Universidade de Aveiro

Departamento de Engenharia Mecânica

Ano 2021

Daniel Jesus Camarneiro

Industrial Internet Of Things and connectivity, in Bosch

Internet das Coisas na Indústria, e a conectividade na Bosch

Relatório de Projeto apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob a orientação científica de José Paulo Oliveira Santos, Professor Auxiliar do Departamento de Engenharia Mecânica da Universidade de Aveiro, de Paulo Bacelar Reis Pedreiras, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro e de Rui Luís Andrade Aguiar, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

O presente estudo foi também realizado ao abrigo do Projeto "*Augmented Humanity*" [POCI-01-0247-FEDER-046103], cofinanciado pelo Programa Operacional Competitividade e Internacionalização e pelo Programa Operacional Regional de Lisboa do PORTUGAL 2020, através do Fundo Europeu de Desenvolvimento Regional.

o júri

Presidente	Prof. Doutor Jorge Augusto Fernandes Ferreira Professor Associado da Universidade de Aveiro
Vogais	Prof. Doutor Pedro Alexandre de Sousa Gonçalves Professor Adjunto da Universidade de Aveiro (Arguente)
	Prof. Doutor José Paulo Oliveira Santos Professor Auxiliar da Universidade de Aveiro (Orientador)

agradecimentos

Gostaria de agradecer à minha família e amigos, que me suportaram nesta fase da minha vida, e também ao professor José Paulo Santos por me ter orientado na elaboração deste projeto.

palavras-chave

Gateway, *Middleware*, Comunicação, Sensores, IoT, Energia, Serviços, *Backend*, Bases de Dados, Plataformas *IoT*, Visualização, Controlo, Automação, Informática Industrial.

resumo

A falta de conectividade entre os equipamentos fabris, e os diferentes departamentos da empresa, compromete a gestão, monitorização e controlo do seu funcionamento.

Neste projeto foi desenvolvido um protótipo que permite a aquisição de informação relativamente aos consumos de energia, obtidos pelos analisadores de energia, a supervisão dos processos de fabrico, controlados por autómatos programáveis instalados na produção, e o envio de mensagens de comando, remotamente, para os dispositivos.

Foram estudadas as plataformas *Bosch IoT Suite* e *SCoT*, desenvolvidos no *Eclipse IoT*, e os seus respetivos *brokers MQTT* (*Eclipse Hono* e *Bosch IoT Hub*) e as interfaces gráficas (*dashboard) Bosch IoT Insights* (dos serviços da *Bosch IoT Suite*), *Grafana* e o *dashboard* do *NodeRed*. As Bases de Dados estudadas foram MySQL (*SQL*), *MongoDB* (*NoSQL)* e *InfluxDB* (*timeseries*).

Também foram estudados o *ESP32* e *Raspberry Pi* como dispositivos de *gateway* com a plataforma *IoT*. Os *softwares* utilizados no desenvolvimento foram o *Node.js*, para realizar a ligação entre a plataforma *IoT* e a Base de Dados, *Express.js* no desenvolvimento dos serviços REST API *backend*, o *NodeRed* para estabelecer a comunicação *Rs485 MODBUS RTU* entre o analisador de energia *Janitza* (e *SDM120*) e o *Raspberry*, assim como a comunicação com o módulo OPC-UA, usado para comunicar com os PLCs S7-1200, e o TIA Portal para a programação dos PLCs.

Assim, neste projeto foram propostos *gateways*, que oferecem uma camada de *middleware* entre os equipamentos e as plataformas *IoT* (Bases de Dados). Também foram propostos serviços REST que permitem extrair informação (potências, correntes, frequências, temperaturas, pressões, entre outras) da Base de Dados e enviar mensagens de comando aos dispositivos (ativar/desativar alarmes e interrupções, pedidos de registos de memória, entre outros) através de uma aplicação remota.

keywords

Gateway, *Middleware,* Communication, Sensors, C*loud*, IoT, Energy**,** Services, *Backend*, Databases, *IoT* Platforms, Visualization, Control, Automation, Industrial Computing.

abstract

The lack of connectivity between the manufacturing equipment and the different company's departments compromises management, monitoring and control of its operation.

In this project there was developed a prototype that allows acquisition of information regarding energy consumption, obtained by energy analyzers, supervision of manufacturing processes, controlled by programmable automatons installed in the production, and the sending of command messages, remotely, for devices.

Were studied the platforms Bosch IoT Suite and SCoT, developed in Eclipse IoT, their respective MQTT correctors (Eclipse Hono and Bosch IoT Hub) and the dashboards Bosch IoT Insights (from Bosch IoT Suite services), Grafana and NodeRed dashboard. The databases studied were MySQL (SQL), MongoDB (NoSQL) and InfluxDB (SQL timeseries).

There were also studies on ESP32 and Raspberry Pi as gateway devices with an IoT platform. The software used in the development were Node.js, to make a connection between the IoT platform and the database, Express.js in the development of the REST API backend services, NodeRed to establish an Rs485 MODBUS RTU communication between the energy analyzer Janitza (and SDM120) and the Raspberry, as well as the communication with the OPC-UA module, used to communicate with the S7-1200 PLCs, and the TIA Portal for programming the PLCs.

So, in this project were proposed gateways, which are a middleware layer between the equipment and as IoT platforms (Databases). There were also proposed REST services that allow extracting information (powers, currents, frequencies, notes, pressures, among others) from the Database and sending command messages to devices (activate / disable alarms and interruptions, requests for memory registration, among others) via a remote application.

Índice

Lista de Tabelas

Lista de Figuras

Acrónimos

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CNC	Computer Numeric Control
CoAP	Constrained Application Protocol
CPU	Central Process Unit
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
MÊS	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
PLC	Programmable Logic Controller
SCoT	Smart Cloud of Things
SQL	Structured Query Language
WS	WebSocket

Capítulo 1

Introdução

É incrível como atualmente estamos a presenciar uma evolução tecnológica como nunca antes vista. O aumento da capacidade de processamento, da densidade de armazenamento e da velocidade de transferências permitiram desenvolver um conjunto de aplicações inimagináveis há vinte anos atrás. *Smartphones*, *Home Assistants,* 4G/5G, entre outros, são alguns exemplos frutos da evolução inflacionada pela procura da otimização e o desenvolvimento de novos produtos e serviços.

A Indústria foi uma das maiores influenciadoras desta evolução. O desejo do aumento da produção e a diminuição dos custos promoveu o investimento em diversas ferramentas, as quais provocaram diversas revoluções industriais ao longo dos anos. Atualmente estamos a presenciar uma nova revolução, a **Indústria 4.0**, baseada na "digitalização" das estruturas fabris. Esta revolução permitirá diminuir a intervenção humana na gestão de fábricas, promovida pela conetividade de todos os dispositivos físicos e as suas funcionalidades, denominada por ***Internet of Things (IoT)***. Esta conetividade permitirá a implementação de serviços autónomos, que analisam os resultados e atribuem soluções que visam a otimização dos processos fabris. A customização do produto final de acordo com as especificações de cada indivíduo é outro objetivo da Indústria 4.0. A sua implementação valorizará estas infraestruturas comparativamente àquelas que não evoluíram. Contudo, estas especificações ainda são um projeto que necessita de muito desenvolvimento e investigação para alcançar os resultados desejados nesta independência produtiva.

Na perspetiva da aquisição e controlo dos equipamentos na instalação fabril, foi desenvolvido um protótipo que permite comunica com estes equipamentos, nomeadamente os analisadores de energia e os autómatos programáveis instalados na produção, assim como o armazenamento e visualização da informação adquirida. O envio de informação entre os *gateways* e a *cloud* foi realizada com recurso a uma plataforma *IoT* (foram utilizados o *Bosch IoT Suite* durante a fase de testes e o *SCoT* no projeto final), com recurso a um *broker MQTT*, onde esta depois será armazenada numa Base de Dados (*InfluxDB*) e visualizável num *dashboard* (*Grafana*).

Os equipamentos estudados para realizar a função de *gateway* foram o ESP32 e o Raspberry Pi. A programação dos serviços realizados no servidor foram em *Node.js* (ligação entre a plataforma *IoT* e a Base de Dados) e *Express.js* (no desenvolvimento da REST API), enquanto o *NodeRed* foi utilizado para estabelecer a comunicação *Rs485 MODBUS RTU*

entre o analisador de energia *Janitza* (e *SDM120*) e o *Raspberry*. O *NodeRed* também foi utilizado para estabelecer a comunicação com o módulo OPC-UA, usado para comunicar com os PLCs S7-1200, e o TIA Portal para a programação, em *Ladder*, dos PLCs.

Os serviços REST propostos que permitem facilitar a interação entre o utilizador e o protótipo desenvolvido, nomeadamente na aquisição de informação (potências, correntes, frequências, temperaturas, pressões, entre outras) existente na Base de Dados, e no envio de mensagens de comando aos dispositivos (ativar/desativar alarmes e interrupções, pedidos de registos de memória, entre outros), através desta aplicação remota.

1.1. Objetivos

Os principais objetivos propostos para a realização deste projeto foram os seguintes:

- Disponibilizar um conjunto de serviços WEB (REST/SOAP), reutilizáveis e genéricos, para um grande lote de equipamentos fabris que permitam monitorizar o seu funcionamento e os seus consumos de energia.
- Desenvolvimento dos módulos de interface com os sensores e atuadores locais;
- Desenvolvimento do código/*firmware* para que estes módulos possam ser acedidos a partir da internet.

Considerando a análise do estado da arte e as reuniões com as entidades responsáveis, as seguintes etapas demonstram, mais detalhadamente, os processos e características do protótipo desenvolvido:

- Definição e desenvolvimento de um módulo base, microcontrolado, com a capacidade de processamento, programável, configurável e de baixo custo;
- Capacidade de comunicar por Rs485 MODBUS RTU, protocolo utilizado pelos analisadores de energia instalados em chão de fábrica;
- Capacidade de comunicar por Ethernet OPC-UA, protocolo utilizado pelos PLCs e módulos OPC.
- Realização da publicação das medições numa plataforma *IoT*;
- Adição da informação publicada numa base de dados;
- Utilização de uma plataforma fácil e interativa para a visualização dos registos efetuados;
- Desenvolvimento de uma REST API para interagir com os módulos microcontrolados e a base de dados utilizada.

No fim deste projeto é pretendido desenvolver um protótipo, que integre uma plataforma *IoT,* onde seja permitido a monitorização e interação entre um número elevado

de dispositivos diferentes. Este desenvolvimento será focalizado na monitorização dos analisadores de energia, componente importante para a empresa.

1.2. Organização

Este documento está organizado em sete capítulos:

- **Capítulo 1:** realiza-se a apresentação do problema e o seu ambiente integrante, assim como os objetivos propostos desenvolver e solucionar;
- **Capítulo 2:** apresentação da empresa onde fora desenvolvido o trabalho;
- **Capítulo 3:** análise do estado da arte, onde é apresentado o conjunto de tecnologias e metodologias exploradas durante o desenvolvimento do projeto com o objetivo de apresentar uma solução eficaz;
- **Capítulo 4:** contextualização dos componentes a implementar para viabilizar a solução desejada;
- **Capítulo 5:** implementação dos componentes de *hardware* e software utilizados e o seu respetivo desenvolvimento para solucionar os objetivos indicados anteriormente;
- **Capítulo 6:** análise da performance da arquitetura proposta perante os objetivos propostos;
- **Capítulo 7:** conclusões relativamente ao protótipo desenvolvido e indicação de alguns trabalhos futuros.

Capítulo 2

A Empresa

2.1. Grupo Bosch

O Grupo Bosch atualmente é líder no fornecimento de serviços e tecnologia, nomeadamente em soluções inovadoras para cidades e casas inteligentes, mobilidade e indústria conectada. O Grupo opera em quatro áreas de negócio: Bens de Consumo, Tecnologia Industrial, Soluções de Mobilidade e Tecnologia de Energia e Edifícios, e emprega cerca de 395 000 colaboradores ao redor do mundo.

Atualmente o Grupo Bosch possui quatro unidades implementadas em Portugal, a Bosch Car Multimedia, em Braga, focalizada no desenvolvimento e produção de soluções de multimédia e sensores automóveis, a Bosch Security System, em Ovar, que produzem soluções inovadoras para sistemas de segurança e comunicação, alarmes de incêndios, displays eletrónicos e outros produtos para unidades de negócio do Grupo, A Bosch Termotecnologia, em Aveiro, onde reside o desenvolvimento de soluções de água quente residencial, e a Robert Bosch S.A., em Lisboa, sede comercial do Grupo no mercado português[2].

2.2. Bosch Termotecnologia S.A.

A Bosch Termotecnologia S.A., inaugurada em 1988 com a aquisição da Vulcano Termodomésticos, é a unidade responsável pela unidade de negócios de água quente residencial do Grupo Bosch. Atualmente a empresa fornece soluções de água quente para todo o mundo, através de vários equipamentos distintos, nomeadamente caldeiras, bombas de calor e esquentadores a gás e elétricos.

Atualmente a Bosch é um dos principais dinamizadores da economia na região de Aveiro, onde são empregues cerca de 1300 colaboradores e possui uma rede de parceiros que trabalham em harmonia pelo seu sucesso [10].

Figura 2.1: Bosch Termotecnologia

Capítulo 3

Estado da Arte

Neste capítulo será realizada uma revisão e análise do estado da arte atual. Esta análise permitirá aprofundar o conhecimento dos elementos constituintes de projetos desta natureza, permitindo assim formular uma solução concisa perante os objetivos pretendidos, através da análise de trabalhos previamente desenvolvidos e implementados por outros autores.

No apêndice [A] encontram-se alguns conceitos utilizados e explorados durante a realização deste projeto, contudo os conceitos apresentados são comuns em aplicações *IoT*. Numa perspetiva de focalizar o leitor no trabalho desenvolvido, retiraram-se estes conceitos do corpo principal, permitindo a sua consulta rápida se existir a necessidade.

3.1. Cloud Servers

Um servidor *cloud* é uma infraestrutura composta por múltiplos servidores físicos, subdivididos em vários servidores virtuais (*clusters*). Estes são utilizados para processar tarefas que necessitem de um elevado nível de desempenho e/ou de um elevado volume de armazenamento de informação. Estes sistemas foram criados com o objetivo de facilitar a implementação de projetos em servidores locais, assim como a diminuição da sua manutenção. A maioria dos serviços permite ao utilizador customizar a capacidade (de processamento e de armazenamento) necessárias para a aplicação final [14].

3.2. Comunicação entre sensores e a *cloud*

Independentemente do meio de comunicação utilizado, é fundamental existir um dispositivo de *middleware* que realize a ponte entre os sinais provenientes dos sensores [A.1] e o servidor. Assim, este dispositivo de *middleware*, designado por *gateway,* necessita de possuir vários elementos que o viabilizem em aplicações comuns:

- ser capaz de interpretar uma variedade de protocolos de comunicação, o que proporciona uma maior flexibilidade na seleção dos sensores e nas metodologias de comunicação compatíveis com o servidor;

- permitir realizar trocas de mensagens num curto intervalo de tempo, tanto na aquisição de dados, como no envio de comandos para o equipamento em monitorização;
- também deve possuir alguma segurança uma vez que, ao estar conectado à internet, constitui uma possível vítima de *hack*. Também é necessário constatar que o *gateway* encontra-se normalmente conectado a vários dispositivos, existindo a necessidade de estabelecer uma linha de defesa contra ameaças digitais externas, uma vez que este representa uma possível porta de entrada indesejada;
- apresentar a capacidade de permanecer funcional perante o ambiente inserido, nomeadamente ao ruído (a nível mecânico e magnético), humidade e temperatura ambiental;
- outro fator que poderá despertar importância é o seu consumo (potência elétrica), pois existem situações onde não existe uma fonte de alimentação contínua, tornando-se necessário uma análise do balanço entre o consulo (elétrico), a capacidade de bateria e computacional [27].

3.3. REST API

Uma API é um conjunto de definições, funções e procedimentos utilizados no desenvolvimento e na integração de aplicações. A sua utilização permite facilitar a interação do utilizador (representado por outro programa ou um ser humano) com o sistema através de pedidos exemplificados na documentação fornecida pelo desenvolvedor, uma vez que a sua utilização cria uma camada abstrata entre o utilizador e o funcionamento interno do sistema desenvolvido [44].

A utilização de REST define a aparência da API através de um conjunto de regras pré-definidas. Os pedidos realizados à REST API denominam-se por *request*, realizados por um URL, e a resposta por *response*. Os *request* são compostos por quatro parâmetros:

- **Endereço (*endpoint)*:** representa o URL utilizado. Este URL agrega o domínio e o caminho do pedido realizado, assim como as variáveis e os parâmetros desejados. No exemplo *http://example.com/api/v2.0/:object?color=blue&size=big*:
  - *'http://example.com'* representa o domínio;
  - *'/api/v2.0/:object'* representa o caminho do pedido;
  - *'/:object'* representa uma variável;
  - *'?color=blue&size=big'* representam dois parâmetros.
- **Método (*method*):** representa o tipo de pedido realizado ao servidor:
  - *GET*: operação destinada à leitura de recursos existente no servidor;
  - *POST:* operação destinada à criação de novos recursos no servidor;

- *PUT & PATCH*: operações destinadas à atualização de recursos no servidor;
- *DELETE*: operação destinada à eliminação de recursos no servidor.

- **Cabeçalhos (*Headers*):** são utilizados para enviar informação adicional, por exemplo credenciais de autenticação ou o tipo de informação contido no corpo da mensagem.
- **Corpo (*Body*):** contêm a informação a enviar ao servidor. A utilização do método *GET* não requere este parâmetro [25].

3.4. Plataformas *IoT*

A evolução das soluções de gestão dos dispositivos *IoT* é uma das componentes que engloba a crescente utilização destas tecnologias, onde desenvolvimento destas plataformas propõe melhorar a sua eficiência de desempenho e utilização no nosso quotidiano.

3.4.1 Bosch IoT Suite

O *Bosch IoT Suite* é a plataforma *IoT* desenvolvida pela *Bosch* que proporciona uma variedade de serviços *cloud* utilizados em projetos de *IoT*. Esta plataforma possibilita a comunicação direta dos dispositivos através de mensagens de telemetria, ou indiretamente através um *gateway*. Também permite criar representações virtuais (conhecidas como ***digital twins***) dos dispositivos reais na *cloud,* que permitem armazenar e atualizar as propriedades dos dispositivos reais. O *Bosch IoT Suite* também oferece um serviço de recolha, processamento, armazenamento e visualização dos dados recolhidos para a sua utilização e análises futuras, o que complementa a sua implementação numa arquitetura autónoma [12] [11].

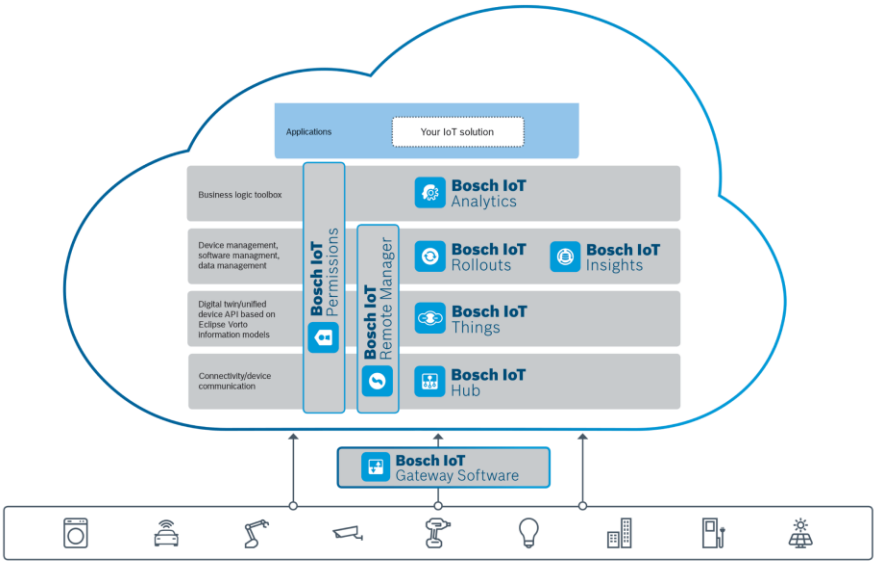

Figura 3.1: Arquitetura da plataforma Bosch IoT Suite

3.4.2 SCoT

O *Smart Cloud of Things* é uma plataforma *IoT* desenvolvida pelo departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro que permite estabelecer uma ligação entre o servidor e os dispositivos em chão de fábrica (sensor solo ou um *gateway*) Estes dispositivos possuem o seu respetivo ***digital twin*** identificado no serviço, onde é possível realizar a sua caracterização. Esta plataforma também permite guardar e visualizar os dados obtidos [21].

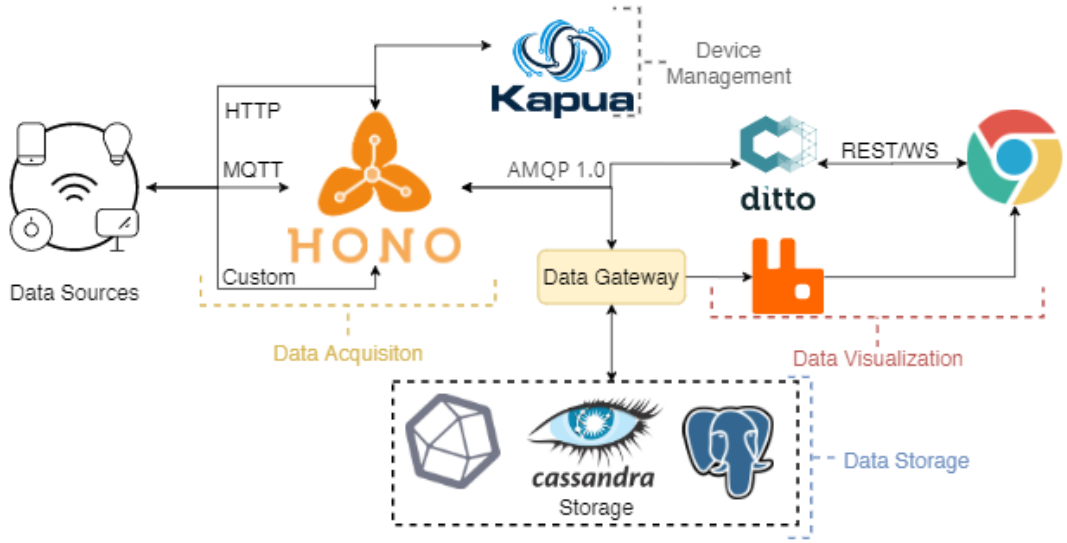

Figura 3.2: Arquitetura da plataforma SCoT

O funcionamento das duas plataformas exploradas é semelhante, uma vez que a base da sua implementação é igual. O seu desenvolvimento tem por base os serviços alguns

serviços *IoT* desenvolvidos em *Eclipse IDE* (ambiente de desenvolvimento integrado baseado em java).

3.4.3 Eclipse Hono

O *Eclipse Hono* é um conjunto de serviços que permite conectar um número elevado de dispositivos através de diferentes protocolos (MQTT, AMQP, HTTP e CoAP) e adapta estas mensagens para o protocolo AMQP. A sua utilização abstrai a comunicação realizada as comunicações utlizadas pelos dispositivos. O *Eclipse Hono* também apresenta uma API definida para realizar o envio de mensagens, dividida em mensagens de telemetria (*Telemetry*), eventos (*Events*), e comando & controlo *(Command & Control)* [18].

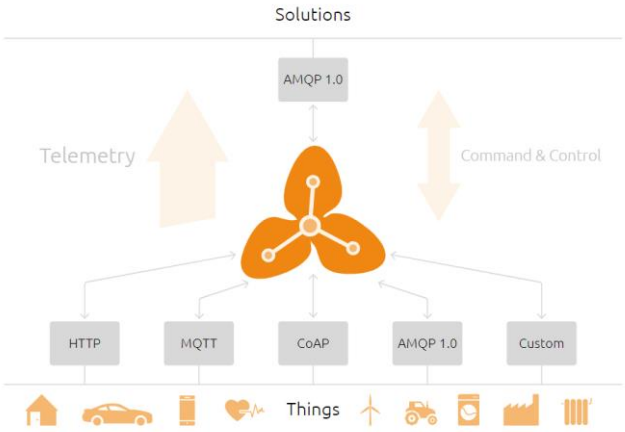

Figura 3.3: Arquitetura do Eclipse Hono

3.4.4 Eclipse Ditto

O *Eclipse Ditto* é o serviço responsável por implementar o conceito de ***digital twins*** na plataforma. Esta solução permite simplificar o desenvolvimento de soluções *IoT*, uma vez que os desenvolvedores não necessitam de saber como a *"thing"* física está conectada. A utilização do *Ditto* permite utilizar a *"thing"* como um *web servisse* comum através do seu *digital twin* [17].

3.4.5 Eclipse Kapua

O *Eclipse Kapua* permite gerir o funcionamento da plataforma *IoT*. Este componente é o responsável pela gestão dos dispositivos, assim como o registo de utilizadores e a sua autenticação [19].

3.5. Base de Dados

Uma base de dados é um conjunto de informação guardada num banco de dados. Para realizar a gestão desses arquivos utilizam-se programas denominados por Sistemas de Gestão de Bases de Dados. Contudo, com o avanço da tecnologia apareceram novas necessidades, que promoveram a criação Sistema de Gestão otimizados próprios. Os Sistemas de Gestão de Base de Dados (denominados regularmente apenas por Base de Dados) podem ser divididos nas seguintes categorias:

- **Bases de Dados Relacionais:** bases de dados convencionais, onde os dados são dispostos em tabelas, com colunas e linhas. A manipulação da informação existente nas bases de dados é normalmente realizada com linguagem ***SQL*** (*Structured Query Language).*
- **Bases de Dados Não Relacionais:** estas bases de dados surgiram como alternativa às bases de dados relacionais em aplicações web. Estas bases de dados não funcionam de acordo com a linguagem *SQL* (por isso denominadas por *Not Only SQL,* ***NoSQL***), o que permite guardar e manipular dados não estruturados ou semiestruturados, nomeadamente documentos ou JSON. As bases de dados *NoSQL* podem ser divididas nas seguintes categorias:
    - ***Key-Value:*** Estas bases de dados indexam a informação por chaves.
    - ***Document:*** Estas bases de dados organizam coleções de documentos, onde cada do possui uma chave única.
    - ***Wide-Column***: Estas bases de dados guardam a informação em tabelas que podem ser partilhadas entre múltiplos nós [22] [36].

A aplicação de uma base de dados numa operação de aquisição e manipulação de dados relativos ao histórico dos equipamentos não é trivial. A gestão do fluxo de informação e o espaço dedicado ao seu armazenamento necessitaria de uma boa otimização de desempenho, mesmo recorrendo a bases de dados *NoSQL*. Assim surgiram as **bases de dados temporais** (*time series data bases)*. Estas bases de dados foram desenvolvidas propositadamente para a evolução das necessidades da indústria *IoT*, possibilitando gerir elevadas quantidades de informação rapidamente, assim como agregar a informação após algum tempo, diminuindo a capacidade de armazenamento necessária [42].

3.6. Trabalhos realizados por outros autores

O crescente desenvolvimento e o aumento da procura de soluções desta natureza proporcionaram um ambiente fértil a serviços *IoT* nos últimos anos, promovendo uma grande variedade de soluções para problemas em comum. Assim, o objetivo principal deste trabalho não é encontrar uma solução para o problema, mas sim selecionar e implementar

uma eficaz. Neste subcapítulo ser-se-á realizada a análise dos trabalhos utilizados como referência no desenvolvimento deste projeto.

3.6.1 Cyber-physical Machine Tool – The Era of Machine Tool 4.0

Liu e Xu, no artigo [26], demonstram um exemplo dos processos realizados para adaptar uma máquina-ferramenta convencional num equipamento apto a ser utilizado na Indústria 4.0. A proposta final teve em consideração um conjunto de requisitos necessários para o seu funcionamento, nomeadamente as interações com o módulo CNC e o seu respetivo PLC, a aquisição dos dados provenientes de todos os equipamentos instalados na máquina, a necessidade da identificação de um ***digital twin*** nessa plataforma e as interações humanas com o sistema. A solução proposta neste exemplo está dividida em três níveis:

- *Physical Level*: que engloba todos dos equipamentos físicos existentes em chão de fábrica, desde maquinaria aos seus sensores;
- *Cyber space*: onde existem representações abstratas dos equipamentos físicos (*Digital Twin).* Nesta camada também existem algoritmos que analisam a informação recolhida, permitindo assim monitorizar o estado atual dos equipamentos físicos, assim como guardam a informação destes equipamentos numa base de dados;
- *Service cloud*: onde é possível adicionar uma variedade de serviços diferentes (planeamento de produção, manutenção de ferramentas, entre outros) tendo como base os dados existentes no *Cyber Space*.

