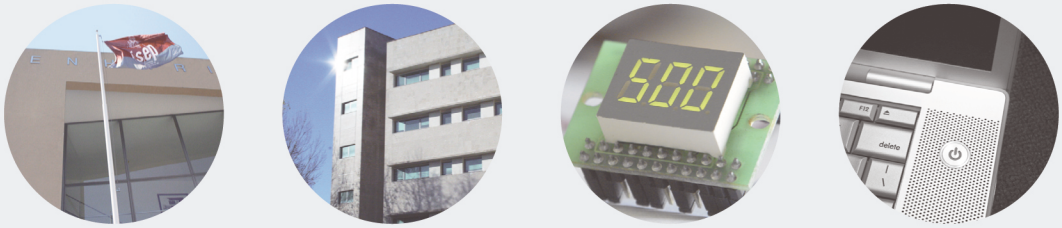
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Controlador de Domótica ZigBee

ANDRÉ ESTEVÃO CRUZ MAKRILOU
Outubro de 2020

POLITÉCNICO
DO PORTO

Controlador de Domótica ZigBee

André Estevão Cruz Makrilou

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: André Estevão Cruz Makrilou, Nº 1141249, 1141249@isep.ipp.pt
Orientação científica: Engenheiro Nuno Filipe da Foncesa Bastos Gomes, nbg@isep.ipp.pt
Empresa: CentralCasa - Projetos de Domótica Lda.
Supervisão: Carlos Silva, carlos.silva@centralcasa.pt

Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização de Automação e Sistemas
Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
2020

"You should be glad that bridge fell down.
I was planning to build thirteen more to that same design"

Isambard Kingdom Brunel

Resumo

Com os avanços tecnológicos é cada vez mais comum o uso de sistemas de pequena dimensão com vista a executar as tarefas realizadas no quotidiano. Este documento relata a implementação de um controlador de domótica residencial suportado pela tecnologia ZigBee, capaz de controlar e monitorizar as diversas funções que a automação residencial oferece.

Como tal, é utilizada uma placa de desenvolvimento responsável pelo processamento e exibição de dados, sendo adicionada a capacidade desta comunicar com dispositivos desta tecnologia com recurso a um periférico externo.

Tendo em conta a comunicação ZigBee e para facilitar a sua interpretação é utilizada uma biblioteca *standard*, *zigbee2mqtt* cuja principal finalidade é converter a comunicação num protocolo leve de mensagens, utilizado por sensores e dispositivos de pequena dimensão, designado por *Message Queuing Telemetry Transport* (MQTT). É através desta biblioteca que ocorre a gestão da rede, sendo reportados os estados dos diversos dispositivos, incluindo a remoção e adição de novos equipamentos.

É desenvolvido um servidor com a principal finalidade de interagir com esta biblioteca e suportar as comunicações com os diversos clientes que este sistema pode ter. De acordo com a função executada pelo cliente, o servidor toma decisões e publica mensagens no *zigbee2mqtt*. Os dados do sistema são armazenados com recurso a uma base de dados local.

A fácil interação do utilizador com o sistema é garantida recorrendo a uma interface gráfica onde o utilizador tem a capacidade de atuar os dispositivos e verificar o seu estado, proceder à sua remoção, exibir dados em função do eixo temporal, efetuar cálculos num intervalo de tempo e ainda criar regras/cenários.

Estas regras permitem automatizar o sistema, assim o utilizador não está dependente de aceder ao sistema a fim de atuar um determinado dispositivo. São dispostos dois tipos de regras: a atuação de um dispositivo em função de um outro e ainda a atuação de um dispositivo conforme o tempo.

A interface é hospedada num servidor externo, permitindo o controlo remoto do sistema. No entanto, com o objetivo de prevenir eventuais anomalias neste

contexto opta-se por servir esta aplicação também na placa de desenvolvimento. Deste modo, o utilizador tem constante acesso ao sistema independentemente da sua ligação à Internet.

O sistema geral é implementado tendo em conta o menor investimento possível e com funcionalidades que permitem não sobrecarregar a sua operabilidade. A aplicação é leve e eficaz, com a capacidade de executar todas as tarefas que um controlador de domótica comum oferece.

Palavras-Chave: controlador de domótica, ZigBee, placa de desenvolvimento, MQTT, *zigbee2mqtt*, base de dados, interface gráfica.

Abstract

With the technological advances it is increasingly common to use small systems in order to perform the tasks that become routine on a daily basis. This document reports the implementation of a home automation controller supported by ZigBee technology, capable to control and monitor the several functions that home automation offers.

To do so, a development board responsible for the data processing and a display is developed and the system has the ability to communicate with devices of this similar technology using an external peripheral support.

Taking into account the ZigBee communication and in order to facilitate its interpretation, a *standard zigbee2mqtt* library is used whose main purpose is to convert the communication into a lightweight messaging protocol, used by sensors and small devices, called MQTT. It is through this library that the network management occurs and the states of the different devices are reported, including the function of removing and adding new equipment.

A server is developed with the main purpose of interacting with this library and supporting the communications with the existing clients that this system may have. According to the function performed by the client, the server makes decisions and publishes messages in *zigbee2mqtt*. The system data is stored using a local database.

User-friendly interaction with the system is guaranteed using a graphical interface where the user has the ability to operate the devices and check their status, remove them, display data according to the time axis, perform calculations in a time interval and also create rules/scenarios.

These rules allow automatic action, so that the user is not required to access the system in order to act on certain device. Two types of rules are arranged: the actuation of one device depending on the state of the other and also the actuation of a device according to time.

The interface is hosted on an external server, allowing remote control of the system. However, in order to prevent any anomalies in this context, it is chosen

to serve this application also on the development board. With this, the user has constant access to the system regardless of his Internet connection.

The general system is implemented taking into account the smallestpossible investment and with functionalities that grant not to overload its operability. The application is light and effective, with the ability to perform all the tasks that a common home automation controller offers.

Keywords: home automation controller, development board, ZigBee, MQTT, *zigbee2mqtt*, database, graphical interface.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excerto de códigos

Acrónimos

IoT *Internet Of Things*

ACK *Acknowledgement*

RF Radiofrequência

PL *Powerline*

TP *Twisted Pair*

IP Internet Protocol

IEEE *Institute of Electrical and Electronics Engineers*

WLAN *Wireless LAN*

WPAN *Personal Area Network*

BAN *Body Area Network*

PSK *Phase Shift Keying*

BPSK *Binary Phase Shift Keying*

O-QPSK *Ortognal-Quadrature Phase Shift Keying*

DSSS *Direct Sequence Spread Spectrum*

OSI *Open Systems Interconnection*

MAC *Medium Access Control*

FFD *Full Function Devices*

RFD *Reduced Function Devices*

MPDUs *MAC Protocol Data Units*

PAN *Personal Area Network*

CSMA-CA *Carrier Sense Multiple Access - Collision Avoidance*

AES *Advanced Encryption Standard*

BP *Backoff Period*

FCS *Frame Check Sequence*

GTS *Guarantee Time Slots*

NLDE *Network Layer Data Entity*

NLME *Network Layer Management Entity*

APS *Application Support Sublayer*

ZDO *ZigBee Device Object*

AF *Application Framework*

MIC *Message Integrity Code*

MQTT *Message Queuing Telemetry Transport*

HTTP *Hypertext Transfer Protocol*

GPIO *General Purpose Input/Output*

USB *Universal Serial Bus*

M2M *machine-to-machine*

OpenHAB *Open Home Automation Bus*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

Capítulo 1

Introdução

Neste capítulo é feita a introdução deste projeto, com a contextualização e os seus objetivos. É apresentada a calendarização do mesmo e a organização deste documento.

1.1 Contextualização

O presente documento relata o projeto desenvolvido no âmbito da unidade curricular de Tese/Dissertação (TEDI) do 2º ano do Mestrado em Engenharia Eletrotécnica e de Computadores do Instituto Superior de Engenharia do Porto para obtenção do grau mestre do curso anteriormente mencionado com especialização em Automação e Sistemas.

Este projeto que surge com a proposta de estágio curricular apresentada pela empresa Central Casa, associada à automação através da domótica.

Sediada em Vila Nova de Gaia, esta empresa surgiu através de uma outra designada por PontoPR que desde o ano da sua implementação, 1999, procurou unir três grandes áreas como a Internet, o e-marketing e a domótica. Em setembro de 2002 a CentralCasa, ainda dentro da PontoPR, lança o primeiro portal de domótica em Portugal.

A empresa é legalmente constituída em agosto de 2003 com o principal objetivo de oferecer as melhores soluções especializas na área da domótica. Atualmente, a CentralCasa atua em quatro áreas: projetos para empreendimentos e moradias, formação, distribuição de produtos, investigação e desenvolvimento.

No entanto, tendo em conta a situação epidemiológica que se vive à data, o projeto, é desenvolvido em ambiente residencial.

1.2 Objetivos

O principal objetivo deste projeto consiste na implementação de um controlador de domótica residêncial através de uma placa de desenvolvimento para o efeito suportado pelo protocolo ZigBee.

Tratando-se de um projeto com a finalidade de ser equiparado aos controladores de domótica já existentes, é necessário garantir que o cliente tem a capacidade de aceder e controlar a automação na sua residência remotamente. Assim sendo, é esperada a implementação de um servidor Web capaz de comunicar com o cliente para que este controle a sua residência remotamente.

Este sistema deverá ser eficaz e fiável tendo em conta o menor investimento possível. Resumidamente identificam-se os seguintes objetivos:

- O estudo dos conceitos teóricos associados ao desenvolvimento deste projeto, destacando-se a Domótica e o protocolo Zigbee;
- A análise do estado da arte, dos protocolos e controladores de domótica já existentes;
- O estudo das plataformas e interfaces *open source* disponíveis;
- A implementação de um servidor local capaz de comunicar com os dispositivos e de uma base de dados que armazene os dados destes.
- O desenvolvimento de uma interface gráfica de fácil interação para o utilizador.

1.3 Calendarização

A tabela 1.1 apresenta as diferentes fases de desenvolvimento do protótipo funcional e da elaboração do relatório em função das semanas.

	maio					junho				julho				agosto					setembro				outubro			
	1ª	2ª	3ª	4ª	5ª	6ª	7ª	8ª	9ª	10ª	11ª	12ª	13ª	14ª	15ª	16ª	17ª	18ª	19ª	20ª	21ª	22ª	23ª	24ª	25ª	26ª
Estudo dos protocolos																										
Estudo promenorizado ZigBee																										
Análise de requesitos																										
Testes com o zigbee2mqtt																										
Implementação do servidor																										
Implementação da interface gráfica																										
Implementação dos cenários																										
Elaboração do relatório																										

Tabela 1.1: Calenderização do projeto

1.4 Organização da dissertação

O primeiro capítulo relata a introdução deste documento e a contextualização da escolha do projeto. É descrita a empresa empregadora destacando os pontos

mais importantes, como a sua fundação e o seu mercado. Para uma melhor precessão do trabalho desenvolvido em função do eixo temporal é ainda apresentada a calendarização deste projeto.

O segundo capítulo aborda todos os conceitos teóricos necessários para a implementação do protótipo funcional, começando pela definição de Domótica, e da *Internet Of Things* (IoT). São analisados os principais protocolos de comunicação no setor da domótica residencial, sendo feita a comparação entre estes. De seguida é apresentado o estado de arte. São analisados com detalhe a tecnologia ZigBee e o protocolo de mensagens MQTT, utilizado na comunicação entre dispositivos de pequenas dimensões numa rede. Por fim, são analisados as ofertas de mercado relativas ao setor e os principais *software open-source*.

O terceiro capítulo descreve o projeto em geral, com a apresentação do *hardware* e *software* utilizado no desenvolvimento do protótipo, sendo especificadas as suas principais carateristicas e o porquê da sua escolha. Para finalizar, é apresentado um diagrama geral e uma estimativa de custos associada a este projeto.

O quarto capítulo detalha toda a fase de implementação, sendo descritos os passos até a concretização da solução.

No quinto capítulo são apresentados os testes realizados e os resultados obtidos com esta implementação. O protótipo é ilustrado através de figuras que fazem a descrição de cada componente.

No último capítulo estão presentes as conclusões do desenvolvimento e as possíveis melhorias a implementar.

Capítulo 2

Estado de arte

Neste capítulo são apresentados todos os conceitos teóricos associados à elaboração desta dissertação. Inicialmente, é descrito o conceito da automação residêncial, os protocolos de comunicação já existentes para o efeito, sendo feita uma abordagem mais concreta ao protocolo utilizado ZigBee. São apresentadas algumas das ofertas para os controladores disponíveis no mercado.

2.1 Domótica

Este conceito surgiu da junção da palavra *Domus*, "casa" em latim, com a palavra robótica. O grande objetivo é tornar os equipamentos elétricos e eletrónicos que utilizamos quotidianamente nas nossas casas em dispositivos autónomos que podem ser controlados no local ou remotamente. O controlo da iluminação é um exemplo, ao utilizar um sensor de movimento assim que este detete movimento a luz acende, dispensando o utilizador de realizar esforços para acender ou apagar a luz. Além do conforto, esta tarefa automática permite otimizar o consumo de energia elétrica.

Este tipo de tecnologia originou o termo "*Smart Home*" (casa inteligente), sendo possível automatizar de forma simples a habitação através de uma central de controlo. Isto torna a habitação mais moderna, mais prática e fácil de usar, mais acessível em termo de controlo, mais económica energicamente e mais segura [1]. Na figura 2.1 é apresentado um sistema geral deste tipo.

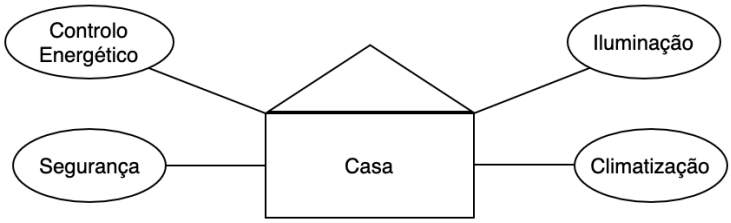


Figura 2.1: Sistema geral de Automação Residêncial

> The smart home concept is the integration of different services within a home by using a common communication system.
>
> *R. Lutof*

Utilizando a citação de R. Lutof a domótica procura integrar diversos tipos de serviços numa habitação recorrendo ao mesmo meio de comunicação. Como tal, uma rede de domótica é constituída por:

- **Sensores** que são dispositivos de entrada uma vez que recolhem informações e enviam para os outros equipamentos do sistema;
- **Atuadores** que são dispositivos de saída, o seu estado é alterado consoante a informação que é enviada pelo sistema;
- **Controladores**, conhecidos também por *gateways*, que são considerados os cérebros do sistema, têem como função receber os dados provenientes dos sensores e enviar tomadas de decisão para os atuadores.

O utilizador consegue aceder e configurar o sistema através de uma interface gráfica, associada ao controlador, que normalmente está disposta localmente a partir de um *tablet* ou computador. Em alternativa e de forma a satisfazer as necessidades do utilizador, existe a possibilidade de conectar o controlador à Internet permitindo assim o controlo remoto.

2.1.1 Internet of Things

O conceito IoT é geralmente utilizado para todas as tecnologias que permitem a ligação de um dispositivo à Internet. A sua finalidade é interligar sistemas de informação que utilizamos diariamente como o computador, *tablet* e *smartphone* de forma a receber informações em tempo real. Estes dados são utilizados para

monitorizar, controlar e transferir informações para outros dispositivos através da Internet. Numa rede IoT é possível envolver diferentes dispositivos, plataformas, sistemas operativos e serviços conectados entre si, utilizando protocolos de comunicação diferentes [2].

Assim, graças à automação residêncial IoT é possível que a habitação saiba quando realizar determinadas ações podendo executá-las de forma automática. Este é o grande propósito da IoT. Em [3] são apresentadas as seguintes aplicações:

- **Iluminação** - É essêncial para qualquer habitação possuir pontos de luz, no entanto é importante que para além da alteração de estado, ligar e desligar, seja possível o controlo da intensidade luminosa. Graças à IoT é possível a criação de rotinas como por exemplo só ligar a luz assim que anoiteça, ou até regular esta intensidade para as diferentes fases do dia;
- **Portas** - Para além da comodidade é importante garantir a segurança, com os avanços tecnológicos já existem soluções que permitem o reconhecimento facial ou um dispositivo inteligente para aceder ao interior da habitação;
- **Janelas** - Assim como na iluminação, e para satisfazer a comodidade do utilizador é possível que as persianas abram de acordo com a temperatura ou intensidade luminosa exterior ou interior;
- **Rega** - Para os utilizadores que possuam jardim, de forma a manter o terreno húmido é necessário proceder à sua rega, a IoT com suporte a sistemas de irrigação oferece a possibilidade de apenas ativar o sistema de rega assim que detete que o parametro da humidade atinja um determinado limite;

Estas são algumas das diversas aplicações que uma habitação residêncial pode ter, com isto, fica ciente que a principal finalidade da IoT, juntamente com a Domótica, é satisfazer as necessidades dos seus utilizadores dispensando-os de realizar tarefas comuns e repetitivas no seu quotidiano.

2.1.2 Protocolos de Comunicação

X10

É o protocolo mais antigo, lançado em 1975 pela *Pico Electronics*, consiste num sistema simples que faz uso das linhas elétricas da habitação ou comunicação rádio para garantir a comunicação entre dispositivos e electrodomésticos. É um protocolo fiável, porém está sujeito à interferência de outros dispositivos presentes no circuito. Para solucionar este potencial problema existem filtros de ruído que atenuam esta interferência. Os pacotes transmitidos consistem num código da casa, seguido de um ou mais códigos de unidade, finalmente seguido de um comando todos eles de quatro bits. Sendo 256 os endereços possíveis (16 códigos

de casa x 16 códigos de unidade), o dispositivo quando instalado é configurado para responder a um só endereço [4]. Uma das principais desvantagens é o facto das mensagens/comandos serem enviados à vez, isto implica que se forem enviadas mensagens em simultâneo estas poderão não chegar ao destinatário. Face aos restantes protocolos que são apresentados a seguir, é de referir que não é enviado um *Acknowledgement* (ACK) [1], sendo um dos parâmetros indispensáveis neste tipo de comunicação [5]. Como benefícios destacam-se os factos de ser possível adicionar a rede X10 às residências já existentes sem a necessidade de instalação de novos fios e ainda o facto desta tecnologia utilizar a linha elétrica sendo mais fiável do que a comunicação sem fio.

Insteon

O Insteon, lançado em 2005 pela *Smartlabs*, é o único protocolo que combina tecnologia com fio e sem fio, wireless e *Powerline* (PL). Opera através de linhas elétricas, Radiofrequência (RF), ou ambos. Assim, dá ao utilizador a flexibilidade de instalar dispositivos em qualquer lugar da habitação desde que exista sinal elétrico ou alcance sem fio. A alimentação é utilizada como *backup* da RF usada no sistema, com isto, a transmissão de pacotes ocorre com pouca ou nenhuma interferência [4]. Neste protocolo é formada uma rede de malha dupla, onde cada dispositivo transmite, recebe e repete mensagens [6]. Cada mensagem recebida num determinado dispositivo é rastreada por um controlo de deteção e correção de erros até ser retransmitida. Desta forma é melhorada a fiabilidade do sistema. A taxa de transmissão de dados numa rede deste tipo é cerca de 180 bits por segundo, tendo alcance máximo de aproximadamente 120 metros sem obstáculos. A principal vantagem desta tecnologia é a redundância que oferece, caso a rede sem fio desligue ou ocorra algum tipo de interferência o sistema adapta-se para comunicação exclusivamente com fio (linha elétrica) e vice-versa. Tem também como vantagem o facto de ser compatível com X10, podendo assim adicionar a capacidade de comunicação sem fio a uma rede X10 já existente [7].

KNX

O protocolo KNX opera da mesma forma que a tecnologia Insteon, através de linhas elétricas (PL) e RF com a particularidade de suportar a transmissão de dados por infravermelhos, par entrançado (*Twisted Pair* (TP)) e cabos Ethernet (Internet Protocol (IP)). A vantagem de utilizar a configuração de par entrançado neste sistema é a eliminação da interferência elétrica. Opera com uma taxa de transmissão de 9600 bps na configuração par entrançado e 1200 bps através da linha elétrica. Para os dispositivos que façam uso da RF a banda de frequência na Europa fixa-se nos 868 MHz, sendo a taxa de transmissão 16,4 kbps.

[1] *Acknowledgedment* - Caractere de controlo usado para indicar que uma mensagem transmitida foi recebida sem interferência ou sem erros, ou que o receptor está disponível a aceitar novas transmissões.

ZigBee

Designado em 2004 pela a *Zigbee Aliance*, este protocolo funciona no padrão de rádio 802.15.14 e faz uso de uma rede em malha para estabelecer a comunicação entre os dispositivos. Esta comunicação é tipicamente de baixa velocidade, largura de banda estreita e de curtas distâncias. A frequência de trabalho deste protocolo fixa-se nos 2,4 GHz, pode ainda assim funcionar nos Estados Unidos da América nos 915 MHz. O alcance de cada dispositivo pode atingir os 100 metros sem obstáculos, na prática este valor tende para os 10 metros em espaços anteriores. No entanto, sendo uma rede *mesh* as mensagens podem ser retransmitidas até um limite máximo de seis dispositivos aumentando este valor até 60 metros. A comunicação é bidirecional e a receção das mensagens podem ser confirmadas através do ACK. É um dos protocolos mais conceituados na área uma vez que os dispositivos apresentam baixo consumo energético. Ainda assim, tem como principal desvantagem o número limitado de dispositivos no mercado [4].

Bluetooth

A comunicação por Bluetooth tende a ser mais rápida do que por ZigBee e Z-Wave, porém, funciona em curtas distâncias. Opera na frequência de 2,4 GHz e a comunicação funciona com a configuração *master/slave*. Ao consumir pouca energia, este protocolo tem como vantagem a possibilidade dos dispositivos funcionarem através de bateria, e como desvantagem a diminuição do alcance entre dispositivos. Relativamente à taxa de transmissão de dados, o Bluetooth 1.0 pode atingir 1 Mbps, enquanto que nas versões 3.0 e 4.0 chega aos 24 Mbps [4]. Esta tecnologia é barata, fácil de implementar tendo um dos melhores métodos de emparelhamento, e é amplamente utilizada na maioria dos telemóveis e *tablets*. Como limitações é de referir que utiliza a banda dos 2,4 GHz, sendo esta muito movimentada, razão pela qual poderá causar interferência noutros dispositivos sem fio.

Wi-Fi

Tendo em conta que a grande maioria das residências atualmente possuem Wi-Fi faz sentido usufrir desta tecnologia para a automação residêncial. As comunicações Wi-Fi usam frequências na ordem dos 2,4 GHz ou 5 GHz. O alcance teórico é relativamente longo, mas para uso doméstico o alcance máximo é de cerca de 18 metros [8]. O rendimento das comunicações é classificado como bastante satisfatório, e as velocidades variam entre 10 Mbps e 100 Mbps. A maior desvantagem é que assim como noutros protocolos, o sistema esta suscetivel a interferências de outros dispositivos, principalmente nesta tecnologia, uma vez que podem estar conectados diversos *gadgets*. O facto de usar uma ampla largura de banda e altas velocidades faz com que os dispositivos consumam mais energia do que comparando com outros protocolos.

Z-Wave

Z-Wave é um protocolo de comunicação sem fio desenvolvido pela empresa *Zensys*, em 2001, especificamente para o mercado de automação residencial. Nos Estados Unidos da América o sistema opera nas bandas de rádio de 908 e 916 MHz, enquanto que na Europa 868,42 MHz. Cada dispositivo pode enviar e receber comandos e até passar comandos para outros dispositivos. Este tipo de rede é designada por rede em malha (*mesh*), também conhecida como rede *ad-hoc*. Isto permite que se um dispositivo pretender comunicar com um outro, fora do alcance, a comunicação pode ser estabelecida através de um terceiro que esteja ao alcance destes dois, a mensagem é transmitida até chegar ao dispositivo recetor. Porém, a mensagem só pode ser retransmitida caso o número de saltos entre dispositivos seja inferior ou igual a quatro. Comparativamente aos protocolos Ethernet, Wi-Fi, ZigBee e Thread a transmissão de dados tende a ser mais lenta com valores entre 40 e 100 Kbps [8] [4]. A comunicação é bidirecional e a receção das mensagens podem ser confirmadas através do ACK. O alcance da transmissão ao ar livre é de aproximadamente 100 metros, no entanto, o alcance na habitação entre dois dispositivos é 25 metros [4].

Thread

O Thread é uma tecnologia de rede em malha de baixo consumo de energia, baseada em IPv6, para produtos IoT. É o protocolo mais recente, tendo sido formado por grandes empresas de tecnologia como a Samsung, Google/Nest e Apple. Faz uso da tipologia *mesh*, permitindo um máximo de 36 saltos entre dispositivos, cada salto tem um alcance de aproximadamente 30 metros. Opera com base no padrão IEEE 802.15.4 com frequências na ordem dos 868 MHz na Europa e 2,4 GHz nos Estados Unidos da América. A transmissão de dados alcança os 250 Mbps [8], o que é mais do que satisfatório para dispositivos de automação residêncial. Assim como Z-Wave e ZigBee, os dispositivos apresentam baixo consumo de energia e comunicações em malha. Ao utilizar o protocolo IP, esta tecnologia trás como vantagem o factor de ser possível criar uma rede Thread entre *gadgets* já existentes, não sendo necessário um controlador principal [8]. Como principal desvantagem prende-se o facto de ser um protocolo muito recente e ainda não existirem no mercado muitos dispositivos que o suportem.

A tabela 2.1 apresenta de forma resumida a comparação entre os protocolos de comunicação mais conceituados na automação residêncial. No sub capítulo 2.2 irá ser abordado com maior detalhe o protocolo ZigBee, protocolo este que será utilizado na implementação do protótipo funcional.

	X10	Insteon	KNX	ZigBee	Bluetooth	Wi-Fi	Z-Wave	Thread
Ano de lançamento	1975	2005	1999	2004	1999	1998	2001	2015
Norma	Própria	Própria	Própria	IEEE 802.15.4	IEEE 802.15.1	IEEE 802.11a/b/g	Própria	IEEE 802.15.4
Comunicação	RF, PL	RF, PL	RF, TP, IP, PL	RF	RF	RF	RF	RF
Encriptação	X	X	128-bit AES	128-bit AES	40-bit RC4	128-bit AES	128-bit AES	128-bit AES
Consumo	Alto	Alto	Médio	Baixo	Baixo	Alto	Alto	Baixo
Taxa de transmissão	20 - 200 bps	180 bps	16.4 kbps|9600 bps|1200 bps	>20 kbps	1 Mbps	2.4 Gbps	40 kbps	250 kbps
Comunicação (2 sentidos)	X	✓	✓	✓	✓	✓	✓	✓
Alcance (RF)	30 m	10 m	20 m	10 m	2 m	18 m	25 m	10 m
Compatibilidade	Baixa	Média	Alta	Média	Alta	Alta	Alta	Baixa

Tabela 2.1: Comparação entre os protocolos estudados

2.2 ZigBee

O protocolo ZigBee é um protocolo de comunicação padrão, desenvolvido pela *Zigbee Aliance*, para redes de malha sem fios. Trata-se de um protocolo de baixa potência com alto fator de fiabilidade. Apesar de apresentar um número limitado de dispositivos no mercado, é de referir que o facto de apresentar uma norma (IEEE 802.15.4) permite uma grande compatibilidade entre os diversos dispositivos independentemente do fabricante.

2.2.1 Norma IEEE 802.15.4

O *Institute of Electrical and Electronics Engineers* (IEEE), em português Instituto de Engenheiros Eletrotécnicos e Eletrónicos, é uma organização profissional fundada nos Estados Unidos da América dedicada à evolução tecnológica. Tem como objetivo primordial oferecer oportunidades de aprendizagem no âmbito das ciências de engenharia, investigação, e tecnologia. Para tal, são organizados congressos e actividades possibilitando ao seu público a aquisição do conhecimento sobre novas tecnologias, como normas e publicações.

Enquadramento da norma IEEE 802

IEEE 802 é uma família de normas IEEE com a capacidade de fornecer as especificações necessárias para o desenvolvimento e gestão de redes locais, de área pessoal e redes de área metropolitana. Estas especificações estão alojadas nas camadas: física (PHY) e controlo de acesso ao meio (MAC) do modelo *Open Systems Interconnection* (OSI)[2]. Esta norma é aplicada aos diversos tipos de rede entre os quais se destacam:

- 802.3 - Ethernet;
- 802.06 - Redes metropolitanas;
- 802.09 - Redes locais;
- 802.11 - Redes locais sem fios (*Wireless LAN* (WLAN));
- 802.15 - Redes de área pessoal sem fios (*Personal Area Network* (WPAN)).

Tratando-se este de um projeto com recurso a uma rede sem fios de área pessoal será abordada a norma 802.15, especificamente a norma 802.15.4 que está associada a redes sem fios de baixo consumo energético.

[2]Modelo OSI - é um modelo conceptual que caracteriza e normaliza as funções de comunicação de um sistema de telecomunicações ou de computação sem ter em conta a sua estrutura interna e tecnologia subjacente.

Redes WPAN 802.15

As redes de área pessoal sem fios surgiram com a finalidade de estabelecer comunicação entre dois dispositivos na mesma rede, distanciados até 300 metros. Dentro deste conceito surgiram os seguintes conjuntos de normas:

- 802.15.1: WPAN / Bluetooth - Norma baseada na tecnologia Bluetooth, define aos especificações da camada física e da camada de controlo de acesso ao meio para a conectividade de dispositivos fixos e portáteis;
- 802.15.2: Coexistência - Norma que aborda a coexistência de redes de área pessoal sem fios (WPAN) com outros dispositivos sem fios que operam em bandas de frequências diferentes e não licenciadas como as redes locais sem fios (WLAN);
- 802.15.3: WPAN de alta velocidade - norma para redes de alta velocidade WPANs (11 a 55 Mbit/s), usada para aplicações de multimédia e que requeiram uma qualidade de serviço elevada;
- 802.15.4: WPAN de baixa velocidade - norma que surge associada à transmissão de dados de baixa velocidade numa rede WPAN. Este tipo de redes caracteriza-se pelo baixo consumo de energia o que permite à bateria durar longos períodos;
- 802.15.5: Redes de malha - norma que fornece a capacidade de dispositivos WPAN formarem redes de malha sem fios interoperáveis, estáveis e escaláveis;
- 802.15.6: Redes *Body Area Network* (BAN) - norma para comunicação sem fios de baixa potência, curto alcance, e fiável dentro da área circundante ao corpo humano, suportando grandes velocidades para a transmissão de dados;
- 802.15.7: Comunicação de luz visível - norma que fornece as especificações à camada física e de controlo de acesso ao meio para a comunicação ótica utilizando a luz visível.

2.2.1.1 Camada física

A camada física é a camada mais próxima do hardware, controla e comunica diretamente com o emissor/recetor de rádio. Entre as funcionalidades desta camada destacam-se ainda a sincronização de bits entre os dois dispositivos com base numa frequência de *clock*, o controlo da taxa de bits, a topologia física, ou seja, de que forma é que estão dispostos os nós/dispositivos na rede e o modo de transmissão. Este pode ser *simplex*, *half-duplex* ou *full-duplex*, estes modos são exemplificados através da figura 2.2.

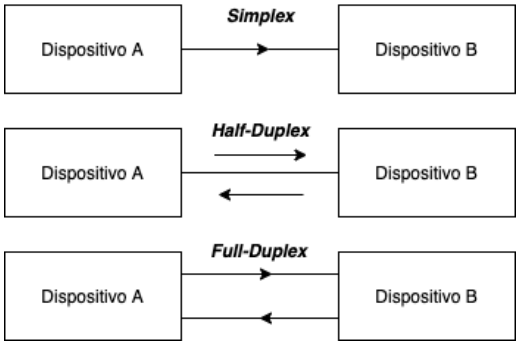


Figura 2.2: Modos de transmissão

Simplex permite a transmissão num sentido, *half-duplex* permite nos dois sentidos mas não em simultâneo , e por fim a *full-duplex* permite nos dois sentidos e em simultâneo.

