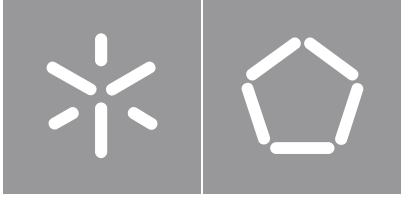
Universidade do Minho
Escola de Engenharia

Ruben Emanuel Fernandes Figueiredo

Desenvolvimento de um Sistema de Controlo e Monitorização Residencial Através de Tecnologias Associadas à Internet das Coisas

outubro de 2019



Universidade do Minho
Escola de Engenharia

Ruben Emanuel Fernandes Figueiredo

Desenvolvimento de um Sistema de Controlo e Monitorização Residencial Através de Tecnologias Associadas à Internet das Coisas

Dissertação de Mestrado
Mestrado Integrado em Engenharia
Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor José Augusto Afonso
Professor Doutor Vítor Monteiro

outubro de 2019

Agradecimentos

Gostaria de agradecer ao Professor Doutor José Augusto Afonso e ao Professor Doutor Vítor Monteiro por todo o acompanhamento ao longo do trabalho e pela disponibilidade demonstrada. O esclarecimento de dúvidas e o conhecimento partilhado por ambos foi fundamental para o sucesso do resultado alcançado.

Por último, gostaria de agradecer aos meus pais, à minha irmã e aos meus amigos por todo o apoio, carinho e conselhos durante o meu percurso escolar. Esta dissertação não seria concretizável sem a ajuda de todos eles.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

A presente dissertação visa o desenvolvimento de um sistema de controlo e monitorização residencial baseado numa plataforma IoT (Internet of Things). O sistema assenta numa arquitetura de redes sem fios híbrida que integra vários dispositivos Bluetooth Low Energy (BLE) e/ou Wi-Fi com diferentes funções, como nós sensores, nós atuadores e um *hub/gateway*. A comunicação entre entidades da plataforma é assegurada pelo protocolo MQTT (Message Queuing Telemetry Transport) através do modelo *publish/subscribe.*

A realização deste projeto assenta em vários aspetos essenciais, dos quais se destacam: especificação da arquitetura enquadrada com as necessidades e funcionalidades desejadas; seleção e programação de dispositivos para controlo e monitorização de equipamentos da *smart home;* implementação de base de dados local para o armazenamento de dados relevantes; implementação de *gateway* (BLE-Wi-Fi) de forma a permitir a comunicação entres dispositivos com diferentes protocolos; desenvolvimento de aplicação Android para controlar e monitorizar em tempo real os equipamentos através de uma interface gráfica.

Para a validação experimental foram realizados diversos testes para verificar o atraso e a fiabilidade das comunicações num ambiente habitacional. Foi possível verificar que não ocorreram perdas de pacotes e que o atraso máximo em todos os casos foi inferior a 1 segundo, permitindo que o sistema reaja rapidamente a alterações do valor de corrente, evitando que o disjuntor na habitação atue e possibilitando que o utilizador receba/envie dados de forma quase instantânea. A aplicação Android desenvolvida funcionou como pretendido, enviando e recebendo dados através da rede com o protocolo MQTT, e os sensores utilizados funcionaram com a precisão adequada. Um sistema de carregamento de veículos elétricos já existente foi integrado com sucesso no sistema e foi desenvolvido um protótipo de uma tomada elétrica inteligente que calcula os valores eficazes de tensão e corrente e controla o estado de operação do equipamento que a ela estiver ligado. O desenvolvimento do sistema teve em conta a utilização de software/firmware de utilização livre e hardware de baixo custo. Conforme validado com os resultados experimentais, os objetivos estabelecidos inicialmente foram cumpridos.

Palavras-chave: *smart home*; IoT, MQTT, Wi-Fi, Bluetooth Low Energy.

Abstract

This dissertation aims the development of a home control and monitoring system based on an IoT (Internet of Things) platform. The system is based on a hybrid wireless architecture that integrates multiple Bluetooth Low Energy (BLE) and/or Wi-Fi devices with different functions such as sensor nodes, actuator nodes and a hub/gateway. Communication between platform entities is ensured by the MQTT (Message Queuing Telemetry Transport) protocol through the publish/subscribe model.

For the realization of this project it was essential: to specify an architecture that met the needs and desired functionality; select and program devices to efficiently control and monitor smart home equipment; implement a local database for storing relevant data; implement a gateway (BLE-Wi-Fi) to allow communication between devices with different protocols; and develop an Android application to control and monitor equipment in real time through a graphical interface.

Experimental tests were performed to verify the delay and reliability of communications in a residential environment. No packet loss occurred and the maximum delay in all cases was less than 1 second, allowing the system to react quickly to changes in current value, preventing the circuit breaker in the home from tripping and allowing the user to receive/send data in a timely manner. The Android application worked as intended by sending and receiving data over the network using the MQTT protocol and the sensors worked with the proper accuracy. An existing electric vehicle charging system has been successfully integrated into the system and a prototype smart electrical socket has been developed to measure the current and voltage and control the operating state of the connected equipment (ON/OFF). The development of the system considered the use of free software/firmware and low-cost hardware. As validated with the experimental results, the initially established objectives were met, allowing even new approaches.

Keywords: smart home; IoT, MQTT, Wi-Fi, Bluetooth Low Energy.

Índice de conteúdos

Lista de figuras

Lista de tabelas

Lista de acrónimos e siglas

ADC	Analog-to-Digital Converter
AC	Alternating Current
API	Application Programming Interface
APT	Advanced Packaging Tool
BLE	Bluetooth Low Energy
BSS	Basic Service Set
DC	Direct Current
DCF	Distributed Coordination Function
DEX	Dalvik Executable
DNS	Domain Name System
DSP	Digital Signal Processor
DSSS	Direct Sequence Spread Spectrum
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPIO	General Purpose Input/Output
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IDE	Integrated Design Environment

IoT	Internet of Things
ISM	Industrial Scientific Medical
IP	Internet Protocol
JAR	Java ARchive
JSON	Java Script Object Notation
LAN	Local Area Network
LP-WAN	Low Power Wide Area Network
LTE	Long-Term Evolution
MAC	Media Access Control
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
NTC	Negative Temperature Coefficient
OFDM	Orthogonal Frequency Division Multiplexing
OLED	Organic Light-Emitting Diode
OTA	Over the Air
OS	Operating System
PCF	Point Coordination Function
PWM	Pulse Width Modulation
SIG	Special Interest Group
SMS	Short Message Service
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol

UI	User Interface
USB	Universal Serial Bus
UUID	Universally Unique Identifier
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

1. Introdução

Este capítulo começa por apresentar o enquadramento e as motivações associadas a sistemas de controlo e monitorização residencial baseados na Internet of Things. De seguida são referidos os principais objetivos delineados e, por último, é feita uma descrição da estrutura dos restantes capítulos desta dissertação.

1.1 Enquadramento e motivação

Uma *smart home* (casa inteligente) é, essencialmente, caracterizada pela integração de sistemas de controlo e monitorização numa casa, de modo a simplificar e melhorar a qualidade de vida, assim como garantir maior segurança, conforto e comodidade [1]. Além disso, pode contribuir também para o aumento da eficiência energética através do controlo, entre outros, da iluminação, da refrigeração e do aquecimento e da gestão das diversas fontes de energia disponíveis como a rede de distribuição, sistemas fotovoltaicos e bateria de um veículo elétrico [2].

Através do recente paradigma de comunicação denominado Internet of Things (IoT), torna-se possível que qualquer equipamento elétrico seja conectado à Internet. Um aspirador, uma torradeira ou uma simples lâmpada podem ser adaptados com o objetivo de se tornarem "inteligentes", de forma a serem monitorizados e/ou controlados à distância, a partir de sensores e atuadores, por um smartphone ou qualquer outro dispositivo de rede. Com isto, é possível efetuar a gestão de múltiplos dispositivos e objetos conectados, transformando os dados obtidos em informação útil tanto para o utilizador como para o sistema de gestão [3].

Nesta dissertação foi desenvolvido um sistema para uma *smart home* que, com base em tecnologias associadas à IoT, torna possível a monitorização ou o controlo de qualquer equipamento que se pretenda com recurso a uma aplicação num smartphone Android. Além disso a arquitetura implementada permite que o controlo dos equipamentos e a gestão de dados seja efetuada autonomamente, sem requerer a presença do utilizador e do próprio smartphone. As comunicações entre equipamentos/dispositivos são suportadas por redes locais sem fios (WLAN – Wireless Personal Area Network) como o Wi-Fi e redes pessoais sem fios (WPAN – Wireless Personal Area Network) como o Bluetooth Low Energy (BLE) e o armazenamento e gestão de dados são realizados numa base de dados local. Do ponto de vista dos

equipamentos da *smart home*, foram considerados um sistema de carregamento de baterias de um veículo elétrico, circuitos de iluminação e eletrodomésticos comuns. A aplicação Android disponibiliza uma interface gráfica que permite controlo do tipo ON/OFF, monitorização em tempo real (e.g., a tensão da rede elétrica e a corrente consumida) e consulta na base de dados a partir de qualquer local, via Internet.

Um dos principais propósitos no desenvolvimento deste sistema foi tornar acessível a qualquer pessoa, de forma fácil e sem custos elevados, a transformação da sua casa tradicional numa *smart home* flexível para gestão energética.

1.2 Objetivos

Esta dissertação consiste no desenvolvimento de um sistema baseado em tecnologias IoT que permite monitorizar e controlar equipamentos numa *smart home* através de um smartphone. A comunicação entre dispositivos/equipamentos é baseada, tanto em módulos Wi-Fi como BLE, sendo que os dados relevantes são automaticamente armazenados numa base de dados local para consulta e análise posterior. De forma sucinta, os objetivos são, portanto:

1. Estudo e especificação da arquitetura do sistema de acordo com as necessidades e funcionalidades desejadas (e.g., topologia e protocolos de comunicação);
2. Seleção e programação dos dispositivos para monitorização e controlo eficiente dos equipamentos da *smart home*;
3. Implementação de uma base de dados local para armazenamento de dados relevantes, permitindo a posterior consulta e análise;
4. Implementação de um *gateway* de forma a permitir a comunicação entre dispositivos com diferentes protocolos (Wi-Fi e BLE) de acordo com a arquitetura especificada;
5. Desenvolvimento de aplicação Android, que permite controlar e monitorizar os equipamentos através de uma interface gráfica;
6. Integração e teste dos componentes desenvolvidos.

1.3 Estrutura da dissertação

Esta dissertação encontra-se dividida em cinco capítulos de acordo com a seguinte ordem: Introdução, Estado da arte, Desenvolvimento do sistema, Testes e resultados e Conclusões e sugestões de trabalho futuro.

No segundo capítulo, "Estado da arte", são apresentadas as principais tecnologias, conceitos e fundamentos relevantes para o desenvolvimento do sistema. Neste capítulo são também referidos os principais trabalhos relacionados.

No terceiro capítulo, "Desenvolvimento do sistema", é apresentada a arquitetura implementada, o hardware utilizado e o firmware/software instalado e desenvolvido.

No quarto capítulo, "Testes e resultados", são descritos todos os testes realizados e os resultados provenientes, tais como atrasos de comunicação.

No quinto e último capítulo, "Conclusões e sugestões de trabalho futuro", é feita uma análise e discussão daquilo que foi realizado e são apresentadas sugestões para trabalhos futuros.

2.Estado da arte

Neste capítulo são apresentados conceitos relevantes no contexto da dissertação, nomeadamente a arquitetura IoT e as principais tecnologias e protocolos disponíveis para uma arquitetura IoT, com ênfase nas que se enquadram melhor neste projeto. Por último, são apresentados os principais trabalhos relacionados.

2.1 Arquitetura IoT

Devido à diversidade de tarefas envolvidas no funcionamento de uma plataforma IoT existem várias propostas de arquitetura [4]–[6], não existindo, portanto, um modelo de referência consensual [7]. Tendo em conta o contexto desta dissertação, a arquitetura IoT que melhor se enquadra é constituída por três camadas principais, tal como apresentado na Figura 1: camada de aplicação, camada de rede e camada de objetos.

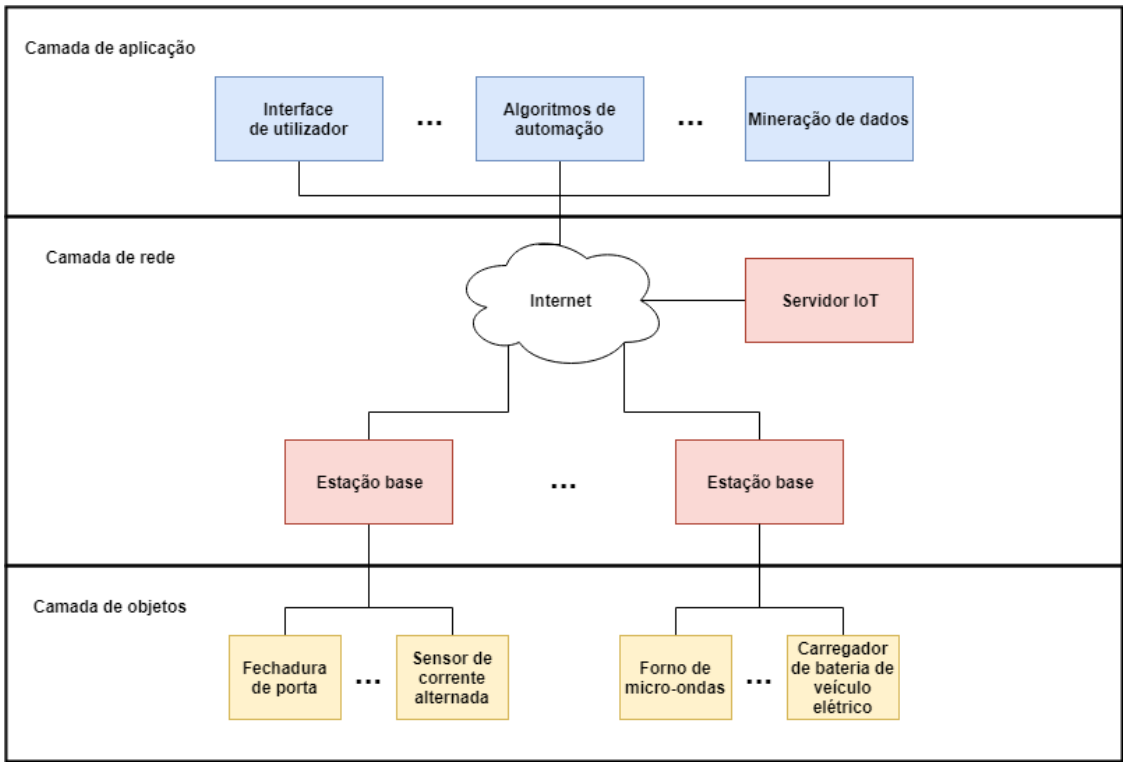


Figura 1. Arquitetura IoT baseada num modelo de três camadas.

A camada de objetos (*objects layer*) é responsável pela interação com o mundo físico. Os componentes principais desta camada são sensores e atuadores que juntamente com outros

componentes de hardware (e.g., microcontroladores e *transceivers*) permitem converter um objeto físico tradicional num objeto inteligente (*smart object*). Um objeto inteligente é um objeto que consegue receber, armazenar, processar dados e interagir com pessoas e/ou outros objetos.

A camada de rede (*network layer*) é responsável por toda a infraestrutura necessária para troca de dados entre entidades da plataforma IoT através de tecnologias de redes sem fios (e.g., BLE e Wi-Fi). Além disso, a camada de rede realiza outras tarefas como armazenamento de dados, entre outras.

A camada de aplicação (*application layer*) fornece serviços específicos aos utilizadores, através da análise dos dados recolhidos na camada objetos. É nesta camada que é implementada a automação de processos através de algoritmos, a interface de utilizador e análises complexas de dados.

2.2 Tecnologias de comunicação utilizadas na IoT

As tecnologias de comunicação da IoT conectam objetos heterogêneos para fornecer serviços inteligentes específicos. Os principais critérios relevantes na escolha de uma tecnologia de comunicação sem fios consistem no consumo energético, alcance e taxa de transmissão de dados. Através desses critérios é possível categorizar essas tecnologias em três tipos de redes sem fios:

- **WPAN** (Wireless Personal Area Network): baixo consumo de energia e de carácter pessoal, ou seja, curto alcance, e.g., IEEE 802.15.4/ZigBee [8] e Bluetooth Low Energy (BLE) [9];
- **LP-WAN** (Low Power Wide Area Network): longo alcance, baixo consumo e baixa taxa de transmissão, e.g., LoRa [10] e Sigfox [11];
- **WLAN** (Wireless Local Area Network): alta taxa de transmissão, alcance e consumo energético superior a redes WPAN, e.g., IEEE 802.11/Wi-Fi [12].

De seguida, nas secções 2.2.1 e 2.2.2 são abordadas, com mais detalhe, as tecnologias com relevância no contexto desta dissertação.

2.2.1 IEEE 802.11/Wi-Fi

O IEEE 802.11 utiliza ondas de rádio nas bandas de frequência de 2,4 GHz e 5 GHz que pertencem à banda ISM (Industrial Scientific Medical) para troca de dados. Estas bandas não requerem licença para a instalação e/ou operação dos equipamentos, tornando-se numa opção bastante atrativa. O alcance

típico é de 100 m, mas pode chegar próximo dos 250 m caso a rede seja do padrão IEEE 802.11n. A Tabela 1 contém todos os padrões existentes com as suas principais características.

Tabela 1. Padrões IEEE 802.11.

Padrão	Data de Lançamento	Banda de frequências	Modulação	Distância máxima indoor/ outdoor (m)	Taxa de transmissão máxima
802.11b	1999	2.4 GHz	DSSS	35/140	11 Mbps
802.11a	1999	5 GHz	OFDM	35/120	54 Mbps
802.11g	2003	2.4 GHz	DSSS/OFDM	38/140	54 Mbps
802.11n	2009	2.4 GHz 5 GHz	OFDM	70/250	600 Mbps
802.11ad	2012	60 GHz	SC OFDM	1/10	6.76 Gbps
802.11ac	2013	5 GHz	OFDM		6.1 Gbps

O IEEE 802.11 é baseado numa arquitetura celular em que cada célula é chamada de conjunto de serviços básicos (BSS - Basic Service Set). Um BSS é um conjunto de estações Wi-Fi móveis ou fixas e um ponto de acesso. O acesso ao meio de transmissão é controlado por meio de um conjunto de regras chamadas de função de coordenação. Existe uma função de coordenação distribuída (DCF - Distributed Coordination Function) que deve ser implementada por todas as estações e uma função de coordenação pontual (PCF - Point Coordination Function), sendo esta última opcional [13].

2.2.2 BLE (Bluetooth Low Energy)

O BLE (Bluetooth Low Energy) foi desenvolvido pelo Bluetooth SIG (Special Interest Group) e possuí um protocolo de comunicação sem fios com baixa taxa de transmissão de dados, curto alcance e baixo consumo. O BLE é estruturado numa pilha de camadas de forma a transferir eficientemente pequenas quantidades de dados com pouca energia, o que o torna numa opção interessante com dispositivos alimentados por bateria.

A banda de frequências de operação é a banda ISM de 2,4 GHz, e o BLE define 40 canais de radiofrequência. Desses canais, três são usados para descobrir dispositivos, estabelecer conexões e transmissões *broadcast*, e os restantes 37 para troca de dados em comunicação bidirecional [14].

Para estabelecer uma ligação BLE entre dois dispositivos é necessário compreender que existem dois tipos de dispositivos definidos pela camada GAP (Generic Access Profile): periférico e central. O dispositivo periférico é aquele que anuncia a sua presença e aceita a conexão do dipositivo central,

enquanto que o dispositivo central procura os anúncios dos dispositivos periféricos e estabelece a comunicação com eles, como demonstrado na Figura 2. Depois de o dispositivo central estabelecer a conexão com o periférico, pode dizer-se que estão conectados por uma ligação BLE. O dispositivo central desempenha o papel de mestre (master) e os periféricos funcionam como escravos (slaves). O intervalo de tempo durante uma troca de mensagens entre o mestre o escravo denomina-se de evento de conexão, e o intervalo de tempo entre eventos de conexão para o mesmo escravo denomina-se intervalo de conexão [15].

Figura 2. Dispositivo central e periférico da camada GAP.

No âmbito da camada GATT (Generic Attribute Profile), o protocolo BLE define duas funções de perfil baseadas na origem e destino dos dados: servidor GATT e cliente GATT (Figura 3). O servidor é o dispositivo que contém os dados e, quando configurado pelo cliente, envia os dados para o cliente ou modifica o seu estado local. O cliente, por sua vez, é o dipositivo que configura o estado do servidor ou recebe dados do servidor.

Figura 3. Servidor e cliente da camada GATT.

Resumindo, depois de se estabelecer uma ligação BLE, o cliente procura todos os dados e o estado atual do servidor. Uma vez encontrados, o cliente consegue configurar e/ou ler/escrever dados ou definir o estado do servidor.

Um servidor usa atributos, características e serviços para representar e abstrair dados num dispositivo BLE. Um atributo é o mais importante "recipiente" de dados da camada GATT, que representa um pedaço de informação. O atributo é composto por: *attribute handle*, que é usado para endereçar o atributo; *attribute type*, que é um identificador universal único de 16 bits atribuído pelo Bluetooth SIG (Special Interest Group) e que especifica os dados contidos; *attribute value*, que contém os dados atuais e *attribute permission*, que especifica permissões de escrita/leitura e os requerimentos de segurança do atributo.

O servidor consiste em vários atributos que estão guardados numa base de dados. O cliente executa operações de leitura/escrita nos atributos guardados na base de dados do servidor usando o *attribute handle*. O utilizador basicamente só precisa saber o *attribute handle* no qual pretende executar uma operação de leitura/escrita.

Uma característica é composta por vários atributos que, quando combinados, definem uma informação do sistema ou um dado significativo, e um serviço é composto por uma ou mais características (Figura 4) relacionadas que definem uma função particular ou uma função do dispositivo [16].

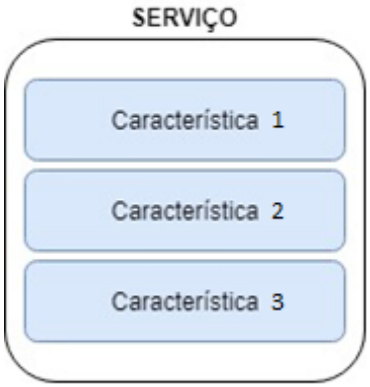


Figura 4. Serviço constituído por várias características.

2.3 Protocolo MQTT

O MQTT (Message Queuing Telemetry Transport) é um protocolo de transporte de mensagens entre cliente/servidor baseado no modelo *publish/subscribe*, tal como representado na Figura 5. É um protocolo *open source*, simples, leve, com garantia de entrega de mensagens, que assenta sobre os protocolos de rede TCP (Transmission Control Protocol) / IP (Internet Protocol), e que manipula a troca de mensagens, mesmo se os endereços IP dos dipositivos na rede mudarem ao longo tempo. É também adequado para dispositivos com recursos limitados que usam ligações não fiáveis ou com pouca largura de banda.

Este protocolo consiste essencialmente em três componentes: subscritor (*subscriber*), publicador (*publisher*) e *broker*. Um dispositivo interessado em receber informação regista-se como subscritor em determinados tópicos do seu interesse, sendo informado pelo *broker* quando publicarem informação sobre estes tópicos. Um publicador atua como um gerador de informação relevante sobre determinados tópicos, transmitindo-a para as entidades interessadas (subscritores) através do *broker* (Figura 6). Além

disso, o *broker* garante a segurança através da verificação da autorização dos publicadores e subscritores [17].

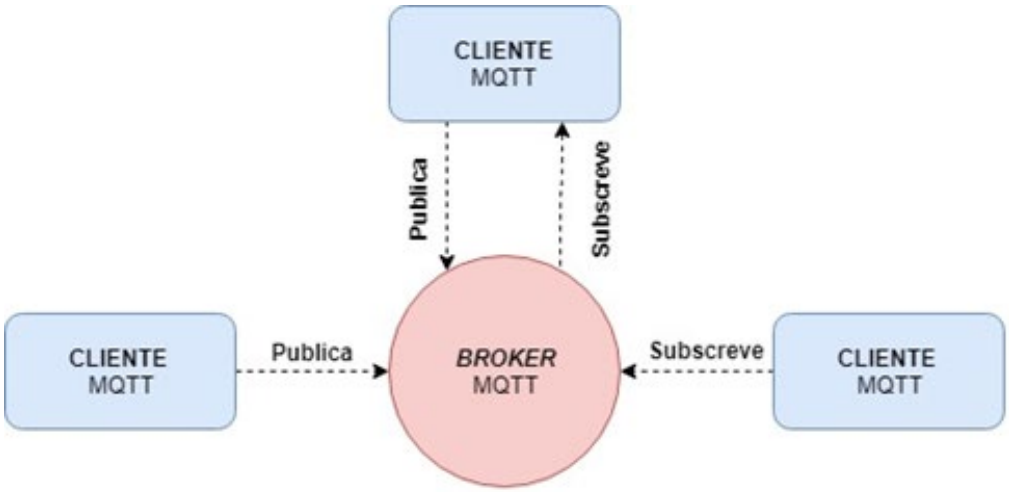


Figura 5. Modelo *publish/subscribe* do protocolo MQTT.

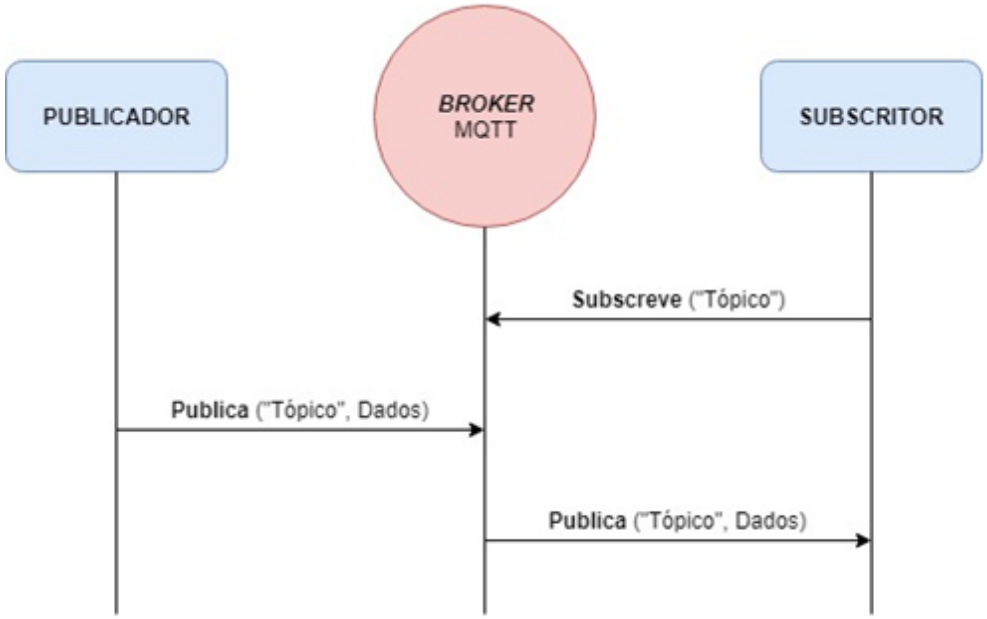


Figura 6. Modelo *publish/ subscribe* do protocolo MQTT.

O MQTT fornece 3 níveis de qualidade de serviço (QoS – Quality of Service) relativamente à entrega de mensagens [18], os quais são descritos de seguida.

- **QoS 0:** A mensagem com os dados (PUBLISH) é entregue no máximo uma vez e o publicador não recebe nenhuma confirmação de entrega (Figura 7). A mensagem não é armazenada e pode ser perdida se o destinatário estiver desconectado ou se o *broker* falhar. É o método mais rápido também conhecido por "*fire and forget*".

- **QoS 1:** A mensagem com os dados (PUBLISH) é sempre entregue pelo menos uma vez. Quando uma mensagem de dados publicada é recebida no destinatário pretendido é transmitida uma mensagem de confirmação (PUBACK) endereçada ao remetente (Figura 8). Caso o remetente não receba uma confirmação, a mensagem é enviada novamente até que uma confirmação seja recebida o que poderá resultar na repetição de mensagens.
- **QoS 2:** A mensagem com os dados (PUBLISH) é sempre entregue apenas uma vez. É o modo de transferência mais seguro, mas mais lento. São necessários pelo menos dois pares de transmissões entre o remetente e o destinatário antes que a mensagem seja excluída do remetente (Figura 9). No primeiro par de transmissões, o remetente transmite a mensagem (PUBLISH) e obtém reconhecimento do destinatário de que a mensagem foi armazenada (PUBREC). Se o remetente não receber uma confirmação, a mensagem será enviada novamente até que uma confirmação seja recebida. No segundo par de transmissões, o remetente diz ao destinatário que pode concluir o processamento da mensagem através do envio da mensagem (PUBREL). Se o remetente não receber uma confirmação por parte do remetente que recebeu a mensagem PUBREL, a mensagem PUBREL será enviada novamente até que uma confirmação seja recebida. O remetente exclui a mensagem que salvou ao receber a confirmação (PUBCOMP).

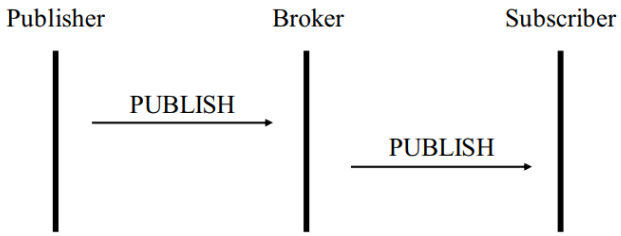


Figura 7. Diagrama do protocolo MQTT com QoS 0 [19].

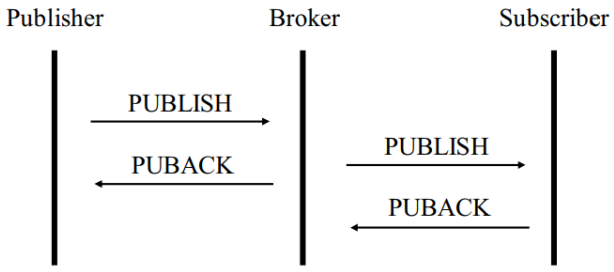


Figura 8. Diagrama do protocolo MQTT com QoS 1 [19].

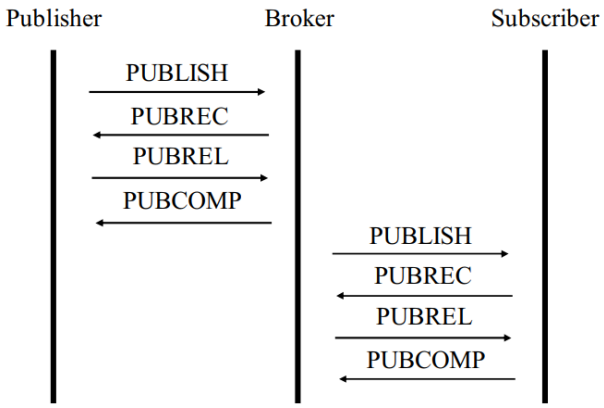


Figura 9. Diagrama do protocolo MQTT com QoS 2 [19].

No MQTT, cada mensagem tem um tópico (*string* UTF-8) que o *broker* usa para filtrar mensagens para cada cliente conectado. O tópico consiste em um ou mais níveis de tópico. Cada nível de tópico é separado por uma barra (separador de nível de tópico). Na Figura 10 está representado um exemplo da estrutura de um tópico MQTT.

Figura 10. Exemplo da estrutura de um tópico MQTT.

O cliente não precisa criar o tópico desejado antes de publicar ou subscrever. O *broker* aceita cada tópico válido sem nenhuma inicialização prévia.

