Online]. Disponível em: https://nodered.org/.
[12] "About Node.js." [Online]. Disponível em: https://nodejs.org/en/about/.
[13] "RFC 7617 - The 'Basic' HTTP Authentication Scheme." [Online]. Disponível em: https://tools.ietf.org/html/rfc7617.
[14] "Entity–attribute–value model - Wikipedia." [Online]. Disponível em: https://en.wikipedia.org/wiki/Entity-attribute-value_model.
[15] "decimal and numeric (Transact-SQL) - SQL Server." [Online]. Disponível em: https://docs.microsoft.com/en-us/sql/t-sql/data-types/decimal-and-numeric-transact-sql.
[16] "LoRaWAN, what is it? - A technical overview of LoRa and LoRaWAN." [Online]. Disponível em: https://lora-alliance.org/resource-hub/what-lorawanr.
[17] "Introduction to IIS Architectures." [Online]. Disponível em: https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/introduction-to-iis-architecture.
[18] "ETSI EN 300 220-1 V2.4.1." [Online]. Disponível em: https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/02.04.01_40/en_30022001v020401o.pdf.
[19] "LoRaWAN Specification 1.0.1." [Online]. Disponível em: https://lora-alliance.org/resource-hub/lorawanr-specification-v101.
[20] "MQTT Version 3.1.1." [Online]. Disponível em: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf.
[21] "Maximum Capacity Specifications for SQL Server - SQL Server." [Online]. Disponível em: https://docs.microsoft.com/en-us/sql/sql-server/maximum-capacity-specifications-for-sql-server.
[22] "Data Centers - Our Controls." [Online]. Disponível em: https://aws.amazon.com/compliance/data-center/controls/.

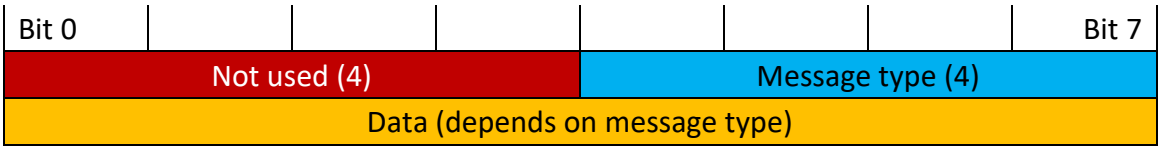
# Anexo A: Protocolo de rádio

De seguida é apresentado um resumo do protocolo de rádio, as mensagens enviadas e recebidas dos contadores são apresentadas, os campos que compõem as mensagens são descritos de forma sucinta a é apresentado um exemplo de cada mensagem.

**Messages to meters**

All messages sent to meters have a header of 8 bits, of these 8 bits currently only the 4 least significant bits are used to identify the type of message. The type of message defines the message structure, the data fields and how many bytes the meter should expect.

There is no error detection and correction mechanism in these messages, data integrity is provided by the underlying LoRaWAN protocol.




| Message type | Description | Type |
|---|---|---|
| 0 | Request result | Response |
| 1 | Timestamp | Response |
| 2 | Meter reading | Request |
| 3 | Battery status | Request |
| 4 | Valve control | Request |
| 5 | Signal mapping point | Request |
| 6 | Interval configuration | Request |
| 7 | Alert configuration | Request |
| 8 | Set reading | Request |
| 9 | Add schedule | Request |
| 10 | Clear schedule | Request |
| 11 | Set consumption limit | Request |

A request message has a matching response message, a response message has a matching request message.

**0 – Request result**

This message reports request execution result.

Example: [0x00, 0x01, 0x01] (response to timestamp request message, no reliable date-time source found)

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |
| 1 | Not used (4) | | | | Confirmed message type (4) | | | |
| Byte 2 | Result code (8) | | | | | | | |

*Timestamp*

| Result code | Description |
|---|---|
| 0 | NOT USED – on success a timestamp response message is sent |
| 1 | No reliable date-time source found |

*Alert*

Alert requests always succeed, the system should store sent alerts in the system log if the alert code is not present in the database.

**1 - Timestamp (response)**

This message sends current system time in Unix timestamp format (number of seconds since 1970-01-01 00:00:00). This timestamp can be obtained from any system with a working NTP client.

Example: [0x01, 0x00, 0x58, 0xDF, 0x9D, 0x48] (2017-04-01 12:30).

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |
| 1 | Unix timestamp (40) | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| Byte 5 | | | | | | | | |

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Unix timestamp | Unsigned 40-bit integer | 1,483,228,800 to 1,099,511,627,775 (2017-01-01 00:00:00 to 36812-02-20 00:36:16) | 0 to 1,099,511,627,775 (1970-01-01 00:00:00 to 36812-02-20 00:36:16) |

Using an unsigned 32-bit integer for the timestamp is not enough as the time would reset after 2106-02-07 06:28:15, an additional byte was added to support dates after 2106, this new byte allows dates up to 36812-02-20 00:36:16 AD (not a typo).

**2 - Meter reading (request)**

This message requests the current meter reading. The message type “0” is sufficient to make the request.

Example: [0x02]

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |

**3 - Battery status (request)**

This message requests the meter battery status. The message type “1” is sufficient to make the request.

Example: [0x03]

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |

**4 - Valve control (request)**

This message controls the meter valve, allowing it to open or close it.

Example: [0x04, 0x01] (open the valve).

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |
| Byte 1 | Not used (7) | | | | | | | Valve control (1) |

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Valve control | Bit | 0 to 1 | 0 to 1 |

| Valve control | Description |
|---|---|
| 0 | Close the valve |
| 1 | Open the valve |

**5 - Signal mapping point (request)**

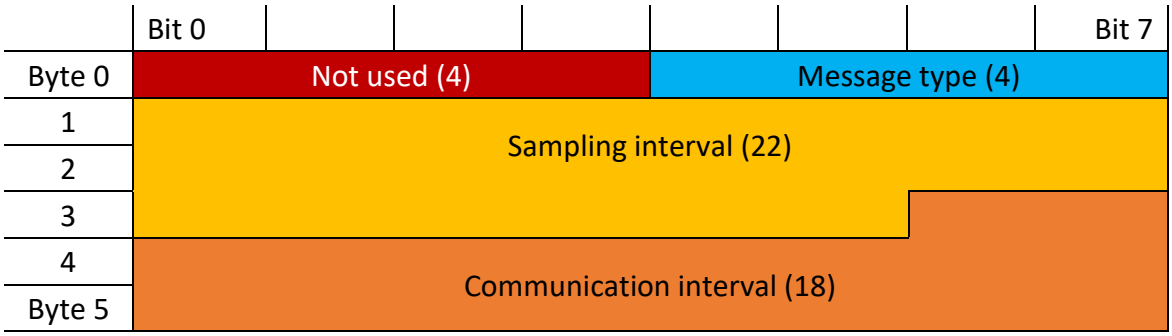
This message requests the current meter coordinates, altitude, information about signal reception conditions, and transmission parameters. The message type "5" is sufficient to make the request.

Example: [0x05]


**6 - Interval configuration (request)**

This message tries to change the meter sampling interval (number of seconds between meter reading acquisitions) and communication interval (number of minutes between meter readings communication).

Example: [0x06, 0x00, 0x38, 0x40, 0x05, 0xA0] (sampling interval of 1 hour, communication interval of 1 day).




| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Sampling interval | Unsigned 22-bit integer | 0 to 2,592,000 | 0 to 4,194,304 |
| Communication interval | Unsigned 18-bit integer | 0 to 86,400 | 0 to 262,144 |

Sampling interval is defined by an unsigned integer of 22 bits, the time unit is seconds. The shortest interval that can be configured is 1 second and the longest is about 48 days. Currently meters only allow a maximum sampling interval of 30 days (2592000 seconds).

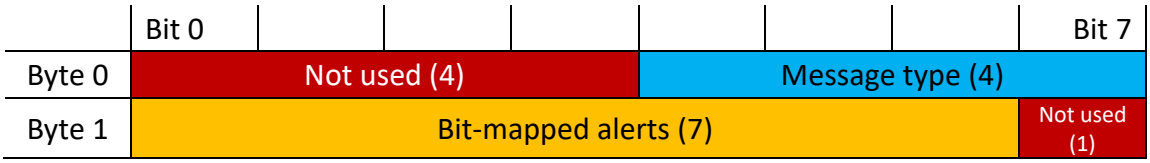
The communication interval is defined by an unsigned integer of 18 bits, the time unit is minutes. The shortest interval that can be configured is 1 minute and the longest is about 182 days. Currently meters only allow a maximum sampling interval of 60 days (86400 minutes) so only 17 bits are used (the most significant bit is ignored).

Setting either value to zero will not change the previous configuration; this allows changing only one parameter at a time.

### 7 - Alert configuration (request)

Each alert is mapped to a bit, if the bit is set to “0”, the alert is disabled, and if the bit is set to “1” the alert is enabled. This simplifies alert configuration since all alerts can be configured by just one small message.

Example: [0x07, 0xC8] (inverse flux, magnetic fraud, and low battery enabled, everything else disabled).




| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Bit-mapped alerts | Unsigned 7-bit integer | 0 to 127 | 0 to 127 |

| Bit | Description |
|---|---|
| 0 (MSB) | Reverse flux |
| 1 | Magnetic fraud |
| 2 | High consumption |
| 3 | Zero consumption |
| 4 | Low battery |
| 5 | Memory error |
| 6 | Wrong date/time |

### 8 - Set reading (request)

This message will clear all readings stored on the meter memory and set the current reading value, this is useful when installing a “smart” water meter on an existing customer who has a mechanical meter or when replacing a malfunctioning smart meter, as the new smart meter should start counting from the old reading, not from zero.

Example: [0x08, 0x00, 0x00, 0x01, 0xF4] (set reading to 500 liters).

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |
| 1 | Not used (2) | | | | Reading (30) | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| Byte 4 | | | | | | | | |

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Reading | Unsigned 30-bit integer | 0 to 999,999 | 0 to 1,073,741,823 |

**9 - Add schedule (request)**

This message will add a scheduled action to the meter schedule.

This function is still being discussed, as it is not clear what actions need to be scheduled besides the valve control and what scheduling options are required.

Example 1: [0x09, 0x02, 0x00, 0x58, 0xDF, 0x9D, 0x48] (open the valve at 2017-04-01 12:30).

Example 2: [0x09, 0x22, 0x08, 0x12, 0x1E, 0x00, 0x00] (open the valve at 18:30:00 every Friday)

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (4) | | | | Message type (4) | | | |
| 1 | Schedule type (3) | | | Action number (5) | | | | |
| 2 | Depend on type of schedule (40) | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

Up to 32 actions can be scheduled, but currently only two are defined, setting the action number to “0” (zero) disables the scheduled action.