Esta camada suporta três bandas de frequência: banda de 2,45 GHz (16 canais), banda de 915 MHz (10 canais), e por fim banda dos 868 MHz (1 canal). As especificações desta camada encontram-se apresentadas na tabela 2.2, tal como sugerido em [9].

	868 MHz	915 MHz	2450 MHz
Nº de canais	1	10	16
Taxa de transmissão	20 kbps	40 kbps	250 kbps
Zona	Europa	EUA	Mundo

Tabela 2.2: Especificações da camada física

Modulações

Phase Shift Keying (PSK) é a técnica de modulação digital em que a fase do sinal portador é alterada através da variação de senos e co-senos num determinado momento. Esta técnica é amplamente utilizada para redes de área pessoal e operações sem fios, juntamente com comunicação RFID (radiofrequência) e Bluetooth [10]. Na norma IEEE 802.15.4 existem duas modulações padrão deste tipo: *Binary Phase Shift Keying* (BPSK) e *Ortognal-Quadrature Phase Shift Keying* (O-QPSK).

A **modulação BPSK** é o tipo de modulação mais simples dentro da PSK, são utilizadas duas fases onde os zeros e uns de uma mensagem binária são representados por dois estados de fase diferentes no sinal da onda portadora: $\theta = 0°$ para nível lógico 1 e $\theta = 180°$ para nível lógico 0. A portadora utilizada é uma onda sinusoidal, o sinal obtido na modulação é resultante da variação de fase desta onda em função dos bits da mensagem. Dentro de um bit de duração T_b, os dois

estados de fase diferentes do sinal da portadora são dados por [11]:

$$s_1(t) = A_c \cos(2\pi f_c t), \qquad 0 \leq t \leq T_b \quad \text{para nível lógico } \mathbf{1}$$
$$s_0(t) = A_c \cos(2\pi f_c t + \pi), \qquad 0 \leq t \leq T_b \quad \text{para nível lógico } \mathbf{0}$$

A_c e f_c correspondem à amplitude e frequência da portadora, respetivamente. t é o instante em segundos, T_b é o período de bits em segundos. Considerando a_k o bit da mensagem, $s_0(t)$ é o sinal da portadora quando $a_k = 0$ e $s_1(t)$ quando $a_k = 1$.

Na figura 2.3 encontra-se representado o diagrama de blocos utilizado neste tipo de modulação [12].

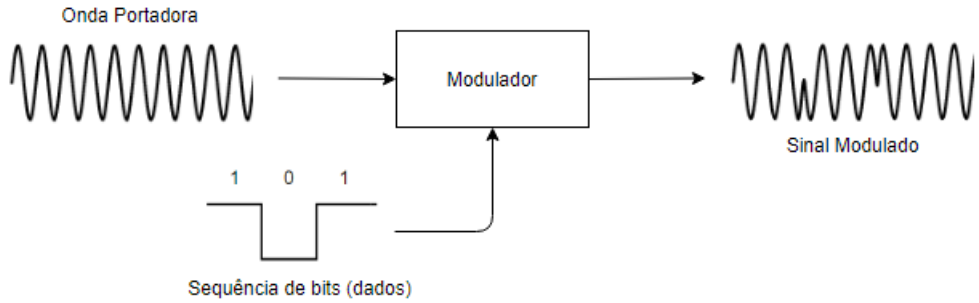
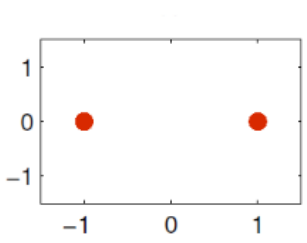
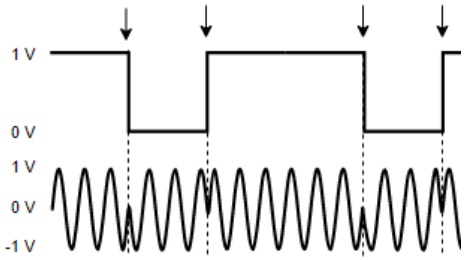


Figura 2.3: Diagrama de blocos do modulador BPSK

O diagrama de fase e quadratura, figura 2.4a, apresenta dois pontos situados exclusivamente no eixo dos x (fase). Ou seja, o sinal modulado terá componente em fase mas não em quadratura (eixo dos y) [11]. Isto é resultado da portadora utilizada na modulação, pois assim que ocorra alteração de nível lógico esta onda é invertida (figura 2.4b).



(a) Diagrama de quadratura e fase



(b) Onda modulada em função dos bits

Figura 2.4: Modulação BPSK

Assim que a mensagem chega ao recetor é feita a desmodulação. Aqui, o sinal modulado BPSK e a onda portadora são misturados com a ajuda um multiplicador e o sinal resultante desta operação é filtrado com um filtro passa-banda.

Após obter este sinal filtrado, é necessário conhecer o *bit clock*[3] para que o circuito detetor consiga produzir o sinal original da mensagem binária. Assim, recomenda-se a taxa de transmissão de bits sub-múltipla da frequência da onda portadora de forma a facilitar este processo. O diagrama de blocos do desmodulador é apresentado na figura 2.5.

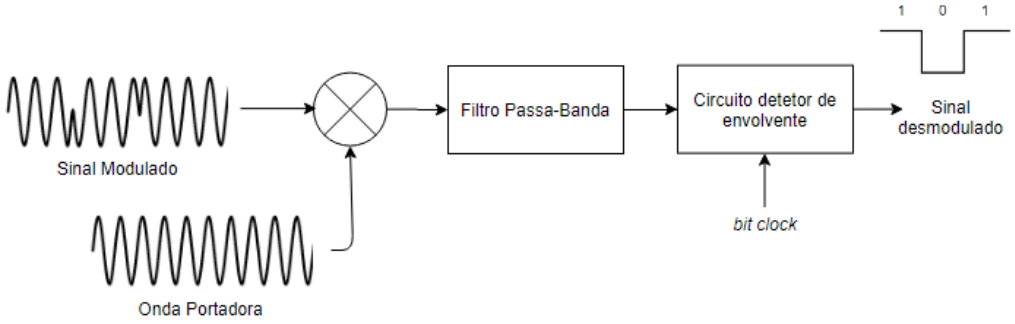
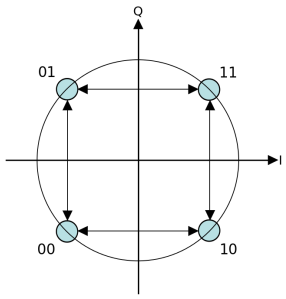
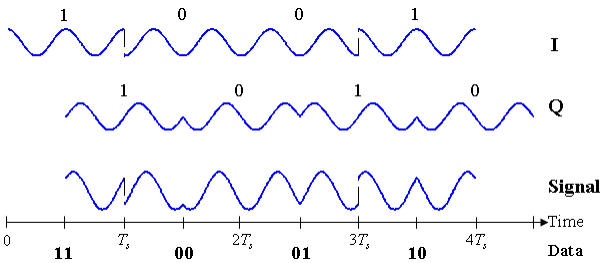


Figura 2.5: Diagrama de blocos do desmodulador BPSK

A **modulação O-QPSK** utiliza dois bits por símbolo, originando quatro resultados (valores de fase) diferentes. Quando o sinal de baixa frequência é filtrado, estas mudanças de fase resultam em grandes variações de amplitude, o que afeta negativamente a transmissão e receção de dados nos sistemas de comunicação como é o caso do ZigBee. Com a compensação da temporização dos bits ímpares e pares por um período de bit, ou meio período de símbolo, as componentes de fase e quadratura nunca se alterarão ao mesmo tempo. Como apresentado no diagrama de constelação, figura 2.6a, a mudança de fase nunca ultrapassará os 90° de cada vez. Esta técnica é preterida quando comparada com a simples QPSK, uma vez que produz flutuações na amplitude muito inferiores.



(a) Diagrama de quadratura e fase



(b) Onda modulada em função dos bits

Figura 2.6: Modulação O-QPSK

[3] *bit clock* - representa um ciclo de um sinal de onda quadrada.

Na figura 2.6b, o fluxo de dados binários está representado em baixo do eixo temporal, as duas componentes de fase e quadratura de sinal são apresentadas respetivamente na parte superior. O sinal combinado obtido na modulação é apresentado em baixo.

O modulador O-QPSK tem como entrada um sinal binário correspondente aos dados, de seguida é utilizado um conversor série-paralelo de 2 bits. De acordo com a estrutura ortogonal, é feita a diferenciação entre o bit de fase e de quadratura, sendo acrescentado um atraso de quadratura [13]. Estes bits são multiplicados pelo sinal da portadora e os sinais originários desta operação são somados um ao outro resultando no sinal O-QPSK modulado. O diagrama de blocos neste tipo de modulador é apresentado na figura 2.7 com atraso de T_b no bit de quadratura. Este atraso garante que as transições de fase nas componentes em fase e quadratura nunca ocorram simultaneamente.

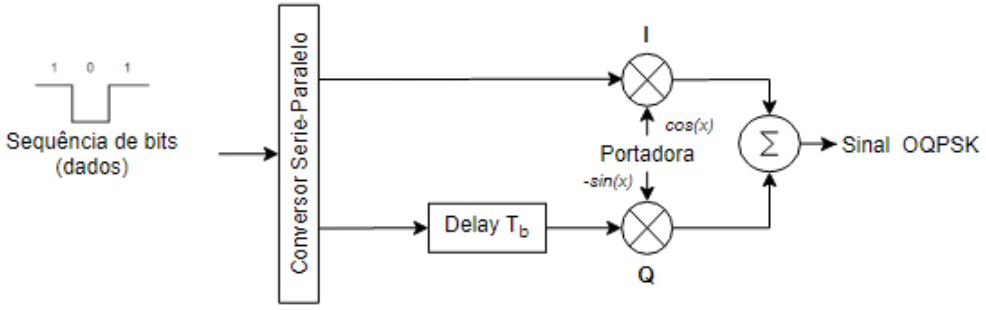


Figura 2.7: Diagrama de blocos modulador O-QPSK

O desmodulador O-QPSK recebe o sinal modulado e separa os bits de fase dos bits de quadratura, após esta divisão estes bits são multiplicados pela onda portadora. No processo inverso ao que acontece na modulação é acrescentado um atraso nos bits de fase para que estes estejam sincronizados com os de quadratura. O resultado do sinal desmodulado (bits) é obtido na saída de um conversor paralelo-série, inverso àquele utilizado na modulação. O diagrama de blocos deste desmodulador é apresentado na figura 2.8.

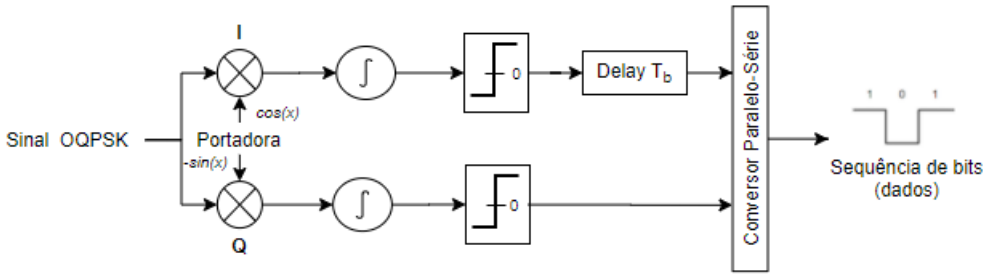


Figura 2.8: Diagrama de blocos desmodulador O-QPSK

Em suma, com a camada física pretende-se uma interface de baixo custo com elevados níveis de integração. Para tal é utilizada a técnica *Direct Sequence Spread Spectrum* (DSSS). Esta é uma técnica de modulação de difusão de espectro utilizada principalmente para reduzir a interferência global do sinal. Os bits da mensagem são modulados por uma sequência designada por sequência de propagação. Cada bit desta sequência tem uma duração muito mais curta (maior largura de banda) do que os bits da mensagem original. Ou seja, quanto menor for esta duração maior será a largura de banda do sinal DSSS resultante [14] [15].

A tabela 2.3 apresenta as especificações das diversas camadas físicas possíveis, juntamente com o tipo de modulação utilizado [15].

	868 MHz	915 MHz	2450 MHz
Nº de canais	1	10	16
Taxa de transmissão	20 kbps	40 kbps	250 kbps
Tipo de modulação	BPSK	BPSK	O-QPSK
Zona	Europa	EUA	Mundo

Tabela 2.3: Especificações da camada física com o tipo de modulação

2.2.1.2 Camada de controlo de acesso ao meio

A camada de controlo de acesso ao meio, MAC, é responsável pelo acesso à camada física para transmissão e receção de dados. É nesta camada que são especificados o tipo de dispositivos e a estrutura de tramas suportadas pela rede [16]. Tem ainda como principais funções: a gestão de *beacons*, o acesso ao meio, gestão GTS, a validação da trama, a verificação se esta foi recebida, a associação e desassociação de dispositivos. Fornece dois tipos de serviço: de dados e de gestão [9].

- O **serviço de informação** permite a transmissão e receção de unidades de dados MAC designadas por *MAC Protocol Data Units* (MPDUs) através do serviço de informação da camada física;

- O **serviço de gestão** controla o acesso aos canais RF para que a transferência de dados ocorra sem erros, suportada por mecanismos de prevenção de colisão *Carrier Sense Multiple Access - Collision Avoidance* (CSMA-CA) e *Guarantee Time Slots* (GTS).

.

O algoritmo CSMA-CA é o mais comum, cada nó escuta o meio antes de transmitir. Possui um limite máximo para o consumo de energia, caso este seja ultrapassado o nó transmissor espera durante um intervalo de tempo aleatório

e tenta novamente. Este intervalo designado por *Backoff Period* (BP) é decrementado por um contador, quando toma valor 0 escuta o meio e no caso de se encontrar livre o pacote é transmitido, caso contrário o intervalo é reposto e o processo reinicia.

A técnica GTS utiliza um nó centralizado (coordenador) que atribui instantes para que cada nó saiba quando pode transmitir. Na totalidade existem 16 *slots* que correspondem a 16 instantes a serem atribuídos. Um nó que pretenda transmitir deve enviar ao coordenador uma mensagem de pedido GTS, de seguida este irá responder com o *slot* atribuído e com o número de *slots* preenchidos.

Uma das funcionalidades implementadas com a norma 802.15.4 consiste na analise do consumo de energia associada ao canal. A ideia é conseguir obter a energia gasta em atividade, ruído e interferências num ou vários canais antes de começar a utilizá-lo. Assim, é possível otimizar o consumo de energia com a escolha de canais livres na configuração da rede. Este processo pode-se dividir em três comportamentos:

- **Energia** → realiza uma analise aos canais e reporta a energia consumida. O facto dos dados apresentados resultarem de outros nós, a tecnologia ou interferências não interferem neste valor. A transmissão só inicia quando este valor esta abaixo de um determinado limite;
- **Sentido portador (CCA)** → realiza uma analise ao meio e reporta se existem transmissões com a norma 802.15.4 . A transmissão apenas ocorre quando o canal se encontrar livre;
- **CCA + Energia** → realiza uma analise ao meio e reporta se existem transmissões com a norma 802.15.4 acima do limite especificado. Caso a resposta seja negativa o canal é utilizado.

2.2.1.3 Endereçamento de rede

No nosso dia a dia existem diversas formas de poder identificar alguém, para tal existem os números de telemóvel ou telefone, endereços de e-mail, ou até própria morada. Numa rede ZigBee acontece a mesma situação ou seja os dispositivos podem ser identificados através de um endereço. Para o envio de uma mensagem é exigido o endereço de destino, que poderá ter diferentes tipos, como mostra a tabela 2.4 [17].

Tipo	Exemplo	Endereço único
64 bits	0013A200403E0750	✓
16 bits	23F7	✓(na rede)
Identificador do nó	Tomada cozinha	Não é garantido

Tabela 2.4: Tipos de endereços

Cada dispositivo tem um número de série único e permanentemente atribuído de 64 bits. Nenhum outro dispositivo ZigBee noutra rede, ou no mundo, poderá ter esse mesmo número de série. Existe um endereço mais curto de 16 bits que é dinamicamente atribuído a cada dispositivo pelo coordenador quando este estabelece uma rede. Este endereço é único apenas dentro de uma determinada rede. Por fim, outra alternativa de endereçamento é uma pequena sequência de texto designada como identificador do nó. Isto permite que o dispositivo seja endereçado com um nome mais *user friendly*.

Endereçamento PAN

Com o principal objetivo de uma melhor gestão da rede ZigBee surgiu o endereçamento da Rede de Área Pessoal (PAN). Com isto, a rede tem a possibilidade de criar para ela própria e para a rede à montante endereços virtuais. O endereço PAN é constituído por um número de 16 bits, 65.536 endereços possíveis, sendo que cada um destes endereços possíveis tem a capacidade de aglomerar mais 65.536 endereços. Assim, e na totalidade, este tipo de endereçamento tem a possibilidade de endereçar 4.294.967.296 nós [17].

Canais

Para que a mensagem seja transmitida com sucesso para além do endereçamento é necessário que ambos os dispositivos tanto o recetor como o emissor estejam na mesma frequência. Quando o coordenador ZigBee escolhe um endereço PAN, verifica também todos os canais disponíveis, normalmente 12 diferentes, e escolhe um único para as comunicações dessa rede. Na figura 2.9 está representado um diagrama de *Venn* com canais e endereçamento.

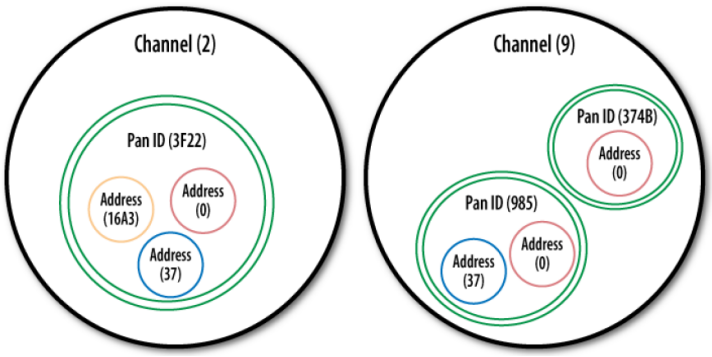


Figura 2.9: Diagrama *Venn* canais, endereçamento e PAN

2.2.1.4 Dispositivos de rede

Com base nas capacidades de processamento de dados na norma e função que desempenham, os dispositivos físicos podem ser classificados como [9]:

- ***Full Function Devices* (FFD)** → Dispositivos que executam todas as funções e operações disponíveis dentro da norma, incluindo o mecanismo de encaminhamento, tarefas de coordenação e deteção de erros.

- ***Reduced Function Devices* (RFD)** → Dispositivos que não reencaminham pacotes e têem de ser associados a um FFD. Estes dispositivos realizam tarefas limitadas, típicos exemplos são sensores e atuadores.

Quanto à lógica, os dispositivos presentes na rede podem ser classificados com três tipos de terminologias [9]:

- **Coordenador** é um dispositivo FFD único responsável por estruturar a rede, por mantê-la segura, e também pela distribuição de endereços por todos os dispositivos.

- ***Router*** é um dispositivo FFD com a capacidade de se juntar às redes existentes, de enviar e receber informação e de funcionar como mensageiro para outros dispositivos. Numa rede ZigBee podem existir múltiplos dispositivos com esta terminologia.

- ***End Device*** pode ser um dispositivo FFD ou RFD, uma vez que possui funcionalidades limitadas, consegue aderir à rede, enviar e receber informação mas apresenta a limitação de não conseguir retransmitir uma mensagem. Trata-se de um dispositivo que entra em hibernação regularmente para consumir a mínimo de energia possível. Assim, a sua utilização implica diretamente o recurso a um *router* ou coordenador para armazenar os seus dados.

2.2.1.5 Topologias de rede

Entende-se por topologia de rede a estrutura de uma rede podendo esta ser retratada física ou logicamente. Representa o meio no qual estão conectados todos os dispositivos, formando assim um canal de fluxo de informação. Cada topologia apresenta efeitos diferentes na forma como as mensagens são encaminhadas e de como estão dispostos os dispositivos. O ZigBee suporta as seguintes topologias [17] [18] :

- **Estrela** (*star*) → é composta por um coordenador e poucos *end devices*. Todos os dispositivos estão ligados a um único nó coordenador e toda a comunicação é feita através deste coordenador. Apresenta como principal desvantagem o facto de não existir nenhum caminho alternativo entre o coordenador e os dispositivos finais;

- ***Cluster Tree*** → é constituída por um coordenador, *routers* e *end devices*. Os *routers* formam uma "espinha dorsal", cada *end device* está associado a um *router* ou coordenador. No caso de um *router* estar incapacitado, os *end devices* associados a este deixam de poder comunicar com os outros dispositivos na rede;

- **Ponto-a-ponto** (*pair*) → é a topologia de rede mais simples, é composta por um *end device* ou *router* associado ao coordenador (2 nós);

- **Malha** (*mesh*) → é semelhante à topologia *Cluster Tree* com a particularidade dos *end devices* não estarem apenas conectados a um nó, ou seja, caso a transmissão num dos caminhos falhar o nó encontrará um caminho alternativo até chegar ao destino.

Na figura 2.10 encontram-se representadas as topologias possíveis numa rede ZigBee conforme abordado anteriormente.

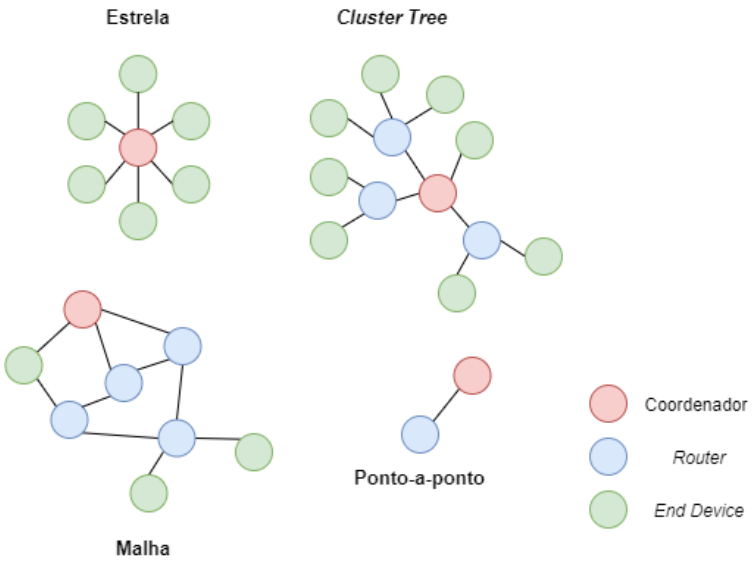


Figura 2.10: Topologias ZigBee

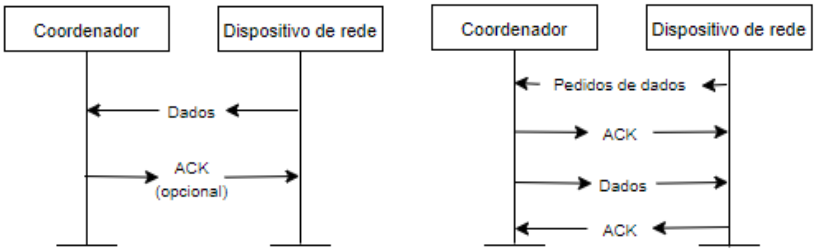
2.2.1.6 Modos operacionais

O protocolo IEEE 802.15.4 suporta dois modos operacionais que podem ser seleccionados por um nó central designado coordenador PAN, capítulo 2.2.1.4. Estes dois modos são: *Non-Beacon* e *Beacon*. Os *beacons* são considerados sinais de presença que são enviados pelos *routers* e/ou coordenador a fim de conhecer os dispositivos da rede, procedendo à sua sincronização e identificação. Estes têm uma grande importância na funcionalidade da otimização da energia associada a esta norma, permitindo prolongar a bateria dos dispositivos. O intervalo entre

beacons pode variar entre os 15.36 ms e os 251.6 s para uma transmissão de 250 kbit/s [19].

Modo *Non-Beacon*

Podem ser utilizadas duas topologias: malha e estrela. Neste modo é pressuposto que cada nó pode comunicar diretamente com os outros nós sem a necessidade do coordenador, e sem quaisquer requisitos de sincronização. Um nó pode transmitir e entrar no modo de suspensão assim que entender, seguindo a sua própria politica de consumo de energia. Todas as transmissões são suportadas com o algoritmo de prevenção de colisão CSMA-CA, enunciado em 2.2.1.2, com a finalidade de verificar se o canal se encontra livre para esta operação. Um dispositivo *non-beacon* transmite a estrutura de um *beacon* apenas como resposta a um pedido *beacon*. Os dispositivos que operam neste modo não necessitam de se sincronizar com os outros dispositivos da rede [19].

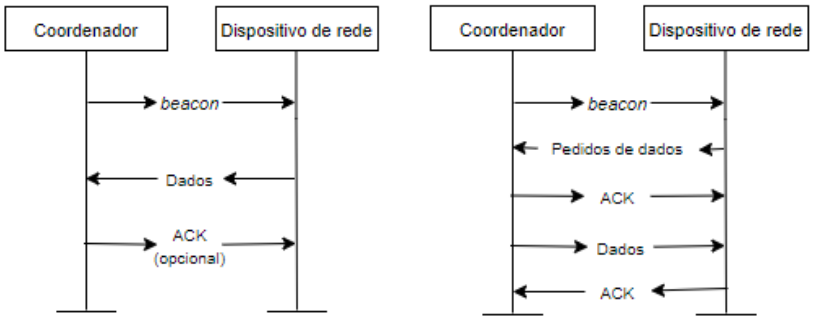


(a) Dispositivo → Coordenador (b) Dispositivo ← Coordenador

Figura 2.11: Modo *Non-Beacon*

Modo *Beacon*

O modo *beacon* centra-se no coordenador da rede, que tem como função gerir o canal de transmissão e o fluxo de mensagens a transmitir. Assim, todos os clientes sabem quando devem comunicar uns com os outros através de períodos definidos. Estes intervalos estão contidos numa *superframe* (supertrama) que indica quando é que deve ser feita a transmissão de dados e quando é que os dispositivos podem entrar no modo de suspensão. A comunicação entre os dispositivos é *full-duplex*. No contexto prático, se um cliente se conectar ao coordenador este irá procurar qualquer mensagem que seja direcionada a este dispositivo. Caso não haja qualquer mensagem pendente o dispositivo entra em suspensão e é reativado num instante especificado pelo coordenador. Assim que a comunicação termine também o coordenador entra em modo *sleep* [20].



(a) Dispositivo → Coordenador (b) Dispositivo ← Coordenador

Figura 2.12: Modo *Beacon*

2.2.1.7 Tramas

A forma de como bits na camada MAC são transmitidos, num determinado momento, definem uma estrutura de uma trama. Esta é composta pelo cabeçalho, *payload* e rodapé MAC. Neste último está presente o *Frame Check Sequence* (FCS).

Octets:2	1	0/2	0/2/8	0/2	0/2/8	0/2/6/10/14	variable	2
Frame Control	Sequence number	Destination PAN identifier	Destination address	Source PAN identifier	Source address	Auxiliar Security Header	Frame payload	Frame check sequence
		Addressing fields						
MAC header							MAC payload	MAC footer

Figura 2.13: Estrutura genérica de uma trama MAC

A estrutura genérica de uma trama na camada MAC, figura 2.13, é constituída por um campo de controlo de 2 octetos. Esta trama transporta a informação útil como o tipo de trama, e o endereçamento.

A norma utiliza o algoritmo de encriptação *Advanced Encryption Standard* (AES) com um comprimento de 128 bits (16 *bytes*). Este não é utilizado apenas para encriptar os dados que são enviados, garante também a integridade dos dados. Tal é garantido graças ao uso de um código designado por código de integridade, *Message Integrity Code* (MIC), que é anexado à mensagem. Este código pode ter diferentes tamanhos: 32, 64 ou 128 bits, no entanto é sempre utilizado o algoritmo de 128 bits. As mensagens entre os dispositivos só são aceites se os códigos MIC forem coincidentes. Quanto maior for o seu tamanho mais segura é a comunicação, porém menor será o tamanho da mensagem a ser transmitida. A segurança dos dados é garantida com a encriptação do *payload* da trama numa chave de 128 bits. Existem três campos na trama genérica MAC que estão associados à segurança [21]:

- *Frame Control*;
- *Auxiliar Security Header*;
- *Frame Payload*;

Frame Control situa-se no cabeçalho MAC, a sua estrutura encontra-se representada na figura 2.14.

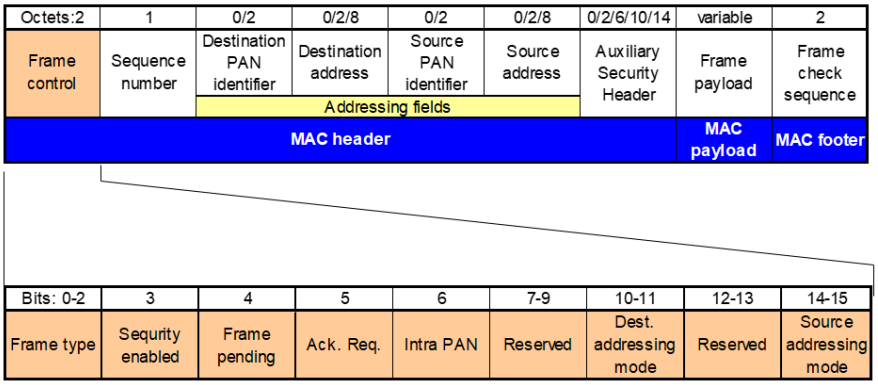


Figura 2.14: Estrutura do campo *Frame Control*

Auxiliary Security Control localizado no cabeçalho MAC, só é ativado se o sub-campo *Security Enabled* do *field* anterior estiver com o nível lógico 1. A estrutura deste campo é apresentada na figura 2.15.

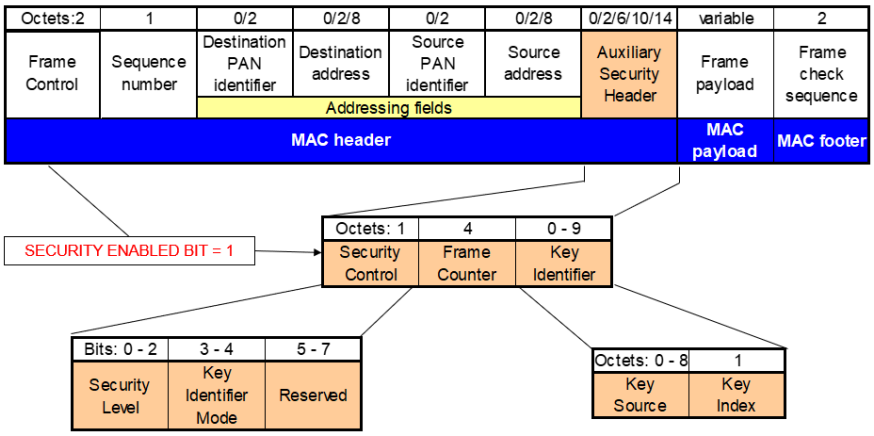


Figura 2.15: Estrutura do campo *Auxiliar Security Control*

É dividido em três partes:

- *Security Control* (1 *byte*) - especifica o tipo de proteção. Os seus dois primeiros bits (campo *Security Level*), tabela 2.5, especificam o tipo de encriptação [21].