2.4 Sistema operativo Android

O Android é um sistema operativo baseado em Linux que foi lançado inicialmente em 2005 e é atualmente desenvolvido pela empresa Google. Este sistema está disponível para, por exemplo, smartphones, televisões, tablets, carros e relógios. A última versão disponível é o Android 9.0 também conhecido por Android Pie. O IDE oficial e utilizado para desenvolvimento de aplicações chama-se Android Studio [20], que, no momento da escrita desta dissertação, se encontra na versão 3.5. De seguida são apresentados e abordados os componentes essenciais no funcionamento de uma aplicação Android.

2.4.1 Arquitetura da plataforma

A arquitetura da plataforma Android consiste em seis componentes principais organizados em cinco camadas, como se observa na Figura 11.

- **System Apps:** oferece um conjunto de aplicações principais ao utilizador como por exemplo e-mail, mensagens SMS (Short Message Service), calendários, navegação na Internet, contatos, entre outros.
- **Java API Framework**: oferece um conjunto de recursos através de APIs (Application Programming Interface) escritas na linguagem Java. Essas APIs formam os blocos de construção para criar aplicações, simplificando a reutilização de componentes, serviços centrais e modulares do sistema.
- **Native C/C++ Libraries**: fornece as bibliotecas de suporte essenciais às aplicações.
- **Android Runtime (ART):** executa várias máquinas virtuais em dispositivos de pouca memória, executando arquivos DEX (Dalvik Executable) para ocupar pouco espaço.
- **Hardware Abstraction Layer (HAL):** fornece interfaces padrão que expõem os recursos de hardware do dispositivo à estrutura da API Java de nível superior.
- **Linux Kernel:** A base da plataforma Android é o *kernel* do Linux, permitindo que sejam aproveitados os seus principais recursos de segurança e que os fabricantes de dispositivos desenvolvam *drivers* de hardware para um *kernel* bem conhecido.

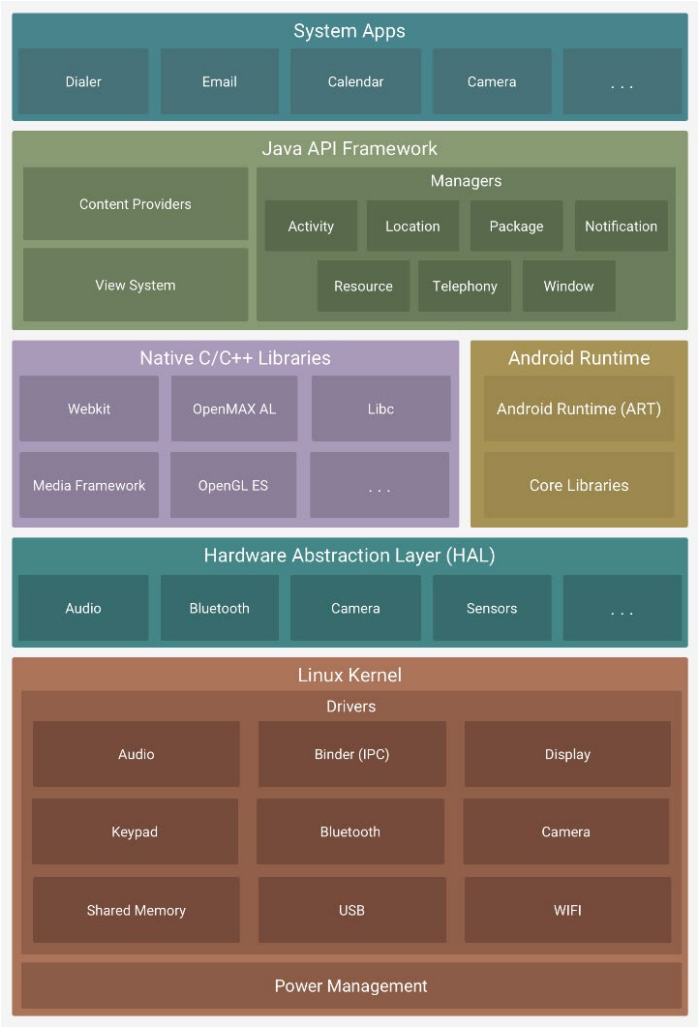


Figura 11. Arquitetura da plataforma Android [21].

2.4.2 Classe Activity

A classe Activity é um componente fundamental numa aplicação Android, e a forma como as Activities são lançadas e dispostas é essencial.

O sistema Android inicia o código numa instância Activity invocando métodos (*lifecycle callbacks*) que correspondem a estágios específicos do seu ciclo de vida (Figura 12), contrariamente a certos paradigmas de programação nos quais as aplicações são iniciadas com o método main(). A classe Activity fornece, portanto, um conjunto principal de seis métodos para navegar pelas transições entre os estágios do ciclo de vida da Activity. De forma sucinta os métodos são:

- **onCreate():** é executada toda a lógica básica de inicialização da aplicação, que deve acontecer apenas uma vez durante toda a vida útil da Activity. É neste método que se define o *layout* da interface gráfica e se atribuem as referências das Views.

- **onStart():** torna a Activity visível para o utilizador, à medida que a aplicação se prepara para a atividade entrar no primeiro plano.
- **onResume():** este é o estado em que o aplicativo interage com o utilizador. A aplicação permanece nesse estado até que algo aconteça para desviar o foco da aplicação.
- **onPause():** o sistema chama este método como a primeira indicação de que o utilizador está a sair da Activity (nem sempre significa que a Activity vai ser destruída).
- **onStop():** executado quando a Activity não está mais visível para o utilizador.
- **onDestroy():** é chamado antes que a atividade seja destruída. É aqui que os componentes do ciclo de vida podem limpar tudo o que precisarem antes que a Activity seja destruída.

Uma Activity fornece o *layout* no qual é desenhada a interface do utilizador, sendo que cada Activity corresponde a um ecrã da aplicação. Dado que a maioria dos aplicativos contém vários ecrãs também existem várias Activities numa aplicação. A primeira Activity é chamada de MainActivity pois é o primeiro ecrã a ser criado e exibido ao utilizador.

Geralmente, cada Activity tem um propósito ou uma função diferente. Por exemplo, uma pode implementar um ecrã de *login* enquanto outra implementa um ecrã para aceder às mensagens de texto. Cada Activity pode iniciar outra Activity para realizar ações diferentes.

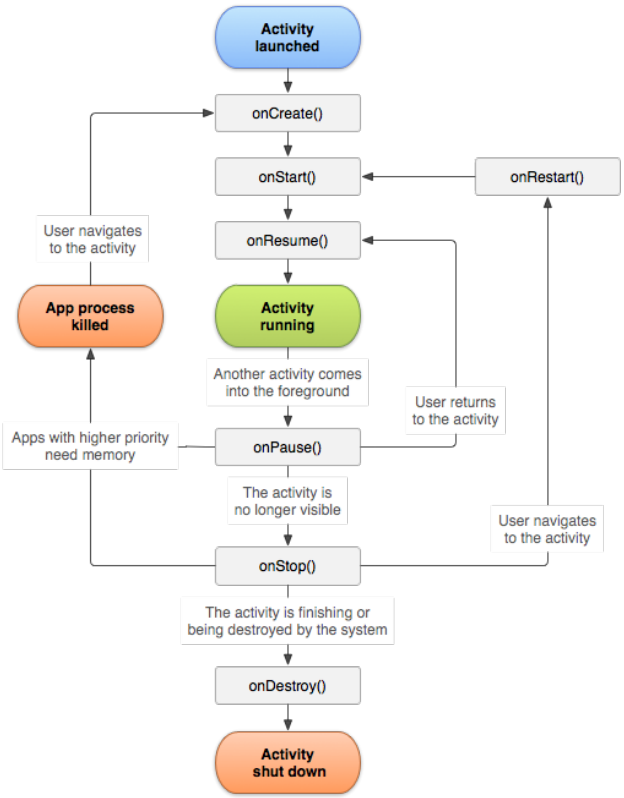


Figura 12. Ciclo de vida de uma Activity [22].

Para usar Activities é necessário registá-las no manifesto da aplicação e gerir os seus ciclos de vida de maneira adequada.

2.4.3 Classe Fragment

Um Fragment é usado para separar diferentes partes da interface gráfica como uma subActivity e pode ser utilizada em diferentes Activities. Possui o seu próprio *layout* e ciclo de vida e recebe seus próprios eventos de entrada. Deve sempre ser hospedado numa Activity e o seu ciclo de vida é diretamente afetado pelo ciclo de vida da Activity que o hospedou. Na Figura 13 é possível verificar que o ciclo de vida do Fragment contém métodos de retorno de chamada semelhantes a uma Activity, como onCreate (), onStart (), onPause () e onStop ().

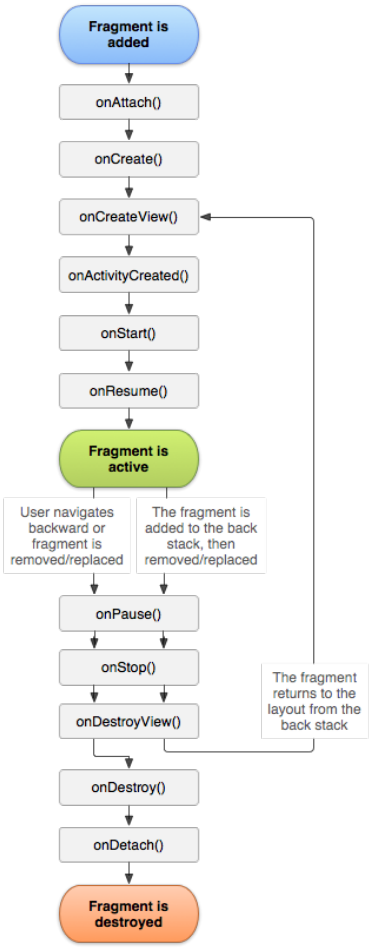


Figura 13. Ciclo de vida de um Fragment [23].

Os eventos que não existem numa Activity são:

- **onAttach**: quando o Fragment é adicionado na Activity.

- **onCreateView:** quando desenha a sua interface de utilizador pela primeira vez.
- **onDestroyView:** quando a View do Fragment é destruída.
- **onDetach:** quando o Fragment é removido da Activity.

2.4.4 Classe Service

Um Service é um componente que permite realizar operações longas em segundo plano. Um Service pode ser iniciado por outro componente e continuará em execução mesmo que o utilizador alterne para outra aplicação. Existem três tipos diferentes de Services:

- **Foreground:** serviço que executa alguma operação que é percetível para o utilizador através de uma notificação. Continua em execução mesmo que o utilizador não esteja a interagir com a aplicação.
- **Background:** serviço que executa uma operação que não é notada diretamente pelo usuário.
- **Bound:** serviço que oferece uma interface cliente-servidor, em que o serviço atua como servidor e os componentes da aplicação como clientes. É possível vincular vários componentes ao serviço de uma só vez, mas quando todos eles são desvinculados, o serviço é destruído.

Assim como uma Activity, um Service tem *callback methods* de ciclo de vida que permitem monitorizar alterações no estado do serviço e executar o que se pretende no momento adequado. A Figura 14 apresenta cada um dos métodos do ciclo de vida para Services do tipo Foreground e Background (lado esquerdo), assim como para Services do tipo Bound (lado direito).

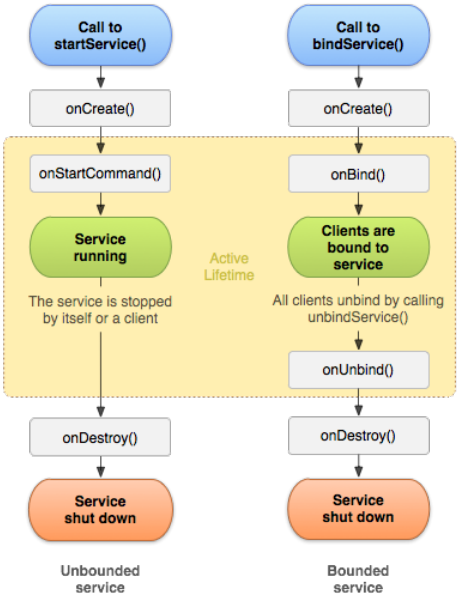


Figura 14. Ciclo de vida de um Service [24].

2.5 Trabalho relacionado

Em [25] foi implementado um sistema de automação residencial inteligente para ajudar a reduzir custos e economizar energia (Figura 15). O sistema possui um microcontrolador com módulo Wi-Fi (ESP8266) e um módulo de relés. Para comunicação foi escolhido o protocolo MQTT e uma plataforma baseada em *cloud* chamada Thinger.io para conectar e gerir os dispositivos inteligentes na rede. É possível aceder a esta plataforma com uma aplicação Android fornecida pelo Thinger.io através de um método de autenticação. Nesta implementação verifica-se que o protocolo MQTT e a placa NodeMCU são uma boa solução para um sistema de automação residencial, no entanto o uso de uma plataforma baseada em *cloud,* neste caso a Thinger.io, requer conexão à Internet e normalmente acarreta custos extras e limitações impostas pela própria plataforma (e.g., número de dispositivos que se podem conectar). Além disso a dependência de Internet pode colocar em causa o sistema caso o acesso a esta seja condicionado. Nesse sentido o sistema desenvolvido nesta dissertação tem uma arquitetura que não requer conexão à Internet para o seu funcionamento.

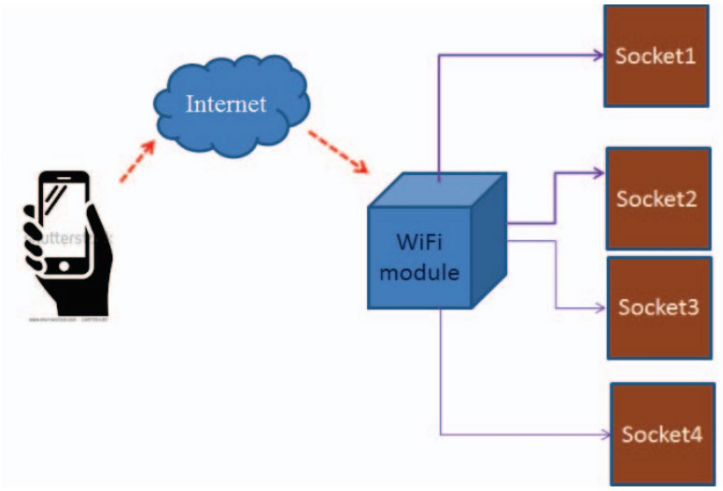


Figura 15. Diagrama de um sistema de automação residencial usando IoT [25].

Em [26] foi feita uma análise geral do protocolo MQTT e implementado um cliente MQTT numa placa NodeMCU ESP8266. Os sensores e atuadores são conectados ao ESP8266 e um *broker* MQTT baseado em Mosquitto é instalado num computador. O autor conclui que este tipo de implementação oferece um sistema de automação residencial inteligente que permite otimizar a eficiência energética. O sistema implementado encontra-se representado na Figura 16. Em relação à utilização de um computador apenas para implementar o *broker* MQTT acaba por ser um desperdício em termos de recursos, uma vez que poderia ter utilizado uma Raspberry Pi como foi feito nesta dissertação.

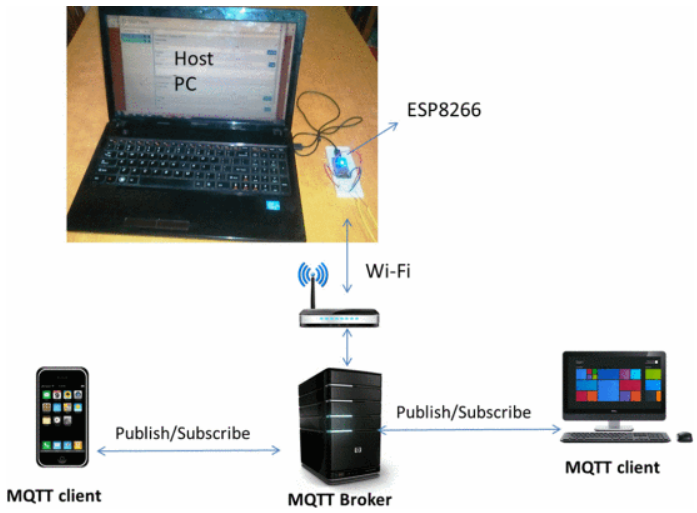


Figura 16. Sistema de automação residencial baseado em MQTT [26].

Em [27] foi projetado um sistema básico de automação residencial para controlar e monitorizar vários dispositivos através de um servidor Web (Figura 17). Este modelo utiliza uma interface simples e amigável para o acesso da Raspberry Pi através de comandos enviados através da página Web em que o algoritmo para o mesmo foi desenvolvido em ambiente Python. São utilizados módulos NRF24L01, possibilitando comunicação sem fios entre a Raspberry Pi e módulos Arduino UNO. Cada Arduino UNO está diretamente conectado ao dispositivo que pretenda controlar, sendo a Raspberry Pi o controlador. Embora a arquitetura do sistema possa usar HTTP convencional ou o protocolo MQTT, o autor conclui após análise de performance que o MQTT consome menos energia e consegue enviar 100 vezes mais mensagens utilizando a mesma energia que o HTTP. Na solução proposta a placa NodeMCU utilizada nesta dissertação teria sido uma opção mais barata e versátil que o Arduino UNO com NRF24L01.

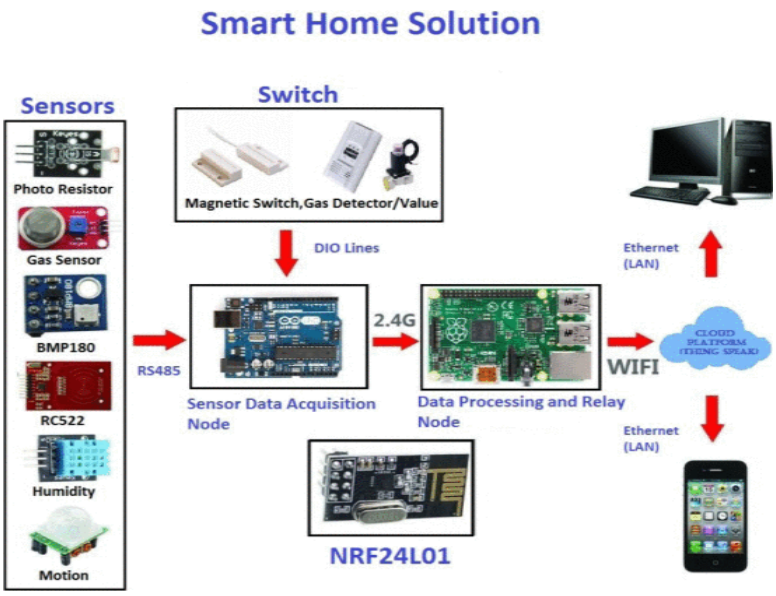


Figura 17. Sistema de automação residencial [27].

O sistema proposto em [28], apresentado na Figura 18, permite utilizar qualquer dispositivo (e.g., telemóveis e computadores) que tenha um navegador Web que suporte o protocolo HTTP (Hypertext Transfer Protocol) para controlar e monitorizar equipamentos domésticos através de uma interface gráfica. Essa interface gráfica foi desenvolvida em HTML e JavaScript e hospedada num servidor Web na Raspberry Pi. Pode ser acedida localmente através da rede Wi-Fi e Ethernet ou remotamente via Internet. Caso o utilizador não possua Internet e deseje iniciar comandos, existe a possibilidade de enviar SMS pela rede GSM (Global System for Mobile Communications).

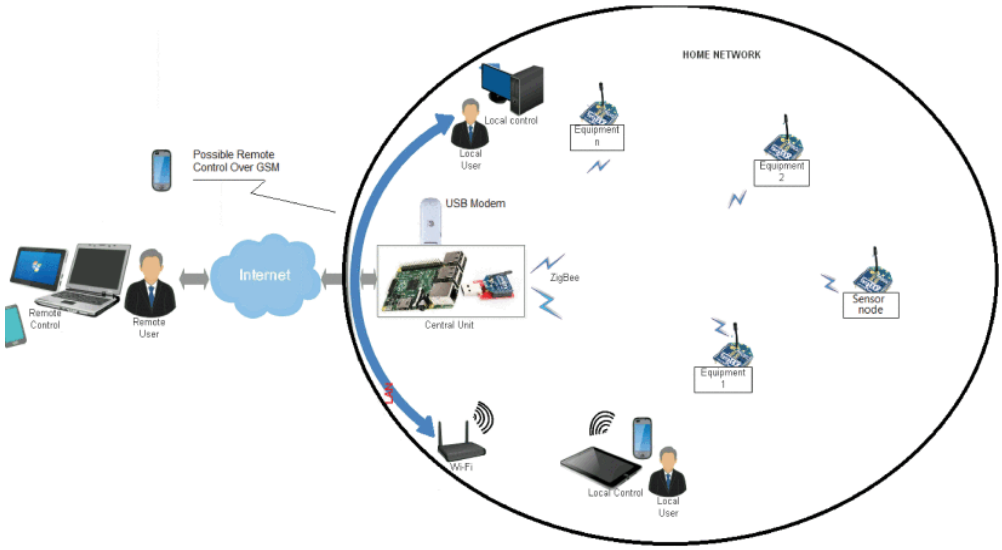


Figura 18. Sistema de controlo doméstico inteligente [28].

Para a transferência de dados na rede residencial é utilizado o protocolo ZigBee [8], devido ao seu baixo custo e baixo consumo de energia. A unidade central é composta por uma placa Raspberry Pi B+, um coordenador XBee e um modem GSM. O módulo coordenador XBee e o modem GSM são conectados à Raspberry Pi através de uma porta USB (Universal Serial Bus) e trocam dados através de um protocolo série. Os *routers* XBee são montados em placas inteligentes desenvolvidas, que são equipadas com um circuito de alimentação, circuito de comando e uma placa de sensores. Existem duas maneiras possíveis de monitorizar e controlar: através da Internet baseada em serviços da Web REST ou através de uma solução alternativa GSM. Caso se pretenda, a arquitetura do sistema utilizado nesta dissertação possui flexibilidade para permitir a integração das tecnologias ZigBee e GSM [29], utilizadas neste sistema.

Em [30] foi projetado, implementado e testado o sistema de monitorização de energia residencial representado na Figura 19. O sistema é composto por dispositivos de monitorização, um *gateway* e uma interface gráfica de utilizador (GUI). Para comunicação foi utilizada uma solução híbrida composta por Bluetooth (BLE) e Wi-Fi e um *gateway* para permitir a troca de dados entre os diferentes protocolos assim

como nesta dissertação. Os dispositivos que fazem a monitorização da energia contêm sensores de tensão e corrente, condicionadores de sinal e uma unidade de comunicação sem fios (BLE) e de processamento. É importante referir que neste sistema os nós sensores/atuadores apenas são compostos por módulos BLE enquanto que nesta dissertação estes podem ser compostos por módulos BLE ou Wi-Fi.

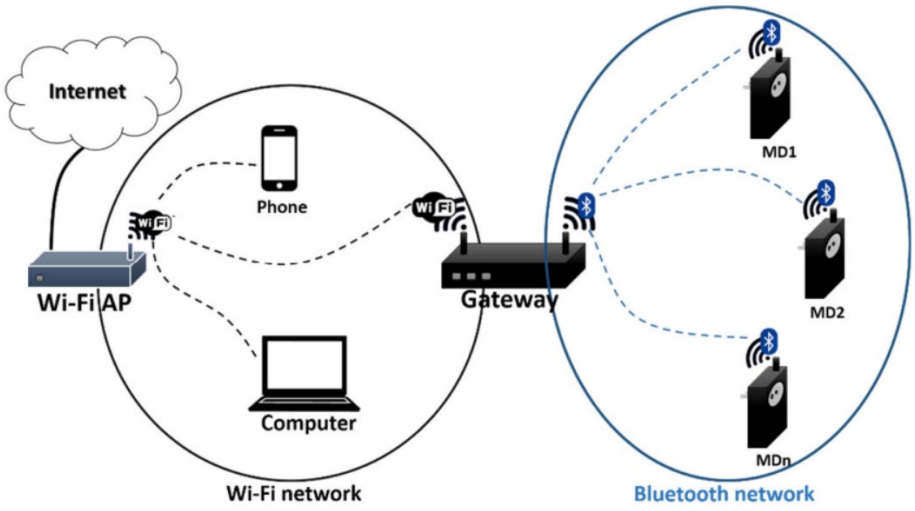


Figura 19. Sistema de monitorização de energia residencial [30].

Em [31] foi desenvolvido e testado um sistema sem fios que permite a monitorização remota de eventos de consumo de energia e qualidade de energia de vários aparelhos elétricos. O sistema proposto está estruturado em quatro partes: o hardware do nó sensor; o hardware da estação base; o software incorporado nos nós da rede; e o software para PC (UI de monitorização). O protótipo desenvolvido está representado na Figura 20, e no que concerne ao nó sensor, é de certa forma semelhante ao protótipo descrito na secção 4.3.2, utilizando no entanto ZigBee para comunicar os dados obtidos e calculados.

Figura 20. Protótipo do sistema de monitorização de [31].

3. Desenvolvimento do sistema

O presente capítulo apresenta a arquitetura implementada, o hardware utilizado e o software/firmware instalado e implementado no desenvolvimento do sistema.

3.1 Arquitetura implementada

O sistema desenvolvido tem uma arquitetura de redes sem fios híbrida (diferentes topologias de rede), tal como apresentado na Figura 21. Como se pode verificar, o sistema é constituído essencialmente por objetos inteligentes (*smart objects*) integrando dispositivos Bluetooth Low Energy (BLE) e Wi-Fi como nós de sensores/atuadores, um *hub/gateway*, um *router* Internet e clientes IoT.

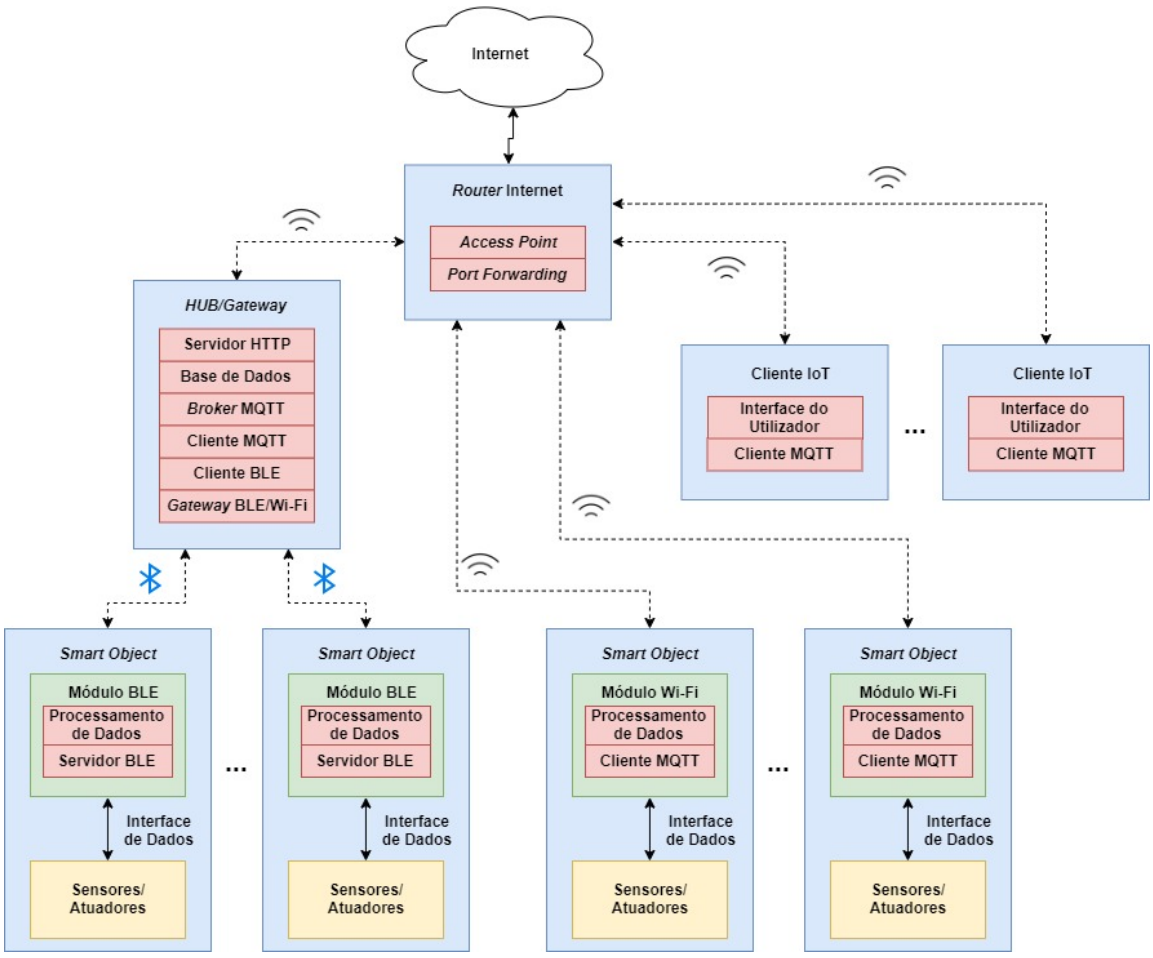


Figura 21. Arquitetura do sistema implementado.

Um dispositivo só se pode conectar diretamente a uma plataforma IoT quando estão implementados protocolos da pilha protocolar da Internet, por exemplo, IP (Internet Protocol) e TCP (Transmission Control Protocol). Estes protocolos estão implementados em dispositivos Wi-Fi, mas não em dispositivos BLE. O BLE implementa o seu próprio protocolo, otimizado para poupança de energia. Sendo assim, o dispositivo *hub/gateway* é responsável por converter os pacotes trocados entre as duas partes, dos protocolos da rede BLE para os protocolos da Internet e vice-versa, permitindo a integração de dispositivos BLE na plataforma IoT.

A comunicação entre entidades da plataforma IoT é assegurada pelo protocolo MQTT através do modelo *publish/subscribe*. Os publicadores (e.g., nós sensores) geram dados que os subscritores (e.g., nós atuadores) pretendem receber, e o *broker* faz toda a gestão da troca de dados entre eles. Os clientes MQTT (publicadores e subscritores) são implementados apenas nos dispositivos Wi-Fi do sistema, pois o protocolo MQTT assenta sobre os protocolos de rede TCP/IP, de modo que surge a necessidade do *gateway* Wi-Fi/BLE para lidar com os dados provenientes dos objetos BLE.

Para o armazenamento de dados, foi implementada uma base de dados no *hub/gateway* para que todos os dados relevantes e de interesse que passem no *broker* sejam adquiridos através de um cliente MQTT e armazenados localmente. Adicionalmente, para que fosse possível administrar a base de dados através de uma aplicação Web (phpMyAdmin) e realizar consultas com a aplicação Android desenvolvida, foi instalado um servidor HTTP e implementada uma Web API.

O *router* Internet permite conexão à Internet e que dispositivos de uma rede exterior se conectem a um dispositivo da rede local através do mecanismo *port forwarding.* O router dispõe ainda da funcionalidade de AP (*access point*), possibilitando a comunicação sem fios dentro da rede Wi-Fi.

Os clientes IoT são os dispositivos (e.g., smartphones, tablets ou computadores) que, por intermédio de um cliente MQTT, permitem monitorizar e controlar os serviços da *smart home* através de uma interface gráfica. O acesso à *smart home*, por parte dos clientes IoT, pode ser realizado dentro de casa através da rede Wi-Fi ou fora de casa através da Internet.

3.2 Hardware utilizado

A arquitetura do sistema, em termos de hardware, encontra-se representada na Figura 22, para uma melhor compreensão dos principais componentes utilizados. Cada componente foi escolhido tendo em

conta as suas características, que são descritas nas próximas secções, e as necessidades da arquitetura implementada. A partir disso chegou-se às seguintes seleções:

- *Hub/gateway*: Raspberry Pi 4 B.
- *Router* **Internet**: Huawey HG8247Q.
- **Módulo BLE**: FireBeetle ESP32.
- **Módulo Wi-Fi**: NodeMCU ESP8266 ESP-12E.
- **Cliente IoT**: Smartphone Android.
- **Sensores**/**Atuadores:** sensor DTH11, sensor de tensão ZMPT101B, sensor de corrente Gravity 20 A, módulo relé e TMDSDOCK28335 como interface com o sistema de carregamento do veículo.