Two types of schedule are supported; exact date-time and weekly repeating. If the “exact date-time” schedule type is chosen the action will be performed in the specified date-time. If the “weekly repeating” schedule type is chosen the action will repeat weekly on the specified days of the week, at the specified hour, minute and second.

| Schedule type | Description |
|---|---|
| 0 | Exact date |
| 1 | Weekly |

| Action | Description |
|---|---|
| 0 | None |
| 1 | Close the valve |
| 2 | Open the valve |

Exact date-time (schedule type)

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Unix timestamp (40) | | | | | | | |
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| Byte 4 | | | | | | | | |

Weekly repeating (schedule type)

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Bit-mapped days of week (7) | | | | | | | Not used (1) |
| 1 | Not used (2) | | | Hour (5) | | | | |
| 2 | Not used (2) | | | Minute (6) | | | | |
| 3 | | | | Second (6) | | | | |
| Byte 4 | Not used (8) | | | | | | | |

Bit-mapped days of week

| Bit | Description |
|---|---|
| 0 (MSB) | Monday |
| 1 | Tuesday |
| 2 | Wednesday |
| 3 | Thursday |
| 4 | Friday |
| 5 | Saturday |
| 6 | Sunday |

All parameters

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Type of schedule | Unsigned 3-bit integer | 0 to 1 | 0 to 7 |
| Action | Unsigned 5-bit integer | 0 to 1 | 0 to 31 |
| Bit-mapped days of week | Unsigned 7-bit integer | 0 to 127 | 0 to 127 |
| Hour | Unsigned 5-bit integer | 0 to 23 | 0 to 31 |
| Minute | Unsigned 6-bit integer | 0 to 59 | 0 to 63 |
| Second | Unsigned 6-bit integer | 0 to 59 | 0 to 63 |
| Unix timestamp | Unsigned 40-bit integer | 1,483,228,800 to 1,099,511,627,775 (2017-01-01 00:00:00 to 36812-02-20 00:36:16) | 0 to 1,099,511,627,775 (1970-01-01 00:00:00 to 36812-02-20 00:36:16) |

**10 - Clear schedule (request)**

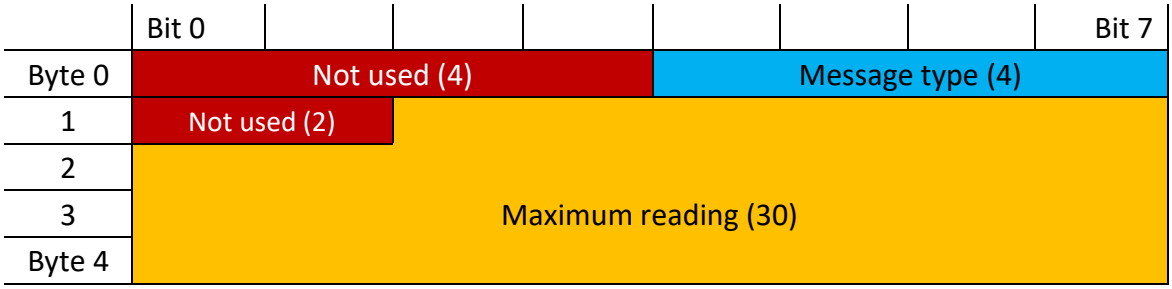
This message clears the meter schedule. The message type “10” is sufficient to make the request.

Example: [0x0A]


**11 - Set consumption limit (request)**

This message will set the consumption limit on the meter memory, this is allows a client to purchase a set amount of water and when the limit is reached the valve is automatically closed. If the client purchases more water this limit can be increased.

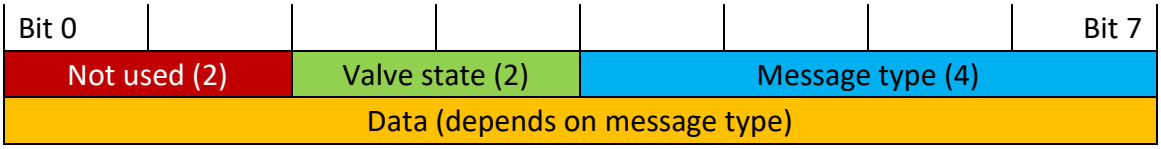
Example: [0x0B, 0x80, 0x00, 0x3A, 0x98] (set consumption limit to 15000 liters).



**Messages from meters**

All messages received from meters have a header of 8 bits, of these 8 bits the 4 least significant bits are used to identify the type of message, the middle 2 report the valve state and the 2 most significant bits are currently not used. The type of message defines the message structure, the data fields and how many bytes the recipient system should expect.

There is no error detection and correction mechanism in these messages, data integrity is provided by the underlying LoRaWAN protocol.



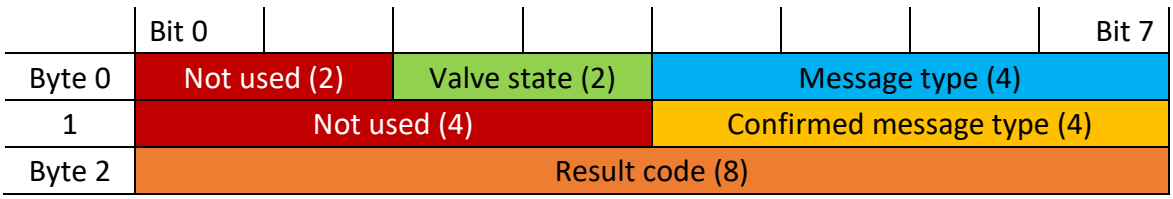

| Valve state | Description |
|---|---|
| 0 | Closed |
| 1 | Half-open |
| 2 | Could not read state |
| 3 | Open |

| Message type | Description | Type |
|---|---|---|
| 0 | Request result | Response |
| 1 | Timestamp | Request |
| 2 | Meter reading | Response |
| 3 | Battery status | Response |
| 4 | Alert | Request |
| 5 | Signal mapping point | Response |

**0 – Request result**

This message reports command execution and configuration validation result.

Example: [0x00, 0x06, 0x04] (response to interval configuration message, invalid communication interval, the valve is closed)



*Meter reading*

| Result code | Description |
|---|---|
| 0 | NOT USED - meter reading requests always succeed, if the meter has never been used or if it was reset it should send zero. |
| 1 | The meter reading request message has an incorrect size (currently it should be 8 bits) |

Battery status

| Result code | Description |
|---|---|
| 0 | NOT USED – on success a battery status response message is sent |
| 1 | The battery status message has an incorrect size (currently it should be 64 bits) |
| 2 | Could not read battery voltage (battery voltage is the only required parameter for this type of message) |

*Valve control*

| Result code | Description |
|---|---|
| 0 | Valve control command executed successfully |
| 1 | The valve control message has an incorrect size (currently it should be 16 bits) |
| 2 | Could not open or close the valve completely, it is in an intermediate state (half-open) |
| 3 | Could not read valve state after executing command |

*Signal mapping point*

| Result code | Description |
|---|---|
| 0 | NOT USED – on success a signal mapping point response message is sent |
| 1 | The signal mapping point message has an incorrect size (currently it should be 160 bits) |
| 2 | Meter has no GPS receiver |
| 3 | No GPS signal |
| 4 | Not enough GPS satellites |
| 5 | No position fix |

Interval configuration

| Result code | Description |
|---|---|
| 0 | Sampling and communication intervals were changed successfully |
| 1 | The interval configuration message has an incorrect size (currently it should be 48 bits) |
| 2 | Invalid sampling and communication intervals, intervals were not changed |
| 3 | Invalid sampling interval, intervals were not changed |
| 4 | Invalid communication interval, intervals were not changed |

*Alert configuration*

| Result code | Description |
|---|---|
| 0 | Alert configuration was changed successfully |
| 1 | The alert configuration message has an incorrect size (currently it should be 16 bits) |

*Set reading*

| Result code | Description |
|---|---|
| 0 | Meter reading was changed successfully |
| 1 | The reading changing message has an incorrect size (currently it should be 40 bits) |
| 2 | Reading value is larger than maximum supported reading |

*Add schedule*

| Result code | Description |
|---|---|
| 0 | Schedule was added successfully |
| 1 | The schedule adding message has an incorrect size (currently it should be 56 bits) |
| 2 | Invalid schedule type |
| 3 | Invalid action |
| 4 | Invalid hour |
| 5 | Invalid minute |
| 6 | Invalid second |

*Clear schedule*

| Result code | Description |
|---|---|
| 0 | Schedule was cleared successfully |
| 1 | The schedule adding message has an incorrect size (currently it should be 8 bits) |

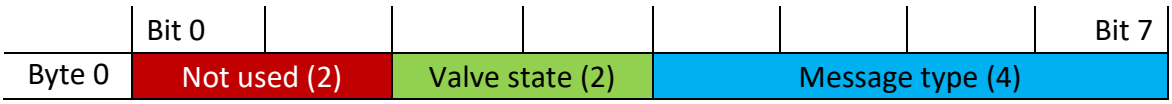
*Set consumption limit*

| Result code | Description |
|---|---|
| 0 | Consumption limit was changed successfully |
| 1 | The consumption limit changing message has an incorrect size (currently it should be 40 bits) |
| 2 | Consumption limit value is larger than maximum supported reading value |

**1 - Timestamp (request)**

This message requests the current date and time on the server. This date and time will be used to set the meter internal clock when the meter starts up. The message type "1" is sufficient to make the request.

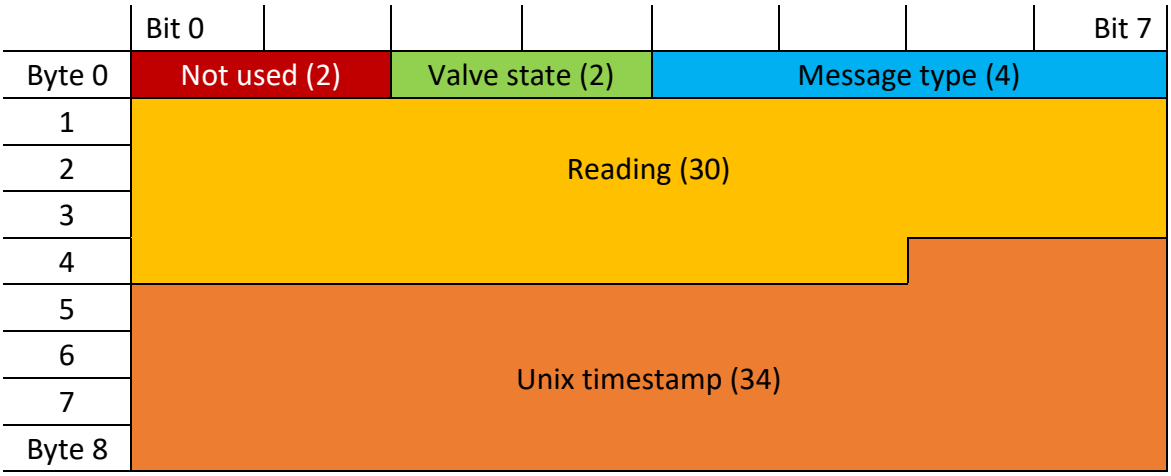
Example: [0x01]




**2 - Meter reading (response)**

This message reports a meter reading at a specific date. Readings are sent periodically by the meter or can be requested by the server. This message is optimized to the least number of bytes possible because in normal operation, it should represent most of the data sent by the meter.