Valor	Descrição	Encriptação	Autenticidade
0x00	Sem segurança	X	X
0x01	AES-CBC-MAC-32	X	✓
0x02	AES-CBC-MAC-64	X	✓
0x03	AES-CBC-MAC-128	X	✓
0x04	AES-CTR	✓	X
0x05	AES-CCM-32	✓	✓
0x06	AES-CCM-64	✓	✓
0x07	AES-CCM-128	✓	✓

Tabela 2.5: Valores *Security Level* do campo *Security Control*

O valor 0x00 não estabelece nenhuma encriptação, logo a autenticidade dos dados não é validada. De 0x01 até 0x03, os dados são autenticados com recurso ao código MIC. O valor 0x04 encripta exclusivamente o *payload* da mensagem, enquanto que de 0x05 até 0x07 é assegurada a confidencialidade dos dados e a sua antenticidade.

- *Frame Counter* (4 *bytes*) - contador que incrementa por cada transmissão realizada, é utilizado para não voltar a proteger o conteúdo da mensagem. Cada mensagem tem uma única sequência de identificação neste campo;

- *Key Identifier* (0 a 9 *bytes*) - define o tipo de chave que deve ser utilizada pelo remetente e o recetor. Os valores possíveis são [21]:

  - **0** → A identificação da chave é conhecida implicitamente pelo emissor e recetor, não está especificada na mensagem;
  - **1** → A identificação da chave é determinada explicitamente pelo índice da chave, *Key Index* 1 *byte*, e pela *macDefaultKeySource*.
  - **2** → A identificação da chave é determinada explicitamente pelo índice da chave, *Key Index* 1 *byte*, e pela fonte da chave, *Key Source* 4 *bytes*.
  - **3** → A identificação da chave é determinada explicitamente pelo índice da chave, *Key Index* 1 *byte*, e pela fonte da chave, *Key Source* 8 *bytes*.

Data Payload situa-se no *payload* MAC e pode ter três configurações diferentes, dependendo dos campos de segurança previamente definidos. As configurações são as seguintes [21]:

- **AES-TR** → todos os dados são encriptados utilizando a chave definida de 128 bits e o algoritmo AES. O *Frame Counter* define o ID único da mensagem, caso este contador atinja o limite máximo a camada de aplicação utiliza o contador de chaves, *Key Counter*;
- **AES-CBC-MAC** → O código de autenticidade da mensagem (MAC) é anexado no fim do *payload*;
- **AES-CCM** → É a mistura dos métodos anteriores.

No padrão IEEE 802.15.4 existem quatro tipos de trama definidos [22]:

Tipo de trama	Descrição
000	Trama *beacon*
001	Trama de dados
010	Trama ACK
011	Trama de comando MAC
100-111	Reservados

Tabela 2.6: Tipos de trama

Trama *beacon*

A trama *beacon* é utilizada para que todos os dispositivos na rede possam entrar no modo de suspensão e para proceder à reativação de um determinado dispositivo é enviada uma mensagem com um *beacon*. A estrutura deste tipo de trama é representada na figura 2.16.

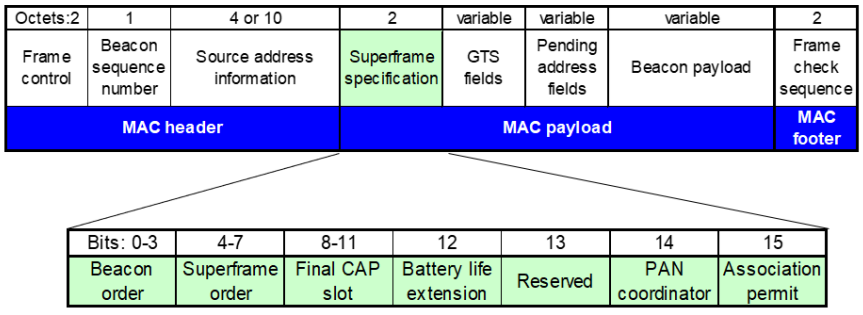


Figura 2.16: Estrutura da trama *beacon*

O campo *Superframe Specification* permite a formação de uma super trama (*superframe*) permitindo aumentar a largura de banda em certas situações, e baixa a latência da transmissão. O campo *GTS Fields* é também utilizado para que não existam atrasos na transmissão do pacote.

Tramas de dados

As tramas de dados são utilizadas por todos os dispositivos da rede, independentemente da topologia adotada. Uma vez estabelecida a ligação, a trama de dados carrega a mensagem a transmitir. A constituição deste tipo de trama é semelhante ao formato genérico e é apresentada na figura 2.17.

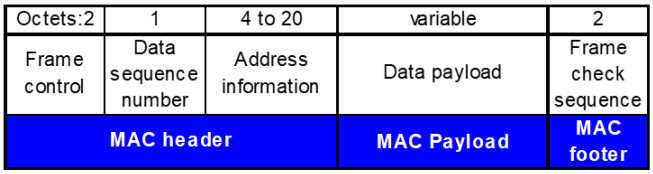


Figura 2.17: Estrutura da trama de dados

Tramas ACK

As tramas de ACK são utilizadas por todos os dispositivos da rede para confirmar que os dados foram recebidos no dispositivo de destino e a transmissão ocorreu sem erros. A constituição deste tipo de trama é apresentada na figura 2.18.

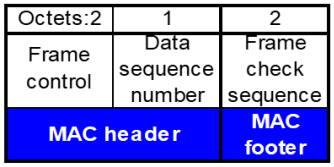


Figura 2.18: Estrutura da trama de reconhecimento

Tramas de comando MAC

As tramas de comando MAC são responsáveis pela configuração/controlo remota dos nós. São utilizadas pelo coordenador a fim de configurar os nós da rede. A constituição deste tipo de trama é apresentada na figura 2.19.

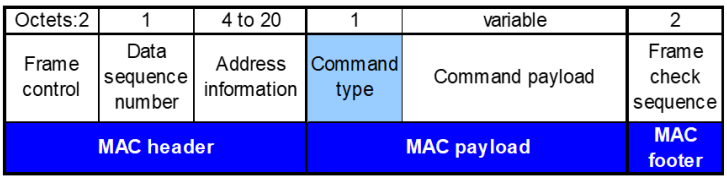


Figura 2.19: Estrutura da trama de comando MAC

O protocolo ZigBee suporta diferentes tipos de comandos que podem ser consultados na tabela 2.7.

Comando	Descrição
0x01	Pedido de associação (Tx)
0x02	Resposta da associação (Rx)
0x03	Notificação de dissociação (Tx,Rx)
0x04	Pedido de dados (Tx)
0x05	Notificação de conflito e identificação de PAN (Tx)
0x06	Notificação de orfão (Tx)
0x07	Pedido de *beacon* (Tx)
0x08	Realinhamento do coordenador (Rx)
0x09	Pedido GTS
0x0A-0xFF	Reservados

Tabela 2.7: Tipos de comandos

2.2.2 Estrutura

O modelo OSI, como já mencionado no sub capítulo 2.2.1, é um modelo concetual que caracteriza e normaliza as funções de comunicação de um sistema de telecomunicações ou de computação, sem ter em conta a sua estrutura interna e tecnologia subjacente. O seu objectivo é a interoperabilidade de diversos sistemas de comunicação com protocolos de comunicação padrão. Assim como todos os protocolos de rede, o protocolo ZigBee estrutura-se seguindo este modelo, utilizando quatro das sete camadas possíveis como apresentado na figura 2.20.

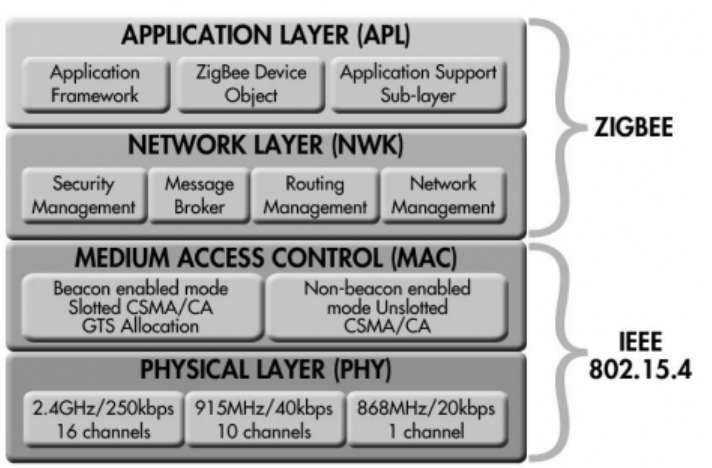


Figura 2.20: Camadas ZigBee

As camadas mais baixas, camada física e camada de controlo de acesso ao meio, já foram abordadas nos sub capítulos 2.2.1.1 e 2.2.1.2, respetivamente. A estas são adicionadas duas camadas superiores provenientes do protocolo ZigBee.

A **camada de rede** é responsável pela formação da rede, pela descoberta e

configuração de novos dispositivos, e determina a rota mais adequada desde o nó remetente até ao nó destinatário (encaminhamento). Esta camada é fundamental para o correto funcionamento da comunicação entre as camadas definidas pelo padrão IEEE 802.15.4 e as camadas superiores, de aplicação. Nesta comunicação, esta camada fornece dois serviços de comunicação: para dados e para gestão da rede. Designa-se por *Network Layer Data Entity* (NLDE) o serviço de dados, e tem como função garantir a segurança e gerar os pacotes de dados desta camada. O serviço de gestão de rede é designado por *Network Layer Management Entity* (NLME) além de inicializar a rede é também responsável pela entrada e saída de dispositivos da rede, pela configuração de novos dispositivos, pelo endereçamento destes e pela descoberta de novas rotas [15].

As diversas funcionalidades e serviços que a **camada de aplicação** oferece diferem de acordo com a finalidade de cada dispositivo de rede. Entre estas destacam-se o endereçamento, fragmentação de dados, filtros de mensagens destinados a grupos de dispositivos ou *broadcast* e a descoberta de funcionalidades de dispositivos remotos. No topo desta camada estão as aplicações definidas pelo o utilizador e que estão contidas nas sub camadas *Application Support Sublayer* (APS), *Application Framework* (AF) e *ZigBee Device Object* (ZDO). Esta última é considerada a mais importante uma vez que é responsável por descobrir novos dispositivos na rede e por estabelecer bons níveis de segurança para a comunicação [15].

Em [17] enumeram-se algumas vantagens da implementação destas duas camadas:

- ***Routing*** - Através das tabelas *routing* é definido o caminho da mensagem e são registados todos os nós percorridos desde o nó inicial até ao nó final;
- **Implementação de redes *Ad Hoc*** - redes sem fio que dispensam o uso de um ponto de acesso comum aos computadores conectados nela, permitindo que todos os dispositivos ligados à rede funcionem como *routers*, encaminhando informações que vêm de dispositivos vizinhos;
- **Malha *Self-healing*** - mecanismo que reconfigura a rede automaticamente no caso da avaria ou ausência de um *router*;

2.2.3 Segurança

O ZigBee afirma fornecer ferramentas de segurança de última geração, oferecendo aos seus utilizadores alguns dos dispositivos sem fios mais seguros da IoT. A sua segurança baseia-se na criptografia de chave simétrica, em que duas partes devem partilhar as mesmas chaves para comunicar. Utiliza assim chaves matemáticas para a encriptação de dados, que podem ser classificadas em dois

tipos: chaves de rede e chaves de ligação [17]. Se for necessário, estas podem operar em simultâneo.

As **chaves de rede** protegem os dados à medida que estes passam entre os nós da rede. Em cada nó o pacote de dados é desencriptado e encriptado para posteriormente ser enviado para o próximo "salto" na rede. Isto garante a segurança na transmissão, mas proporciona algum atraso ou latência. Outra limitação é o facto de terem de ser reservados 18 *bytes* da chave para a encriptação de dados, limitando assim o tamanho da informação a ser transmitida. Noutra perspectiva, é necessário enviar mais pacotes para transmitir a mesma quantidade de informação.

As **chaves de ligação** proporcionam uma camada adicional de proteção desde a origem até ao destino. Os dados são encriptados pelo remetente e permanecem seguros à medida que se difundem ao longo da rede. Assim, ao utilizar este tipo de segurança, os nós intermediários não conseguem ter acesso aos dados numa rede partilhada. Cada pacote só é desencriptado quando chega ao seu destino.

2.3 MQTT

Um dos protocolos mais utilizados no nível de aplicação para transferência de dados é o protocolo *Hypertext Transfer Protocol* (HTTP) que é um protocolo ASCII com base em pedidos e respostas, mas que apresenta como principal inconveniente o facto de enviar de informação desnecessária para um sistema IoT. Assim, o tamanho da informação a ser enviada/transferida pelos sistemas embebidos é considerável, e como os projetos IoT apresentam recursos de *hardware* limitados isto torna a utilização do protocolo HTTP uma solução pouco eficiente. Em alternativa, surgiu o MQTT que é um protocolo *machine-to-machine* (M2M) desenvolvido para suportar soluções no mundo do IoT. Usa a topologia de transporte de mensagens *publish/subscribe*, tornando-se extremamente leve. O MQTT opera com *payloads* em *bytes* binários e não texto como o protocolo HTTP. O formato dos pacotes de dados, (*packets*), é representado na figura 2.21.

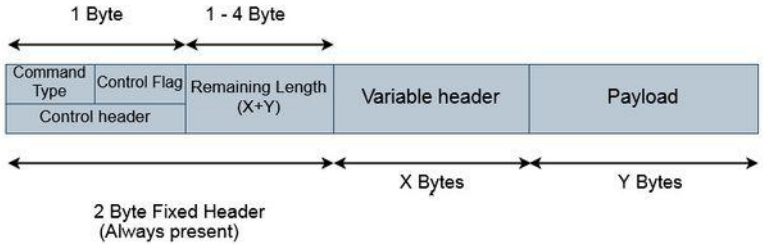


Figura 2.21: Estrutura de um *packet* MQTT

Dos dois bytes do *Fixed Header*, o primeiro *byte* corresponde ao campo de

controlo (8 bits) que está dividido em dois sub campos de 4 bits. Os primeiros 4 bits mais significativos indicam tipo de comando, cuja a lista pode ser consultada na tabela 2.8. Os restantes 4 bits pertencem ao campo *Control Flag*, e são exclusivamente utilizados pelo comando *publish*:

- O bit da posição 0 reporta se a mensagem publicada foi retida;
- Os bits da posição 1 e 2 são utilizados para selecionar a qualidade do serviço;
- O terceiro bit reporta se a mensagem é duplicada.

O segundo *byte* tem comprimento variável, e corresponde à soma dos comprimento dos campos *Variable Header* e *payload*. Este *Variable Header* não está presente em todos os pacotes MQTT, e é utilizado por comandos e mensagens que servem para fornecer informação adicional. O campo final, *Payload*, contém os dados que serão enviados. Este campo é opcional e varia com o tipo de pacote.

Comando	**Valor**	**Direção**	**Descrição**
Reservado	0000 (0)	X	Reservado
CONNECT	0001 (1)	Cliente→Servidor	Pedido de conexão
CONNACK	0010 (2)	Servidor→Cliente	Pedido ACK
PUBLISH	0011 (3)	Servidor↔Cliente	Publica mensagem
PUBACK	0100 (4)	Servidor↔Cliente	Publica ACK
PUBREC	0101 (5)	Servidor↔Cliente	Publica recebido
PUBREL	0110 (6)	Servidor↔Cliente	Publica publicado
PUBCOMP	0111 (7)	Servidor↔Cliente	Publica concluído
SUBSCRIBE	1000 (8)	Cliente→Servidor	Pedido de subscrição
SUBACK	1001 (9)	Servidor→Cliente	P. de subscrição ACK
UNSUBSCRIBE	1010 (10)	Cliente→Servidor	Cancela subscrição
UNSUBACK	1011 (11)	Servidor→Cliente	Cancela subscrição ACK
PINGREQ	1100 (12)	Cliente→Servidor	Pedido PING
PINGRESP	1101 (13)	Servidor→Cliente	Resposta PING
DISCONNECT	1110 (14)	Cliente→Servidor	Cliente desconectado
Reservado	1111 (15)	X	Reservado

Tabela 2.8: Tipos de comandos MQTT

Publish/Subscribe

O sistema *publish/subscribe* consiste na publicação de uma mensagem proveniente de um dispositivo num determinado tópico. Após esta publicação, a mensagem só será entregue aos dispositivos que tenham subscrito esse mesmo tópico. O dispositivo que envia a mensagem não tem conhecimento se a mensagem chegou ou não ao(s) dispositivo(s) pretendido(s). Os tópicos são representados por strings separados por *slashes* e cada *slash* representa uma hierarquia. Na figura 2.22 é apresentado um exemplo prático.

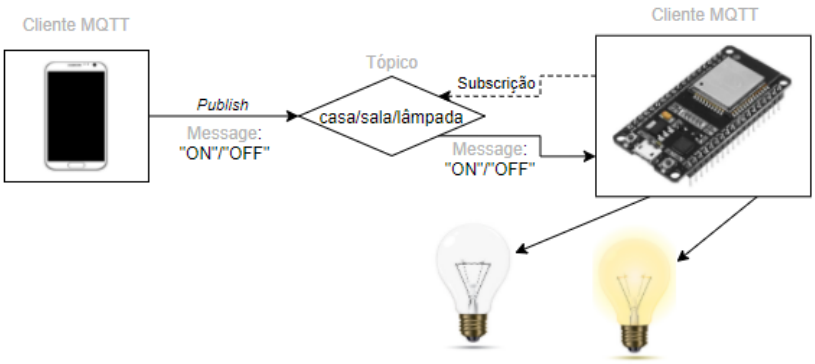


Figura 2.22: Exemplo de aplicação método *publish/subscribe*

Broker

O *broker* é responsável por receber e filtrar de acordo com os tópicos todas as mensagens vindas dos diferentes dispositivos e entregá-las aos dispositivos subscritos a esse mesmo tópico. Dependendo do tópico a que a mensagem se destina, o *broker* irá enviar exclusivamente para todos os dispositivos subscritos a esse tópico. Na prática, o *broker* pode ser instalado tanto num RaspBerry Pi como num computador pessoal ou servidor. Existem diversos MQTT *brokers* públicos e gratuitos, que permitem maior facilidade na implementação de projetos deste tipo, sem necessidade de grande investimento. Na figura 2.23 é apresentado um exemplo prático de um *broker*.

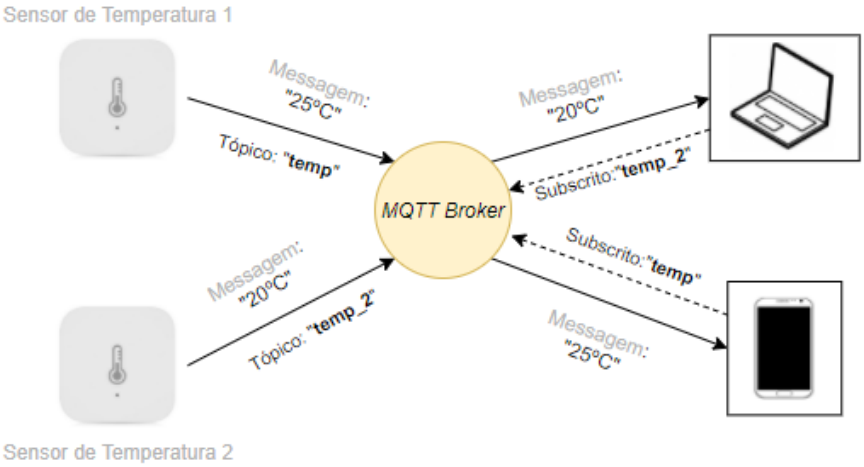


Figura 2.23: Exemplo de um *Broker*

2.4 Controladores de Domótica Residencial

Os fabricantes de controladores de domótica residencial procuram acompanhar a evolução do protocolo ZigBee. Destacam-se entre estes fabricantes a Fibaro, Vera, Xiaomi e a Zipato, que oferecem diversas alternativas de escolha e *software* de fácil interação para o cliente. Em alternativa a estes equipamentos, existe *software open-source* que possibilita o desenvolvimento de controladores sem necessidade de recorrer a um fabricante. Para tal, basta recorrer a uma antena e um computador.

Fibaro

A Fibaro apresenta uma das melhores opções de controlador de domótica residencial, o Home Center 3. No entanto este é bastante dispendioso, custa cerca de 599,00€. Esta é a versão mais recente de duas já lançadas para o mercado, porém só esta última é que suporta a tecnologia ZigBee. A Fibaro oferece as mais recentes tecnologias de proteção de dispositivos. Disponabiliza funcionalidades que mais nenhum controlador consegue dar graças ao seu processador de 4 núcleos com frequência de *clock* de 1,2 GHz e à sua memória RAM de 2GB [23]. Apresenta uma grande interoperabilidade uma vez que permite utilizar equipamentos de outros fabricantes e com tecnologias diferentes. Outra características deste equipamento é a sua durabilidade já que se baseia em componentes de alto desempenho.

Figura 2.24: Home Center 3

Vera

A Vera é uma das marcas mais conceituadas neste sector, oferece diversas alternativas no entanto só duas é que apresentam a capacidade de suportar equipamentos de tecnologia ZigBee. Estes equipamentos são o *VeraPlus* e o *VeraSecure*, custam respetivamente 189,95€ e 299,99€. O controlador *VeraPlus* opera nas tecnologias mais conhecidas, como Wi-Fi, ZigBee, e Bluetooth. Permite a inclusão de mais de 200 dispositivos na rede e apresenta como vantagem, no caso da Internet falhar, o controlo local, ou seja, não há a necessidade de uma *cloud* [24]. O *VeraSecure*, quando comparado com o anterior, tem a capacidade de operar em 3G/4G. Estes *gateways* têm a possibilidade de criação de cenários utilizando a linguagem LUA / LUPP e diversas integrações como o *Google Assistant*, a *Alexa* e aplicações para *smartphone* (iOS e Android).

(a) VeraPlus (b) VeraSecure

Figura 2.25: Controladores Vera

Software Open-Source

- O ***Home Assistant*** é um *software* de domótica totalmente gratuito e *open-source* para controlar casas inteligentes. Desenvolvido em Python, tem como principal funcionalidade o controlo local e a privacidade. Tem uma vasta gama de dispositivos e *plugins* com integrações para diferentes tecnologias, sistemas e serviços de IoT. Permite controlar ações localmente e remotamente, como o controlo de iluminação, climatização, sistemas de entretenimento e dispositivos genéricos. Podem ser desenvolvidos cenários controlados por *scripts*, comandos de voz, aplicações móveis ou controladas através da interface de utilizador *Home Assistant*;
- O ***Open Home Automation Bus*** **(OpenHAB)** é um *software open-source* de domótica desenvolvido em Java. É controlado localmente e tem compatibilidade com dispositivos e serviços de diferentes fabricantes. Permite criar a criação de regras tal como o *Home Assistant*, no entanto a sua interface é considerada menos *user-friendly*. É de salientar que suporta a grande maioria dos protocolos utilizados neste setor e permite ao utilizador a instalação de *plugins* que permitem a integração com IFTTT.

Capítulo 3

Projeto

Neste capítulo é feita a descrição do projeto, com a análise dos seus componentes físicos. Posteriormente é abordada a sua arquitetura e por fim, são apresentados o diagrama geral do sistema e uma estimativa de custos.

Entende-se por *hardware* os equipamentos físicos a serem utilizados na implementação do protótipo e por *software* as linguagens de programação, sistemas operativos e programas utilizados.

Na escolha de *hardware* é tido em conta o menor investimento possível, permitindo um sistema fiável e capaz de executar todas as funcionalidades que um controlador de domótica comum oferece.

Relativamente ao *software* a usar, existem diversas alternativas gratuitas para a concretização da solução. No entanto, a escolha é feita com base na facilidade da implementação e na integração da lógica associada.

Em suma, o controlador a implementar deverá fazer uso de uma placa de desenvolvimento para a comunicação e análise do estado de dispositivos ZigBee. Como tal, será criada uma interface gráfica com a finalidade de controlar os diversos equipamentos e fornecer informações relativas ao seu estado.

Tendo em conta, que uma habitação poderá dispor de mais do que um habitante, deverá também ser desenvolvido um servidor local (nesta placa de desenvolvimento) responsável por apresentar aos clientes conetados os dados do sistema.

3.1 Hardware

Para a implementação deste controlador será necessário recorrer a uma placa de desenvolvimento. Dentro da gama, as placas mais fiáveis e viáveis, com mais implementações no contexto de projetos IoT são dos fabricantes RaspBerry Pi e Arduino. No entanto, estas alternativas apenas suportam comunicação por Wi-Fi e Bluetooth.

Para um sistema deste tipo que a principal finalidade é suportar a tecnologia ZigBee será necessária uma antena externa ou um *dongle*. Existem no mercado variadíssimas opções dentro destes equipamentos que permitem adicionar a um simples computador pessoal a funcionalidade de suportar outro tipo de comunicação.

3.1.1 Placa de Desenvolvimento

A escolha da placa de desenvolvimento recaiu sob um RaspBerry Pi 4 Model B. Dentro da gama RaspBerry Pi existem diversas opções como as versões 2 e 3. No entanto, opta-se pelo modelo mais recente. Este é um computador pequeno e de baixo custo que oferece diversas funcionalidades para projetos IoT. A sua arquitetura de 64 bits permite executar *software* como se de um computador pessoal se trata-se. Suporta distribuições Linux, o que permite a interoperabilidade com a grande maioria do *software*. Na tabela 3.1 são apresentadas as suas principais especificações.

Tabela 3.1: Caraterísticas RaspBerry Pi 4 Model B

Chip	Broadcom BCM2711
Processador	Cortex-A72 (ARM v8) 64-bit
Nº de núcleos	4
Frequência de *clock* CPU	1.5GHz
Placa Gráfica	VideoCore VI
Memória RAM	2GB, 4GB ou 8GB
Comunicação *Wireless*	802.11ac 2.4 GHz/5.0 GHz
Bluetooth	5.0 (BLE)
Consumo	600 mA

Esta placa é amplamente utilizada em muitas áreas graças ao seu baixo custo e elevada portabilidade. Na figura 3.1 é apresentada a placa de desenvolvimento utilizada na concretização da solução.

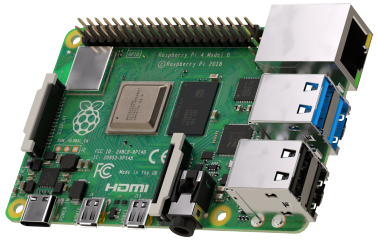


Figura 3.1: RaspBerry Pi Model 4 B

3.1.2 Periférico externo ZigBee

Após selecionada a placa de desenvolvimento, procedeu-se à escolha da antena ou *dongle* ZigBee. O principal parâmetro a ter em conta é a capacidade de comunicar com o RaspBerry Pi. Apesar da possibilidade da comunicação através dos *General Purpose Input/Output* (GPIO) opta-se pela comunicação *Universal Serial Bus* (USB).

Dentro das opções e tendo em conta o menor investimento possível a solução encontrada foi o USB *dongle* CC2531. É um dispositivo que quando conectado a computadores pessoais permite suportar comunicações com a norma IEEE 802.15.4, incluindo o ZigBee.

Normalmente, já vem com uma versão de *firmware* previamente instalada, porém, existe a possibilidade do utilizador instalar a versão que lhe convém. Para tal, é necessário o *CC Debugger*, que é um equipamento principalmente utilizado para programação *flash* e de *software*.

Nesta implementação é necessário o seu uso uma vez que a versão de *firmware* utilizada é diferente da original. Na figura 3.2 são ilustrados o periférico ZigBee utilizado e o *CC Debugger* utilizado na sua configuração.

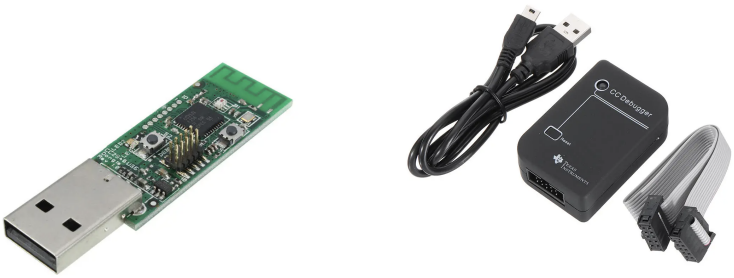


Figura 3.2: CC2531 e *CC Debugger*

3.2 Software

Dentro do tópico *software* opta-se pelo desenvolvimento de um servidor local, hospedado no RaspBerry Pi, e uma interface gráfica *online*. É utilizada a biblioteca *zigbee2mqtt* que "traduz" o protocolo ZigBee para um protocolo de mensagens, MQTT. O servidor deve ter a capacidade de interagir com esta biblioteca a fim de controlar os dispositivos ZigBee em função das funcionalidades executadas na interface gráfica pelo o utilizador.

3.2.1 Sistema Operativo

O RaspBerry Pi oferece diversas alternativas para a escolha dos sistemas operativos a utilizar, é recomendada a utilização de sistemas desenvolvidos pela empresa RaspBerry, como o *Debian*. No entanto, este sistema tem como inconveniente o facto de ainda não suportar a arquitetura de 64 bits utilizada pelo processador do RaspBerry Pi em questão. Entre as soluções que mais se destacam e que são mais recorrentes nestes equipamentos esta o *Ubuntu*. A versão utilizada é a 20.04 LTS (*Focal*), optou-se pela escolha desta em detrimento de outras por ser mais leve e oferecer as funcionalidades necessárias para a implementação do protótipo final.

3.2.2 zigbee2mqtt

O *zigbee2mqtt* é uma biblioteca/módulo que permite converter as comunicações entre dispositivos ZigBee em mensagens, através do protocolo MQTT. Este é dividido em três partes:

- O módulo *zigbee-herdsman* liga o adaptador ZigBee e fornece uma API para as camadas mais altas do protocolo. Neste caso, para o *hardware* da *Texas Instruments*, o *zigbee-herdsman* utiliza a API de monitorização e teste *TI zStack* para comunicar com o adaptador;

- O módulo *zigbee-herdsman-converters* lida com o mapeamento dos diversos dispositivos para os *clusters* ZigBee que eles suportam. Os *clusters* ZigBee são as camadas do topo do protocolo que definem a forma de como os dispositivos comunicam uns com os outros através da rede ZigBee;

- O módulo geral *zigbee2mqtt* gere o *zigbee-herdsman* e converte as mensagens ZigBee para mensagens MQTT.

A arquitetura deste módulo pode ser consultada na figura 3.3.

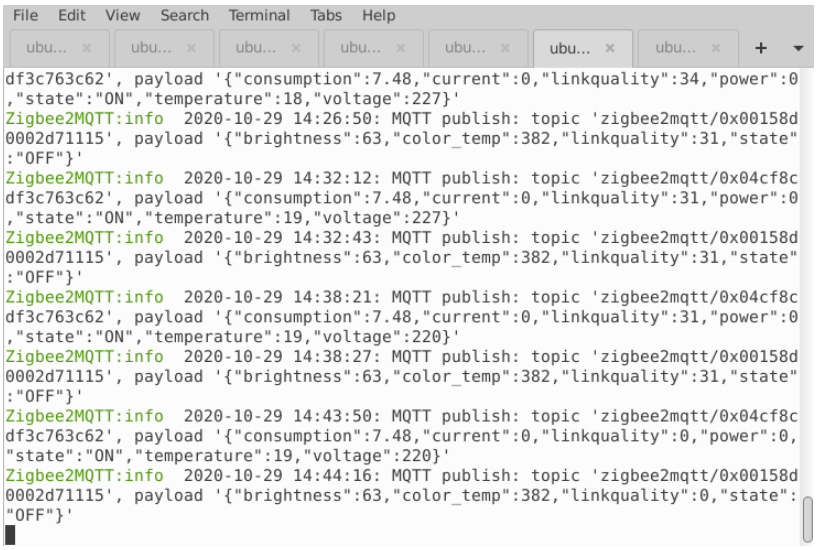


Figura 3.3: Arquitetura *zigbee2mqtt*

3.2.3 Servidor Local

Nas aplicações deste tipo, existe a possibilidade dos servidores serem hospedados em plataformas externas de forma a poupar recursos de processamento, mas isto implica o inconveniente destas aplicações estarem dependentes do funcionamento destas plataformas. Assim, opta-se por hospedar o servidor localmente, no RaspBerryPi, garantindo que este fica independente de eventuais anomalias externas. Este servidor deve:

- Ser um cliente MQTT, estando em permanente contacto com o cliente *zigbee2mqtt* de forma a receber os dados dos diversos dispositivos ZigBee;
- Alojar estes mesmos dados numa base de dados, garantindo a sua fácil interpretação;
- Atualizar constantemente a interface gráfica de acordo com os dados recebidos.