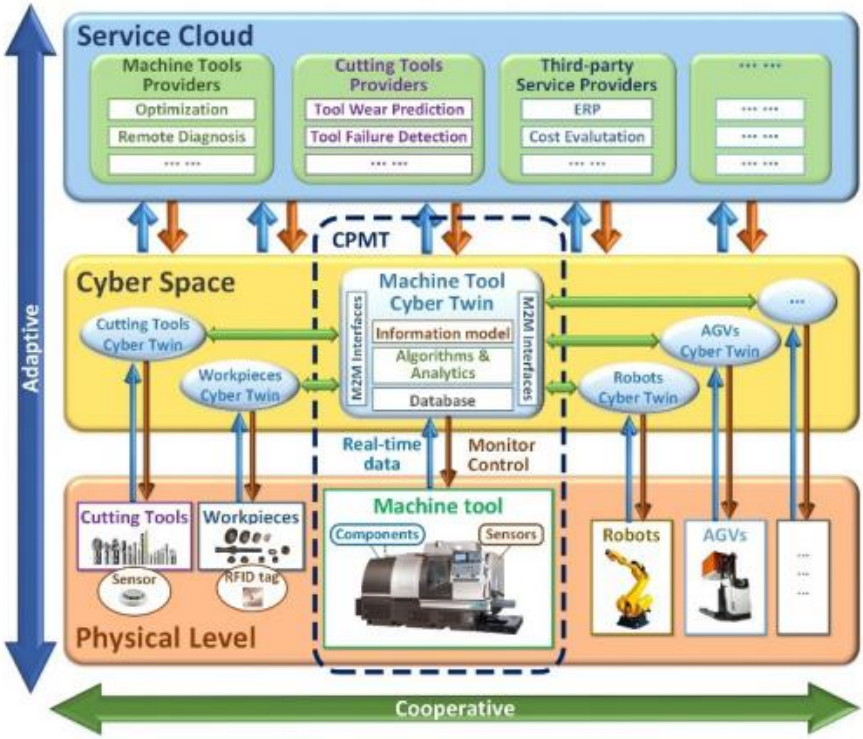

Figura 3.4- Arquitetura proposta por Liu & Xu

A análise deste artigo permitiu compreender um exemplo de estrutura aplicável numa plataforma *IoT* e os respetivos fluxos de informação entre os diferentes níveis. Também introduziu o conceito de ***digital twins***, um dos conceitos base para o funcionamento de uma plataforma *IoT*

Este artigo também é referido no livro *"Machine Tool: From the Digital Twin to the Cyber-Physical Systems"[7]*, onde é exemplificado a arquitetura tipicamente utilizada em problemas desta natureza. A atribuição de três camadas, divididas em camada física, rede e analítica, permanece como uma solução favorável no desenvolvimento. A sincronização dos equipamentos físicos com os abstratos criados na rede, com recurso a *digital twins*, é uma condição fundamental na organização da estrutura fabril. Outra vantagem da utilização de ***digital twins*** é a possibilidade de comparar dados reais com simulados, permitindo melhorar a eficiência da produção.

3.6.2 Architecture and Implementation of a Scalable Sensor Data Storage and Analysis System Using Cloud Computing and Big Data Technologies

Aydin et al, no artigo [8], exploram a implementação de uma arquitetura constituída por várias tecnologias *open source*, eficaz na gestão do fluxo de informação proveniente da crescente utilização de diversos tipos de sensores, por exemplo a aquisição das coordenadas geográficas por GPS. A sua análise permitiu introduzir um conjunto de novos conceitos:

- ***Big Data:*** baseia-se na gestão de grandes quantidades de informação, difíceis de armazenar, gerir e analisar através do uso de programas e bases de dados convencionais. Neste artigo utilizam um conjunto de teorias, algoritmos e *frameworks* com o objetivo de melhorar o seu desempenho.
- ***Cloud Computing:*** utilização de *clusters* (conjunto de computadores que funcionam em conjunto, como um equipamento único) na obtenção da eficiência necessária perante as necessidades existentes. Neste artigo foi utilizada a *framework Hadoop* [45], desenvolvida pela *Apache* destinada a facilitar a implementação de *clusters*.
- ***NoSQL*:** o aumento do número de sensores afeta a eficiência na utilização de uma base de dados *SQL* convencional. Estas bases de dados permitem realizar distribuir a informação entre vários servidores e adicionar novos atributos dinamicamente.

3.6.3 Forecasting Applicances Failures: A Machine-Learning Approach to Predictive Maintenance

Fernandes et al, no artigo [21], apresentam um protótipo desenvolvido com o objetivo de explorar soluções de manutenção preditiva através de algoritmos de *machine-learning*, contudo foi apresentada a utilização de uma plataforma *IoT* destinada a aplicações de telecomunicações, o ***SCoT***. Esta plataforma proporciona um ambiente otimizado e flexível em projetos de *IoT* que requeiram o uso de vários dispositivos. Assim, esta plataforma permite criar e gerir um conjunto de serviços e aplicações (***digital twins***, bases de dados, gestão de dispositivos, etc.) que otimizam e facilitam o desenvolvimento de novas soluções *IoT*.

3.6.4 Gestão Remota de Dispositivos

Bastos, na dissertação [9], apresenta o desenvolvimento de uma ferramenta responsável pela gestão de dispositivos, que permite realizar a aquisição de informação do seu funcionamento, assim como atualizar o seu *software/firmware* a partir das configurações desta ferramenta. O seu desenvolvimento teve como base outra plataforma *IoT*, o ***Bosch IoT Suite***. Esta plataforma foi desenvolvida pela *Bosch* com o objetivo de auxiliar a implementação das novas tecnologias *IoT* nas empresas, fornecendo um conjunto de serviços na instalação, monitorização e gestão de redes complexas, originadas pela crescente utilização de dispositivos *IoT*. Assim, esta plataforma fornece serviços de ***digital twins***, gestão de dispositivos, aquisição, análise e visualização da informação.

Capítulo 4

Arquitetura Conceptual

A análise das arquiteturas exploradas no capítulo anterior permitiram criar uma visão global dos componentes necessários existir perante a implementação de uma solução viável relativamente ao problema a resolver.

4.1. Estrutura da arquitetura

A arquitetura conceptual será dividida em três níveis:

- **Nível físico:** engloba todos os equipamentos físicos existentes em chão de fábrica, desde os destinados à manufatura (e os seus sensores), aos dispositivos *middleware* responsáveis por encaminhar as mensagens recebidas;
- **Nível digital:** onde se encontram as imagens virtuais dos diversos equipamentos em chão de fábrica (*digital twins*). Esta camada permite monitorizar o estado atual destes equipamentos, assim realizar um *backup* do histórico dos dados recebidos;
- **Nível de serviços:** nível dedicado à implementação de serviços customizáveis, permitindo assim flexibilizar a sua utilização perante os objetivos propostos.

4.2. Componentes da arquitetura

A contextualização dos níveis necessários para a implementação de uma arquitetura desta natureza não indica a constituição dos seus componentes necessários. Na figura seguinte é possível observar os componentes necessários, assim como a sua posição.

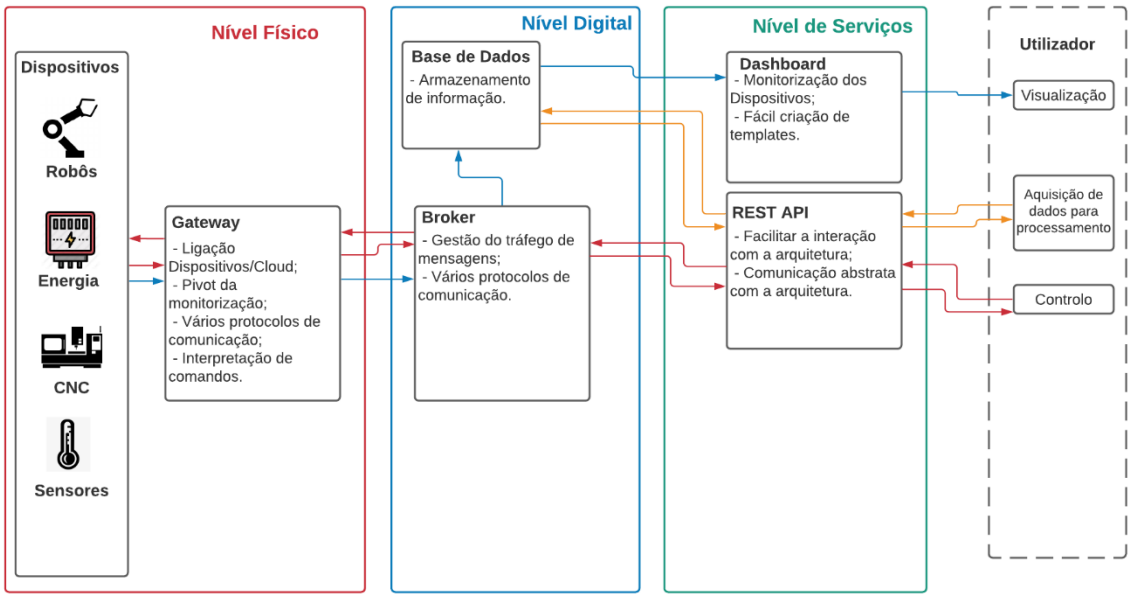

Figura 4.1: Diagrama da Arquitetura Conceptual

4.2.1 Dispositivos

Conjunto de todos os equipamentos dedicados à produção existentes em chão de fábrica. Estes equipamentos necessitam de possuir um conjunto de sensores (por defeito ou posteriormente adicionados) habilitados a comunicar o(s) seu(s) registos(s), preferencialmente por um protocolo de comunicação. Também existem equipamentos que poderão necessitar modificar o seu funcionamento dependendo de uma mensagem externa. Nestes casos, será necessária a existência de um equipamento computorizado que adapte a mensagem digital numa ação física. Um autómato é um exemplo de um equipamento que possa necessitar desta funcionalidade (p. ex. ligar/desligar um alarme).

4.2.2 Gateway

Neste contexto, um *gateway* é um componente de *middleware* capaz de receber, interpretar e modificar a informação recebida em protocolos diferentes permitindo, assim, realizar uma 'ponte' entre os vários equipamentos existentes e a plataforma *IoT*. Estes equipamentos também poderão possuir uma base de dados local para *backup*, caso o servidor falhe, assim como uma API básica para interações rápidas e um *dashboard* local dos dados recolhidos. A segurança informática destes equipamentos é outro fator a ponderar se existir a necessidade de conectar estes dispositivos a uma rede externa (p. ex. a internet), porque pode permitir a ocorrência de um ataque informático.

4.2.3 Broker

O *broker* é o programa responsável por realizar a organização da troca de mensagens entre os dispositivos e o servidor. A inclusão deste programa na arquitetura permite verificar, com confiança, que as mensagens enviadas chegam aos seus respetivos destinatários e, se este não as conseguir encaminhar, retém-nas até o dispositivo de destino confirmar a sua receção.

Outro fator importante que expande o conceito de *broker* comum é a atribuição de *digital twins* nos endereços utilizados para a identificação dos dispositivos de destino, porque a aplicação deste conceito permite otimizar a organização dos vários componentes, a fluidez no reencaminhamento das mensagens trocadas, e configurar o seu funcionamento. Esta expansão do conceito de *broker* está diretamente relacionada com a implementação de uma plataforma *IoT*.

4.2.4 Base de Dados

A função da base de dados é guardar e gerir a informação provenientes dos dispositivos. Por isso, a base de dados selecionada tem de possuir os requisitos mínimos necessários que permitam assegurar uma boa fiabilidade dos seus serviços a longo prazo. Contudo, a oferta de bases de dados é elevada, permitindo explorar os seus desempenhos perante os objetivos propostos (organização de inventário, histórico do funcionamento dos dispositivos, gestão de colaboradores, entre outras) e selecionar a(s) base(s) de dados que apresentaram a performance melhor.

4.2.5 Dashboard

A visualização do funcionamento dos dispositivos é um fator importante na gestão e manutenção de todos os equipamentos e fluxos de mercadorias em chão de fábrica, mas a organização deste sistema não é rígida, existindo assim a necessidade de modificar facilmente e rapidamente perante as necessidades atualmente exigidas. Assim, a utilização de um programa que permita facilmente interligar várias bases de dados (nomeadamente a(s) selecionada(s)) e que ofereça serviços intuitivos para a criação do(s) *template(s)* para essa visualização proporciona um ambiente favorável à longevidade da sua utilização.

Se o programa de *dashboard* utilizado não possuir uma API para comunicar com a fonte dos dados (Base de Dados ou broker), apresentando apenas a capacidade de desenvolver *templates* para a visualização de dados, existe a necessidade de desenvolver uma extensão para realizar esta aquisição de informação (diretamente com a fonte ou através da *REST API* desenvolvida).

4.2.6 REST API

Serviço dedicado à comunicação entre o sistema e o utilizador (humano ou máquina). A utilização deste serviço permite que o utilizador interaja com os vários componentes da arquitetura, através de simples pedidos HTTP (GET e POST), o que origina um ambiente agnóstico, por parte do utilizador. A existência deste agnosticismo favorece o utilizador, porque este não necessita de conhecer os processos e protocolos utilizado internamente entre os componentes da arquitetura.

O desenvolvimento de uma API não é um processo rígido, é uma evolução iterativa que visa ao progresso das funcionalidades e otimização dos serviços fornecidos, sendo assim favorável a discussão e avaliação da sua qualidade/quantidade por parte do utilizador.

4.2.7 Utilizador

Este componente da arquitetura representa as diferentes interações possíveis com o sistema proposto. Nesta fase apenas são requeridas três funcionalidades: visualizar o estado de funcionamento dos dispositivos, extração de valores para analises futuras (nomeadamente manutenção preditiva) e o envio de comando à distância, mas o sistema tem de ser flexível o suficiente para desenvolver outras funções além destas.

Capítulo 5

Implementação

Neste capítulo será abordada a implementação da arquitetura conceptual, assim como a seleção dos programas utilizados em cada componente a sequência de operações entre componentes, relativamente à informação das mensagens, e a exemplificação do código desenvolvido para o seu funcionamento. Na Figura 5.1 é possível visualizar a constituição final do protótipo desenvolvido. Esta figura tem como objetivo apresentar o desenvolvimento da arquitetura proposta, servindo de referência para a elaboração deste capítulo.

5.1. Dispositivos

Os parâmetros utilizados para a seleção dos dispositivos durante as fases de testes foram os seguintes:

- Aplicação prática perante o problema proposto, permitindo assim realizar a sua implementação e avaliação os resultados obtidos (análise de energia principalmente);
- Diversidade de protocolos de comunicação, o que gera um ambiente diversificado para a aplicação de outros dispositivos que utilizem os mesmos protocolos;
- Disponibilidade dos dispositivos existentes/ utilizados atualmente em chão de fábrica;
- Diversidade das suas funcionalidades, numa perspetiva de variar a interatividade entre as várias grandezas dos registos efetuados.

Assim, os dispositivos utilizados foram os seguintes:

- **Janitza UMG 103:** analisador de energia que comunica por Rs485 MODBUS RTU. Este analisador é utilizado atualmente em chão de fábrica e permite realizar a análise de um circuito que utiliza energia trifásica para o seu funcionamento;
- **PLC Siemens S7-1200:** autómato programável que comunica por S7/ OPC-UA. Dispositivo utilizado em chão de fábrica;
- **Módulo OPC IBH Link UA:** módulo OPC Server que comunica por OPC-UA. Este dispositivo é utilizado na gestão do funcionamento dos autómatos através de um único dispositivo. Atualmente encontra-se em fases de testes para a sua implementação futura em chão de fábrica;

- **BME 280:** Sensor das condições atmosféricas que comunica por I^2C. Utilizado como exemplo de um dispositivo que utiliza um protocolo de comunicação típico de equipamentos mais simples.

Outro dispositivo utilizado na fase inicial do projeto foi um analisador de energia **SDM 120** (monofásico), que posteriormente foi substituído pelo Janitza.

A implementação da comunicação por Rs485 MODBUS RTU encontra-se no apêndice [B] e a por OPC-UA no apêndice [E].

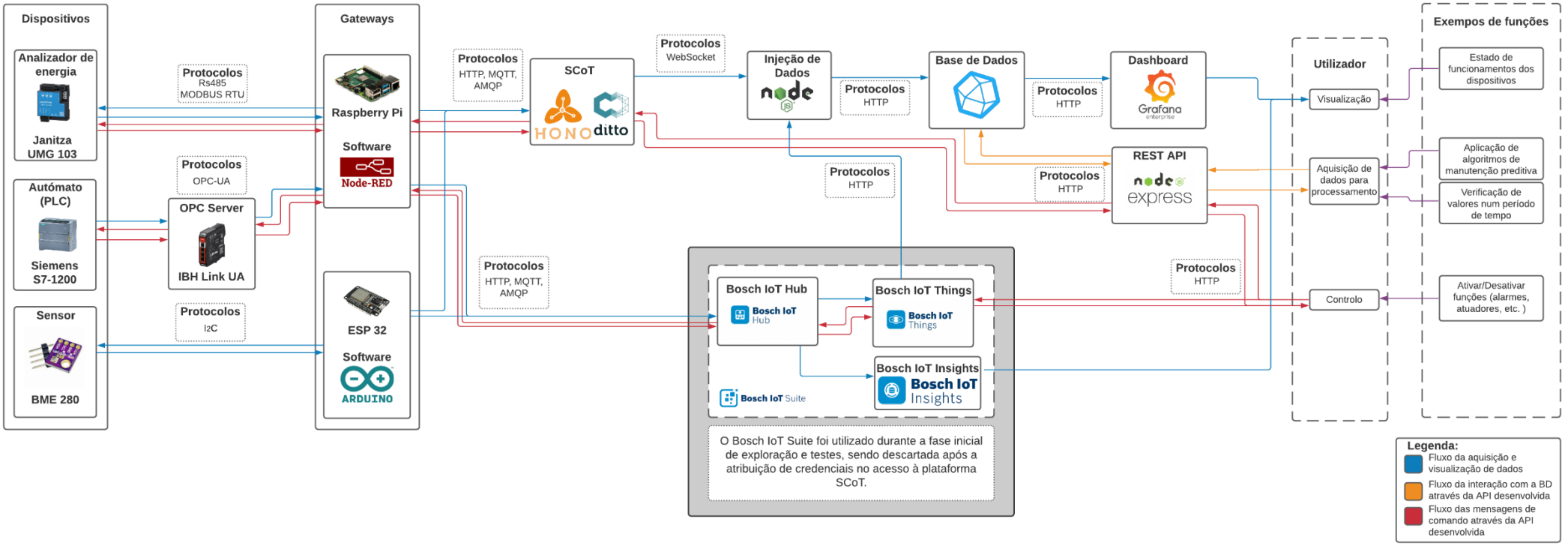

Figura 5.1: Diagrama do Arquitetura Proposta

5.2. Gateways

Assim como os dispositivos, a diversidade de equipamentos habilitados a exercer a atividade de *gateway* pretendida é elevada. A seleção destes equipamentos teve como base os seguintes parâmetros:

- Utilização, abundância e reputação do dispositivo no mercado, o que facilita a procura de informação relativamente a erros na internet;
- Facilidade de utilização e programação;
- Capacidade de execução nas condições envolventes do ambiente de inserção.

Os equipamentos utilizados foram os seguintes:

5.2.1 Raspberry Pi

Este pequeno computador permite utilizar um sistema operativo baseado em Linux, nomeado *Raspian*, possibilitando que sejam executados as mesmas aplicações como computador com outra variante do Linux, com limitações na sua capacidade de processamento. Esta flexibilidade de utilização permitiu que este equipamento viesse a ser muito utilizado atualmente em vários projetos diversificados: controlador de robôs, reprodutor de conteúdo media (vídeo e áudio) e videojogos, aquisição e segmentação de imagens, Web Server, (…) e também como *gateway* IoT entre os sensores e o servidor [13] [32].

O desenvolvimento do código utilizado foi realizado em **Node-Red**, programa *open-source* desenvolvido pela IBM para visualizar e manipular mapeamento de variáveis entre tópicos MQTT, que posteriormente foi expandido a outras funções devido à sua facilidade de utilização. Este programa recorre a uma programação visual (ligação entre blocos de funções), o que facilita o desenvolvimento e compreensão das diversas funcionalidades do programa. A linguagem utilizada é o *Javascript* e as funções são executadas em *Node.js*, o que cria um ambiente favorável ao desenvolvimento de novas funcionalidades pela comunidade (designadas por *palette*), uma vez que esta dupla é muito utilizada atualmente em desenvolvimento de serviços *backend [3]*.

O programa desenvolvido realiza quatro atividades em paralelo:

- Estabelecer a ligação entre o *gateway* e o *broker*;
- Estabelecer uma comunicação com os dispositivos físicos;
- Guardar a informação recebida numa base de dados local para o seu *backup*;
- Visualizar a informação recebida num *dashboard* local (se requerida);
- Executar mensagens de comando provenientes do *broker*.

Se algum processo não conseguir ser executado, o *NodeRed* emite uma mensagem de erro, não abortando o programa, mas sim realizando uma nova tentativa após um intervalo de tempo configurável.

Na figura seguinte encontra-se um diagrama que explica as etapas do código desenvolvido.

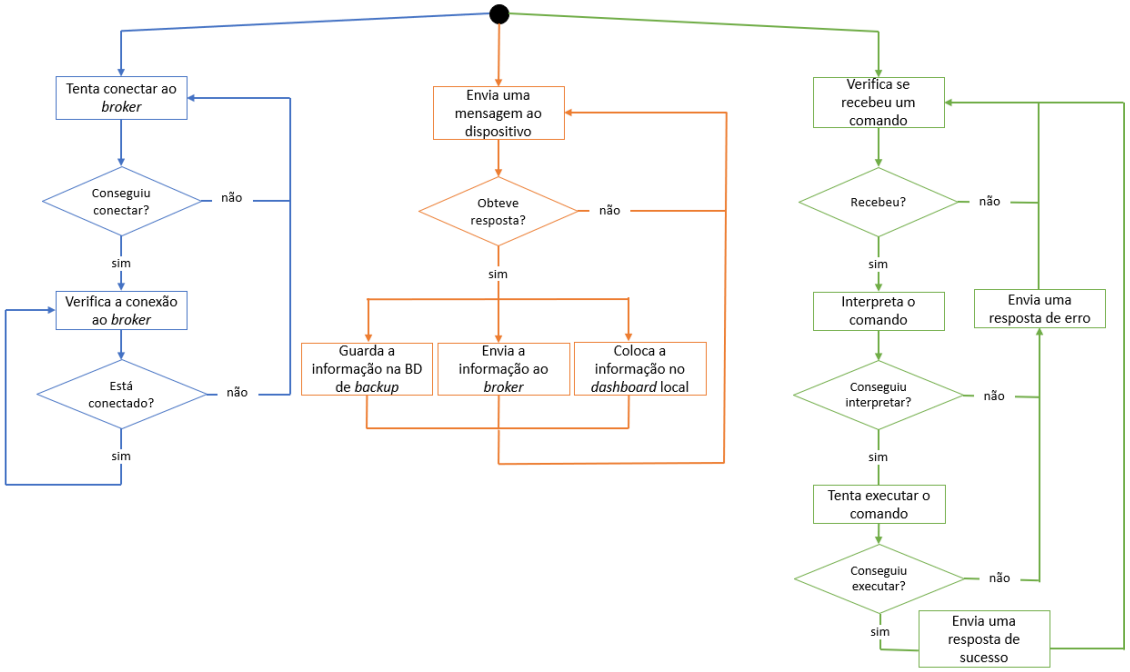

Figura 5.2: Diagrama UML de atividade do programa desenvolvido em Node-Red

5.2.2 ESP 32

O ESP 32 é um microcontrolador de baixo custo e consumo, desenvolvido especificamente para aplicações IoT. Estes microcontroladores são equipados por processadores com dois cores, possui alguns equipamentos integrados, nomeadamente um *shield* de Wi-Fi e *Bluetooth*, e *interfaces* I^2C, SPI e Ethernet [20][31][41].

O desenvolvimento do código foi realizado em **Arduino**, um IDE *open-source* baseado em C, utilizado no desenvolvimento de programas para *Arduino*, que após instaladas as devidas bibliotecas, permite o desenvolvimento de programas para ESPs.

O programa desenvolvido realiza a seguinte sequência de funções:

1. Tenta estabelecer uma ligação Wi-Fi com o *router*. Se não conseguir, volta a tentar.
2. Tenta estabelecer uma ligação MQTT com o broker. Se não conseguir, volta a tentar.
3. Subscreve a um tópico MQTT. Como não existia a necessidade de controlo remoto com algum equipamento conectado, esta ação não foi implementada no protótipo final.
4. Verifica se está conectado ao broker. Se não estiver, tenta reconectar.
5. Caso tenha sido implementada a subscrição a um tópico, verifica se existe alguma mensagem. Se não existir passa para a etapa 8.
6. Verifica se reconhece o comando. Se não reconhecer, responde com uma mensagem de erro.
7. Executa o comando. Envia uma resposta (positiva ou negativa) de acordo com o sucesso da execução.

8. Envia uma mensagem a pedir informação ao dispositivo.
9. Recebe a informação e formata-a para publicar ao *broker*.
10. Tenta enviar ao *broker*. Se não conseguir, informa (pela porta série) que não conseguiu publicar e volta à etapa 4. Se publicou, volta à etapa 4.

Na figura seguinte encontra-se um diagrama que explica as etapas do código desenvolvido.

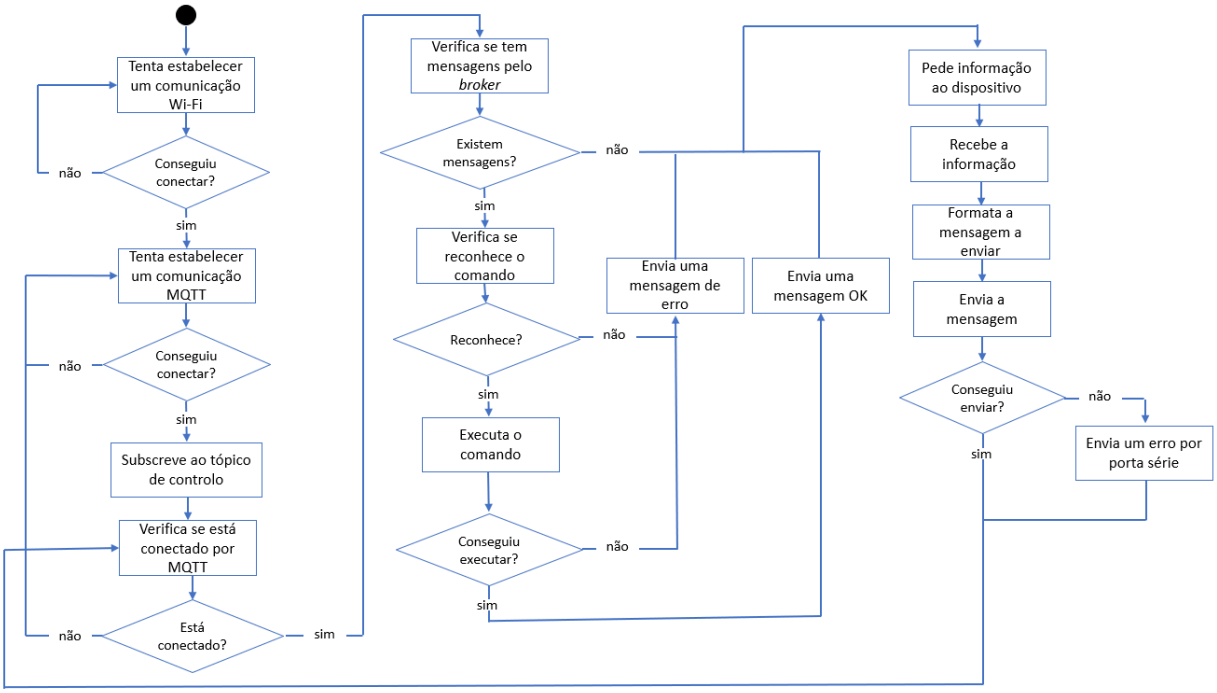

Figura 5.3: Diagrama UML de atividade do programa desenvolvido em Arduino

Os equipamentos utilizados são ideais no desenvolvimento rápido e fácil de um protótipo funcional, contudo a sua implementação em ambientes industriais de elevado ruído poderá comprometer a sua fiabilidade, o que cria a necessidade de migração para um outro equipamento, desenvolvido especificamente para estes ambientes adversos.

5.3. Broker

Os serviços utilizados na arquitetura proposta expandem o conceito convencional de *broker*, porque além de funcionarem como gestores do tráfego das mensagens entre os dispositivos e o servidor, também realizam a gestão dos próprios dispositivos através da atribuição automática de um endereço após a sua criação no sistema.

Durante a fase de desenvolvimento foram utilizados os seguintes serviços das seguintes plataformas *IoT*:

5.3.1 Bosch IoT Suite

Plataforma desenvolvida pela Bosch baseada no *Eclipse IoT*, que oferece um conjunto de serviços dedicados à interatividade e conectividade para aplicações IoT. Os serviços utilizados durante o desenvolvimento do projeto foram:

- **Bosch IoT Hub:** baseado no *Eclipse Hono*, estabelece e organiza o fluxo de informação entre o servidor e os dispositivos. A principal vantagem do *Hono* é a sua flexibilidade na implementação de protocolos de comunicação, uma vez que este realiza a adaptação de mensagens HTTP, MQTT e AMQP em AMQP, facilitando o desenvolvimento de serviços *backend*.
- **Bosch IoT Things:** baseado no *Eclipse Ditto*, permite realizar a criação e gestão dos *digital twins*.
- **Bosch IoT Insights:** serviço que reúne a função de *dashboard* e de base de dados (MongoDB).