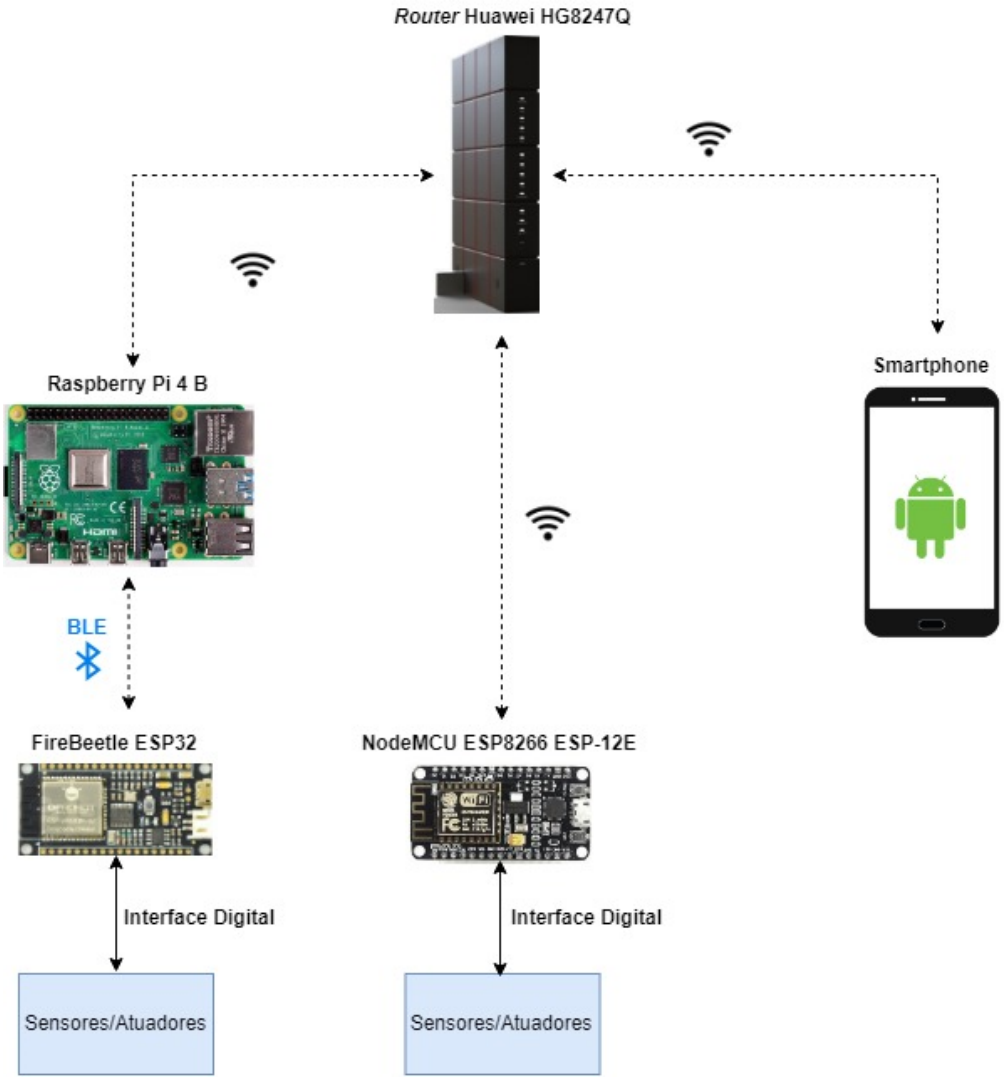


Figura 22. Visão geral dos componentes utilizados na arquitetura do sistema.

3.2.1 NodeMCU ESP8266 ESP-12E

O módulo NodeMCU ESP-12E (Figura 23) é uma placa de desenvolvimento de baixo custo com circuito integrado ESP8266 da Espressif que permite acesso a redes Wi-Fi. A programação pode ser realizada pela sua SDK (Software Development Kit) nativa, usando LUA, ou com o Arduino IDE. Possui 10 pinos GPIO (General Purpose Input/Output), suportando funções como PWM (Pulse Width Modulation), I2C (Inter-Integrated Circuit) e 1-wire, antena embutida, conversor USB TTL (Transistor-Transistor Logic) CP2102 Serial RS232 integrado e programação via OTA (Over the Air). Estas características aliadas ao baixo custo fundamentaram a decisão da sua seleção como módulo Wi-Fi do sistema.

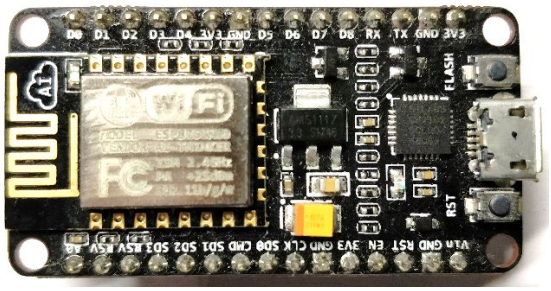


Figura 23. NodeMCU ESP8266 ESP-12E.

3.2.2 FireBeetle ESP32

O DFRobot FireBeetle (Figura 24) é uma placa de baixo consumo de energia que integra um módulo ESP-WROOM-32 Dual-Core, que suporta comunicação dual-mode: Wi-Fi (IEEE 802.11 b/g/n/d/e/I/k/r) e Bluetooth v4.2. Possui 10 pinos digitais (entrada/saída), 5 pinos analógicos, SPI, I2C e I2S (Inter-IC Sound). A sua programação pode ser realizada com MicroPython ou através do Arduino IDE. Esta placa tem a possibilidade de funcionar como módulo Wi-Fi, no entanto foi o suporte ao Bluetooth v4.2 (BLE) e o baixo custo que justificaram a sua escolha como módulo BLE do sistema.

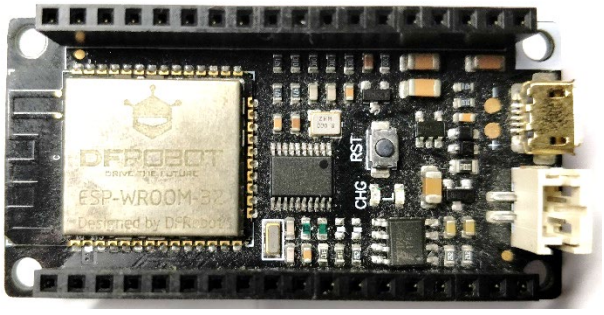


Figura 24. FireBeetle ESP32.

3.2.3 Raspberry Pi 4 Model B

A Raspberry Pi 4, apresentada na Figura 25, é o mais recente computador de tamanho reduzido desenvolvido pela Raspberry Pi Foundation, e foi utilizada para implementar o *hub/gateway* do sistema. As suas principais características são o processador Broadcom BCM2711 Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz, dual-band 2.4GHz e 5GHz wireless LAN, Bluetooth 5.0 (BLE), LPDDR4-2400 SDRAM e portas USB 3.0.

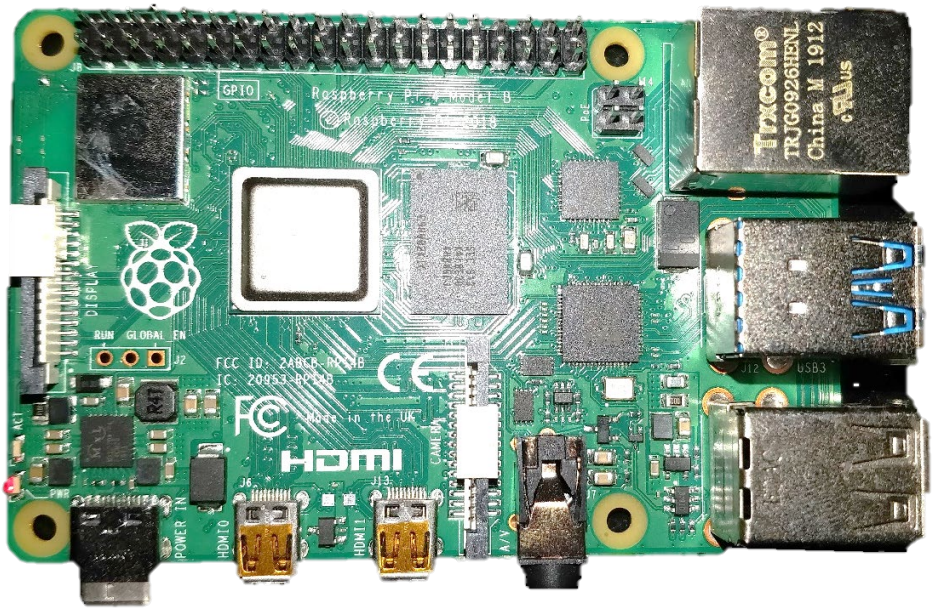


Figura 25. Raspberry Pi 4.

3.2.4 Texas Instruments TMDSDOCK28335

O TMDSDOCK28335 (Figura 26) é um kit de desenvolvimento da Texas Instruments (TI) que consiste essencialmente numa *docking station* e num TMS320F28335 MCU (Microcontroller Unit). É uma ferramenta que permite implementar e comprovar soluções para muitas aplicações de controlo em tempo real, como por exemplo, acionamento de motores, inversores solares, veículos elétricos e processamento de sinal [32]. No contexto desta dissertação, o TMS320F28335 foi utilizado para controlar e monitorizar o carregamento da bateria de um veículo elétrico.

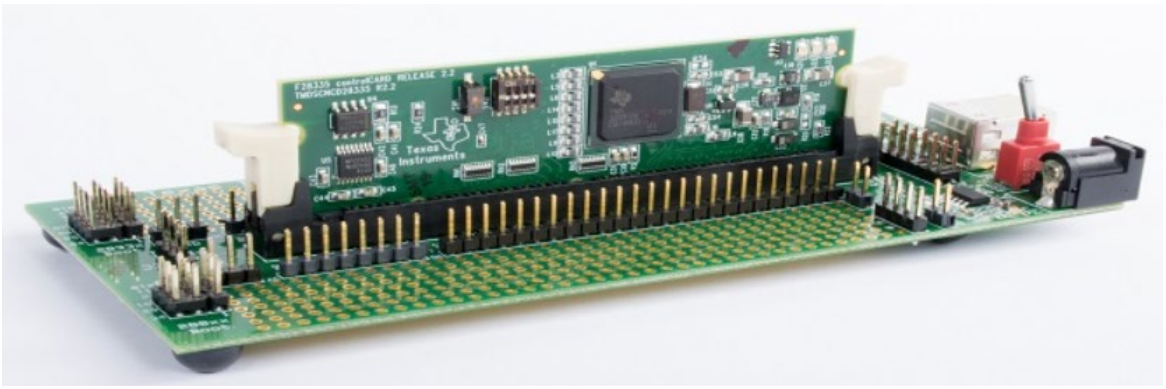


Figura 26. Texas Instruments TMDSDOCK28335 [33].

A *docking station* é uma pequena *baseboard* que aceita qualquer DIMM100 plug-in *control card* e permite acesso a todos os principais GPIOs e sinais ADC do MCU. Disponibiliza duas áreas de prototipagem (uma em cada lado do conector DIMM100), RS232, conector JTAG e *on board* USB *JTAG emulation*.

O F28335 é um MCU de 32 bits que pertence à família de microcontroladores de alta performance Delfino™ da linha C2000™ da Texas Instruments. Tem velocidade de processamento até 150 MHz e possui 256K x 916 de *flash memory* e 34K x 916 de SRAM. Disponibiliza interfaces de comunicação como SPI, UART e I2C, 64 pinos de I/O de propósito geral e vários blocos integrados, incluindo um conversor analógico-digital (ADC) de 12 bits com 80 ns de tempo de conversão [34].

A própria TI fornece o IDE Code Composer Studio, composto por um conjunto de ferramentas para desenvolver e depurar aplicativos incorporados. Inclui um compilador C/C ++ otimizado, editor de código-fonte, ambiente de criação de projeto, depurador, *profiler*, entre outros recursos [35].

3.2.5 Display OLED 0.91' WS

De forma a facilitar a depuração foi adicionado um módulo de display OLED da Waveshare nos módulos Wi-Fi. Este display tem diagonal de 0,91 polegadas, resolução de 128x32 pixels, controlador incorporado e comunicação I2C (Figura 27).

Figura 27. Display OLED 0.91' [36].

O controlador incorporado é o SSD1306 para ecrãs de resolução 128x64. Este OLED tem apenas 128x32 pixels, por isso usa apenas parte do buffer do SSD1306. A tensão de alimentação do módulo pode variar entre 3,3 V e 5 V. A Figura 28 permite a visualização dos pinos referentes à alimentação e à comunicação I2C (SDA - Data Line e SCL - Clock Line).

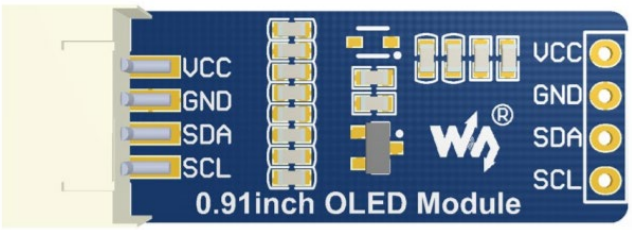


Figura 28. Interface I2C do ecrã OLED 0.91' [36].

3.2.6 Sensor DHT11

O DHT11 (Figura 29) é um sensor de temperatura e humidade de baixo custo com saída de sinal digital. Foi utilizado juntamente com os módulos Wi-Fi e BLE para determinar o valor da temperatura e da humidade em cada divisão da casa.

Figura 29. Sensor DHT11.

O elemento sensor de temperatura é um termístor do tipo NTC (Negative Temperature Coefficient) e o sensor de humidade é do tipo HR202. O circuito interno faz a leitura dos sensores e comunica a um microcontrolador através de um sinal serial de uma via. Na Tabela 2 encontram-se as principais especificações técnicas do sensor.

Tabela 2. Especificações técnicas do sensor DHT11.

Fonte de energia	3-5,5V (DC)
Sinal de saída	Sinal digital de barramento único (*single-bus*)
Faixa de medição	Humidade: 20-90% RH Temperatura: 0-50° Celsius
Tempo de resposta	2 segundos
Precisão	Humidade: ± 4% RH Temperatura: ± 2° Celsius

Todas as leituras do sensor são enviadas usando apenas um barramento de dados com um fio, o que reduz o custo e aumenta a distância. Para enviar dados através de um barramento único, é necessário haver um protocolo para que o transmissor e recetor se possam entender. O protocolo utilizado no DHT11 pode ser dividido em 4 etapas: pedido, resposta, transmissão de dados e fim de transmissão.

1) **Pedido:** Para fazer com que o DHT11 envie as leituras do sensor é necessário realizar um pedido/solicitação.
2) **Resposta:** Após o pedido por parte do microcontrolador é enviada pelo DHT11 uma resposta automática que indica que recebeu o pedido com sucesso.
3) **Transmissão de dados:** O que vem a seguir à reposta automática são os dados do sensor. Estes dados são armazenados num pacote de 5 segmentos de 8 bits cada (Figura 30) em que os dois primeiros segmentos dizem respeito ao valor da humidade relativa (parte inteira mais parte decimal), os dois segmentos seguintes ao valor de temperatura (parte inteira mais parte decimal) e o último segmento à soma dos 4 primeiros segmentos. Se o valor do *checksum* não corresponder à soma dos 4 primeiros segmentos, significa que os dados não foram recebidos corretamente.

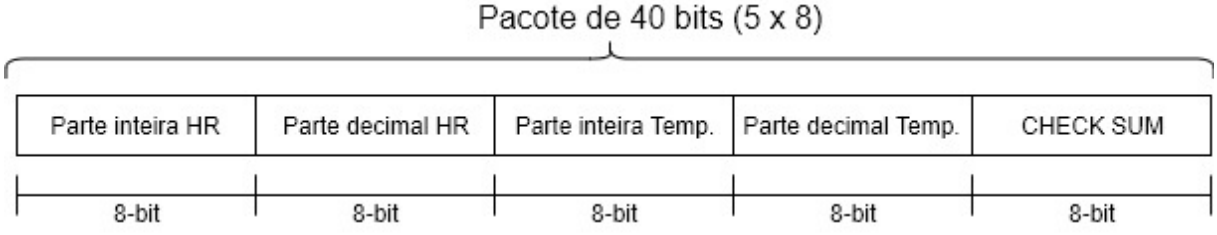


Figura 30. Protocolo DHT11: Pacote de dados.

4) **Fim de transmissão:** Após o envio do pacote o DHT11 finaliza a transmissão.

3.2.7 Módulo relé de 1 canal

O Módulo Relé 5V de 1 canal (Figura 31) é equipado com 1 relé eletromecânico com LED indicador do seu estado. Possui isolamento via acoplador ótico, a tensão de saída é de 30/28 V DC a 10 A ou 250/125 V AC a 10 A e o tempo de reposta é de 5 a 10 ms. Pode ser controlado diretamente por microcontroladores Arduino, AVR, PIC, ARM entre outros.

Figura 31. Módulo Relé de 1 canal [37].

O seu funcionamento consiste no seguinte:

- Quando a entrada do sinal está em nível baixo, o LED indicador do estado acende e o acoplador ótico gera uma corrente na base do transístor. Isso faz com que a bobina do relé seja percorrida por uma corrente e o contato normalmente aberto do relé seja fechado.
- Quando a entrada do sinal está em nível alto, o LED indicador do estado desliga e o contato normalmente fechado do relé é fechado.

O módulo foi utilizado no protótipo da tomada inteligente da secção 4.3.2 para controlar cargas e dispositivos como motores AC/DC, lâmpadas, eletrodomésticos, entre outros.

3.2.8 Sensor de tensão AC ZMPT101B

O sensor de tensão AC ZMPT101B (Figura 32) é um módulo de alta precisão que foi utilizado no protótipo da tomada inteligente para detetar a existência de tensão alternada e fazer a medição do valor.

Figura 32. Módulo sensor de tensão AC [38].

As características principais são:

- Transformador: ZMPT101B
- Tensão de alimentação do módulo: 5 a 30 V DC
- Tensão de entrada: 0 a 250 VAC
- Corrente de entrada e saída nominal: 2 mA
- Proporção: 1000:1000
- Faixa linear: 0 - 1000 V
- Linearidade: 0,2%

3.2.9 Sensor de corrente AC DFRobot Gravity 20 A

O sensor analógico de corrente AC da DFRobot (Figura 33) foi utilizado no protótipo da tomada inteligente pois elimina a necessidade de cortar fios ou reconectar os circuitos. Basta prender a ponta de prova do transformador de AC (abre para abraçar o fio e depois fecha) na linha AC e, em seguida, conectar o *jack* de 3,5 mm ao módulo de conversão de sinal para ler o valor atual da corrente. A saída analógica foi projetada para ser compatível com as entradas analógicas dos microcontroladores (3,3 V ou 5 V) e permite leituras de 0 a 20 A.

Figura 33. Sensor de corrente AC [39].

3.2.10 Multiplexador Analógico 4051

O multiplexador 4051 (Figura 34) é um comutador de oito canais analógicos controlado digitalmente pelos pinos A, B e C. Foi utilizado no protótipo da tomada inteligente para permitir a ligação de dois sensores ao único ADC do módulo NodeMCU.

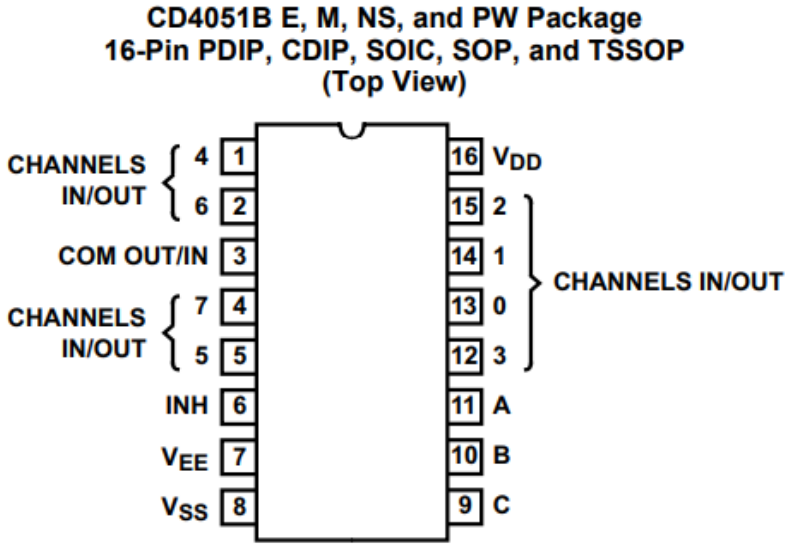


Figura 34. Pinagem do multiplexador 4051 [40].

Na Tabela 3 estão representadas todas as combinações possíveis dos pinos de controlo e as respetivas saídas. L corresponde a baixo nível de tensão e H corresponde a baixo nível de tensão.

Tabela 3. Tabela de funcionamento do multiplexador 4051.

A	B	C	Saída
L	L	L	Canal 0
L	L	H	Canal 1
L	H	L	Canal 2
L	H	H	Canal 3
H	L	L	Canal 4
H	L	H	Canal 5
H	H	L	Canal 7
H	H	H	Canal 8

3.3 Firmware para NodeMCU

O firmware para esta placa foi desenvolvido utilizando o Arduino IDE devido à sua flexibilidade e simplicidade. Uma vez que o Arduino não tem suporte nativo para este módulo, foi necessário adicionar o Arduino *core* para ESP8266 [41], representado na Figura 35, através do gestor de placas. Este Arduino *core* contém bibliotecas que permitem comunicar através de Wi-Fi utilizando TCP e UDP (User Datagram Protocol), configurar a placa como servidor HTTP, multicast DNS e servidor DNS (Domain Name System), fazer atualizações OTA, usar um sistema de arquivos na memória *flash*, e trabalhar com cartões SD, servomotores, e periféricos SPI e I2C.

Figura 35. Arduino *core* para o módulo ESP8266.

3.3.1 Conexão à rede Wi-Fi

Para gerir a conexão Wi-Fi da placa NodeMCU foi utilizada a biblioteca WiFiManager [42] (Figura 36).

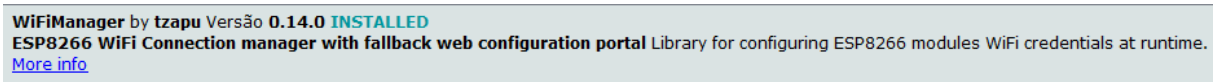


Figura 36. Biblioteca WiFiManager para gerir a conexão Wi-Fi.

Esta biblioteca permite que as credenciais da rede Wi-Fi sejam introduzidas e guardadas em tempo de execução através de um portal de configuração (página Web). Esta funcionalidade é bastante útil, pois deixa de ser necessário alterar as credenciais no código caso seja preciso ligar a outra rede. Além disso, o WiFiManager permite a recolha de parâmetros extras, como se verifica na Figura 41.

O modo de funcionamento do WiFiManager é o seguinte:

- A placa ao ligar é configurada no modo *station* e tenta se conectar a um *access point* guardado anteriormente (Figura 37).

```
*WM: AutoConnect
*WM: Connecting as wifi client...
*WM: Using last saved values, should be faster
```

Figura 37. NodeMCU no modo *station* (*output* da porta série).

- Caso não se consiga conectar (ou não haja redes anteriores guardadas) coloca a placa no modo *access point* e ativa o DNS e o servidor Web (Figura 38). O endereço IP, o nome e a password do *access point* da placa podem e foram configurados.

```
*WM: Connection result:
*WM: 0
*WM:
*WM: Configuring access point...
*WM: Smart Home Device
*WM: password
*WM: AP IP address:
*WM: 192.168.4.1
*WM: HTTP server started
```

Figura 38. Configuração da NodeMCU no modo *access point* (*output* da porta série).

- A conexão a este *access point* é possível através de um dispositivo habilitado para Wi-Fi e com navegador Web (e.g., computador, smartphone ou tablet) (Figura 39).

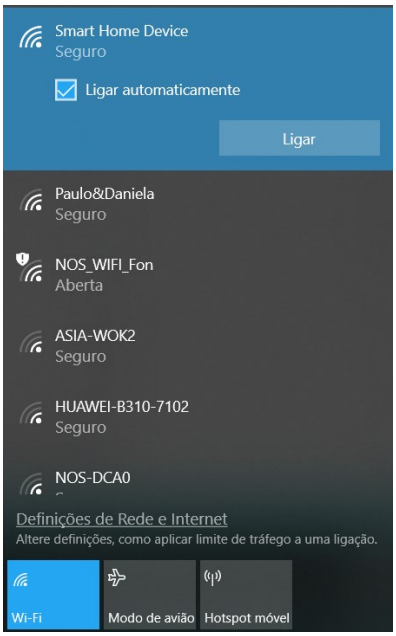


Figura 39. NodeMCU na lista de *access points* disponíveis com o nome *Smart Home Device* (nome atribuído).

- Após a conexão ao *access point*, devido ao portal cativo e servidor DNS, o utilizador é redirecionado para o portal de configuração (Figura 40) através do browser.

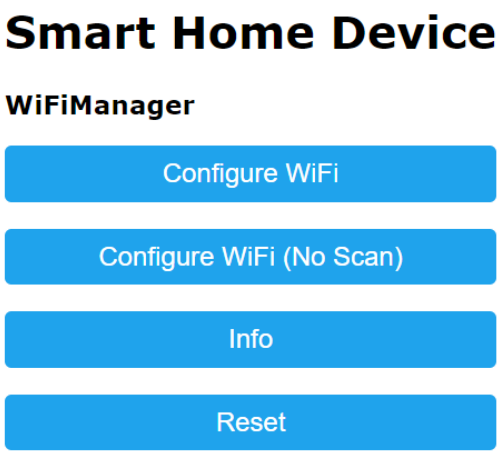


Figura 40. Portal de configuração do WiFiManager.

- Selecionando a opção "Configure WiFi", surge uma lista dos *access points* disponíveis para que seja introduzido o SSID e *password* da rede Wi-Fi que se pretende conectar. Adicionalmente, foram também configurados parâmetros extra para que fosse possível introduzir e guardar o endereço, porta, *username* e *password* do *broker* MQTT (Figura 41).

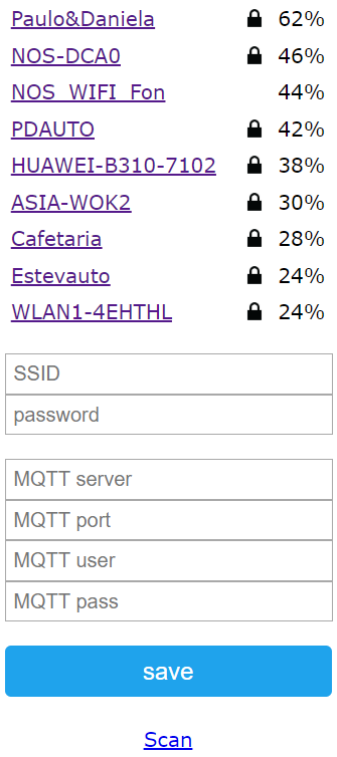


Figura 41. Introdução de credenciais para a rede Wi-Fi e para a conexão ao *broker* MQTT.

- Após clicar na opção "save" o dispositivo tenta a conexão à rede. Se for bem-sucedido, guarda as credenciais e continua a execução do código na placa. Caso contrário, continua no modo *access point* (Figura 42).

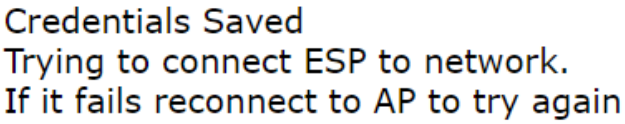


Figura 42. Mensagem no browser após salvar os dados introduzidos no portal.

É importante referir que, como foram adicionados parâmetros extras (credenciais do MQTT broker), foi necessário guardar (e carregar quando necessário) esses valores. Para esse fim foi utilizado o SPIFFS (Serial Peripheral Interface Flash File System), um sistema de arquivos leve para microcontroladores com *SPI flash chip*. Visto que esta placa tem 4 MB de memória *flash*, foi particionado 1 MB para o SPIFFS e 3 MB para o firmware (Figura 43).

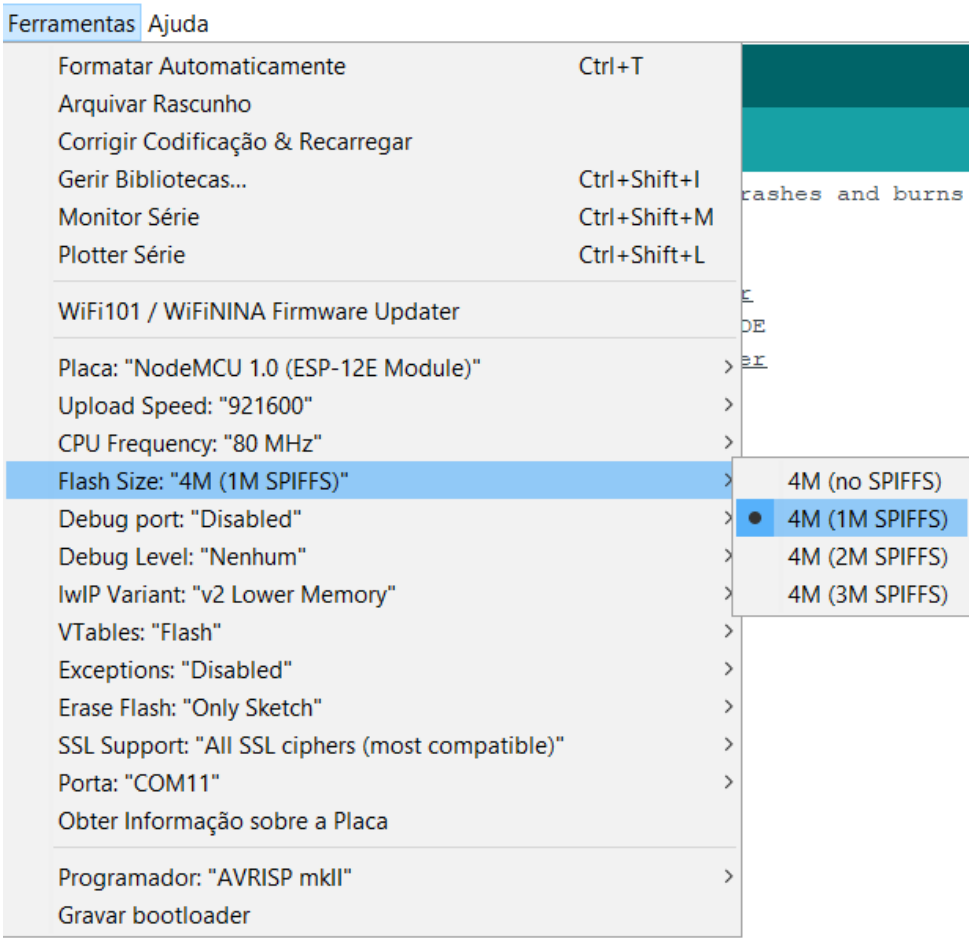


Figura 43. Particionamento da memória flash de 4MB.

De modo a estruturar os parâmetros de uma forma mais compacta no SPIFFS, foi criado um objeto JSON (JavaScript Object Notation) utilizando a biblioteca ArduinoJson [43] (Figura 44).

```
{
  "mqtt_server": "192.168.1.213",
  "mqtt_port": "1883",
  "mqtt_user": "rubenfigueiredo",
  "mqtt_pass": "raspberry"
}
```

Figura 44. Objeto JSON com os parâmetros extra.

3.3.2 Atualizações via OTA

A atualização via OTA (Over the Air) é o processo de carregar firmware num módulo utilizando a conexão Wi-Fi, em vez de uma porta série. Esta funcionalidade é extremamente útil em caso de acesso físico limitado ou inexistente ao módulo. Pode ser realizada através dos seguintes métodos: Arduino IDE, Web Browser e HTTP Server. O método utilizado foi com o Arduino IDE pois destina-se sobretudo à fase de desenvolvimento de software [44] (Figura 45).