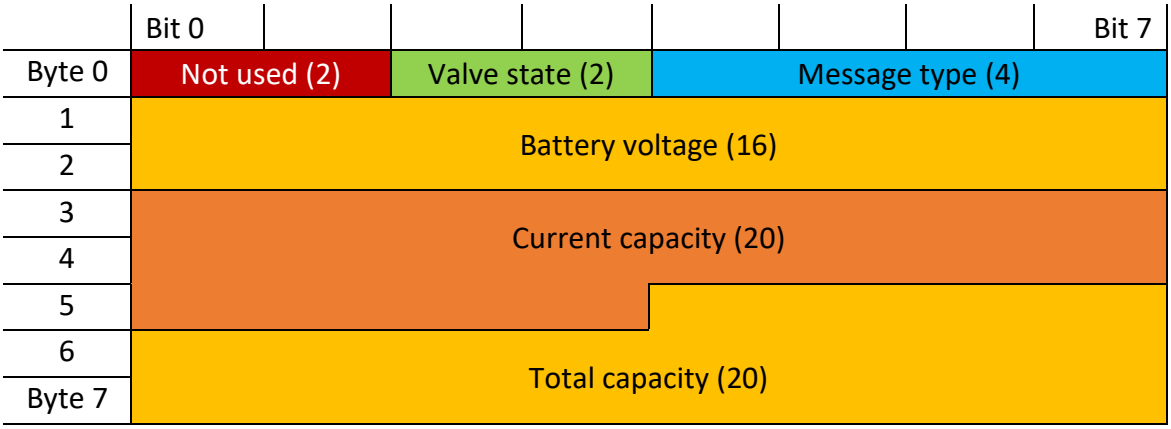
Example: [0x02, 0x00, 0x01, 0xF4, 0x00, 0x58, 0xDE, 0xED, 0x80] (reading of 500 liters at 2017-04-01 00:00:00).




**3 - Battery status (response)**

This message reports current battery voltage, current battery capacity and total battery capacity. This information is used to detect if the battery should be charged or replaced. A battery status message is sent automatically after sending a “Low battery” alert.

Example: [0x03, 0x0E, 0x10, 0x01, 0xBE, 0x40, 0x3F 0x98] (battery voltage is 3600 mV, current capacity is 7,140 mWh, total capacity of 16,280 mWh)




| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Battery voltage | Unsigned 16-bit integer | 3,000 to 4,350 | 0 to 65,535 |
| Current capacity | Unsigned 20-bit integer | 0 to total capacity | 0 to 1,048,575 |
| Total capacity | Unsigned 20-bit integer | 0 to battery capacity | 0 to 1,048,575 |

Battery voltage is reported in millivolts and can go up to 65,535 mV (roughly 65 Volts). Supports currently used type of lithium batteries (approximately 3 to 4.2 Volts) and other types of batteries.

As some meters may have a battery management system, current and total capacity can also be reported. Capacity values are reported in milliwatts-hour and can go up to 1,048,575 mWh (roughly 1,048 watts-hour), in order to support special situations where there is a need for a big capacity battery. Using these values it is possible to compute the battery state of charge.

$$soc = \frac{current\ \ capacity}{total\ capacity} * 100$$

If the meter does not have a BMS and can only report the battery voltage, it is possible to estimate the state of charge using a reference voltage discharge curve, obtained by analyzing data from another meter which has a BMS (the actual curve depends on the type of battery, on the load caused by the meter operation and on the battery wear level).

**4 - Alert (request)**

This message is sent as soon as possible, when an alert condition is detected by the meter.

Example: [0x30, 0x44] (low battery alert, the valve is open).

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (2) | | Valve state (2) | | Message type (4) | | | |
| Byte 1 | Level (2) | | Alert (6) | | | | | |

| Level | Description |
|---|---|
| 0 | Fatal |
| 1 | Error |
| 2 | Warning |
| 3 | Troubleshooting |

| Alert | Description | Level |
|---|---|---|
| 0 | Inverse flux | 2 |
| 1 | Magnetic fraud | 2 |
| 2 | High consumption | 2 |
| 3 | Zero consumption | 2 |
| 4 | Low battery | 2 |
| 5 | Memory error | 0 |
| 6 | Wrong date/time | 1 |

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Level | Unsigned 2-bit integer | 0 to 3 | 0 to 3 |
| Alert | Unsigned 6-bit integer | 0 to 6 | 0 to 63 |

The level defines if the system cannot continue working (fatal), if the system has a problem that can be fixed remotely (error), if some hazardous condition was detected (warning) or if the alarm was sent for troubleshooting purposes.

The alert is represented by a code; the server must know each alert code.

**5 - Signal mapping point (response)**

This message contains the signal reception conditions and transmission parameters for the corresponding request message at the current meter coordinates and altitude.

This message is optimized to use the least number of bits possible while sending multiple values, so as to not reach the duty cycle limit.

Example: [0x05, 0x00, 0x3D, 0x5D, 0x96, 0xFF, 0xFF, 0x87, 0xC8, 0x01, 0xFA, 0x03, 0x98, 0x33, 0x0D, 0xC6, 0xF4, 0x3E, 0xC5, 0xA3] (ESTGOH coordinates and altitude, SNR is 9.2, RSSI is -51 dBm, frequency is 902,900 kHz, bandwidth is 125kHz, code rate numerator is 4, code rate denominator is 5, spreading factor is 10, channel is 3)

| | Bit 0 | | | | | | | Bit 7 |
|---|---|---|---|---|---|---|---|---|
| Byte 0 | Not used (2) | | Valve state (2) | | Message type (4) | | | |
| 1 | Latitude (32) | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | Longitude (32) | | | | | | | |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
| 9 | Altitude (16) | | | | | | | |
| 10 | | | | | | | | |
| 11 | Signal-to-noise ratio (16) | | | | | | | |
| 12 | | | | | | | | |
| 13 | Received signal strength indicator (8) | | | | | | | |
| 14 | Frequency (24) | | | | | | | |
| 15 | | | | | | | | |
| 16 | | | | | | | | |
| 17 | Bandwidth (9) | | | | | | | |
| 18 | | Code rate numerator (3) | | | Code rate denominator (4) | | | |
| Byte 19 | Spreading Factor (4) | | | | Channel (4) | | | |

| Parameter name | Data type | Used range | Supported range |
|---|---|---|---|
| Latitude | Signed 32-bit integer | -900,000,000 to 900,000,000 | -2,147,483,648 to 2,147,483,647 |
| Longitude | Signed 32-bit integer | -1,800,000,000 to 1,800,000,000 | -2,147,483,648 to 2,147,483,647 |
| Altitude | Signed 16-bit integer | -345 to 8848 | -32,768 to 32,767 |
| Signal-to-noise ratio | Signed 16-bit integer | -20.00 to 20.00 (example) | -32,768 to 32,767 |
| Received signal strength indicator | Unsigned 8-bit integer | -150 to 0 (signal is inverted) | 0 to 255 |
| Frequency | Unsigned 24-bit integer | 433,000 to 927,500 | 0 to 16,777,215 |
| Bandwidth | Unsigned 9-bit integer | 125 to 500 | 0 to 511 |
| Code rate numerator | Unsigned 3-bit integer | 4 | 0 to 7 |
| Code rate denominator | Unsigned 4-bit integer | 5 to 8 | 0 to 15 |
| Spreading Factor | Unsigned 4-bit integer | 6 to 12 | 0 to 15 |
| Channel | Unsigned 4-bit integer | 1 to 10 | 0 to 15 |

The first 32 bits define the latitude, the following 32 bits define the longitude, both in the NMEA GPGGA format (DDDMM.MMMMM), to decode this format the 5 decimal numbers to the right of the decimal separator represent the decimal part of the minutes (DDDMM.MMMMM), the first 2 numbers to the left of the decimal separator represent the integer part of the minutes (DDDMM.MMMMM), the rightmost 3 numbers to the left of the decimal separator represent the integer decimal degrees (DDDMM.MMMMM), a negative latitude means south, and a negative longitude means west.

Altitude (meters above mean sea level) is defined in meters, as the measurements probably will be taken on land, this value should be between -432 meters (the lowest point on dry land, the shore of the Dead Sea, in Jordan/Palestine/Israel) and 8848 meters (the highest point on the Earth's surface, Mount Everest, in Nepal/China) however altitudes of up to about 32 km below and above the sea level are supported, as most commercial GPS receivers have a big altitude error margin (frequently about 15 meters).

Signal-to-noise ratio is represented in decibels, depends on transceivers/antennas but should be below 20 dB, some of the most sensitive LoRa transceivers can operate under ambient noise, as such the SNR can be negative. The SNR value is multiplied by 100 before sending, as this is a fixed-point number.

Received signal strength is defined in decibel-milliwatts, depends on transceivers/antennas but should be between -150 and 0 dBm, it is sent as an unsigned integer but should always be interpreted as a negative number or zero.

Frequency is defined in kilohertz and is expected to be between 43,000 and 927,500 kHz, as currently that’s the lowest and highest frequency in the LoRaWAN regional parameters specification (EU433 and AU915-928).

Bandwidth is defined in kilohertz and should be between 125 and 500 kHz, some transceivers support bandwidths as low as 7.8 kHz, but they will not be used in this project.

Code rate numerator, defines the numerator part of the code rate fraction or the number of useful bits sent, for example it could be 4.

Code rate denominator, defines the denominator part of the code rate fraction or the number of actual sent bits sent (useful bits + redundant bits), for example it could be 5.

Spreading factor is defined by the chip rate divided by the symbol rate, as the spreading factor increases the data rate decreases (an increment of 1 in the spreading factor doubles the airtime needed to transmit the same amount of data), depends on transceiver but is usually between 6 and 12.

Channel number, depends on used band, location and direction, for instance in the European Union channel 1 of the EU863-870 band is 868,100 kHz for uplink/downlink, but channel 1 for the US902-928 band is 903,900 kHz for uplink and 923,300 kHz for downlink.