5.3.2 SCoT

Plataforma desenvolvida pelo Departamento de Eletrónica, Telecomunicações e Informática (DETI) de Aveiro, também baseada no *Eclipse IoT*.

Apesar de se basearem no mesmo programa para a sua implementação, existem algumas diferenças na implementação dos protocolos de comunicação:

- No *Bosch IoT Suite*, a comunicação MQTT requere a utilização de um certificado SSL/TLS entre o *gateway* e o *broker*, no SCoT esta implementação não é necessária.
- A extração de informação do *Things* é realizada por HTTP, existindo a necessidade de autenticação prévia (por credenciais ou por JWT). O SCoT permite realizar a subscrição ao um tópico, por WebSocket, onde são reencaminhadas todas as mensagens provenientes dos *gateways*.
- Assim como na extração de informação, o envio de mensagens de comando pelo *Bosch IoT* é realizado por autenticação prévia. No SCoT os comandos também são executados por HTTP, mas além da necessidade de autenticação, a informação é codificada em base64.

A plataforma utilizada no protótipo final foi o SCoT, uma vez que esta plataforma será implementada num servidor da instalação fabril, contudo utilizou-se o *Bosch IoT Suite* na fase inicial para explorar e aprofundar os conhecimentos dos serviços oferecidos pela versão grátis da plataforma, úteis na implementação do *SCoT* devido às suas semelhanças.

O exemplo da implementação da comunicação entre o *gateway* e o *broker* da plataforma *SCoT* encontra-se no apêndice [C.1].

5.4. Injeção de Dados

Na constituição da arquitetura conceptual não foi considerada como seria executada a ligação entre o *broker* e a base de dados, contudo durante o desenvolvimento do protótipo exemplo para a arquitetura proposta, houve a necessidade de contruir um serviço que solucionasse este problema. O programa foi desenvolvido em *Node.js*, uma ferramenta que utiliza *JavaScript* para construir aplicações Web.

O programa desenvolvido realiza a seguinte sequência de funções:

1. Tenta estabelecer uma ligação WebSocket com o *broker*. Se não conseguir, tenta novamente.
2. Envia um pedido de subscrição das mensagens de telemetria.
3. Verifica se a comunicação WebSocket ainda está ativa. Se não estiver, volta para a fase 1.
4. Espera por uma mensagem proveniente do *broker*. Se não receber, volta para a fase 3.
5. Formata a mensagem recebida e insere a informação na base de dados. Volta para a fase 3.

Na figura seguinte encontra-se um diagrama que explica as etapas do código desenvolvido.

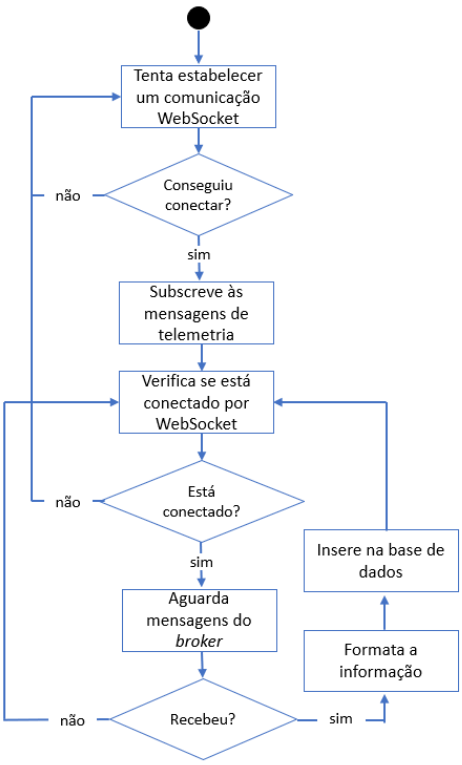

Figura 5.4: Diagrama UML de atividade do programa desenvolvido para injetar na base de dados

O exemplo da implementação do serviço responsável pela injeção dos valores provenientes da plataforma *IoT* (SCoT) na Base de Dados encontra-se no apêndice [C.2].

5.5. Base de Dados

Como fora indicado na arquitetura conceptual, o desenvolvimento da base de dados consiste na seleção de uma capaz de executar o problema em questão. No projeto desenvolvido, a necessidade básica da base de dados é registar a sequência da informação enviada pelos *gateways* para o *broker*. Assim, origina-se a necessidade de realizar o registo histórico desta informação, sendo necessária a utilização de uma base de dados dedicada a históricos (ou *logs* em inglês).

A base de dados selecionada foi a **InfluxDB**, uma base de dados *SQL timeseries* que é executada como um serviço no sistema. Esta base de dados oferece uma API que permite realizar as interações entre os serviços e a base de dados através de simples pedidos HTTP. Outras vantagens desta base de dados é a sua flexibilidade na gestão das medidas realizadas, porque a base de dados não necessita a elaboração prévia das tabelas (e as suas respetivas colunas), e a gestão da informação guardada ao longo do tempo, uma vez que é possível configurar a base de dados para agrupar estes dados pelo tempo dependendo da idade do registo realizado, o que diminui o tamanho da memória necessária a longo prazo.

Também foram realizados testes em outras duas bases de dados: a ***MySQL*** e a ***MongoDB***. O ***MySQL*** é uma base de dados tradicional *SQL* que apresenta alguma inflexibilidade em aplicações *IoT* relativamente à aquisição de dados para históricos, nomeadamente no limite das medições realizadas (linhas), a adição de novos parâmetros requere a intervenção externa (manual ou automática) e não possui um gestor de agregação automática da informação dependendo da sua idade, aumentando o tempo de pesquisa e capacidade de memória necessária. O ***MongoDB*** é uma base de dados *NoSQL Document*, permitindo a aquisição de uma grande variedade de informações diferentes, contudo a sua performance dependente da configuração das chaves utilizadas, que conjugada com a necessidade de configuração da gestão temporal da informação, requer um cuidado acrescido na sua implementação em projetos de maior dimensão [36].

5.6. Dashboard

A seleção do *dashboard* é semelhante à seleção da base de dados, tem de estabelecer os requisitos propostos na arquitetura conceptual. Assim, o programa de *dashboard* utilizado é o **Grafana**, um *dashboard* que possui uma *interface* fácil e intuitiva, tanto na criação de *templates* para os valores registados, assim como na elaboração das ligações com a base de dados. Outra grande vantagem deste *dashboard* é a sua interatividade com um amplo número de base de dados (nomeadamente o *InfluxDB*).

Outros *dashboard* utilizados foram o ***Bosch IoT Insights*** e o *dashboard* do ***Node-Red***. A utilização do ***Bosch IoT Insights*** é vantajosa numa arquitetura que utilize o *Bosch IoT Suite* como plataforma *IoT*, mas requere a necessidade de conhecimentos em *MongoDB* para a elaboração de *queries* para a extração de dados. Este *dashboard* foi descartado após a migração para o *SCoT*. O *dashboard* do ***Node-Red*** é flexível na sua organização e criação de *templates*, contudo é dependente do *Node-Red*, uma ferramenta desenvolvida para problemas menos complexos. Outra desvantagem deste *dashboard* é a necessidade do conhecimento em *javascript* para configurar a disposição da informação nos gráficos.

No apêndice [C.3] encontra-se um exemplo da criação e implementação de um *template* com o *Grafana*.

5.7. REST API

O programa da REST API fi desenvolvido em *Javascript/Node.js*, contudo a utilização do *Node.js* para um serviço multi-interativo compromete a sua fiabilidade devido ao seu funcionamento: o *Node.js* é executado em apenas uma *thread* do *cpu*, o que provoca uma ineficiência do seu acesso caso seja realizado um pedido e o serviço estiver a executar outro realizado anteriormente. Para contornar este problema utilizou-se a biblioteca **Express.js**, biblioteca desenvolvida especificamente para aplicações *Web*.

Em apêndice [D] é possível observar um exemplo simples que apresenta a base da REST API desenvolvida, onde foram posteriormente adicionados os seguintes serviços:

- Pedido de informação à base de dados relativamente aos dispositivos existentes em intervalos de tempo pré-definidos;
- Visualizar os dispositivos registados na base de dados;
- Enviar *queries* customizáveis a uma base de dados;
- Enviar *queries* customizáveis à *shell* do sistema de gestão da base de dados;
- Pedido de comandos a um dispositivo.

Como o desenvolvimento de uma API é um processo iterativo, a implementação de novos serviços é simples e intuitiva, expandido a longevidade da sua utilização e manutenção. Também existe disponível uma documentação básica relativamente aos serviços disponíveis.

O programa desenvolvido realiza a seguinte sequência de funções:

1. Indica a porta onde está a ser executado o serviço;
2. Permanece à espera por um pedido de um utilizador;
3. Se receber um pedido, verifica qual a função pedida (conectar com a Base de Dados ou envio de mensagem de comando);

Pedido à Base de Dados

4. Formata o pedido realizado numa *query* e envia à API da Base de Dados;
5. Se obtiver resposta, formata-a de csv para JSON e envia ao utilizador. Se não obtiver, envia uma mensagem de erro ao utilizador;
6. Retorna à espera de um novo pedido.

Mensagem de comando

7. Formata o pedido realizado, o tempo do pedido e envia para o *broker*;
8. Se não obtiver resposta do *broker* ou este não conseguiu executar o pedido, envia uma mensagem de erro ao utilizador. Se o pedido foi executado, reencaminha-o para o utilizador;
9. Retorna à espera de um novo pedido.

Na Figura 5.5 é possível observar o funcionamento da API desenvolvida, através da exemplificação das várias etapas executadas.

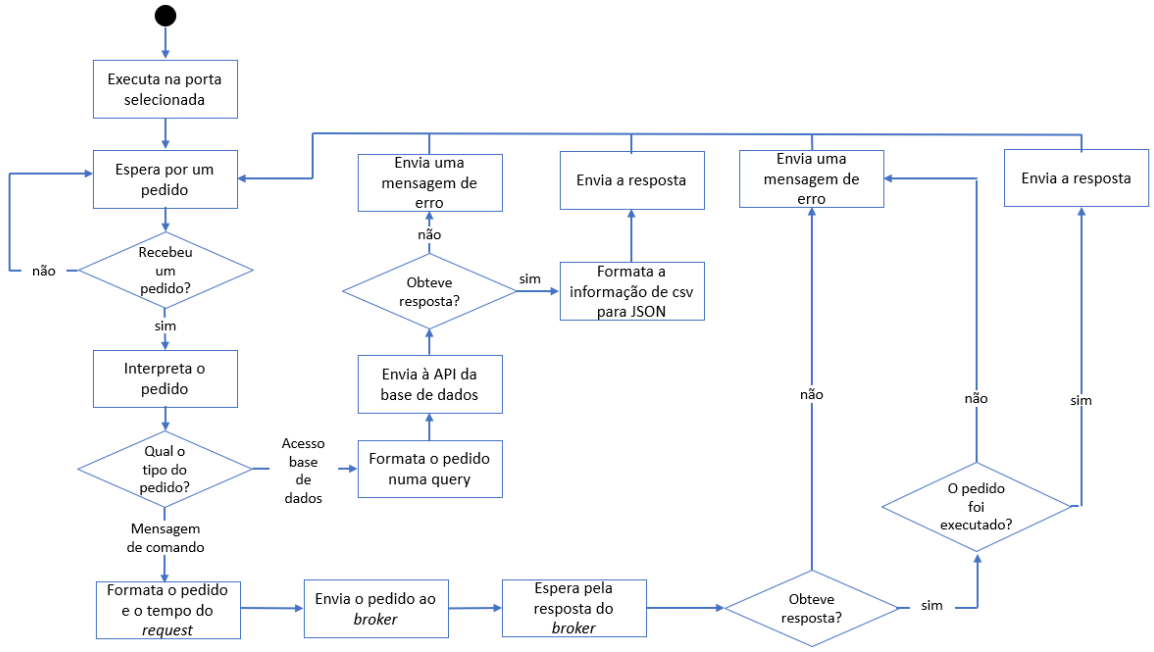

Figura 5.5: Diagrama UML de atividade dos pedidos realizados à API desenvolvida

5.8. Fluxo de informação

A explicação dos vários componentes da arquitetura proposta permite compreender a função de cada um e a sua interatividade com cada um dos seus vizinhos, mas não demonstra, de forma clara e objetiva, como é realizado o fluxo da informação da origem até ao consumidor final. Nos subcapítulos seguintes é possível visualizar um conjunto de diagramas que explicar como é realizado o tráfego de informação em cada uma das interações implementadas, onde cada cor representa o respetivo fluxo de

informação na Figura 5.1: azul para a aquisição e visualização de informação, amarelo para a interação da Base de Dados a partir da *REST API*, e vermelho para o envio de mensagens de comando pela *REST API*.

5.8.1 Aquisição e visualização de informação

Na Figura 5.6 é possível visualizar o fluxo da informação, desde o início do registo pelo *gateway* (Raspberry e ESP32), que envia um pedido a um dos equipamentos (analisador de energia, módulo OPC e sensor meteorológico), formata a sua resposta e publica no *broker* da plataforma *IoT.* Depois a plataforma reencaminha-a para uma aplicação que subscreveu (por WebSocket) aos pedidos de telemetria realizados e adiciona-a na Base de Dados, para posteriormente ser acedida.

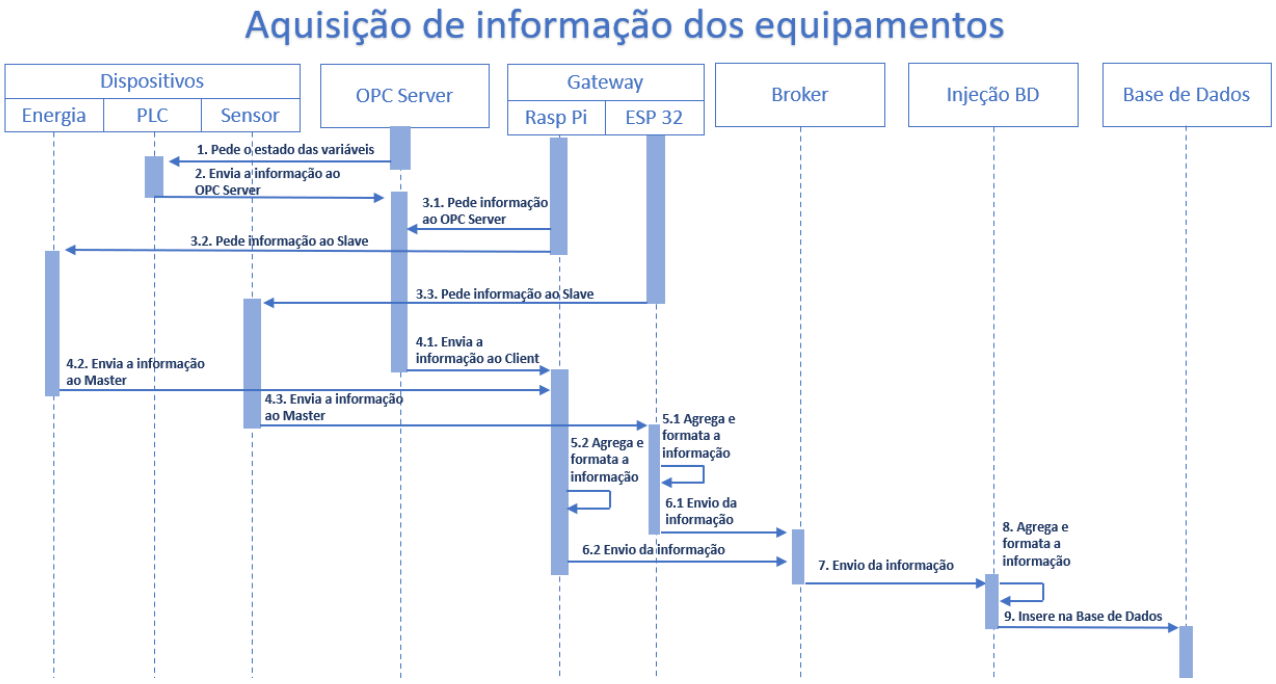

Figura 5.6: Diagrama UML de sequência do fluxo de informação na obtenção de dados dos equipamentos

Na Figura 5.7 encontra-se o diagrama do fluxo de informação na interatividade com o *dashboard*. O pedido inicial é realizado pelo utilizador, que ao abrir o *template* do equipamento pretendido, indica ao *dashboard* a informação que pretende observar. Então o *dashboard* realiza um pedido dessa informação à Base de Dados, de onde posteriormente é concedida uma resposta. Com a informação em posse, o *dashboard* formata-a nos gráficos pré-estabelecidos e envia ao utilizador a interface gráfica desenvolvida.

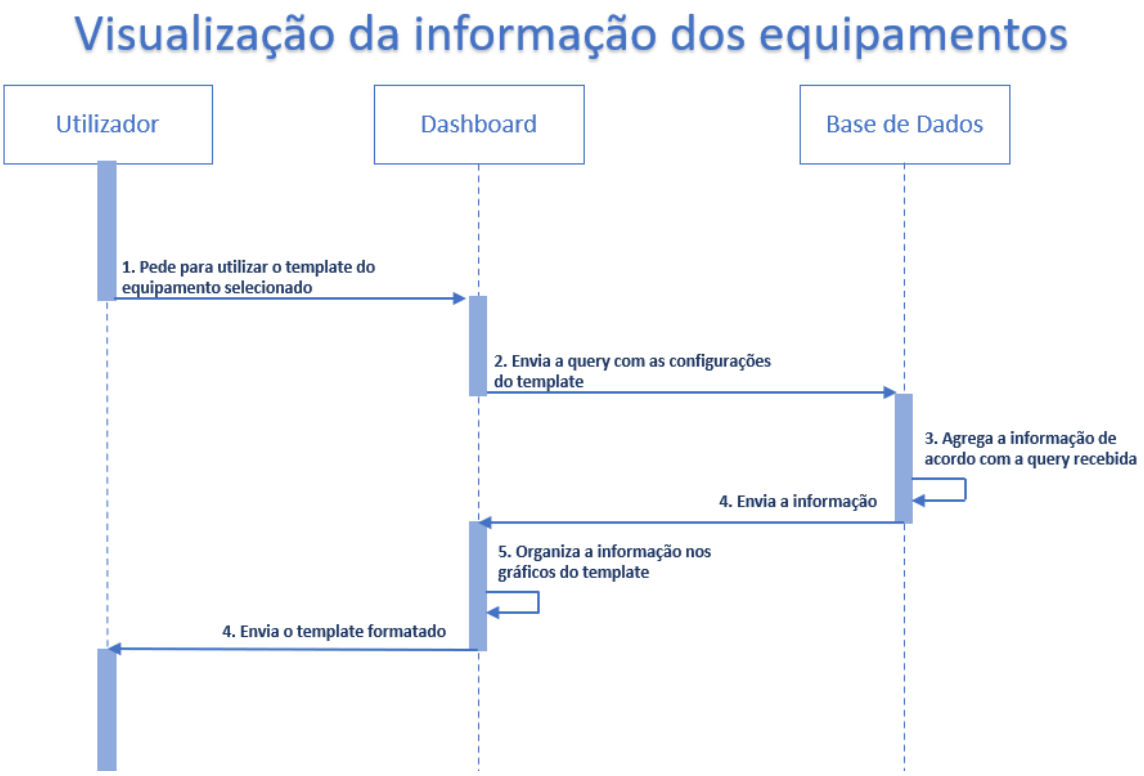

Figura 5.7: Diagrama UML de sequência relativo ao fluxo de informação na utilização do dashboard

5.8.2 Aquisição de informação da Base de Dados

Na Figura 5.8 é possível visualizar o fluxo informação relativamente à aquisição de informação existente na Base de Dados, onde o utilizador realiza o pedido *HTTP* à API desenvolvida, seguida da conversão do pedido numa *query* que posteriormente é enviada para a Base de Dados. Depois de receber a informação da Base de Dados, a API converte a informação no formato JSON e reencaminha-a para o utilizador.

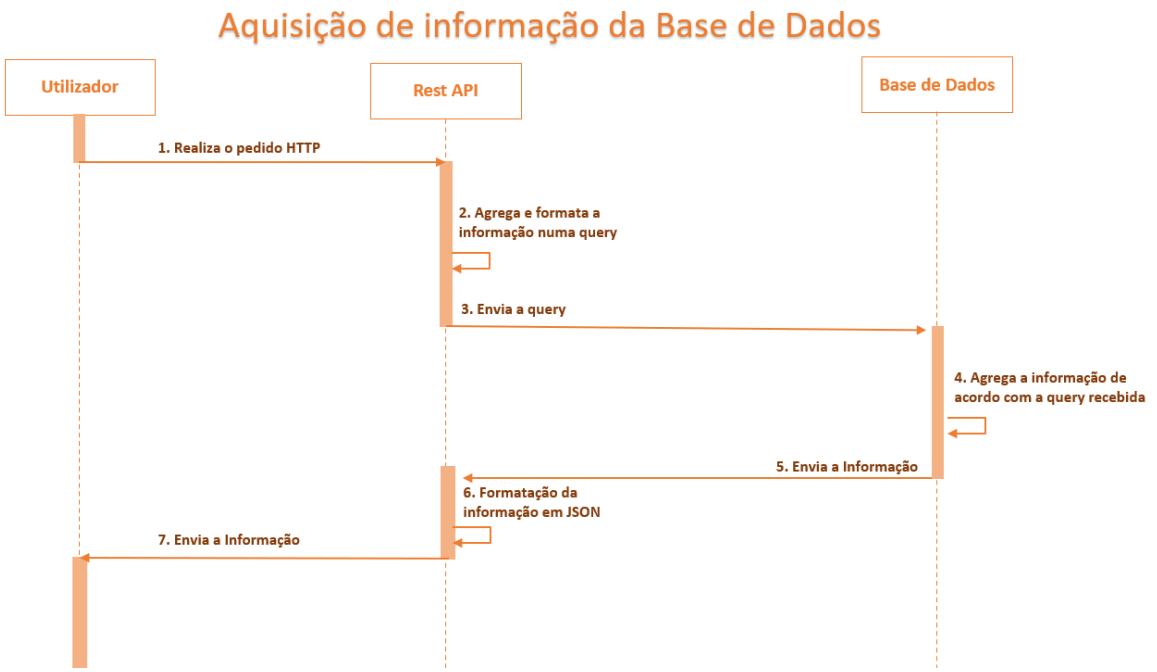

Figura 5.8: Diagrama UML de sequência relativo ao fluxo de informação na utilização da REST API para interagir com a base de dados

5.8.3 Envio de mensagens de comando

Na Figura 5.9 é possível visualizar o percurso da informação relativamente ao envio de mensagens de comando a um dos dispositivos utilizados, onde o utilizador realiza o pedido *HTTP* à API desenvolvida, que adapta o pedido de acordo com os parâmetros da plataforma *IoT* (*broker)*. Após receber o pedido, o *broker* reencaminha-o para o *gateway* onde o equipamento se encontra ligado. Quando o *gateway* recebe o pedido, verifica se o compreende (se não o compreender, envia uma resposta a indicar o ponto da situação) e executa comando requerido até receber uma resposta do equipamento, onde depois envia uma mensagem de resposta para a plataforma *IoT*, que a reencaminha para a API desenvolvida. Por fim, a API desenvolvida envia a mensagem de resposta recebida para o utilizador.

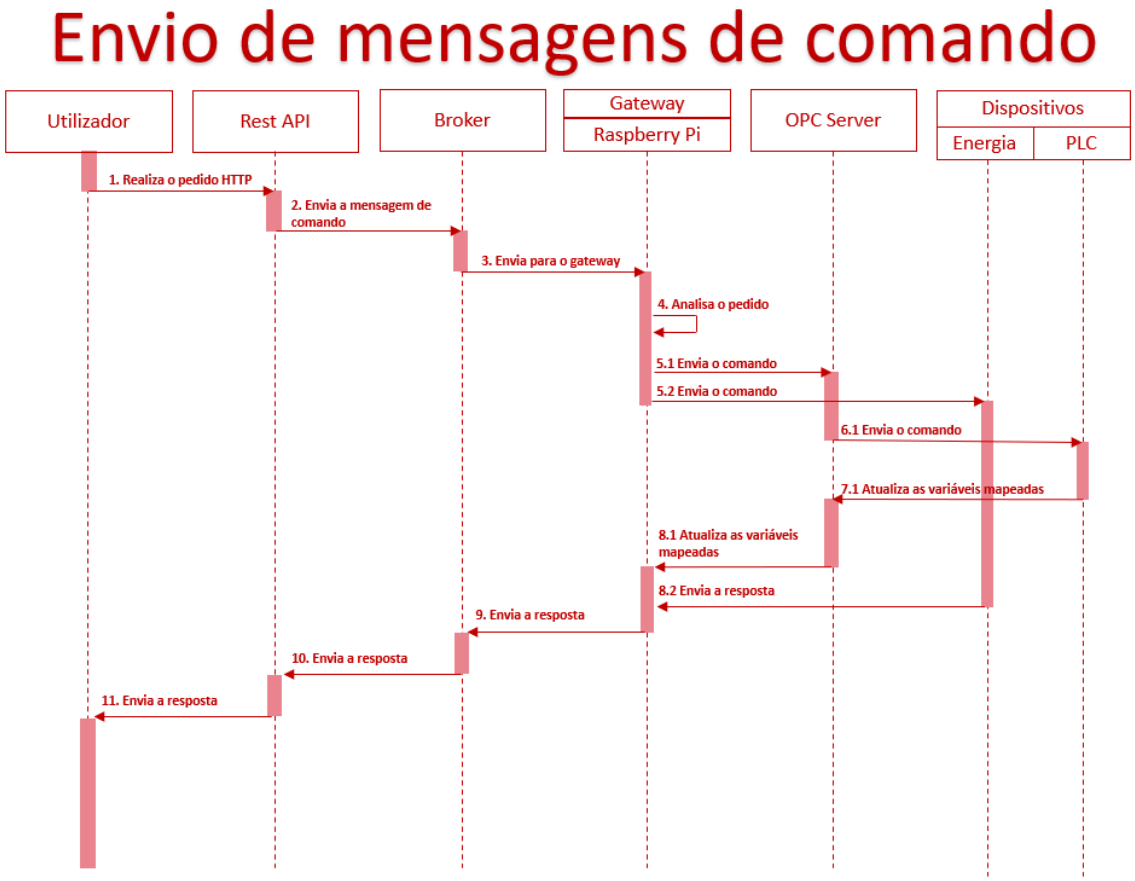

Figura 5.9: Diagrama UML de sequência relativo ao fluxo de informação na utilização da REST API para enviar mensagens de comando

Capítulo 6

Análise dos resultados

A análise do desempenho da arquitetura proposta, perante os objetivos inicialmente indicados, será efetuada ao longo deste capítulo. Esta análise foi dividida nas seguintes etapas:

- **Aquisição e visualização de dados de energia:** esta etapa permite verificar o funcionamento de todos os componentes implementados na arquitetura relativamente à aquisição, armazenamento e visualização da informação proveniente dos equipamentos (dispositivos (analisador de energia), *gateways* (Raspberry Pi), plataforma *IoT* (*SCoT*), Injeção na BD, Base de Dados (*InfluxDB*) e *dashboard* (*Grafana*)).
- **Interatividade com a REST API:** nesta etapa será realizada a análise do funcionamento dos serviços desenvolvidos na REST API, nomeadamente a **comunicação com a base de dados** (na aquisição de informação dos dispositivos (analisador de energia)) e no **envio de mensagens de comando** aos dispositivos (analisador de energia).
- **Latência entre o *gateway* e a base de dados:** onde será analisado o tempo entre a publicação de uma mensagem, por *MQTT*, pelo *gateway* e a sua receção pelo programa responsável pela sua injeção na base de dados, por *WebSocket*.
- **Desempenho da REST API**: nesta etapa será executado um pequeno teste de stress, através do envio sucessivo de mensagens de comando e de acesso à base de dados, para verificar a taxa de sucesso dos pedidos realizados.
- **Latência de mensagens de comando:** consiste na análise de dois intervalos de tempo no envio de mensagens de comando através do uso da REST API desenvolvida, uma entre o pedido do utilizador e a sua chegada ao *gateway*, e outra entre a realização do pedido pelo utilizador e a sua resposta.
- **Consumo do *gateway*:** permite verificar o consumo de energia médio pelo *gateway* (*Raspberry Pi*), assim como os custos associados.

Para a elaboração dos ensaios previamente indicados, foi utilizado o *setup* indicado na Figura 6.1, onde se utilizou o Raspberry Pi para comunicar por *Rs485 MODBUS RTU* com o analisador de energia Janitza e publicar esta informação para a plataforma *IoT,* pela internet, uma vez que esta plataforma não se encontra na mesma rede local que os restantes componentes. A Base de Dados, *Dashboard* e *REST API* encontram-se em execução num computador.