Para o desenvolvimento opta-se pela linguagem de programação JavaScript, especificamente Node.js. O Node.js é um *software open-source* que pode ser instalado no sistema operativo escolhido e que permite a implementação de servidores Web e ferramentas de rede utilizando o JavaScript. Suporta módulos que permitem adicionar diversas funcionalidades à aplicação. Neste caso em concreto, são necessários *frameworks* que permitam estabelecer a comunicação entre o cliente (interface gráfica) e o servidor, possibilitando a gestão de uma base de dados, permitindo ainda ao servidor comportar-se como um cliente MQTT, como abordado nos pontos anteriores.

3.2.4 Base de dados

A base de dados será responsável por armazenar e organizar os dados provenientes do servidor. Para além deste objetivo prioritário, é esperado que o utilizador tenha a capacidade de anexar os diversos dispositivos a cada divisão da residência.

Esta base de dados pode ser hospedada num servidor externo, mas por opção decide-se recorrer ao armazenamento local. Os dados são armazenados exclusivamente no RaspBerry Pi. Esta poderia ser considerada uma limitação do sistema uma vez que existe a possibilidade de atingir a capacidade máxima de armazenamento do dispositivo. No entanto, esta escolha teve como fundamento o tamanho dos documentos que irão ser armazenados e o facto da maioria dos serviços de armazenamento *online* ter um custo associado.

A escolha do *software* recai sobre o MongoDB, suporta *Ubuntu* e é o *software open-source* mais popular em aplicações deste tipo. É de fácil interpretação e permite o armazenamento através de ficheiros do tipo JSON. Como inconveniente surge o facto da versão que irá ser utilizada para a implementação deste projeto, pois suporta apenas sistemas de 64 bits, para RaspBerry Pi's mais antigos a instalação não é compatível.

3.2.5 Interface Gráfica

Tal como acontece com o servidor, este tipo de interface pode ser hospedada também num servidor externo. No entanto, caso este servidor tenha algum tipo de anomalia o sistema é afetado diretamente, impossibilitando o utilizador de controlar a residência. Assim, opta-se por hospedar esta interface de duas formas: localmente e num servidor externo. Esta é a única componente do sistema que o utilizador terá acesso, aqui poderá:

- Adicionar e remover dispositivos da rede;
- Adicionar e remover divisões da residência;
- Controlar as funcionalidades dos diversos dispositivos;
- Adicionar e remover gráficos que esbocem os dados dos dispositivos em função do eixo temporal;
- Adicionar e remover regras/cenários que permitam o controlo dos diversos dispositivos;

A linguagem de programação escolhida para a sua elaboração é equivalente à do servidor, JavaScript, já que oferece recursos que facilitam a implementação de páginas *Web*. O React é a biblioteca mais conceituada para este efeito, é

open-source e é utilizada em grande parte dos *Websites*, como é o caso da Netflix, Facebook e Instagram.

3.3 Diagrama do sistema

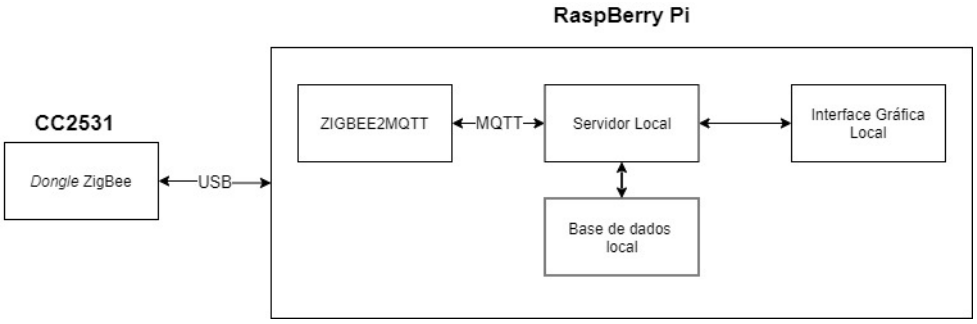


Figura 3.4: Diagrama do sistema

3.4 Estimativa de custos

Este é um pontos mais importantes a ter em consideração no desenvolvimento de um projeto IoT.

A escolha da tecnologia ZigBee permite ao utilizador ter mais facilidade em adquirir equipamentos em termos monetários quando comparado com outros protocolos de comunicação. A escolha do *software* não implicou custos acrescidos ao projeto visto ser totalmente gratuito. Relativamente ao *hardware*, o sistema mínimo para executar este projeto e os respetivos custos são apresentados na tabela 3.2.

Equipamento	**Custo**
Raspberry Pi 4 Modelo B 2GB	41,75€
Dongle ZigBee (CC2531)	5€
CC Debugger + Cabo de *download*	5€
Total	**51,75€**

Tabela 3.2: Estivativa de custos

É de mencionar, como frisado no sub-capítulo 3.1.2, que para a primeira utilização do *dongle* CC2531 é necessário proceder à configuração através do *CC debugger*, cujo *firmware* é carregado neste dispositivo através de um cabo de *download*. É recomendado ainda a utilização de uma extensão USB para a comunicação com o RaspBerry Pi, a fim de afastar o máximo possível o *dongle* de eventuais interferências provenientes deste equipamento.

Os valores apresentados na tabela 3.2 representam os preços atuais do mercado, sendo o valor do RaspBerry Pi de uma loja física nacional. Os restantes dois componentes de uma loja *online*. Estes preços variam de acordo com o mercado.

Capítulo 4

Implementação

Este capítulo apresenta de forma detalhada os passos que foram seguidos até a obtenção do protótipo funcional. É dividido em dois conceitos designados por *back-end* e *front-end*. O *back-end* surge associado ao *software* de suporte, ou seja, a parte de processamento de dados. Já o *front-end* corresponde ao *software* que é desenvolvido para interface com o utilizador final.

4.1 Base de dados

Como mencionado no capítulo 3.2.4, o *software* escolhido para suportar a base de dados é o MongoDB, na versão 4.4. Aqui, são armazenados os dados relativos aos dispositivos, divisões, gráficos e cenários. Trata-se de uma base de dados local que pode ser transferida para o RaspBerry Pi através do terminal com o comando:

```
ubuntu@ubuntu$ sudo apt-get install -y mongodb-org
```

Com este comando a versão instalada é a mais recente. Para efeitos de implementação, é necessário proceder à configuração desta distribuição.

O pacote oficial do MongoDB inclui um ficheiro de configuração que está localizado no diretório `/etc/mongod.conf`. Em caso de alteração, se a aplicação estiver a ser executada, é necessário proceder ao seu reinício para que a configuração seja aplicada. Neste ficheiro é possível configurar parâmetros como a localização dos dados e a sua replicação.

Um dos pontos essênciais nesta fase da implementação é o conceito de replicação, em inglês *replication*. Como o própria definição de réplica indica, uma *replica set* consiste num grupo de processos que mantêm o mesmo conjunto de dados. No MongoDB, assim como noutro *software* deste tipo, a replicação é utilizada para garantir a segurança dos dados.

A arquitetura de uma *replica set* é constituída por um nó primário e os restantes secundários, como apresentado na figura 4.1.

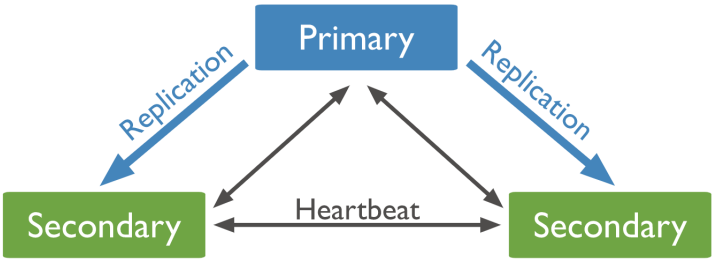


Figura 4.1: Arquitetura de uma *replica set*

Em [25] é apresentado o modo de funcionamento de uma *replica set*:

- Só o nó primário é que recebe operações de escrita;
- Todos os nós recebem leituras;
- Quando há operações de escrita no nó primário, este replica para todos os nós secundários;
- Os nós monitorizam-se uns aos outros (*Heartbeat*);
- Em caso de falha do nó primário, um secundário assume a sua posição e passa a receber escrita;

Existem diversas formas de criar uma *replica set*, sendo que a mais comum utiliza o comando de inicialização do *software*:

```
ubuntu@ubuntu$ mongod --port 27017 --replSet rs0
```

1. `mongod` é a designação do pacote a ser executado;
2. `--port 27017` evoca a porta do RaspBerry Pi cuja distribuição irá ser executada;
3. `--replSet rs0` cria uma replicação designada por `rs0`.

Em alternativa, é possível configurar esta base de dados para iniciar sempre com uma replicação. Para tal, é necessário aceder ao ficheiro de configuração mencionado neste sub-capítulo. Aqui, existe um campo "Replication" que permite designar uma *replica set*. Assim, com a inicialização da distribuição também é executada esta replicação.

Após a conclusão dos passos anteriores, é necessário manter o *software* a executar para proceder à inicialização da réplica. O comando `mongo` abre a consola desta distribuição e de seguida é necessário inicializar a réplica, `rs.initiate()`.

Neste projeto são utilizados quatro documentos que são responsáveis pelo armazenamento de dados, designados por coleções. Estas coleções são apresentadas na tabela 4.1.

Coleção	Descrição
Devices	Dados relativos aos dispositivos;
Measurements	Dados relativos ao estado dos dispositivos;
Divisions	Dados relativos à divisão;
Graphics	Dados referentes aos gráficos;
Scenes	Dados associados aos cenários;

Tabela 4.1: Coleções da base de dados

4.2 Servidor

O servidor requer a instalação do Node.js, que pode ser instalado no RaspBerry Pi através do terminal com o comando:

```
ubuntu@ubuntu$ sudo apt install nodejs
```

Após a conclusão da instalação, é necessário criar um diretório constituído por um ficheiro JavaScript, `app.js`. Aqui será adicionada toda a lógica do servidor. Tendo em conta que esta será uma aplicação composta por vários módulos, para facilitar o processo de instalação da aplicação deve-se executar o comando `npm init`. Com esta execução irão ser solicitados ao utilizador vários campos como o nome, a versão, descrição e o designação do ficheiro principal, neste caso `app.js`. Após submetidas as respostas será gerado um ficheiro designado por `package.json`.

Para iniciar a aplicação deve-se executar o comando `npm start`, ou em alternativa `node app.js`. Note-se que o primeiro irá procurar no diretório o ficheiro denominado como principal.

A fim de criar um servidor na máquina opta-se pelo módulo `express.js`, pois é através deste que serão estabelecidas as comunicações com os diversos clientes.

Recomenda-se o uso do campo `--save` na instalação de qualquer módulo, já que assim ficará registado no ficheiro `package.json` como pacote requisitado para o correto funcionamento da aplicação. O comando para a instalação é o seguinte:

```
ubuntu@ubuntu$ npm install --save express
```

No excerto de código A.1, localizado no apêndice A deste documento, é exemplificada a implementação de um servidor HTTP seguindo esta estrutura. No

exemplo, a aplicação é servida na porta 3000 da máquina. Ao efetuar um pedido `GET` ao endereço `http://localhost:3000` o utilizador terá como resposta a mensagem "`Hello World!`".

4.2.1 Comunicação com o cliente

Para esta comunicação opta-se pelo uso de *sockets*. Estes são utilizados em variadíssimas aplicações, principalmente no contexto de *chats*, já que permitem comunicação em tempo real e bidirecional. Desta forma poupam-se recursos de processamento, uma vez que não são necessários pedidos periódicos entre o servidor e os respetivos clientes.

O Node.js oferece um módulo que suporta esta técnica, `Socket.IO`. Este é dividido em duas bibliotecas distintas, uma para o servidor e outra para os respetivos clientes. Para a sua instalação deverá ser executado o seguinte comando:

```
ubuntu@ubuntu$ npm install --save socket.io socket.io-client
```

No excerto de código A.2 é apresentado um servidor com a capacidade de comunicar via *socket*. Repare-se que esta comunicação é estabelecida com recurso ao servidor HTTP. Quando é conectado um cliente, é impressa a mensagem "User Connected!", no caso inverso "User Disconnected".

Já no excerto A.3 é apresentada a estrutura de um cliente suportado por esta técnica. Assim que a comunicação é estabelecida com o servidor é impresso na consola "true", garantindo que a comunicação é bem sucedida, em caso de término é impressa a mensagem "false".

O fluxograma da lógica da comunicação entre o servidor e o cliente é apresentado na figura 4.2.

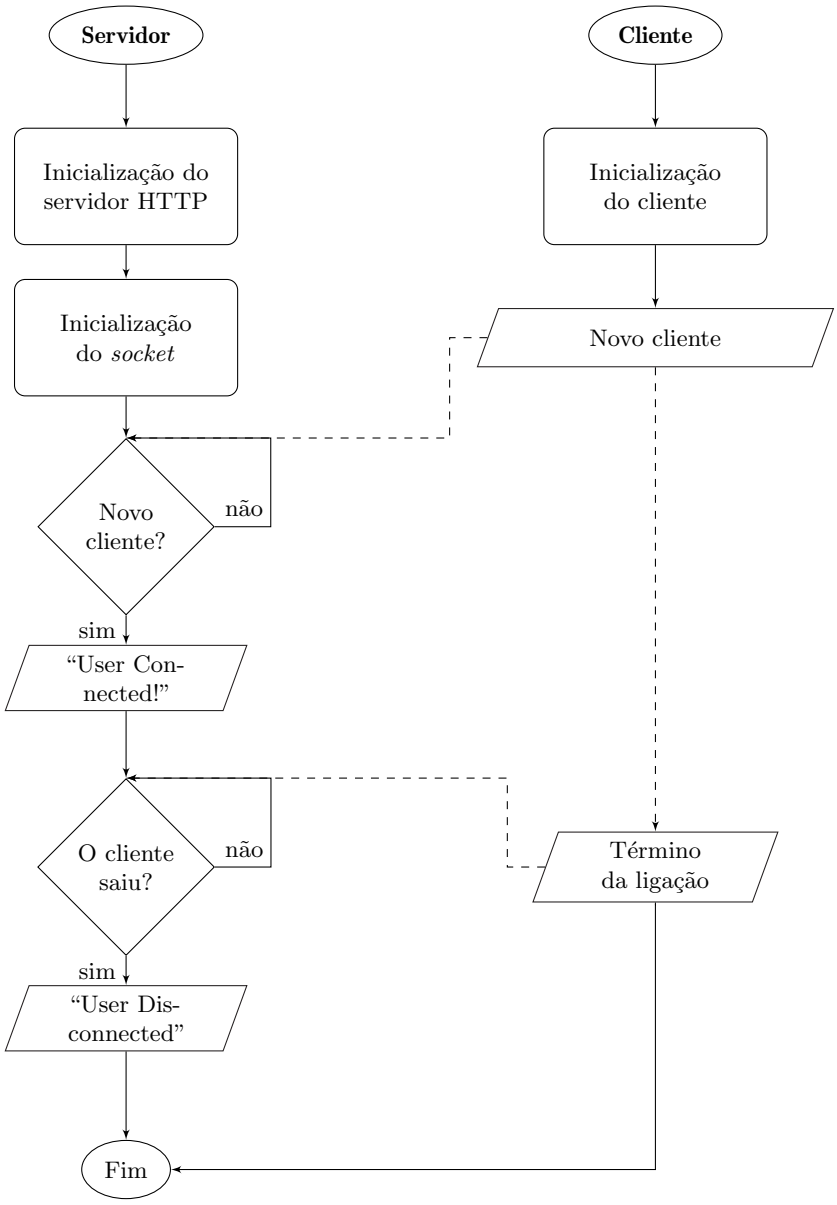


Figura 4.2: Fluxograma da comunicação via *socket* entre o servidor e o cliente

Na comunicação via *socket* existem dois métodos distintos, um de escuta designado por `socket.on()` e outro de escrita `socket.emit()`. O excerto A.4 apresenta um exemplo de aplicação com a escrita de uma mensagem num canal do *socket*. O servidor fica à escuta neste canal, conforme o excerto A.5, quando recebida a mensagem é impressa na consola.

4.2.2 Comunicação com a base de dados

O servidor deverá também proceder à leitura e escrita na base de dados, como tal, é necessário proceder à configuração das diferentes coleções (tabela 4.1). O `mongoose` é um módulo do Node.js que é utilizado para conectar a aplicação com a base de dados MongoDB. Este permite manipular estas coleções de forma a criar, editar e eliminar documentos. Estes irão receber dados de diversos tipos, que devem ser explícitos em ficheiros do tipo JavaScript de forma a serem interpretados pelo `mongoose`. Para a instalação, de forma análoga aos anteriores, é necessário executar o seguinte comando:

```
ubuntu@ubuntu$ npm install --save mongoose
```

No excerto A.6, é apresentada uma estrutura exemplo para a coleção *Devices*. Neste caso, são apenas definidas variáveis do tipo `String`. Só são registados na base de dados valores conforme o tipo declarado neste ficheiro. Ou seja, na tentativa de registo em qualquer campo da coleção *Devices* dados do tipo `Number` a operação é ignorada. Isto previne eventuais erros na leitura de dados.

Assim, para cada coleção é necessário um ficheiro deste tipo. Após a sua implementação é necessário proceder à importação no servidor para que este consiga aceder e gerir as coleções. No entanto, é necessário conectar o servidor à base de dados. O excerto de código A.7 representa esta operação. Quando a conexão é estabelecida é impressa uma mensagem de sucesso na consola do servidor.

Conforme abordado no capítulo 3.2.4, o uso de *replica set's* teve como fundamento poupar recursos de processamento e garantir a segurança dos dados. Através desta técnica é possível utilizar a funcionalidade disponível no MongoDB que permite detetar alterações numa coleção em tempo real. O excerto de código é apresentado no excerto A.8.

Assim, garante-se a atualização constante dos dados nos respetivos clientes. Esta técnica substitui o uso de *timmers* que enviam periodicamente os dados, porém as atualizações não são instantâneas e podem ser enviados dados repetidos, algo que implica processamento desnecessário do sistema. O fluxograma correspondente a este conceito é apresentado na figura 4.3.

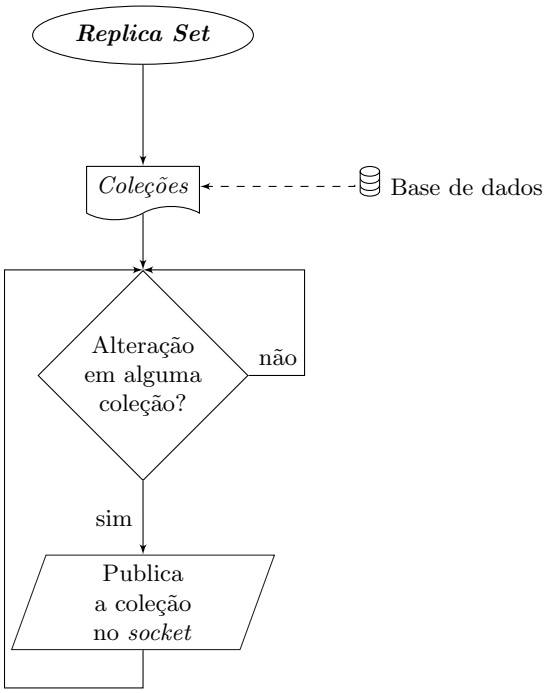


Figura 4.3: Fluxograma de envio de dados através da *replica set*

4.2.3 Cliente MQTT

Como cliente MQTT, este servidor deve ser capaz de comunicar constantemente com o cliente *zigbee2mqtt*, de forma a obter constantes atualizações do sistema. Dados como os dispositivos conectados à rede e as suas respetivas medições devem ser armazenados. Noutra perspetiva, este cliente deve ser capaz de adicionar e remover dispositivos da rede. A escolha do *zigbee2mqtt* teve também como fundamento a facilidade em operar com este, graças à possibilidade de controlar a rede através de tópicos MQTT. Para a implementação deste servidor não foram necessários todas as funcionalidades oferecidas pelo *zigbee2mqtt*, como tal nos pontos seguintes são apresentados os tópicos utilizados e a sua descrição:

- `zigbee2mqtt/bridge/state` → Tópico que indica se o sistema está a ser executado (`online`), ou não (`offline`);
- `zigbee2mqtt/bridge/log` → Tópico que reporta o estado do sistema, os principais tipos de mensagem são:
  - `pairing` → Mensagem enviada quando um dispositivo esta a ser emparelhado;
  - `device_connected` → Mensagem enviada quando um dispositivo é conectado;

  - `device_removed` → Mensagem enviada quando um dispositivo é removido;
  - `device_removed_failed` → Mensagem enviada em caso de falha na remoção de um dispositivo;
  - `device_force_removed` → Mensagem enviada quando um dispositivo é removido no modo *force*;
  - `device_force_removed_failed` → Mensagem enviada em caso de falha na remoção de um dispositivo no modo *force*;
  - `devices` → Mensagem enviada com a lista de dispositivos;

- `zigbee2mqtt/bridge/config/devices/get` → A publicação de uma mensagem em branco neste tópico retorna os dispositivos conectados para o tópico `zigbee2mqtt/bridge/config/devices`;
- `zigbee2mqtt/bridge/config/permit_join` → Tópico que permite gerir a rede com a entrada ou não de dispositivos. Ao enviar a mensagem:
  - `true` é permitida a entrada de dispositivos;
  - `false` é recusada a entrada de dispositivos;
- `zigbee2mqtt/bridge/config/remove` → Tópico utilizado para remover dispositivos da rede, a mensagem a enviar consiste no endereço do dispositivo;
- `zigbee2mqtt/bridge/config/force_remove` → Tópico utilizado em caso de falha de remoção utilizando o tópico anterior, têm os dois a mesma finalidade;
- `zigbee2mqtt/[device_address]` → Tópico que reporta o estado do dispositivo com endereço `[device_address]`;
- `zigbee2mqtt/[device_address]/set` → Tópico que controla o estado do dispositivo com endereço `[device_address]`;

Este servidor irá receber as mensagens paralelamente *zigbee2mqtt* e a lógica irá depender de cada tópico. O pacote `mqtt` permite ao servidor subscrever tópicos e publicar mensagens. O conceito é semelhante à comunicação por *sockets*, existe uma função de escrita `mqtt.publish()` e de escuta `mqtt.subscribe()`. Sendo que neste contexto, os tópicos comportam-se como os canais.

No excerto A.9 é apresentada a estrutura de código que permite à aplicação comportar-se como um cliente para o *broker* instalado no RaspBerry Pi. A porta utilizada pelo *broker* é 1883 por *default* e, conforme apresentado no exemplo, assim que a comunicação seja estabelecida, o tópico `#` do *zigbee2mqtt* é subscrito.

As mensagens quando recebidas são dissociadas separando o seu tópico e o seu conteúdo, conforme apresentado no excerto A.10. Assim, através desta dissociação é possível que a aplicação conheça se algum dispositivo foi conetado ou removido, ou num caso mais específico se a mensagem reporta a leitura de dados de um determinado equipamento.

Assim, tal como a descrição dos tópicos em cima apresentada, é esperado que na receção de uma mensagem cujo o tópico seja `zigbee2mqtt/bridge/state` esta indique o estado do sistema, se este está *online* ou *offline*. Este processo pode e deve ser análogo para os restantes tópicos.

Note-se que é necessário recorrer à função `JSON.parse()`, disponibilizada pelo JavaScript, de forma a converter a mensagem num objeto. Só assim o utilizador tem a capacidade de aceder aos diversos campos que uma mensagem pode ter.

Na figura 4.4 é apresentado o fluxograma que traduz a lógica da conexão deste servidor com o *broker*.

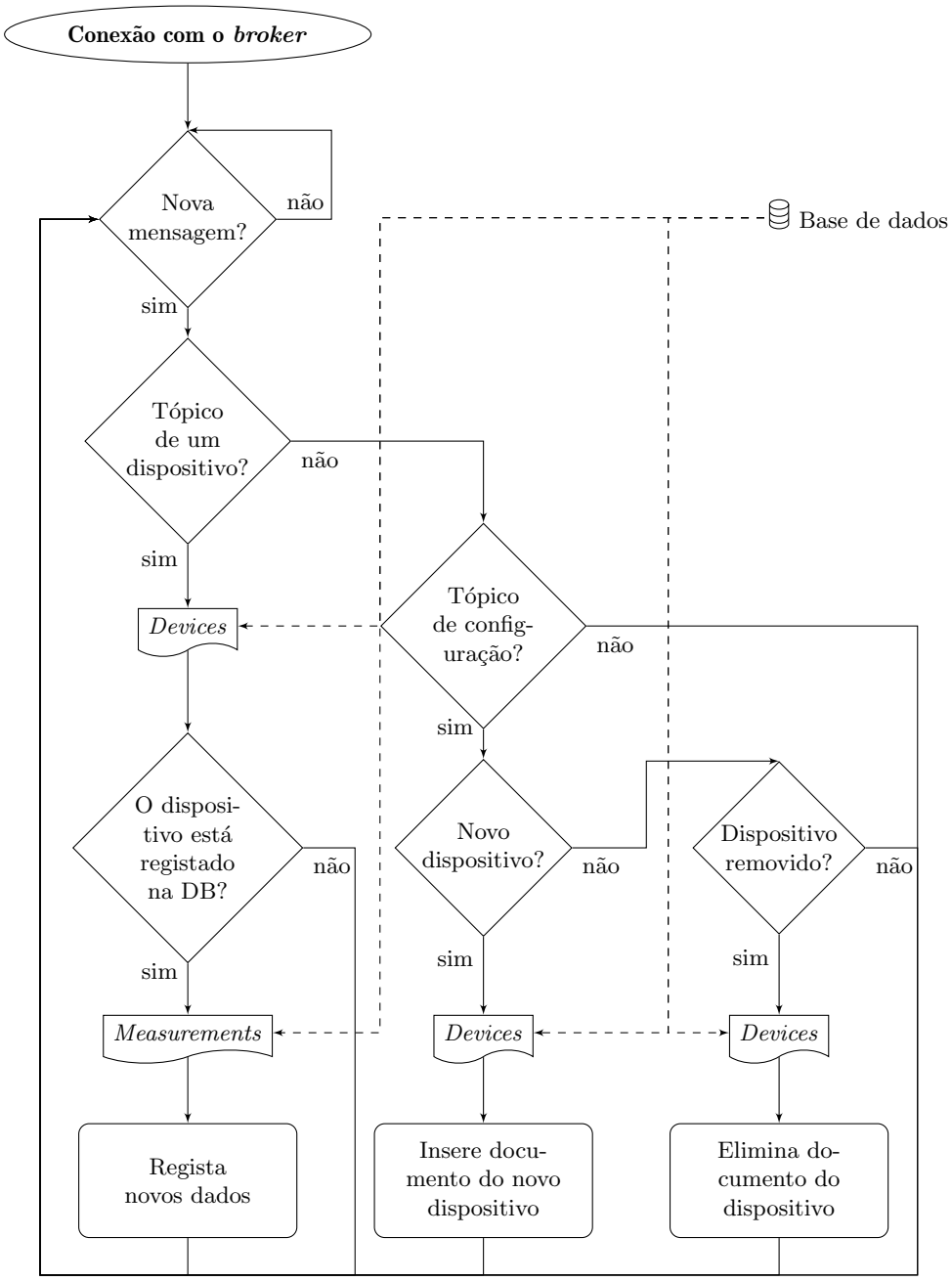


Figura 4.4: Fluxograma da conexão do servidor com o *broker*

Refira-se que é executado um processo de verificação e caso o *zigbee2mqtt* envie a mensagem para o *broker* com os dados de um dispositivo que não está devidamente registado na base de dados deste servidor, a mensagem é ignorada.

Conforme apresentado, a atualização das coleções é efetuada sempre que se justifique. Assim, e com recurso aos *sockets* e à *replica set*, os dados são apresentados de forma instântanea aos clientes conetados. Esta operação é realizada de

forma otimizada sem requerer demasiados recursos de processamento.

No algoritmo A.11, a condição compara o campo `type` com a *string* "device_connected", em caso de igualdade procede-se à inserção de um novo dispositivo na coleção *Devices*. O processo de adição por parte do *zigbee2mqtt* tende a demorar algum tempo, como tal este processo é dividido em duas etapas. Na primeira, quando um dispositivo é conectado é inserido na coleção um documento constituído exclusivamente pelo seu endereço. Desta forma, o utilizador tem a percessão que está a decorrer a configuração de um novo dispositivo. Após este passo, é aguardada a mensagem de configuração bem sucedida por parte do *zigbee2mqtt*. Esta mensagem é também constituída pelas especificações deste dispositivo como o modelo, fabricante e a sua descrição. Estas duas fases estão implementadas nos excertos de código A.11 e A.12.

Na remoção de um dispositivo é necessário proceder à eliminação do documento correspondente na coleção *Devices*. Os documentos desta coleção são filtrados de acordo com o endereço do dispositivo associado, e caso a condição se confirme, o documento é eliminado, conforme apresentado no excerto A.13. No entanto, opta-se por não eliminar os dados do dispositivo removido, já que, caso volte a ser adicionado, é possível aceder ao seu histórico.

Como apresentado nos excertos A.11, A.12 e A.13, o `mongoose` oferece várias funções que permitem manipular as coleções. Para criar um documento é utilizada a função `new()`, para eliminar a função `deleteOne()` e para a atualização `findOneAndUpdate()`.

Relativamente aos dados e medições, apesar da possibilidade de disponibilizar um único documento para cada dispositivo, prefere-se a inserção de um novo documento sempre que sejam reportados novos valores, independentemente do dispositivo. Esta escolha teve como fundamento uma das funcionalidades a implementar na interface gráfica que consiste na exibição destes valores em função do eixo temporal.

Na receção de uma mensagem do tópico `zigbee2mqtt/[device_address]`, conforme apresentado no excerto A.14, após a verificação do registo do dispositivo na base de dados, são analisados todos os campos da mensagem e é feita a comparação destes com o esquema *mongoose*. Em caso de sucesso, os dados são registados. Cada documento é registado com a data da sua inserção possibilitando a comparação de dados em função do tempo, algo que não é reportado pelo *zigbee2mqtt*.

4.3 Interface Gráfica

A interface gráfica é implementada em React, uma biblioteca de JavaScript que é instalada na máquina através do comando:

```
ubuntu@ubuntu$ npx create-react-app interface_grafica
```

A descrição de cada campo é a seguinte:

1. `npx` é o pacote responsável pela a instalação de um determinado pacote no sistema local, semelhante ao `npm`;
2. `create-react-app` indica o módulo a ser instalado;
3. `interface_grafica` é o diretório de destino.

Após a execução deste comando são instalados no diretório `app` todos os módulos necessários para o correto funcionamento da aplicação. Esta já apresenta um código pré-definido, localizado no sub-diretório `src`. Serve como guia e deve ser editado de acordo com as funcionalidades a implementar.

Assim como acontece com o servidor, a estrutura do diretório da aplicação é constituída por um ficheiro designado por `package.json`. Aqui, são apresentadas todas as especificações da aplicação. Como tal, na instalação de um novo módulo é necessário evocar o campo `--save` para que este seja considerado como indispensável para o correto funcionamento da aplicação.