# Anexo B: Especificação da base de dados principal

De seguida é apresentado um resumo da especificação da base de dados, as tabelas que compõem a base de dados principal são descritas de forma sucinta e as colunas que compõem essas tabelas são apresentadas de forma detalhada, apresentando o nome da coluna, o tipo de dados, se pode guardar valores nulos e uma breve descrição. As linhas a amarelo representam a chave primária, as linhas a laranja representam as chaves forasteiras.

Não é apresentado um diagrama de entidade-relacionamento pois o diagrama não pode ser mostrado em apenas uma página.

### dbo.__MigrationHistory

Stores data about previous and current database schema. Part of the Entity Framework.

| Column | Data type | Null | Description |
|---|---|---|---|
| MigrationId | nvarchar(150) | No | Surrogate key |
| ContextKey | nvarchar(300) | No | Database context key |
| Model | varbinary(-1) | No | Database model |
| ProductVersion | nvarchar(32) | No | Entity Framework version |

### dbo.Alerts

Stores data about system alerts and custom alerts that can be sent by meters.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(50) | No | Alert name |
| Description | nvarchar(1000) | Yes | Alert description |
| System | bit | No | Defines if the alert belongs to the system (all tenants) |
| Code | int | No | Identifies the alert (cannot be used as primary key because multiple tenants could use the same codes) |
| TenantId | int | Yes | Tenant foreign key |

### dbo.Areas

Stores data about areas where meters can be installed. The synoptic uses this data to allow navigation between related areas.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(50) | No | Area name |
| Description | nvarchar(1000) | Yes | Area description |
| SvgShapeId | int | Yes | Svg shape foreign key |
| FloorPlanId | int | Yes | Floor plan foreign key |
| ParentAreaId | int | Yes | Parent area foreign key (references this table 'Id' column) |
| TenantId | int | No | Tenant foreign key |

### dbo.AspNetRoles

It's part of the ASP.NET Identity system. Stores data about user roles (levels of permission).

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | nvarchar(128) | No | Surrogate key |
| Name | nvarchar(256) | No | Role name |

### dbo.AspNetUserClaims

It's part of the ASP.NET Identity system. Stores data used by the claims authorization mechanism. Currently not used.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| UserId | nvarchar(128) | No | User foreign key |
| ClaimType | nvarchar(-1) | Yes | Unique claim identifier |
| ClaimValue | nvarchar(-1) | Yes | Unique right identifier |

### dbo.AspNetUserLogins

It's part of the ASP.NET Identity system. Stores data about logins made using external login providers. Currently not used.

| Column | Data type | Null | Description |
|---|---|---|---|
| LoginProvider | nvarchar(128) | No | Name of the service which provided the login |
| ProviderKey | nvarchar(128) | No | Unique key associated with the user on the login provider |
| UserId | nvarchar(128) | No | User foreign key |

**dbo.AspNetUserRoles**

It's part of the ASP.NET Identity system. Associates user with roles.

| Column | Data type | Null | Description |
|---|---|---|---|
| UserId | nvarchar(128) | No | User foreign key |
| RoleId | nvarchar(128) | No | Role foreign key |

**dbo.AspNetUsers**

It's part of the ASP.NET Identity system. Contains information about the users that can be authenticated in the system.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | nvarchar(128) | No | Surrogate key |
| Created | int | No | Date-time when the user was created |
| Modified | datetime | Yes | Date-time when the user was modified |
| LastLogin | datetime | Yes | Last date-time when the user logged in |
| Enabled | datetime | No | Defines if the user can log in (Enabled=1) |
| FullName | nvarchar(150) | Yes | User full name |
| Email | nvarchar(256) | Yes | User email |
| EmailConfirmed | bit | No | Defines if the user email has been confirmed, with a confirmation code sent to the user's email (EmailConfirmed=1) |
| PasswordHash | nvarchar(-1) | Yes | User password hash and salt |
| SecurityStamp | nvarchar(-1) | Yes | Random value that should change when the user credentials have changed |
| PhoneNumber | nvarchar(-1) | Yes | User phone number |
| PhoneNumberConfirmed | bit | No | Defines if the user phone number has been confirmed, with a SMS message sent to the user's phone (PhoneNumberConfirmed=1) |
| TwoFactorEnabled | bit | No | Defines if this user has two-factor authentication enabled (TwoFactorEnabled=1) |
| LockoutEndDateUtc | datetime | Yes | Defines the date after which the user can log in, if the 'LockoutEnabled' column value is '1' |
| LockoutEnabled | bit | No | Defines if the user is not allowed to log in, if the date in the 'LockoutEndDateUtc' column is greater than the current date (LockoutEnabled=1) |
| AccessFailedCount | int | No | Number of failed login attempts of the user |
| UserName | nvarchar(256) | No | User name (used to log in) |
| TenantId | int | Yes | Tenant foreign key |

### dbo.BatteryStatus

Stores historical data about meter batteries.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | bigint | No | Surrogate key |
| MeasurementDateTime | datetimeoffset | No | Date-time when the battery status was acquired |
| Millivolts | int | No | Battery voltage, in millivolts |
| CurrentCapacity | int | Yes | Battery current capacity, in milliwatts-hour |
| TotalCapacity | int | Yes | Battery total capacity, in milliwatts-hour |
| MeterId | uniqueidentifier | No | Meter foreign key |

### dbo.CustomCommandFields

Stores data about fields that that make up custom commands.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Type | int | No | Field type (variable or constant) |
| DataType | int | No | Field data type (e.g. 32-bit signed integer) |
| Name | nvarchar(50) | No | Field name |
| Value | nvarchar(4000) | Yes | Field value, if the 'Type' column value is 'constant' |
| CustomCommandId | int | No | Custom command foreign key |

### dbo.CustomCommands

Stores custom commands that can be added to meter templates.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(40) | No | Custom command name |
| MeterTemplateId | int | No | Meter template foreign key |
| TenantId | int | Yes | Tenant foreign key |

**dbo.DashboardCharts**

Stores chart settings for dashboard charts.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| ChartType | int | No | Chart type (e.g. bar chart) |
| URL | nvarchar(255) | Yes | URL of the chart title link |
| Title | nvarchar(40) | No | Chart title |
| Subtitle | nvarchar(30) | Yes | Chart subtitle |
| Color | nvarchar(25) | Yes | Chart box color |
| Min | int | No | Minimum value to be displayed on this chart (minimum scale value) |
| Max | int | No | Maximum value to be displayed on this chart (maximum scale value) |
| Size | int | No | Chart size (1=small) |
| Order | int | No | Defines the order of presentation (1=first) |
| TimeFrame | int | No | Time frame of the readings from the data sources |
| DashboardId | uniqueidentifier | No | Dashboard foreign key |

**dbo.DashboardChartSources**

Stores data about data sources to be displayed on dashboard charts.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| DisplayName | nvarchar(20) | No | Data source name to be displayed to the user |
| Color | nvarchar(25) | Yes | Data source color on chart |
| ComputeType | int | No | Readings grouping function, does not apply to gauges, as gauges display real time readings |
| MeterId | uniqueidentifier | No | Meter foreign key |
| MeterSensorId | int | Yes | Meter sensor foreign key |
| DashboardChartId | int | No | Dashboard chart foreign key |

**dbo.Dashboards**

Contains dashboard data. Dashboards are displayed on the website homepage and contain readings graphs.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | uniqueidentifier | No | Surrogate key |
| Name | nvarchar(20) | No | Dashboard name |
| Order | int | No | Defines the order of presentation (1=first) |
| UserId | uniqueidentifier | No | User foreign key |

**dbo.FloorPlans**

Stores data about areas floor plan images.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Width | int | Yes | Floor plan image width |
| Height | int | Yes | Floor plan image height |
| ImagePath | nvarchar(260) | Yes | Floor plan image path |
| AreaId | int | No | Area foreign key |

**dbo.Gateways**

Stores gateway data and settings.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | uniqueidentifier | No | Surrogate key |
| Name | nvarchar(50) | No | Gateway name |
| Enabled | bit | No | Defines if the IIS web service and website can send or receive data from the gateway (enabled=1) |
| Latitude | decimal(9,6) | Yes | Gateway latitude (geographic coordinate) |
| Longitude | decimal(9,6) | Yes | Gateway longitude (geographic coordinate) |
| Altitude | smallint | Yes | Gateway altitude (meters above sea level) |
| LoRaAppId | int | Yes | Gateway LoRa application id |
| Debug | bit | No | Defines if the gateway should increase the detail of it's web service responses and log records detail |
| Protocol | nvarchar(5) | No | Gateway web service protocol |
| Host | nvarchar(253) | No | Gateway web service DNS name or IP address |
| Port | int | No | Gateway web service port |
| Password | nvarchar(128) | No | Gateway web service password |
| ApiVersion | int | No | Gateway API version |
| WebServiceProtocol | nvarchar(5) | No | System web service protocol |
| WebServiceHost | nvarchar(253) | No | System web service DNS name or IP address |
| WebServicePort | int | No | System web service port |
| WebServicePassword | nvarchar(128) | No | System web service password |
| TenantId | int | Yes | Tenant foreign key |

**dbo.LogRecords**

Stores events that occur in the system.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | bigint | No | Surrogate key |
| Level | smallint | No | Event severity level |
| CreatedAt | datetimeoffset | No | Log record creation date-time |
| SourceName | nvarchar(128) | Yes | Name of the log record source (e.g. meter name) |
| SourceId | nvarchar(256) | Yes | Id of the log record source |
| Method | nvarchar(512) | Yes | Fully qualified name of the method that created the log record |
| Message | nvarchar(2000) | No | Log record message |
| ExtraInfo | nvarchar(2000) | Yes | Additional information, such as operation parameters |
| System | bit | No | Defines if the log record belongs to the system (all tenants) |
| TenantId | int | Yes | Tenant foreign key |

**dbo.MeterSensors**

Stores meter sensor settings.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| DatabaseName | nvarchar(40) | No | Database column name |
| DisplayName | nvarchar(30) | No | Sensor name to be displayed to the user |
| DataType | tinyint | No | Database data type |
| IntegerDigits | tinyint | No | Together with the 'DecimalPlaces' column value, defines the 'precision' parameter if database data type is 'decimal' |
| DecimalPlaces | tinyint | No | Defines the 'scale' parameter if 'DataType' column value is 'decimal', otherwise just limits the number of decimal places to be displayed |
| Unit | nvarchar(10) | Yes | Sensor measurement unit (e.g. L) |
| ValueType | tinyint | No | Defines if the sensor measurement is absolute (e.g. a thermometer reports absolute temperatures) or relative to the last reading (e.g. water meter reading always increase) |
| ChartType | tinyint | No | Chart type used to display this sensor data |
| ChartMinValue | bigint | No | Minimum value to be displayed on this sensor chart (minimum scale value) |
| ChartMaxValue | bigint | No | Maximum value to be displayed on this sensor chart (maximum scale value) |
| ChartColor | nvarchar(25) | Yes | Color to represent this sensor data series on a chart |
| MeterTemplateId | int | No | Meter template foreign key |