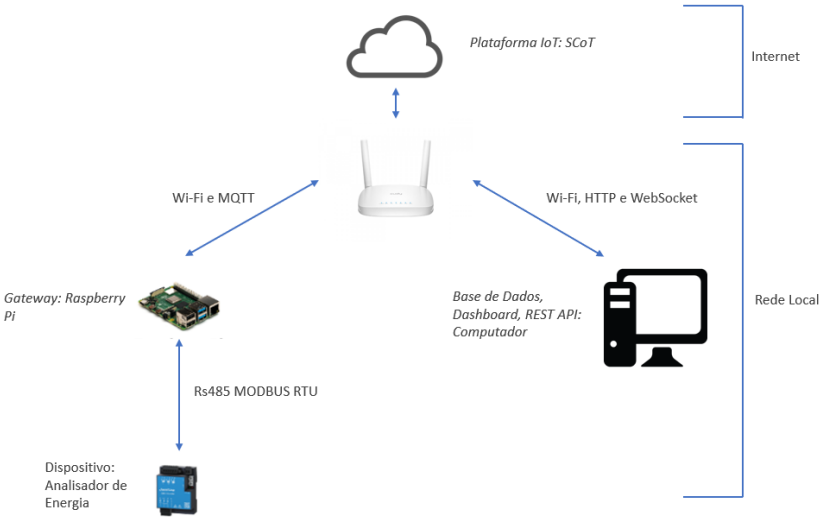

Figura 6.1: Setup utilizado na obtenção de resultados

6.1. Aquisição e visualização de dados de energia

As exemplificações dos processos utilizados para a realização desta avaliação encontram-se nos apêndices [B] e [C]. Nas figuras seguintes é possível observar os resultados obtidos no *template* do *dashboard* utilizado.

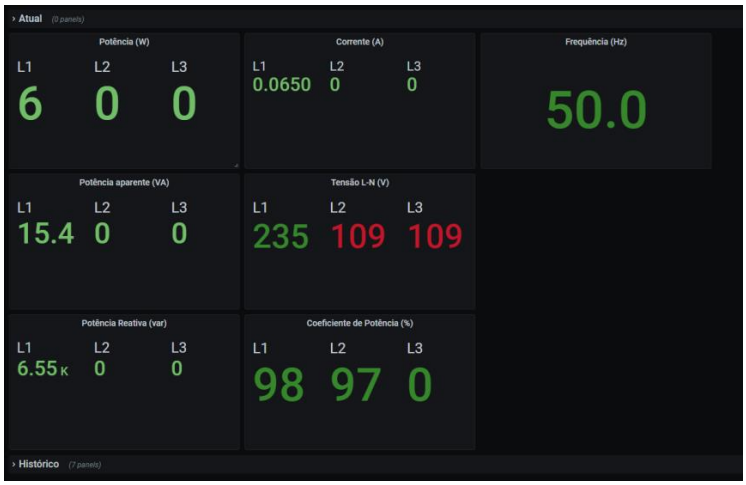

Figura 6.2: Template do analisador de energia dos dados 'live'

Figura 6.3: Template do analisador de energia dos dados histórico

Na Figura 6.1 é possível observar os dados "live" do analisador de energia, e na Figura 6.2 o histórico destes dados. Também é possível alterar o intervalo de tempo para visualizar a informação desse intervalo, mas os dados "live" continuam a demonstrar o estado atual do funcionamento do dispositivo. Na Figura 6.2 apenas é apresentada a informação entre a fase1 e o neutro (foram utilizados equipamentos monofásicos durante a fase de testes), contudo também é realizada a aquisição das propriedades das outras duas fases (que não se encontram em funcionamento atualmente). A publicação dos dados do PLC para a plataforma não foi efetuada devido ao tempo limitado de manuseamento do dispositivo, contudo os resultados do seu funcionamento encontram-se no apêndice [E].

6.2. Interatividade com a REST API

Esta avaliação será dividida em duas etapas: a comunicação com a base de dados e com um *gateway*.

6.2.1 Comunicação com a base de dados

Nesta análise utilizou-se um exemplo em *NodeRed*, onde se realiza um *HTTP request* à API. O URL utilizado na realização do pedido foi o seguinte: http://localhost:8080/api/devices/energy/data/teste/all/time/30m, onde se pede toda a informação ('all') existente na medição "teste" nos últimos 30 minutos.

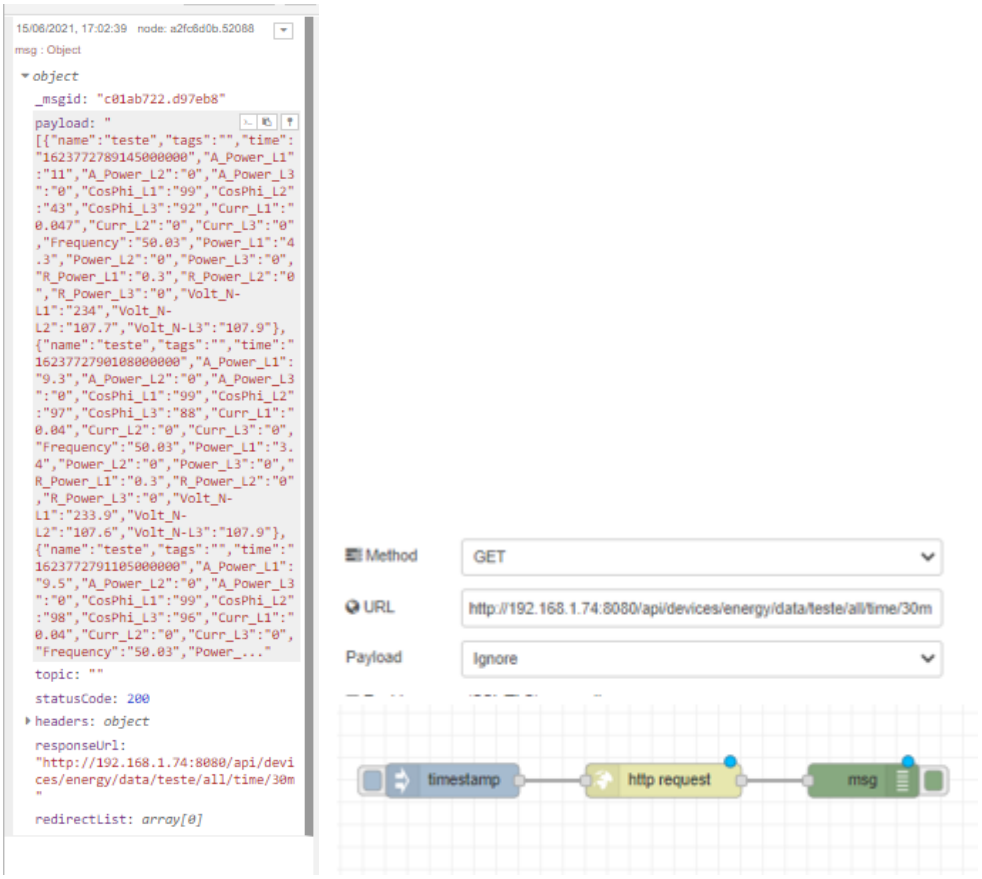

Figura 6.4: Pedido e resultados da ligação API - base de dados

Como é possível observar na figura anterior, a API realizou a "ponte" entre o utilizador e a base de dados, devolvendo um JSON com toda a informação pedida no URL enviado.

6.2.2 Comunicação com o dispositivo

A análise de desempenho da API para comunicar com os dispositivos também foi realizada em *NodeRed.* Neste exemplo realiza-se um pedido ao equipamento "teste" para realizar a leitura de uma variável interna no analisador de energia (200 neste caso, que é a tensão entre a fase1 e o neutro, multiplicado por 10). O URL utilizado foi o seguinte: *http://localhost:8080/api/devices/energy/command/teste/200*.

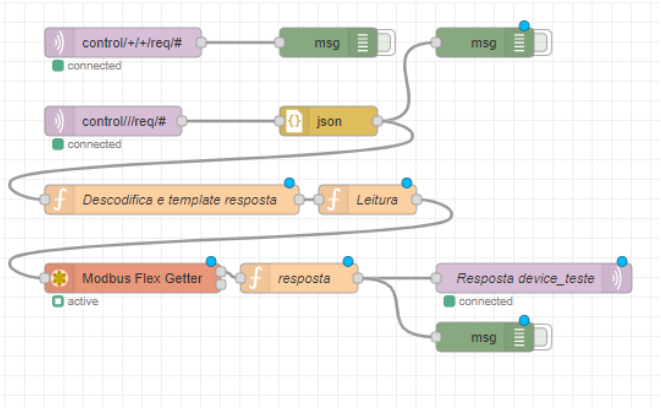

Figura 6.5: Programa implementado no gateway

Figura 6.6: Resposta do comando realizado

Na imagem anterior é possível observar a resposta ao pedido (‘*payload*’) realizado pela API, onde se obteve a resposta 2356, que corresponde a 235,6 Volts, tensão habitual no espectro fornecido pelas empresas de energia na Europa.

6.3. Latência *gateway* / base de dados

A avaliação de desempenho relativamente à latência da informação entre o *gateway* e a base de dados será realizada com recurso a um programa, desenvolvido em Node-Red, que envia uma mensagem com o *timestamp* atual ao *broker* por MQTT, subscreve as mensagens de eventos do *broker* por WebSocket e subtrai ao *timestamp* atual, o recebido na mensagem. Como a plataforma *IoT* não se encontra integrada na rede local, realizaram-se variações na utilização da largura de banda, através de um outro computador, uma vez que esta ligação (por Internte) apresenta o maior efeito gargalo do protótipo desenvolvido. A publicação de mensagens foi num intervalo de 10ms.

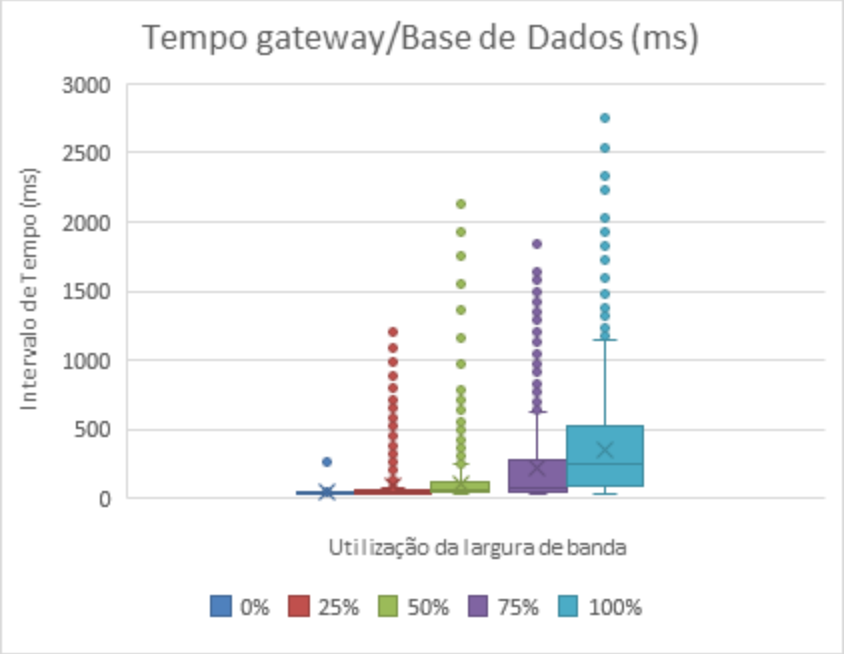

Figura 6.7: Latência entre o gateway e a base de dados, em ms

Tabela 6.1: Valores estatísticos dos resultados obtidos

WI-FI LOAD (%)	MEAN (MS)	MEDIAN (MS)	ST. DEVIATION (MS)
0	41	39	13
25	88	42	148
50	110	60	158
75	226	78	298
100	356	247	343

Nos resultados obtidos é possível observar a performance do sistema nas condições ótimas (com uma utilização de 0%) e a evolução deste intervalo de tempo com o aumento da utilização da largura de banda por outro equipamento (computador) na mesma rede. Entre 25% a 50% de utilização, já é possível observar uma degradação na consistência deste envio de mensagens, podendo comprometer a sua utilização na monitorização de equipamentos mais delicados. Quando a 75% e 100%, a sua utilização é possível em equipamentos onde a sua monitorização não intervenha imediatamente no funcionamento e saúde do ambiente produtivo (por exemplo a monitorização do consumo energético com a única finalidade de mapear o consumo deste equipamento num intervalo de tempo), mas não aconselhada.

A latência da comunicação pela internet é afetada por um conjunto de parâmetros diferentes (velocidade da ligação, distância entre o cliente e o servidor, protocolos utilizados, entre outros) que comprometem a consistência destes resultados, contudo, nas condições utilizadas (±30km de distância com o *broker* e utilizando uma comunicação

ADSL). Outros parâmetros extrínsecos à ligação pela Internet, como a qualidade de serviço (QoS), o tamanho da mensagem e o número de equipamentos conectados à mesma rede local, interferem nos resultados obtidos, podendo afetar o seu desempenho final.

6.4. Desempenho da REST API

A análise da eficiência da REST API desenvolvida terá como objetivo verificar a eficácia dos serviços desenvolvidos através da realização de pedidos consecutivos. A realização do programa para testes foi realizada em Node-Red, onde se enviou uma sequência de 10 mensagens (5 de comando e 5 à base de dados), com um intervalo de 100ms, e verificou-se a existência de uma resposta para cada uma, assim como o intervalo de tempo entre respostas.

Na Figura 6.8 é possível visualizar os resultados obtidos com o programa desenvolvido. Neste exemplo realizou-se o pedido do valor da tensão entre a fase 1 e o neutro 5 vezes, seguidos de 5 pedidos dos valores de todos os parâmetros do analisador de energia, nos últimos 30 minutos. Os resultados o bom funcionamento da API, porque além de executar todos os pedidos realizados (com sucesso), o tempo necessário para os executar foi reduzido, permitindo assim afirmar que o serviço criado não compromete o funcionamento da arquitetura.

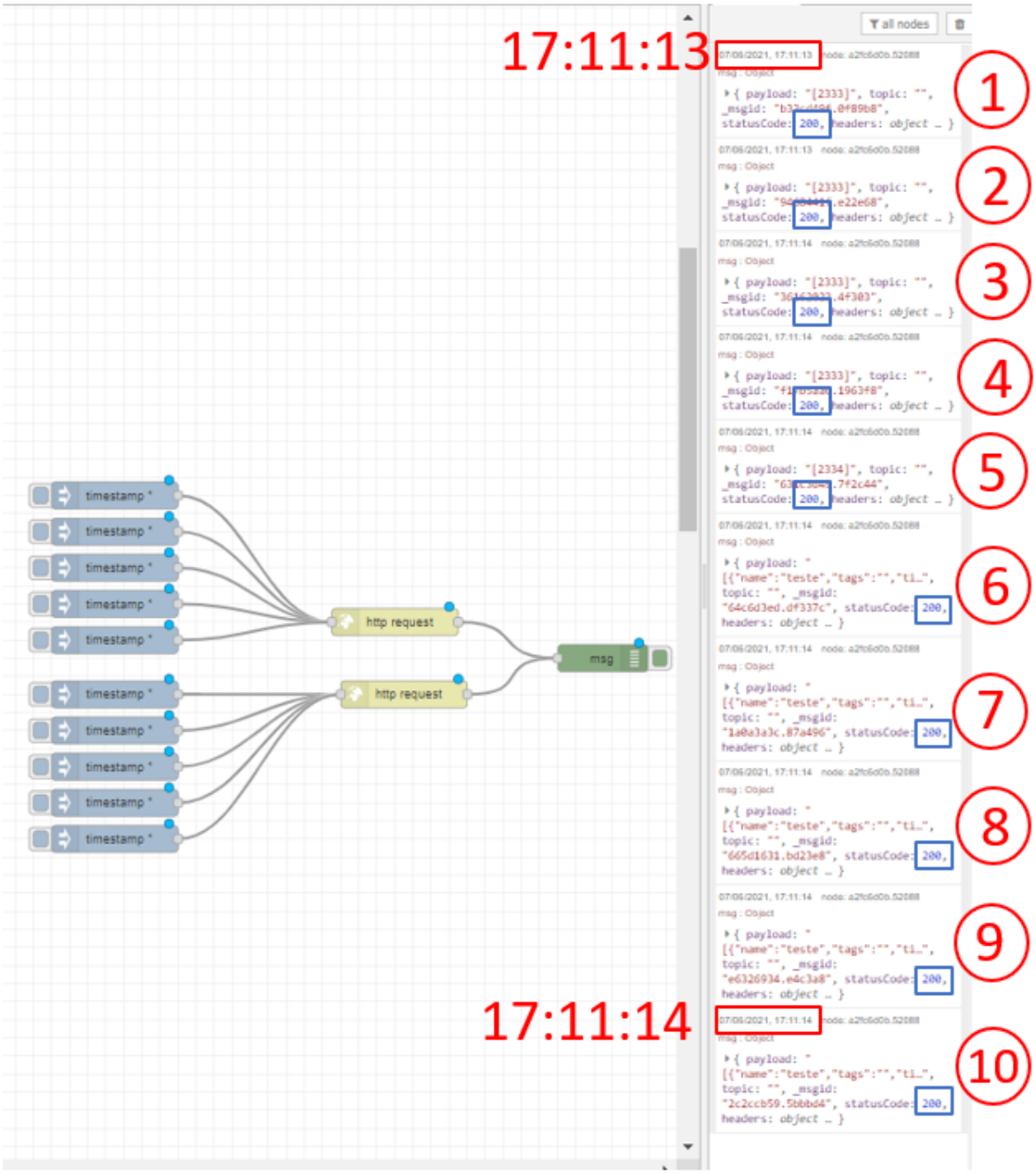

Figura 6.8: Resultados obtidos no teste da REST API desenvolvida

6.5. Latência de mensagens de comando

A avaliação da latência das mensagens de controlo será análoga à realizada para as mensagens de telemetria entre o *gateway* e a base de dados, contudo o pedido de comando é realizado por HTTP, em vez de WebSocket. Neste exemplo serão analisados dois intervalos de tempo: um entre o cliente e o *gateway* e outro entre o *gateway* e o cliente, portanto será enviado um pedido ao *gateway* onde é enviado o *timestamp* atual,

o *gateway* responde com o seu *timestamp* atual e, no final, realiza-se a subtração entre o *timestamp* final e os dois *timestamps* das mensagens. A publicação de mensagens foi num intervalo de 1s.

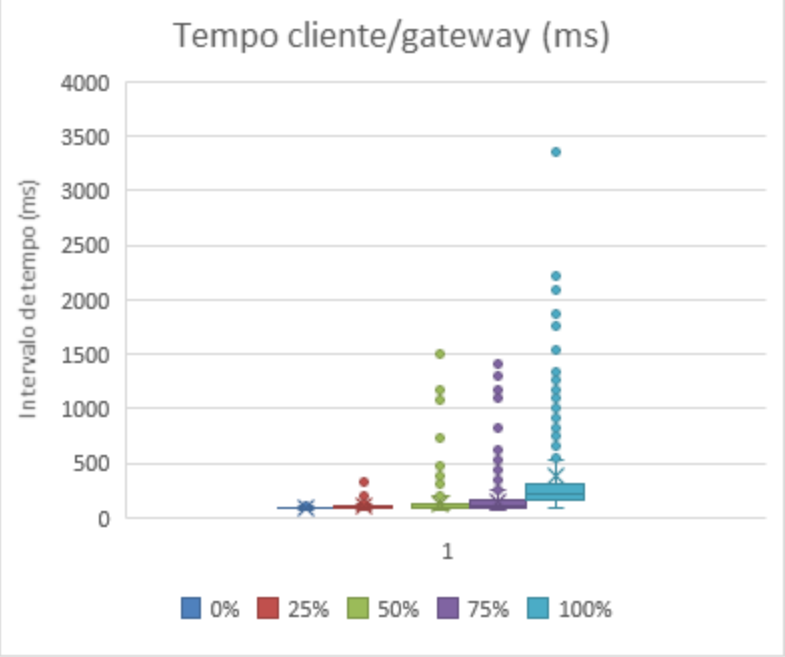

Figura 6.9: Latência entre o cliente e o gateway

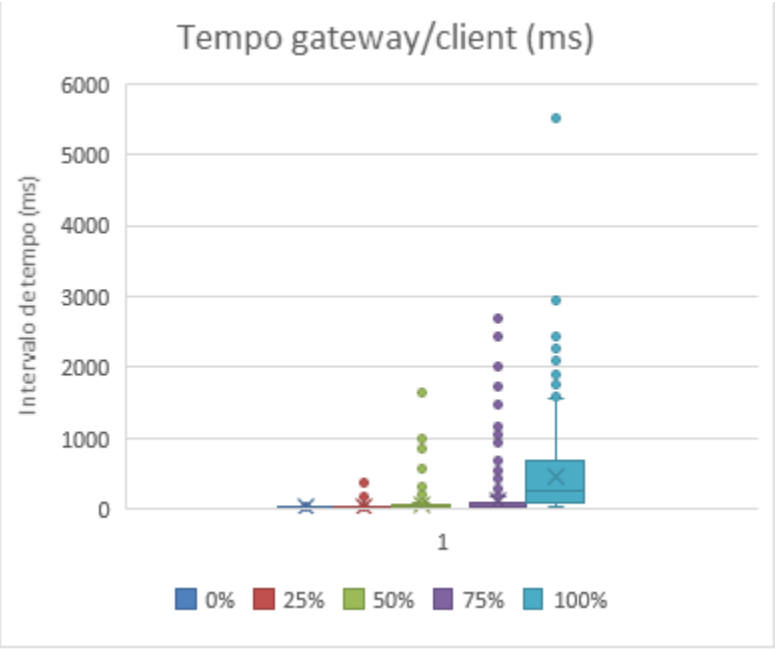

Figura 6.10: Latência entre o gateway e o cliente

Tabela 6.2: Valores estatísticos dos resultados entre o cliente e o gateway

WI-FI LOAD (%)	MEAN (MS)	MEDIAN (MS)	ST. DEVIATION (MS)
0	93	91	8
25	101	90	32
50	130	92	128
75	151	109	151
100	378	218	418

Tabela 6.3: Valores estatísticos dos resultados entre o gateway e o cliente

WI-FI LOAD (%)	MEAN (MS)	MEDIAN (MS)	ST. DEVIATION (MS)
0	37	36	3
25	46	36	39
50	75	36	146
75	130	44	272
100	469	269	551

Este exemplo foi realizado nas mesmas condições da primeira análise realizada, contudo utilizou-se a API desenvolvida para auxiliar na implementação dos pedidos, com o objetivo de obter resultados mais realistas. Ou fator que afeta a latência das mensagens é a avaliação das condições dos dispositivos relativamente ao comando recebido (resposta após o envio do comando ou após o comando ser executado), contudo, neste exemplo não foi realizado um comando a um dispositivo, mas sim um pedido do *timestamp* atual do *gateway*, que por sua vez provocaria um aumento do tempo de resposta. Assim, esta análise apresenta uma avaliação simples e básica de uma resposta teste pelo *gateway.*

A utilização de comandos remotos não é uma necessidade atual que requeira a sua implementação, contudo os resultados obtidos são favoráveis para a sua futura implementação. A 0% é possível observar os resultados em condições ótimas, contudo a 25% também existem resultados favoráveis em aplicações delicadas (contrariamente à monitorização, analisada anteriormente, uma vez que a frequência de pedidos foi muito inferior (100x), diminuindo o tráfego de mensagens entre componentes). A 50%, a análise é análoga relativamente à monitorização, mas a 75% e 100%, devido à natureza de utilização, os resultados obtidos invalidam a sua utilização.

6.6. Consumo do *gateway*

A avaliação da potência permitirá avaliar o consumo energético do *gateway* instalado (neste caso um Raspberry Pi 4 B). Os valores desta análise foram obtidos do *dashboard* do analisador de energia, que por sua vez apenas possuía o transformador do *gateway* instalado.

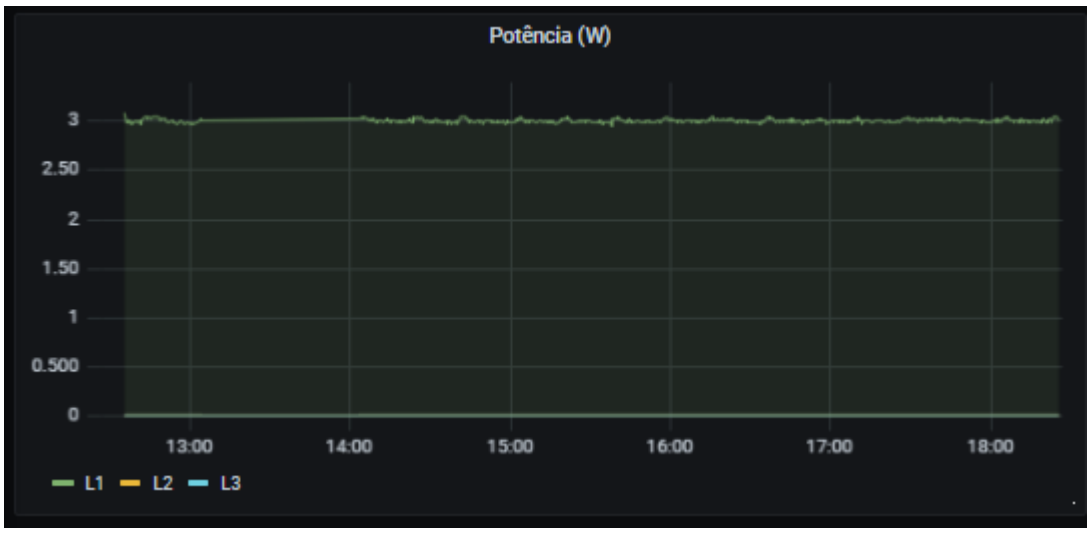

Figura 6.11: Potência elétrica utilizado pelo gateway durante um intervalo de tempo

Na imagem anterior é possível observar a evolução da potência requeria pelo *gateaway* ao longo de 5 horas. É possível verificar que existiu um pico quando se inicializou o equipamento, seguido de uma necessidade constante de 3W. Tendo em consideração que o custo de 1kWh é de 0,151€[39], o custo de operação anual do *gateway* é de 3,97€ anualmente (24h por dia). A fórmula utilizada para o cálculo foi a seguinte:

$$V = C * \frac{P * h * d}{1000}$$

onde, *V* é o custo anual do funcionamento do *gateway* em euros, *C* é o preço de um kWh em euros, *P* é a potência média do *gateway* em Watts, *h* é o número de horas de funcionamento e *d* é o número de dias de funcionamento por ano.

Capítulo 7

Conclusão e trabalhos futuros

A solução proposta comprovou-se capaz de satisfazer todos os requisitos propostos inicialmente numa aplicação piloto, permitindo a aquisição dos dados do funcionamento de diversos dispositivos, assim como o envio de mensagens de comando através de uma rede (local ou externa). Contudo, a flexibilidade da gestão e manuseamento desta solução é limitada, o que força a intervenção humana no *software* desenvolvido.

A possibilidade de subscrever a todas as mensagens de eventos (telemetria) dos dispositivos, por WebSocket, favoreceram a flexibilidade e simplicidade na utilização da plataforma *IoT SCoT*, contudo o *Bosch IoT Suite* possui uma documentação mais explícita de todas as suas funcionalidades.

O *InfluxDB* apresenta resultados favoráveis na aquisição e armazenamento dos dados temporais, contudo apenas permite a utilização de dados em forma de texto. Caso seja necessário expandir as funcionalidades para o armazenamento de imagens, seria necessário adicionar uma base de dados NoSQL (por exemplo o *MongoDB*).

O *Grafana* é uma plataforma de *dashboard* fácil de utilizar, onde a sua uma API fornece serviços de ligação a um amplo conjunto de bases de dados, contudo existem dois fatores que possam vir a comprometer o seu funcionamento: a taxa mínima de atualização é de 5 segundos, o que pode comprometer a monitorização de algum equipamento em específico, e o aumento de dispositivos conectados a esta interface podem dificultar a sua organização.

A REST API desenvolvida apresenta bons resultados nos serviços desenvolvidos, onde executou todos os pedidos realizados num curto período de tempo e a sua utilização facilita a interação com a arquitetura desenvolvida.

Na perspetiva de melhorar o funcionamento e o desempenho desta arquitetura propunham-se as seguintes tarefas:

- Criação de um programa que identifique os parâmetros necessários para a implementação automática de um dispositivo novo na arquitetura. Estes parâmetros poderão, por exemplo, estar identificados numa configuração do *digital twins*, onde o *gateway* realizaria uma verificação automática sistemática destes parâmetros e adaptasse o seu funcionamento a partir desse registo. Esta implementação permitiria flexibilizar o manuseamento e a instalação dos dispositivos em chão de fábrica.