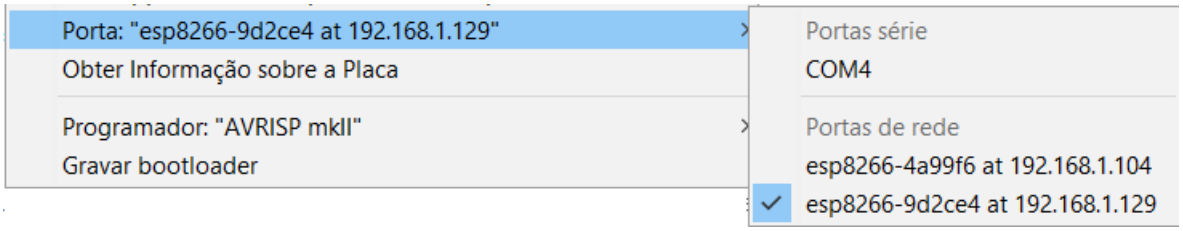


Figura 45. Atualização via OTA através de portas de rede no Arduino IDE.

As principais vantagens da atualização via OTA são a redução de tempo para atualizar cada módulo e o facto de um único computador ser capaz de enviar atualizações para vários módulos que estejam na mesma rede. É fundamental que o código que permite realizar atualizações via OTA esteja sempre presente no firmware que é carregado para a placa, ou seja, ao código do firmware desenvolvido é acrescentado o código responsável por esta função. As bibliotecas essenciais para a implementação desta funcionalidade foram a ESP8266WiFi.h para o acesso à rede Wi-Fi e a ArduinoOTA.h para tratar o recebimento de código e auto-gravação via Wi-Fi.

3.3.3 Cliente MQTT

Foi necessário implementar um cliente MQTT compatível com a placa, para que esta conseguisse comunicar com as outras entidades da plataforma IoT através deste protocolo. De forma geral o cliente MQTT nesta placa publica os dados adquiridos pelo(s) sensor(es) e subscreve os tópicos que permitem saber ou calcular o valor adequado para a saída do(s) atuador(es). A biblioteca utilizada para este propósito chama-se MQTT e foi desenvolvida por Joel Gaehwiler [45] e importada através do gestor de bibliotecas do Arduino (Figura 46). A última versão disponível é a 2.4.3 e permite implementar clientes MQTT 3.1.1, assim como publicar e subscrever com diferentes níveis de QoS (0,1 e 2).

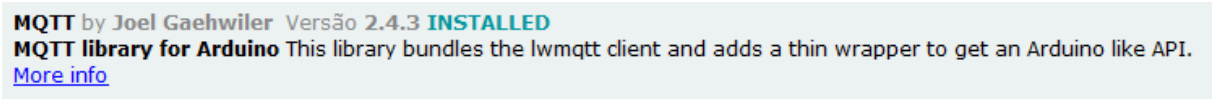


Figura 46. Biblioteca utilizada para implementar o cliente MQTT.

Dois exemplos da estrutura dos tópicos utilizados estão representados na Figura 47.

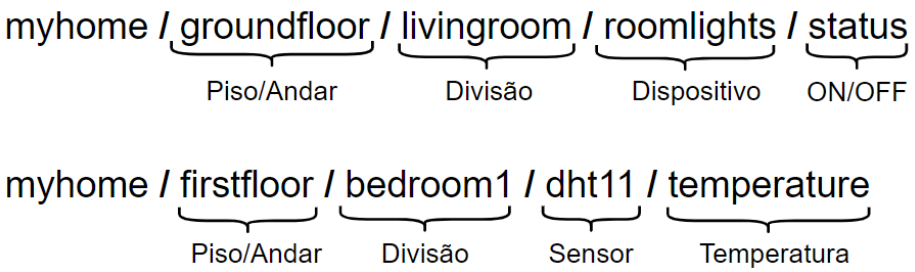


Figura 47. Exemplo da estrutura dos tópicos utilizados.

3.3.4 Comunicação I2C com display OLED

Foi utilizado o módulo de *display* OLED, referido na secção 3.2.5, para demonstrar o estado da conexão do cliente MQTT (Figura 48) facilitando assim o processo de depuração.

Figura 48. Módulo de display OLED com o estado da conexão do cliente MQTT.

A comunicação entre a placa NodeMCU e o ecrã OLED é realizada através do protocolo I2C, que utiliza apenas 2 pinos do MCU, permitindo que os restantes pinos sejam utilizados para conectar

sensores. A implementação foi realizada com recurso às bibliotecas Adafruit GFX [46] e Adafruit SSD1306 [47] (Figura 49).

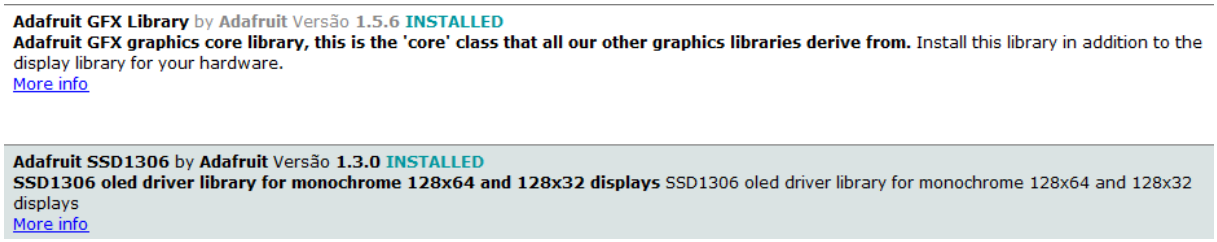


Figura 49. Bibliotecas para comunicar com ecrã OLED.

Este *display* conecta-se à placa NodeMCU seguindo o esquema da Figura 50, com dois fios para alimentação (VCC e GND) e dois fios para dados (SCL – Serial Clock e DAS – Serial Data).

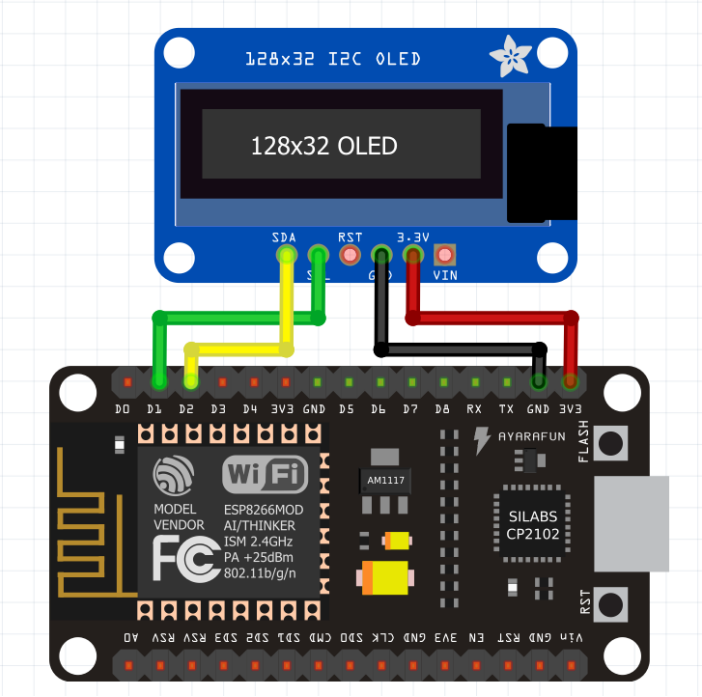


Figura 50. Esquema de conexão da placa NodeMCU com ecrã OLED.

3.3.5 Comunicação com a placa TMDSDOCK28335 através da porta série

Foi estabelecida comunicação série utilizando níveis lógicos TTL entre a placa NodeMCU e a placa TMDSDOCK28335 (secção 3.2.4) com uma taxa de transmissão de 9600 bps. Conforme representado na Figura 51, o fio verde foi ligado ao pino TX da placa TMDSDOCK28335 (GPIO22) e ao pino RX da NodeMCU, enquanto o fio branco foi ligado ao pino RX da placa TMDSDOCK28335 (GPIO23) e ao pino TX da NodeMCU. O fio preto apenas conecta o *ground* de ambas as placas.

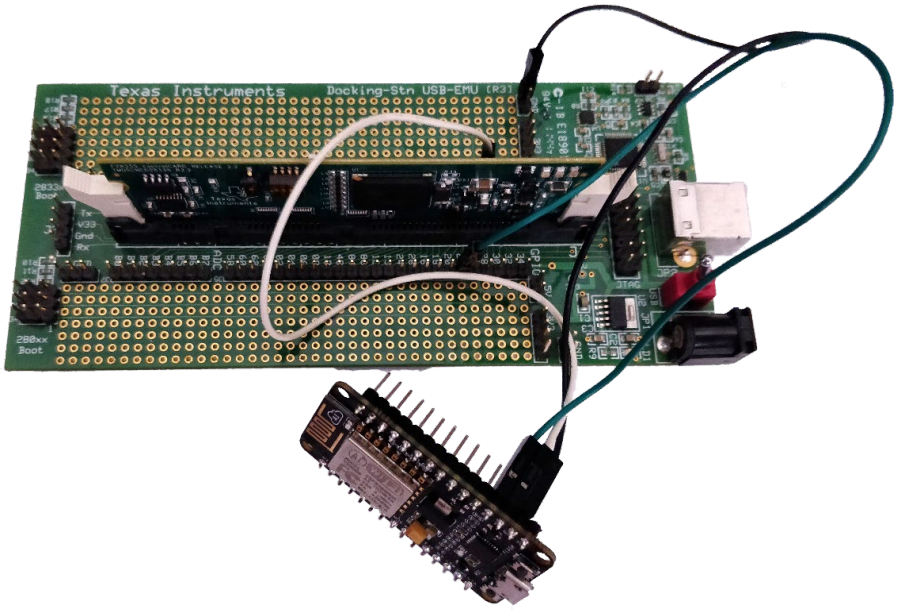


Figura 51. NodeMCU conectada a TMDSDOCK28335 através da porta série.

Esta comunicação permite o envio e receção de mensagens/comandos via porta série de forma a controlar e monitorizar o carregamento da bateria do veículo elétrico. O algoritmo de comunicação implementado na NodeMCU está representado na Figura 52 e o algoritmo de comunicação implementado na TMDSDOCK28335 está representado na Figura 53 para uma melhor compreensão.

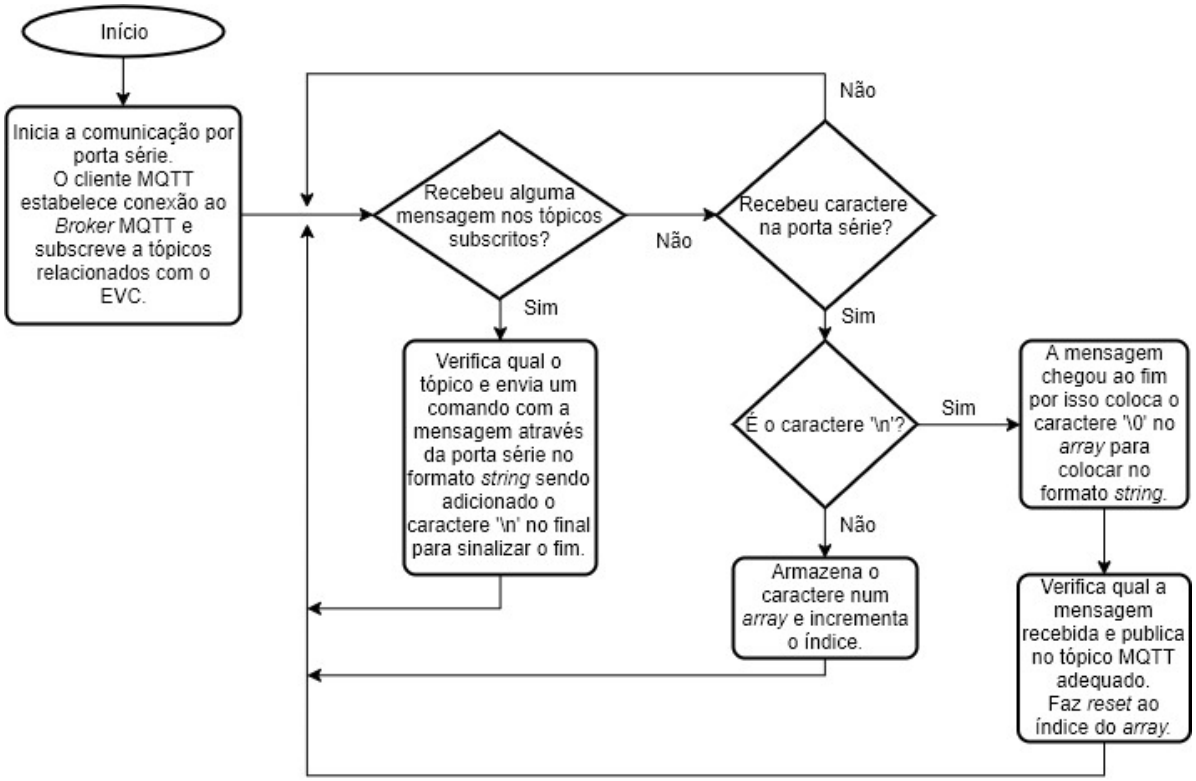


Figura 52. Fluxograma do algoritmo de comunicação implementado na NodeMCU.

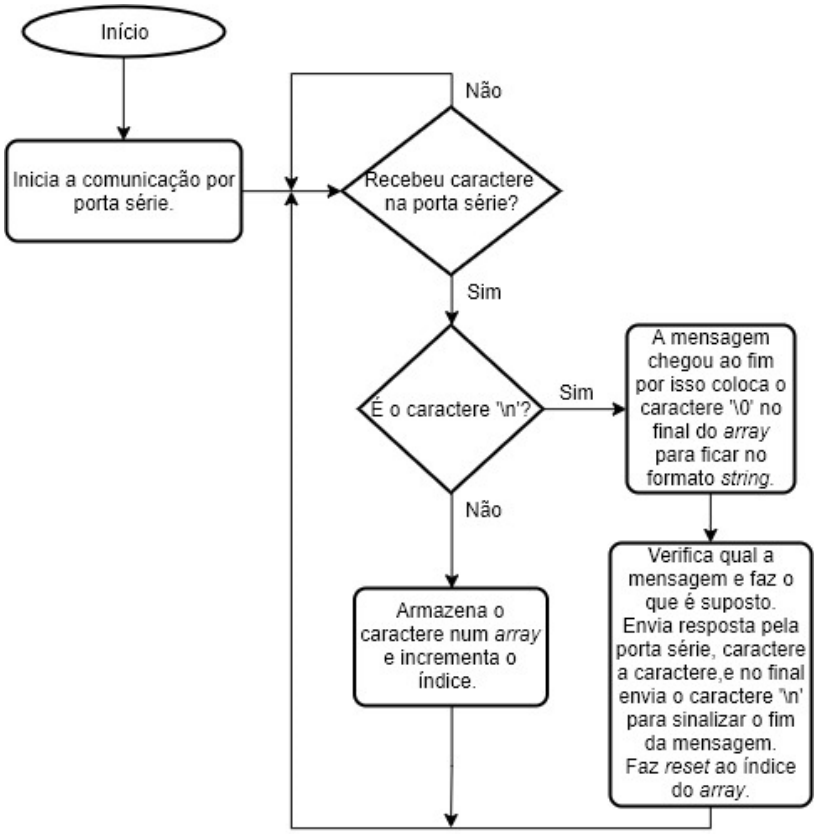


Figura 53. Algoritmo de comunicação implementado na TMDSDOCK28335.

3.3.6 Comunicação com o sensor DHT11

De forma a adquirir os valores de temperatura e humidade obtidos pelo sensor DHT11 foi necessário importar/instalar uma biblioteca que implementasse o protocolo de comunicação. A biblioteca utilizada é disponibilizada pela empresa Adafruit, está disponível em [48] e representada na Figura 54.

DHT sensor library by **Adafruit** Versão **1.3.7** **INSTALLED**
Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors
More info

Figura 54. Biblioteca utilizada para comunicar com o sensor DHT11.

O esquema de conexão entre a placa NodeMCU e o sensor DHT11 está representado na Figura 55, sendo que o fio vermelho e preto correspondem à alimentação do sensor e o fio azul à transferência de dados. A obtenção de valores de temperatura e humidade pode ser realizada periodicamente ou sempre que surge uma alteração do seu valor. Neste caso foi implementada uma leitura de 1 em 1 minuto através de um *timer.*

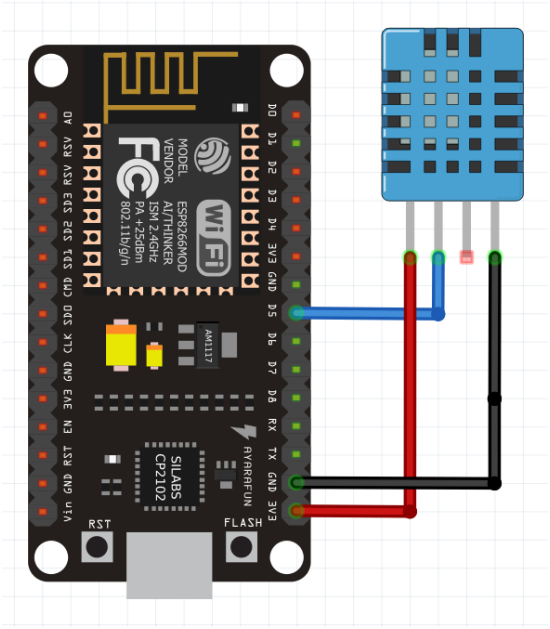


Figura 55. Esquema de conexão da placa NodeMCU com o sensor DHT11.

3.3.7 Obtenção dos valores do sensor de tensão AC e cálculo do valor eficaz

Inicialmente foi necessário ajustar o sensor através de um potenciómetro com o auxílio de um osciloscópio de forma que a onda sinusoidal gerada quando ligado à rede não estivesse saturada na parte superior e não houvesse perda de valores. O sensor foi alimentado com 3,3 V da placa NodeMCU e, na Figura 56, é possível ver a onda gerada quando conectado a 240 V. Como a tensão alternada tem parte negativa e parte positiva, e isso não pode acontecer num ADC unipolar, o sensor soma um *offset* (1,65 V neste caso) para tornar a onda sempre positiva, i.e., com valor médio de 1,65 V.

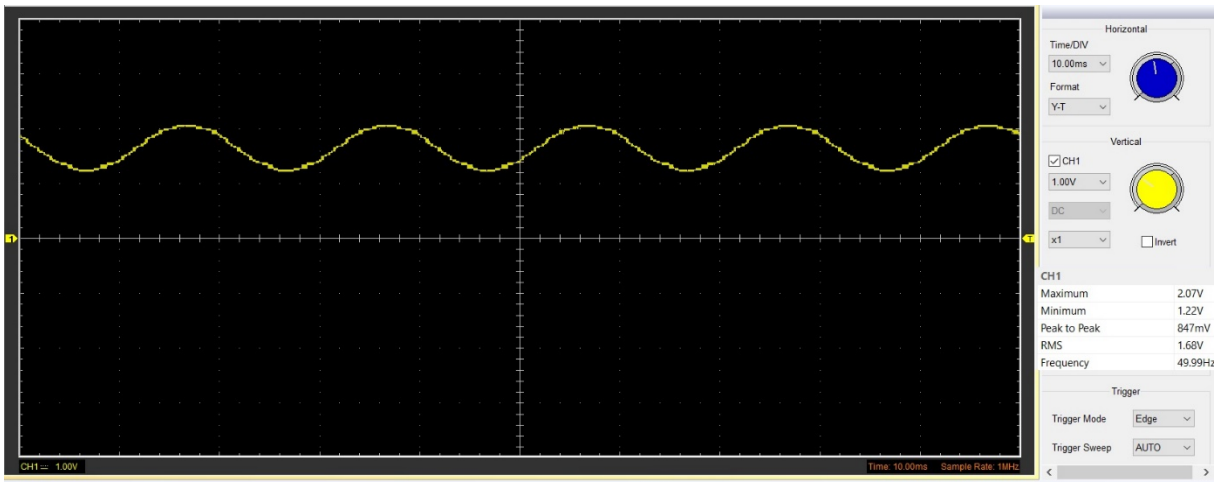


Figura 56. Valores de saída do sensor de tensão quando conectado à rede elétrica.

Para converter os valores analógicos gerados pelo sensor de tensão em valores digitais foi utilizado o ADC da NodeMCU que tem 10 bits de resolução, ou seja, 1024 (2^{10}) valores diferentes possíveis. A onda resultante está representada na Figura 57.

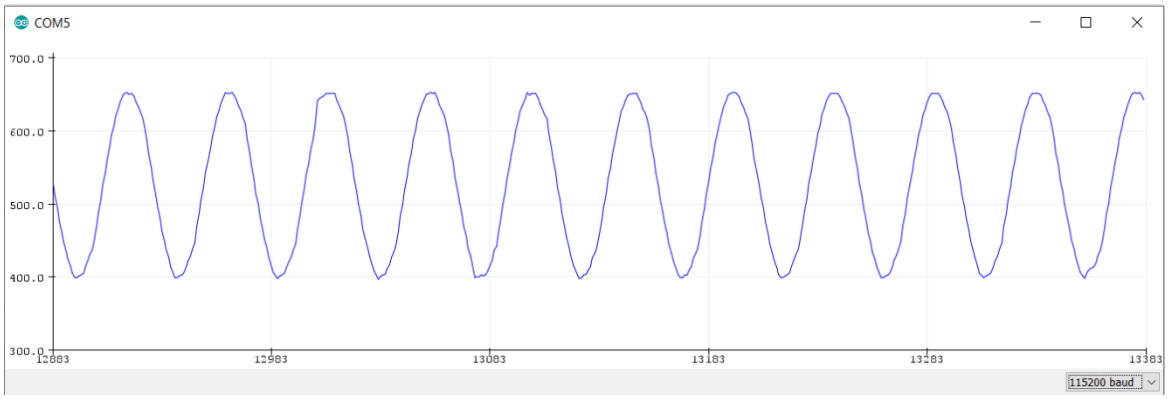


Figura 57. Valores de saída do ADC quando está conectado o sensor de tensão à rede elétrica.

Por último foi utilizada a biblioteca EmonLib [49] para obter os valores de pico e calcular a tensão eficaz através da equação (1), em que V_p é o valor de tensão máximo (pico).

$$V_{eficaz} = \frac{V_p}{\sqrt{2}} \quad (1)$$

3.3.8 Obtenção dos valores do sensor de corrente AC e cálculo do valor eficaz

Para a obtenção e cálculo da corrente eficaz foi utilizado o módulo de conversão de sinal de corrente alternada (Figura 58) disponibilizado pela DFRobot que converte o sinal adquirido pela sonda de transformador AC de 0 a 1 V AC num sinal analógico de 0,2 a 2,8 V DC.

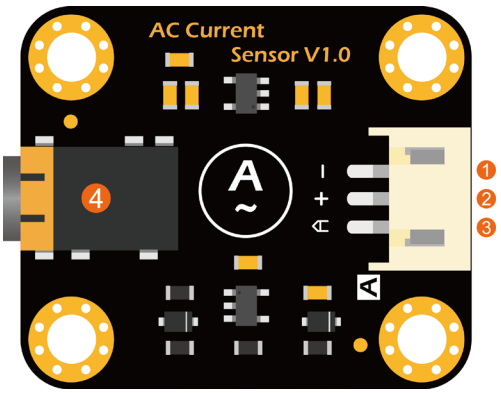


Figura 58. Módulo de conversão de sinal de corrente alternada [50].

Na Figura 59 é possível verificar uma onda sinusoidal com frequência de 50 Hz (frequência da rede elétrica), proveniente da sonda, quando um equipamento está ligado. Esse sinal é utilizado como entrada

no módulo de conversão através da inserção de um *jack* de 3,5 mm na ficha identificada pelo número quatro da Figura 58.

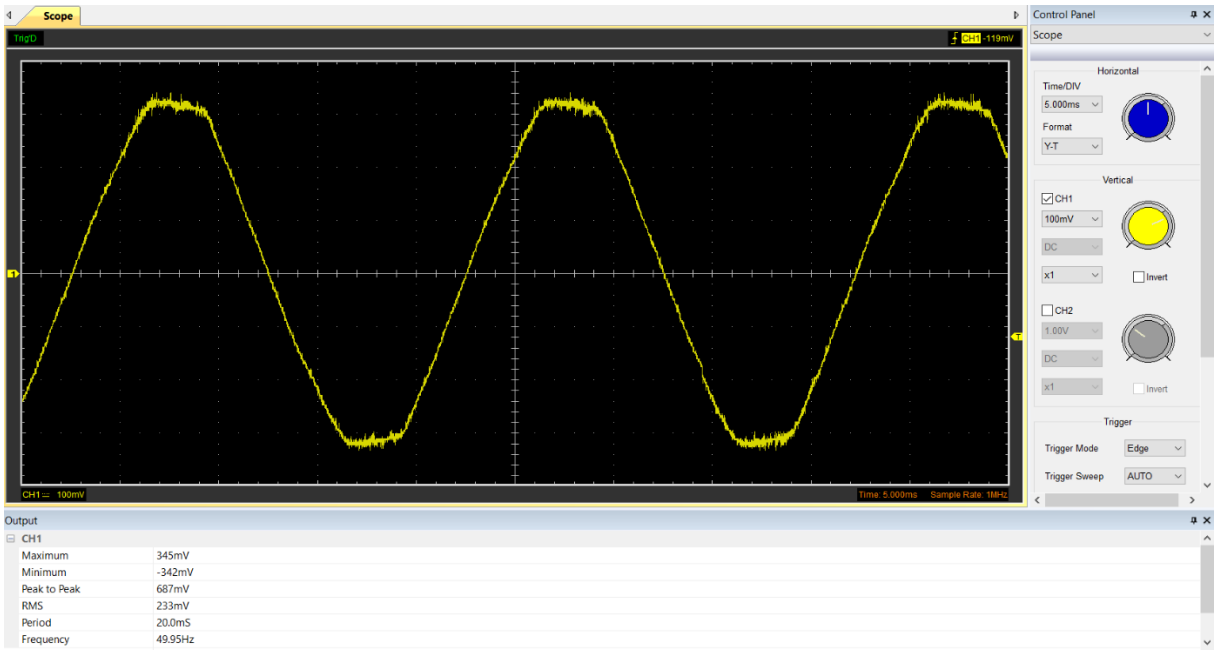


Figura 59. Sinal adquirido pela sonda de transformador de corrente alternada de tipo aberto.

O sinal convertido é apresentado na Figura 60 e, como era de esperar, a componente alternada foi retirada resultando num sinal DC positivo que pode ser colocado diretamente no ADC da NodeMCU.

Figura 60. Sinal de saída do módulo de conversão de sinal de corrente alternada.

Visto que o ADC, neste caso, tem 10 bits de resolução e a tensão de referência (3,3 V) corresponde ao valor 1024, para 0,603 V (valor médio do sinal analógico para este caso) obtém-se:

$$\frac{0{,}603 \times 1024}{V_{ref}} \cong 187 \tag{2}$$

Para confirmar este valor foi desenhado um gráfico ao longo do tempo com os valores calculados pelo ADC (Figura 61).

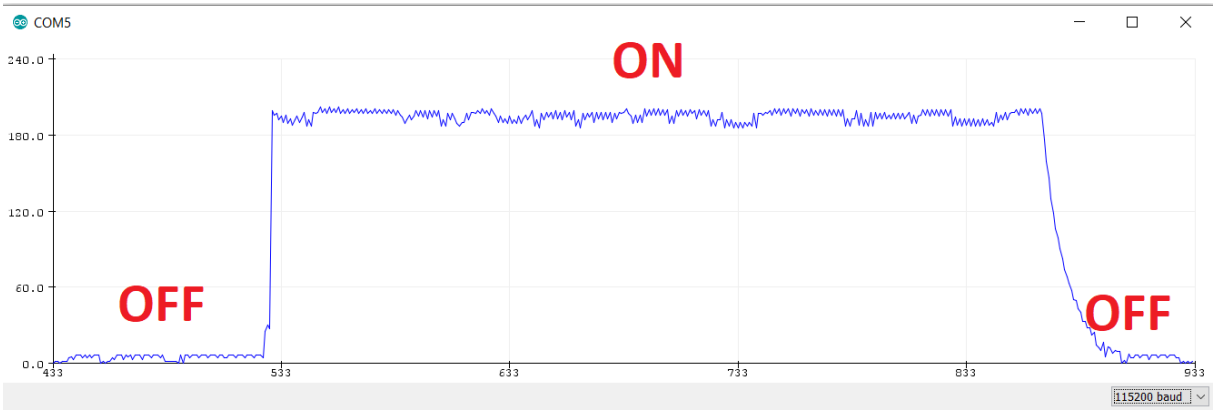


Figura 61. Sinal convertido pelo ADC de 10 bits de resolução do módulo.

Finalmente para calcular o valor de corrente são retirados alguns valores do ADC para fazer uma média (devido às pequenas variações). De seguida é calculado o valor eficaz, convertido de volta para um sinal analógico, divido por 2 (porque o circuito é amplificado 2 vezes) e por último multiplicado por 20 (porque evolui linearmente de acordo com o valor máximo do sensor). Como exemplo, para um valor de 187 obtém-se:

$$I_{medida} = \frac{\left(\frac{(187 \times 0{,}707)}{1024} \times V_{ref}\right)}{2} \times 20 \cong 4{,}26\ A \tag{3}$$

3.4 Firmware para Firebeetle ESP32

O firmware para esta placa foi desenvolvido com o Arduino IDE. Dado que não existe suporte nativo para este módulo, foi adicionado o Arduino *core* para a ESP32 [51], desenvolvido pela Espressif Systems, Figura 62, através do gestor de placas.

Figura 62. Arduino core para o módulo ESP32.

Após a instalação do *core* para ESP32, surgem na lista de placas instaladas diversas opções, como as que estão apresentadas na Figura 63. Obviamente a placa FireBeetle ESP32 foi a opção selecionada, por ser a placa que está a ser utilizada.

Figura 63. Seleção da placa FireBeetle ESP32 na lista de placas instaladas na IDE do Arduino.

3.4.1 Servidor BLE

De modo a implementar as funcionalidades do BLE na placa ESP32 foram importadas as bibliotecas disponibilizadas por Neil Kolban, cuja documentação pode ser consultada em [52]. Foi criado um servidor BLE que, assim que recebe uma conexão, envia notificações com os dados pretendidos. As etapas necessárias à implementação do servidor BLE foram:

1. Criar um servidor BLE.
2. Criar um serviço BLE com um UUID (Universally Unique Identifier).
3. Criar uma característica no serviço BLE com um UUID.
4. Criar um descritor na característica BLE para ativar notificações.
5. Iniciar o serviço.
6. Iniciar o anúncio na rede.

As propriedades atribuídas à característica foram de leitura, escrita e notificação, como se pode verificar na Figura 64.

```
// Create a BLE Characteristic
pCharacteristic = pService->createCharacteristic(
                    CHARACTERISTIC_UUID,
                    BLECharacteristic::PROPERTY_READ   |
                    BLECharacteristic::PROPERTY_WRITE  |
                    BLECharacteristic::PROPERTY_NOTIFY //|
                    //BLECharacteristic::PROPERTY_INDICATE
                  );
```

Figura 64. Criação da característica BLE.

Após a implementação do servidor BLE com o serviço pretendido foi utilizada a aplicação Android "nRF Connect for Mobile" da Nordic Semiconductor para verificar se tudo foi bem configurado [53]. Esta aplicação é uma ferramenta genérica que permite procurar, anunciar e explorar dispositivos Bluetooth de baixa energia (BLE) e comunicar com eles. Na Figura 65 é possível observar que a ESP32 aparece nos dispositivos BLE disponíveis para conexão com o nome ESP32 BLE DEVICE 1 (nome atribuído).