Tratando-se esta de uma aplicação com diversas funcionalidades opta-se pela implementação de várias páginas, de forma a garantir a boa organização do conteúdo e uma fácil interação com o cliente. Tal é possível graças ao pacote `react-router` que permite a distribuição destas páginas em função do endereço que o cliente está a aceder. O comando que deve ser executado para a instalação deste módulo na publicação é o seguinte:

```
ubuntu@ubuntu$ npm install --save react-router-dom
```

No exemplo da figura 4.5, assim que o cliente acede ao endereço `localhost` é verificada a *slash* correspondente ao pedido. Caso esta coincida com `/home` é exibida a *Home Page*, em caso negativo é feita mais uma comparação com `/test` e caso se confirme é apresentada a página *Test Page*. Caso as duas condições falhem é apresentada a uma página de erro.

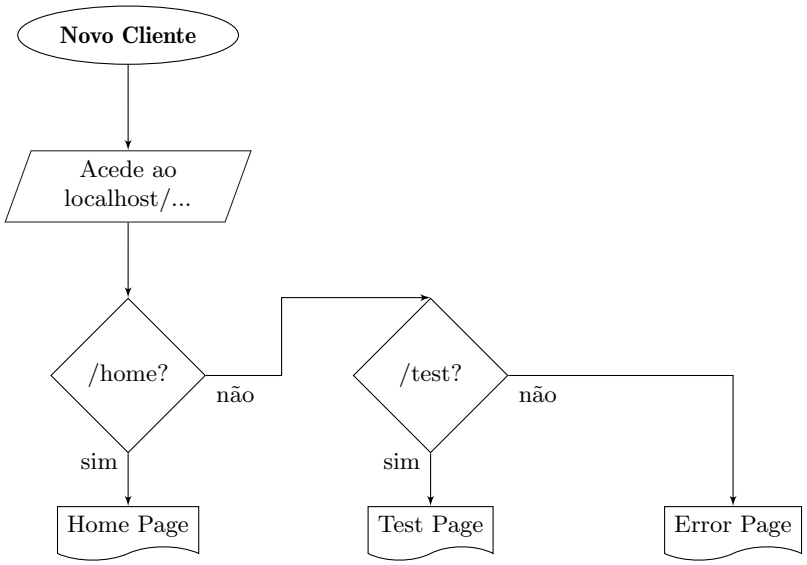


Figura 4.5: Fluxograma do funcionamento *react-router*

4.3.1 Autenticação

Um dos pontos a ter em conta nesta implementação prende-se com o facto desta ser uma aplicação para controlo de dispositivos ZigBee presentes numa residência ou empresa. Como tal, é necessário proceder à autenticação de segurança dos clientes para que só pessoas autorizadas consigam intervir no sistema.

A *Firebase* é uma plataforma desenvolvida pela Google para a desenvolvimento de aplicações móveis e *Web*. Oferece diversas funcionalidades aos seus utilizadores entre as quais se destacam: a base de dados, tal como o MongoDB, e a possibilidade de criar uma aplicação com autenticação de forma simples e segura. Nos pontos seguintes serão enunciados os passos seguidos até obter uma aplicação com acesso exclusivo a utilizadores capacitados.

Para este efeito é necessário desenvolver duas páginas uma para início de sessão, *Log In*, e outra para registo de um novo utilizador, *Sign Up*. Note-se que esta ultima é desenvolvida para que o sistema tenha algum cliente, após o registo há a possibilidade de ocultar esta página para que um único utilizador tenha acesso.

Na *Firebase* é necessário escolher que credenciais serão aceites para um novo cliente, esta configuração é acessível na aba `Authentication/Sign-in Method` do projeto e são oferecidas diversas opções de escolha entre as quais se destacam os métodos: e-mail/password, o número de telefone ou as redes sociais (*Facebook* e *Twitter*). Perante estas opções, e considerando que o uso das redes sociais seria uma limitação, opta-se pelo método mais comum: e-mail e password. Após a seleção do método, a configuração da aplicação é concluída, gerando uma chave

associada que terá de ser importada no projeto principal. Para aplicações *Web*, a chave é semelhante ao exemplo apresentado no excerto A.15.

O fluxograma da autenticação a implementar é apresentado na figura 4.6.

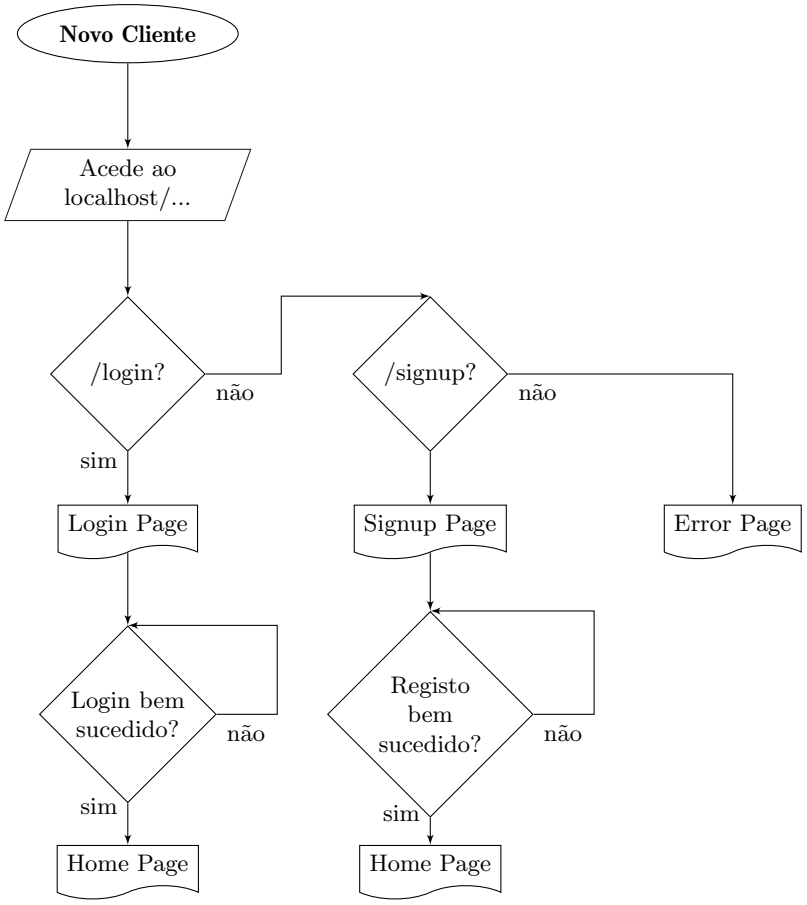


Figura 4.6: Fluxograma da autenticação da interface gráfica

Como apresentado, as páginas de *Log In* e *Sign Up* são públicas e em ambos os casos após o sucesso das operações é apresentada a página principal ao cliente.

Como medida restrita, é necessário configurar a aplicação para que esta só possa ser acedida caso o utilizador esteja autenticado. Para tal, cria-se a componente `PrivateRoute` que verifica a autenticação e em caso de sucesso são dados todos os privilégios ao utilizador. Tal é apresentado no excerto A.16.

A *Firebase* oferece também a possibilidade de controlar os acessos à aplicação e apresenta a informação dos clientes que foram ou estão registados. Tal informação pode ser acedida na aba `Authentication/Users`.

Como inconveniente, pode-se referir que esta autenticação só poderá funcionar caso a aplicação se encontre conectada à Internet. Tendo em conta o con-

texto deste projeto, é necessário encontrar uma alternativa para quando alguma anomalia ocorra. Assim, irá ser desenvolvida uma aplicação local, semelhante à aplicação a ser servida no servidor externo, sem autenticação. Desta forma, os utilizadores da residência podem ter acesso constante ao sistema, mesmo sem ter acesso à Internet.

4.3.2 Aplicação

Como mencionado no capítulo 4.2.1, é preciso garantir que esta interface seja um cliente para o servidor. Para tal, é necessário adicionar o módulo `socket.io-client` a esta aplicação a fim de estabelecer a comunicação com o servidor. Após instalado deve-se proceder à sua importação no projeto. A comunicação é estabelecida com recurso ao endereço do servidor, tal como apresentado no excerto A.17.

Para além das páginas de autenticação e tendo em conta os objetivos estipulados é necessário desenvolver no mínimo quatro páginas: *Devices*, *Divisions*, *Statistics* e *Scenes*. O processo de associação entre os endereços e as páginas é apresentado no fluxograma da figura 4.7.

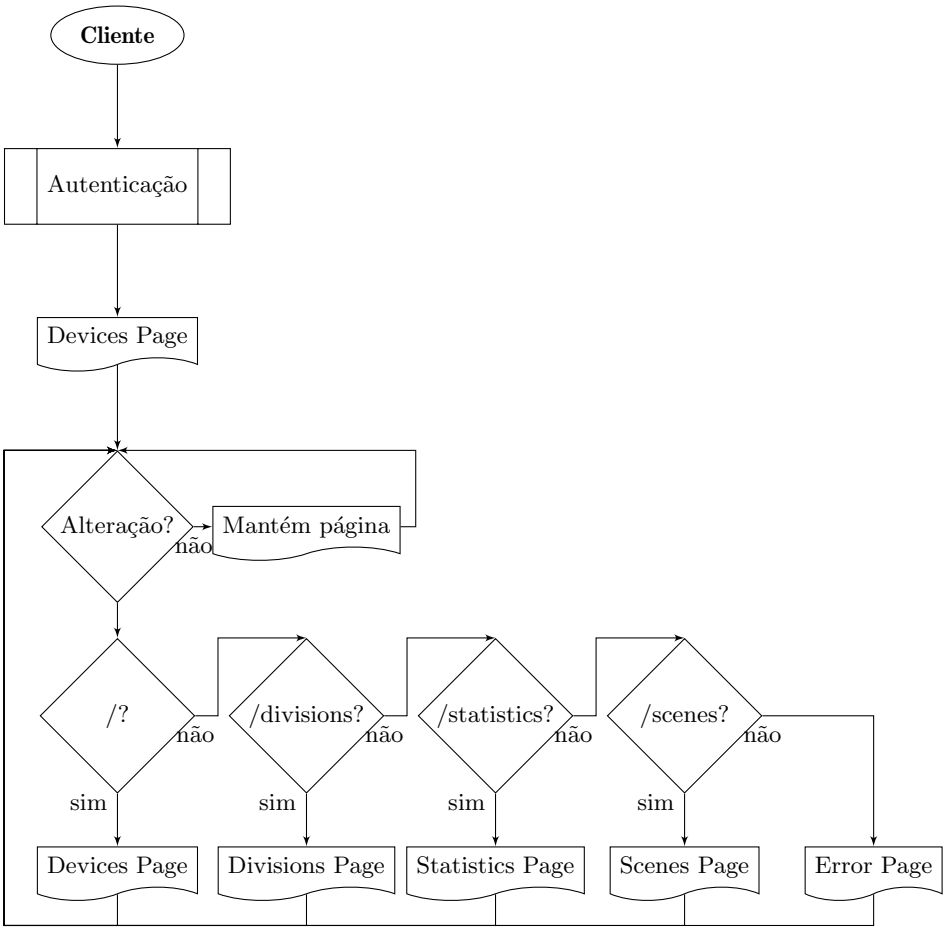


Figura 4.7: Fluxograma das páginas da interface gráfica

Conforme apresentado, as páginas são alteradas em função do endereço acedido. Caso não ocorra alteração, a página é mantida. No excerto A.18 é apresentada a estrutura de código responsável por esta operação.

Nesta fase é importante mencionar que a conexão com o servidor é realizada uma única vez, logo após à autenticação. Assim, garante-se que o servidor não fica sobrecarregado com a entrada e saída de clientes sempre que é alterada a página.

O *socket* é partilhado ao longo das componentes com recurso ao método `Props`. Esta é uma abreviação de *properties*, ou propriedades, e corresponde às informações que podem ser passadas para um componente.

Devices Page

Esta será a *homepage*, página principal, onde serão exibidos os dispositivos conetados, as respetivas informações e controlos. Quando esta página é acedida são enviados dois pedidos ao servidor através do *socket*, conforme apresentado no excerto A.19.

O servidor, ao receber este pedido, responde com os dados provenientes das coleções *Devices* e *Measurements*, respetivamente. O cliente ficará à escuta, e quando os dados forem recebidos estes serão alocados em *arrays* locais. O excerto responsável por esta operação é apresentado no exemplo A.20.

O fluxograma apresentado na figura 4.8 apresenta a lógica desta componente.

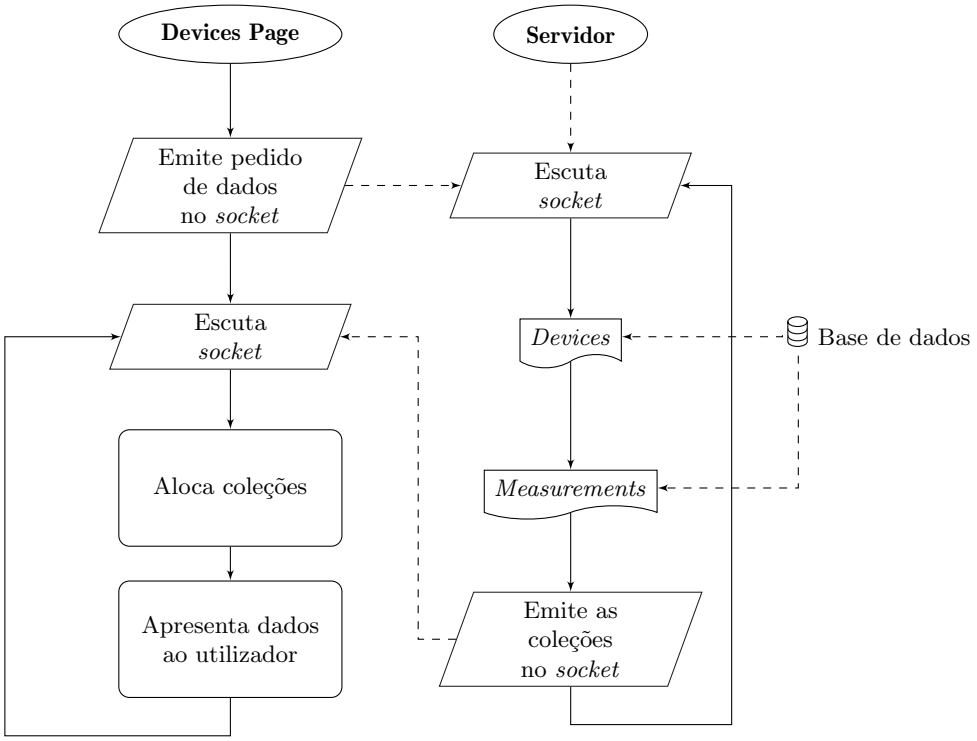


Figura 4.8: Fluxograma da Devices Page

O React oferece a funcionalidade `useState` que permite alocar dados diretamente numa variável. Este facto é aproveitado no exemplo A.20, a informação recebida no campo "`Devices`" é alocada no *array* `devices` e a do campo "`Measurements`" no *array* `data`.

No entanto, para além de efetuar este pedido ao servidor, esta página estará sempre à escuta através do *socket*. Sempre que é enviada nova informação esta é alocada seguindo o método abordado no parágrafo anterior.

Aqui, o utilizador tem a capacidade de analisar algumas especificações de cada dispositivo através do *array* `devices` cuja estrutura exemplo é apresentada no excerto A.21. Paralelamente, são apresentadas as medições ou dados recolhidos pelo servidor à cerca de cada dispositivo.

A impressão destes dados é possibilitada através do método `.map()` que permite aceder aos diversos campos dentro de um *array*. Um exemplo de aplicação, com recurso a esta técnica é apresentado no excerto A.22.

Para a exibição da última informação reportada pelo dispositivo, procede-se à filtragem do *array* `data` de acordo com o seu endereço e com recurso ao método `splice` são eliminados todos os objetos exceto o mais recente. Assim, é originado um novo *array* com a última informação inserida na base de dados tal como sugere o excerto A.23.

As medições são exibidas conforme a sua existência no *array* anterior. O React permite a renderização seguindo condições tal como sugere o excerto A.24. Caso a condição falhe, o parâmetro não é renderizado. Isto facilita esta fase de implementação, sendo necessário efetuar uma única vez a declaração de variáveis para todos os dispositivos. É importante referir que na condição de valor nulo, a aplicação considera a não existência do parâmetro. Desta forma, para algumas condições neste excerto é utilizada a comparação com um valor negativo garantindo assim que, caso a medição seja nula, o parâmetro é impresso.

Apesar de todas a funcionalidades que o *zigbee2mqtt* apresenta, pode-se considerar o inconveniente de este não reportar todos os parâmetros suportados pelo dispositivo. Assim, a interação do utilizador com os dispositivos é limitada. De forma a contrariar esta situação opta-se pelo desenvolvimento de um ficheiro JSON constituido por um *array* cujos objetos apresentam o modelo do dispositivo e os parâmetros suportados. Este ficheiro deve ser acedido nesta página para que, mesmo que *zigbee2mqtt* falhe, o utilizador tenha o controlo total do dispositivo. Uma estrutura exemplo deste ficheiro é apresentado no excerto A.25.

Assim, a filtragem deste ficheiro nesta página permite exibir os controlos em função do modelo do dispositivo, tal como sugere o excerto A.26. Note-se que neste exemplo o *array* `DevicesParameters` corresponde ao *array* do ficheiro JSON.

Cada parâmetro está associado a uma componente que será responsável pelo envio de mensagens para o servidor a fim de controlar o dispositivo. No exemplo do excerto A.27, a condição verifica se existe o campo `state` no objeto que corresponde ao modelo do dispositivo. Em caso afirmativo a componente `<State />` é exibida.

Neste caso em concreto, esta componente é constituída por botões que quando selecionados emitem no *socket* mensagens para ligar ou desligar o dispositivo, tal

como sugerido no algoritmo A.28. O conteúdo destas mensagens para além do estado inclui também o endereço do dispositivo a atuar.

O servidor fica à escuta e após receber a mensagem publica no tópico correspondente a informação, no caso do algoritmo A.29, é publicado o estado do dispositivo. É feita uma verificação se a mensagem enviada para o *socket* contém os tópicos `device_id` e `state` para que não ocorram erros na publicação no *zigbee2mqtt*.

O fluxograma que traduz este conceito é apresentado na figura 4.9. Neste caso, é publicada a mensagem de ligar o dispositivo no tópico *zigbee2mqtt*.

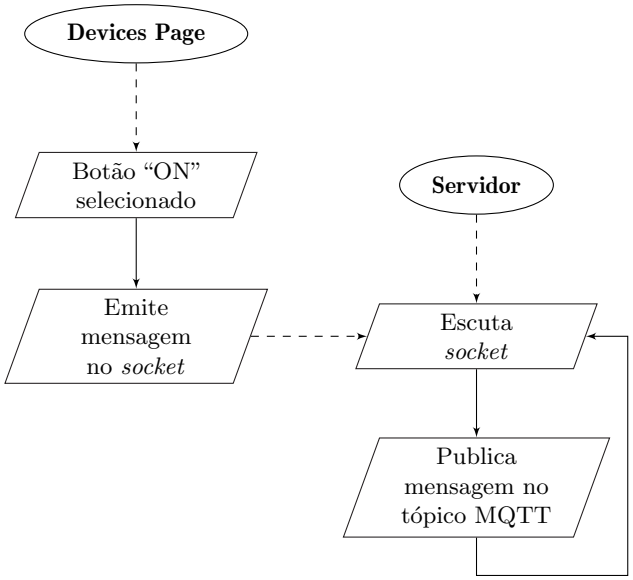


Figura 4.9: Fluxograma da comunicação cliente-servidor para controlo de dispositivos

Nesta implementação opta-se ainda pelo desenvolvimento de mais três componentes que estão associadas ao controlo de brilho, temperatura de cor e cor de lâmpadas. Recorre-se a *range-sliders* que permitem ao utilizador ajustar cada parâmetro conforme a sua preferência. As mensagens são enviadas pelo método análogo ao exemplo do algoritmo A.28 e fluxograma 4.9.

Devices Settings Page

Na sequência da componente anterior, esta página tem como finalidade permitir ao utilizador uma experiência mais *user-friendly*. Aqui, são enviadas mensagens para o servidor que permitem a edição da coleção associada aos dispositivos. O utilizador tem a capacidade de alterar a designação do dispositivo e a sua localização. Esta é uma componente importante para esta interface, já que é a partir desta que ocorre remoção de dispositivos na rede.

Na edição do nome do dispositivo é questionado ao utilizador qual será a nova designação. Após submetida a resposta, a mensagem será enviada para o campo do *socket* correspondente, algoritmo A.30.

O servidor atualiza a coleção *Devices* após a recessão desta mensagem conforme apresentado no algoritmo A.31. Esta deverá ser constituída pelo novo nome e o nome antigo do dispositivo, só assim poderá ser feita a atualização do documento.

Na figura 4.10 é apresentado o fluxograma de edição da designação do dispositivo. Note-se que a primeira designação corresponde ao endereço do dispositivo na rede.

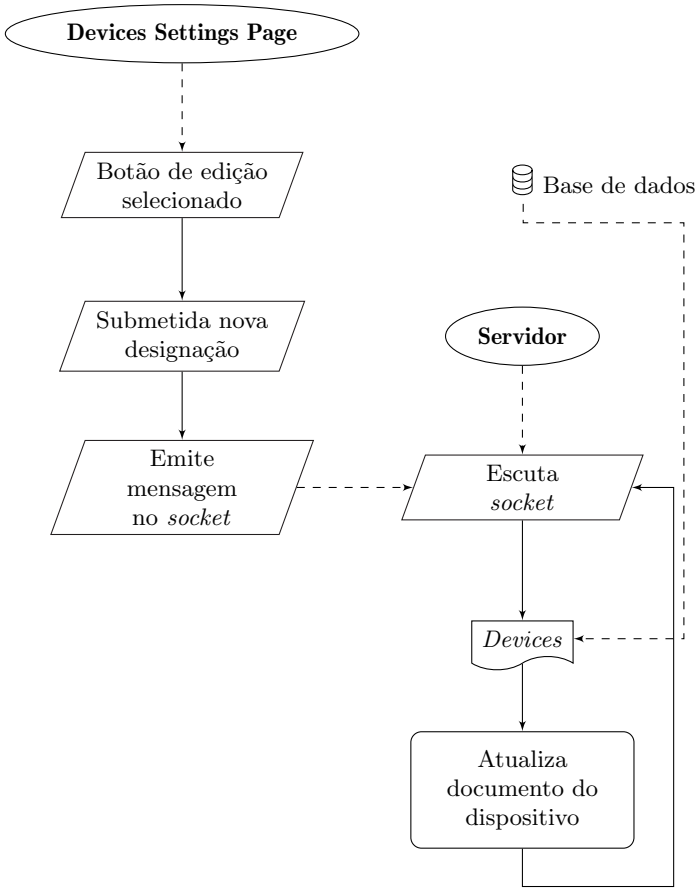


Figura 4.10: Fluxograma da edição da designação de um dispositivo

A eliminação de um dispositivo ocorre de forma semelhante, quando é pressionado o botão de remoção é enviada uma mensagem no *socket* constituída pelo endereço de rede do dispositivo a remover, algoritmo A.32.

No servidor esta é publicada no tópico que permite a remoção de dispositivos, tal como sugere o algoritmo A.33. O fluxograma que traduz esta lógica é

apresentado na figura 4.11.

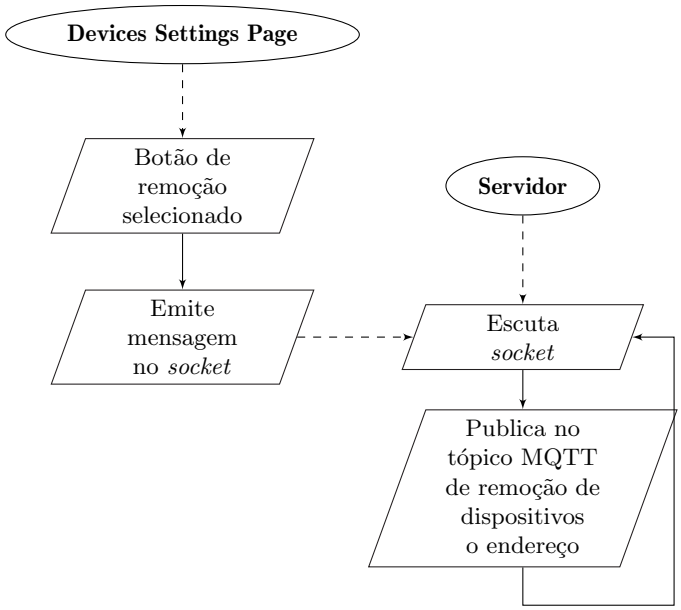


Figura 4.11: Fluxograma da remoção de um dispositivo

Relativamente à localização, esta é dependente da página que é apresentada de seguida. No entanto, de acordo com as divisões já inseridas na base de dados, o utilizador tem a capacidade de alterar a localização de cada dispositivo.

Para tal, deverá selecionar a nova localização sendo emitida uma mensagem no *socket*, o servidor à escuta recebe esta mensagem e efetua as modificações na coleção dos dispositivos. O fluxograma desta operação é apresentado na figura 4.12.

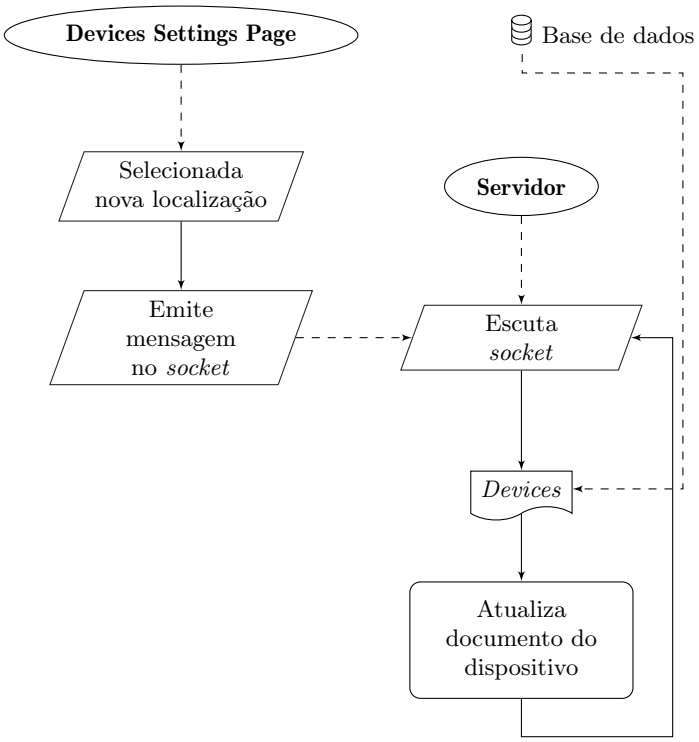


Figura 4.12: Fluxograma da alteração de localização de um dispositivo

Divisions Page

Nesta página, o utilizador terá a capacidade de criar, editar e eliminar divisões. Esta funcionalidade é acrescentada com a finalidade de organizar os diversos equipamentos que uma instalação poderá ter.

De forma análoga ao que acontece na Devices Page, nesta página é efetuado um pedido de dados ao servidor que responde com as coleções *Devices* e *Divisions*. Após esta receção estas duas coleções são alocadas nos *arrays* `devices` e `divisions`. Na interface os dispositivos são agrupados conforme a sua localização como sugere o excerto de código A.34.

Quando acedida, esta página é efetuado um pedido de dados ao servidor ao qual este responde com as coleções *Devices* e *Divisions*. Estas são alocadas em *arrays* locais, e os dados são apresentados ao utilizador. O fluxograma da lógica desta pagina é apresentado na figura 4.13.

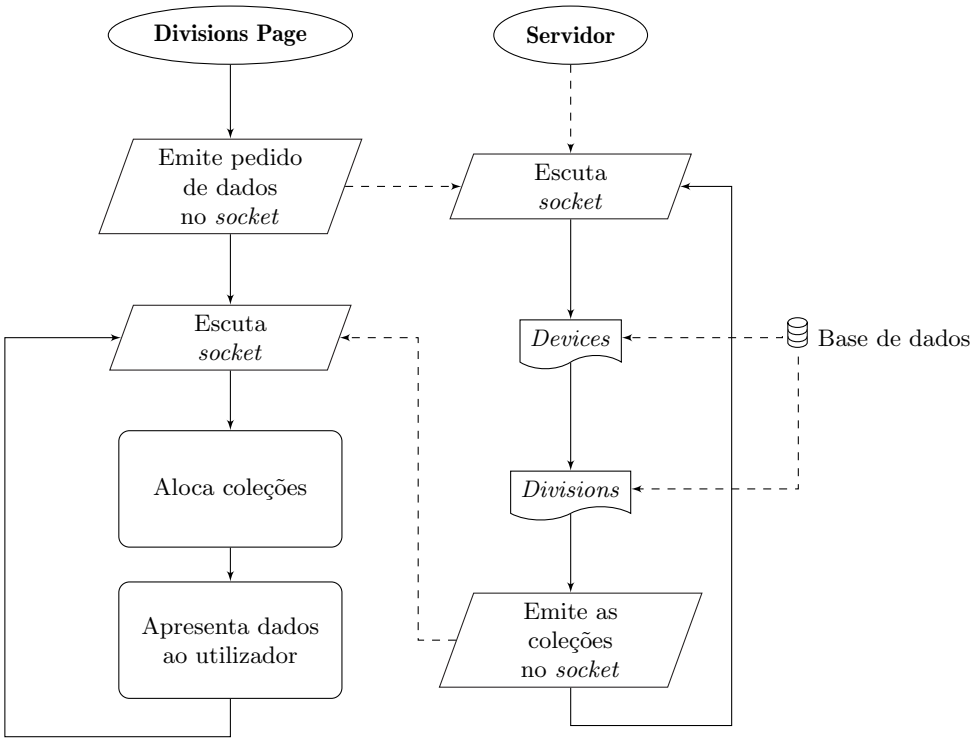


Figura 4.13: Fluxograma da Divisions Page

Para uma nova divisão é questionada a sua designação e após a sua submissão é enviada uma mensagem para o servidor com a sua designação. Após receção no servidor, é publicado um novo documento na coleção *Divisions*, tal como sugere o fluxograma da figura 4.14.

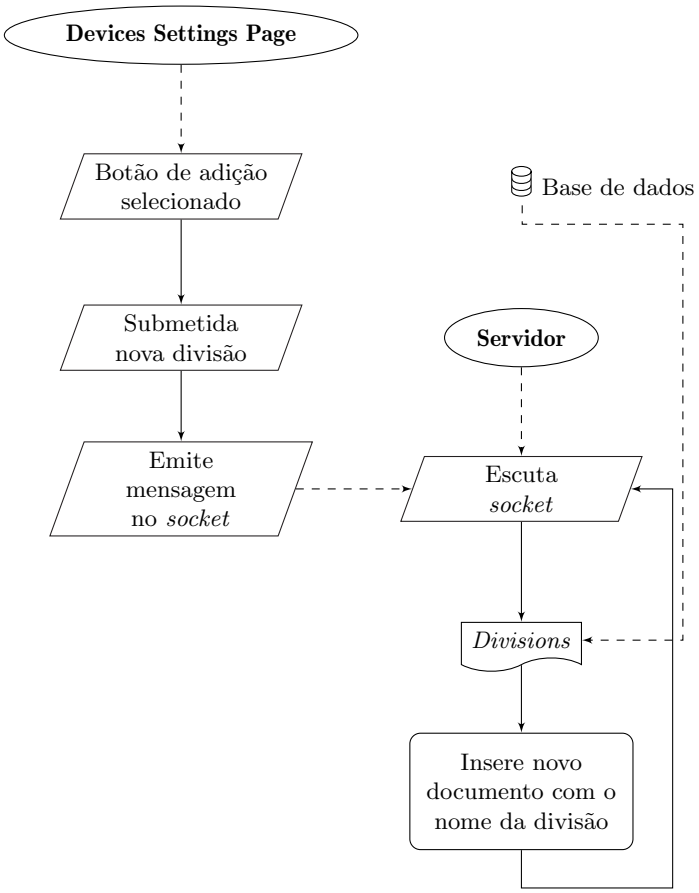


Figura 4.14: Fluxograma da adição de uma divisão

O processo de eliminação é análogo ao da remoção do dispositivo da rede, com a particularidade desta operação se realizar única e exclusivamente na base de dados. Após selecionado o botão é emitida uma mensagem no *socket* com a designação do dispositivo a remover. O servidor, à escuta, acede à coleção *Divisions* e elimina o respetivo documento. O fluxograma da figura 4.15 traduz esta operação.