**dbo.Meters**

Stores meter data and settings.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | uniqueidentifier | No | Surrogate key |
| Name | nvarchar(50) | No | Meter name |
| SerialNumber | nvarchar(128) | Yes | Serial number |
| BatteryPercentage | tinyint | Yes | Battery state of charge |
| LastBatteryUpdate | datetimeoffset | Yes | Battery state of charge last update date |
| CreateDate | datetimeoffset | No | Meter creation date |
| LastUpdate | datetimeoffset | Yes | Meter last update date |
| Enabled | bit | No | Defines if the system can send or receive data from the meter (Enabled=1) |
| Eui | nvarchar(23) | Yes | Meter EUI |
| Latitude | decimal(9,6) | Yes | Meter latitude (geographic coordinate) |
| Longitude | decimal(9,6) | Yes | Meter longitude (geographic coordinate) |
| Altitude | smallint | Yes | Meter altitude (meters above sea level) |
| SamplingIntervalMinutes | int | No | Interval at which the meter acquires readings |
| CommIntervalMinutes | int | No | Interval at which the meter sends readings |
| AlertReverseFlux | bit | No | Defines if the meter should send reverse flux alerts (AlertReverseFlux=1) |
| AlertMagneticFraud | bit | No | Defines if the meter should send magnetic fraud alerts (AlertMagneticFraud=1) |
| AlertHighConsumption | bit | No | Defines if the meter should send high consumption alerts (AlertHighConsumption=1) |
| AlertZeroConsumption | bit | No | Defines if the meter should send zero consumption alerts (AlertZeroConsumption=1) |
| AlertLowBattery | bit | No | Defines if the meter should send low battery alerts (AlertLowBattery=1) |
| AlertMemoryError | bit | No | Defines if the meter should send memory error alerts (AlertMemoryError=1) |
| AlertWrongDateTime | bit | No | Defines if the meter should send alerts if the meter date-time is too far apart from the system date-time (AlertWrongDateTime=1) |
| MeterTemplateId | int | No | Meter template foreign key |
| GatewayId | uniqueidentifier | Yes | Gateway foreign key |
| AreaId | int | Yes | Area foreign key |
| TenantId | int | No | Tenant foreign key |

**dbo.MeterTemplateAlerts**

Associates alerts with meter templates.

| Column | Data type | Null | Description |
|---|---|---|---|
| MeterTemplateId | int | No | Meter template foreign key |
| AlertId | int | No | Alert foreign key, associated only with alerts with value 'system=false' |
| Enabled | bit | No | Defines if the system should consider the association between the alert and the meter template |

**dbo.MeterTemplates**

Stores data about meter templates.

| Column | Data type | Null | description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(25) | No | Meter template name |
| Description | nvarchar(25) | Yes | Meter template description |
| Brand | nvarchar(25) | Yes | Brand of meters that this meter template supports (if applicable) |
| Model | nvarchar(25) | Yes | Model of meters that this meter template supports (if applicable) |
| TenantId | int | Yes | Tenant foreign key |

**dbo.Notifications**

Stores notifications. Notifications may be issued by alerts, triggers, meter command responses and the system.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | bigint | No | Surrogate key |
| Title | nvarchar(250) | No | Notification title |
| DateTime | datetimeoffset | No | Date-time when the notification was issued |
| Message | nvarchar(1000) | Yes | Notification message |
| Viewed | bit | No | Defines if the notification has been viewed by at least one user (Viewed=1) |
| Type | tinyint | No | Notification type (e.g. meter log) |
| SourceId | Int | Yes | Id of the notification source (int) |
| SourceGuid | uniqueidentifier | Yes | Id of the notification source (Guid) |
| SourceName | nvarchar(250) | Yes | Name of the notification source (e.g. meter name) |
| Subtype | tinyint | Yes | Notification subtype (e.g. battery status report) |
| TenantId | int | Yes | Tenant foreign key |

**dbo.SignalMappingPoints**

Stores data about meters RX and TX signal conditions at the meter location.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | bigint | No | Surrogate key |
| Latitude | decimal(9,6) | No | Point latitude (geographic coordinate) |
| Longitude | decimal(9,6) | No | Point longitude (geographic coordinate) |
| Altitude | smallint | No | Point altitude (meters above sea level) |
| MeasurementDateTime | datetimeoffset | No | Date-time when the signal conditions were received by the gateway |
| MeterRSSI | smallint | No | RSSI measured by the meter |
| MeterSNR | decimal(5,2) | No | Signal-to-noise ratio measured by the meter |
| MeterSpreadingFactor | tinyint | No | Spreading factor measured by the meter |
| MeterBandwidth | smallint | No | Bandwidth of the message received by the meter in kHz |
| MeterFrequency | int | No | Frequency of the message received by the meter in kHz |
| MeterChannel | tinyint | No | Channel of the message received by the meter |
| MeterCodeRateN | tinyint | No | Code rate numerator (e.g. N/5) of the message received by the meter |
| MeterCodeRateD | tinyint | No | Code rate denominator (e.g. 4/D) of the message received by the meter |
| GatewayRSSI | smallint | No | RSSI measured by the gateway |
| GatewaySNR | decimal(5,2) | No | Signal-to-noise ratio measured by the gateway |
| GatewaySpreadingFactor | tinyint | No | Spreading factor measured by the gateway |
| GatewayBandwidth | smallint | No | Bandwidth of the message received by the gateway in kHz |
| GatewayFrequency | int | No | Frequency of the message received by the gateway in kHz |
| GatewayChannel | tinyint | No | Channel of the message received by the gateway |
| GatewayCodeRateN | tinyint | No | Code rate numerator (e.g. N/5) of the message received by the gateway |
| GatewayCodeRateD | tinyint | No | Code rate denominator (e.g. 4/D) of the message received by the gateway |
| SignalMappingSessionId | int | No | Signal mapping session foreign key |

**dbo.SignalMappingSessions**

Groups meter signal mapping points in sessions.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(50) | No | Session name (e.g. 'Test downtown signal coverage') |
| Description | nvarchar(1000) | Yes | Session description |
| Created | datetimeoffset | No | Date-time when the session was created |
| Active | bit | No | Defines if new signal mapping messages should be stored in this session (Active=1). There can only be only active session per meter |
| MeterId | uniqueidentifier | No | Meter foreign key |

**dbo.SvgShapes**

Stores data about vector shapes into which an area can be divided.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Type | tinyint | No | Defines the SVG shape type |
| D | nvarchar(-1) | Yes | Path data |
| Cx | int | Yes | Circle and ellipse center X position |
| Cy | int | Yes | Circle and ellipse center Y position |
| R | int | Yes | Circle and ellipse radius |
| Rx | int | Yes | Ellipse horizontal radius |
| Ry | int | Yes | Ellipse vertical radius |
| X | int | Yes | Rectangle left position |
| Y | int | Yes | Rectangle top position |
| Width | int | Yes | Rectangle width |
| Height | int | Yes | Rectangle height |
| AreaId | int | No | Area foreign key |

**dbo.Tenants**

Groups users into sets that have access to the same data.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| Name | nvarchar(50) | No | Tenant name |
| Enabled | bit | No | Defines if users belonging to this tenant can log in (Enabled=1) |
| WebsiteName | nvarchar(20) | Yes | Website name, to be displayed on the page title |

### dbo.TriggerConditions

Stores conditions in which triggers will be triggered.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | int | No | Surrogate key |
| ReadingInterval | tinyint | No | Test condition on each reading or on a group of readings (e.g. daily) |
| ComputeType | tinyint | Yes | Group function when 'ReadingInterval' column value is set to any group of readings (e.g. average) |
| Condition | tinyint | No | Trigger condition (e.g. between 'LowerValue' and 'UpperValue' column values) |
| LowerValue | decimal(38,6) | Yes | Condition lower value (if applicable) |
| UpperValue | decimal(38,6) | Yes | Condition upper value (if applicable) |
| ReadingComparison | tinyint | No | Test condition on absolute reading value or relative to last reading value (difference) |
| TriggerId | int | No | Trigger foreign key |
| MeterSensorId | int | Yes | Meter sensor foreign key |

### dbo.Triggers

Stores data about triggers that can be triggered by meters.

| Column | Data type | Null | Description |
|---|---|---|---|
| Id | Int | No | Surrogate key |
| Name | nvarchar(50) | No | Trigger name |
| Description | nvarchar(1000) | Yes | Trigger description |
| MeterTemplateId | int | No | Meter template foreign key |
| TenantId | int | Yes | Tenant foreign key |

# Anexo C: Especificação do *web service* de *gateway*

De seguida é apresentado um resumo da especificação do *web service* de *gateway*, os métodos que compõem este *web service* são descritos de forma sucinta e os respetivos parâmetros dos pedidos, esquemas dos pedidos e respostas são apresentados.

### Common responses

These responses can be sent by any method

| Code | Description | JSON schema |
|---|---|---|
| 500 | This error happens when the gateway configuration is missing. Gateway configuration is needed for authentication. | { "message": "Missing gateway configuration" } |
| 401 | Authentication failed, invalid gateway GUID or password. | { "message": "Authentication failed" } |

### Meter

### 1 – Valve control

| | |
|---|---|
| Method | POST |
| Path | /api/meters/valve-control |
| Description | Sends a command to a meter to open or close its valve. |

### URI Parameters

None

### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| state | boolean | Boolean | No |

**JSON schema example**

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "state": true
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the valve command was added to the queue of messages to be sent to the meter. | { "message": "Valve command was sent" } |
| 400 | Valve state is missing or its data type is not “boolean”. | { "message": "Invalid valve state" } |
| 400 | The meter GUID is not present in the request. | { "message": "Missing meter GUID" } |
| 404 | No meter was found with the specified GUID in the gateway meter list. | { "message": "Meter is not associated with this gateway" } |
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | { "message": "Meter has no EUI configured" } |
| 500 | Gateway could does not have data about any meter. | { "message": "No meter data" } |

**2 – Valve status**

| | |
|---|---|
| Method | GET |
| Path | /api/meters/valve-state |
| Description | Returns the last valve state reported by the meter. |

**URI Parameters**

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |

**Body Parameters**

None

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | The last valve status received is returned in the response. | { " valveState": true } |
| 404 | No message was received from the meter, so the valve status is unknown. | { "message": "Meter valve status is unknown" } |
| 400 | The meter GUID is not present in the request. | { "message": "Missing meter GUID" } |
| 404 | No meter was found with the specified GUID in the gateway meter list. | { "message": "Meter is not associated with this gateway" } |
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | { "message": "Meter has no EUI configured" } |
| 500 | Gateway could does not have data about any meter. | { "message": "No meter data" } |