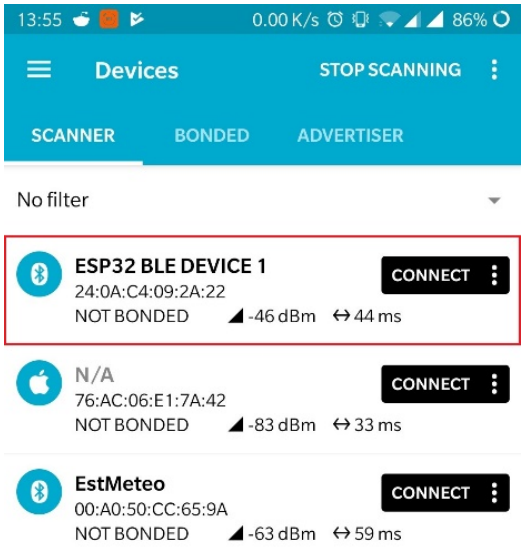


Figura 65. ESP32 BLE DEVICE 1 na lista de dispositivos BLE encontrados.

Ao efetuar a conexão é obtida informação sobre os serviços, características e descritores disponíveis. (Figura 66).

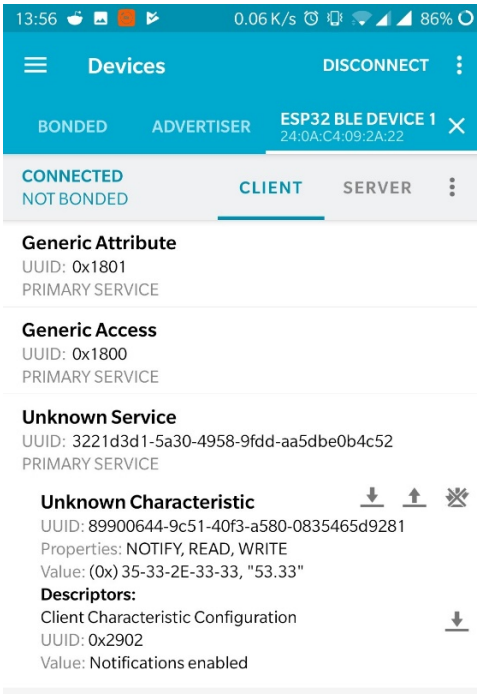


Figura 66. Visualização dos serviços e características do servidor BLE através da aplicação "nRF Connect for Mobile"

Na Figura 67 está representado de forma sucinta o fluxo de execução de código sendo que quando o dispositivo está conectado, ou seja, algum cliente BLE se conectou ao servidor, apenas são escritos na característica os valores obtidos pelo sensor que são diferentes dos anteriores.

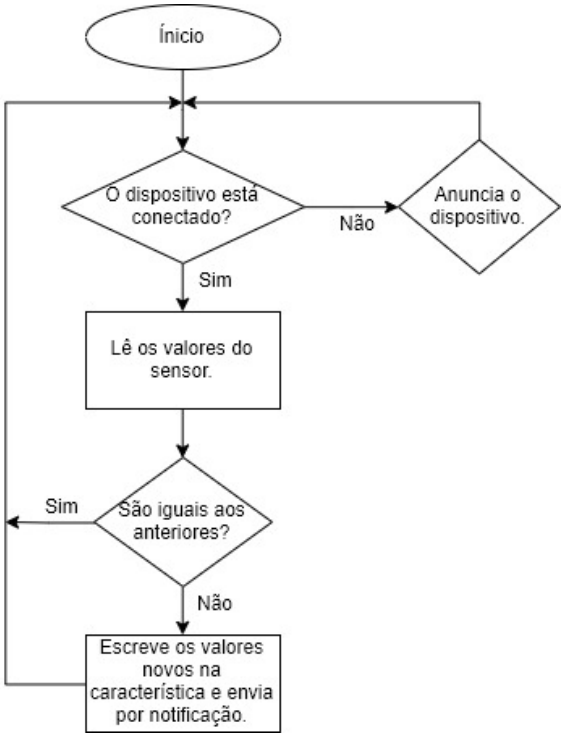


Figura 67. Fluxograma do funcionamento do código implementado na ESP32.

3.5 Software e firmware para Raspberry Pi 4

A Raspberry Pi requereu a instalação de um sistema operativo para a sua utilização. Neste caso foi usada a versão do sistema operativo Raspbian Buster, versão 10, lançada no dia 10 de julho de 2019. O Raspbian OS (Operating System) inclui diversas funcionalidades como o servidor VNC, que permite o controlo remoto da Raspberry através de outros dispositivos.

3.5.1 Atribuição de endereço IP estático

Por predefinição, os endereços de rede são atribuídos dinamicamente aos dispositivos que se conectam à rede através de um protocolo instalado no *router* chamado DHCP (Dynamic Host Configuration Protocol). Para além de um endereço de rede (IP), o DHCP faz a configuração automática da máscara de rede, *default gateway*, servidor(es) de DNS, domínio a que as máquinas pertencem, entre outros.

De forma que não fosse atribuído um endereço dinâmico à Raspberry Pi, foi necessário configurar um endereço estático. Desta forma é possível utilizar sempre o mesmo endereço para aceder aos serviços e aplicações localizados na Raspberry Pi. O primeiro passo foi descobrir o endereço de rede do *gateway* padrão, ou seja, do *router* da rede. Todos os dispositivos utilizam este endereço quando querem comunicar com o *router* e/ou aceder à Internet. O segundo passo foi descobrir o endereço do servidor de DNS. Os servidores DNS convertem os nomes de domínio em endereços IP (e.g., um dos endereços do domínio google.com é o 216.58.201.174). Na Figura 68 verifica-se que o endereço IP do *gateway* da rede e do servidor DNS é o 192.168.1.1, ou seja, o endereço IP do *router* Huawey HG8247Q.

```
pi@raspberrypi:~ $ route -ne
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 wlan0
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 wlan0
pi@raspberrypi:~ $ cat /etc/resolv.conf
# Generated by resolvconf
domain lan
nameserver 192.168.1.1
```

Figura 68. Endereço do *gateway* da rede e do servidor DNS (ambos no router).

O terceiro e último passo foi modificar o ficheiro dhcpcd.conf adicionando um endereço estático às interfaces de rede eth0 e wlan0, o endereço do *router* e o endereço do servidor DNS (Figura 69).

```
interface eth0
static ip_address=192.168.1.212

interface wlan0
static ip_address=192.168.1.213


static routers=192.168.1.1
static domain_name_servers=192.168.1.1
```

Figura 69. Modificação do ficheiro dhcpcd.conf para atribuição de endereço IP estático às interfaces de rede com fios (eth0) e sem fios (wlan0) da Raspberry Pi.

3.5.2 Broker MQTT

O *broker* escolhido para gerir as publicações e subscrições dos clientes MQTT foi o Eclipse Mosquitto [54] versão 1.6.3 (Figura 70). Trata-se de um software de código aberto que implementa as versões 5.0, 3.1.1 e 3.1 do protocolo MQTT, no entanto a versão utilizada e suportada pelos clientes MQTT (NodeMCU, Aplicação Android e Raspberry Pi) foi a 3.1.1. É adequado para uso em todos os dispositivos, desde computadores de baixa potência até servidores completos, sendo, portanto, adequado para uso na Raspberry Pi 4.

```
pi@raspberrypi:~ $ mosquitto -v
1563900647: mosquitto version 1.6.3 starting
1563900647: Using default config.
1563900647: Opening ipv4 listen socket on port 1883.
1563900647: Opening ipv6 listen socket on port 1883.
```

Figura 70. Eclipse Mosquitto versão 1.6.3.

Para a sua instalação, no Raspbian OS, foi necessário adicionar o repositório http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key e disponibilizá-lo para o APT (Advanced Packaging Tool). Após a instalação foram adicionadas credenciais (*username* e *password*) para proteger o sistema contra acessos indevidos.

3.5.3 Base de dados

Nesta seção é apresentado todo o software/firmware utilizado e desenvolvido para a implementação e administração da base de dados.

3.5.3.1 Servidor de base de dados

Para o armazenamento local de dados foi implementado um servidor de base de dados chamado MariaDB versão 10.3.15 (Figura 71), que é um dos servidores de base de dados *open source* mais populares do mundo, tendo sido desenvolvido pelos criadores do MySQL. Este software possui todos os comandos, interfaces, bibliotecas e APIs que existem no MySQL e ainda diversas novas funcionalidades [55].

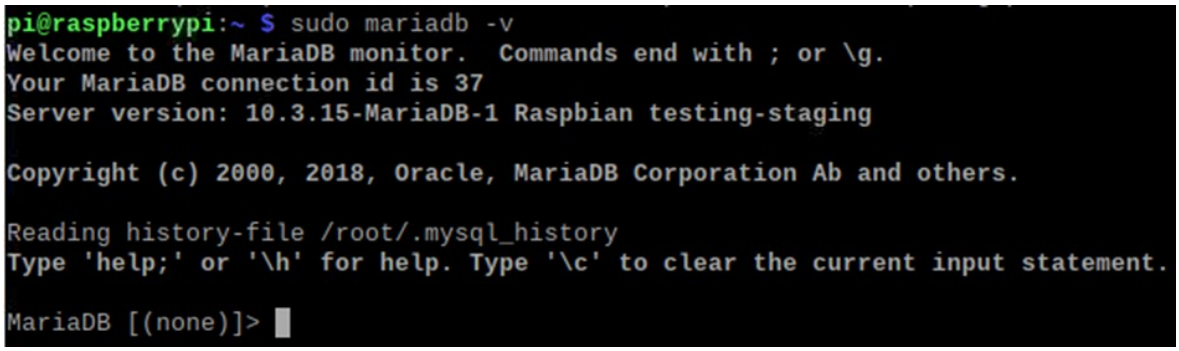


Figura 71. Versão utilizada do software MariaDB.

Após a instalação do servidor foi criada uma base de dados com o nome smarthomerf com cinco tabelas (Figura 72) para o armazenamento e gestão de dados de interesse da *smart home*. Nas tabelas devices_current, devices_voltage e device_status são armazenados os valores de corrente, tensão e o estado (ON/OFF) de cada dispositivo num determinado momento, respetivamente, e nas tabelas home_temperature e home_humidity são armazenados os valores de temperatura e humidade de cada divisão.

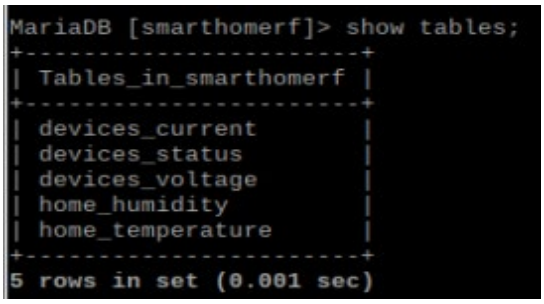


Figura 72. Tabelas da base de dados smarthomerf.

Ao utilizador Ruben foram garantidos todos os privilégios disponíveis sobre a base dados smarthomerf, sendo estes a capacidade de criar ou excluir novas tabelas ou bancos de dados, a capacidade de excluir, inserir ou atualizar linhas em tabelas e a capacidade de usar o comando SELECT para ler o banco de dados (Figura 73).

```
MariaDB [(none)]> SELECT User, Db, Host from mysql.db;
+------------+-------------+-----------+
| User       | Db          | Host      |
+------------+-------------+-----------+
| phpmyadmin | phpmyadmin  | localhost |
| Ruben      | smarthomerf | localhost |
+------------+-------------+-----------+
```

Figura 73. Lista de utilizadores, bases de dados e tipo de acesso.

3.5.3.2 Servidor HTTP

O servidor HTTP utilizado foi o Apache, versão 2.4.38 [56] (Figura 74) pois é um software livre, usado a nível mundial, fácil de configurar, com bom desempenho e segurança. Está visível e acessível à rede local e à Internet através do redireccionamento de portas no *router* e foi utilizado para permitir o acesso à base de dados através da aplicação Web phpMyAdmin (secção 3.5.3.3) e através da aplicação Android desenvolvida.

```
pi@raspberrypi:~ $ apache2 -v
Server version: Apache/2.4.38 (Raspbian)
Server built:   2019-04-07T18:15:40
```

Figura 74. Versão do servidor HTTP Apache.

O seu princípio geral de funcionamento é o seguinte: quando um dispositivo móvel/computador efetua um pedido de informação (pedido HTTP), o servidor Apache aciona um interpretador de código e processa a solicitação no seu módulo correspondente, recolhendo a informação desejada. Após obter a informação requerida, o servidor Apache conclui o processo, enviando a informação (resposta HTTP) num determinado formato (Figura 75).

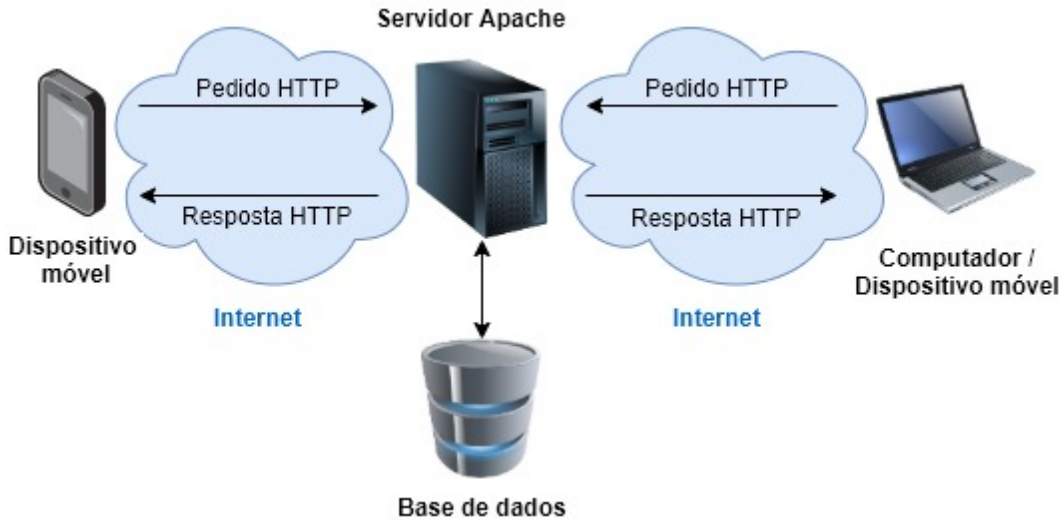


Figura 75. Transferência de dados entre o servidor e os dispositivos (Dispositivo móvel pela aplicação Android e Computador pela aplicação Web phpMyAdmin).

No caso do phpMyAdmin foi apenas realizada a sua instalação e configuração com o servidor Apache, mas para que a aplicação Android desenvolvida fosse capaz de obter dados da base de dados foi necessário implementar uma Web API própria, constituída por *scripts* em PHP. Dependendo da solicitação HTTP, o servidor interpreta o *script* correspondente, faz uma consulta especifica na base de dados e retorna os resultados no formato JSON (Figura 76). O formato JSON foi escolhido pois consome pouca largura de banda e apresenta os resultados de forma rápida e estruturada.

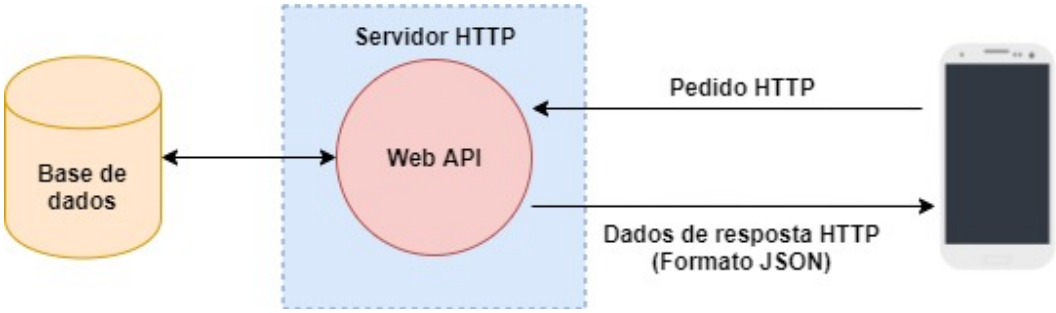


Figura 76. Diagrama da Web API implementada.

3.5.3.3 Administração da base de dados com o phpMyAdmin

Com o intuito de administrar a base de dados de uma forma mais intuitiva, fluída e organizada, foi utilizado o phpMyAdmin [57]. O phpMyAdmin é uma ferramenta de software *open source* escrita em PHP que permite lidar com a administração e gestão de bancos de dados por meio de uma interface gráfica com o utilizador através da Web (Figura 79).

O phpMyAdmin é acedido através do browser a partir de uma URL formada pelo endereço IP do servidor HTTP (caso o acesso seja feito na LAN é o endereço IP da Raspberry Pi, caso seja feito através da Internet é o endereço IP público do *router*) e o subdiretório phpMyAdmin, como se pode ver na Figura 77.

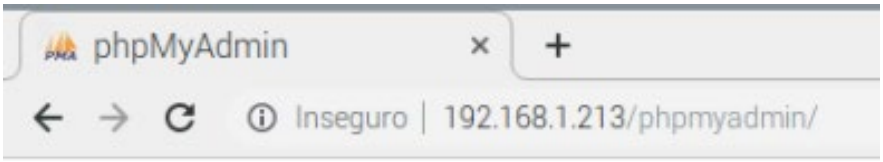


Figura 77. URL para aceder ao phpMyAdmin através do browser (acesso através da LAN).

A partir daí surge a página de "boas-vindas", apresentada na Figura 78, para a introdução das credenciais que permitem o acesso à base de dados pretendida.

Figura 78. Página Web do phpMyAdmin para introdução das credenciais de acesso.

Após o *login* surge uma página com informações sobre os servidores, definições gerais, base de dados disponíveis ao utilizador, configurações de aspeto, entre outros (Figura 79).

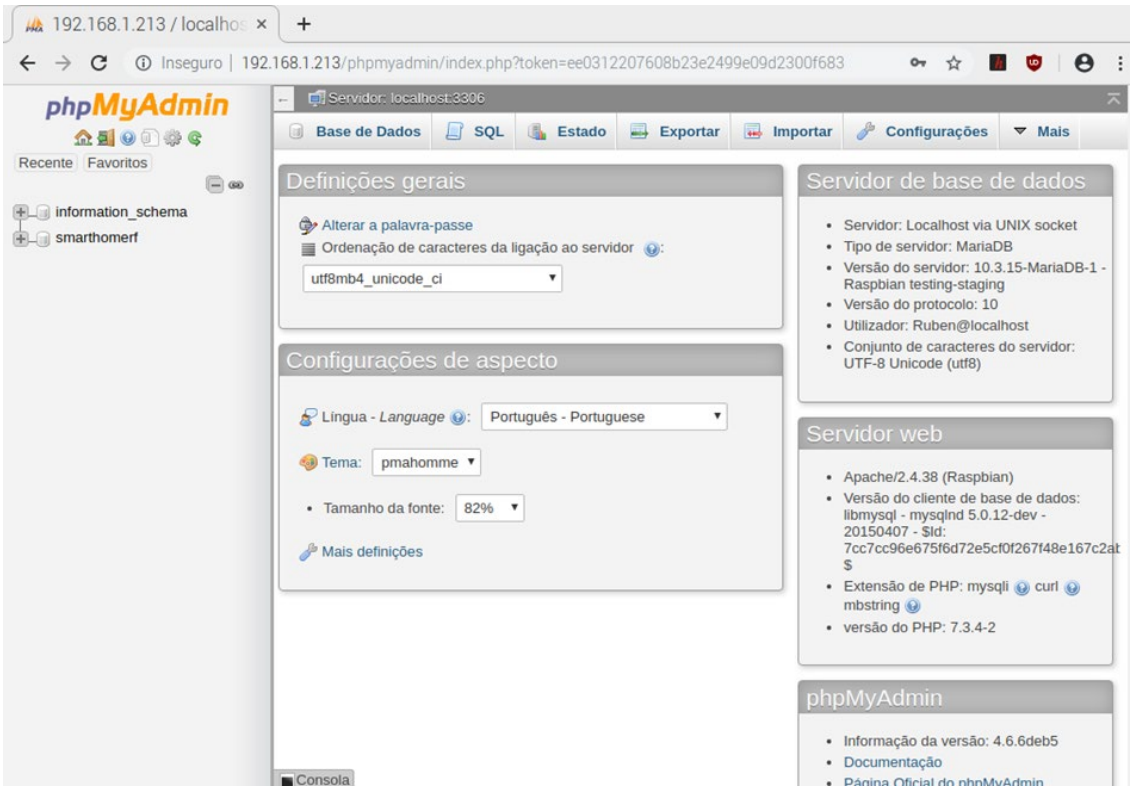


Figura 79. Interface gráfica do phpMyAdmin no browser.

O phpMyAdmin permite realizar consultas nas tabelas de forma a selecionarmos a informação que pretendemos através de *queries*. Por exemplo, na Figura 80, foi realizada uma consulta para saber os últimos 10 valores de corrente consumida pelo ar condicionado do quarto 1 da tabela devices_current referenciada na seção 3.5.3.1. Os campos/colunas das tabelas têm relação com os tópicos hierárquicos do MQTT definidos (e.g., Figura 47) sendo o tópico myhome/firstfloor/bedroom1/aircond/current responsável pelos valores de corrente (*current*) do ar condicionado (*device* = *aircond*) do quarto 1 (*room=bedroom*1) localizado no primeiro andar (floor = *firstfloor*) da casa.

Figura 80. Consulta na tabela devices_current com o phpMyAdmin.

Com os resultados obtidos na consulta é possível obter gráficos como o da Figura 81 (em que as legendas são personalizáveis) e/ou exportar os dados num determinado formato (e.g., PDF, CSV, SQL, JSON, XML).

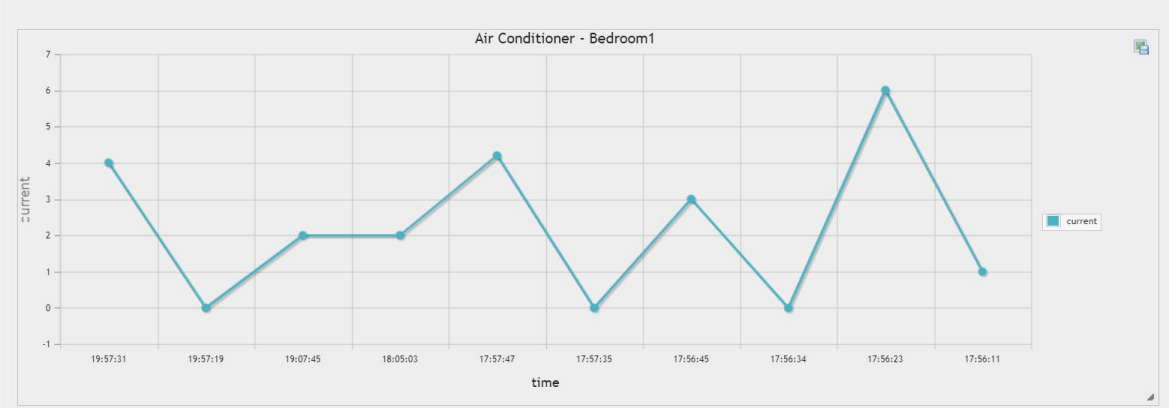


Figura 81. Gráfico no phpMyAdmin com valores da consulta efetuada.

3.5.3.4 Inserção automática de dados na base de dados

No sentido de obtenção e inserção de dados de forma automática foi necessária a elaboração de um *script* em Python 3. Nesse *script* foi implementado um cliente MQTT 3.1.1 através da importação da biblioteca paho-mqtt 1.4.0 [58]. Esta biblioteca fornece uma classe de cliente que permite que as aplicações se conectem a um *broker* MQTT para publicar mensagens, subscrever tópicos e receber mensagens publicadas. Além disso foi necessária uma interface para estabelecer a conexão ao servidor de base de dados MariaDB. Para tal foi utilizado o módulo PyMySQL [59], pois contém uma biblioteca cliente MySQL compatível com MariaDB e com Python 3. A conexão à base de dados é sempre terminada após a inserção da mensagem na tabela para libertar recursos e garantir uma maior segurança. O fluxograma apresentado na Figura 82 permite um melhor entendimento das etapas que são realizadas neste *script* e da sua ordem de execução.

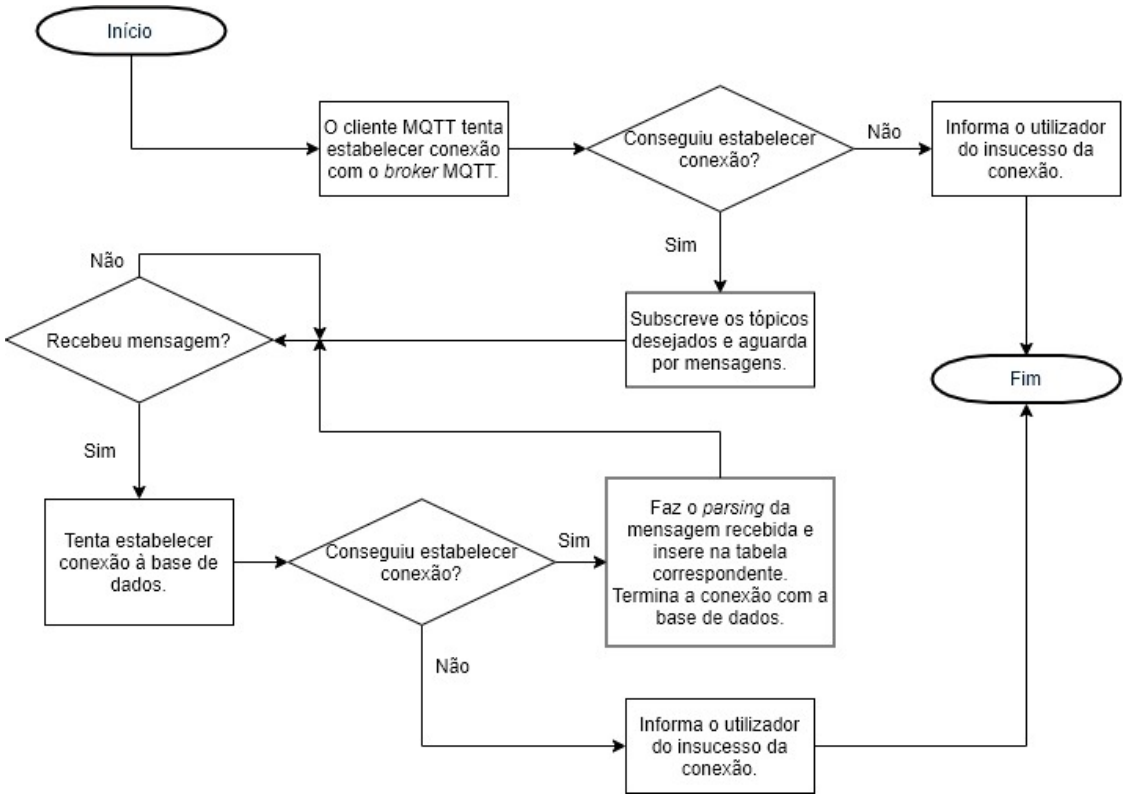


Figura 82. Fluxograma do *script* que insere dados na base de dados.

Inicialmente o cliente tenta estabelecer conexão com o *broker* MQTT mas caso isso não seja possível o *script* pára a execução do *script* e informa o utilizador. Se a conexão se realizar com sucesso o cliente MQTT subscreve a todos os tópicos da *smart home* e aguarda por mensagens. Sempre que surge uma nova mensagem é estabelecida uma conexão à base de dados antes do *parsing* pois não vale a pena gastar recursos e tempo se esta não se realizar com sucesso. Portanto após se verificar o sucesso da

conexão à base de dados é realizado o *parsing* que consiste primeiro na identificação da tabela correspondente à informação e à separação do conteúdo da mensagem pelos campos da tabela correspondente. Após a informação ser inserida na tabela a conexão com a base de dados é desfeita e o *script* retorna ao ponto onde aguarda por novas mensagens.

3.5.4 Gateway BLE/Wi-Fi

Com o objetivo de incluir dispositivos BLE nesta plataforma foi desenvolvido um *script* em Python 3 de forma a realizar um *gateway* BLE/Wi-Fi. Neste sentido, foram implementados na Raspberry Pi um cliente BLE para receber os dados de um servidor BLE, e um cliente MQTT para publicar os dados recebidos num determinado tópico.

Para a implementação do cliente MQTT foi utilizada a mesma biblioteca que foi referida na secção anterior [58], e para o cliente BLE foi utilizada a interface Bluepy [60], que permite acesso a dispositivos BLE a partir de código Python. Apesar da Raspberry Pi 4 ter BLE integrado, obteve-se uma conexão mais estável com a utilização de um micro-adaptador Bluetooth v4 via porta USB.

O *script* começa por fazer a importação de todas as bibliotecas necessárias e pela implementação do cliente MQTT e das suas funções de *callback.* De seguida, o cliente MQTT estabelece conexão com o Mosquitto Broker através da introdução das credenciais necessárias: *username* e *password*, porta de comunicação e endereço IP. Sendo estabelecida a conexão ao *broker*, passa-se a estabelecer conexão ao dispositivo BLE que que contém os dados pretendidos através do seu endereço MAC (Media Access Control). A seguir, é ativado o método de notificação, ou seja, o cliente BLE recebe dados de uma determinada característica (conhecida e especificada) à medida que os dados são enviados pelo servidor BLE. Sempre que uma notificação é recebida é chamado um método no qual é possível publicar os dados num determinado tópico através do cliente MQTT. Na Figura 83 está representado um fluxograma que ilustra as etapas descritas anteriormente, assim como a sua sequência de execução.

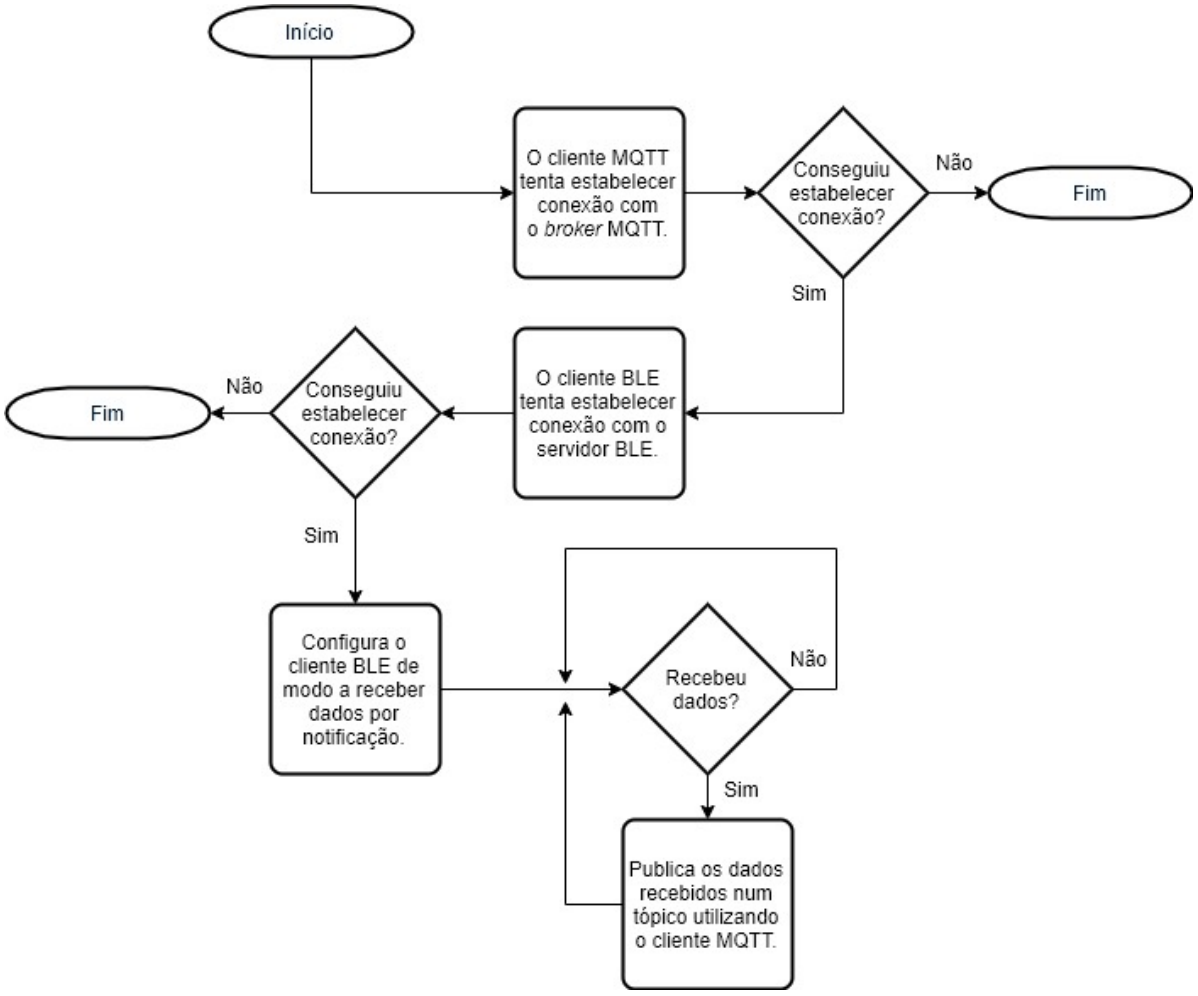


Figura 83. Fluxograma do *script* que implementa o *gateway* BLE/Wi-Fi.