- O protótipo foi desenvolvido com o objetivo de realizar uma implementação do zero, contudo, em arquiteturas previamente implementadas, é provável existir uma rede dedicada à aquisição destes dados, através do uso de serviços alocados num servidor. O desenvolvimento de um serviço *"virtual gateway"* que funcione paralelamente permitirá utilizar esta infraestrutura para a obtenção dos resultados, diminuindo os custos associados aos *gateways* físicos e os seus periféricos.
- A implementação de um sistema de segurança na API desenvolvida (por exemplo credenciais) permitirá gerir as permissões de acesso a diferentes níveis de utilizadores.
- O desenvolvimento de um *dashboard* que permitisse criar uma planta interativa da instalação fabril promoveria a interatividade entre o utilizador e os dispositivos. Este *dashboard* permitiria ao utilizador identificar a posição dos dispositivos em chão de fábrica, visualizar o seu funcionamento e de realizar a agregação dos vários dispositivos por áreas, realizando um balanço médio dos dados recebidos.
- Atualmente os *gateways* comunicam com o *broker* a partir de uma rede Wi-Fi, contudo, a evolução do desempenho das redes móveis (por exemplo 4G e 5G) vem permitindo um aumento da densidade de dispositivos conectados, sendo este um dos principais desafios da Indústria 4.0. A utilização de um dispositivo como *gateway* que estivesse habilitado a comunicar por uma rede móvel diminuiria a necessidade da implementação de redes locais em chão de fábrica.
- Os dispositivos utilizados no protótipo para *gateway* poderão comprometer o seu desempenho em alguns ambientes industriais (com ruído elétrico, mecânico, etc., que podem interferir no processamento do *gateway*). A migração para um *gateway* desenvolvido especificamente para aplicações industriais aumentaria a fiabilidade da arquitetura proposta.

As propostas realizadas para possíveis trabalhos futuros demonstram que os desenvolvimentos de soluções desta natureza não possuem um fim, mas sim objetivos que solucionem as atuais (e futuras) necessidades, através de um processo iterativo entre o utilizador e o desenvolvedor.

Referências

[1] *1.4. Differences between AMQP 0-10 and AMQP 1.0*. (25 de janeiro de 2021). Obtido de Red Hat Customer Portal: https://access.redhat.com/documentation/en-us/red_hat_enterprise_mrg/3/html/messaging_programming_reference/differences_between_amqp_0-10_and_amqp_1.0

[2] *A Bosch no mundo*. (10 de junho de 2021). Obtido de Bosch: https://www.bosch.pt/a-nossa-empresa/o-grupo-bosch-no-mundo/

[3] *About Node-Red*. (7 de junho de 2021). Obtido de Node-Red: https://nodered.org/about/

[4] *AMQP 0-9-1 Model Explained*. (25 de janeiro de 2021). Obtido de RabbitMQ: https://www.rabbitmq.com/tutorials/amqp-concepts.html

[5] *An overview of HTTP*. (22 de janeiro de 2021). Obtido de MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[6] Antunes, M., Barraca, J. P., Gomes, D., Oliveiro, P., & Aguiar, R. L. (2015). Smart Cloud of Things: An Evolved IoT. pp. 14-16.

[7] Armendia, M., Ghassempouri, M., Ozturk, E., & Peysson, F. (2019). *Twin-Control.* Springer.

[8] Aydin, G., Hallac, I. R., & Karakus, B. (30 de março de 2015). Architecture and Implementation of a Scalable Sensor Data Storage and Analysis System Using Cloud Computing and Big Data Technologies. *Hindawi*, pp. 6-9.

[9] Bastos, J. P. (20 de abril de 2021). Gestão Remota de Dispositivos.

[10] *Bosch Aveiro*. (10 de junho de 2021). Obtido de Bosch: https://www.bosch.pt/a-nossa-empresa/bosch-em-portugal/aveiro/

[11] *Bosch IoT Suite*. (9 de junho de 2021). Obtido de Huawei Cloud: https://www.huaweicloud.com/en-us/solution/bosch-iot-suite/

[12] *Capabilities of the Bosch IoT Suite*. (22 de janeiro de 2021). Obtido de Bosch: https://developer.bosch-iot-suite.com/capabilities-bosch-iot-suite/

[13] Cawley, C. (10 de dezembro de 2019). *26 Awesome Uses for a Raspberry Pi*. Obtido de Make Use Of: https://www.makeuseof.com/tag/different-uses-raspberry-pi/

[14] *Cloud Servers*. (22 de janeiro de 2021). Obtido de IBM: https://www.ibm.com/cloud/learn/cloud-server

[15] *Different Types of Sensors*. (20 de janeiro de 2021). Obtido de Electronics Hub: https://www.electronicshub.org/different-types-sensors/

[16] Dotis-Georgiou, A. (6 de outubro de 2020). *Why You Should Migrate from SQL to NoSQL for Time Series Data*. Obtido de Jaxenter: https://jaxenter.com/nosql-time-series-data-172822.html

[17] *Eclipse Ditto*. (22 de abril de 2021). Obtido de Eclipse Foundation: https://www.eclipse.org/ditto/

[18] *Eclipse Hono*. (13 de abril de 2021). Obtido de Eclipse Foundation: https://www.eclipse.org/hono/

[19] *Eclipse Kapua*. (6 de junho de 2021). Obtido de Eclipse Foundation: https://www.eclipse.org/kapua/

[20] *ESP32 Product*. (7 de junho de 2021). Obtido de EspressIF: https://www.espressif.com/en/products/socs/esp32

[21] Fernandes, S., Antunes, M., Santiago, A. R., Barraca, J. P., Gomes, D., & Aguiar, R. L. (2020 de Abril de 14). Forecasting Appliances Failures: A Machine-Learning Approach to Predictive Maintenance. *Information*, pp. 3-5.

[22] Hammink, J. (7 de março de 2018). *The Types of Modern Databases*. Obtido de Alooma: https://www.alooma.com/blog/types-of-modern-databases

[23] *Introduction to I²C and SPI protocols*. (22 de janeiro de 2021). Obtido de Byte Paradigm: https://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/?/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

[24] Jost, D. (10 de junho de 2019). *What is a sensor?* Obtido de Fierce Eletronics: https://www.fierceelectronics.com/sensors/what-a-sensor

[25] Liew, Z. (20 de maio de 2021). *Understanding And Using REST APIs*. Obtido de Smashing Magazine: https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/

[26] Liu, C., & Xu, X. (2017). Cyber-physical Machine Tool – The Era of Machine Tool 4.0. *ScienceDirect*, pp. 4-6.

[27] McClelland, C. (25 de janeiro de 2021). *IoT 101: What is a Gateway?* Obtido de IoT for all: https://medium.com/iotforall/iot-101-what-is-a-gateway-be066763b98d

[28] *Modbus FAQ*. (20 de janeiro de 2021). Obtido de Modbus: https://modbus.org/faq.php

[29] *MQTT Essentials*. (25 de janeiro de 2021). Obtido de HiveMQ: https://www.hivemq.com/mqtt-essentials/

[30] *OPC Unified Architecture*. (9 de maio de 2021). Obtido de OPC Foundation: https://opcfoundation.org/about/opc-technologies/opc-ua/

[31] *Overview of ESP32 features. What do they practically mean?* (7 de junho de 2021). Obtido de Explore Embedded: https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F

[32] *Raspberry Pi OS Documentation*. (7 de junho de 2021). Obtido de Raspberry: https://www.raspberrypi.org/documentation/usage/

[33] Romano, M. (4 de agosto de 2017). *Guia Definitivo: Entenda tudo sobre o que é OPC UA e como isso pode impactar a sua indústria*. Obtido de Logique: https://www.logiquesistemas.com.br/blog/opc-ua/

[34] *Rs-485 cable specification guide*. (20 de janeiro de 2021). Obtido de Maxim Integrated: https://www.maximintegrated.com/en/design/technical-documents/tutorials/7/763.html

[35] *Rs485 serial information*. (20 de janeiro de 2021). Obtido de Lammert Bies: https://www.lammertbies.nl/comm/info/rs-485

[36] Smallcombe, M. (19 de maio de 2020). *SQL vs NoSQL: 5 Critical Differences*. Obtido de Xplenty: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/

[37] Stragio, C. E. (20 de janeiro de 2021). *Data Communication Basics*. Obtido de Cami Research: https://www.camiresearch.com/Data_Com_Basics/data_com_tutorial.html

[38] Stragio, C. E. (20 de janeiro de 2021). *The Rs232 Standart*. Obtido de Cami Research: https://www.camiresearch.com/Data_Com_Basics/RS232_standard.html

[39] *Tarifa de Acesso às redes Eletricidade 2021*. (6 de junho de 2021). Obtido de EDP Comercial:https://helpcenter.edp.pt/media/2175/tarifa-de-acesso-as-redes_2021.pdf

[40] Teja, R. (2 de abril de 2021). *What is a Sensor? Different Types of Sensors and their Applications*. Obtido de Electronics Hub: https://www.electronicshub.org/different-types-sensors/

[41] *The Internet of Things with ESP32*. (7 de junho de 2021). Obtido de ESP32: http://esp32.net/

[42] *Time series database (TSDB) explained*. (9 de junho de 2021). Obtido de InfluxData: https://www.influxdata.com/time-series-database/

[43] *WebSockets - A Conceptual Deep Dive*. (17 de maio de 2021). Obtido de Ably: https://ably.com/topic/websockets

[44] *What is a REST API?* (8 de maio de 2020). Obtido de RedHat: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[45] *The Apache Hadoop* (28 de julho de 2021). Obtido de Hadoop: https://hadoop.apache.org/

Apêndices

Apêndice A

Conceitos básicos

Este apêndice tem como objetivo explicar os conceitos básicos utilizados no âmbito do tema desenvolvido, permitindo assim focalizar a leitura do leitor no desenvolvimento do trabalho. Este documento também permite consultar alguma informação desconhecida por parte do leitor.

A.1. Sensores

Um sensor é um dispositivo que permite detetar uma quantidade física e emite-a numa estrutura capaz de ser interpretada por um ser humano ou uma máquina.

A classificação entre sensores varia entre autores, tendo em vista que a definição base varia dependendo da sua área de uso. Contudo, tendo em ponderação o ambiente integrativo deste projeto, é possível subclassificar em passivo ou ativo e analógico ou digital. Sensores ativos são aqueles que necessitam de energia externa para realizar medições, enquanto os sensores passivos funcionam sem alimentação externa. Sensores analógicos emitem um sinal contínuo variável, onde a intensidade deste representa a intensidade da quantidade física detetada, enquanto os sensores digitais emitem dois sinais: um mínimo e um máximo [24] [40].

A.2. Protocolos de Comunicação

A aquisição das propriedades físicas obtidas pelos sensores varia dependendo da sua natureza. Habitualmente, o *output* de um sensor consiste num sinal analógico, onde a intensidade do sinal está relacionada com a propriedade adquirida. Contudo, existem equipamentos que possuem vários sensores integrados, o que possibilita a aquisição e registo das várias propriedades num só dispositivo, assim como usufruir de métodos de comunicação através de sinais digitais. A utilização destes métodos de comunicação permitem diminuir a incidência de erros provocados por fatores externos que iriam corromper a informação proveniente de um sinal analógico. Nos subcapítulos seguintes serão enunciados um conjunto de protocolos que têm como base a comunicação por sinais digitais.

A.2.1. EIA 232

Este protocolo foi proposto nos anos 60 para diminuir as corrupções das mensagens transmitidas por sinais analógicos entre vários dispositivos. Este protocolo veio implementar uma normalização dos métodos de comunicação, estabelecendo várias propriedades físicas necessárias para o sucesso da comunicação. Dependendo da finalidade da comunicação foram implementados dois tipos de cabos: 9 e 25 pinos, sendo que o processo básico para estabelecer a comunicação é análogo: o *Transmitted Data* (TxD) conecta com o *Received Data* (RxD) e o *Request to send* (RTS) conecta com o *Clear to Send* (CTS). Neste protocolo um sinal com voltagem negativa (-3 a -25 V) representa o valor lógico '1' e o positivo (+3 a +25V) '0', relativamente a um pino *ground* comum entre os dois dispositivos. Este protocolo permite a comunicação simultânea entre dois equipamentos, conhecido por *"full-duplex"[38]*.

As mensagens neste protocolo são constituídas por um start bit ('0'), seguidas de 8 bits (1 byte) de *data*, um bit de paridade, se este tiver habilitado, e um stop bit ('1'). O bit de paridade é um bit de segurança que previne a corrupção do sinal enviado. O seu funcionamento consiste em verificar se o somatório dos bits da mensagem a enviar resulta num número par ou ímpar e, se o resultado for diferente da paridade pretendida, este assume o valor de '1'[37].

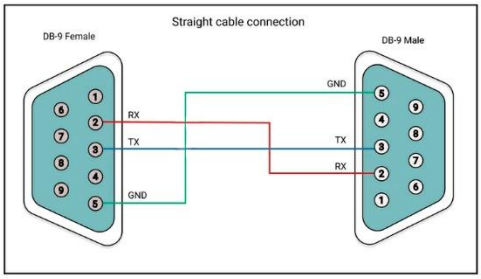

Figura A.1: Exemplo do funcionamento do EIA 232

A.2.2. EIA 485

Este protocolo é uma evolução do EIA 232, com o proveito de melhorar o seu desempenho. Em EIA 485 existem dois pinos para TxD e RxD (ao contrário de apenas um como no EIA 232), entrelaçados entre si, permitindo que a interpretação dos sinais se realize entre os dois pinos do mesmo canal de transmissão em vez do pino e o *ground*. O entrelaçamento dos dois fios permite que a ocorrência de erros na mensagem seja minimizada, uma vez que o efeito de elementos externos é aplicado simultaneamente nos dois fios condutores. Dependendo do comprimento dos fios utilizados, deve-se aplicar uma resistência nos terminais igual à impedância dos fios, assim como uma resistência de *pull up* e de *pull down*. Este protocolo também permite a utilização de uma comunicação *"full-duplex"*, com o recurso a 4 fios[34]. O intervalo de tensão utilizado neste protocolo varia

entre -7 e 12 V e a diferença entre os dois fios tem de ser +- 1,5 V, dependendo se este é um ‘0’ (a tensão em ‘A’ é superior a ‘B’) ou um ‘0’.

Uma vez que este protocolo proporciona a troca de informação em distâncias maiores que o seu antecessor, então possibilita a implementação de uma topologia em barramento, permitindo assim a troca de informação entre vários dispositivos[35].

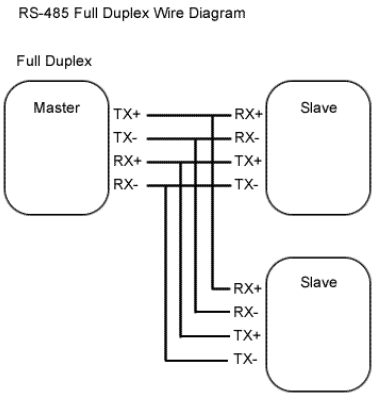

Figura A.2: Exemplo do funcionamento do EIA 485

A.2.3. MODBUS

O MODBUS é um protocolo desenvolvido para facilitar a comunicação em rede com mais de dois dispositivos. Neste protocolo existem dois tipos de dispositivos: o *master*, que organiza o fluxo de informação na rede, e os *slaves,* que executam as instruções provenientes do *master*. Este protocolo permite a utilização de três tipos de formatos: RTU, ASCII e TCP. A sequência entre de mensagem entre RTU e ASCII é muito semelhante: a mensagem começa com uma sequência de bits pré-estabelecida, seguida do endereço do *slave* destinatário, da função a ser executada pelo *slave,* da posição e comprimento da memória para ser acedida, uma verificação da redundância (CrC para RTU e LrC para ASCII) e finaliza com uma sequência de bits pré-estabelecida (silêncio para RTU e CR/LF para ASCII). Após emitida a mensagem, o *master* aguarda a resposta do *slave* pretendido, com a informação desejada [28].

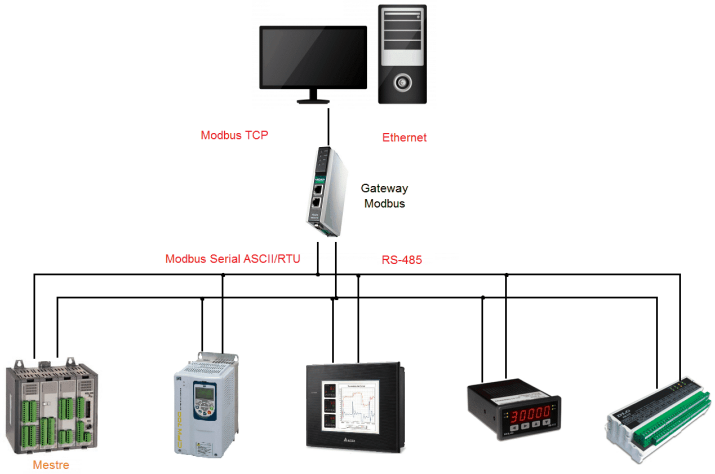

Figura A.3: Exemplo do funcionamento do MODBUS

A.2.4. Serial Peripheral Interface (SPI)

O protocolo SPI foi proposto pela Motorola para estabelecer uma ligação *master/slave* entre dois dispositivos (ou mais, dependendo do número de saídas do *master*). Para realizar a troca de informação, são necessárias 4 ligações: uma de *Serial clock* para sincronizar os dois dispositivos, um *Chip Select* para indicar qual o *slave* a comunicar, um *Master Out-Slave in* e um *Master In-Slave*. A existência de duas linhas distintas para o fluxo de informação permite a implementação de uma comunicação *"full-duplex"*. Neste protocolo, nem o *master* ou o *slave* têm conhecimento da existência do outro (não existe uma função de *acknowledge*), por isso o *master* tem de enviar informação sem utilidade para que o *slave* emita a mensagem a enviar[23].

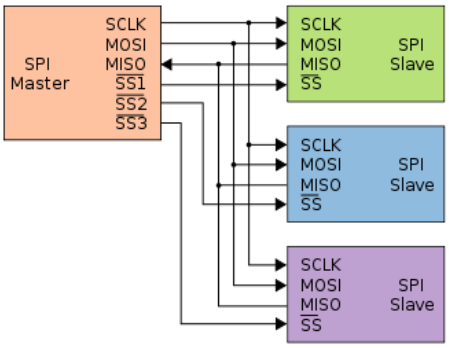

Figura A.4: Exemplo do funcionamento do SPI

A.2.5. I²C

O protocolo I²C estabelece uma ligação *master/slave* entre vários dispositivos, utilizando apenas dois fios: um *Serial Data* (SDA) e um *Serial Clock* (SCL), além da necessidade da implementação de duas resistências de *pull-up* para funcionarem. Este

protocolo permite aos *masters* enviarem e receberem simultaneamente, contudo se dois *masters* enviarem ao mesmo tempo informação para o barramento, o *master* que estiver a escrever um '1' é interrompido e espera algum tempo antes de reenviar a mensagem. Um *master* apenas envia uma nova mensagem após detetar uma condição de *STOP*. Quando é enviada uma condição de *START*, os *slaves* verificam se o endereço na mensagem é o deles e se este não corresponder, vão suspender a leitura de mensagem enquanto não aparecer uma condição de *STOP,* onde voltam a escutar o barramento. Esta ligação permite o reconhecimento entre dispositivos, uma vez que após a receção de 8 bits de dados, o *slave* força a SDA a '0' para dar o *acknowledge* da mensagem ao *master[23]*.

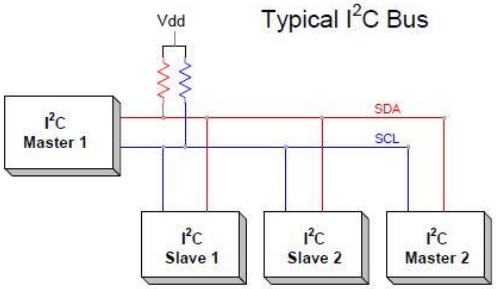

Figura A.5: Exemplo do funcionamento do I²C

A.2.6. OPC UA

Open Platform Communicarions Unified Architecture, ou OPC UA, corresponde à evolução do seu antecessor, OPC *Classic*. O protocolo OPC *Classic* foi proposto no final da década de 90 com o objetivo de padronizar os protocolos de comunicação utilizados em chão de fábrica, uma vez que cada fornecedor implementava o seu próprio protocolo nos seus dispositivos. O OPC *Classic* possui três especificações: OPC DA (*Data Access)*, que define os processos das trocas de dados entre dispositivos, OPC A&E (*Alarm and events*), que define o processo de troca de mensages relativas a eventos e alarmes, assim como o estado das variáveis de monitorização, e OPC HDA (*Historical Data Access*), que permite a consulta e análise dos dados em histórico. Com a evolução do mercado, começaram a aparecer insatisfações relativamente a algumas limitações deste protocolo, nomeadamente a sua dependência com o sistema operativo *Windows*, assim nasceu o OPC UA como um protocolo de comunicação entre diferentes sistemas operativos.

O protocolo OPC UA especifica a utilização de dois programas: um OPC *Server* e um OPC *Client*, compatíveis com uma API padrão. Este desacoplamento permite a existência de uma maior flexibilidade e compatibilidade, uma vez que o cliente não necessita de conhecer as especificações do servidor. Este protocolo suporta dois formatos de mensagens: UA *Binary*, utilizado principalmente em comunicações de baixo nível, devido à reduzida capacidade computacional dos dispositivos, e XML, utilizado em equipamentos com maior capacidade de processamento. Tendo em consideração os formatos de mensagens utilizados, existem dois protocolos para o transporte de mensagens: OPC TCP,

específico dos clientes OPC UA, utiliza a formatação das mensagens UA *Binary* e transmite-as através de um *socket*, e SOAP/HTTP(S), que utiliza a formatação XML e o protocolo HTTP(S) para o transporte[33].

Atualmente também já é possível realizar a troca de informação através de um mecanismo de Publicação-Subscrição, semelhante aos protocolos MQTT e AMQP[30].

A.2.7. HTTP

Hypertext Transfer Protocol (ou HTTP) é um protocolo de comunicação cliente-servidor, utilizado para transferir informação na *Web*, principalmente ficheiros de texto, áudio e vídeo. Neste protocolo, o pivot da comunicação é o cliente (usualmente um *Browser*) que envia mensagens *requests* a um servidor, que retorna mensagens *responses[5]*.

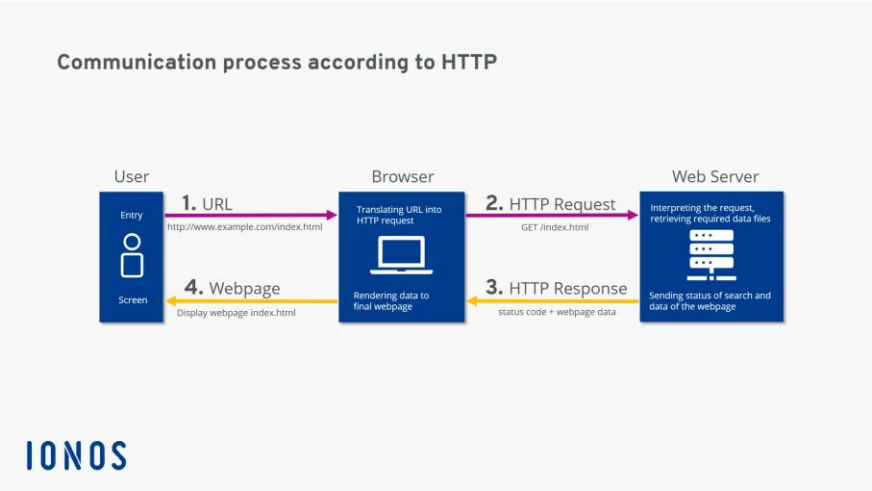

Figura A.6: Exemplo de mensagem HTTP

A.2.8. WebSocket

Protocolo desenvolvido para realizar comunicações em tempo real a partir da internet, anteriormente realizadas por *HTTP*. A sua utilização permite eliminar a utilização constante de *HTTP requests* para verificar se existem atualizações no servidor, atividade que comprometia a sua eficiência. A comunicação com *WebSockets* inicia-se por *HTTP request*, seguida do *acknowledge* por parte do servidor, estabelecendo uma comunicação *full-duplex* persistente entre as duas entidades [43].

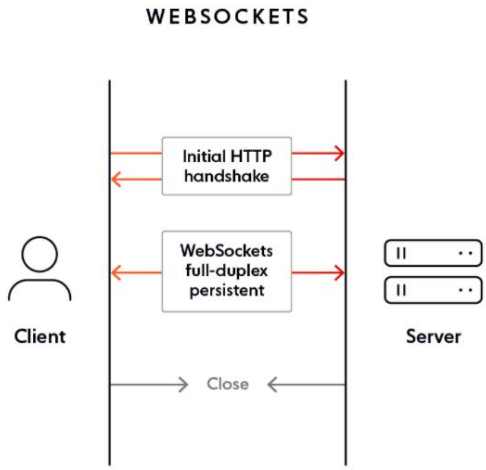

Figura A.7: Exemplo da comunicação por WebSocket

A.2.9. MQTT

Message Queuing Telemetry Transport (ou MQTT) é um protocolo de comunicação cliente-servidor simples, leve e fácil de implementar, utilizado para pequenas trocas de informação entre dispositivos. A utilização deste protoloco consiste na subscrição e publicação de tópicos previamente criados no *broker*, permitindo a troca de mensagens entre os clientes e o *broker[29]*.

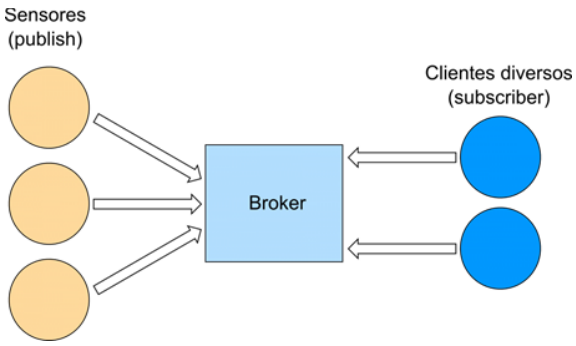

Figura A.8: Exemplo da comunicação por MQTT

A.2.10. AMQP

Advanced Message Queuing Protocol (ou AMQP) é um protocolo de comunicação semelhante ao MQTT. Neste protocolo, as mensagens são publicadas para um *Exchange* (que reside num *broker)*, que funciona como uma caixa postal, e são distribuídas em *queues*. A estas *queues* podem ser enviadas para um consumidor que a subscreveu (como em MQTT) ou podem residir no *broker* até serem requeridas por um consumidor (Figura

A.9)[4]. A última atualização do protocolo (AMQP 1.0) permitiu a implantação deste sem recurso a um *broker*, sendo apenas especificado a utilização de um *broker* nos protocolos AMQP 0-10[1].

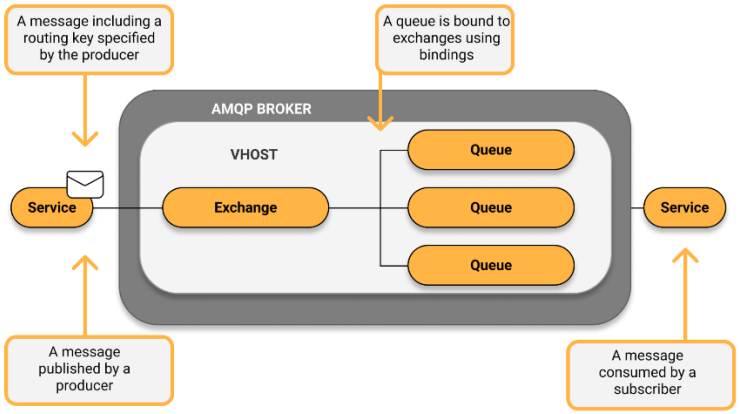

Figura A.9: Exemplo da comunicação por AMQP

A.3. Latência

Numa aplicação real, o impacto da latência é um fator importante em sistemas que realizem várias medições num curto período. Por isso, a comparação da latência entre vários protocolos é um fator de decisão no método a utilizar. Tendo como base a análise realizada no artigo *"Smart Cloud of Things: An Evolved IoT Plataform for Telco Provider" [6]*, onde foram realizados vários ensaios relativamente ao envio de mensagens com diferentes comprimentos através de diferentes protocolos, utilizado a plataforma ***SCoT***. Nos resultados é possível observar que o MQTT é, dos protocolos utilizados, aquele que além de possuir a menor latência, também apresenta a menor variância nos seus resultados, permitindo assim afirmar que este protocolo é uma solução adequada para problemas que necessitem uma elevada taxa de aquisição de informação.

Apêndice B

Comunicação Equipamento-Gateway

Nos apêndices seguintes serão apresentados alguns exemplos genéricos que permitem demonstrar o desenvolvimento e aplicação das diversas tecnologias utilizadas na formulação da arquitetura proposta.

Neste apêndice será demonstrado um exemplo de um programa desenvolvido em NodeRed para um Raspberry Pi, onde é possível comunicar com um analisador de energia (Janitza UMG 103) para a extração de dados relativos ao circuito elétrico inserido.