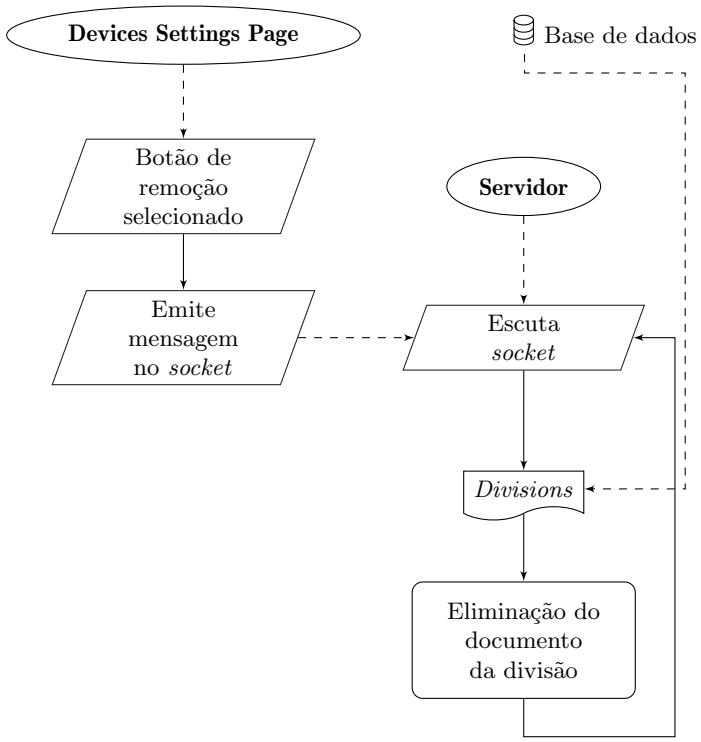


Figura 4.15: Fluxograma da eliminação de uma divisão

Statistics Page

Esta componente apresenta os dados dos diversos dispositivos em função do eixo temporal. O utilizador tem a possibilidade de adicionar e eliminar os gráficos. Para além desta representação opta-se ainda pelo acréscimo da funcionalidade de cálculo. Esta permite ao utilizador calcular o valor médio de um determinado parâmetro num intervalo de tempo.

Quando acedida é efetuado um pedido de dados ao servidor, de forma análoga às páginas anteriores, ao qual é respondido as coleções *Devices*, *Measurements* e *Graphics*. As duas primeiras são responsáveis pela a exibição dos gráficos contidos na coleção *Graphics*. Esta contém os gráficos a exibir, já selecionados pelo o utilizador. O fluxograma desta página é apresentado na figura 4.16.

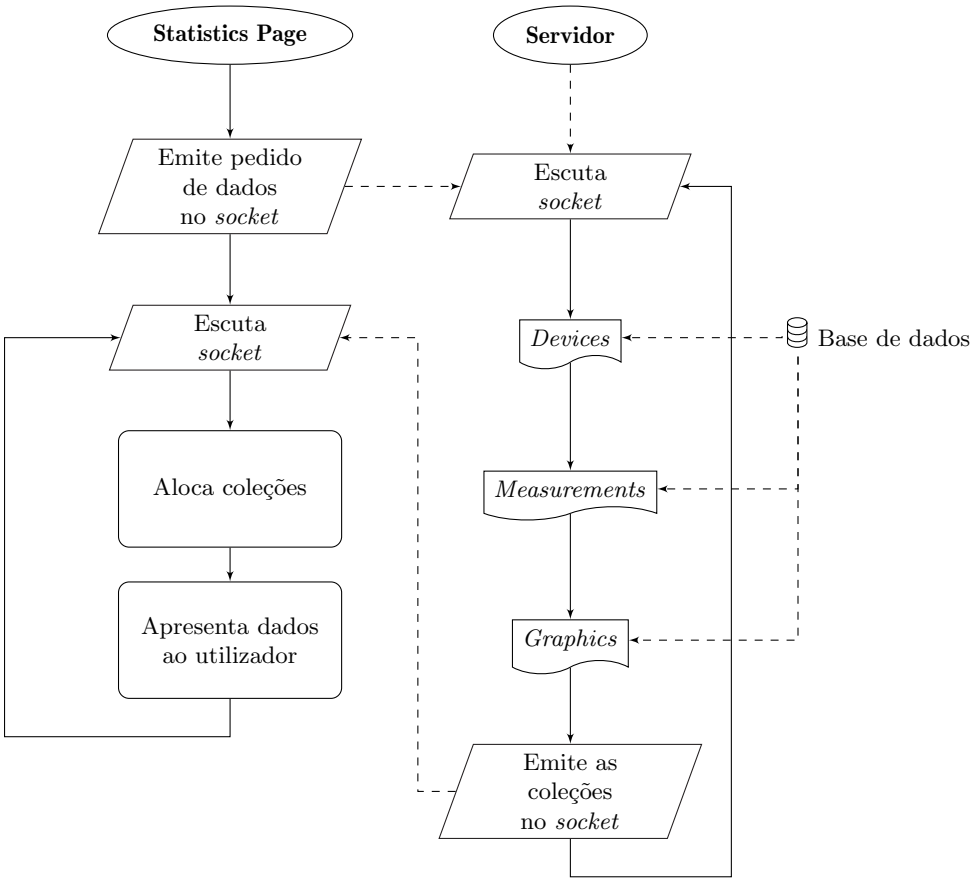


Figura 4.16: Fluxograma da Statistics Page

Os gráficos são atualizados sempre que ocorra uma alteração na coleção *Measurements*, garantindo assim atualizações em tempo real do estado do sistema.

Nesta página são analisados os documentos (da coleção *Graphics*) já inseridos na base de dados e, em função das medições de cada dispositivo, é feita a representação no eixo temporal. Tal é possível através da adição do instante em que o documento é inserido na base de dados, conforme já mencionado.

Para a adição de um novo gráfico o utilizador deverá selecionar o respetivo botão, sendo solicitados o parâmetro e o dispositivo a representar. Após submetidos é enviada uma mensagem ao servidor, e este insere um novo documento na coleção *Graphics*, tal como sugerido no fluxograma da figura 4.17.

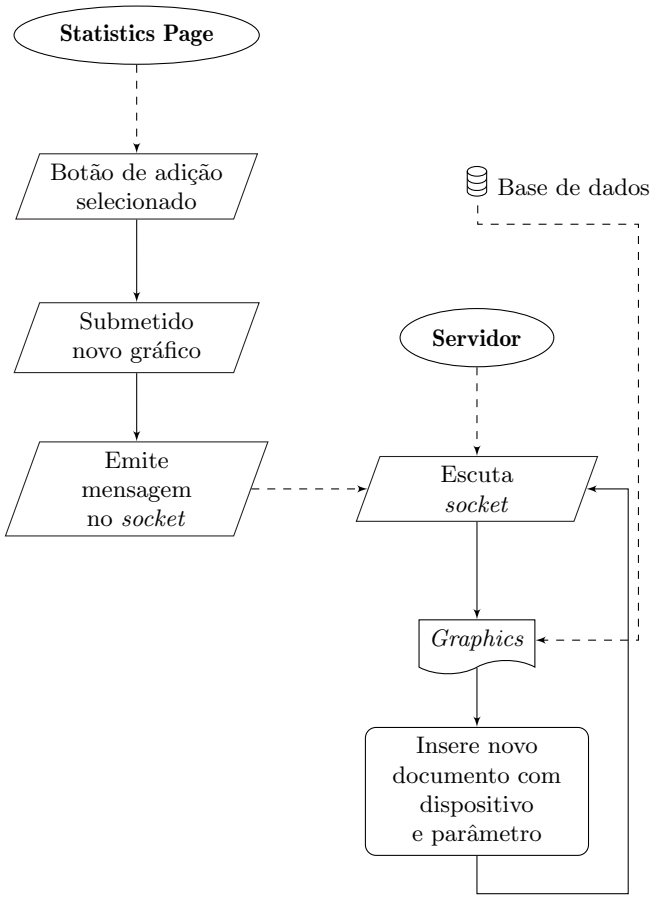


Figura 4.17: Fluxograma da adição de um gráfico

Se o utilizador pretender eliminar um gráfico deverá selecionar o seu botão de remoção, sendo emitida uma mensagem no *socket* com a identificação do gráfico. Após recebida no servidor, é eliminado o documento associado na coleção *Graphics*. Tal operação é representada no fluxograma da figura 4.18.

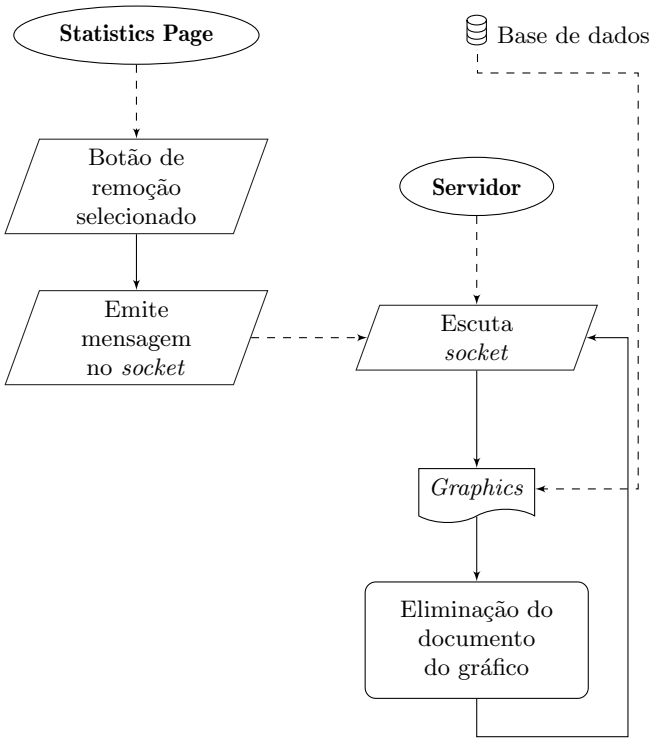


Figura 4.18: Fluxograma da eliminação de um gráfico

Para a exibição destes valores recorre-se à biblioteca `react-chartjs-2` que oferece diversas opções de representação. Esta requer a instalação do módulo `chart.js` na aplicação. Esta exibição ocorre seguindo uma estrutura de lógica conforme apresentado no excerto A.35.

As `labels` correspondem aos valores do eixo das abcissas. No *array* `datasets` está contida a informação relativa ao tipo de gráfico, as cores e os dados para o eixo das ordenadas.

Para uma melhor perceção e considerando que são reportados dados sucessivos pelos dispositivos opta-se por filtrar os últimos 100 objetos associados a cada dispositivo, ou seja, só será feita a representação dos últimos 100 valores do parâmetro.

Para efeito de cálculo é solicitado ao utilizador o parâmetro, o dispositivo e o intervalo de tempo. Após a inserção é feita uma verificações se os dados são válidos. Em caso afirmativo é feita uma filtragem do *array* associado às medições tendo em conta o intervalo selecionado. De seguida é processado o cálculo e é apresentado o resultado final ao utilizador, tal como sugerido no fluxograma da figura 4.19.

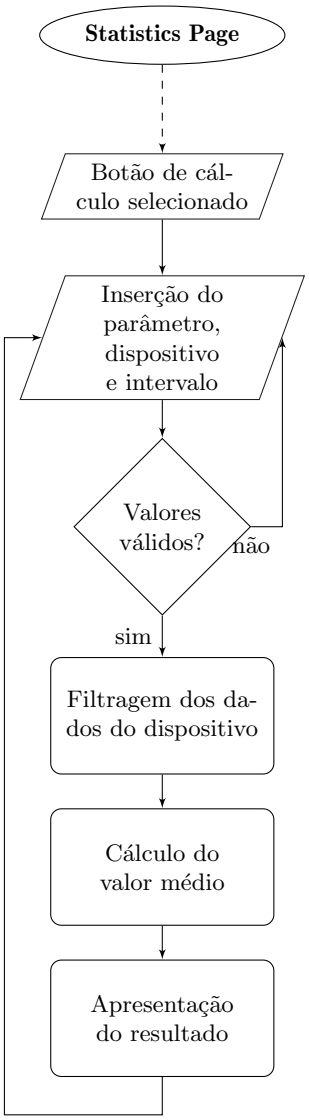


Figura 4.19: Fluxograma do cálculo do valor médio

De seguida é apresentado um exemplo para facilitar a compreensão do algoritmo a desenvolver. A tabela 4.2 apresenta um exemplo de dados obtidos no sistema.

Objeto	**Potência (W)**	**Hora do dia**
Data 1	7,5	01h45
Data 2	6	03h15
Data 3	8	03h40
Data 4	3	04h40

Tabela 4.2: Exemplo de dados

Considerando que o utilizador pretende o valor médio para o intervalo entre 2 horas e 20 minutos e as 4 horas, estes instantes tomam o valor do objeto anterior, ou seja, $7,5W$ e $8W$, respetivamente. Assim, obtém-se uma nova tabela com os novos dados que devem ser considerados para efeito de cálculo, tal como sugere a tabela 4.3 e a figura 4.20.

Objeto	**Potência (W)**	**Horas**	Δt **(s)**
Data 1	7,5	02h20	0
Data 2	6	03h15	3300
Data 3	8	03h40	1500
Data 3	8	04h00	1200

Tabela 4.3: Exemplo de dados para o cálculo do valor médio

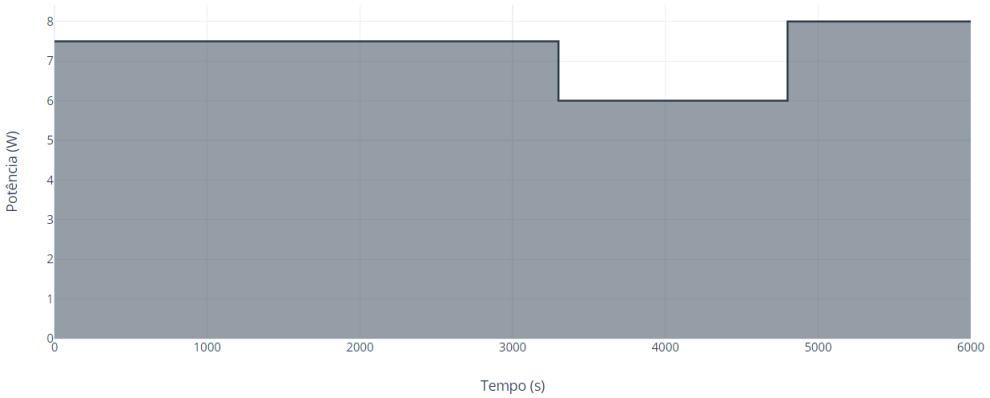


Figura 4.20: Gráfico associado à tabela 4.3

Considerando os dados da tabela 4.3, o calculo do valor médio é dado por 4.1 e a energia é dada por 4.2.

$$\overline{P} = \frac{7,5 \times 3300 + 6 \times 1500 + 8 \times 1200}{3300 + 1500 + 1200} \approx 7,23W \tag{4.1}$$

$$E = P \times \Delta t = 7,5 \times 3300 + 6 \times 1500 + 8 \times 1200 = 43350J \approx 0,012kWh \tag{4.2}$$

Como o parâmetro em questão é a potência acrescenta-se a capacidade de cálculo da energia. Com este valor é possível calcular o custo monetário do consumo de um determinado dispositivo num intervalo de tempo específico. Para tal é solicitado o valor do custo por kWh. Para o exemplo anterior, supondo que este custo variável é de 0,1539€/kWh. O preço final do consumo registado pelo dispositivo vem em 4.3.

$$C = 0,1539 \times 0,012 \approx 0.0018€ \tag{4.3}$$

Scenes Page

Esta página surge associada aos cenários, entende-se por cenários regras que permitem tornar o sistema autónomo. Quando acedida é efetuado um pedido ao servidor ao qual este responde com a coleção *Scenes*. Esta é constituída por campos que indicam as especificações do sensor e do atuador para cada uma das regras. O fluxograma desta página é apresentado na figura 4.21.

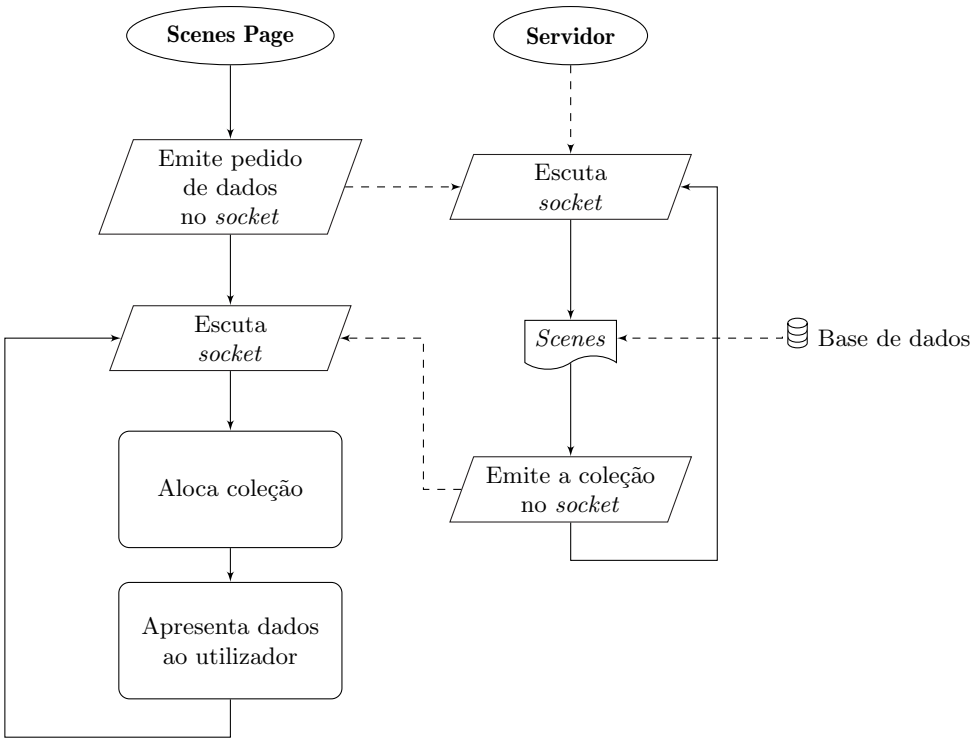


Figura 4.21: Fluxograma da Scenes Page

O utilizador terá a capacidade de ativar e desativar as regras, sendo para isso necessário definir um campo de *status* na coleção. Quando esta se encontra ativa, o campo toma valor *true* e no caso inverso *false*. Para a alteração, é considerado um *switch* que quando selecionado envia uma mensagem ao servidor a fim de alterar o *status*. Esta funcionalidade é apresentada no fluxograma da figura 4.22.

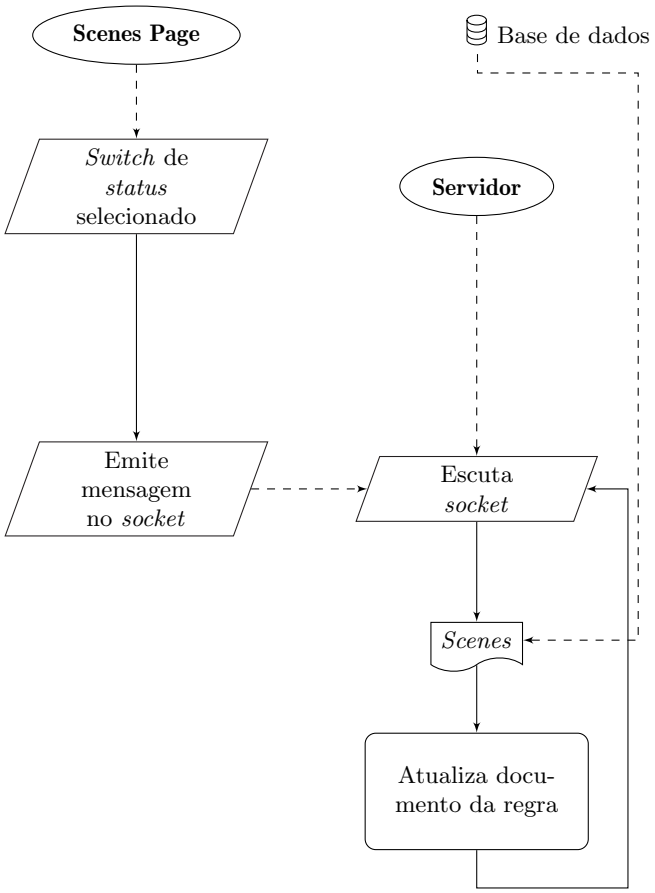


Figura 4.22: Fluxograma da edição do *status* de uma regra

Dentro deste conceito opta-se pela implementação de dois tipos de cenários: um em função do estado dos dispositivos e outro função do tempo. Isto é uma mais-valia para qualquer controlador de domótica residencial já que assim o utilizador não está obrigado a recorrer ao sistema constantemente.

Para o primeiro, o utilizador terá que selecionar o sensor, o parâmetro, e o respetivo valor. Assim que estes três campos sejam preenchidos deverá ser selecionado o dispositivo a ser atuado, o parâmetro e o seu valor.

No segundo, será questionado o instante de tempo, o dispositivo a atuar, o parâmetro e o seu valor.

As mensagens só poderão ser enviadas para o servidor caso todos os campos estejam preenchidos, este ponto é essencial para prevenir eventuais erros.

O fluxograma da figura 4.23 apresenta a adição de uma nova regra no sistema. Já na figura 4.24 é apresentado o fluxograma da lógica para a eliminação de uma regra.

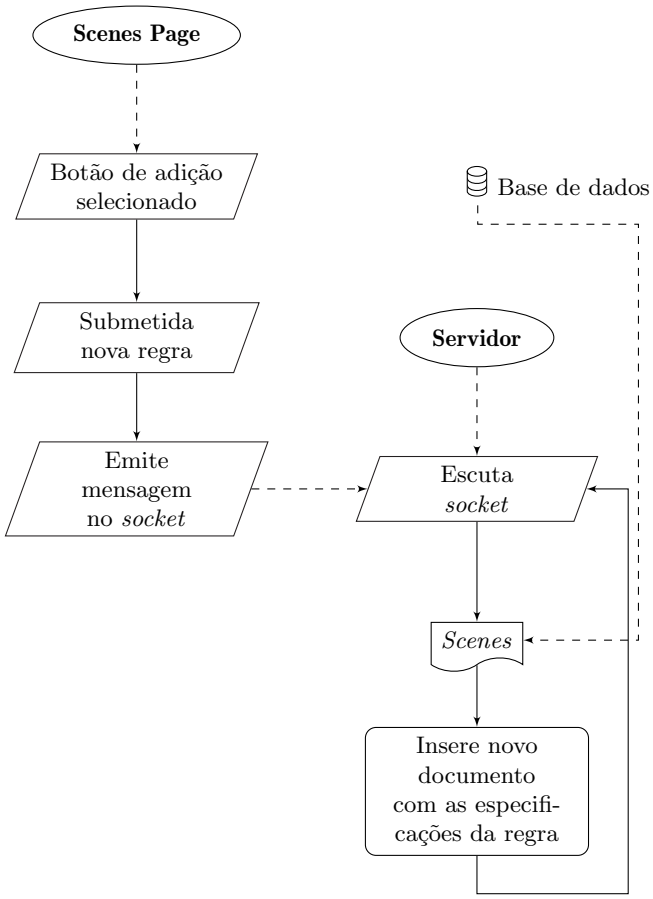


Figura 4.23: Fluxograma da adição de uma regra

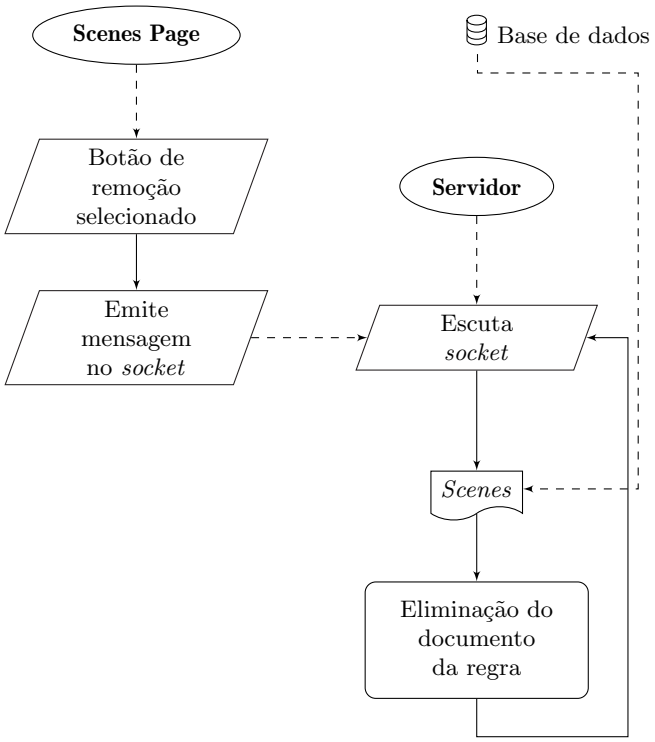


Figura 4.24: Fluxograma da eliminação de uma regra

Toda a interface gráfica é servida num servidor externo através do pacote `surge` do Node.js, sendo necessária apenas a execução de um único comando. Para a instalação na máquina é necessário executar o comando:

```
ubuntu@ubuntu$ npm install --g surge
```

É semelhante à instalação dos pacotes já referidos neste documento mas apresenta a particularidade do módulo ser instalado na máquina e não no diretório.

Com a implementação da interface concluída, é necessário criar uma aplicação estática. Isto é possível com o comando:

```
ubuntu@ubuntu$ npm run build
```

Desta forma gera-se uma pasta composta por todos os ficheiros da aplicação, permitindo assim a otimização da mesma. A aplicação é disposta no servidor através da execução do comando `surge` no terminal, sendo apenas questionadas as credenciais e a pasta da aplicação estática. Esta aplicação é disposta no endereço http://www.zigbee.surge.sh, tendo sido criado um utilizador teste com as credenciais de email 123@123.com e *password* 123456.

4.4 Cliente para os cenários

Para a execução das regras definidas pelo o utilizador, é necessário recorrer a um cliente para o servidor. Este irá ser responsável por analisar o estado do sistema e tomar decisões. O ambiente de programação é o Node.js, o mesmo utilizado na implementação do servidor.

Este cliente estará em contacto permanente com a base de dados, caso ocorra alguma alteração na coleção *Measurements* irá ser analisada a coleção *Scenes* a fim de verificar se o dispositivo associado à última inserção tem alguma regra como sensor.

Em caso afirmativo, se a regra se encontrar com *status* ativo, é verificado o estado do dispositivo a atuar. Se o estado deste for diferente daquele apresentado na regra é enviada uma mensagem para o servidor a fim deste proceder à alteração no tópico MQTT do dispositivo. Isto previne, a sobrecarga de mensagens tanto no servidor como no *zigbee2mqtt*, quando o estado do dispositivo a atuar já é o desejado. O fluxograma da função responsável por esta operação é apresentado na figura 4.25.

Paralelamente, ocorre a verificação dos cenários de segundo a segundo com o objetivo de encontrar alguma regra cujo parâmetro temporal seja o instante atual. O procedimento é semelhante ao anterior, é feita a verificação do estado do dispositivo, fluxograma da figura 4.26 representa esta lógica.

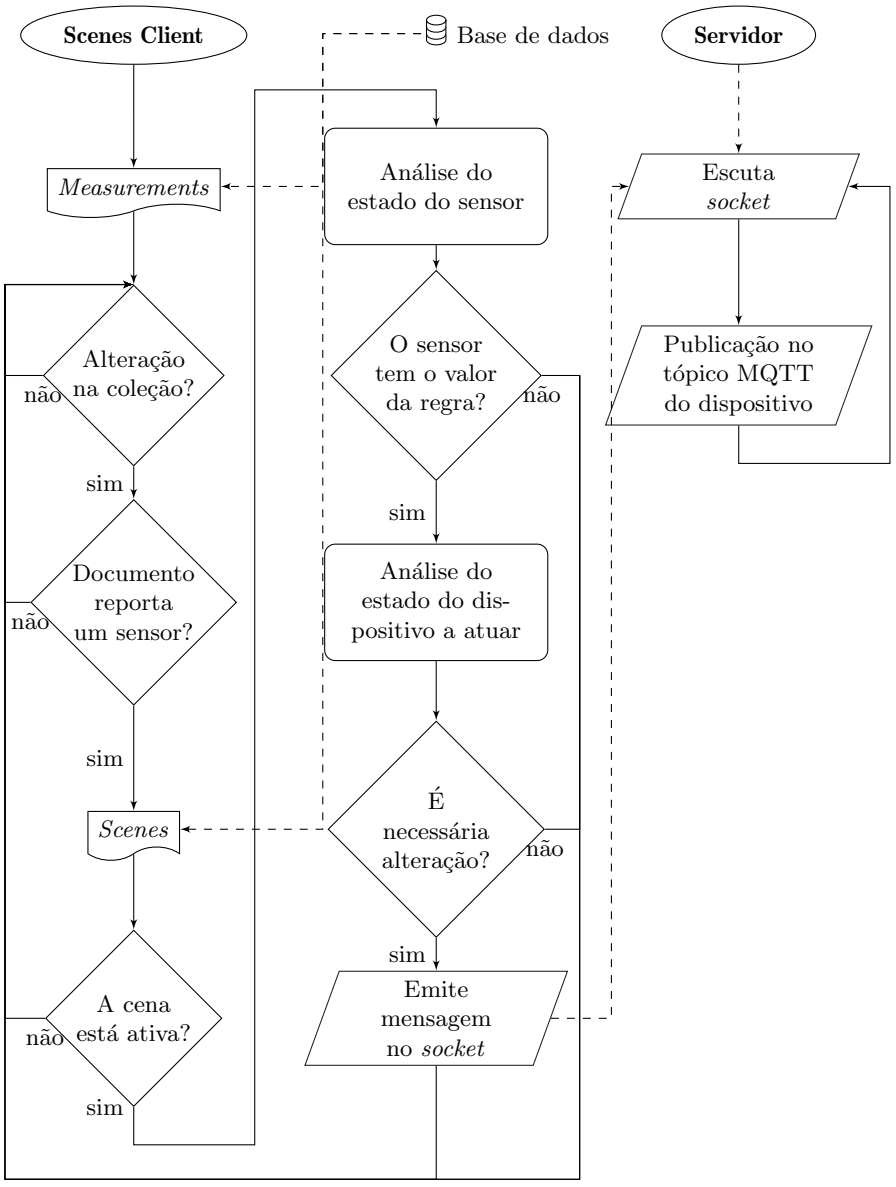


Figura 4.25: Fluxograma da função responsável pelos cenários simples

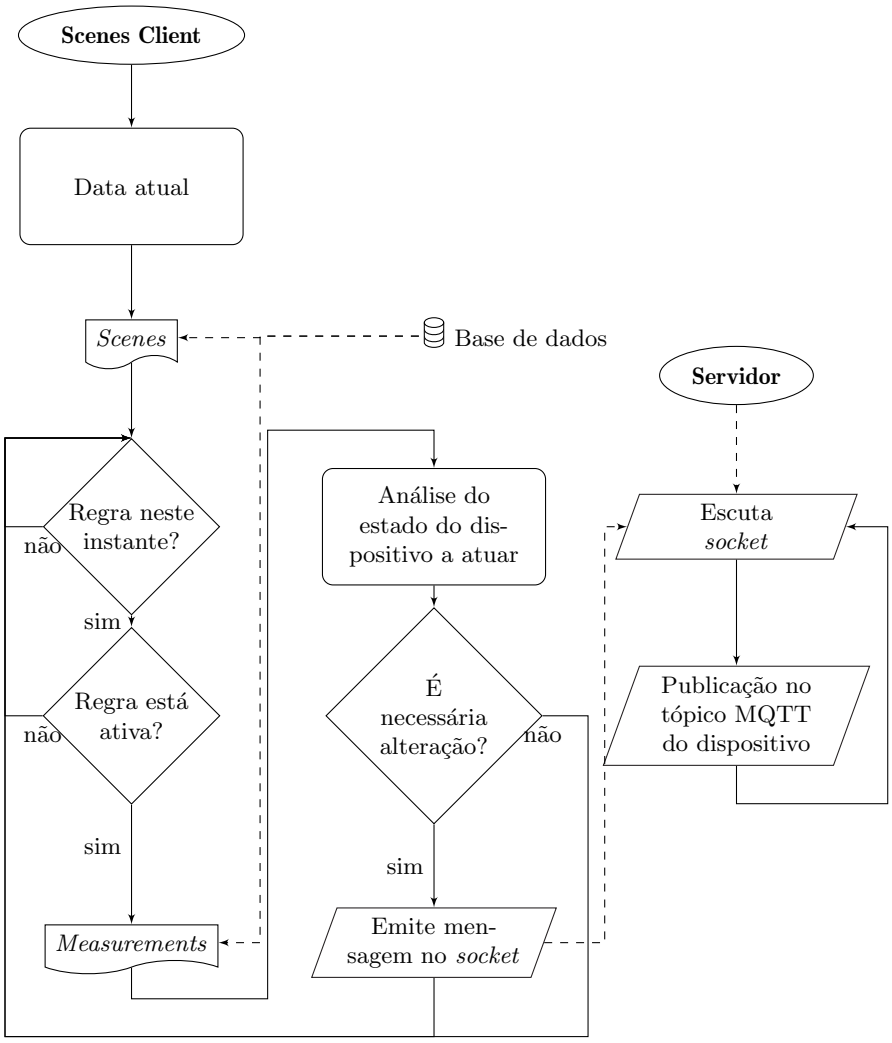


Figura 4.26: Fluxograma da função responsável pelos cenários com data

Capítulo 5

Testes e Resultados

Neste capítulo são apresentados os resultados obtidos após vários testes realizados para o correto funcionamento do sistema. É dividido em três sub capítulos associados a cada fase da implementação: servidor, base de dados e interface gráfica.