3.5.5 Gestão da corrente disponível na casa

Visto que todas as casas têm uma certa potência contratada, e que muitas vezes esta não é suficiente para todos os dispositivos/equipamentos que pretendemos usar, torna-se relevante haver um sistema que faça a sua gestão de forma a evitar que a corrente consumida ($I_{consumida}$) seja superior à corrente máxima permitida (I_{max}) e o consequente disparo do disjuntor na habituação [61].

$$I_{consumida} \leq I_{max} \tag{4}$$

Sendo assim, foi elaborado um *script* em Python que verifica, constantemente, se a corrente consumida está dentro do limite. A corrente consumida é obtida periodicamente (x vezes por segundo) através de um sensor de corrente conectado a uma NodeMCU localizada no quadro elétrico da casa e que corresponde à corrente utilizada pelos equipamentos da casa e pelo sistema de carregamento do veículo elétrico.

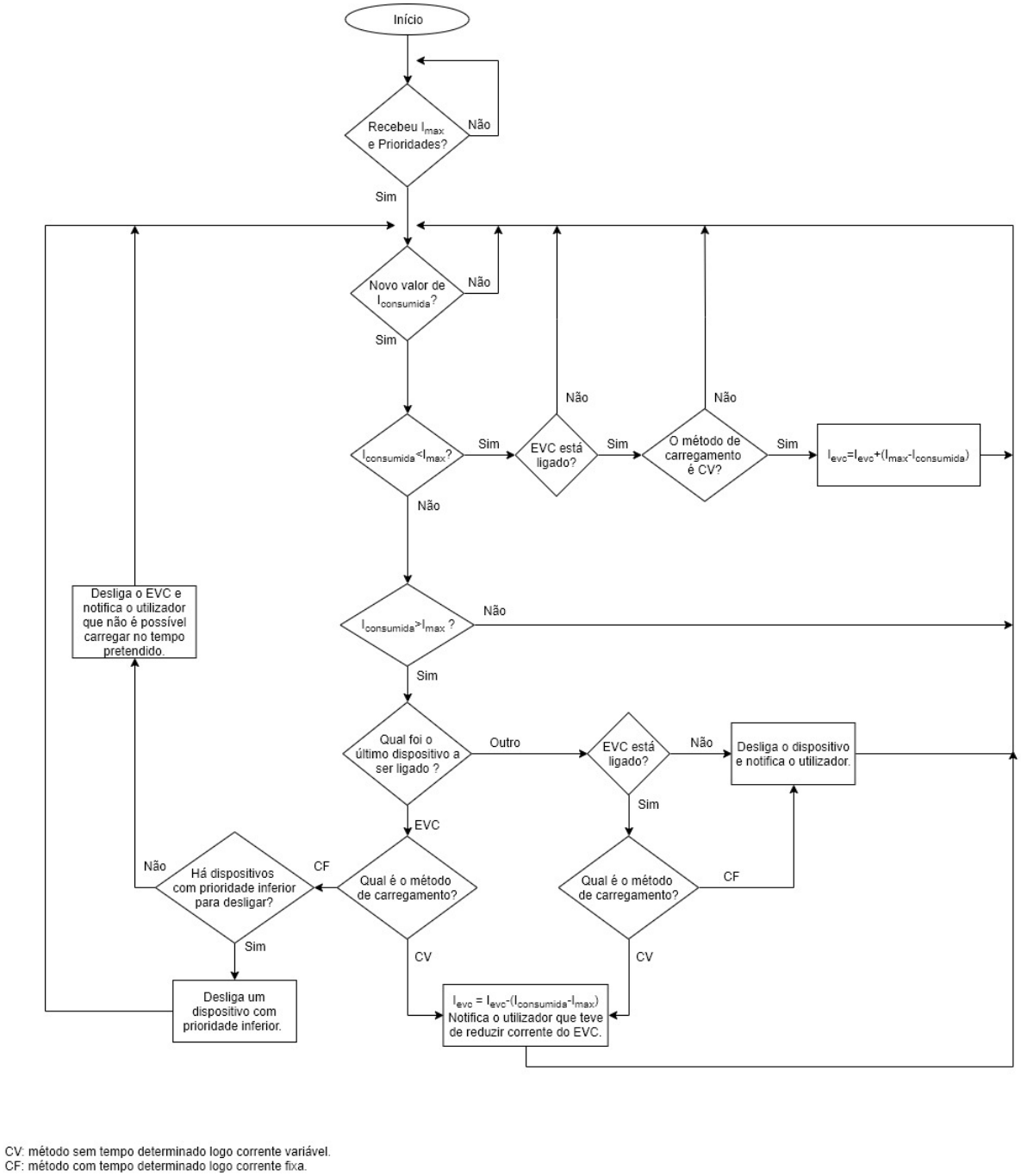
$$I_{consumida} = I_{evc} + I_{equipamentos} \tag{5}$$

O sistema de carregamento do veículo elétrico tem dois métodos de carregamento: CV (corrente variável) e CF (corrente fixa). O método CV é adotado quando o utilizador não define um tempo de carregamento determinado. Neste caso, a corrente de carregamento vai-se adaptando à corrente disponível. O método CF é adotado quando o utilizador pretende que a bateria carregue num determinado intervalo de tempo. Neste caso, a corrente de carregamento é fixa e é calculada através da equação (6), em que C_T é a capacidade total da bateria (kWh), C_R a capacidade atual da bateria (kWh), V_c a tensão de carregamento (Volts) e Δt o tempo de carregamento pretendido (horas). Nesta circunstância é a corrente utilizada pelos equipamentos da casa que se adapta à corrente disponível.

$$I_{evc} = \frac{(C_T - C_R)}{V_c \times \Delta t} \times 1000 \qquad (6)$$

O algoritmo implementado está representado na Figura 84 e consiste no seguinte: sempre que é recebido um novo valor de corrente consumida é verificado se é superior ou inferior ao limite (I_{max}).

- Caso seja inferior, é verificado se o sistema de carregamento do carro elétrico está ligado. Se estiver e o método de carregamento for CV é aumentada a corrente de carregamento de forma a aproveitar toda a corrente disponível, para possibilitar um carregamento mais rápido.
- Caso seja superior, é verificado qual o equipamento que causou esse aumento consultando o último dispositivo que foi ligado:
  - Se for o sistema de carregamento do veículo elétrico e o método for CV a corrente de carregamento é reduzida, no entanto, se o método de carregamento for CF a corrente é fixa e, portanto, são desligados dispositivos com menor prioridade até que a corrente consumida seja inferior ao limite. Pode acontecer que mesmo desligando todos os equipamentos com prioridade inferior ao sistema de carregamento, a corrente continue superior ao limite levando ao desligamento do sistema de carregamento. O desligamento do sistema de carregamento faz com que o utilizador seja "obrigado" a colocar outro tempo ou a mudar o método de carregamento.
  - Se for outro dispositivo é verificado se o sistema de carregamento está ligado e se o método de carregamento (CV) permite que lhe seja reduzida a corrente que o equipamento necessita para permanecer ligado. Caso isso não seja possível o último dispositivo ligado é desligado e o utilizador notificado.



CV: método sem tempo determinado logo corrente variável.
CF: método com tempo determinado logo corrente fixa.

Figura 84. Fluxograma da gestão da corrente da casa.

3.6 Mapeamento de portas no router

O *router* Huawei HG8247Q disponibilizado pela ISP Vodafone foi utilizado nesta dissertação para gerir a comunicação entre dispositivos da mesma rede (rede local) ou de redes diferentes (Internet). Para que o utilizador fosse capaz de monitorizar e controlar a sua casa através da Internet, ou seja, fora da rede local foi necessário utilizar o mecanismo *port forwarding* que basicamente permite que o tráfego de

entrada, proveniente da Internet, chegue a uma determinada aplicação ou serviço em execução, num dado dispositivo local. Portanto, através da opção mapeamento de portas IPv4 foram configuradas portas para o *broker* MQTT, servidor HTTP, servidor VNC e servidor base de dados e, como se pode verificar na Figura 85, o endereço IP do anfitrião interno é sempre o mesmo pois corresponde ao endereço da Raspberry Pi onde são executadas estas aplicações/serviços.

Como foi referido na secção 3.5.3.3 para aceder, nesse caso, ao servidor HTTP é utilizado o endereço público do *router.* O endereço público do *router* nesta circunstância é dinâmico pois a ISP Vodafone apenas fornece um endereço estático a empresas com custos atribuídos, no entanto, o endereço é raramente alterado mesmo se o *router* for desligado.

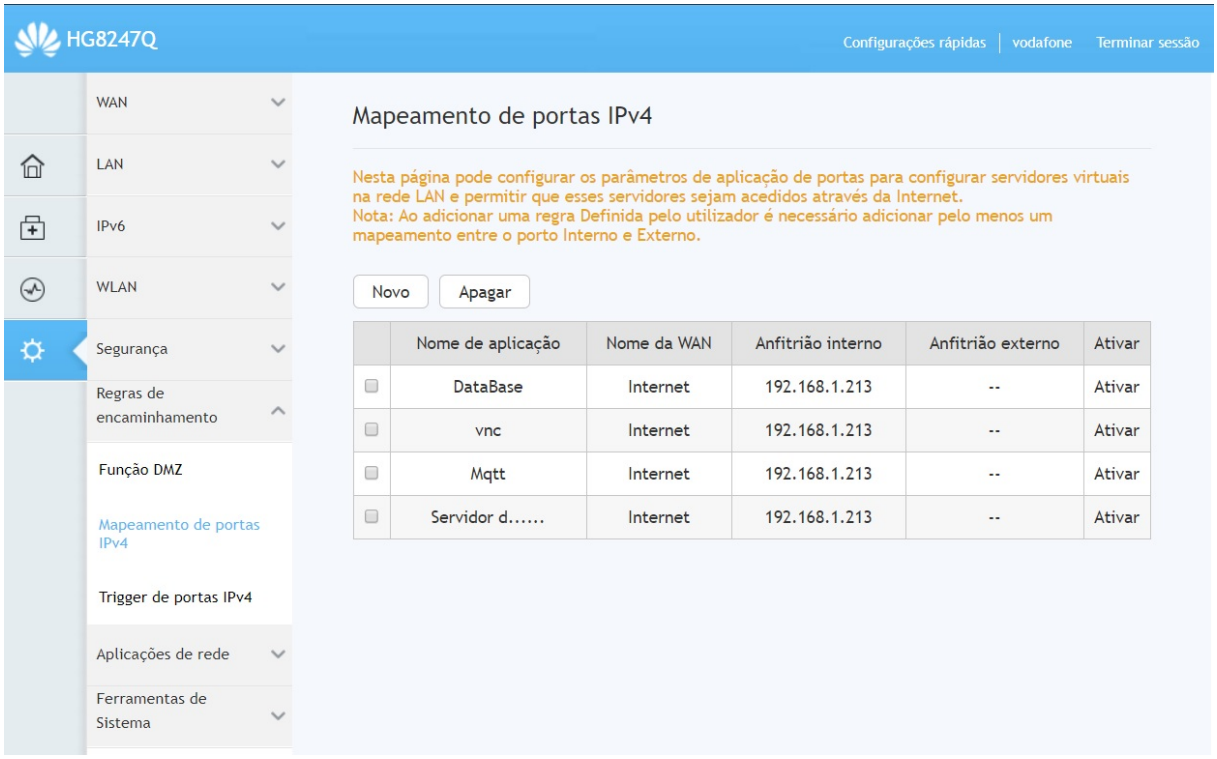


Figura 85. Mapeamento de portas no *router* HG8247Q da Vodafone.

3.7 Tomada inteligente

Foi desenvolvida uma tomada inteligente que permite controlar o estado de um equipamento e obter os seus valores eficazes de corrente e tensão através de um cliente MQTT no módulo Wi-Fi. Para que isso fosse possível foi necessário utilizar: um sensor de corrente (3.2.9); um sensor de tensão (3.2.8); um relé (3.2.7); um conversor AC-DC; um módulo Wi-Fi (3.2.1); um multiplexador analógico (3.2.10); uma tomada macho e fêmea. O conversor AC-DC converte os 240 V (AC) em 5 V (DC) para alimentar o

módulo Wi-Fi (NodeMCU) e o relé e o multiplexador. Os sensores são alimentados pela NodeMCU a 3,3 V pois é a tensão de referência do seu ADC. O multiplexador foi utilizado para possibilitar a conexão dos dois sensores a um único ADC da NodeMCU e o relé permite desligar/ligar o equipamento da rede. Na Figura 86 está representado o diagrama do circuito da tomada inteligente. Na secção 4.3.2 encontra-se o protótipo desenvolvido.

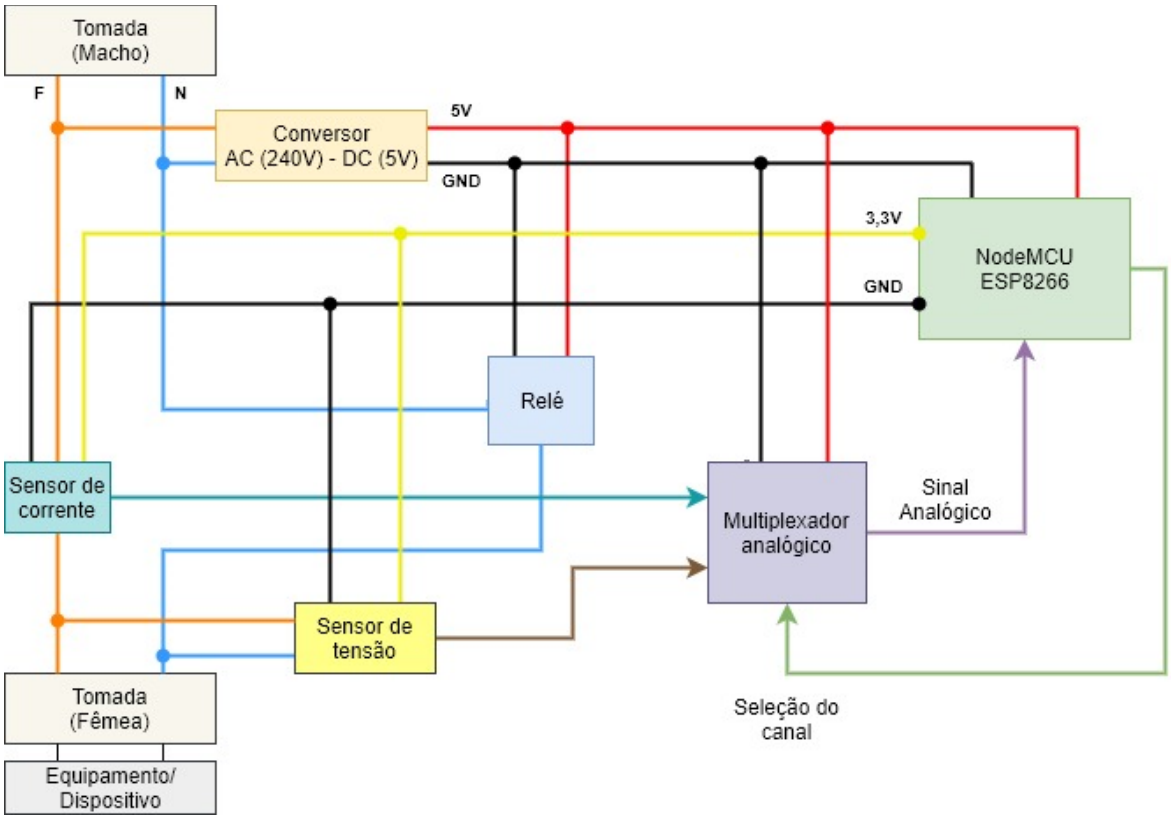


Figura 86. Diagrama do circuito da tomada inteligente.

3.8 Aplicação Android

Nesta secção é abordada a aplicação Android desenvolvida que permite monitorizar e controlar os equipamentos e a energia de uma casa com três pisos/andares: garagem, rés-do-chão e primeiro andar. O ambiente de software utilizado para o seu desenvolvimento foi o Android Studio 3.5 [20].

3.8.1 Dependências

No Android Studio, as dependências permitem incluir bibliotecas externas, arquivos JAR locais ou outros módulos de bibliotecas no nosso projeto. Algumas das dependências mais relevantes e necessárias foram a MPAndroidChart:v3.1.0 [62] para desenhar gráficos, a gson:2.8.5 [63] para utilizar

objetos JSON e a org.eclipse.paho.android.service:1.0.2 [64] para implementar clientes MQTT. Na Figura 87 estão todas as dependências utilizadas para o desenvolvimento da aplicação.

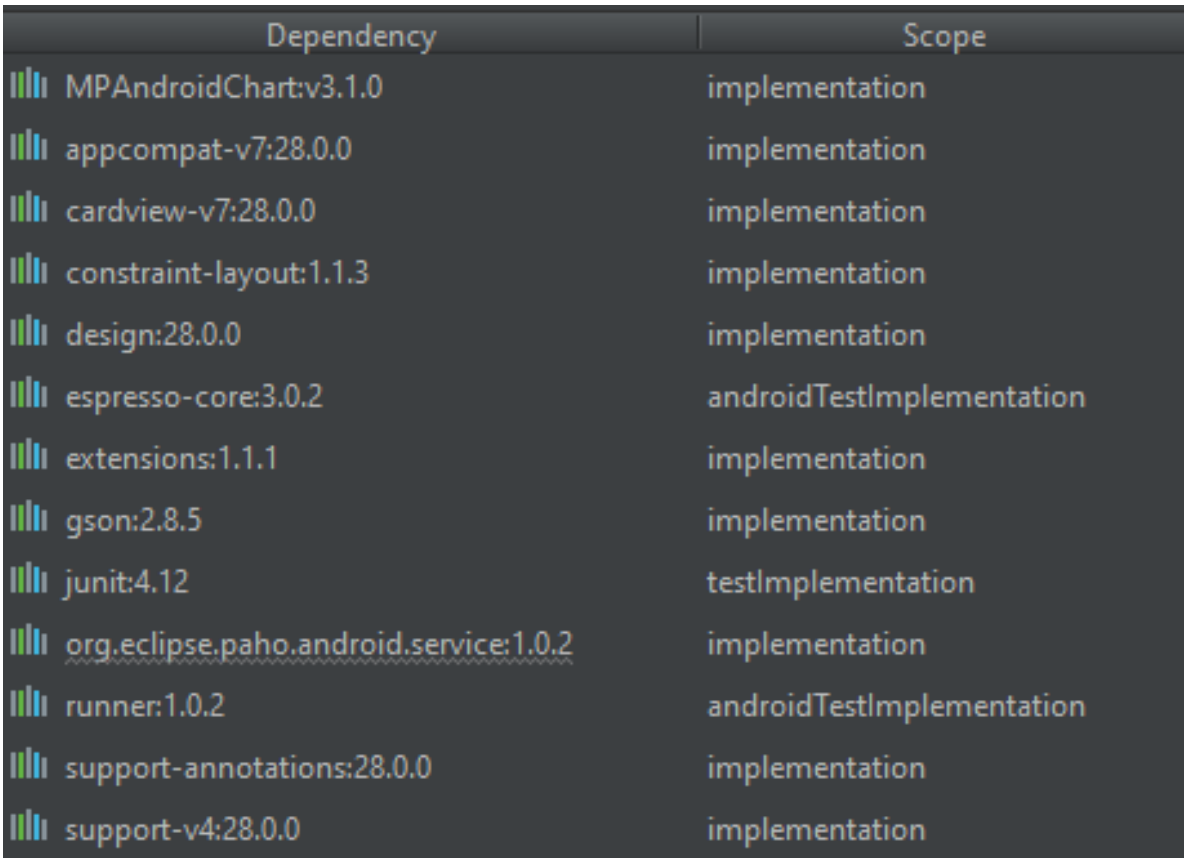


Figura 87. Dependências utilizadas no projeto Android.

3.8.2 Fluxograma geral da aplicação

A aplicação começa por apresentar um Splash Screen (Figura 92) que age como um ecrã de boas-vindas com o logótipo da aplicação. Após um certo período de tempo o utilizador é direcionado para uma Activity Login (Figura 92) para inserção de dados, caso seja necessário, ou para a Main Activity (Figura 94). A Main Activity como o nome indica é a Activity principal e permite aceder às restantes Activities. O fluxograma geral da aplicação encontra-se representado na Figura 88 para uma melhor compreensão.

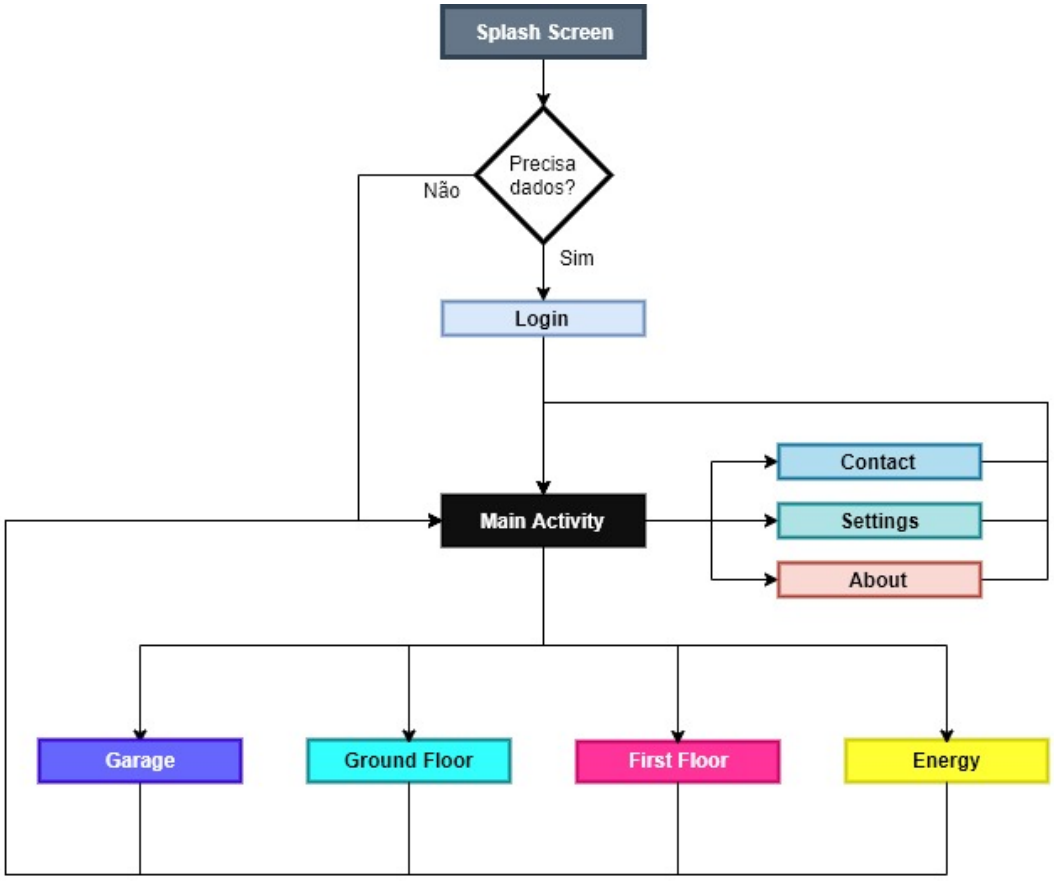


Figura 88. Fluxograma geral da aplicação Android.

3.8.3 Componentes principais

Nesta secção são apresentados os componentes principais e indispensáveis para o funcionamento da aplicação desenvolvida.

3.8.3.1 Service "MyService"

O MyService é um Service de primeiro plano (*Foreground Service*) que continua em execução, mesmo quando o utilizador não interage com a aplicação. Nesse sentido foi implementado um cliente MQTT, através da classe MQTT_connection, que estabelece e mantém a conexão com o *broker* MQTT e subscreve aos tópicos da *smart home*. Sempre que surge uma nova mensagem MQTT é verificado qual o tópico correspondente para armazenar o seu valor numa variável e além disso, é enviado um *broadcast* para notificar os outros componentes (e.g., Fragments) de que foram recebidos novos dados. Na Figura 89 está representado um fluxograma do Service e na Figura 90 está representada a relação entre o Service MyService e a classe MQTT_connection.

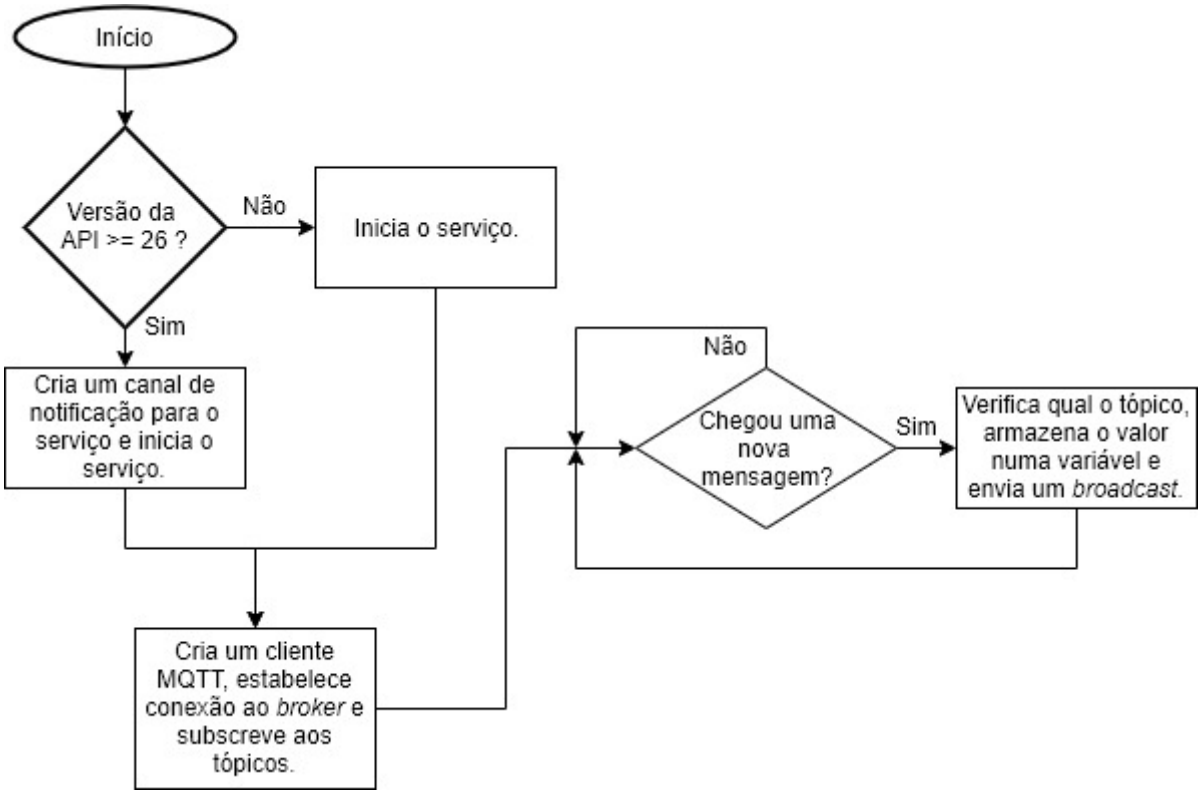


Figura 89. Fluxograma do Service "MyService".

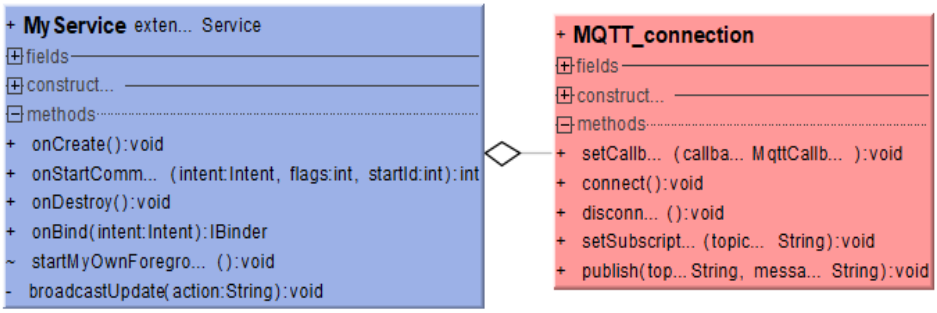


Figura 90. Diagrama UML que representa a relação entre o Service "MyService" e a classe MQTT_connection.

A criação de um Service de primeiro plano requereu a exibição permanente de uma notificação (Figura 91) e a criação de canais de notificação para APIs iguais ou superiores ao nível 26 (Android Oreo).

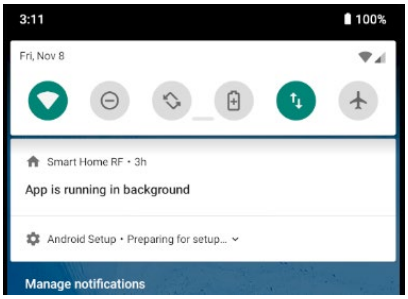


Figura 91. Notificação permanente do MyService.

3.8.3.2 Activity Login

A Activity Login (Figura 92) é onde são inseridos todos os dados necessários para o funcionamento correto da aplicação. O utilizador começa por introduzir o seu nome e email, as credenciais necessárias para se conectar com o *broker* MQTT pretendido, a potência contratada da casa e o tipo de sistema de energia (monofásico ou trifásico). Por último, considerando que o utilizador tem um veículo elétrico, existe um campo para a inserção da capacidade da bateria do veículo.

Figura 92. Aplicação Android: Splash Screen (esquerda) e Activity Login (direita).

Após a introdução dos dados é utilizada a interface SharedPreferences para guardar e modificar os dados de preferência do utilizador. Cada valor armazenado apresenta-se sob formato chave-valor, ou seja, cada preferência armazenada possui uma identificação e associada a ela está um valor (Figura 93).

```
Ed.putString("username", username.getText().toString());
Ed.putString("password", password.getText().toString());
Ed.putString("mqtthost", host.getText().toString());
Ed.putString("power", power.getText().toString());
Ed.putString("name",name.getText().toString());
Ed.putString("email",email.getText().toString());
Ed.putString("EVBCapacity",carbattery.getText().toString());
```

Figura 93. Armazenamento de dados no formato chave-valor através da interface SharedPreferences.

3.8.3.3 Activity "Main Activity"

A Main Activity, além de ser uma espécie de menu principal para aceder a outras Activities, é também responsável por iniciar o Service MyService (caso ainda não esteja a correr). Esta Activity é composta essencialmente por 4 botões (Garage, Ground Floor, First Floor e Energy) e um *navigation drawer* que permite visualizar o perfil do utilizador, contactar o desenvolvedor e obter informações sobre a aplicação (Figura 94).