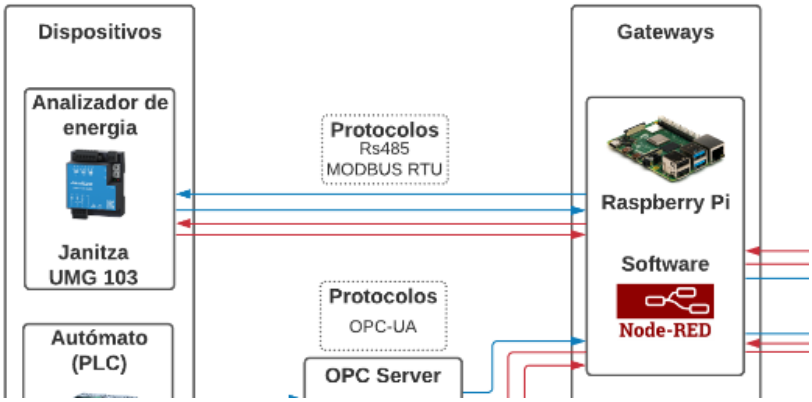

Figura B.1: Componentes da Arquitetura Proposta que englobam o código em exemplo

Na Figura B.1 é possível observar o analisador de energia (à esquerda), o *gateway* (à direita) e o protocolo de comunicação utilizado entre os dois equipamentos.

B.1. Instalação elétrica

Na figura seguinte é possível observar o circuito elétrico utilizado na instalação dos vários componentes necessários. Os componentes utilizados foram os seguintes:

- 1 Janitza UMG 103;
- 1 Raspberry Pi 4 B 4GB (com os devidos utensílios);
- 1 base de 3 tomadas;
- 1 adaptador USB-Rs485;
- 1 adaptador de tomada (macho).

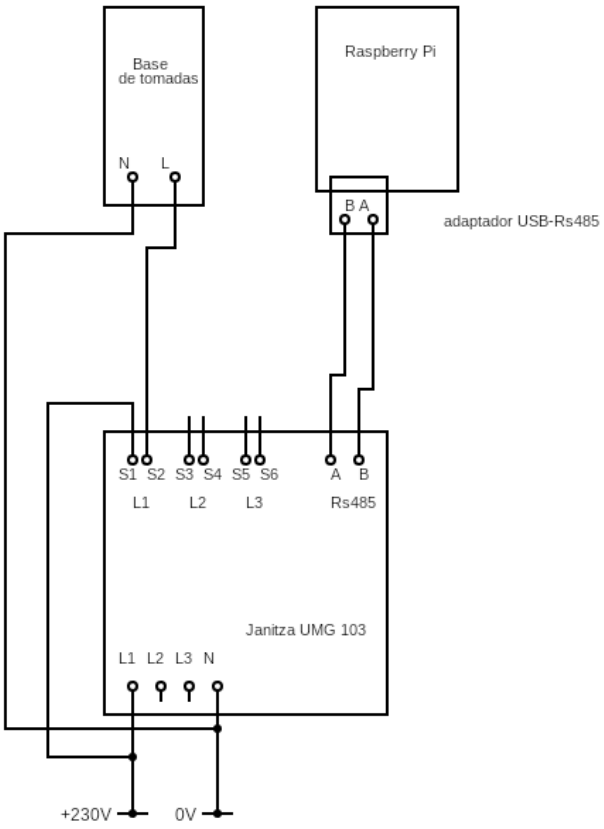

Figura B.2: Diagrama da instalação do Janitza

Na figura B.2 é possível observar o esquema da instalação utilizada. O analisador é alimentado em L1 (fase) e em N (neutro). Em S1 coloca-se a fase proveniente da rede elétrica e em S2 a fase de alimentação do equipamento fabril. A comunicação Rs485 é implementada entre o analisador e o Raspberry, com o recurso de um adaptador USB-Rs485 ligado nos pinos A e B.

É preciso salientar que neste exemplo realizou-se a instalação da fase1, uma vez que os equipamentos utilizados para os testes eram monofásicos. Outro fator a considerar é a corrente máxima suportada pelo analisador, neste caso é de 5 amperes. Se existir a necessidade de utilizar correstes mais elevadas, é necessário instalar um transformador de corrente (TC) (Figura B.3) para não danificar o aparelho.

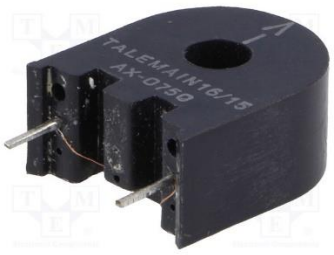

Figura B.3: Exemplo de um TC

B.2. Comunicação Rs485

A comunicação com o analisador de energia selecionado é efetuada por MODBUS Rs485. Por esta razão foi instalado um adaptador USB – Rs485 no Raspberry Pi. A nível de *software* foi utilizado o node-red, com a biblioteca 'node-red-contrib-modbus' para desenvolver o código e a configuração dos parâmetros da mensagem em MODBUS entre os dois dispositivos.

Para configurar o node-red são necessários os seguintes blocos:

- 1 inject para iniciar a mensagem;
- 1 Modbus Getter para configurar a mensagem;
- 1 debug para visualizar o resultado.

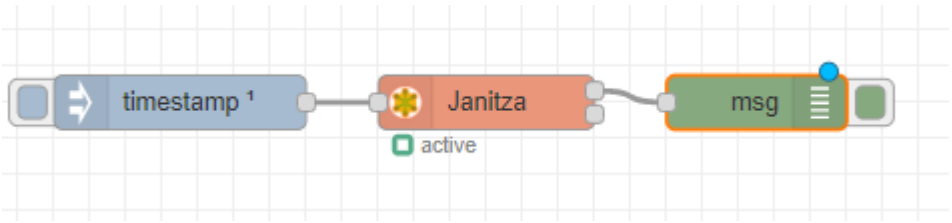

Figura B.4: Nós utilizados para desenvolver o programa

Os parâmetros da configuração da mensagem, assim como do endereço de memória de cada registo apresentam-se no *datasheet* do equipamento selecionado. A configuração implementada pode ser visualizada nas imagens seguintes.

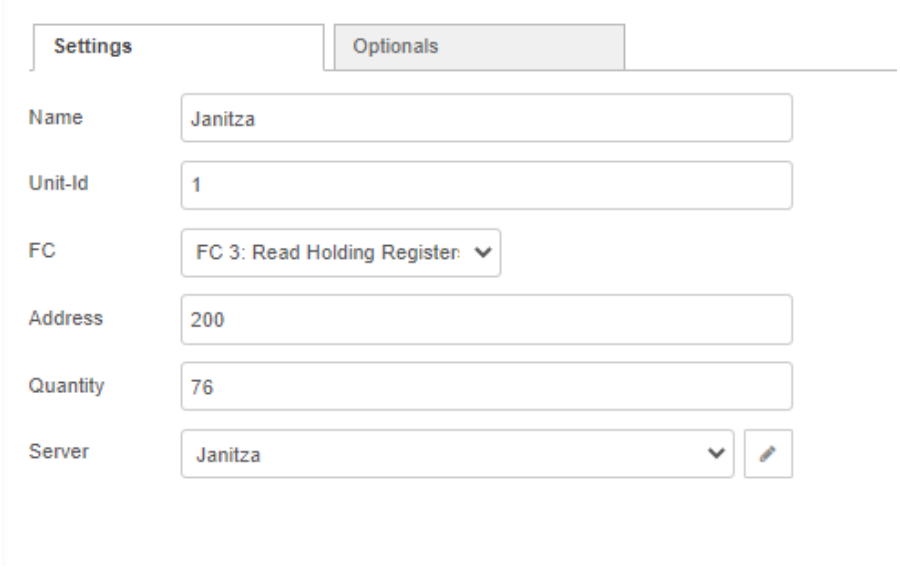

Figura B.5: Configuração da mensagem MODBUS

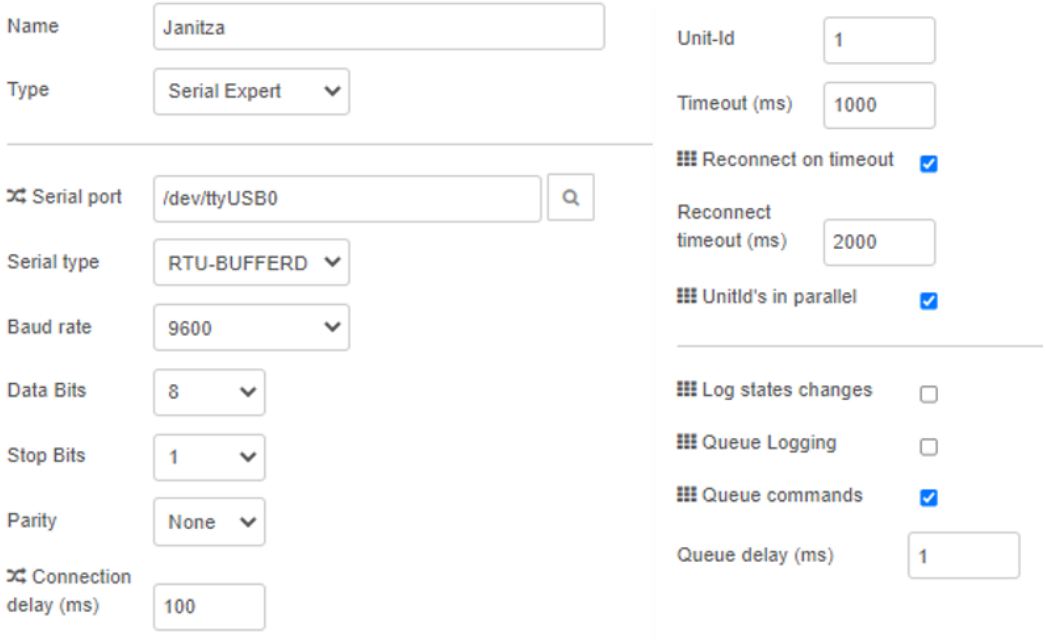

Figura B.6: Configuração da comunicação Rs485

Na Figura B.5 realiza-se a configuração da mensagem MODBUS RTU a enviar ao *slave* (01), onde se realiza o pedido de registos (03) desde o endereço de memória 200 até ao endereço 275 (75+1). A Figura B.6 representa a configuração da comunicação Rs485. Estes parâmetros encontram-se no *datasheet* do equipamento instalado.

Apêndice C

Comunicação Gateway-Dashboard

Neste apêndice serão apresentados alguns exemplos genéricos dos programas utilizados para estabelecer uma ligação entre o *gateway* e o *dashboard* para a visualização dos dados recolhidos. As etapas necessárias para estabelecer esta ligação são as seguintes:

- Publicar a informação recolhida para o *broker*. Neste exemplo será utilizado o SCoT como *broker* e o *NodeRed* para estabelecer a ligação.
- Receber a informação do broker e injetar numa base de dados. Neste exemplo será utilizado o *InfluxDB*.
- Configurar o *dashboard* para aceder à base de dados. Neste exemplo será utilizado o *Grafana*.

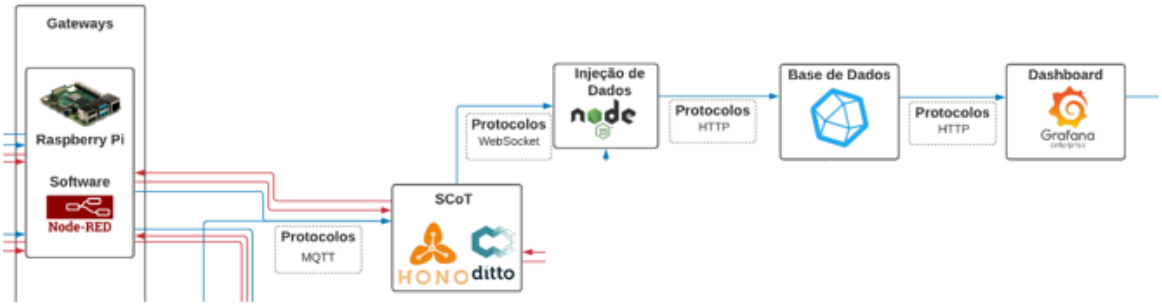

Figura C.1: Componentes da Arquitetura Proposta que englobam o código em exemplo

Na Figura C.1 é possível observar o *gateway* (à esquerda) e o protocolo utilizado no envio de mensagens (*MQTT*), a plataforma *IoT* (SCoT) que recebe as mensagens e as reencaminha para um programa em *Node.js*, por WebSocket, que adiciona a informação recebida à Base de Dados (*InfluxDB*). À direita é possível observar o *dashboard* (*Grafana*) que realiza um pedido de informação à Base de Dados para a dispor numa interface gráfica.

C.1. Publicação da informação

Este exemplo corresponde à continuação do programa desenvolvido no apêndice anterior. Uma vez estabelecida a ligação entre o analisador de energia *Janitza* e o *raspberry*,

segue-se a formatação e o envio dessa informação. Os blocos necessários para esta atividade são os seguintes:

- 1 *funtion* para formatar a mensagem;
- 1 *mqtt out* para publicar a informação;
- 1 *debug* para visualizar a mensagem formatada (opcional).

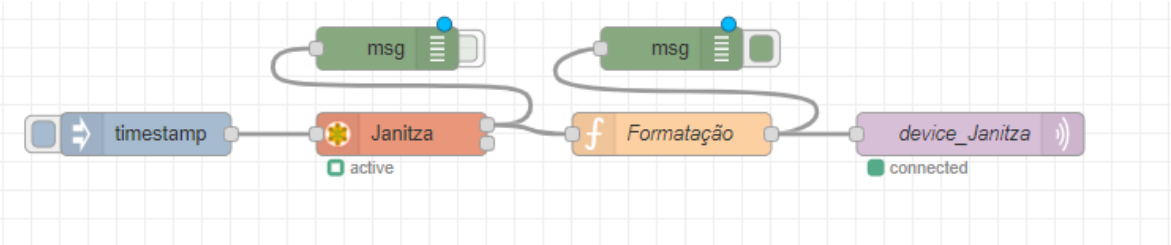

Figura C.2: Nós utilizados para desenvolver o programa

C.1.1. Formatação da mensagem

A mensagem enviada necessita de possuir a seguinte estrutura para que o SCoT a consiga interpretar:

```
{ topic : 'telemetry',
  data: <data em formato JSON>,
  qos: 0}
```

Na imagem seguinte é possível visualizar o código utilizado na formatação da mensagem:

Figura C.3: Código desenvolvido na função

Neste exemplo procedeu-se à seleção e tratamento das variáveis de interesse provenientes da mensagem MODBUS RTU.

C.1.2. Ligação ao *broker*

A comunicação efetuada entre o *gateway* e o *broker* (SCoT) é com base no protocolo MQTT. Para a sua conceção são necessários os seguintes parâmetros para que o *broker* reconheça o *gateway*:

- Tenant_id;
- Device_id;
- Device_password.

Nas imagens seguintes serão demonstradas as várias etapas para corretamente configurar o bloco MQTT.

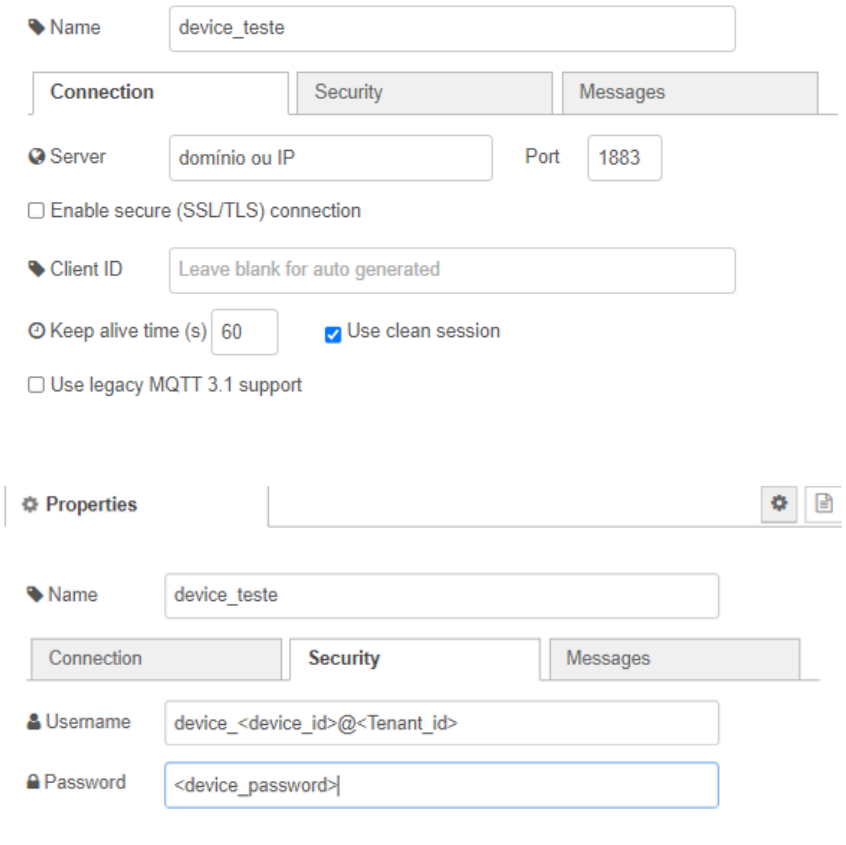

Figura C.4: Configuração do broker MQTT

Uma vez estabelecida a publicação de mensagens para o *broker*, é possível proceder para a aquisição destas num outro dispositivo, nomeadamente o servidor onde ficarão alocados a base de dados e o *dashboard*.

C.2. Aquisição e injeção dos dados numa base de dados

Desta etapa em diante será realizada a transposição para os serviços a ser executados por um servidor, onde serão alocados todos os programas necessários para a implementação da arquitetura proposta. Os exemplos seguintes serão desenvolvidos em *Node.js*.

Para a realização deste serviço foram instaladas as seguintes librarias:

- Nodemon: permite reexecutar o serviço após a gravação do mesmo;
- Ws: para estabelecer uma comunicação *Websocket* entre o *Node.js* e o *broker*;
- Influx: para facilitar a injeção dos dados na base de dados através da sua API.

Os parâmetros do *broker* necessários são os seguintes:

- Tenant_id;
- Tenant_password.

Também é necessário já existir uma base de dados ('energia') previamente criada no *InfluxDB*.

C.2.1. Código

```
// Importa a libraria 'ws'
const WebSocket = require('ws');

// Estabelece comunicação WS com o SCoT
const ws = new WebSocket('ws://<Tenant_id>:<Tenant_pass>@<domain>/ws/2');

// Envia uma mensagem para pedir os eventos recebidos
ws.on('open', function open() {
  ws.send('START-SEND-EVENTS?extraFields=attributes/counter,features/ConnectionStatus');
});

// Importa e configura a comunicação com a BD
const Influx = require('influx');
const influx = new Influx.InfluxDB('http://localhost:8086');

// Caso receba uma mensagem…
ws.on('message', function incoming(data) {
  try {
      // Verifica se está no formato JSON
      var value = JSON.parse(data)
       // Isola os nomes dos parâmetros
      var iso_topics = Object.keys(value["value"]);

      // Formatação da informação para a BD
      var toinject = "{";
      for (i=0; i<Object.keys(iso_topics).length; i++)
      {if (i==0){
          toinject = toinject + "\"" + iso_topics[i]+ "\"" + ": " + value["value"][iso_t
opics[i]]["properties"]["value"];
        }
        else{
          toinject = toinject + ", \"" + iso_topics[i]+ "\"" + ": " + value["value"][iso
_topics[i]]["properties"]["value"];
```

```
        }
      }

      toinject = toinject + "}"

       // Escrever na BD
      influx.writePoints([
        {
           // Nome do device
          measurement: value["headers"]["device_id"],
           // Dados
          fields: JSON.parse(toinject),
           // Tempo
          timestamp: new Date(value["timestamp"]).getTime(),
        }
      ], {
         // Seleção da BD
        database: 'energia',
        precision: 'ms',
      })
      .catch(error => {
        console.error(`Error saving data to InfluxDB! ${err.stack}`)
      });
  }
  catch(error){
    // Erro comum após o pedido dos eventos
    console.error(error);
  }
});
```

Na figura seguinte é possível observar o resultado pretendido (Figura C.3) obter da variável *toinject*, que contém a informação a inserir na base de dados.

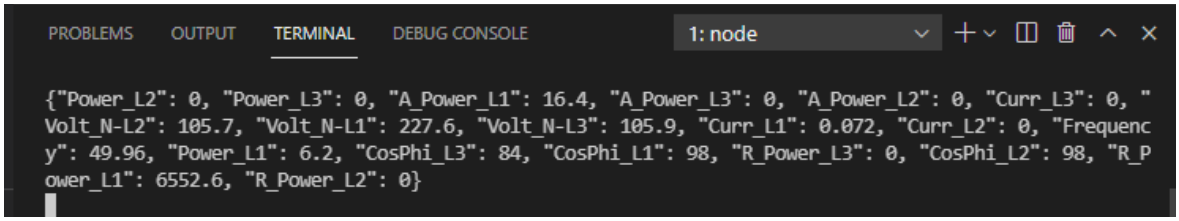

Figura C.5: Exemplo de uma mensagem recebida

C.3. Visualização da informação no *dashboard*

Uma vez estabelecida a ligação entre o *broker* e a base de dados, só falta estabelecer a ligação entre a base de dados e o *dashboard*. A utilização do *Grafana* facilita a implementação desta ponte, uma vez que esta possui várias APIs para comunicar com diversas bases de dados, sendo uma delas o *InfluxDB*.

C.3.1. Configuração de uma base de dados

Antes da criação de um *template* para visualizar os dados é preciso configurar a fonte dos dados.

1) Selecionar a base de dados (*InfluxDB*);

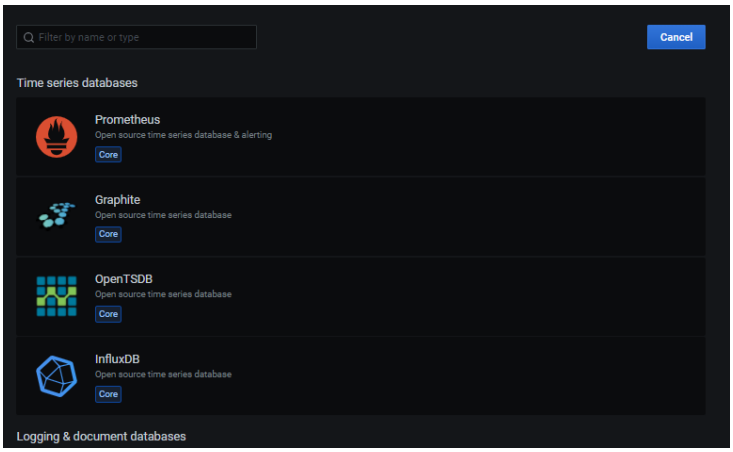

Figura C.6: Etapa 1 na configuração do dashboard

O *Grafana* possui uma API que facilita na interação com um conjunto de Bases de Dados diferentes. Neste exemplo utilizou-se o *InfluxDB*, contudo é possível utilizar o *Grafana* com outras Bases de Dados simultaneamente.

2) Configurar os parâmetros da base de dados e gravar;

A configuração da Base de Dados no *Grafana* depende dos parâmetros estabelecidos na sua criação. Neste caso, a configuração foi a seguinte:

- *Query Language:* InfluxQL
- *URL:* http://localhost:8086
- *Database:* energia
- *HTTP Method:* GET

Esta configuração encontrasse na figura seguinte.

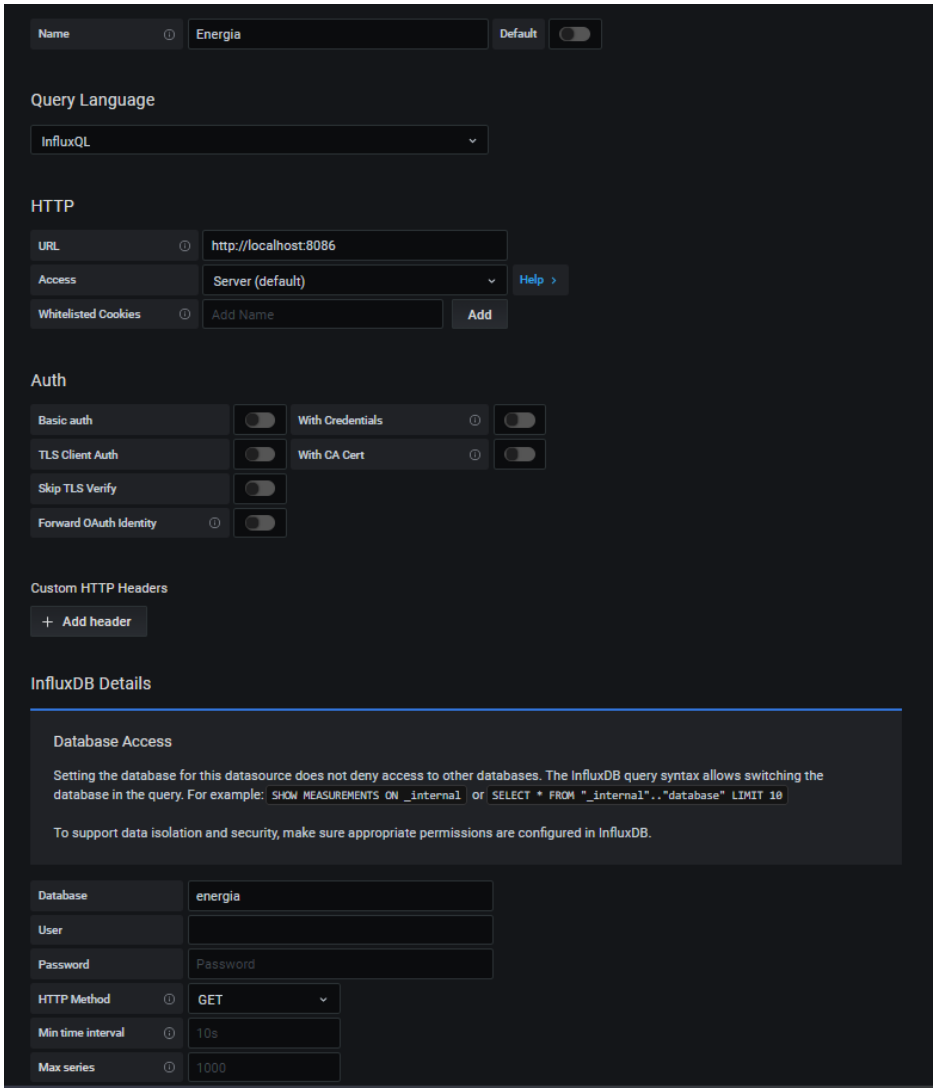

Figura C.7: Etapa 2 na configuração do dashboard

3) Criar um *dashboard* e adicionar um *template*.

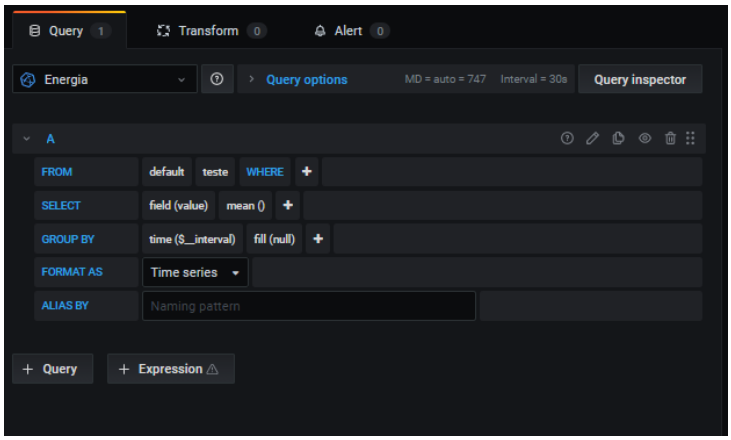

Figura C.8: Etapa 3 na configuração do dashboard

Como é possível observar nas imagens anteriores, a API do *Grafana* facilita na criação de *queries* com a base de dados, mas também permite escrever a *query* manualmente. Após algumas configurações é possível obter uma *dashboard* semelhante à seguinte.

Figura C.9: Exemplo de uma configuração display de um template

Esta figura permite visualizar o funcionamento *'live'* do equipamento, assim como o histórico de informação perante o intervalo de tempo selecionado.

Apêndice D

Serviços API

Uma vez implementadas as etapas necessárias para estabelecer a sequência de operações entre a aquisição de informação à sua visualização, ser-se-ão apresentados dois exemplos de serviços básicos na interação com a arquitetura proposta:

- Extração de informação proveniente na base de dados para múltiplos afins;
- Envio de mensagens para os dispositivos em chão de fábrica.

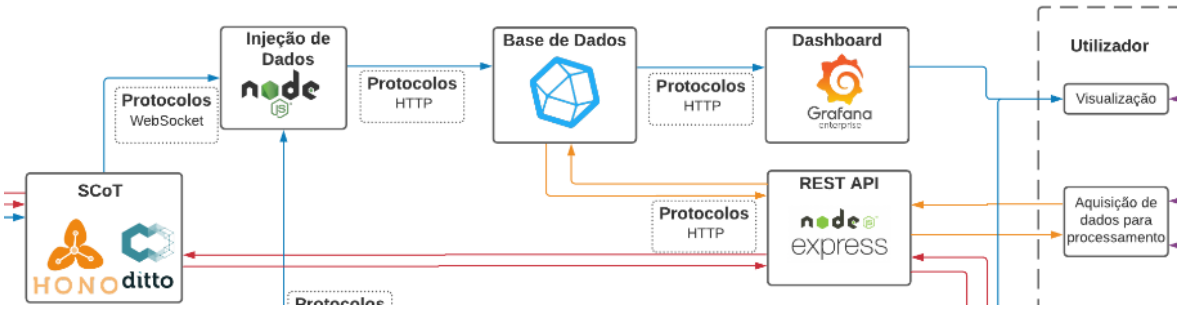

Figura D.1: Componentes da Arquitetura Proposta que englobam o código em exemplo

Na Figura D.1 é a evolução da Figura C.1, onde se adicionou um programa em *Express.js* que fornece serviços API REST entre o utilizador e a plataforma, nomeadamente o pedido de informação à Base de Dados e o envio de mensagens de comando aos equipamentos.