5.1 Servidor

É acrescentada a funcionalidade para que o utilizador consiga ter acesso ao número de clientes que estão a utilizar este servidor, tal como sugere a figura 5.1. Todas as mensagens provenientes do *zigbee2mqtt* são registadas na base de dados, sendo impresso na consola o tópico correspondente à operação. Opta-se também pela impressão nesta consola de todas as mensagens recebidas pelos clientes.

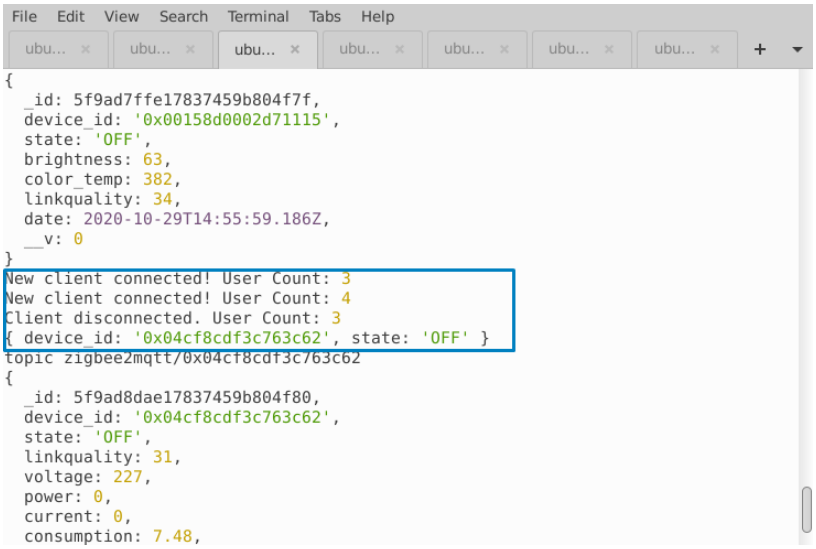


Figura 5.1: Consola do servidor implementado

Na adição de um novo dispositivo na rede a consola imprime o objeto inserido na coleção *Devices*. Indicando o endereço do dispositivo, modelo, fabricante e a sua descrição, conforme apresentado na figura 5.2.

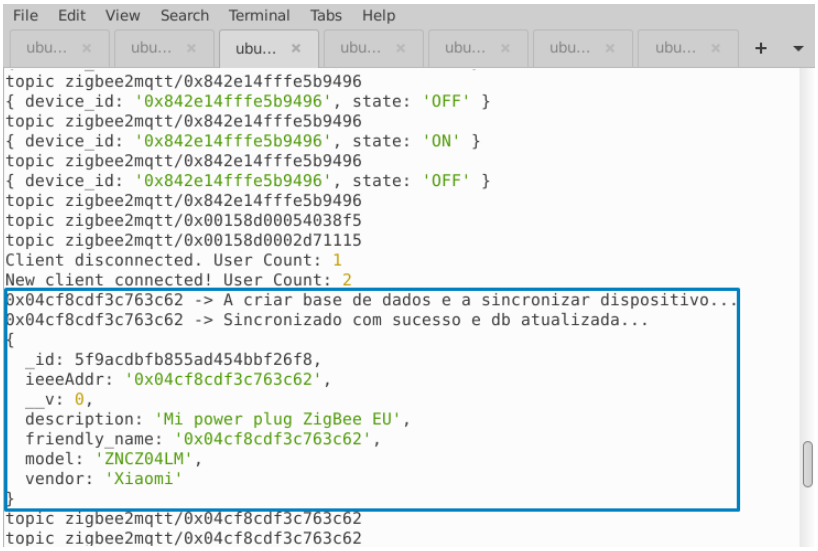


Figura 5.2: Consola do servidor na adição de um novo dispositivo

A figura 5.3 representa a remoção de um dispositivo na rede. A consola imprime uma mensagem informando o utilizador que o documento associado ao dispositivo é eliminado na base de dados.

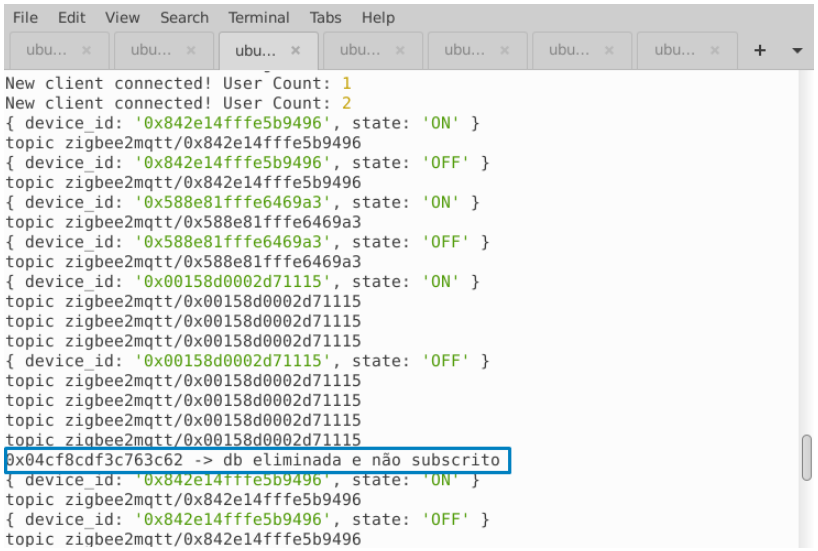


Figura 5.3: Consola do servidor na remoção de um novo dispositivo

Na figura 5.4 é apresentada a consola no instante em que é enviada uma nova

medição pelo dispositivo. É impresso o objeto inserido na coleção *Measurements*.

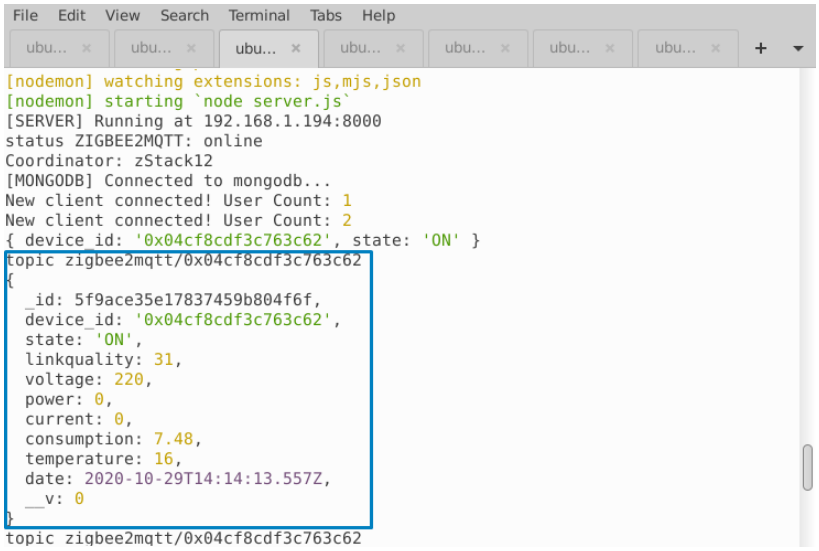


Figura 5.4: Consola do servidor na adição de uma nova medição

Um dos propósitos deste servidor servir uma interface gráfica *online*. Como tal, é necessário proceder à configuração para que a comunicação com um sistema externo seja possível. Neste sentido, é configurado o *router* da moradia para que torne a porta responsável publica. Os pontos seguintes relatam o procedimento para o *router* TG789vac v2:

1. Na aba "Ferramentas" da interface é necessário aceder ao tópico "Partilha de jogos e aplicações";

2. Cria-se uma nova aplicação, para este exemplo, designada por "SERVER";

3. Após a conclusão do passo anterior, deve-se associar essa aplicação a um dispositivo. É questionado o endereço do dispositivo na rede e a porta a tornar pública.

É de salientar que o *router* configura o acesso às portas publicas para dois protocolos: *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP). O resultado obtido após esta configuração é apresentado na figura 5.5.

Jogo ou aplicação	Dispositivo	Log	CONE	Versão IP	Protocolo	Intervalo de portas	Converter em...	Protocolo de trigger	Port tran
FTP Server	192.168.1.253	Não	Sim	IPv4	TCP	21 - 21	21 - 21	-	-
FTP Server	192.168.1.253	Não	Não	IPv4	TCP	21800 - 21805	21800 - 21805	-	-
SERVER	ubuntu-4	Não	Não	IPv4	TCP	8000 - 8000	8000 - 8000	-	-
SERVER	ubuntu-4	Não	Não	IPv4	UDP	8000 - 8000	8000 - 8000	-	-

Figura 5.5: Configuração de portas públicas do *router*

5.2 Base de dados

A figura 5.6 apresenta a consola da base de dados, onde o utilizador consegue acessar a toda a informação armazenada. A base de dados para este projeto é designada por `UserDB`. Com o método `find()` é possível aceder ao conteúdo das coleções. No exemplo desta figura, são apresentados os documentos associados aos dispositivos conectados na rede através da coleção *Devices*.

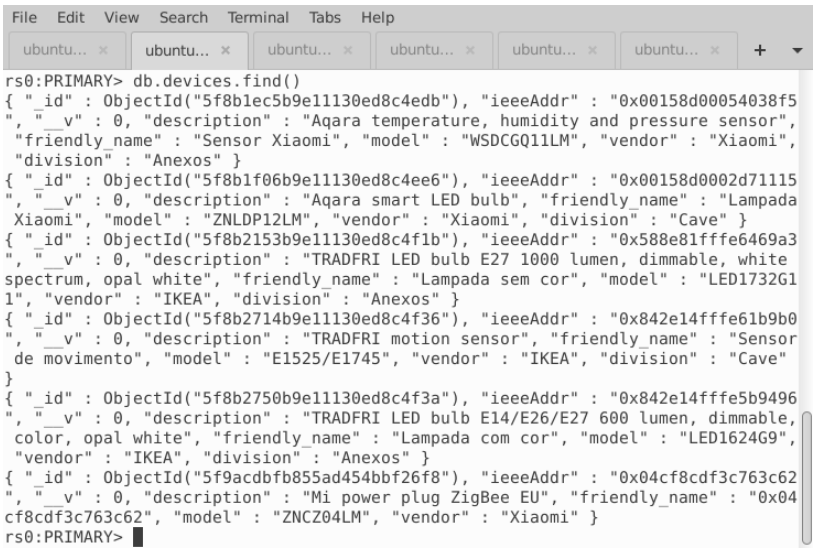


Figura 5.6: Consola da base de dados

De forma análoga, para as restantes coleções, é necessário executar na consola `mongo` os comandos apresentados na tabela 5.1.

Comando	Função
`db.divisions.find()`	Conteúdo da coleção *Divisions*
`db.graphics.find()`	Conteúdo da coleção *Graphics*
`db.scenes.find()`	Conteúdo da coleção *Scenes*

Tabela 5.1: Comandos para aceder às respetivas coleções MongoDB

5.3 Interface Gráfica

A interface gráfica, como já mencionado, opta-se por hospedar num servidor externo. No entanto, a mesma interface terá um servidor local. Isto permite que, em caso de falha da Internet, o utilizador através do seu *router* tenha a capacidade de gerir a rede com o controlo dos dispositivos, gráficos e cenários. Para que esta interface possa ser acessível em todos os *gadgets* sem que ocorram erros na formatação, recorreu-se à implementação de uma aplicação responsiva. Isto é, esta ajusta-se de acordo com o ecrã a apresentar. Tal é possível graças ao método `Grid` do módulo `@material-ui` do React. Este módulo oferece diversas funcionalidades à aplicação como o uso de símbolos, botões responsivos e modais. Designam-se modais as janelas tipo *pop-up* que são exibidas quando um botão é selecionado.

Como pretendido, na página inicial da aplicação são exibidos os dispositivos, os respetivos dados e controlos. Esta página é representada na figura 5.7.

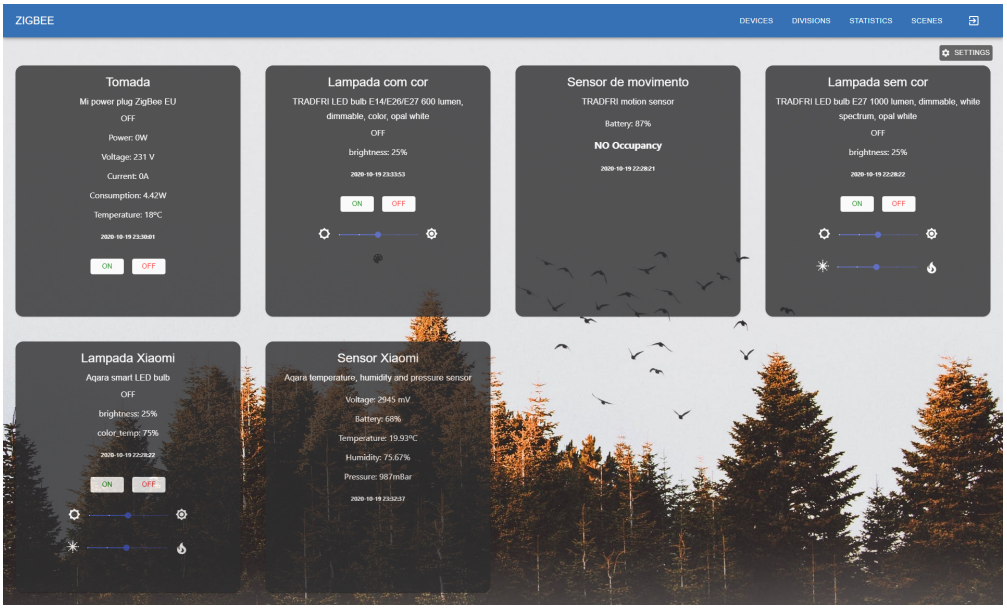


Figura 5.7: *Devices Page*

No canto superior direito desta página encontra-se o botão “Settings” que quando clicado apresenta a página associada à configuração dos dispositivos, figura 5.8. Aqui, o utilizador para além de analisar as especificações de cada dispositivo, tem a capacidade de alterar a localização, de editar a nomenclatura e de remover os dispositivos da rede.

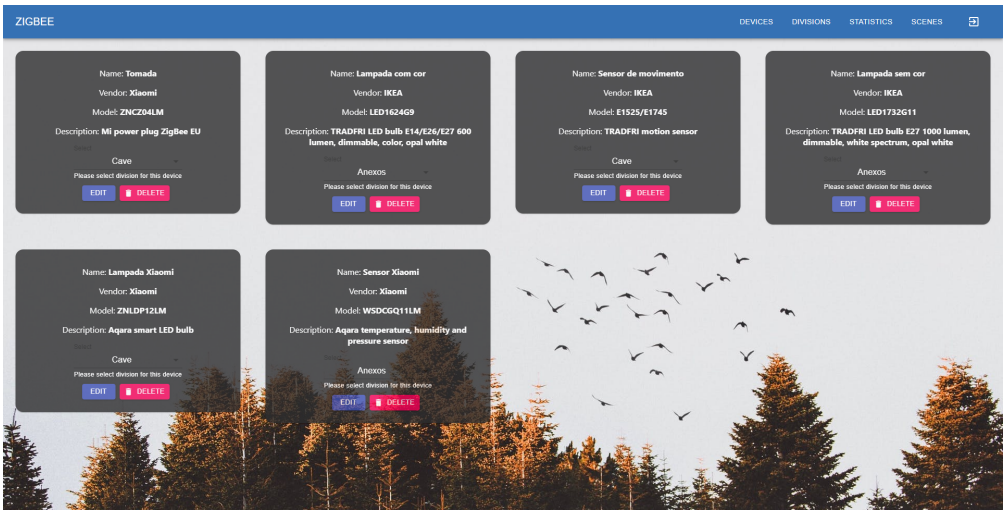


Figura 5.8: *Devices Settings Page*

Quando selecionado o botão "Edit" é apresentado um modal que indica o nome atual do dispositivo e questiona o utilizador pela nova designação, como apresentado na figura 5.9.

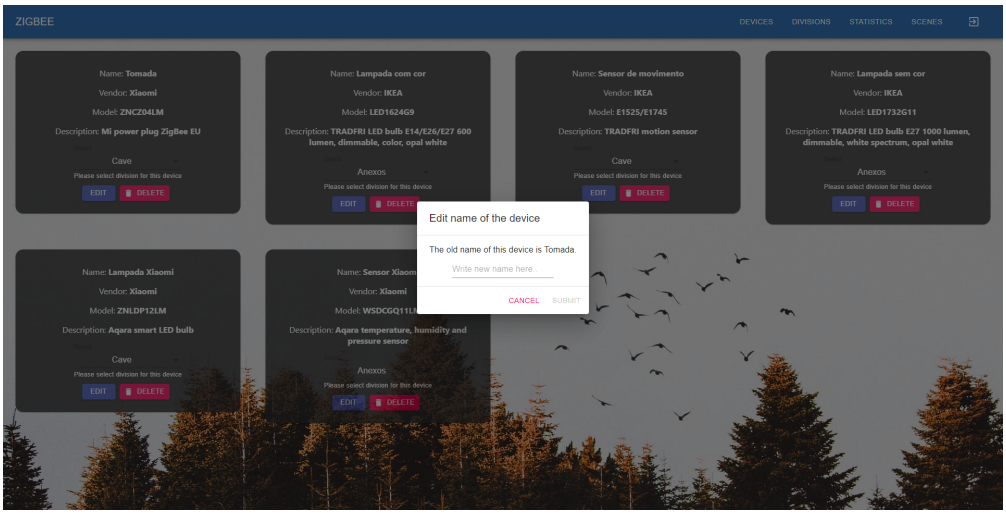


Figura 5.9: Modal da *Devices Settings Page*

Relativamente à localização do dispositivo, o utilizador ao aceder à *Divisions Page* é informado com as divisões já criadas e os dispositivos associados. Tal é apresentado na figura 5.10.

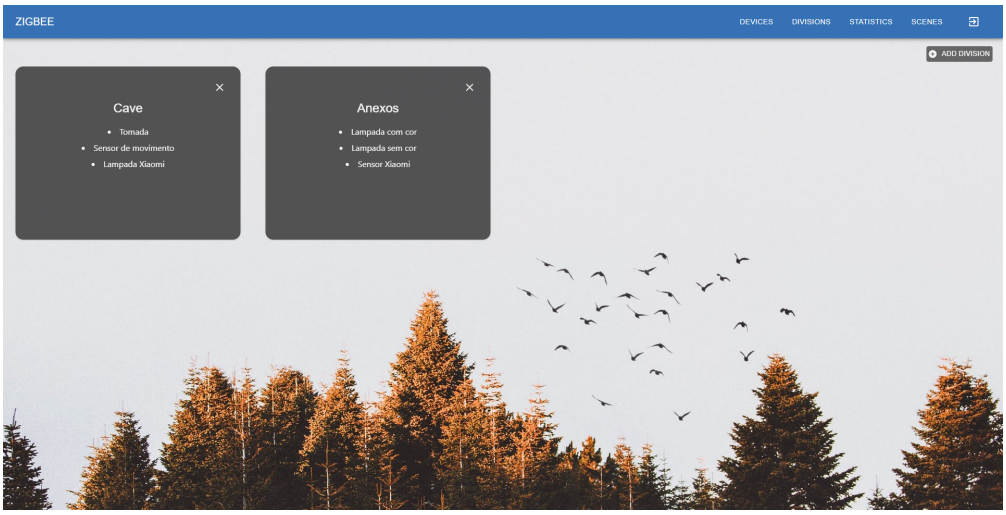


Figura 5.10: *Divisions Page*

Para novas divisões, o utilizador necessita de selecionar o botão "Add Division", no canto superior direito, e aqui será questionada a nova designação. Após esta inscrição é apresentada a nova divisão. O modal responsável por esta operação é apresentado na figura 5.11.

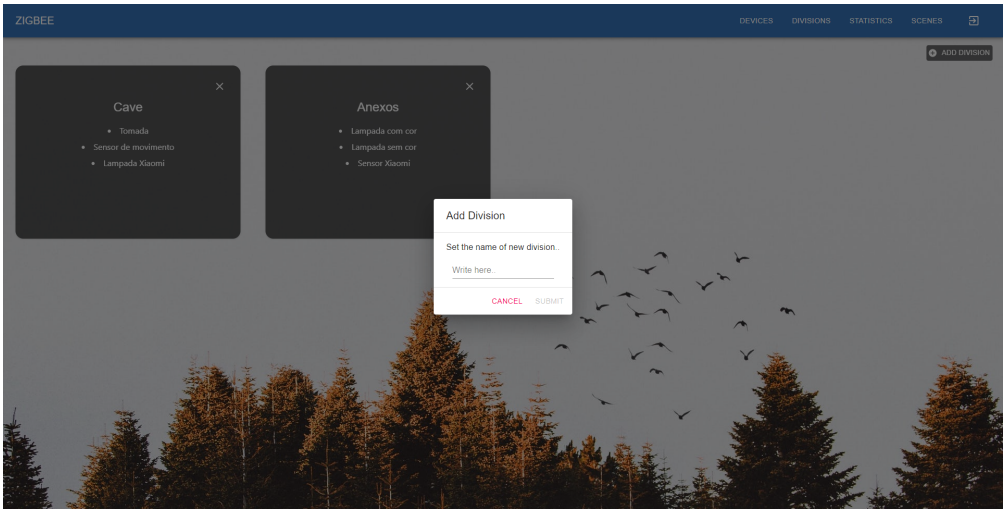


Figura 5.11: Modal da *Divisions Page*

A *Statistics Page* é responsável por apresentar ao utilizador a alteração, ou não, de dados em função do eixo temporal, tal como apresentado na figura 5.12.

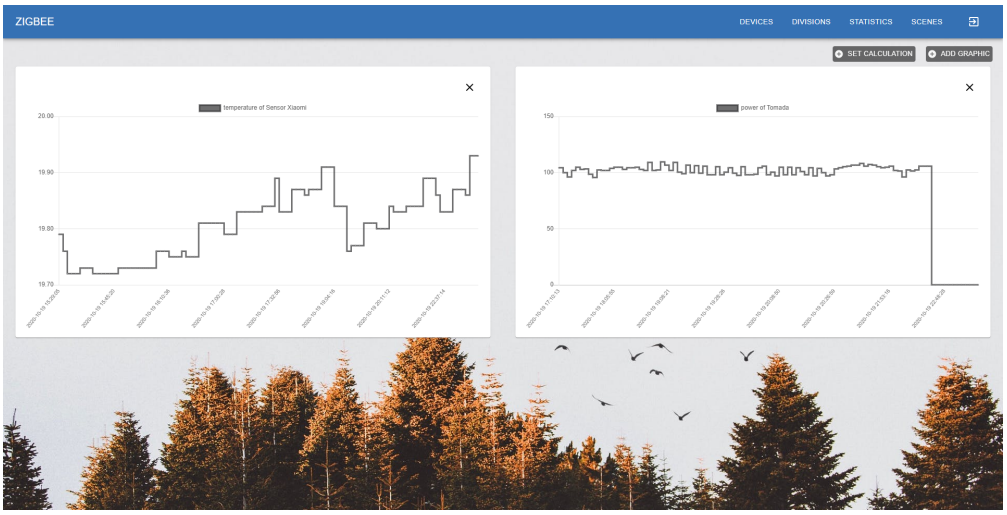


Figura 5.12: *Statistics Page*

Nesta página, os gráficos são apresentados conforme a escolha do utilizador. Para a inserção de um novo gráfico na base de dados é necessário selecionar o botão "Add Graphic", o utilizador será questionado sobre o dispositivo e o respetivo parametro a apresentar, figura 5.13. Após a submissão o gráfico é apresentado.

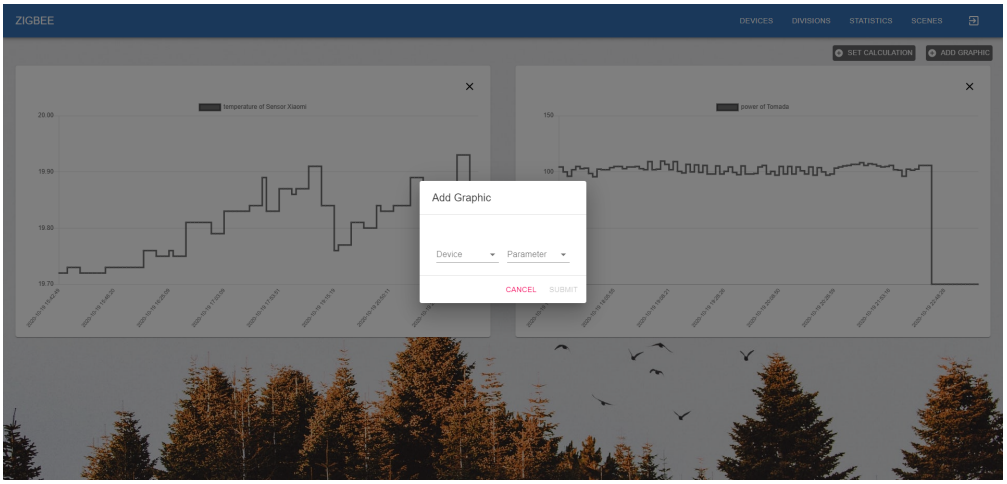


Figura 5.13: Modal I da *Statistics Page*

Para além da exibição dos dados através de gráficos, é acrescentada a funcionalidade de calculo do valor médio do parâmetro durante um intervalo de tempo. Para esta operação, o utilizador deverá selecionar o botão "Set Calculation". Irão ser questionados o dispositivo, o parâmetro e o intervalo de tempo associado. O exemplo da figura 5.14 apresenta o valor médio da tensão do dispositivo no intervalo de tempo escolhido.

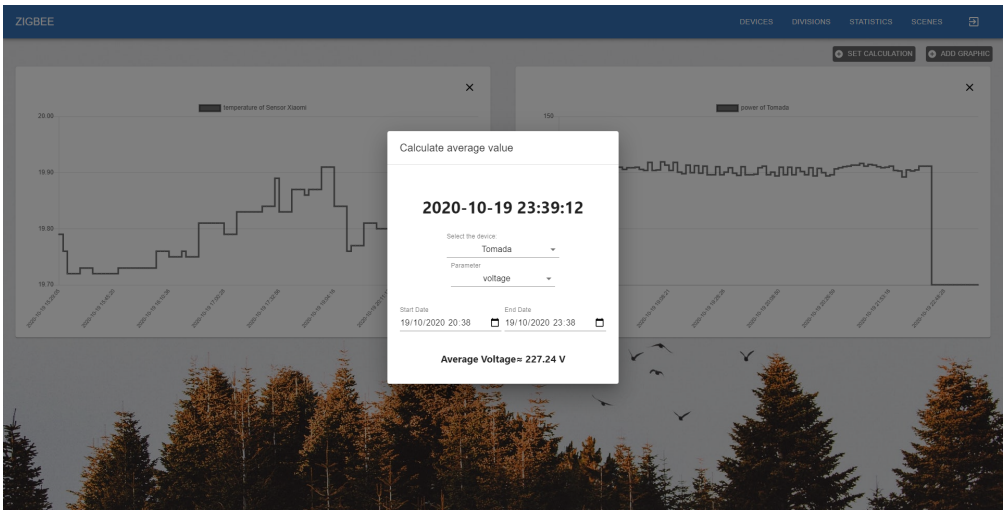


Figura 5.14: Modal II da *Statistics Page*

Ainda dentro deste contexto, caso o utilizador selecione o parâmetro "Power" é efetuado o calculo da energia para o mesmo intervalo de tempo. Trata-se de uma funcionalidade que permite estimar consumos e custos associados aos dispositivos sendo questionado o custo do kWh, como apresentado na figura 5.15.

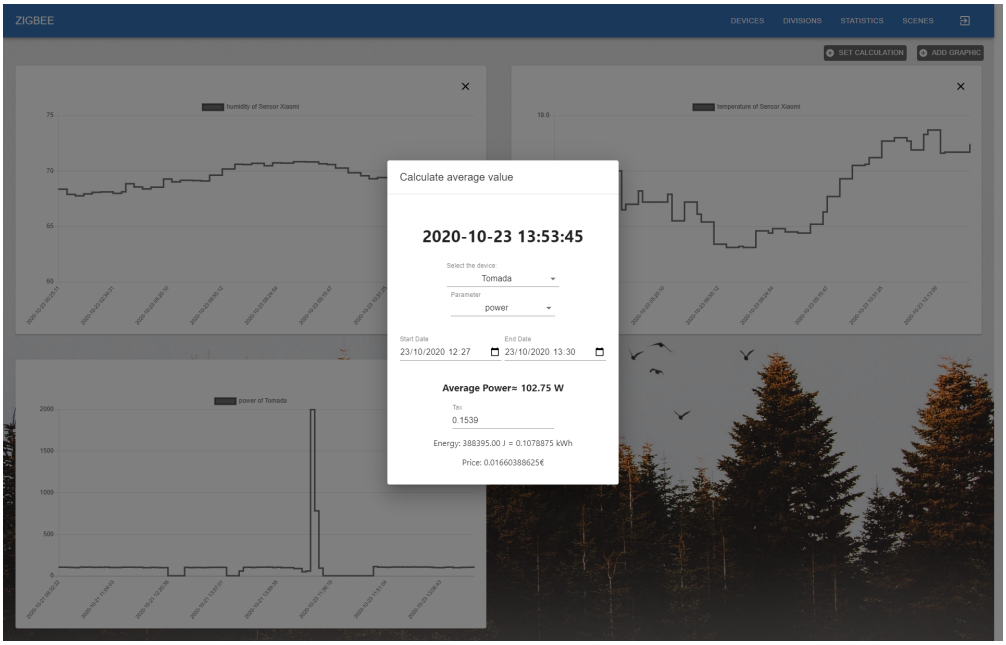


Figura 5.15: Modal III da *Statistics Page*

Os cenários são apresentados na *Scenes Page*, o utilizador tem a possibilidade de analisar as regras já criadas e proceder à sua ativação ou desativação. Esta

página é representada na figura 5.16. As condições são dispostas de forma a garantir a fácil interpretação do utilizador.