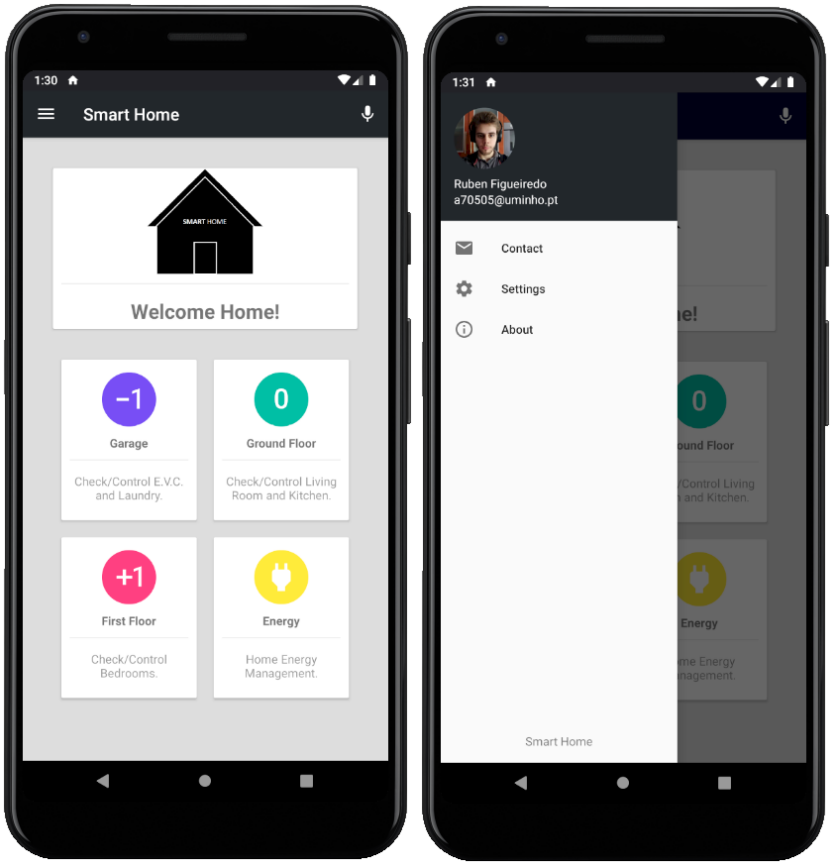


Figura 94. Aplicação Android: Main Activity (esquerda) e Navigation drawer (direita).

3.8.3.4 Activity Garage

A Activity Garage é constituída por 2 Fragments: EVC e Laundry Room (Figura 95). O EVC permite controlar e monitorizar o sistema de carregamento da bateria do veículo elétrico e o Laundry Room ligar e/ou desligar equipamentos da lavandaria.

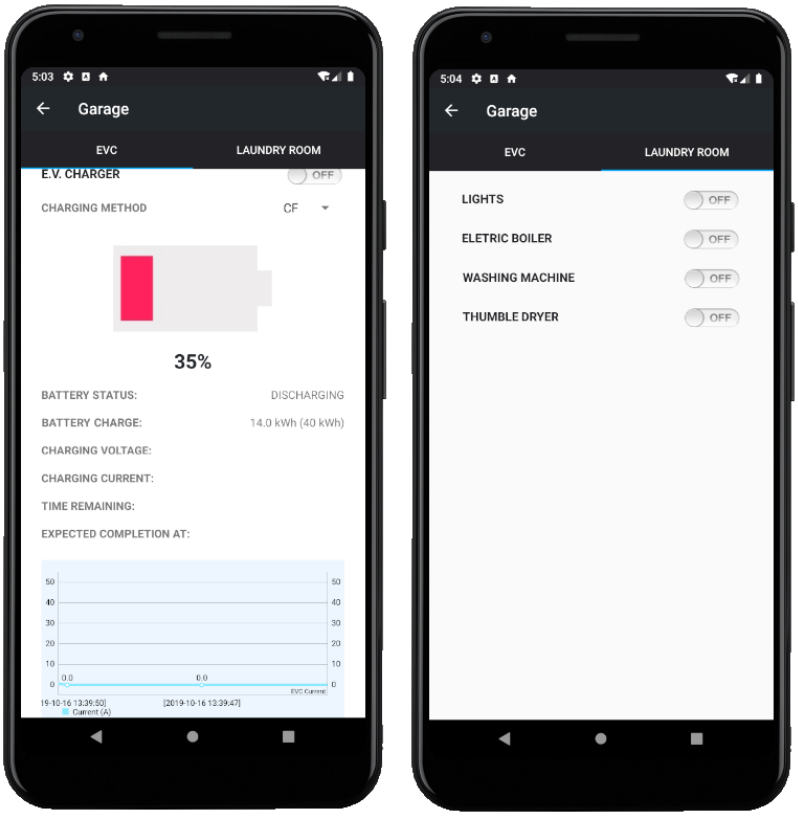


Figura 95. Aplicação Android: Activity Garage (Fragment EVC e Fragment Laundry Room).

No Fragment EVC é possível desligar e/ou ligar o carregamento da bateria do veículo elétrico, definir o método de carregamento (CV ou CF, conforme explicado na seção 3.5.5), verificar a percentagem/capacidade a que se encontra a bateria, o valor da tensão e da corrente de carregamento e obter uma estimativa do tempo e do momento em que a bateria estará a 100% da sua capacidade. Para tal, recorre-se à equação (7) em que T é o tempo em horas, C_T a capacidade total da bateria, C_R a capacidade atual da bateria, V_C a tensão de carregamento e I_{evc} a corrente de carregamento. O gráfico localizado na parte inferior permite uma visualização da variação do valor da corrente de carregamento ao longo do tempo (o valor mais recente é o ponto mais à esquerda). Sempre que surge um novo valor de corrente (no tópico correspondente) é feito um pedido HTTP (seção 3.5.3.2, Figura 76) que retorna os últimos valores na base de dados, atualizando, portanto, o gráfico.

$$T = \frac{(C_T - C_R)}{V_c \times I_{evc}} \times 1000 \qquad (7)$$

O método de carregamento CF requer ao utilizador a introdução de um tempo de carregamento para o cálculo da corrente necessária enquanto que o método CV utiliza a corrente disponível na casa, calculada a partir da diferença entre a corrente máxima e a corrente que está a ser utilizada pelos outros equipamentos num dado momento.

3.8.3.5 Activity Ground Floor

Nesta Activity é possível controlar os equipamentos e monitorizar a temperatura e a humidade de cada divisão do piso 0 (rés do chão). É constituída por dois Fragments, um que corresponde à sala de estar (Living Room) e outro à cozinha (Kitchen), conforme representado na Figura 96.

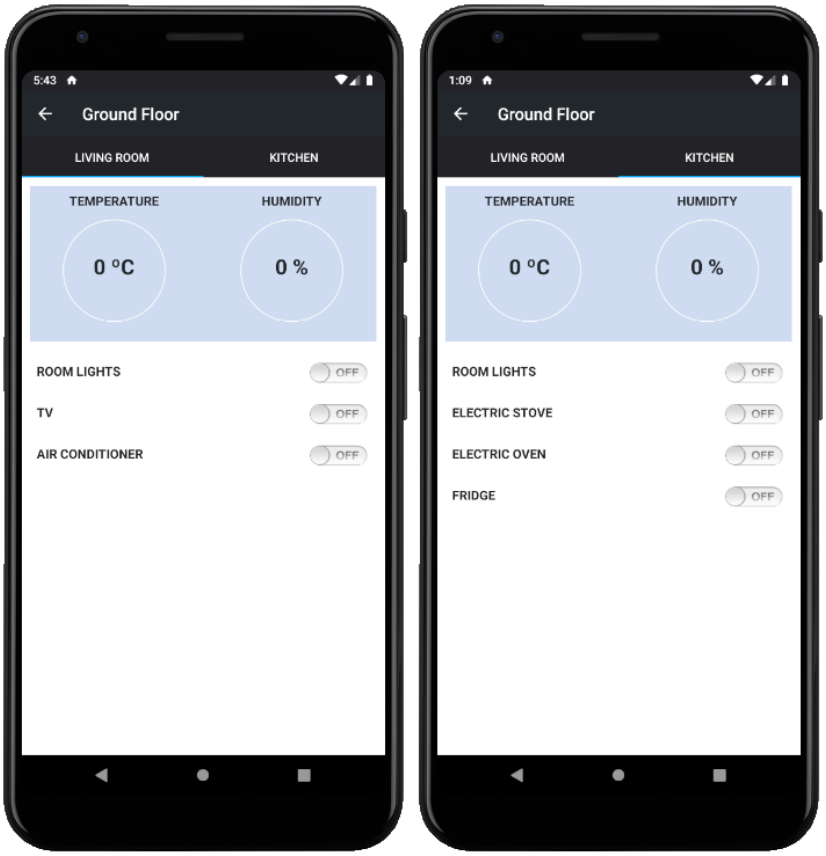


Figura 96. Aplicação Android: Activity Ground Floor (Fragment Living Room e Fragment Kitchen).

3.8.3.6 Activity First Floor

A Activity First Floor é onde se encontram os quartos localizados no primeiro andar da casa. É composta por dois Fragments que permitem controlar os equipamentos e monitorizar a temperatura e a humidade de cada quarto (Figura 97).

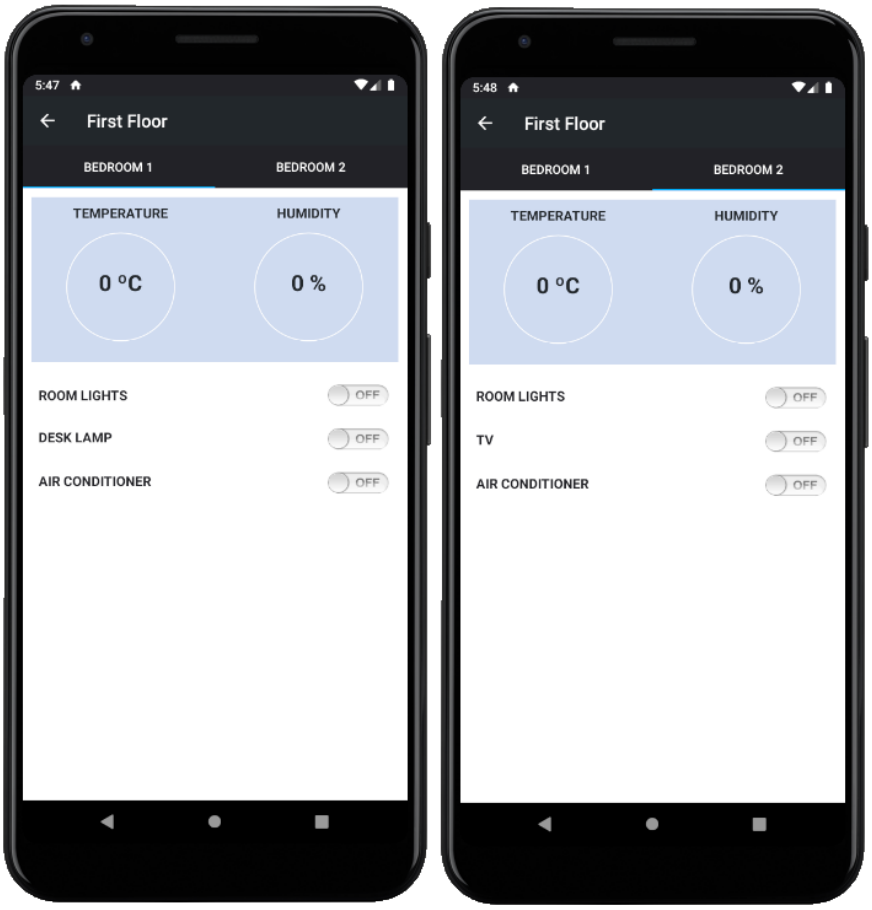


Figura 97. Aplicação Android: Activity First Floor (Fragment Bedroom 1 e Fragment Bedroom 2).

3.8.3.7 Activity Energy

Esta Activity foi desenvolvida para abranger os tópicos relacionados com a energia da casa. É composta por três Fragments: Current Distribution, Home Consumption e Energy Priority over EVC (Figura 98).

No Fragment Current Distribution é possível monitorizar em tempo real a percentagem de corrente que cada divisão da casa está a utilizar, a corrente total que está a ser consumida e a corrente máxima permitida. Os valores são atualizados sempre que é recebido um *broadcast* ou seja, surge um novo valor nos tópicos relacionados.

No Fragment Home Consumption é representado o consumo da casa através de um gráfico com os valores do sensor de corrente e tensão do quadro elétrico. A obtenção destes valores é feita através do servidor HTTP e do método referenciado na secção 3.5.3.2.

No terceiro e último Fragment, Energy Priority over EVC, são definidas as prioridades de alguns dispositivos em relação ao carregamento da bateria do veículo elétrico. Se a prioridade de um dispositivo

for LOWER, ou seja, mais baixa, o dispositivo pode ser desligado caso não haja corrente suficiente na casa para o sistema de carregamento.

Os gráficos (*pie chart* e *line chart*) da Figura 98 não são reais servindo apenas para ilustração neste caso, no entanto, são apresentados na secção 4.2.3.

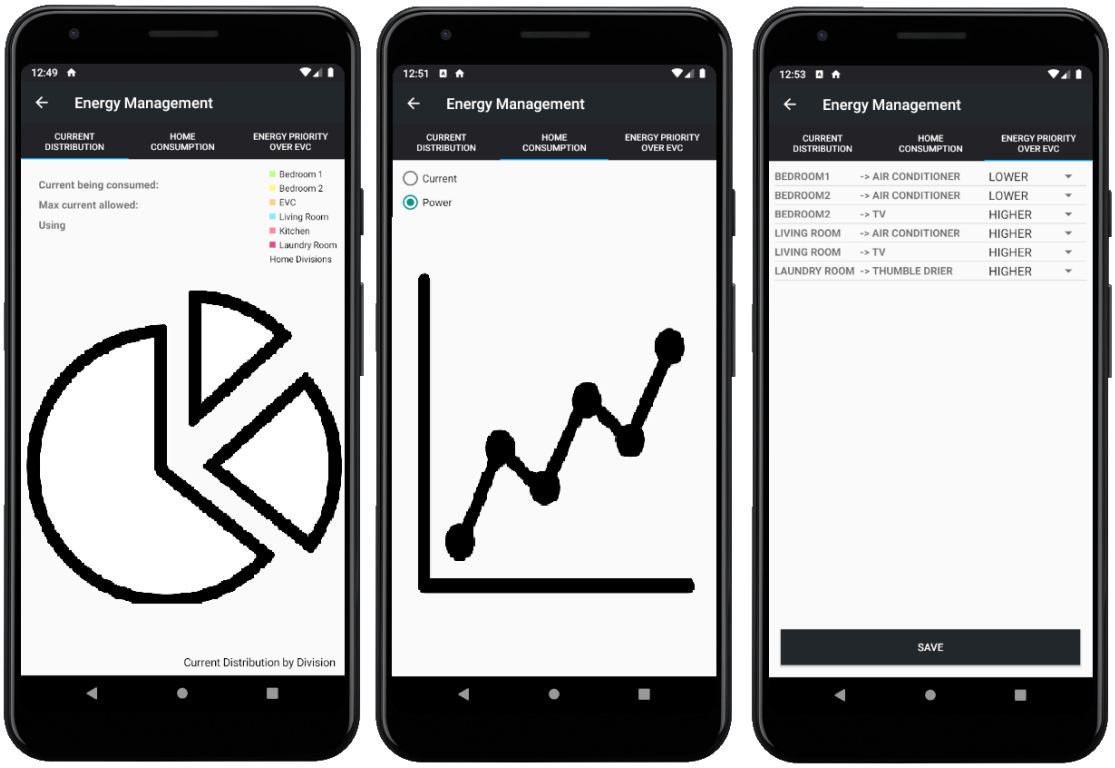


Figura 98. Aplicação Android: Activity Energy (Fragment Current Distribution, Fragment Home Consumption e Fragment Energy Priority Over EVC).

4.Testes e resultados

No presente capítulo são apresentados os testes realizados e o protótipo desenvolvido. Nos testes de atraso e fiabilidade não foi considerado o QoS 0, uma vez que no sistema implementado não podem ocorrer perda de pacotes. É importante também referir que os pacotes de dados trocados entre a Raspberry Pi e os módulos Wi-Fi passam sempre pelo *router* da Internet, como representado na Figura 21.

4.1 Testes de atraso e fiabilidade

Os testes avaliam o atraso e a fiabilidade da comunicação entre um nó sensor e um nó atuador utilizando o método representado na Figura 99. Foram realizados testes para dois tipos diferentes de nós sensores: um módulo BLE (dispositivo periférico), representado pela placa ESP32 da seção 3.2.2; e um módulo Wi-Fi, representado pela placa NodeMCU da seção 3.2.1. A principal vantagem de usar um módulo BLE é o menor consumo de energia, no entanto, no cenário de aplicação considerado, tanto o nó sensor como o nó atuador não têm restrição de energia, pois estão ligados à instalação elétrica. Sendo assim, o uso de um módulo Wi-Fi é uma alternativa viável e, por isso, em todos os testes, o nó atuador foi uma placa NodeMCU.

Em cada teste, foram enviadas 1000 amostras do nó sensor para o nó atuador, sendo que cada amostra só era enviada após a receção da anterior no destino. O atraso medido corresponde ao tempo decorrido desde que os dados são enviados pelo nó sensor (hora de início) até serem recebidos no nó atuador (hora de fim). Como são dispositivos diferentes, não compartilham o mesmo relógio, portanto, para permitir a medição do atraso, cada vez que o nó atuador recebesse um pacote de dados, era enviado um sinal de interrupção ao nó sensor, permitindo que esse dispositivo calculasse a diferença entre a hora de fim e a hora de início usando o mesmo relógio (Figura 99). Para avaliar a fiabilidade da comunicação foi incluído um número de sequência dentro de cada pacote de dados enviado pela fonte, e verificado no destino.

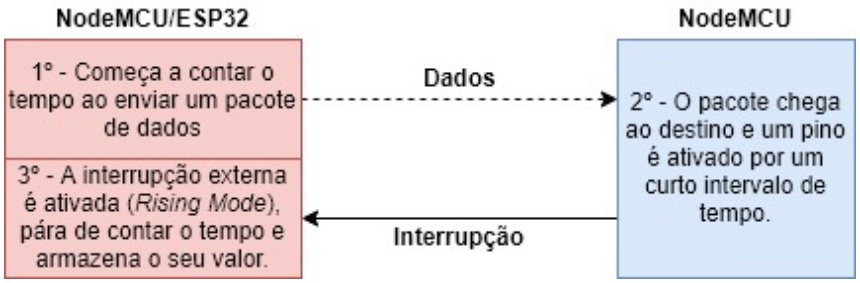


Figura 99. Método utilizado para medir os tempos de atraso.

4.1.1 Nó sensor: NodeMCU

Nesta seção são apresentados os testes realizados com a NodeMCU (Wi-Fi) como nó sensor e como nó atuador para QoS 1 e 2. O percurso dos pacotes é desde o cliente MQTT (publicador) no nó sensor até ao *broker* MQTT no *hub/gateway* (Raspberry Pi) e do *broker* até ao cliente MQTT (subscritor) no nó atuador.

4.1.1.1 QoS 1

Na Figura 100 estão representados, através de um histograma, os tempos de atraso obtidos para o QoS 1 após o envio de 1000 amostras. O atraso de 92% das amostras está compreendido entre 4 e 16 ms e apenas 3,6% têm atraso superior a 24 ms.

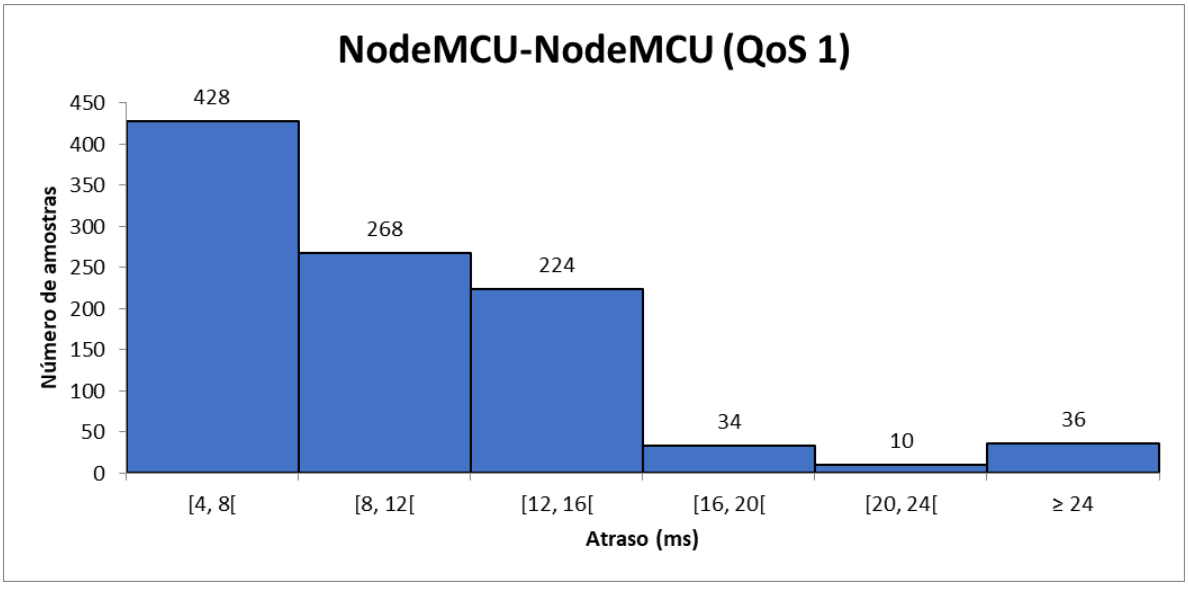


Figura 100. Histograma com os resultados do teste de atraso com a placa NodeMCU como nó sensor e nó atuador (QoS 1).

Através do programa Wireshark [65] foi possível verificar o percurso dos pacotes desde a origem até ao destino (ver Figura 8) e confirmar os tempos de atraso. Como exemplo, na Figura 101, verifica-se que a placa NodeMCU (origem) com endereço IP 192.168.1.113 envia inicialmente o pacote (Publish Message) para o *broker* com endereço IP 192.168.1.213 (endereço da Raspberry Pi). O *broker* confirma a receção enviando um Publish Ack (PUBACK) para a origem e de seguida envia o pacote para o destino com o endereço IP 192.168.1.104 (endereço da outra placa NodeMCU). Finalmente, o destino, após receção do pacote envia um Publish Ack ao *broker* confirmando a receção. Todo este percurso desde a origem até ao destino demorou, neste caso, 11 ms.

No.	Time	Source	Destination	Protocol	Length	Info
8544	20.622964007	192.168.1.113	192.168.1.213	MQTT	70	Publish Message (id=4686) [atraso]
8545	20.623090616	192.168.1.213	192.168.1.113	MQTT	58	Publish Ack (id=4686)
8546	20.623125449	192.168.1.213	192.168.1.104	MQTT	70	Publish Message (id=4690) [atraso]
8547	20.634440609	192.168.1.104	192.168.1.213	MQTT	58	Publish Ack (id=4690)

Figura 101. Análise do protocolo MQTT com a placa NodeMCU como nó sensor e nó atuador (QoS 1) através do programa Wireshark.

4.1.1.2 QoS 2

Na Figura 102 estão representados, através de um histograma, os tempos de atraso obtidos para um QoS 2 após o envio de 1000 amostras. O atraso de 83,6% das amostras está compreendido entre 5 e 25 ms e apenas 5,1% está acima dos 55 ms.

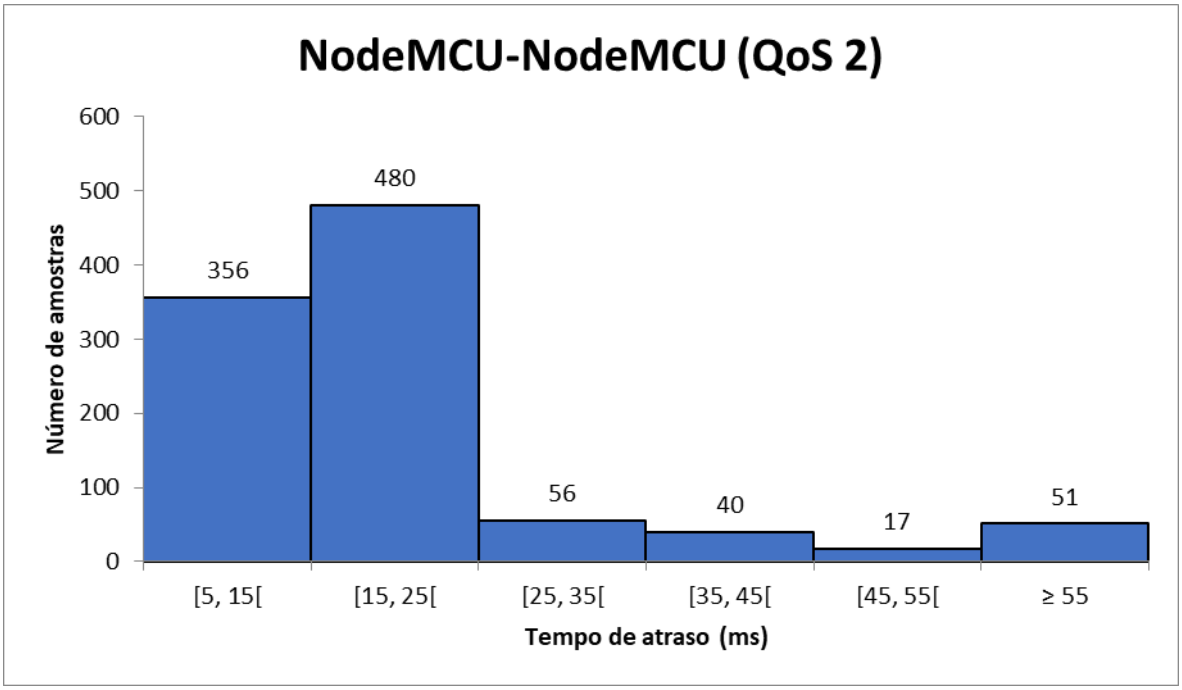


Figura 102. Histograma com os resultados do teste de atraso com a placa NodeMCU como nó sensor e nó atuador (QoS 2).

Na Figura 103 é demonstrado, através do WireShark, o percurso de um pacote desde a origem (192.168.1.104) até ao destino (192.168.1.113) com QoS 2 (ver Figura 9). Neste caso, o envio dos dados requereu aproximadamente 20 ms para completar.

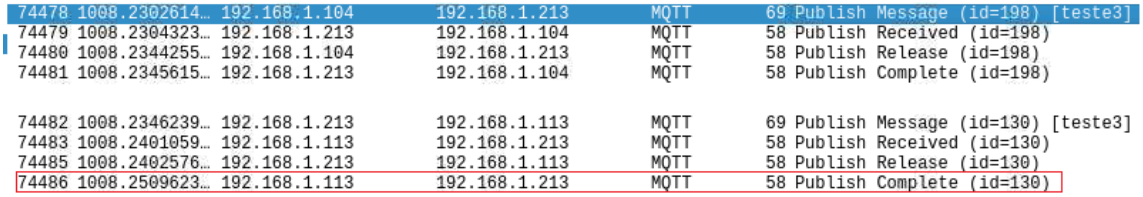


Figura 103. Análise do protocolo MQTT com a placa NodeMCU como nó sensor e nó atuador (QoS 2) através do programa Wireshark.

4.1.2 Nó sensor: ESP32

Nesta seção são apresentados os testes realizados com o módulo ESP32 (BLE) como nó sensor e a NodeMCU (Wi-Fi) como nó atuador para QoS 1 e 2. O percurso dos pacotes começa do servidor BLE no nó sensor até ao cliente BLE na Raspberry Pi. Na Raspberry Pi os dados recebidos através do cliente BLE são transferidos para o cliente MQTT (publicador). Em seguida, os dados são enviados para o *broker* MQTT na Raspberry Pi que, por fim, envia para o subscritor MQTT no nó atuador, via Wi-Fi.

4.1.2.1 QoS 1

Na Figura 104 estão representados, através de um histograma, os tempos de atraso obtidos após o envio de 1000 amostras. O tempo de atraso de 86% das amostras está compreendido entre os 40 e os 55 ms.

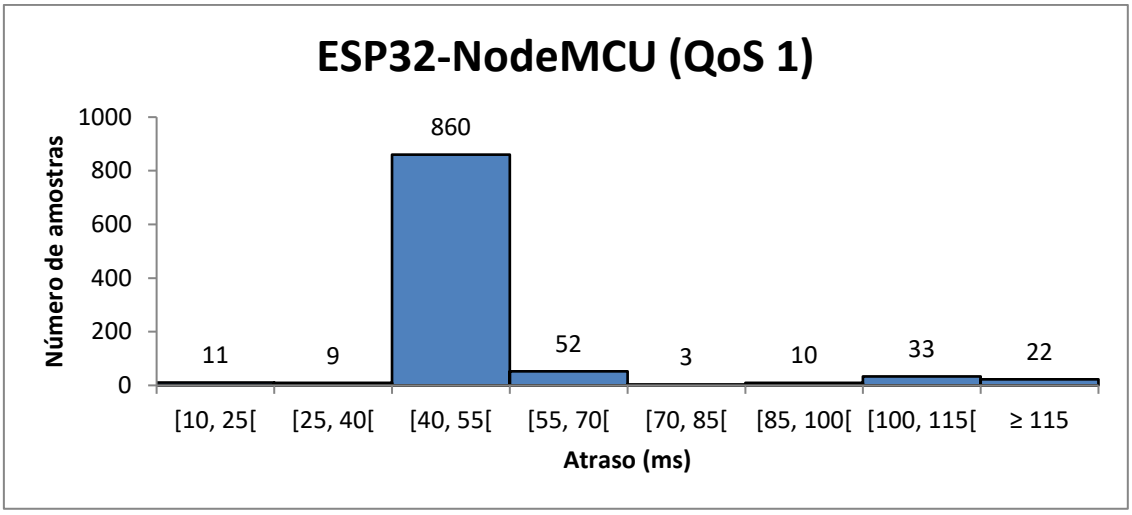


Figura 104. Histograma com os resultados do teste de atraso com a placa ESP32 como nó sensor e a placa NodeMCU como nó atuador (QoS 1).

Como uma amostra é apenas enviada após a receção da amostra anterior no destino, se verificarmos através do programa Wireshark o intervalo de tempo entre cada amostra recebida no cliente BLE da Raspberry (Figura 105), constatamos que é aproximadamente igual ao atraso médio. Não é possível contabilizar o tempo com Wireshark com o método anterior (tempo de atraso = tempo de chegada – tempo de envio) porque o Wireshark só permite capturar uma interface de cada vez e como este método envolve BLE e Wi-Fi seria necessário capturar os dados na interface bluetooth0 e na interface wlan0 ao mesmo tempo.

No.	Time	Source	Destination	Protocol	Length	Info
1570	2965.334597	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1571	2965.377975	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1572	2965.427980	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1573	2965.477981	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1574	2965.527980	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1575	2965.577985	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1576	2965.627982	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1577	2965.677996	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1578	2965.727985	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1579	2965.777978	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)
1580	2965.834586	Espressi_09:2a:22 (…	localhost ()	ATT	15	Rcvd Handle Value Notification, Handle: 0x002a (Unknown: Unknown)

Figura 105. Captura dos pacotes recebidos na interface bluetooth0 da Raspberry através do Wireshark (QoS 1).

4.1.2.2 QoS 2

Na Figura 106 estão representados, através de um histograma, os atrasos obtidos após o envio de 1000 amostras. Com QoS 2 o número de amostras com atraso entre os 40 e os 55 ms diminuiu ligeiramente para os 84,4% e o número de amostras com atraso superior a 115 ms aumentou aproximadamente para o dobro (22 para 50).