**3 – Alert configuration**

| Method | PUT |
|---|---|
| Path | /api/meters/configure-alerts |
| Description | Changes a meter's alert settings. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| reverseFlux | boolean | Boolean | No |
| magneticFraud | boolean | Boolean | No |
| highConsumption | boolean | Boolean | No |
| zeroConsumption | boolean | Boolean | No |
| lowBattery | boolean | Boolean | No |
| memoryError | boolean | Boolean | No |
| wrongDateTime | boolean | Boolean | No |

**JSON schema example**

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "reverseFlux": true,
  "magneticFraud": false,
  "highConsumption": true,
  "zeroConsumption": false,
  "lowBattery": true,
  "memoryError": true,
  "wrongDateTime": true
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the alert settings were added to the queue of messages to be sent to the meter. | `{ "message": "Alert configuration was sent" }` |
| 400 | Some parameters are missing or their data types are incorrect. | `{ "message": "Some alert settings were invalid", "alerts": [ [ "lowBattery", "Parameter is missing" ] ] }` |
| 400 | The meter GUID is not present in the request. | `{ "message": "Missing meter GUID" }` |
| 404 | No meter was found with the specified GUID in the gateway meter list. | `{ "message": "Meter is not associated with this gateway" }` |
| 500 | This error happens when no EUI was configured in the website. | `{ "message": "Meter has no EUI config` |

| | | |
|---|---|---|
| | This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | ured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

### 4 – Request battery status

| | |
|---|---|
| Method | POST |
| Path | /api/meters/battery-status |
| Description | Requests a meter's battery status. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |

#### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad"
}
```

#### Responses

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the request for the meter's battery status was added to the queue of messages to be sent to the meter. | {<br>"message": "Battery status request was sent"<br>} |
| 400 | The meter GUID is not present in the request. | {<br>"message": "Missing meter GUID"<br>} |
| 404 | No meter was found with the specified GUID in the gateway meter list. | {<br>"message": "Meter is not associated with this gateway"<br>} |
| 500 | This error happens when no EUI was configured in the website. | {<br>"message": "Meter has no EUI config |

| | | |
|---|---|---|
| | This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | ured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

### 5 – Get reading

| | |
|---|---|
| Method | POST |
| Path | /api/meters/get-reading |
| Description | Requests a meter's reading. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |

#### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad"
}
```

#### Responses

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the request for the meter's reading was added to the queue of messages to be sent to the meter. | {<br>"message": "Meter reading request was sent"<br>} |
| 400 | The meter GUID is not present in the request. | {<br>"message": "Missing meter GUID"<br>} |
| 404 | No meter was found with the specified GUID in the gateway meter list. | {<br>"message": "Meter is not associated with this gateway"<br>} |
| 500 | This error happens when no EUI was configured in the website. | {<br>"message": "Meter has no EUI config |

| | | |
|---|---|---|
| | This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | ured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

### 5 – Set reading

| | |
|---|---|
| Method | PUT |
| Path | /api/meters/set-reading |
| Description | Sets a meter's current reading. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| reading | number | Decimal | No |

#### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "reading": 5000
}
```

#### Responses

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the request for changing the meter's current reading was added to the queue of messages to be sent to the meter. | {<br>"message": "Reading change request was sent"<br>} |
| 400 | Reading is missing or its data type is not "number". | {<br>"message": "Invalid meter reading"<br>} |
| 400 | The meter GUID is not present in the request. | {<br>"message": "Missing meter GUID"<br>} |

| | | |
|---|---|---|
| 404 | No meter was found with the specified GUID in the gateway meter list. | {<br>"message": "Meter is not associated with this gateway"<br>} |
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | {<br>"message": "Meter has no EUI configured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

### 6 – Set intervals

| | |
|---|---|
| Method | PUT |
| Path | /api/meters/set-intervals |
| Description | Sets a meter's sampling and communication intervals. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| sampling | number | Int32 | Yes |
| communication | number | Int32 | Yes |

#### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "sampling": 60,
  "communication": 180
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the request for changing the meter's sampling and/or communication intervals was added to the queue of messages to be sent to the meter. | {<br>"message": "Intervals change request was sent"<br>} |
| 400 | Some parameters are missing or their data types are incorrect. | {<br>"message": "Some intervals were invalid",<br>"alerts": [<br>[<br>"sampling",<br>"Parameter type is invalid - expected 'number', got 'string'"<br>]<br>]<br>} |
| 400 | The meter GUID is not present in the request. | {<br>"message": "Missing meter GUID"<br>} |
| 404 | No meter was found with the specified GUID in the gateway meter list. | {<br>"message": "Meter is not associated with this gateway"<br>} |
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | {<br>"message": "Meter has no EUI configured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

**7 – Set Schedule**

| | |
|---|---|
| Method | PUT |
| Path | /api/meters/set-schedule |
| Description | Sets a meter's scheduled actions. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| scheduledActions | array (of objects) | IList<ScheduledAction> | No |
| scheduleType | number | Byte | No |
| dateAndOrTime | string | DateTimeOffset | No |
| daysOfTheWeek | array (of numbers) | IList<Byte> | Yes |
| action | number | Byte | No |

**JSON schema example**

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "scheduledActions": [
    {
      "scheduleType": 2,
      "dateAndOrTime": "1970-01-01T18:30:00Z",
      "daysOfTheWeek": [4],
      "action": 1
    }
  ]
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the request for changing the meter's sampling and/or communication intervals was added to the queue of messages to be sent to the meter. | `{ "message": "Intervals change request was sent" }` |
| 400 | Some parameters are missing or their data types are incorrect. | `{ "message": "Some intervals were invalid", "alerts": [ [ "sampling", "Parameter type is invalid - expected 'number', got 'string'" ] ] }` |
| 400 | The meter GUID is not present in the request. | `{ "message": "Missing meter GUID" }` |
| 404 | No meter was found with the specified GUID in the gateway meter list. | `{ "message": "Meter is not associated with this gateway" }` |

| | | |
|---|---|---|
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | {<br>"message": "Meter has no EUI configured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

### 8 – Send custom command

| | |
|---|---|
| Method | POST |
| Path | /api/meters/custom-command |
| Description | Sends a custom command to a meter. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| commandFields | array (of objects) | IList<CommandField> | No |
| dataType | number | Byte | No |
| size | number | UInt16 | Yes |
| value | string, number or boolean | String, Decimal or Boolean | No |
| name | string | String | Yes |

#### JSON schema example


```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "commandFields": [
    {
      "dataType": 10,
      "size": null,
      "value": 120,
      "name": "Duration (minutes)"
    }
  ]
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, the custom command was added to the queue of messages to be sent to the meter. | {<br>"message": "Custom command request was sent"<br>} |
| 400 | Command fields are missing or their data types are incorrect. | {<br>"message": "Command fields are missing"<br>} |
| 400 | Some fields values or data types are missing or incorrect. | {<br>"message": "Some command fields were invalid",<br>"fields": [<br>{<br>"number": 3,<br>"error": "Invalid data type",<br>"name": " Duration (minutes)"<br>}<br>]<br>} |
| 400 | The meter GUID is not present in the request. | {<br>"message": "Missing meter GUID"<br>} |
| 404 | No meter was found with the specified GUID in the gateway meter list. | {<br>"message": "Meter is not associated with this gateway"<br>} |
| 500 | This error happens when no EUI was configured in the website. This gateway implementation communicates using the LoRaWAN protocol, and this protocol requires a meter EUI to send messages to the meter. | {<br>"message": "Meter has no EUI configured"<br>} |
| 500 | Gateway could does not have data about any meter. | {<br>"message": "No meter data"<br>} |

**Gateway**

**1 – Update associated meters**

| | |
|---|---|
| Method | PUT |
| Path | /api/gateway/meters |
| Description | Updates the list of meters associated with the gateway, which is stored in the gateway memory. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| metersData | array (of objects) | IList<MeterData> | No |
| eui | string | String | No |
| guid | string | Guid | No |
| template | number | Int32 | No |
| name | string | String | Yes |

**JSON schema example**

```
{
  "metersData": [
    {
      "eui": "3f-9e-f1-5a-ae-db",
      "guid": "8b5b935e162944e9930bcb96c2b6e8ad",
      "template": 2,
      "name": "Water meter #15",
    }
  ]
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, meter list was updated in the gateway memory. | `{ "message": "Meter list was updated" }` |
| 400 | The “metersData” property is missing or its data type is incorrect. | `{ "message": "Missing 'metersData' property" }` |
| 400 | Some parameters are missing or their data types are incorrect. | `{ "message": "Data from some meters is invalid", "meters": [ { "number": 1, "error": "Missing EUI", "name": "Water meter #15" } ] }` |
| 500 | Error while updating meter list (e.g. insufficient file permissions). | `{ "message": "Could not update meters list" }` |

### 2 – Update associated templates

| Method | PUT |
|---|---|
| Path | /api/gateway/meter-templates |
| Description | Updates the list of meter templates used by the meters associated with the gateway. This list is stored in the gateway memory. |

#### URI Parameters

None

#### Body Parameters

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| templates | array (of objects) | IList<MeterTemplate> | No |
| id | number | Int32 | No |
| name | string | String | Yes |
| sensors | array (of strings) | IList<String> | No |

#### JSON schema example

```
{
  "templates": [
    {
      "id": 2,
      "name": "Water meter",
      "sensors": [
        "liters"
      ]
    }
  ]
}
```

#### Responses

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, meter template list was updated in the gateway memory. | { "message": "Meter template list was updated" } |
| 400 | The “templates” property is missing or its data type is incorrect. | { "message": "Missing 'templates' property" } |
| 400 | Some parameters are missing or their data types are incorrect. | { "message": "Data from some meter templates is invalid", |

| | | |
|---|---|---|
| | | "templates": [<br>{<br>"number": 1,<br>"error": "Missing sensors",<br>"name": "Water meter"<br>}<br>]<br>} |
| 500 | Error while updating meter list (e.g. insufficient file permissions). | {<br>"message": "Could not update meter templates list"<br>} |

### 3 – Change configuration

| | |
|---|---|
| Method | PUT |
| Path | /api/gateway/settings |
| Description | Updates gateway settings used by the gateway to authenticate requests received by the web service, to make requests to the IIS web service, and to route messages to the appropriate LoRa application. These settings are stored in the gateway memory. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Website data type | Null |
|---|---|---|---|
| wsHost | string | string | No |
| wsPort | number | UInt16 | No |
| wsProtocol | string | String | Yes |
| wsPassword | string | String | No |
| gwId | string | string | |
| gwPassword | string | String | |
| appId | number | Int32 | |
| debug | boolean | Boolean | |
| testSettings | boolean | Boolean | |

### JSON schema example