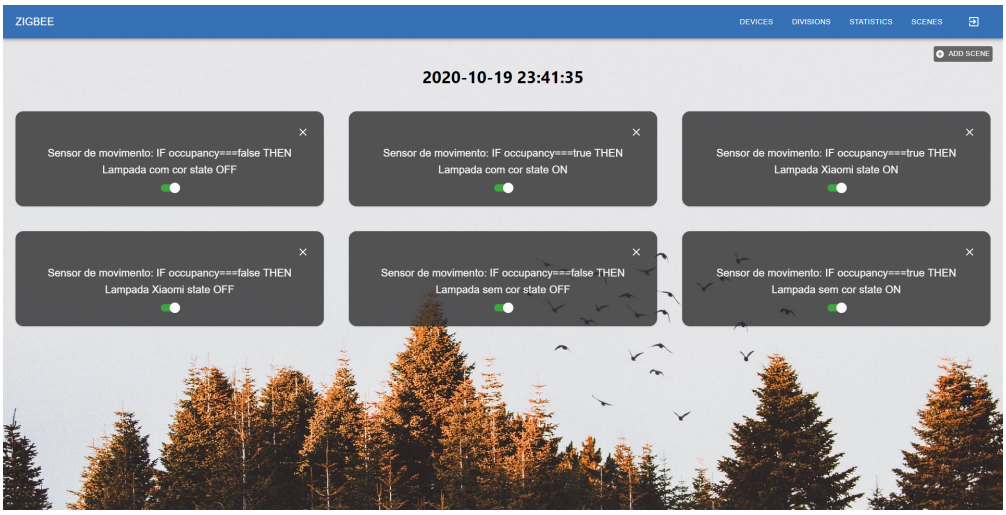


Figura 5.16: *Scenes Page*

Ao selecionar o botão "Add Scene" é apresentado o modal que permite ao utilizador a inserção de uma nova regra no sistema. Aqui o utilizador pode optar por uma regra simples ou uma regra com data. A lógica a definir para uma regra simples é apresentada na figura 5.17. O utilizador terá de selecionar os dispositivos sensor e atuador, os parametros e os respetivos valores.

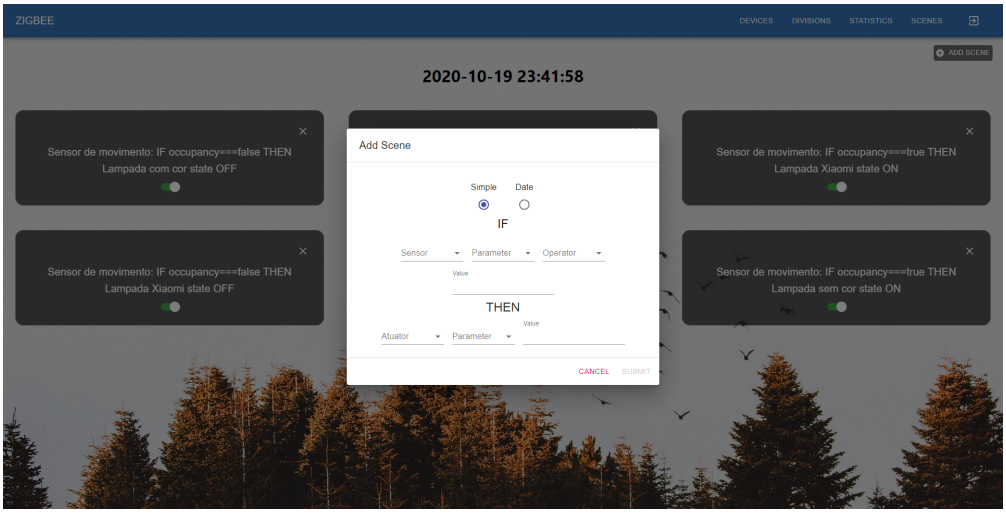


Figura 5.17: Modal I da *Scenes Page*

Para a segunda opção, o utilizador terá de selecionar o instante de tempo para atuar um dispositivo. Tem ainda como parâmetros requeridos a escolha do

dispositivo a atuar, o parâmetro e o respetivo valor. Tal opção é apresentada na figura 5.18.

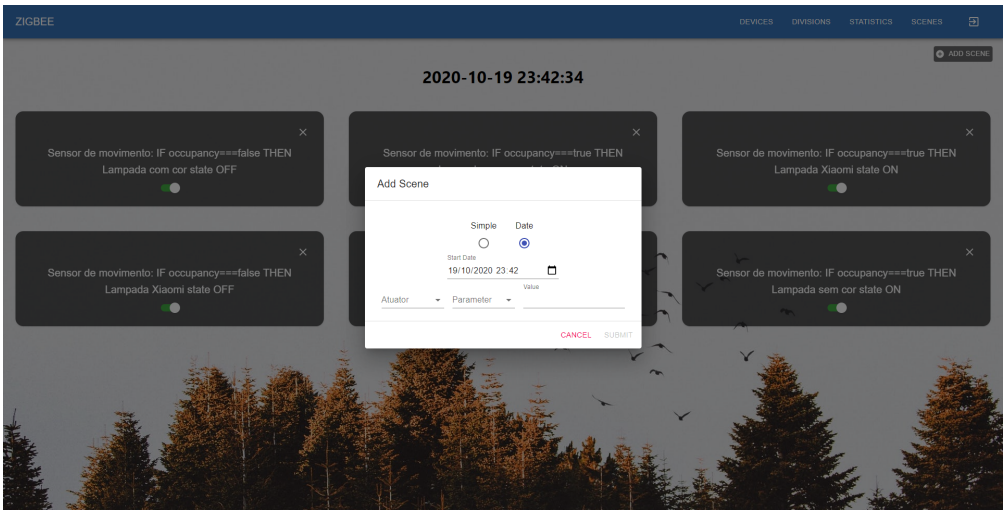


Figura 5.18: Modal II da *Scenes Page*

5.4 Scenes Client

A figura 5.19 representa a consola deste cliente. Aqui são analisados os cenários e enviadas as respetivas mensagens para o servidor a fim de atuar os dispositivos. Quando ocorre a alteração da coleção associada aos dados dos dispositivos, é verificado se o ultimo documento inserido corresponde a um dispositivo que tenha alguma regra a funcionar como sensor. Em caso afirmativo é impressa na consola essa mesma mensagem. É ainda analisada a coleção *Measurements* para verificar se é necessário proceder à alteração de estado do dispositivo e caso se confirme, a informação é relatada na consola. Isto previne o envio de informação desnecessária, podendo estar o dispositivo a atuar no estado pretendido.

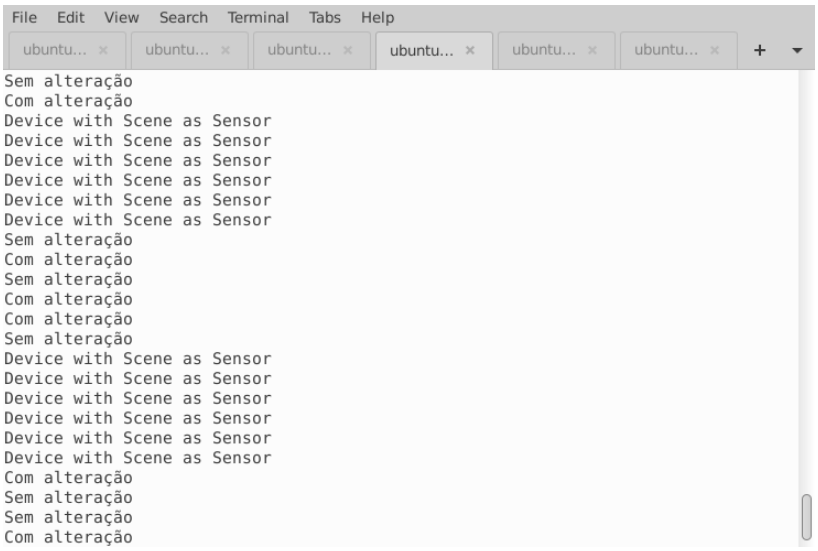


Figura 5.19: Consola *Scenes Client*

5.5 Análise de recursos

Este ponto é crucial para verificar se a aplicação se encontra totalmente otimizada. Para este projeto recorre-se à versão de 8 GB de memória RAM do RaspBerry Pi Model 4 B, no entanto como mencionado no sub-capítulo 3.4 é possível executar as principais funcionalidades na versão de 2 GB do RaspBerry Pi Model 4 B.

Nesta versão, a execução da aplicação (sem interface gráfica local) implica o consumo de memória RAM de cerca de 10% da capacidade máxima, como apresentado na figura 5.20. Assim, seria possível recorrer ao modelo de 1 GB do RaspBerry Pi Model 3, no entanto 70% da capacidade máxima estaria a ser utilizada. O comando `htop` permite verificar o consumo da memória RAM e do uso do processador para a execução das diversas componentes em distribuições Linux.

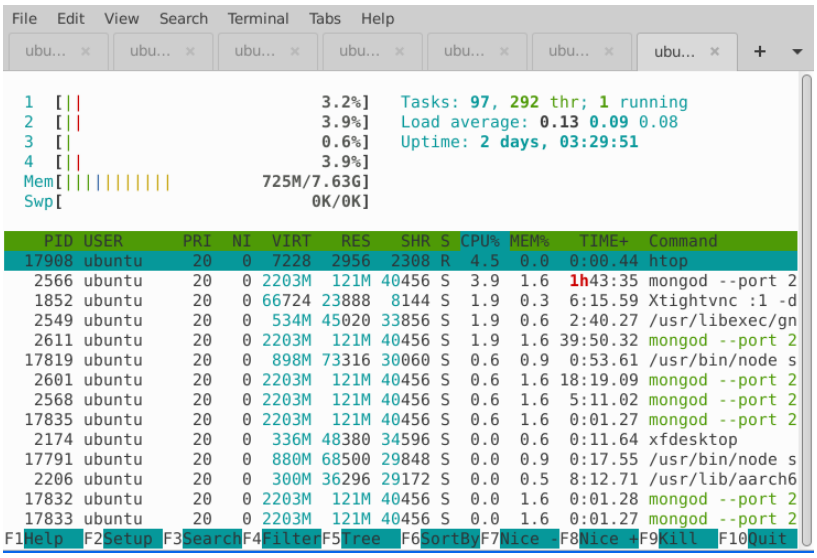


Figura 5.20: Especificações do consumo de recursos da aplicação

Já com a interface gráfica hospedada na máquina, o consumo médio da memória RAM ronda o 1 GB, tal como apresenta a figura 5.21. Algo que já não é recomendado com a versão de 1 GB do RaspBerry Pi Model 3 B. Assim, para o correto funcionamento de todas as componentes sem ser necessário sobrecarregar o equipamento, é recomendado o uso da versão de 2 GB ou superior.

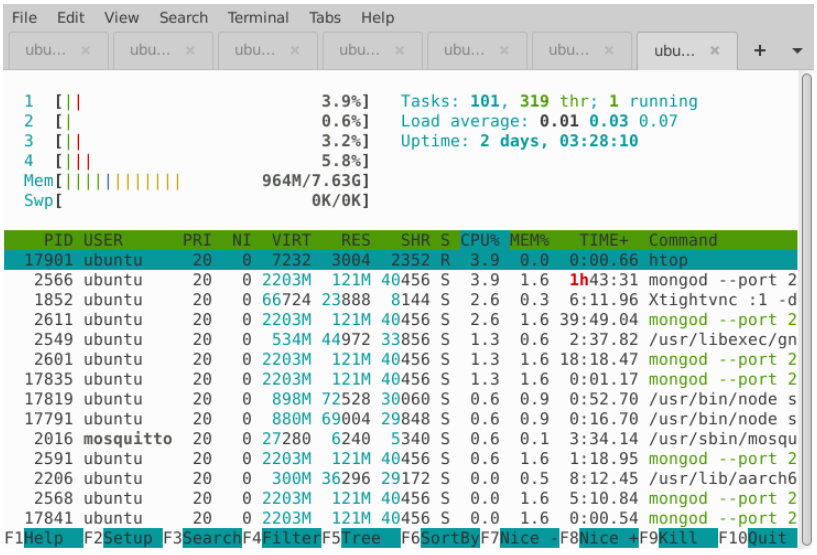


Figura 5.21: Especificações do consumo de recursos da aplicação II

Capítulo 6

Conclusão

Neste capítulo são apresentadas as conclusões do projeto e são referidas eventuais melhorias a implementar.

6.1 Conclusões do trabalho

Apesar do protocolo ZigBee ainda não ter uma grande variedade no mercado existem equipamentos que permitem cumprir com os requisitos mínimos para satisfazer as necessidades dos seus utilizadores. Como vantagem destaca-se o custo dos equipamentos, não sendo necessário recorrer a grandes especificações ou despesas energéticas, uma vez que todos os equipamentos apresentam um consumo de energia otimizado. O uso do RaspBerry Pi como controlador permite uma grande versatilidade de pacotes, já que este se comporta como um computador pessoal comum com a benesse de ser de pequenas dimensões.

Relativamente ao servidor, a escolha do Node.js facilita o processo de implementação já que existem módulos desenvolvidos para as finalidades requeridas. O uso de *sockets* para a comunicação com os clientes permite poupar recursos de processamento. A comunicação é estabelecida uma única vez, não sendo necessário efetuar pedidos de dados periodicamente.

A interface gráfica é desenvolvida com a principal finalidade de garantir uma fácil interação com o utilizador. O uso de um servido externo para hospedar esta aplicação permite o acesso ao sistema mesmo estando fora da residência. O inconveniente de estar dependente deste servidor é contornado através da disposição desta aplicação no RaspBerry Pi, garantindo o acesso local.

Em suma, o protótipo funcional cumpre os requisitos previamente estipulados, sendo uma solução eficaz, otimizada e com o menor investimento possível.

6.1.1 Ideias para Desenvolvimentos Futuros

Neste ponto é de mencionar que na página associada aos cenários o sistema permite apenas a adição de uma condição com um dispositivo sensor e um dispositivo atuador. É possível definir um valor máximo para a potência fornecida pela tomada de forma a garantir que esta desligue se o valor for ultrapassado. Esta aplicação é útil para instalações com paineis fotovoltáicos, já que o funcionamento da tomada poderá depender da potência produzida pelo painel. Em contrapartida, o sistema aqui apresentado não está preparado para somar as potências dos vários dispositivos da instalação, de forma a permitir o deslastre de cargas em função da potência produzida pelo painel.

Será interessante que num desenvolvimento futuro o sistema seja capaz de aceder à potência produzida por uma fonte renovável e o utilizador definir os atuadores como prioritários ou não prioritários, garantido desta forma a otimização do consumo energético da instalação.

Bibliografia

[1] Luz e Som, "Domótica: Definição, Vantagens, Exemplos e Orçamento." https://www.luzesom.pt/pt/instalacao-de-equipamentos/domotica/, 2020. [cited on p. 5]

[2] Centro Nacional de Cibersegurança, "A Internet das Coisas (IoT – Internet of Things)." https://www.cncs.gov.pt/a-internet-das-coisas-iot-internet-of-things/, 2020. [cited on p. 7]

[3] IoT Now Magazine, "IoT and home automation: What does the future hold?." https://www.iot-now.com/2020/06/10/98753-iot-home-automation-future-holds/, 2020. [cited on p. 7]

[4] John Carlsen, "Smart homes: A guide to Home Automation Protocols." https://www.toptenreviews.com/a-guide-to-home-automation-protocols, 2020. [cited on p. 8, 9, 10]

[5] SmartHome, "What is X10 Home Automation." https://www.smarthome.com/pages/sc-what-is-x10-home-automation), 2020. [cited on p. 8]

[6] Margaret Rouse, "What is INSTEON protocol?." https://whatis.techtarget.com/definition/INSTEON, 2020. [cited on p. 8]

[7] Derek Crippin, "Home Automation Protocols Guide." https://www.alarmnewengland.com/blog/home-automation-protocols, 2020. [cited on p. 8]

[8] Michael Noll, "Comparison of Home Automation Protocols." https://theiotpad.com/tips/home-automation-protocols, 2018. [cited on p. 9, 10]

[9] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on zigbee technology," *3rd International Conference on Electronics Computer Technology*, 2011. [cited on p. 14, 18, 20, 21]

[10] Tutorialspoint, "Digital Communication - Phase Shift Keying." https://www.tutorialspoint.com/digital_communication/digital_communication_phase_shift_keying.htm, 2020. [cited on p. 14]

[11] GaussianWaves, "BPSK modulation and demodulation." https://www.gaussianwaves.com/2010/04/bpsk-modulation-and-demodulation-2/, 2010. [cited on p. 15]

[12] All About Circuits, "Digital Phase Modulation: BPSK, QPSK, DQPSK." https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-modulation/digital-phase-modulation-bpsk-qpsk-dqpsk/, 2020. [cited on p. 15]

[13] M. E. Amer Mohamed Daeri and A. R. Zerek, "Quadrature phase shift keying and offset quadrature phase shift keying ber performance comparison," *3rd International Conference on Automation, Control, Engineering and Computer Science*, 2016. [cited on p. 17]

[14] Eletronics Notes, "Direct Sequence Spread Spectrum: the basics." https://www.electronics-notes.com/articles/radio/dsss/what-is-direct-sequence-spread-spectrum.php, 2020. [cited on p. 18]

[15] Leandro Sehnem Heck and Luis Henrique Kleber, "Automação Residêncial," 2007. [cited on p. 18, 30]

[16] Faculdade de Engenharia da Universidade do Porto, "Porquê Zigbee?," 2020. [cited on p. 18]

[17] R. Faludi, *Building Wireless Sensor Networks*. O'REILLY, 2010. [cited on p. 19, 20, 21, 30, 31]

[18] E-SPIN, "Zigbee Network Topology | E-SPIN Group." https://www.e-spincorp.com/zigbee-network-topology/, 2020. [cited on p. 21]

[19] P. Chhimwal, D. Rawat and D. S. Rai, "Study of beacon mode collision problem in the ieee 802.15.4/zigbee," *International Journal of Computer Science and Engineering*, 2013. [cited on p. 23]

[20] Embedded Staff, "Home networking with Zigbee." https://www.embedded.com/home-networking-with-zigbee/, 2004. [cited on p. 23]

[21] Libelium, "Security in 802.15.4 and Zigbee Networks." https://www.libelium.com/libeliumworld/security-802-15-4-zigbee/, 2009. [cited on p. 24, 26]

[22] RF Wireless World, "Zigbee MAC layer Frame Formats." https://www.rfwireless-world.com/Tutorials/Zigbee-MAC-layer-frame-format.html, 2020. [cited on p. 27]

[23] euroX10, "Home Center 3." https://www.eurox10.com/pt/home-center-3, 2020. [cited on p. 34]

[24] Vesternet, "VERA Controller Comparison." `https://www.vesternet.com/pages/vera-controller-comparison`, 2020. [cited on p. 34]

[25] L. Duarte, "Administrando MongoDB: Replicação/Espelhamento." `https://www.luiztools.com.br/post/administrando-mongodb-replicacao-espelhamento/`, 2019. [cited on p. 46]

Anexo A

Excertos de código

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", (req, res) => {
    res.send("Hello World!")
})

app.listen(port, () => {
    console.log("Listening at http://localhost:${port}")
})
```

Excerto de código A.1: Exemplo de um servidor HTTP implementado com *Express.js*

```
const app = require("express")();
const server = require("http").createServer(app);
const options = { /* ... */ };
const io = require("socket.io")(server, options);

io.on("connection", (socket) => {
    console.log("User Connected!");
    socket.on("disconnect", () => {
        console.log("User Disconnected!");
    })
});

server.listen(3000);
```

Excerto de código A.2: Exemplo de um servidor HTTP implementado com *Express.js* e *Socket.io*

```
const io = require("socket.io-client");
const socket = io("http://localhost:3000");

socket.on("connect", () => {
    console.log(socket.connected); // true
});

socket.on("disconnect", () => {
        console.log(socket.connected); // false
});
```

Excerto de código A.3: Exemplo de um cliente suportado por *Socket.io*

```
const io = require("socket.io-client");
const socket = io("http://localhost:3000");

socket.on("connect", () => {
    socket.emit("channel1", "teste");
});
```

Excerto de código A.4: Exemplo de aplicação *socket.emit()*

```
const app = require("express")();
const server = require("http").createServer(app);
const options = { /* ... */ };
const io = require("socket.io")(server, options);

io.on("connection", (socket) => {
    socket.on("channel1", data => console.log(data));
});

server.listen(3000);
```

Excerto de código A.5: Exemplo de aplicação *socket.on()*

```
const moongose = require("mongoose");
const Schema = moongose.Schema;

const eventSchema = new Schema({
    type: {
        type: String
    },
    model: {
        type: String
    },
    vendor: {
        type: String
    },
    description: {
        type: String
    }
});

module.exports = moongose.model("Devices", eventSchema);
```

Excerto de código A.6: Estrutura de um ficheiro *mongoose.js*

```
mongoose.connect(MONGODB_URL, {useNewUrlParser: true,
useUnifiedTopology: true})
.then(() => {
    console.log("[MONGODB] Connected to mongodb...")
})
```

Excerto de código A.7: Conexão do servidor com a base de dados

```
Devices.watch().on("change", () => {
    console.log("NEW DATA");
    //Envio de novos dados para o(s) cliente(s)
})
```

Excerto de código A.8: *Change Streams* no Node.js

```
var mqtt = require("mqtt");
var client = mqtt.connect("mqtt://192.168.1.194:1883");

client.on("connect", () => {
   client.subscribe("#");
})
```

Excerto de código A.9: Conexão servidor - *broker*

```
...
client.on("message", (topic, message) => {
    if (topic === "zigbee2mqtt/bridge/state"){
            console.log("status ZIGBEE2MQTT: ${message}");
    }
})
...
```

Excerto de código A.10: Exemplo de aplicação I módulo MQTT

```
...
    else if(topic === "zigbee2mqtt/bridge/log"){
      context = JSON.parse(message);
      if (context["type"] === "device_connected"){
        const new_device = new Devices({
          ieeeAddr: context["message"].friendly_name
      })
      new_device.save()
      .then(data => {
          console.log(data);
      })
      .catch(err => {
          console.log(err);
           throw err;
      });
    }
...
```

Excerto de código A.11: Inserção de um novo dispositivo na base de dados

```
...
    else if (context["type"] === "pairing" && context["meta"
       ].supported === true){
      Devices.findOneAndUpdate({ieeeAddr:
      context["meta"].friendly_name},{$set:{
        model: context["meta"].model,
        friendly_name: context["meta"].friendly_name,
        vendor: context["meta"].vendor,
        description: context["meta"].description
      }}, {new: true}, (err, doc) => {
        if (err) console.log("something wrong");
            console.log(doc);
      });
    }
...
```

Excerto de código A.12: Atualização do documento de um novo dispositivo

```
...
    else if (context["type"] === "device_force_removed"){
      let id = context["message"];
      Devices.deleteOne({ ieeeAddr: `${id}`},
      function (err) {
        if (err) return handleError(err);
      });
    }
...
```

Excerto de código A.13: Eliminação do documento associado ao dispositivo removido

```
...
    Devices.find().then(data => data.forEach(i => {
        if(topic === "zigbee2mqtt/" + i.ieeeAddr){
          context = JSON.parse(message);
          const new_measurement =  new Measurements({
            device_id: i.ieeeAddr,
            state: context["state"],
            brightness: context["brightness"],
            color_temp: context["color_temp"],
            linkquality: context["linkquality"],
            battery: context["battery"],
            voltage: context["voltage"],
            power: context["power"],
            current: context["current"],
            consumption: context["consumption"],
            temperature: context["temperature"],
            humidity: context["humidity"],
            pressure: context["pressure"],
            color: context["color"],
            occupancy: context["occupancy"],
            date: new Date()
          })
          new_measurement.save()
            .then(data => {
              console.log(data);
            })
            .catch(err => {
                    console.log(err);
                    throw err;
            });
        }
      }));
...
```

Excerto de código A.14: Inserção de novos dados de um dispositivo

```
const firebaseConfig = {
    apiKey: "AIzaSyDYuIqMiOoD440HwSL4soyoOMeCQZg9ATQ",
    authDomain: "application-51871.firebaseapp.com",
    databaseURL: "https://application-51871.firebaseio.com",
    projectId: "application-51871",
    storageBucket: "application-51871.appspot.com",
    messagingSenderId: "911190426067",
    appId: "1:911190426067:web:3489729289d9bb220faf26",
    measurementId: "G-ZWJQPJNL5B"
};
```

Excerto de código A.15: Exemplo de chave de configuração da aplicação de autenticação

```
const App = () => {
  return (
    <AuthProvider>
      <Router>
        <div>
          <PrivateRoute exact path="/" component={Home} />
          <Route exact path="/login" component={Login} />
          <Route exact path="/signup" component={SignUp} />
        </div>
      </Router>
    </AuthProvider>
  );
};
```

Excerto de código A.16: Exemplo de aplicação de autenticação

```
const ENDPOINT = "192.1.1.1"; //server address
const socket = io(ENDPOINT);
```

Excerto de código A.17: Conexão do cliente com o servidor

```
<Route exact path="/" render={(props) =>
    (<DevicesPage {...props} socket={socket}/>)}
/>

<Route exact path="/divisions" render={(props) =>
    (<DivisionsPage {...props} socket={socket}/>)}
/>

<Route exact path="/statistics" render={(props) =>
    (<StatisticsPage {...props} socket={socket}/>)}
/>

<Route exact path="/scenes" render={(props) =>
    (<ScenesPage {...props} socket={socket}/>)}
/>
```

Excerto de código A.18: *Routes* para as páginas da aplicaçãor

```
socket.emit("GETDEVICES");
socket.emit("GETMEASUREMENTS");
```

Excerto de código A.19: Pedido de dados ao servidor efetuado pela *Devices Page*

```
const [devices, setDevices] = useState([]);
const [data, setData] = useState([]);

socket.on("Devices", data => {
    setDevices(data);
})
socket.on("Measurements", data => {
    setData(data);
})
```

Excerto de código A.20: Escuta de dados *Devices Page*

```
[
    {
      ieeeAddr: "0x842e14fffe5b9496",
      model: "LED1624G9",
      vendor: "IKEA",
      description: "TRADFRI LED bulb E14/E26/E27 600 lumen,
         dimmable, color, opal white"
    }
]
```

Excerto de código A.21: Exemplo do *array devices*

```
devices.map(device => {
     <li key={device._id}>
        <p>{device.ieeeAddr}</p>
        <p>{device.model}</p>
        <p>{device.vendor}</p>
        <p>{device.descripton}</p>
     </li>
})
```

Excerto de código A.22: Exemplo de aplicação do método *map*

```
const DataFromDevice = data.filter(device =>
    device.device_id === props.id).splice(0,1);
```

Excerto de código A.23: Exemplo de filtragem do último objeto para cada dispositivo

```
DataFromDevice.map(item => (
<div key={item.device_id}>
    {item.state ? (item.state) : false}
    {item.power > -1 ? <p>Power: {
      item.power
    }W</p> : false}
    {item.voltage > -1 ? (<p>Voltage: {
      item.voltage
    }V</p>) ) : false}
    {item.current > -1 ? <p>Current: {
      item.current
    }A</p> : false}
    {item.temperature > -1 ? <p>Temperature: {
      item.temperature
    }C</p> : false}
    {item.humidity > -1 ? <p>Humidity: {
      item.humidity
    }%</p> : false}
    {item.pressure > -1 ? <p>Pressure: {
      item.pressure
    }mBar</p> : false}
    {item.occupancy === "true" ? <h3>Occupancy</h3> : false}
    {item.occupancy === "false" ? <h3>NO Occupancy</h3> :
       false}
    {item.date ? <h6>{item.date)</h6> : false}
</div>
```

Excerto de código A.24: Exemplo de aplicação II do método *map*

```
[
    {
        "model": "LED1545G12",
        "state": true,
        "brightness": true,
        "color_temp": true
    },
    {
        "model": "E1745",
        "occupancy": true
    }
]
```

Excerto de código A.25: Exemplo ficheiro JSON associado aos parâmetros dos dispositivos

```
const DeviceParameters = DevicesParameters.filter(device =>
   device.model === props.model);
```

Excerto de código A.26: Exemplo de filtragem dos parâmetros de um dispositivo

```
DeviceParameters.map(item => (
    <div key={item.model}>
        {item.state ?
          <State id={props.id} socket={socket} />
        : false}
      </div>
))
```

Excerto de código A.27: Exemplo de aplicação III do método *map*

```
<button onClick={() =>
    socket.emit("state", {device_id: props.id, state: "ON"})}
>ON</button>
<button onClick={() =>
    socket.emit("state", {device_id: props.id, state: "OFF"})
       }
>OFF</button>
```

Excerto de código A.28: Exemplo de aplicação *socket.emit*

```
socket.on("state", (msg) => {
    if(msg.device_id && msg.state){
        client.publish("zigbee2mqtt/"+msg.device_id+"/set",
           msg.state);
    }
})
```

Excerto de código A.29: Exemplo de escuta do servidor com publicação no tópico MQTT

```
socket.emit("EditDevice",{old_name: props.name, name: name});
```

Excerto de código A.30: Exemplo de aplicação II *socket.emit*

```
socket.on("EditDevice", (msg) => {
  if(msg.old_name && msg.name){
    Devices.findOneAndUpdate({friendly_name: msg.old_name},
    {friendly_name: msg.name},
    function(err) {
        if(err) console.log(err);
    })
  }
})
```

Excerto de código A.31: Exemplo de escuta II do servidor com edição na base de dados

```
<button onClick={() => socket.emit("DeleteDevice", {
    ieeeAddr: item.ieeeAddr
})}>Delete Device</button>
```

Excerto de código A.32: Exemplo de aplicação III *socket.emit*

```
socket.on("DeleteDevice", (msg) => {
    if(msg.ieeeAddr){
      client.publish("zigbee2mqtt/bridge/config/force_remove"
          ,
      msg.ieeeAddr);
    }
})
```

Excerto de código A.33: Exemplo de escuta III do servidor com publicação no tópico MQTT

```
divisions.map(division => (
    <li key={division._id}>
      <p>{division.name}</p>
      {devices.map(device =>
        device.division === item.name ? (
        <li key={device.ieeeAddr}>
                {device.friendly_name}
            </li>
          ): false)}
    </li>
))
```

Excerto de código A.34: Exemplo de aplicação IV do método *map*

```
const data = {
  labels: ["January", "February", "March", "April", "May", "
     June", "July"],
  datasets: [
    {
      label: "Example",
      steppedLine: true,
      fill: false,
      lineTension: 0.1,
      backgroundColor: "rgba(75,192,192,0.4)",
      borderColor: "rgba(75,192,192,1)",
      data: [65, 59, 80, 81, 56, 55, 40]
    }
  ]
};
```

Excerto de código A.35: Exemplo da estrutura do objecto *data* da biblioteca *react-chartjs-2*