D.1. Extração de dados

O exemplo que se segue tem como base um serviço *API REST* que formata o pedido do utilizador numa *query* para a base de dados. Este serviço fornecerá uma aplicação fácil por parte do utilizador, pois a comunicação desta etapa em adiante torna-se agnóstica.

Para a realização deste exemplo foram utilizadas as seguintes bibliotecas:

- **Nodemon:** permite reexecutar o serviço após a gravação do mesmo;
- **Express:** biblioteca de *Node.js* utilizada para criar serviços API;
- **Axios:** facilita a implementação de pedidos *HTTP*;
- **Backlash:** facilita a atualização de variáveis.

Numa perspetiva de organização, o programa foi subdividido em quatro ficheiros:

- **API.js:** corpo principal do programa onde se estabelece a sua configuração no sistema;

- **Devices.js:** conjunto de todas as opções de interatividade entre o utilizador e o programa;
- **Func.js:** onde se encontram as funções auxiliares para o desenvolvimento do serviço;
- **Help.html:** documentação da API.

D.1.1.1. API.js

```
// importação das bibliotecas e dos ficheiros
import express from 'express';
import bodyParser from 'body-parser';
import devicesRoutes from './routes/devices.js';
import path from 'path';

// implementação da API na porta selecionada
const app = express();
const PORT = 8080;
const __dirname = path.resolve();

// Admite ficheiros JSON
app.use(bodyParser.json());

// endereço do ficheiro 'devices'
app.use('/api/devices, devicesRoutes);

// documentação
app.get('/api/help', (req, res) => {
    res.sendFile(path.join(__dirname + '/HTML_files/Help.html'))
});

// mensagem com a localização do serviço
app.listen(PORT, () => console.log(`Server Running on port: http://localhost:${PORT}`));
```

Primeiro ficheiro da API, onde se encontram os endereços das restantes funcionalidades desenvolvidas e da documentação.

D.1.1.2. Devices.js

```
// importação das bibliotecas e dos ficheiros
import express from 'express';
import {AcessDevice, AcessDB, send_command} from "../functions/func";
```

```
// informa que é um documento 'express'
const router = express.Router();

// parâmetros da base de dados
const host = 'http://localhost:8086';
const database = 'energia';

// Pede todas as medições existentes na base de dados selecionada
router.get('/', (req, res) => {
    var query = 'SHOW MEASUREMENTS ON ' + database;
    AcessDB(host,query,res);
});

// Pede todas as medições existentes numa medição da base de dados selecionada
router.get('/data/:device/all', (req, res) => {
    var query = 'SELECT * FROM ' + req.params.device
    AcessDevice(host,database,query,res);
});

export default router;
```

Este exemplo possui duas interatividades: visualizar todas as medições (tabelas) na Base de Dados 'energia' e a seleção de todo os registos numa dessas medições. Estas funcionalidades apresentadas servem apenas de exemplo, mas constituem a base de interações mais complexas.

D.1.1.3. Func.js

```
// impostação das bibliotecas e dos ficheiros
import { createRequire } from 'module';
const require = createRequire(import.meta.url);
const axios = require('axios').default;
import _, {map} from 'underscore';

// função para interagir com as medições
export function AcessDevice(host,database,query,res){
    // configuração do headers
    const option ={
        headers:{
```

```
            Accept: 'application/csv',
            'Content-type': 'application/vnd.flux'
        }
    }
    // Pedido HTTP GET a uma medição da base de dados
    axios.get(host + '/query?db=' + database + '&q=' + query,option)
        .then(response => {
            const aux = csvToJson(response.data);
            res.send(aux)
            //console.log(aux)
        })
        .catch(error => {
            //console.log(error)
            res.send(error)
        })
    return
}

// função para interagir com a base de dados
export function AcessDB(host,query,res){
    // configuração do headers
    const option ={
        headers:{
            Accept: 'application/csv',
            'Content-type': 'application/vnd.flux'
        }
    }
    // Pedido HTTP GET à base de dados
    axios.get(host + `/query?q=` + query,option)
        .then(response => {
            const aux = csvToJson(response.data);
            res.send(aux)
            //console.log(aux)
        })
        .catch(error => {
            //console.log(error)
            res.send(error)
        })
    return
}
```

```
// função que formata a informação .csv em JSON
function csvToJson(csv) {
    const content = csv.split('\n');
    const header = content[0].split(',');
    return _.tail(content).map((row) => {
        return _.object(header, row.split(','));
      });
  }
```

Ficheiro criado para simplificar o desenvolvimento das funcionalidades da API. Aqui encontram-se funções auxiliares para interagir com a API da Base de Dados, nomeadamente com as medições realizadas e a *Shell* da mesma. Também existe uma função que transforma a informação de .csv (da API da Base de Dados) para JSON.

D.1.1.4. Help.html

```
<header>
    <b>Welcome to my API documentation!</b><br><br>
</header>
<body>
    <b>Existing devices:</b><br> :domain/api/devices <br><br>
    <b>Access all data on measurement:</b><br>:domain/api/devices/data/:device/all<br><br>
</body>
```

Este ficheiro HTML apresenta as funcionalidades criadas no documento *devices.js* utilizado no código exemplo.

D.2. Envio de mensagens

A implementação de mensagens entre o utilizador e os *devices* requere o desenvolvimento de duas funcionalidades: uma no *gateway* do *device* e a outra na API previamente desenvolvida. Nos exemplos que se seguem, esta implementação foi efetuada nos programas previamente desenvolvidos, mas a sua autonomia como um programa *standalone* também é possível.

D.2.1. Implementação na API

D.2.1.1. Func.js

```
// função para enviar mensagens
export function send_command(device,state,res){
    // credenciais
    var domain = "<domínio>";
    var tenant = '<tenant_id>;
    var pass = '<tenant_password>';

    const option ={
        // configuração do headers
        headers:{
              // random string
            'correlation-id': Math.random().toString(36).substring(4),
              // tempo de espera pela resposta
            'timeout':5,
              // autorização
            "Authorization": "Basic " + new Buffer(tenant+':'+pass).toString("base64")}
    }
    // Envio da mensagem por HTTP POST
    axios.post(domain + '/api/2/things/' + tenant + ':' + device +'/inbox/messages/comma
nd',state,option)
        .then(function (response) {
            //console.log(response.data)
            res.send(response.data)

      })
        .catch(function (error) {
        //console.log(error);
        res.send(error)
      })
    return
}
```

Função a adicionar ao ficheiro *Func.js* que facilita o envio de mensagens para o *broker*.

D.2.1.2. Devices.js

```
// Enviar uma mensagem para um dispositivo
router.get('/command/:device/:state', (req, res) => {
    var device = req.params.device;
    var state = req.params.state;
    send_command(device,state,res);
});
```

Expansão das funcionalidades da API desenvolvida para fornecer o serviço de mensagens de comando com os dispositivos.

D.2.2. Implementação no *gateway*

O exemplo seguinte corresponde ao programa aplicado no *gateway*, que permite obter o valor de um registo do analisador de energia.

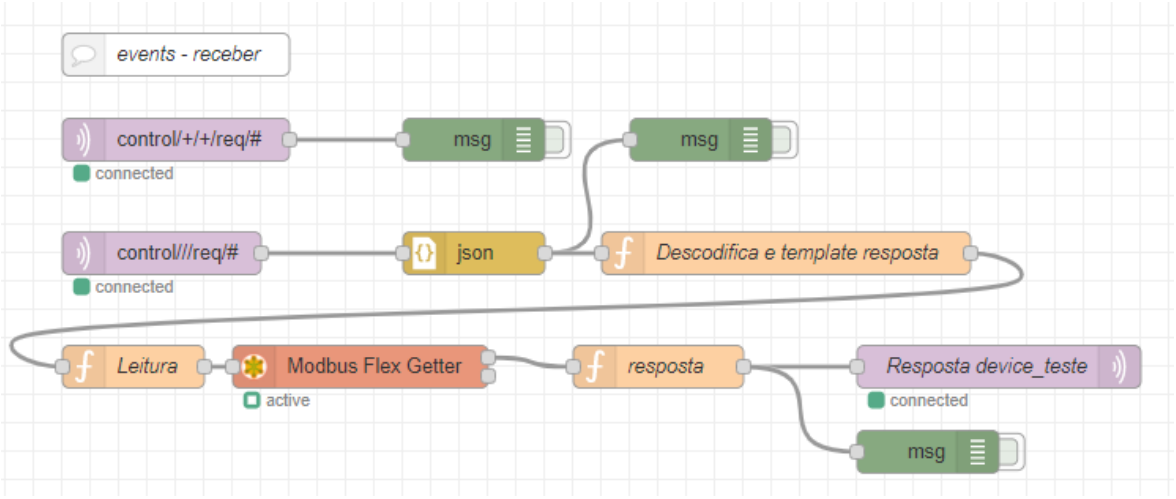

Figura D.2: Nós utilizados para desenvolver o programa

Para subscrever ao *broker* MQTT, em node-red, têm de ser aplicados dois blocos '*mqtt in*': um para o tópico 'control/+/+/req#' e outro para 'control///req#'. No caso de outras linguagens (por exemplo C ou python), é possível subscrever apenas ao primeiro tópico.

D.2.2.1. Descodifica e template resposta

```
// descodifica base64
let buff = new Buffer(msg.payload.value, "base64");
msg.payload.value = buff.toString('ascii');

// resposta genérica
var id = msg.topic.split('/')[4];
var corr = msg.payload.headers["correlation-id"];
var resp = {
        'topic' : 'control///res/' + id + '/200',
        'payload':{
            'topic' : msg.payload.topic,
            'headers' : {'content-type': 'application/json',
            'correlation-id': corr },
            'path':'/inbox/messages/command',
            'value' : {},
            'status': 200,
            'qos': 0}
};

flow.set("resposta",resp);

return msg;
```

Figura D.3: Template da mensagem de resposta

Na figura anterior é possível observar a descodificação da mensagem, em base64, e a estrutura da mensagem a enviar para o *broker*.

D.2.2.2. Leitura

```
msg.payload = {
    'fc': 3,
    'unitid': 1,
    'address': msg.payload.value,
    'quantity': 1
};
return msg;
```

Figura D.4: Função para realizar a leitura do analisador

Nesta figura é possível visualizar a configuração da mensagem *MODBUS RTU* para adquirir a informação na memoria do endereço pedido.

D.2.2.3. Resposta

```
let msg2 = flow.get("resposta");
msg2.payload.value = msg.payload;
return msg2;
```

Figura D.5: Adiciona o valor obtido ao Template

Nesta figura é adicionado o valor do registo recebido no corpo da mensagem a enviar para o *broker*.

Apêndice E

Comunicação OPC-UA

O apêndice que se segue apresenta a metodologia para estabelecer uma comunicação OPC-UA entre um PLC (Siemens S7-1200) e um *gateway* recorrendo a um módulo OPC-UA server (IBH Link UA). Durante a realização do exemplo utilizaram-se as seguintes versões de *firmware*:

- Siemens S7-1200: V4.1.1;
- IBG Link UA: V5.20.

Se o *PLC Siemens S7-1200* utilizado possuir a versão de *firmware* V4.4 instalada e o *TIA Portal* o *add-on Openness* instalado, é possível utilizar o editor OPC-UA fornecido pela IBH, pois esta versão permite facilmente configurar o servidor OPC-UA do PLC.

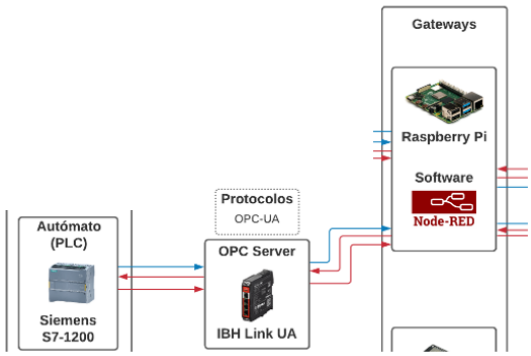

Figura E.1: Componentes da Arquitetura Proposta que englobam o código em exemplo

Na Figura E.1 realiza-se a implementação da comunicação entre o PLC S7-1200 (à esquerda), o módulo IBH Link UA (centro) e o Raspberry Pi (*gateway*, à direita). O módulo realiza, ciclicamente, o mapeamento das variáveis no PLC, que posteriormente serão enviadas para o Raspberry quando este realizar um pedido por OPC-UA.

E.1. Programação do PLC

O programa exemplo seguinte apenas pretende demonstrar a utilização de uma comunicação OPC-UA entre um PLC e o seu *gateway*. Assim, o programa apenas permite realizar duas funções:

- Ativar/desativar uma entrada que liga/desliga uma saída;
- Ativar /desativar uma memória que liga/desliga outra saída.

Este exemplo foi realizado em TIA Portal V15.

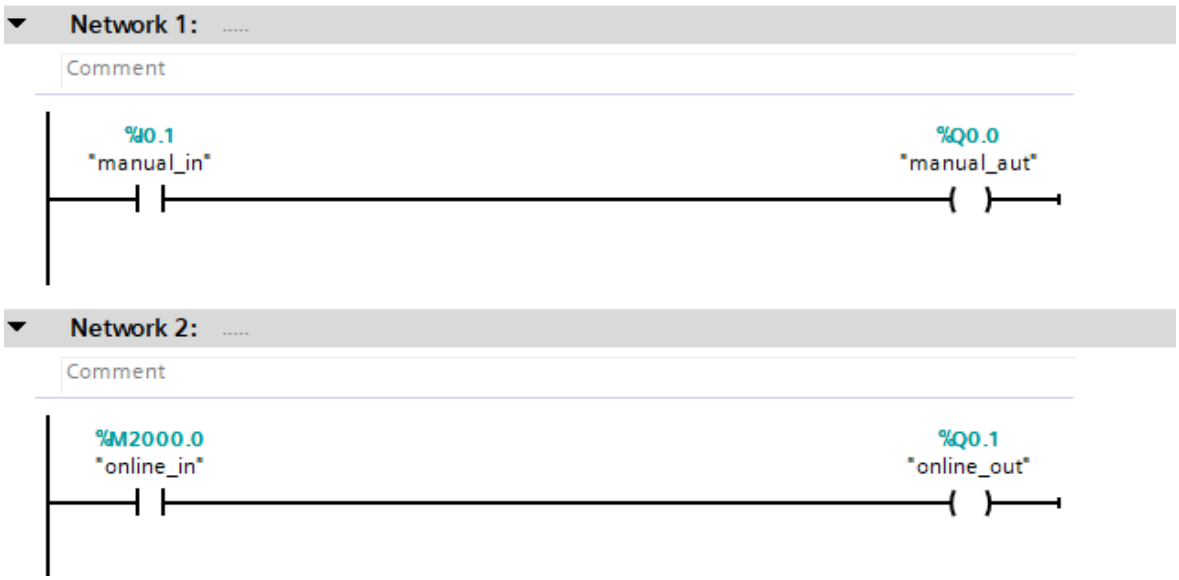

Figura E.2: Programa Ladder utilizado no exemplo

Se a entrada '%I0.1' for ativada, a saída '%Q0.0' é ativada e se a memória '%M2000.0' for ativada, a saída '%Q0.1' é ativada.

Após a criação do programa, segue-se a configuração da porta ethernet do PLC. Um detalhe importante é configurar o PLC, o módulo e o *gateway* na mesma subnet (foi utilizado o IP 192.168.114.xxx para configurar os dispositivos).

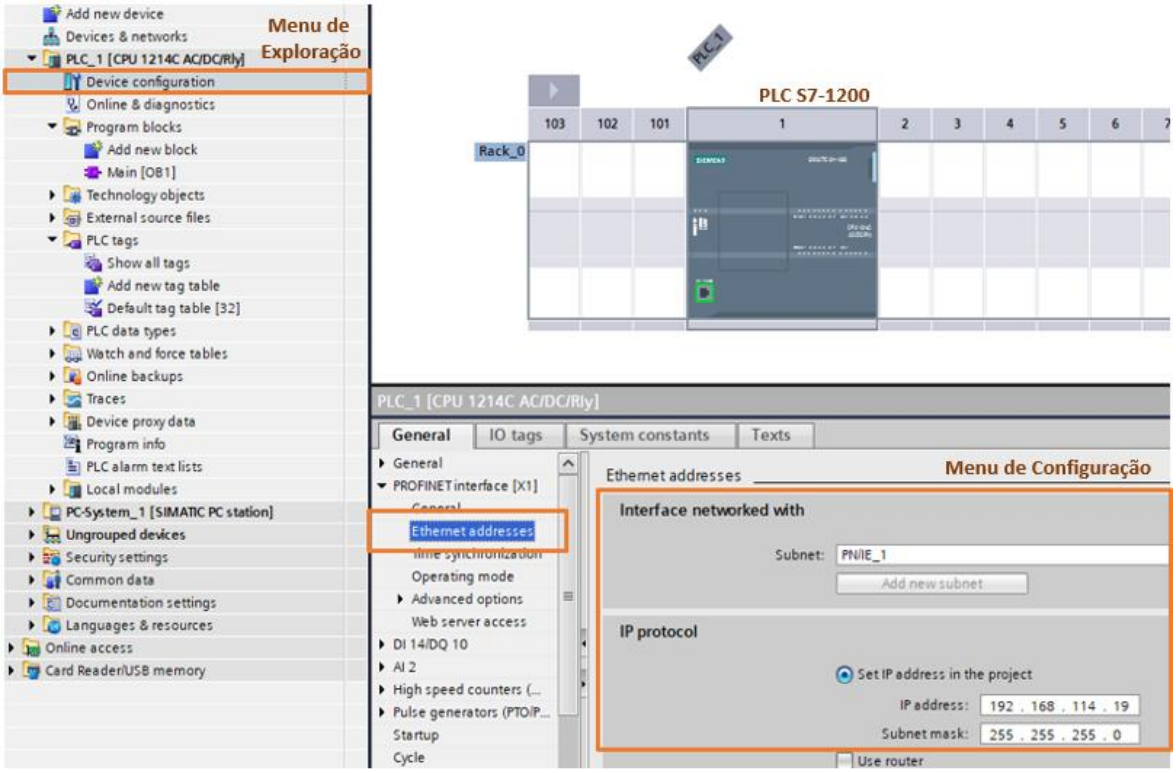

Figura E.3: Configuração do endereço IP do PLC

O IP atribuído ao PLC no exemplo desenvolvido foi 192.168.144.19 na rede local 'PN/IE_1'.

Outra configuração necessária ativar é a permissão por parte de utilizadores externos. Esta permissão permitirá que o módulo OPC interaja com o PLC.

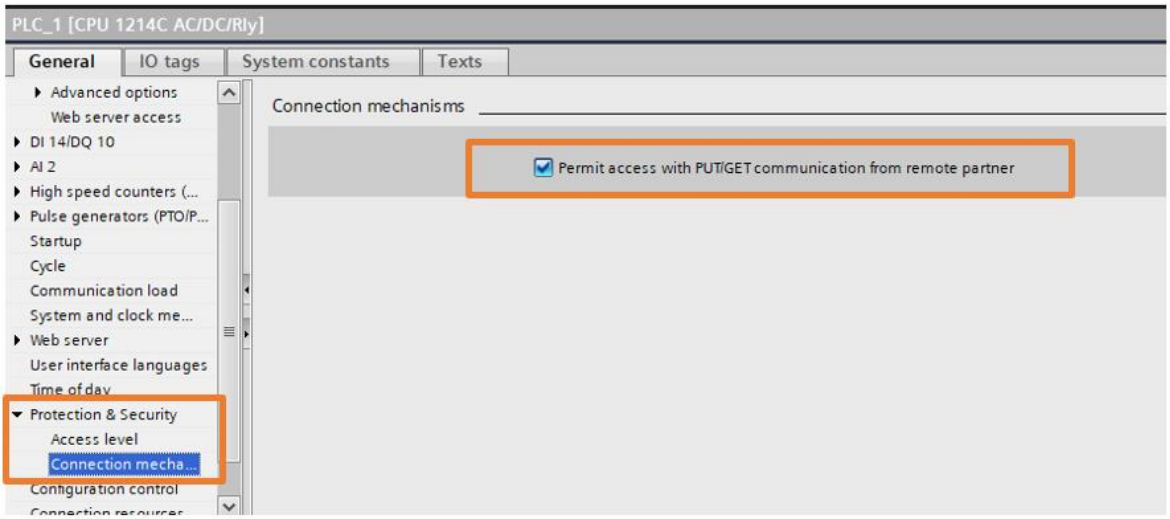

Figura E.4: Habilita a interação de agentes externos

E.2. Configuração do IBH Link UA

Neste subcapítulo serão apresentadas as etapas para configurar o módulo IBH Link UA. Se o IP do módulo for desconhecido, ou estiver numa subnet diferente do PLC, utilize um *browser* para editar o IP (é possível utilizar o IP do módulo ou o endereço 'ibhlinkua_<referencia>' para aceder à página do módulo).

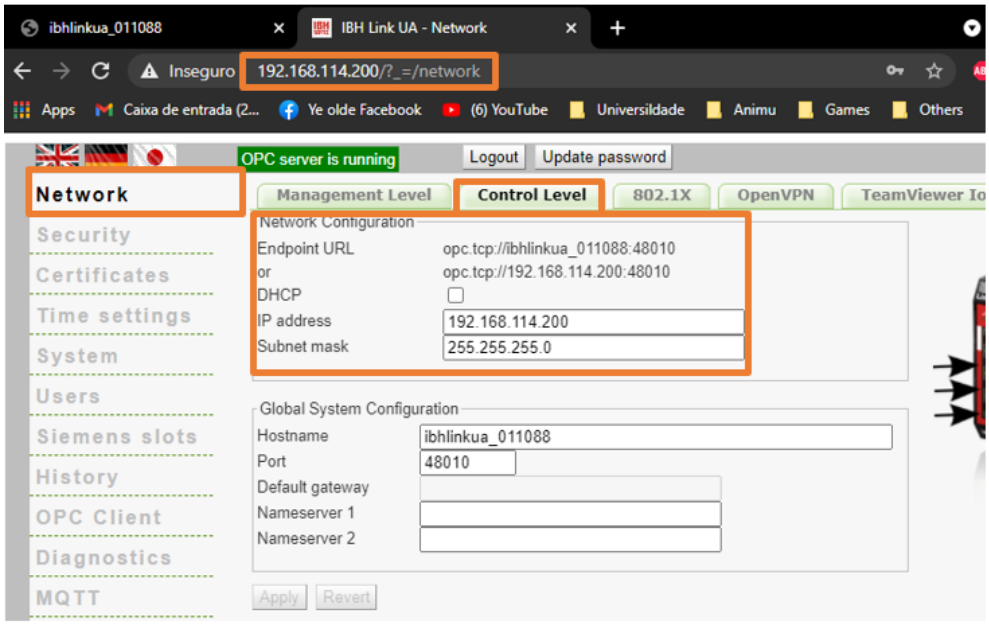

Figura E.5: Configuração do endereço IP do módulo OPC

O IP atribuído ao módulo OPC foi 192.168.114.200.

Configurado o IP do módulo, segue-se a configuração do servidor OPC no TIA Portal. A sequência para a sua configuração é a seguinte:

1. Adicionar um OPC Server genérico com as seguintes configurações;

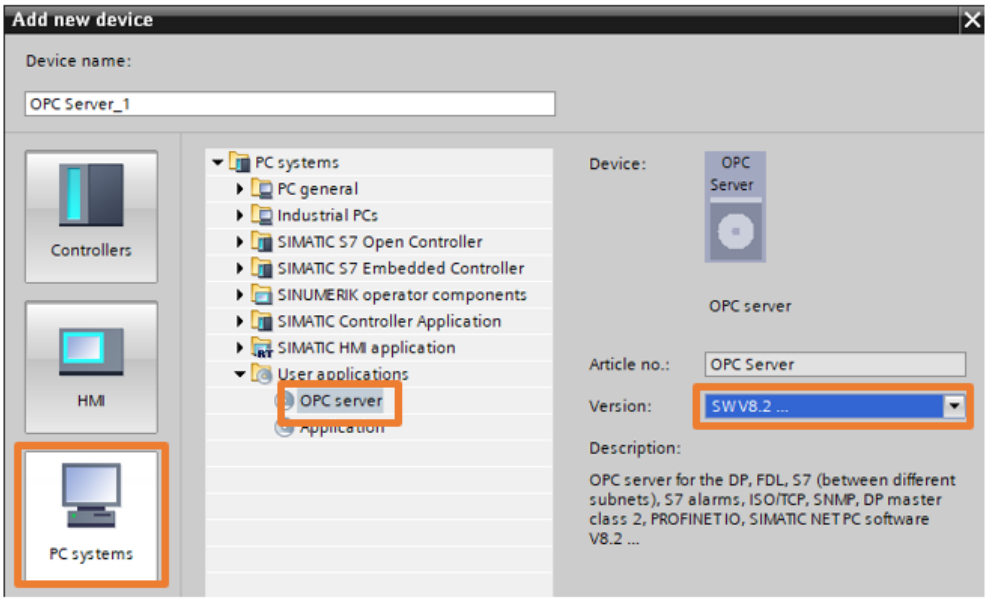

Figura E.6: Etapa 1 na configuração do Servidor OPC

2. Adicionar um adaptador ethernet;

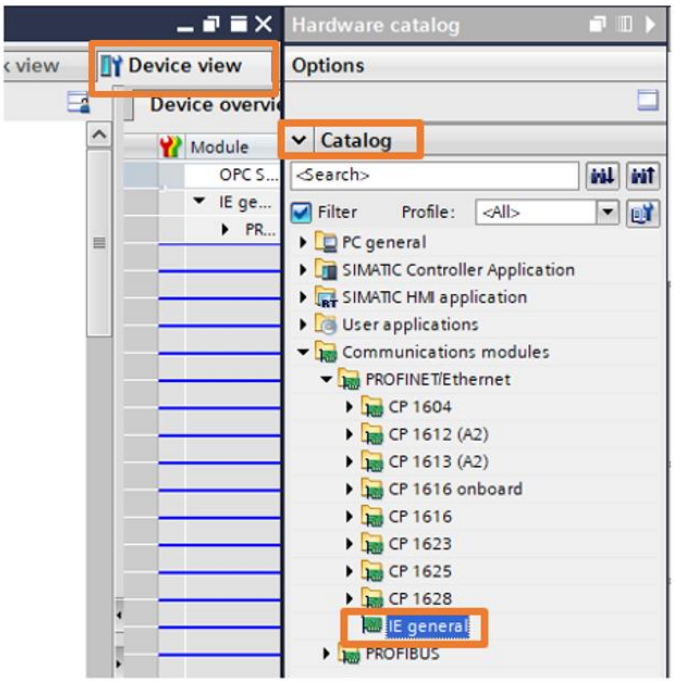

Figura E.7: Etapa 2 na configuração do Servidor OPC

3. Configurar o IP do OPC Server com o IP do IBH Link UA e ativar o Servidor;

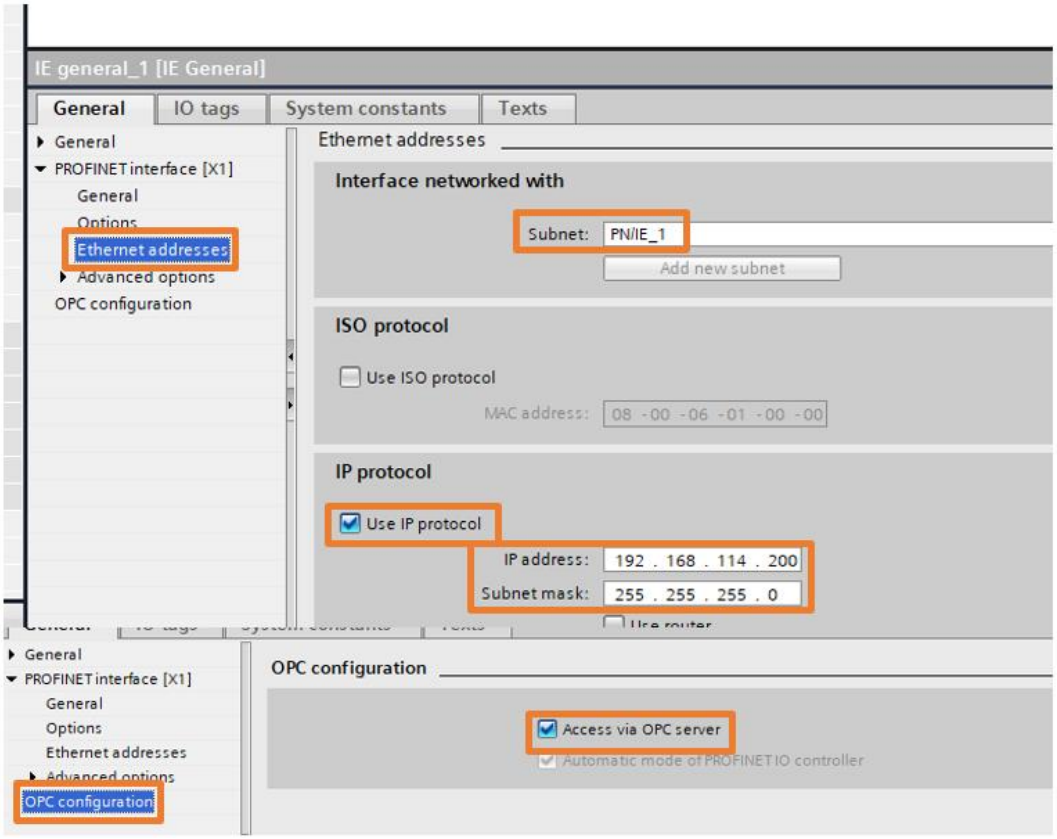

Figura E.8: Etapa 3 na configuração do Servidor OPC

4. Adicionar uma ligação S7 entre o servidor e o PLC;

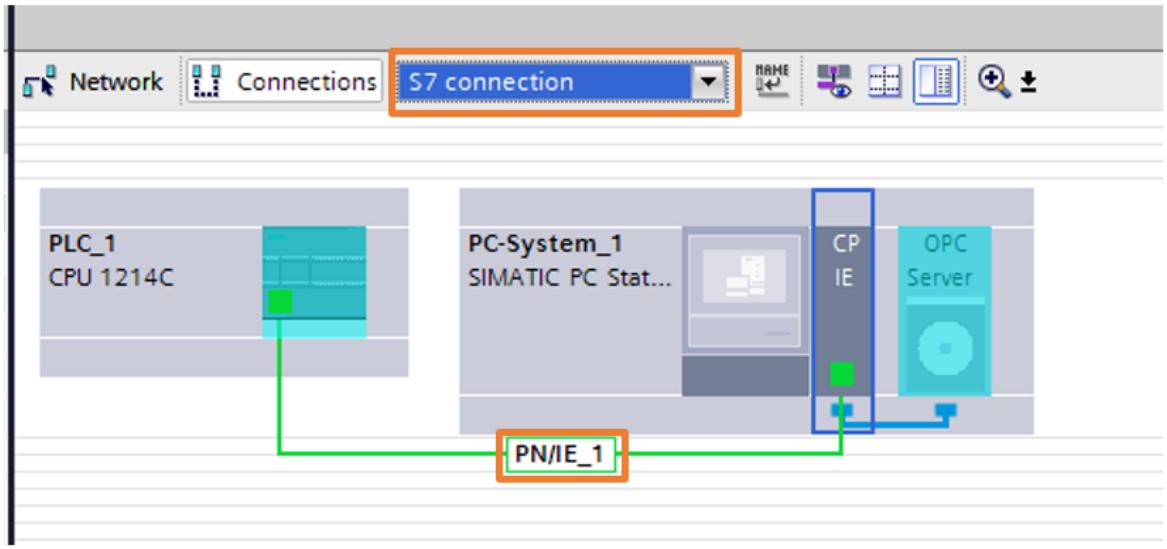

Figura E.9: Etapa 4 na configuração do Servidor OPC

5. Adicionar as variáveis a ser utilizadas pelo Servidor OPC (se forem todas, selecione 'all');

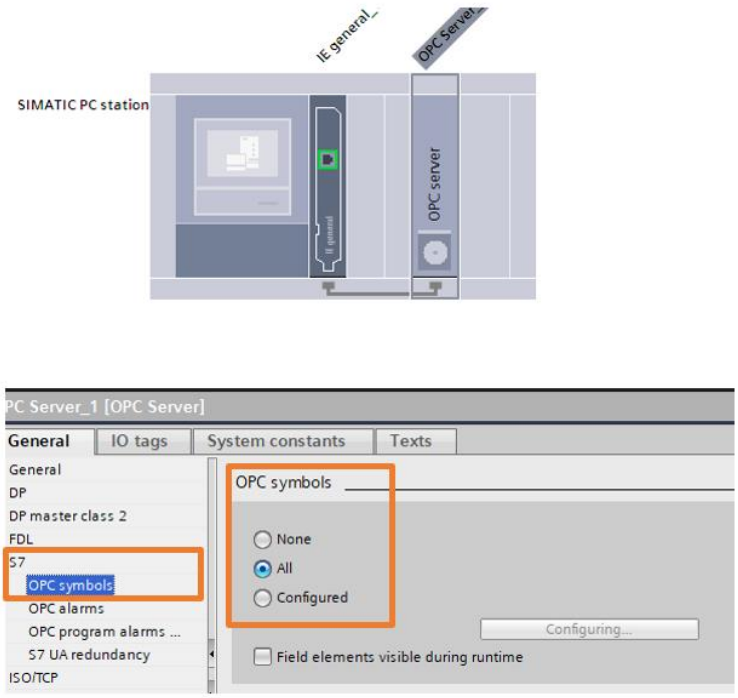

Figura E.10: Etapa 5 na configuração do Servidor OPC

Após realizar a configuração do módulo, segue-se a compilação e a instalação de cada programa para o seu equipamento (programa do PLC para o PLC e o do Servidor OPC para o módulo OPC).

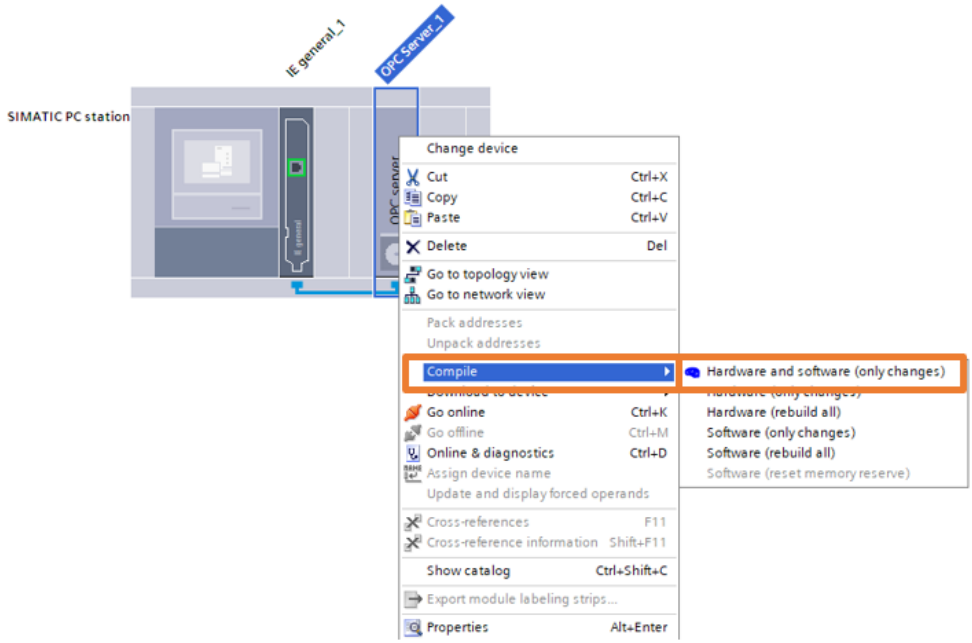

Figura E.11: Compilação do programa desenvolvido

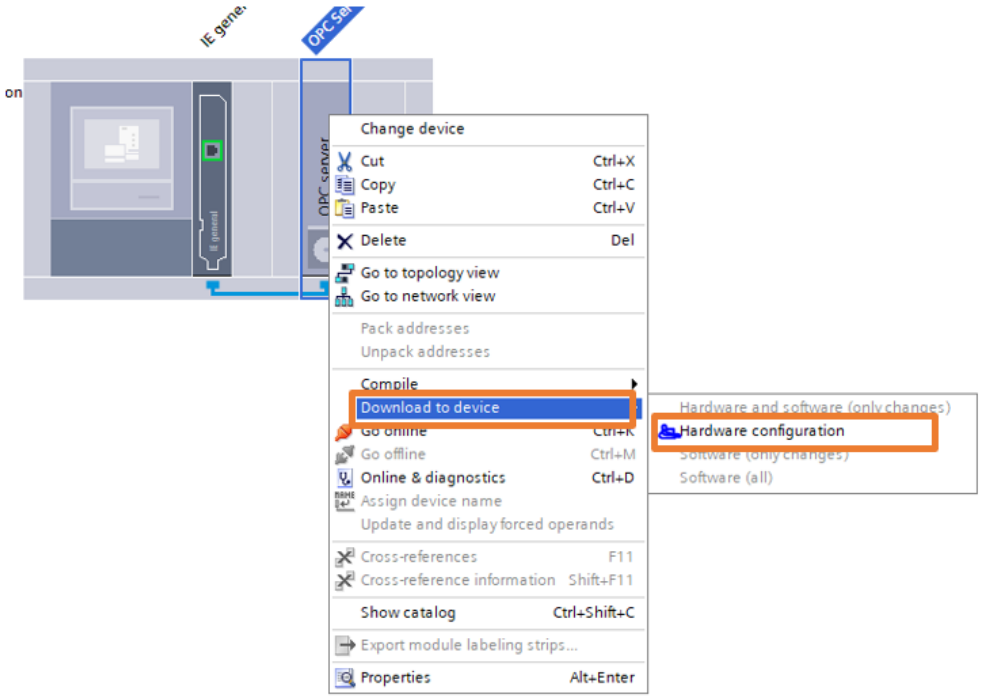

Figura E.12: Envio do programa desenvolvido

Após instalado o programa no módulo é possível visualizar a sua configuração na página de configurações do equipamento.

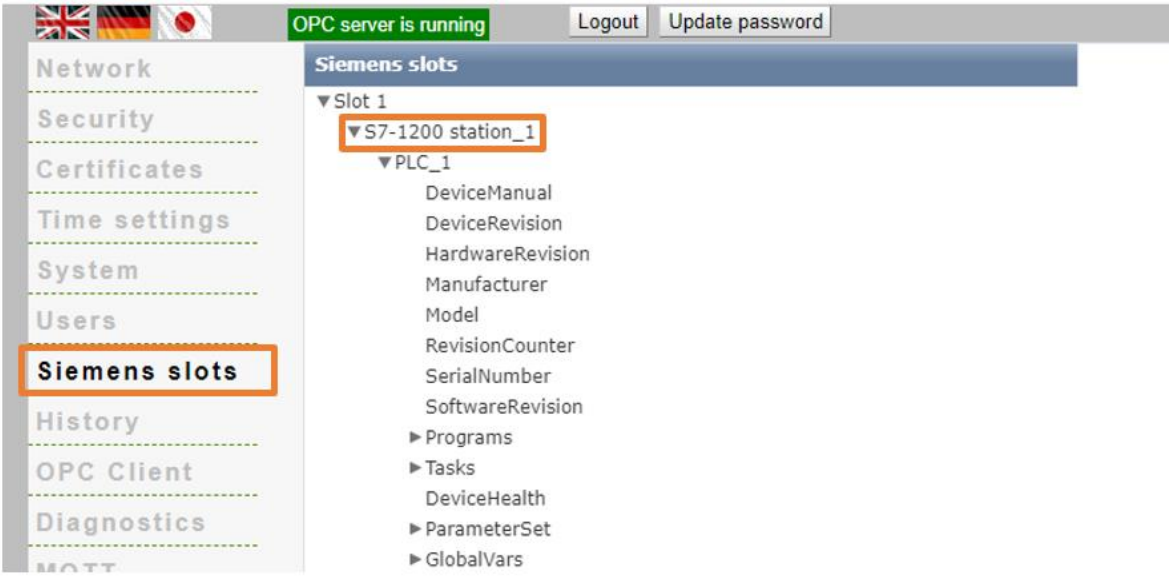

Figura E.13: Visualização do Servidor OPC no módulo OPC

E.3. Visualização das variáveis

A modificação e visualização das variáveis adicionadas ao Servidor OPC é por base de um OPC Client. O programa selecionado para esta fase de teste foi o '*UaExpert*', pois permite visualizar o endereço das variáveis para uso posterior.

As configurações do endereço OPC do módulo e os certificados encontram-se na página web deste.

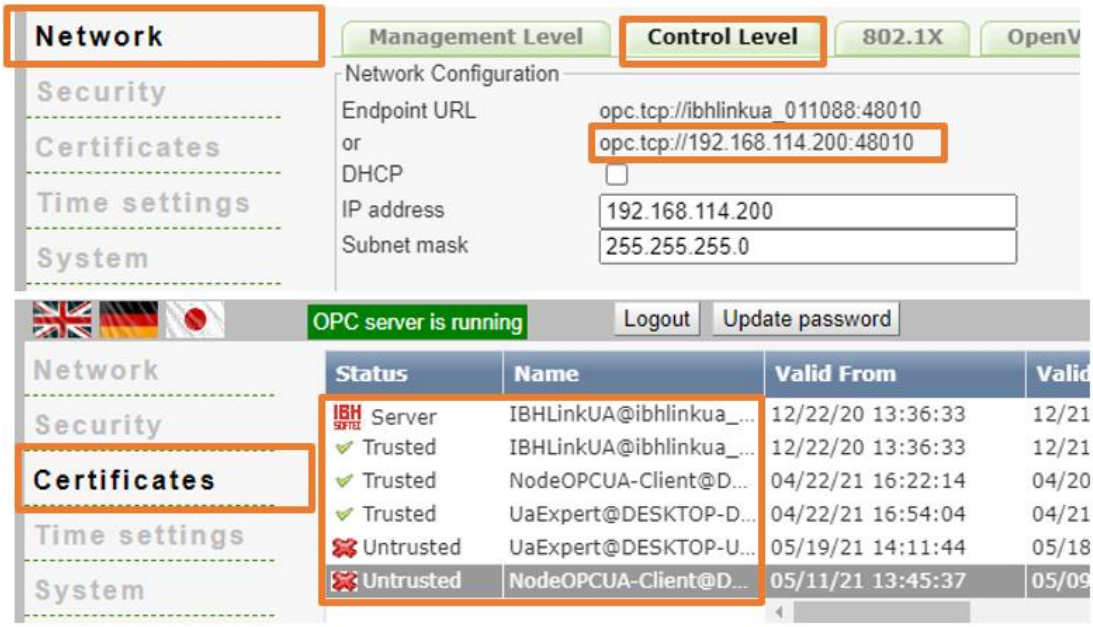

Figura E.14: Configurações e certificados do módulo OPC

Após adicionado o Servidor, é possível explorar as variáveis existentes e visualizar o seu endereço.

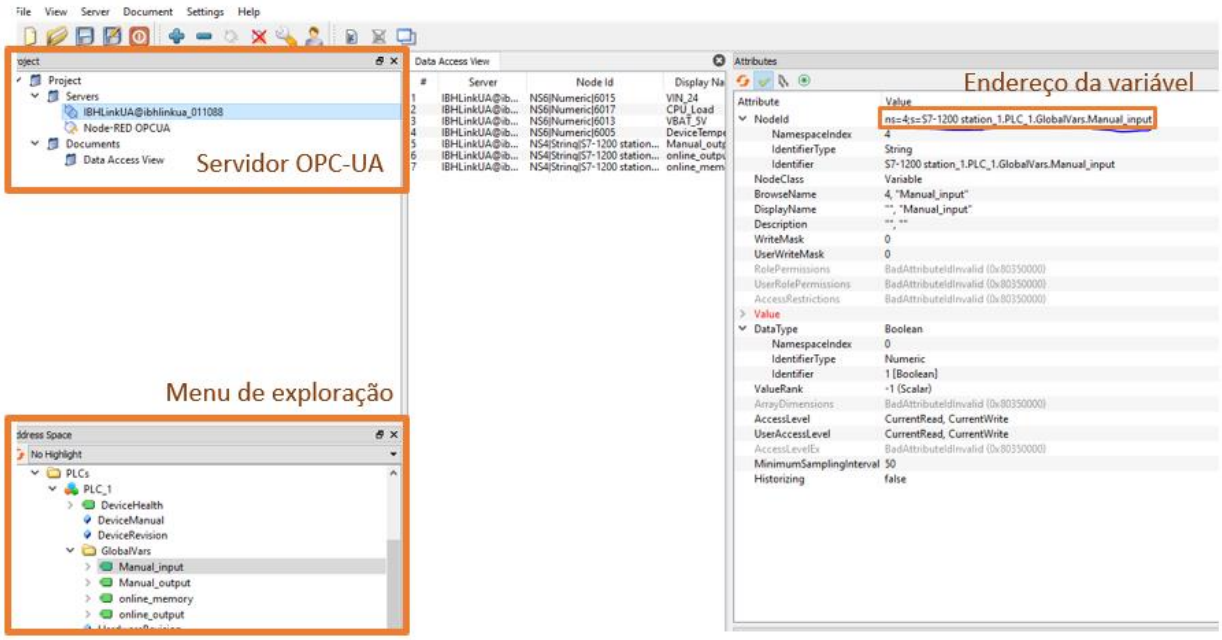

Figura E.15: Visualização das variáveis através de um Cliente OPC

E.4. Programa em Node-Red

Neste subcapítulo será apresentado um exemplo de aplicação de um OPC Client em Node-Red através de um Raspberry Pi. Para a realização deste exemplo foi utilizada a biblioteca *'node-red-contrib-opcua'*. Neste exemplo serão demonstradas duas finalidades: leitura de variáveis e escrita de variáveis. O diagrama de blocos utilizados e a sua configuração encontram-se na sequência de imagens seguintes:

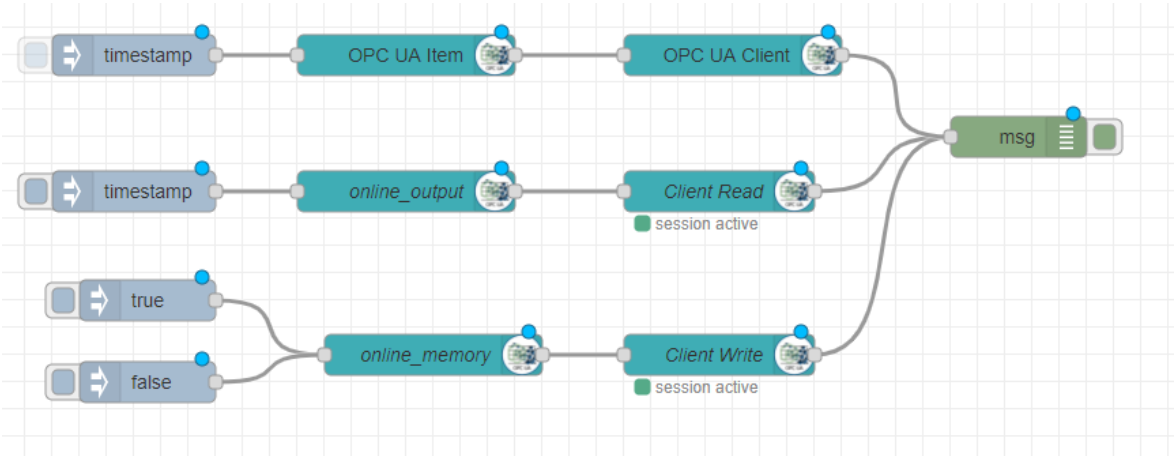

Figura E.16: Nós utilizados para desenvolver o programa

A primeira linha corresponde a uma sequência genérica para identificação dos blocos utilizados, a segunda realiza a leitura da variável *online_output* e a terceira permite escrever *true* ou *false* na variável *online_memory*.

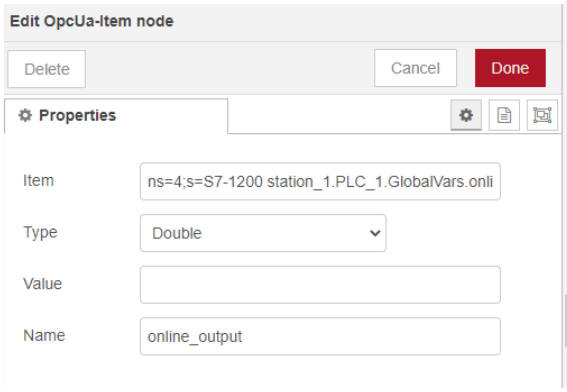

Figura E.17: Configuração da variável de leitura

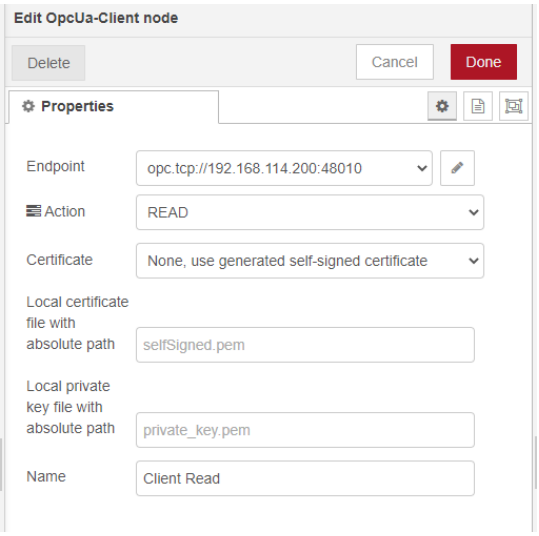

Figura E.18: Configuração do nó destinado à leitura

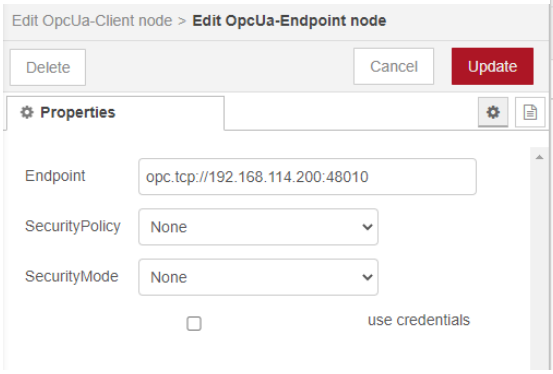

Figura E.19: Configuração do Servidor OPC

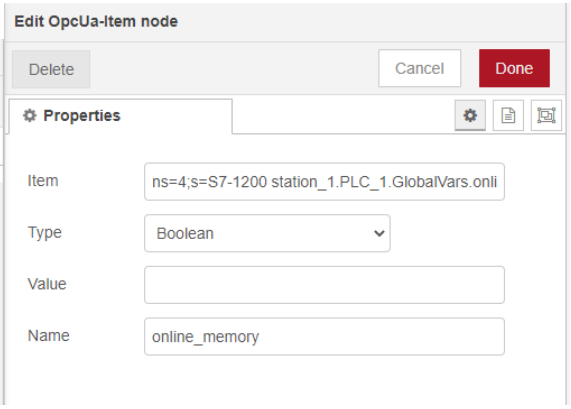

Figura E.20: Configuração da variável de escrita

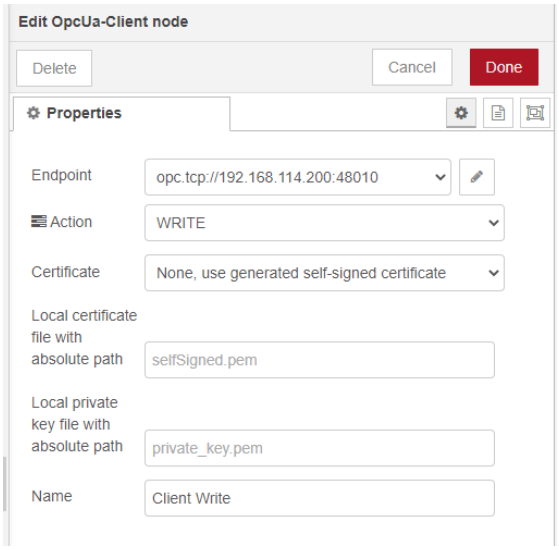

Figura E.21: Configuração do nó destinado à escrita

E.5. Resultados

Terminada a instalação e configuração de todos os componentes, é pretendido a possibilidade de realizar as seguintes operações:

E.5.1. Ativar a saída do PLC

Nesta imagem é possível visualizar que a variável *online_memory* se encontra a *true*, que ativou a segunda saída do PLC (*online_output*). Enviou-se um comando de escrita "*true*" na variável online_memory ao Servidor OPC, alocado no módulo OPC, que reencaminhou essa informação ao PLC. O PLC executou o comando e enviou a atualização ao módulo OPC, que por sua vez atualizou esta variável. Depois foi possível visualizar o seu valor através do cliente OPC-UA a ser executado no computador.

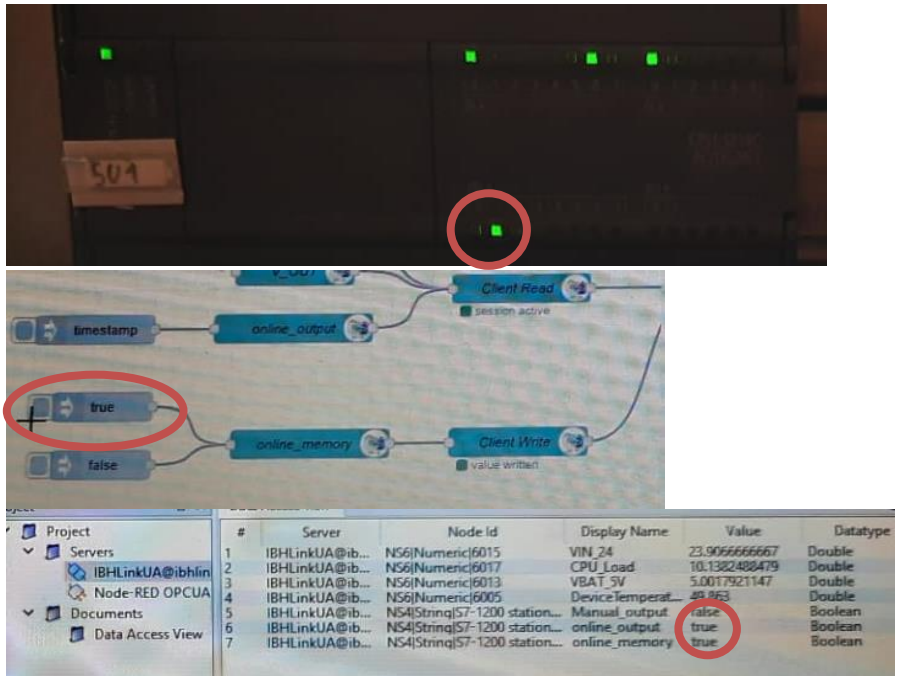

Figura E.22: Ativação da segunda saída do PLC com o Node-Red

E.5.2. Desativar a saída do PLC

Nesta imagem forçou-se a variável *online_memory* a *false*, que desativou a segunda saída. A metodologia de funcionamento é igual à explicada anteriormente.

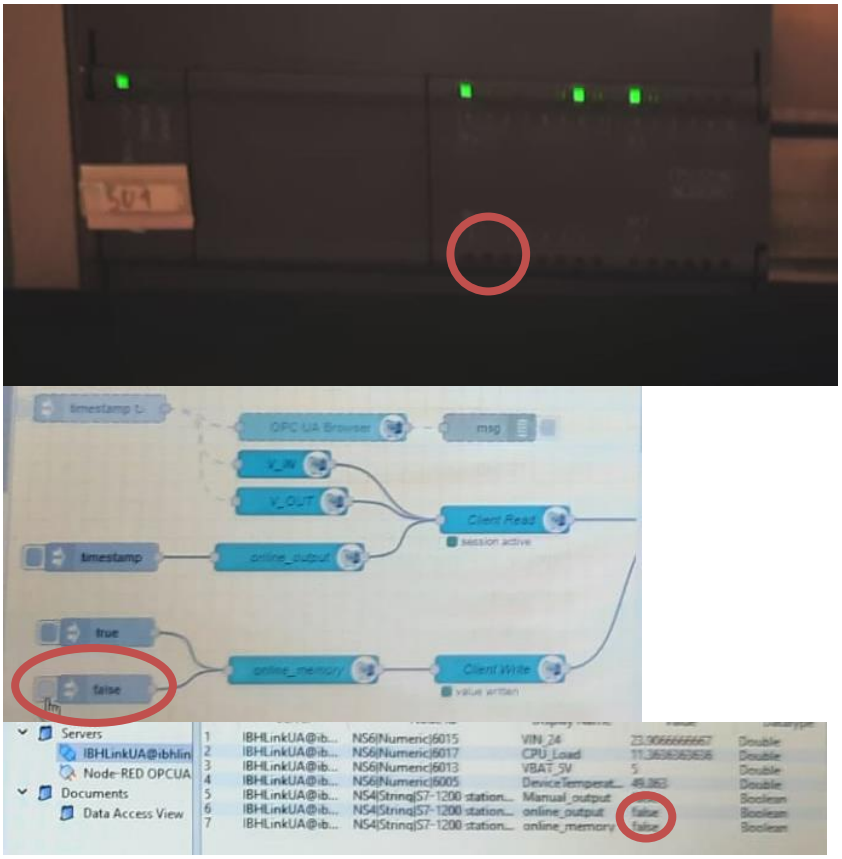

Figura E.23: Desativação da segunda saída do PLC com o Node-Red