```
{
  "wsHost": "server.lan",
  "wsPort": 8080,
  "wsProtocol": "https",
  "wsPassword": "3eb5b3a63ac74e6ba6b956e6f41bb0d9",
  "gwId": "41ec19fd16554735975cc9e5051132a3",
  "gwPassword": "d56509666eff4e10b10b95736bf04860",
  "appId": 1,
  "debug": true,
  "testSettings": false
}
```

### Responses

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, gateway settings were changed in the gateway memory. | `{ "message": "Gateway settings were updated" }` |
| 400 | Some parameters are missing, their data types are incorrect or their values are incorrect. | `{ "message": "Some settings are invalid", "settings": [ { "name": "wsPort", "error": "Value is out of range, it shoud be between 1 and 65535" } ] }` |
| 400 | The “testSettings” parameter was set to “true” and the settings failed the test (e.g. the gateway could not connect to the IIS web service). When the test fails the gateway settings are not changed. | `{ "message": "Some settings failed the test", "settings": [ { "test": "Connect to IIS web service", "error": "No response from server" } ] }` |
| 500 | Error while changing gateway settings (e.g. insufficient file permissions). | `{ "message": "Could not change gateway settings" }` |

# Anexo D: Especificação do *web service* no IIS

De seguida é apresentado um resumo da especificação do *web service* no IIS, os métodos que compõem os vários controllers são descritos de forma sucinta e os respetivos parâmetros dos pedidos, esquemas dos pedidos e respostas são apresentados.

### Common responses

These responses can be sent by any method

| Code | Description | JSON schema |
|---|---|---|
| 401 | Authentication failed, invalid gateway GUID or password. | `{ "message": "Authentication failed" }` |

### Controller "Alerts"

### 1 – Insert alert

| | |
|---|---|
| Method | POST |
| Path | /api/Alerts/Insert |
| Description | Inserts an alert in the database and sends a notification to the website. The alert is identified by the code. |

### URI Parameters

None

### Body Parameters

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| code | number | Int32 | No |
| issueDateTime | string | DateTimeOffset | No |

### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "code": 4,
  "dateTime": "2017-04-03T14:50:40.1454434+01:00"
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, an alarm for the specified meter and event was inserted in the database. | <pre>{<br>  "message": "Alert inserted"<br>}</pre> |
| 404 | No meter found with the specified GUID. | <pre>{<br>  "message": "Meter not found"<br>}</pre> |
| 400 | Meter is not associated with the gateway that sent the request | <pre>{<br>  "message": " Meter is not associated with the gateway that sent the request "<br>}</pre> |
| 404 | No alert found with the specified code | <pre>{<br>  "message": "Alert not found"<br>}</pre> |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the property “code” is missing) | <pre>[<br>  {<br>  "PropertyName": "code",<br>  "Error": "The code field is required."<br>  }<br>]</pre> |
| 500 | An error occurred while storing the alert in the database, this error can be viewed in the system log | <pre>{<br>  "message": "Error storing the alert in the database"<br>}</pre> |

**2 - Insert alerts for multiple meters**

| | |
|---|---|
| Method | POST |
| Path | /api/Alerts/InsertAlertMultipleMeters |
| Description | Stores multiple alerts for all specified meters and sends notifications to the website. This method is used to send all pending alerts. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterAlerts | array (of objects) | IEnumerable<MeterAlerts> | No |
| alerts | array (of objects) | IEnumerable<Alert> | No |
| meterId | string | Guid | No |
| code | number | Int32 | No |
| issueDateTime | string | DateTimeOffset | No |

**JSON schema example**

```
{
  "meterAlerts": [
    {
      "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
      "alerts": [
        {
          "code": 4,
          "dateTime": "2017-04-03T14:50:40.1454434+01:00"
        }
      ]
    }
  ]
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, alerts for all specified meters were inserted in the database and notifications were sent to the website. | `{ "message": "Alerts from all meters were stored" }` |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the “alerts” property is missing) | `[ { "PropertyName": "alerts", "Error": "The alerts field is required." } ]` |
| 404 | At least one of the meters was not found with the specified GUID. | `{ "message": "Some meters were not found", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] }` |
| 404 | At least one of the alerts was not found with the specified code. | `{ "message": "Some alerts were not found", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] }` |

| | | |
|---|---|---|
| | | }<br> |
| 500 | An error occurred storing alerts in the database (e.g. meter was deleted while inserting alerts), this error can be viewed in the system log | {<br>"message": "Error storing alerts in the database",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |
| 400 | At least one of the meters is not associated with the gateway that sent the request | {<br>"message": "Some meters are not associated with the gateway that sent the request",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |

**Controller “Readings"**

**1 – Insert associative readings**

| | |
|---|---|
| Method | POST |
| Path | /api/Readings/InsertAssociative |
| Description | Insert readings for the specified meter. This method is appropriate when the readings come from different sensors and the order may not be the same every time (asynchronous readings acquisition).<br>This method is slower than the associative readings insertion method (the request JSON schema is more complex). |

**URI parameters**

None

**Body parameters**

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| readings | array (of objects) | IEnumerable<Reading> | No |
| dateTime | string | DateTimeOffset | No |
| sensors | array (of objects) | IEnumerable<Sensor> | No |
| name | string | String | No |
| value | string | String | No |

**JSON schema example**

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "readings": [
    {
      "dateTime": "2017-03-30T12:30:00.000Z",
      "sensors": [
        {
          "name": "liters",
          "value": 5000
        }
      ]
    }
  ]
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, readings for the specified meter were inserted in the database. | `{ "message": "Reading inserted" }` |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the property “meterId” is missing) | `[ { "PropertyName": "meterId", "Error": "The meterId field is required." } ]` |
| 404 | No meter found with the specified GUID. | `{ "message": "Meter not found" }` |
| 500 | An error occurred storing readings in the database (e.g. web service could not connect to the database) | `{ "message": "Error storing readings in the database" }` |
| 400 | Meter is not associated with the gateway that sent the request | `{ "message": "Meter is not associated with the gateway that sent the request" }` |

**2 - Insert associative readings for multiple meters**

| | |
|---|---|
| Method | POST |
| Path | /api/Readings/InsertAssociativeMultipleMeters |
| Description | Insert readings for all specified meters. This method is used to send all pending readings. |

### URI parameters

None

### Body parameters

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterReadings | array (of objects) | IEnumerable<AssociativeMeterReadings> | No |
| meterId | string | Guid | No |
| readings | array (of objects) | IEnumerable<Reading> | No |
| dateTime | string | DateTimeOffset | No |
| sensors | array (of objects) | IEnumerable<Sensor> | No |
| name | string | String | No |
| value | string | String | No |

### JSON schema example

```
{
  "meterReadings": [
    {
      "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
      "readings": [
        {
          "dateTime": "2017-03-30T12:30:00.000Z",
          "sensors": [
            {
              "name": "liters",
              "value": 5000
            }
          ]
        }
      ]
    }
  ]
}
```

### Responses

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, readings for all specified meters were inserted in the database. | `{ "message": "Readings inserted for all meters" }` |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the property “meterReadings” is missing) | `[ { "PropertyName": "meterReadings", "Error": "The meterReadings field is required." } ]` |

| | | |
|---|---|---|
| 404 | At least one of the meters was not found with the specified GUID. | {<br>"message": "Meter not found",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |
| 500 | An error occurred storing readings for the specified meters in the database (e.g. a decimal number was received for a meter sensor which expects an integer number) | {<br>"message": "Error storing readings in the database",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |
| 400 | At least one of the meters is not associated with the gateway that sent the request | {<br>"message": "Some meters are not associated with the gateway that sent the request",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |

### 3 – Insert indexed readings

| | |
|---|---|
| Method | POST |
| Path | /api/Readings/InsertIndexed |
| Description | Insert readings for the specified meter. This method is appropriate when the readings come from only one sensor or the order which they arrive at the meter is always the same (e.g.: serial readings acquisition). This method is faster than the associative readings insertion method (the request JSON schema is simple). |

### URI parameters

None

### Body parameters

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| sensors | array (of strings) | IList<String> | No |
| dateTime | array (of strings) | IList<DateTimeOffset> | No |
| readings | array (of arrays of numbers) | IList<IList<decimal>> | No |

### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "sensors": ["liters"],
  "dateTime": ["2017-03-30T12:30:00.000Z"],
  "readings": [
    [5000]
  ]
}
```

### Readings table view

| | sensor [0] | sensor [2] | sensor [2] | ... | sensor [n] |
|---|---|---|---|---|---|
| dateTime [0] | readings [0][0] | readings [0][1] | readings [0][2] | ... | readings [0][n] |
| dateTime [1] | readings [1][0] | readings [1][1] | readings [1][2] | ... | readings [1][n] |
| dateTime [2] | readings [2][0] | readings [2][1] | readings [2][2] | ... | readings [2][n] |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| dateTime [n] | readings [m][0] | readings [m][1] | readings [m][2] | | readings [m][n] |

### Responses

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, readings for the specified meter were inserted in the database. | `{ "message": "Readings inserted" }` |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the “sensors” property is missing) | `[ { "PropertyName": "sensors", "Error": "The sensors field is required." } ]` |
| 404 | No meter found with the specified GUID. | `{ "message": "Meter not found" }` |
| 500 | An error occurred storing readings in the database (e.g. web service could not connect to the database) | `{ "message": "Error storing readings in the database" }` |
| 400 | Meter is not associated with the gateway that sent the request | `{ "message": "Meter is not associated with the gateway that sent the request" }` |

**Controller "Meters"**

**1 – Get meter data**

| Method | GET |
|---|---|
| Path | /api/Meters/Data |
| Description | Get meter data such as GUID, EUI and template id. Used to associate the meter LoRaWAN EUI with the web service GUID and to associate meter readings with configured template sensors. |

**URI parameters**

None (the gateway GUID used to authenticate is used)

**Body parameters**

None

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, meter data was returned (one meter in this example). | {<br>"metersData": [<br>{<br>"eui": "3f-9e-f1-5a-ae-db",<br>"guid": "8b5b935e162944e9930bcb96c2b6e8ad",<br>"template": 2,<br>"name": "Water meter #15",<br>}<br>]<br>} |
| 404 | No gateway found with the specified GUID (gateway may have been deleted before getting its associated meters). | {<br>"message": "Gateway not found"<br>} |

**2 – Get template data**

| Method | GET |
|---|---|
| Path | /api/Meters/Templates |
| Description | Get meter template data, such as template id, template sensors and template name, for all templates used by the associated meters. Used to generate an appropriate JSON object with the meter sensors to send readings to the web service. |

**URI parameters**

None (the gateway GUID used to authenticate is used)

**Body parameters**

None

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 200 | Successful operation, template data for each meter template used by meters associated with this gateway. | {<br>"templates": [<br>{<br>"id": 2,<br>"name": "Water meter",<br>"sensors": [<br>"liters"<br>]<br>}<br>]<br>} |
| 404 | No gateway found with the specified GUID (gateway may have been deleted before getting its associated meters or templates). | {<br>"message": "Gateway not found"<br>} |

**3 – Insert operation result**

| | |
|---|---|
| Method | POST |
| Path | /api/Meters/InsertOperationResult |
| Description | Inserts an operation result in the database and sends a notification to the website. The operation is identified by the operation code. |

**URI Parameters**

None

**Body Parameters**

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| operationCode | number | Int32 | No |
| resultCode | number | Int32 | No |
| dateTime | string | DateTimeOffset | No |

**JSON schema example**

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "operationCode": 4,
  "resultCode": 0,
  "dateTime": "2017-04-03T14:50:40Z"
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, a result code for the specified meter operation was inserted in the database. | `{ "message": "Operation result inserted" }` |
| 404 | No meter found with the specified GUID. | `{ "message": "Meter not found" }` |
| 400 | Meter is not associated with the gateway that sent the request | `{ "message": " Meter is not associated with the gateway that sent the request " }` |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the property “resultCode” is missing) | `[ { "PropertyName": "resultCode", "Error": "The resultCode field is required." } ]` |
| 404 | No operation found with the specified code | `{ "message": "Operation not found" }` |
| 404 | No result found with the specified code | `{ "message": "Result code not found" }` |
| 500 | An error occurred while storing the operation result in the database, request data such as operation code can be viewed in the system log | `{ "message": "Error storing operation result in the database" }` |

**4 – Report battery status**

| | |
|---|---|
| Method | POST |
| Path | /api/Meters/ReportBatteryStatus |
| Description | Stores battery status information for the specified meter. This information can be used to determine if the battery needs to be replaced or recharged. |

### URI parameters

None

### Body parameters

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterId | string | Guid | No |
| millivolts | number | UInt16 | No |
| currentCapacity | number | UInt32 | No |
| totalCapacity | number | UInt32 | No |

### JSON schema example

```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "millivolts": 3700,
  "currentCapacity": 10500,
  "totalCapacity": 16280
}
```

### Responses

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, battery status was stored in the database. | { "message": "Battery status stored" } |
| 404 | No meter found with the specified GUID. | { "message": "Meter not found" } |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the “millivolts” property is missing) | [ { "PropertyName": "millivolts", "Error": "The millivolts field is required." } ] |
| 500 | An error occurred storing the battery status in the database (e.g.: web service could not connect to the database) | { "message": "Error storing battery status in the database" } |
| 400 | Meter is not associated with the gateway that sent the request | { "message": " Meter is not associated with the gateway that sent the request " } |

### 5 – Report multiple meters battery statuses

| Method | POST |
|---|---|
| Path | /api/Meters/BatteryStatusesMultipleMeters |
| Description | Stores multiple battery status information for all specified meters. This method is used to send all pending battery status reports. |

#### URI parameters

None

#### Body parameters

| Name | JSON data type | Internal data type | Null |
|---|---|---|---|
| meterBatteryStatuses | array (of objects) | IEnumerable<MeterBatteryStatuses> | No |
| batteryStatuses | array (of objects) | IEnumerable<BatteryStatus> | No |
| meterId | string | Guid | No |
| millivolts | number | UInt16 | No |
| currentCapacity | number | UInt32 | No |
| totalCapacity | number | UInt32 | No |

#### JSON schema example

```
{
  "meterBatteryStatuses": [
    {
      "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
      "batteryStatuses": [
        {
          "millivolts": 3700,
          "currentCapacity": 10500,
          "totalCapacity": 16280
        }
      ]
    }
  ]
}
```

#### Responses

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, battery statuses for all specified meters were stored in the database. | `{ "message": "Battery statuses for all meters were stored" }` |
| 400 | Validation failed, some property is missing or its value is invalid | `[ {` |

| | | |
|---|---|---|
| | (in this example the “batteryStatuses” property is missing) | "PropertyName": "batteryStatuses", "Error": "The batteryStatuses field is required." } ] |
| 404 | At least one of the meters was not found with the specified GUID. | { "message": "Meter not found", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] } |
| 500 | An error occurred storing battery statuses in the database (e.g. meter was deleted while inserting battery statuses) | { "message": "Error storing battery statuses in the database", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] } |
| 400 | At least one of the meters is not associated with the gateway that sent the request | { "message": "Some meters are not associated with the gateway that sent the request", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] } |

**Controller “Signal mapping”**

**1 - Insert point**

| | |
|---|---|
| Method | POST |
| Path | /api/SignalMapping/SetPoint |
| Description | Store a signal mapping point on map with signal transmission and reception conditions. |

**URI parameters**

None

**Body parameters**

| Name | JSON data type | Internal data type |
|---|---|---|
| meterId | string | Guid |
| latitude | number | Decimal |
| longitude | number | Decimal |
| altitude | number | Int16 |
| dateTime | string | DateTimeOffset |
| mtRSSI | number | Int16 |

| | | |
|---|---|---|
| mtSNR | number | Decimal |
| mtSF | number | Byte |
| mtBw | number | UInt16 |
| mtFreq | number | UInt32 |
| mtCh | number | Byte |
| mtCodeRateN | number | Byte |
| mtCodeRateD | number | Byte |
| gwRSSI | number | Int16 |
| gwSNR | number | Decimal |
| gwSF | number | Byte |
| gwBw | number | UInt16 |
| gwFreq | number | UInt32 |
| gwCh | number | Byte |
| gwCodeRateN | number | Byte |
| gwCodeRateD | number | Byte |

**JSON schema example**


```
{
  "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
  "latitude": 40.360900,
  "longitude": -7.861200,
  "altitude": 506,
  "dateTime": "2017-04-03T13:30:00.000Z",
  "mtRSSI": -51,
  "mtSNR": 9.2,
  "mtSF": 10,
  "mtBw": 125,
  "mtFreq": 902.9,
  "mtCh": 3,
  "mtCodeRateN": 4,
  "mtCodeRateD": 5,
  "gwRSSI": -79,
  "gwSNR": 9.2,
  "gwSF": 10,
  "gwBw": 125,
  "gwFreq": 902.9,
  "gwCh": 3,
  "gwCodeRateN": 4,
  "gwCodeRateD": 5
}
```

**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, a point in map containing signal information was stored in the database. | <pre>{<br>  "message": "Point stored"<br>}</pre> |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the “latitude” property is out of range) | <pre>[<br>  {<br>  "PropertyName": "latitude",<br>  "Error": "Latitude is out of range (-90 to 90)"<br>  }<br>]</pre> |
| 404 | No meter found with the specified GUID. | <pre>{<br>  "message": "Meter not found"<br>}</pre> |
| 500 | Signal mapping points are stored in the meter active signal mapping session, if there is none the signal mapping point cannot be stored | <pre>{<br>  "message": "This meter has no active signal mapping session"<br>}</pre> |
| 500 | An error occurred storing the signal mapping point in the database (e.g. web service could not connect to the database) | <pre>{<br>  "message": "Error storing signal mapping point in the database"<br>}</pre> |
| 400 | Meter is not associated with the gateway that sent the request | <pre>{<br>  "message": "Meter is not associated with the gateway that sent the request"<br>}</pre> |

**2 - Insert point for multiple meters**

| | |
|---|---|
| Method | POST |
| Path | /api/SignalMapping/SetPointMultipleMeters |
| Description | Stores signal mapping points on map with signal transmission and reception conditions for all specified meters. This method is used to send all pending signal mapping points. |

**URI parameters**

None

**Body parameters**

| Name | JSON data type | Internal data type |
|---|---|---|
| meterPoints | array (of objects) | IEnumerable<MetersSignalMappingPoints> |
| points | array (of objects) | IEnumerable<SignalMappingPoint> |
| meterId | string | Guid |
| latitude | number | Decimal |
| longitude | number | Decimal |
| altitude | number | Int16 |
| dateTime | string | DateTimeOffset |
| mtRSSI | number | Int16 |
| mtSNR | number | Decimal |
| mtSF | number | Byte |
| mtBw | number | UInt16 |
| mtFreq | number | UInt32 |
| mtCh | number | Byte |
| mtCodeRateN | number | Byte |
| mtCodeRateD | number | Byte |
| gwRSSI | number | Int16 |
| gwSNR | number | Decimal |
| gwSF | number | Byte |
| gwBw | number | UInt16 |
| gwFreq | number | UInt32 |
| gwCh | number | Byte |
| gwCodeRateN | number | Byte |
| gwCodeRateD | number | Byte |

**JSON schema example**


```
{
  "meterPoints": [
    {
      "meterId": "8b5b935e162944e9930bcb96c2b6e8ad",
      "points": [
        {
          "latitude": -40.7,
          "longitude": 8.6,
          "altitude": 200,
          "dateTime": "2017-03-30T12:30:00.000Z",
          "mtRSSI": 87,
          "mtSNR": 9.2,
          "mtSF": 3,
          "mtBw": 125,
          "mtFreq": 902.9,
          "mtCh": 3,
          "mtCodeRateN": 4,
          "mtCodeRateD": 5,
          "gwRSSI": 902.9,
          "gwSNR": 902.9,
          "gwSF": 902.9,
          "gwBw": 902.9,
          "gwFreq": 902.9,
          "gwCh": 902.9,
          "gwCodeRateN": 902.9,
          "gwCodeRateD": 902.9
        }
      ]
    }
  ]
}
```


**Responses**

| Code | Description | JSON schema |
|---|---|---|
| 201 | Successful operation, points in map containing signal information for all specified meters were inserted in the database. | { "message": "Points inserted for all meters" } |
| 400 | Validation failed, some property is missing or its value is invalid (in this example the “latitude” property is out of range) | [ { "PropertyName": "latitude", "Error": "Latitude is out of range (-90 to 90)" } ] |
| 404 | At least one of the meters was not found with the specified GUID. | { "message": "Meter not found", "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"] } |
| 500 | At least one meter doesn’t have an active signal mapping session | { "message": "This meter has no active signal mapping session", |

| | | |
|---|---|---|
| | | "errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |
| 500 | An error occurred storing signal mapping points in the database (e.g. web service could not connect to the database) | {<br>"message": "Error storing signal mapping points in the database",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |
| 400 | At least one of the meters is not associated with the gateway that sent the request | {<br>"message": "Some meters are not associated with the gateway that sent the request",<br>"errorMeterIds": ["8b5b935e162944e9930bcb96c2b6e8ad"]<br>} |