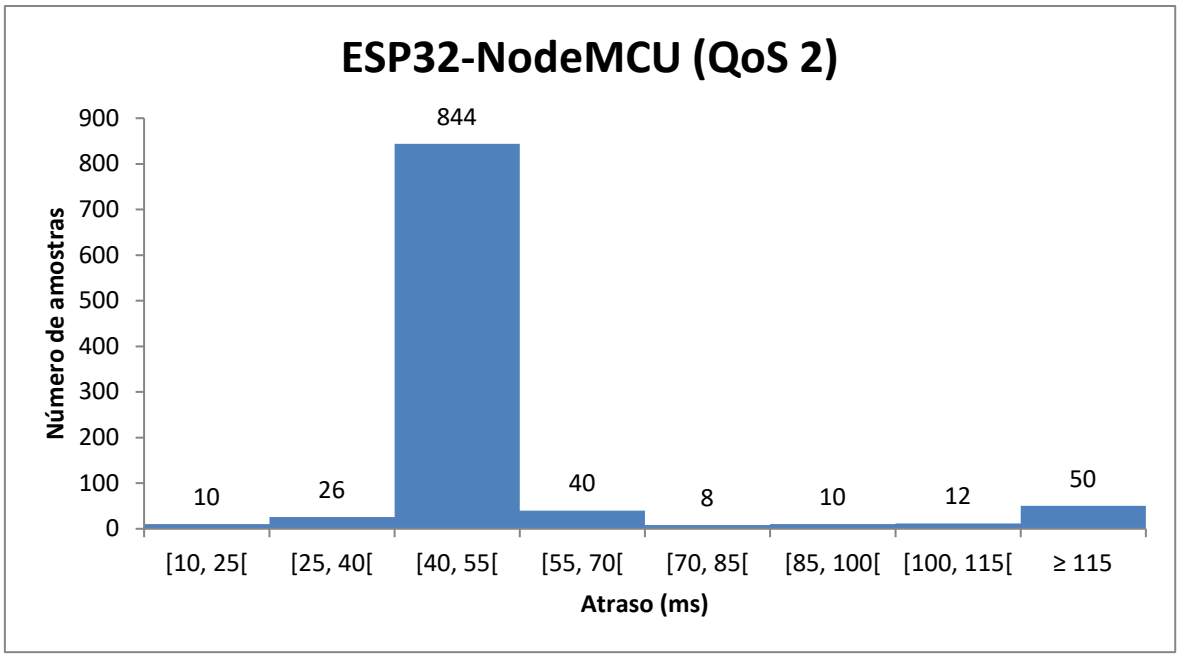
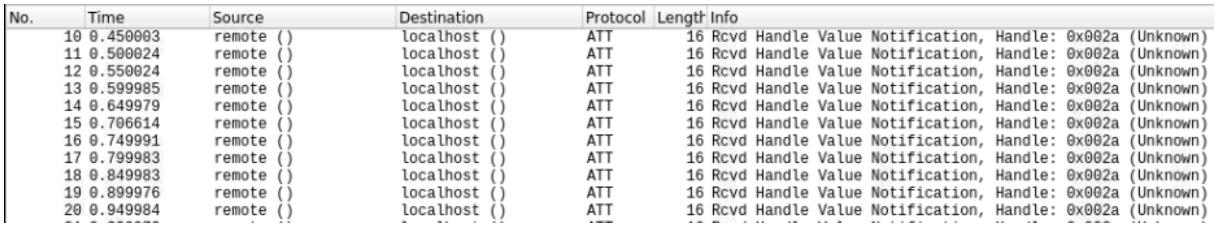


Figura 106. Histograma com os resultados do teste de atraso com a placa ESP32 como nó sensor e a placa NodeMCU como nó atuador (QoS 2).

Verificando novamente com o programa Wireshark, o atraso entre cada amostra recebida no cliente BLE da Raspberry (Figura 107) é aproximadamente o tempo de atraso médio.



No.	Time	Source	Destination	Protocol	Length	Info
10	0.450003	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
11	0.500024	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
12	0.550024	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
13	0.599985	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
14	0.649979	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
15	0.706614	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
16	0.749991	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
17	0.799983	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
18	0.849983	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
19	0.899976	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)
20	0.949984	remote ()	localhost ()	ATT	16	Rcvd Handle Value Notification, Handle: 0x002a (Unknown)

Figura 107. Captura dos pacotes recebidos na interface bluetooth0 da Raspberry através do Wireshark (QoS 2).

4.1.3 Discussão dos resultados

Após a obtenção dos resultados da seção 4.1.1 e 4.1.2 foram colocados na Tabela 4 e Tabela 5, respetivamente, o valor mínimo e máximo, a média e a taxa de perda de pacotes.

Tabela 4. Resultados obtidos quando o nó sensor e o nó atuador são uma NodeMCU (QoS 1 e 2).

	Mínimo (ms)	Média (ms)	Máximo (ms)	Taxa de perdas
QoS 1	4,011	9,783	81,530	0%
QoS 2	7,612	16,548	307,626	0%

Tabela 5. Resultados obtidos quando o nó sensor é uma ESP32 e o nó atuador uma NodeMCU.

	Mínimo (ms)	Média (ms)	Máximo (ms)	Taxa de perdas
QoS 1	12,227	52,567	285,068	0%
QoS 2	23,335	53,501	453,786	0%

Através da análise dos resultados obtidos, verificou-se que o atraso mínimo e médio com o método QoS 1 (4,011 e 7,612) é aproximadamente metade do obtido com o método QoS 2 (9,783 e 16,548) na configuração em que é utilizado apenas Wi-Fi, pois necessita apenas de metade dos pacotes na comunicação. A configuração baseada somente em Wi-Fi apresenta atrasos médios menores (9,783 e 16,548) do que a configuração de rede híbrida BLE/Wi-Fi (52,567 e 53,501), o que pode ser explicado pelo menor atraso de acesso e transmissão de pacotes do Wi-Fi em comparação com o BLE. Em todos os casos não houve perdas de pacotes. Os tempos de atraso máximos, em todos os casos, são inferiores a 1 segundo, o que permite que o sistema reaja rapidamente a alterações no valor de corrente, evitando que o disjuntor na habitação atue e possibilitando que o utilizador receba/envie dados de forma quase

instantânea. Em relação ao método MQTT a ser utilizado no sistema é aconselhável utilizar o QoS 2, pois apesar dos tempos de atraso de QoS1 serem menores podem ocorrer mensagens duplicadas que poderão levar ao mau funcionamento do sistema.

4.2 Controlo e monitorização com a aplicação Android

Nesta secção são apresentados resultados relativamente ao envio/receção de dados (através do cliente MQTT) e a representação dos mesmos com a aplicação Android desenvolvida (secção 3.8). É importante referir que todos os valores de corrente, tensão e percentagem de bateria foram emulados através da modificação do seu valor nos tópicos MQTT correspondentes.

4.2.1 Sistema de carregamento do veículo elétrico

Na Figura 108 foi definido através do método CF um tempo de carregamento de três horas. Tendo em conta que a bateria do veículo é de 40 KWh e que esta estava com 36 KWh, ou seja, 90% da sua carga total, faltavam apenas 4 KWh para os 100%. Com uma tensão da rede elétrica de 230 V, a corrente de referência no lado AC calculada com a equação (6) foi de 5,80 A. No gráfico localizado na parte inferior pode se verificar que a corrente passou de um valor nulo (porque o carregador estava desligado) para o valor calculado.

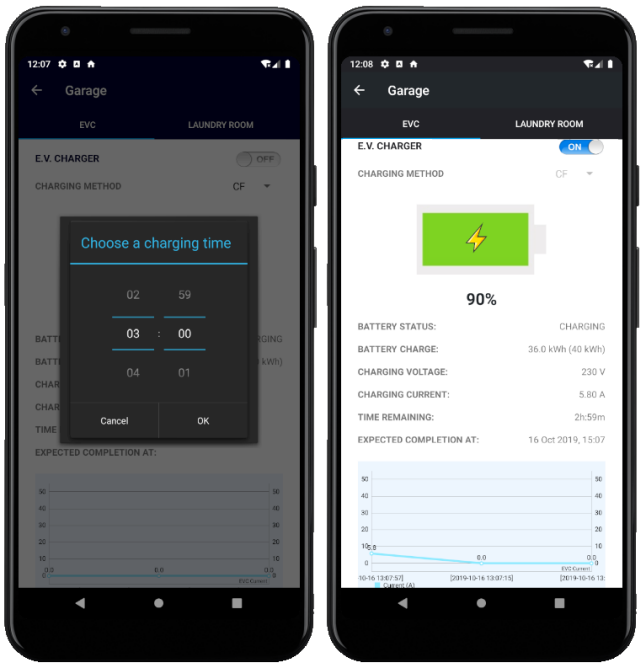


Figura 108. Aplicação Android: Fragment EVC - Carregamento da bateria com o método CF. Definição do tempo de carregamento (esquerda) e dados resultantes dessa escolha (direita).

Na Figura 109 foi selecionado o sistema de carregamento com o método CV, sendo, portanto, toda a corrente disponível na casa utilizada para este método. Neste caso, com uma corrente disponível de 10 A e com carga inicial de 14 kWh, correspondente a 35% da capacidade da bateria, o tempo estimado para a conclusão seria de 11 horas e 18 minutos. Obviamente estes valores podem variar caso haja alterações na corrente utilizada pelos outros equipamentos da *smart home.*

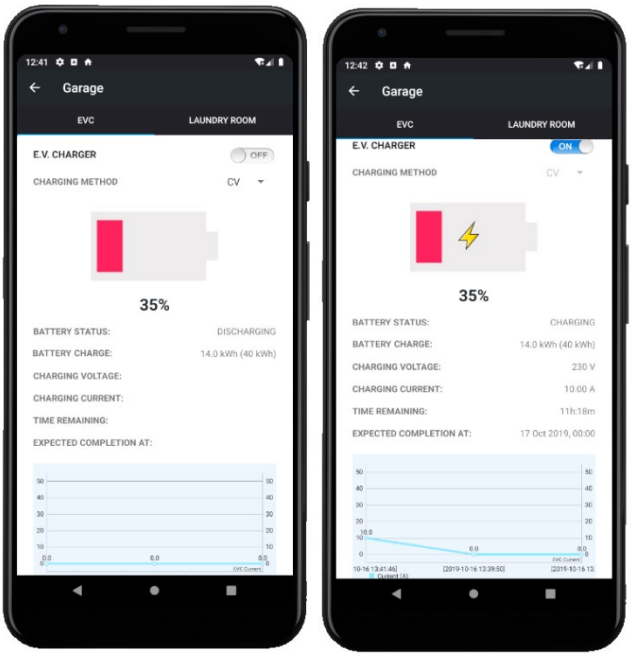


Figura 109. Aplicação Android: Fragment EVC - Carregamento da bateria com o método CV. Seleção do método CV (esquerda) e dados resultantes dessa escolha (direita).

Caso o utilizador deseje ligar o carregamento com o método CV e a corrente inicial disponível seja nula surge um aviso como o que está representado na Figura 110.

Figura 110. Aplicação Android: Fragment EVC: Aviso de que não existe corrente disponível para iniciar o método CV.

4.2.2 Equipamentos e sensores

Na Figura 111 e Figura 112 estão dispostos os Fragments correspondentes às Activities First Floor e Ground Floor. Os valores de temperatura e humidade em todos os casos foram adquiridos periodicamente (de 1 em 1 minuto) por sensores DHT11 (secção 3.2.6) e o controlo de equipamentos permitiu desligar/ligar equipamentos com o módulo relé (secção 3.2.7). Todos os valores foram recebidos com sucesso e os Fragments foram atualizados sempre que surgia um novo valor nos tópicos correspondentes.

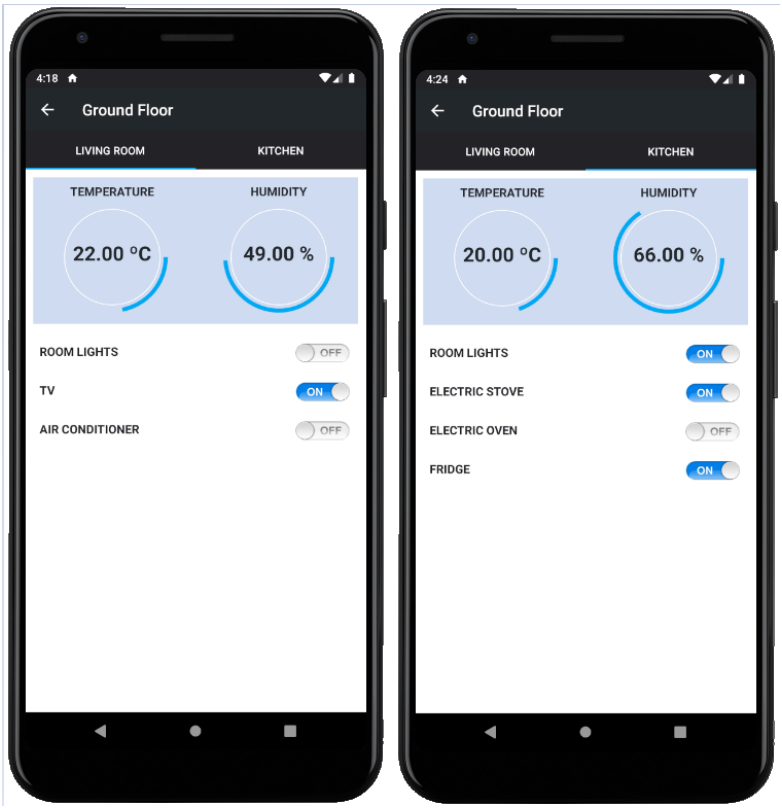


Figura 111. Aplicação Android: Activity Ground Floor (Fragment Living Room e Fragment Kitchen).

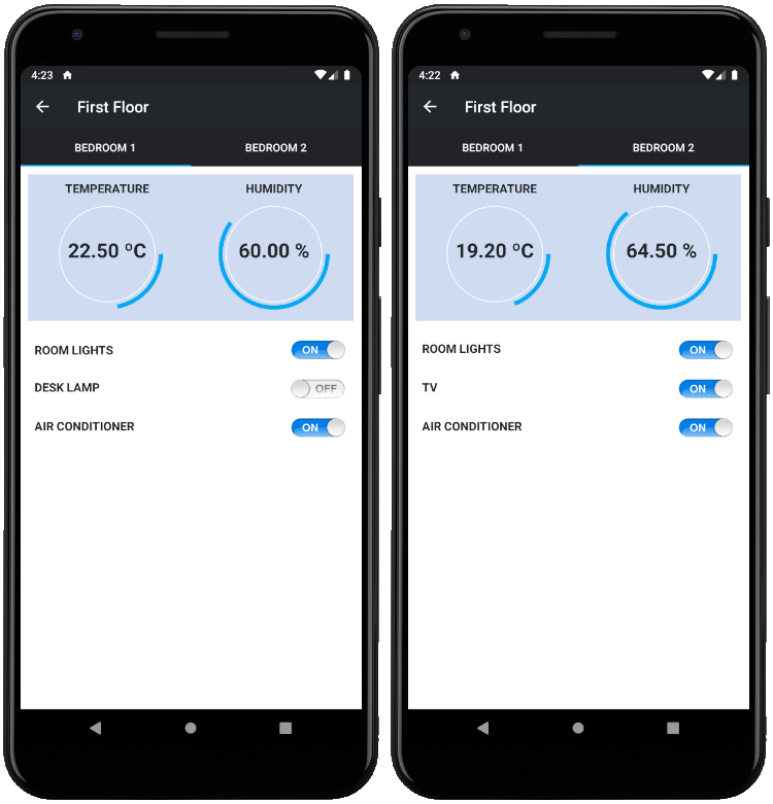


Figura 112. Aplicação Android: Activity First Floor (Fragment Bedroom1 e Fragment Bedroom2).

4.2.3 Consumo energético

Na Figura 113 estão apresentados o Fragment Current Distribution e o Fragment Home Consumption da Activity Energy. No Fragment Current Distribution foi possível visualizar, através de um gráfico circular, a percentagem de corrente que cada divisão utilizava. Visto não haverem sensores de corrente suficientes para todos os equipamentos, os valores de corrente foram introduzidos manualmente em cada tópico MQTT correspondente. Para o caso demonstrado está a ser utilizada 53,63% da corrente máxima, sendo o Bedroom 1 a divisão com maior consumo.

No Fragment Home Consumption foi possível visualizar, através de um gráfico de linhas horizontal, a variação da potência total da casa ao longo do tempo. Esses valores podem ser representados na forma de corrente ou potência aparente, e foram obtidos consultando os últimos valores da base de dados referentes ao sensor de corrente do quadro elétrico da casa. Mais uma vez se destaca que estes valores foram introduzidos manualmente nos tópicos adequados de forma a comprovar o funcionamento correto da aplicação.

Figura 113. Aplicação Android: Activity Energy (Fragment Current Distribution e Fragment Home Consumption).

4.2.4 Notificações

Foram definidas certas notificações quando são recebidos determinados valores em certos tópicos. Na Figura 114, como exemplo, é demonstrada a notificação que é recebida quando a bateria do veículo está totalmente carregada.

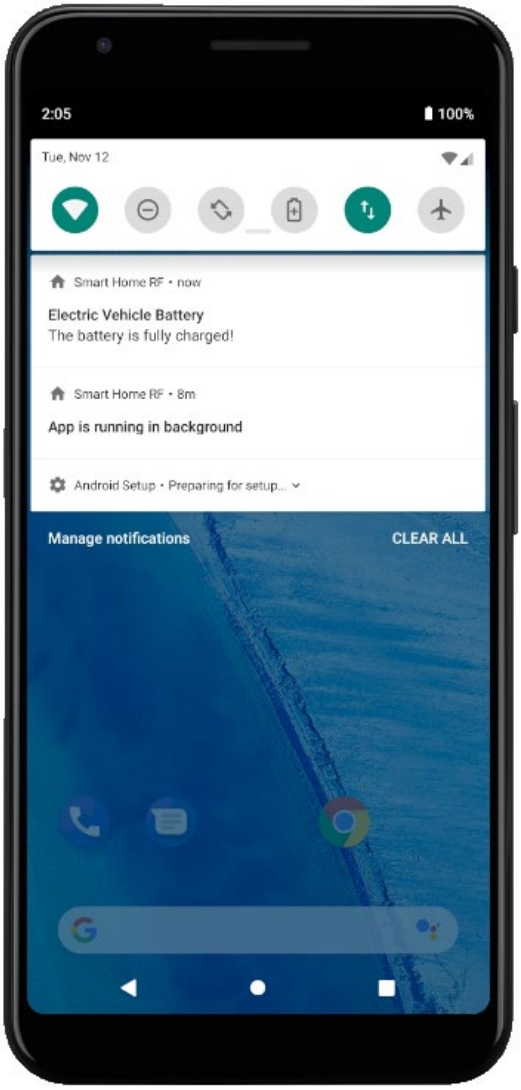


Figura 114. Aplicação Android: Notificação de que a bateria do veículo elétrico está carregada.

4.3 Protótipo

4.3.1 Integração do sistema desenvolvido com o carregador da bateria do veículo elétrico

Visto que a eletrónica do sistema de carregamento do veículo elétrico já estava desenvolvida, foi adicionado um módulo NodeMCU para permitir a comunicação com o sistema desenvolvido via Wi-Fi (Figura 115). O sistema de carregamento já contém todos os componentes necessários para controlo e monitorização, como sensores de corrente e tensão, relés, entre outros [61] e comunica com o módulo Wi-Fi através da porta série (secção 3.3.5).

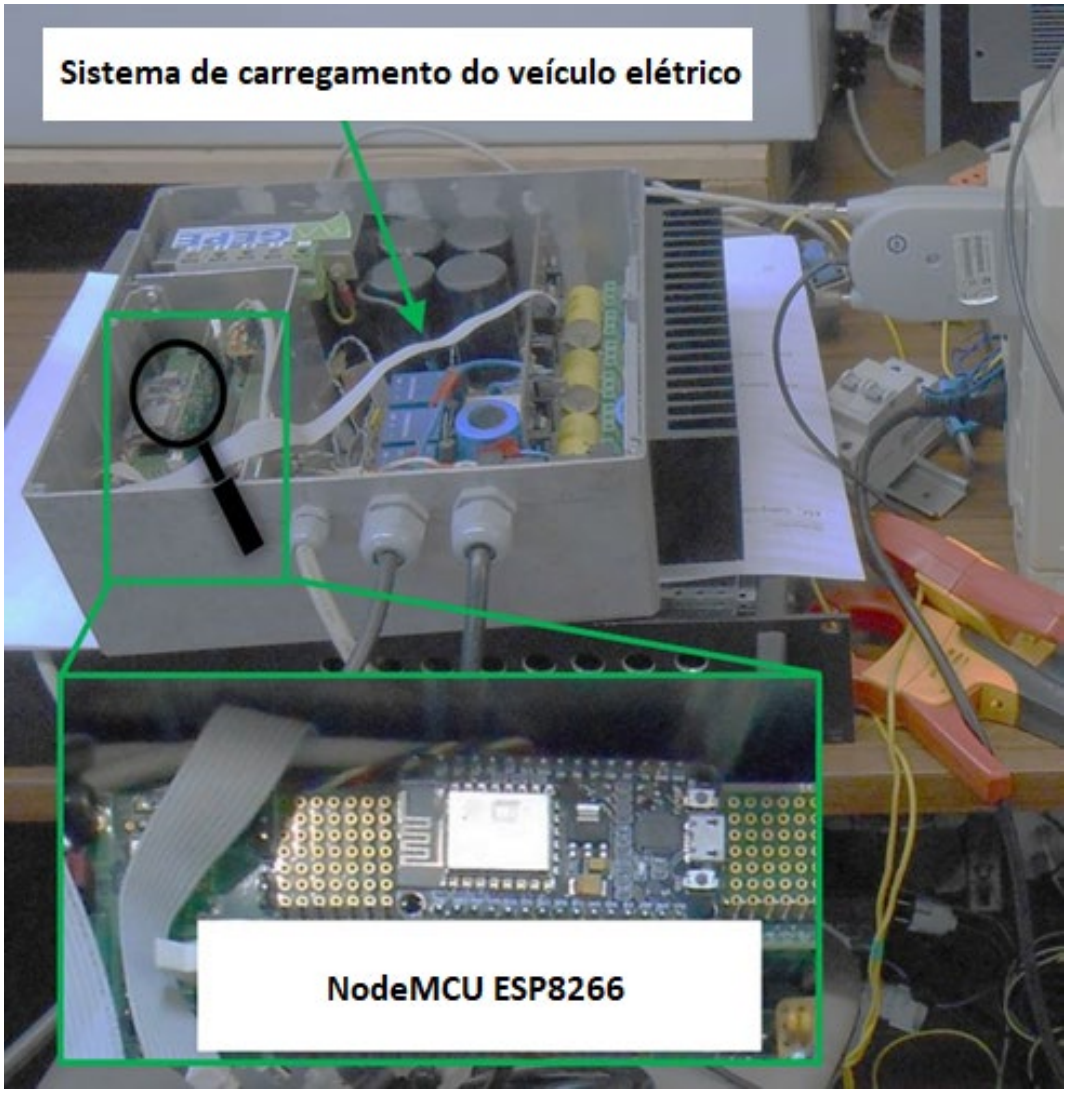
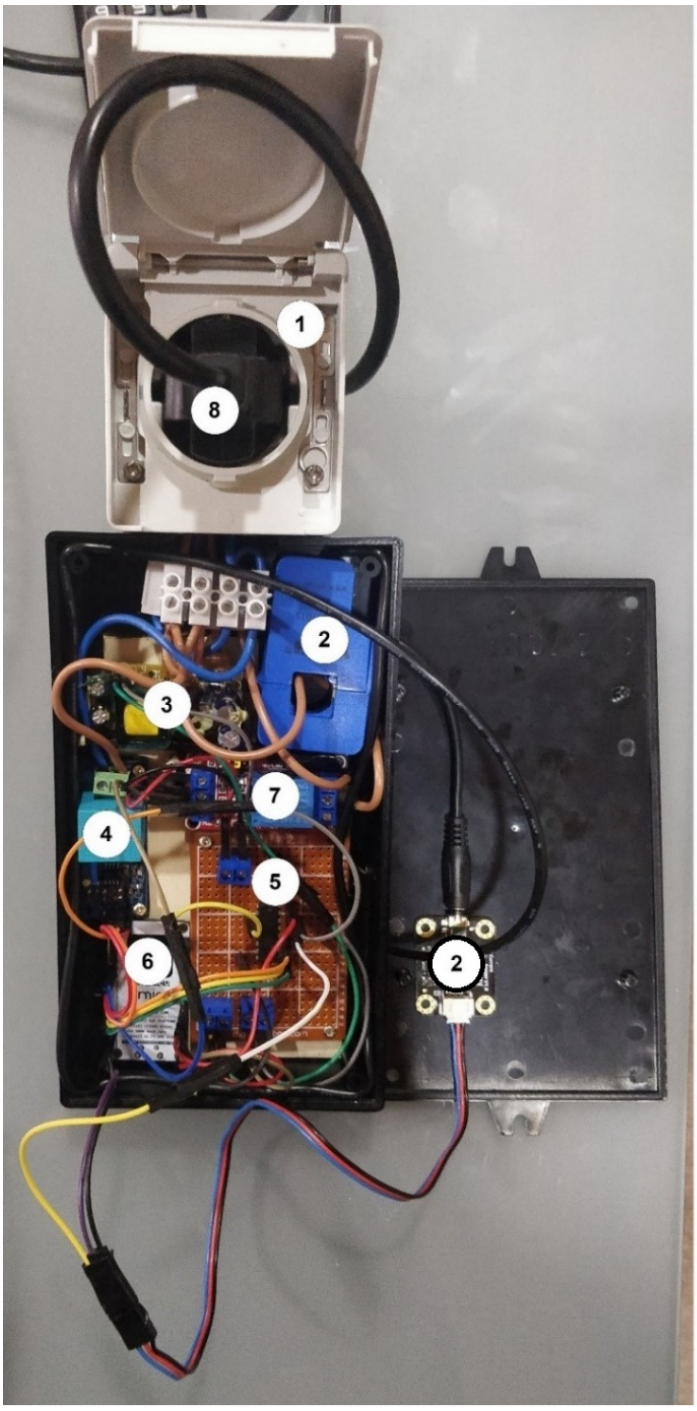


Figura 115. Sistema de carregamento do veículo elétrico com módulo Wi-Fi.

4.3.2 Tomada inteligente

Foi implementado um protótipo da tomada inteligente referida na secção 3.7. Na Figura 116 estão dispostos todos os componentes utilizados na elaboração da tomada inteligente e na Figura 117 o protótipo final.



Componentes

1 - Tomada fêmea

2 - Sensor analógico de corrente AC DFRobot Gravity 20 A

3 - Conversor AC (240V) - DC (5V) 700 mA

4 - Sensor analógico de tensão AC Zmpt101b

5 - Multiplexador analógico 4051

6 - NodeMCU

7 - Módulo relé

8 - Ficha do aparelho elétrico

Figura 116. Descrição dos componentes do protótipo da tomada inteligente.

Figura 117. Vista frontal (esquerda) e lateral (direita) do protótipo da tomada inteligente.

Inicialmente foram realizados testes para verificar se os valores medidos e calculados com a tomada inteligente coincidiam com os valores obtidos com um multímetro. Na Figura 118 estão representados os valores eficazes de corrente (3,09 A) e de tensão (227,32 V) obtidos (de 1 em 1 segundo) pela tomada inteligente, e o valor eficaz de corrente (3,01 A) obtido pelo multímetro quando um aspirador está ligado. A corrente medida em ambos é idêntica, logo a corrente eficaz está a ser bem medida e calculada. Se multiplicarmos o valor da corrente e da tensão, adquiridos pela tomada, obtemos uma potência aparente de 702 VA.

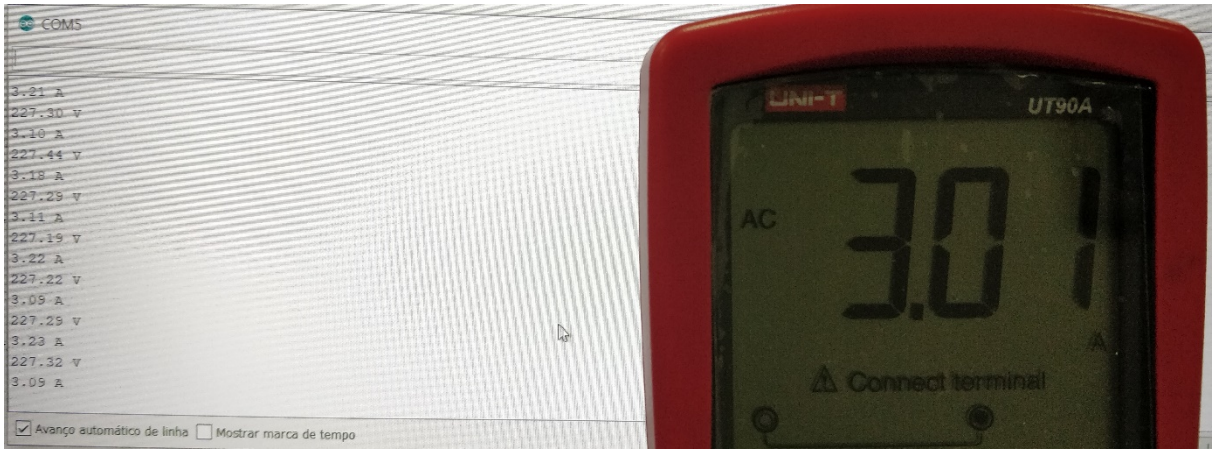


Figura 118. Comparação entre o valor de corrente eficaz obtido pela tomada inteligente e pelo multímetro quando um aspirador está ligado.

Na Figura 119 estão representados os valores de tensão eficaz obtidos pela tomada e pelo multímetro quando nada está ligado (daí não serem apresentados valores de corrente). Verifica-se que os valores são iguais e que, portanto, a tomada está a obter e a calcular corretamente o valor eficaz da tensão.

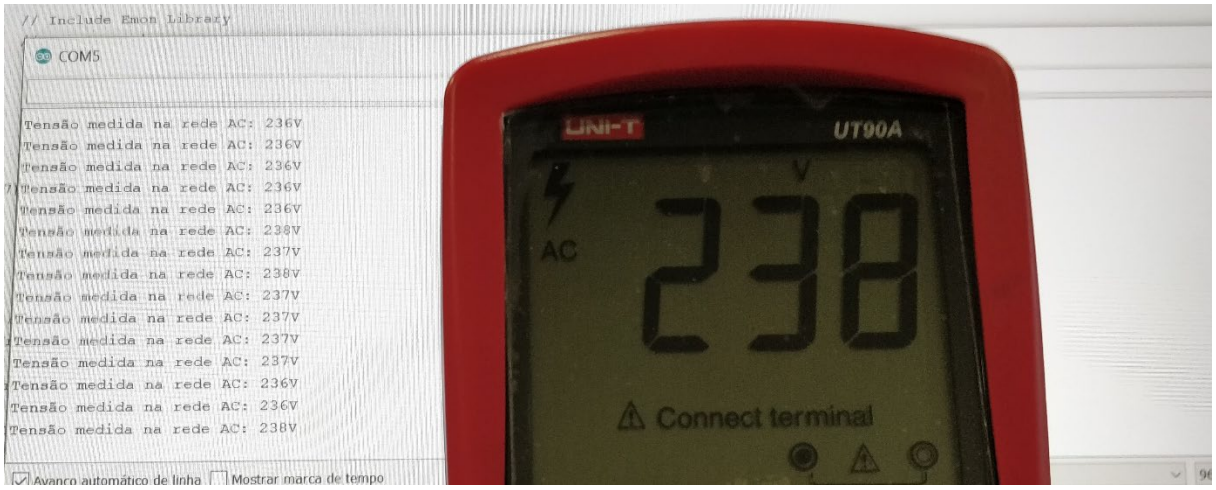


Figura 119. Comparação entre o valor de tensão eficaz obtido pela tomada inteligente e pelo multímetro quando nada está ligado.

Após se confirmar que os valores medidos pela tomada inteligente estavam de acordo com o multímetro, foi realizado um último teste com um aquecedor ligado à tomada, onde foram obtidos (de 1 em 1 segundo) os valores da corrente, da tensão e da consequente potência utlizada por este aparelho. A Figura 120 demonstra uma potência consumida de aproximadamente 1600 VA quando a tomada está ativada (Estado:ON) e a Figura 121 demonstra uma potência consumida de 0 VA quando a tomada está desativada (Estado:OFF).

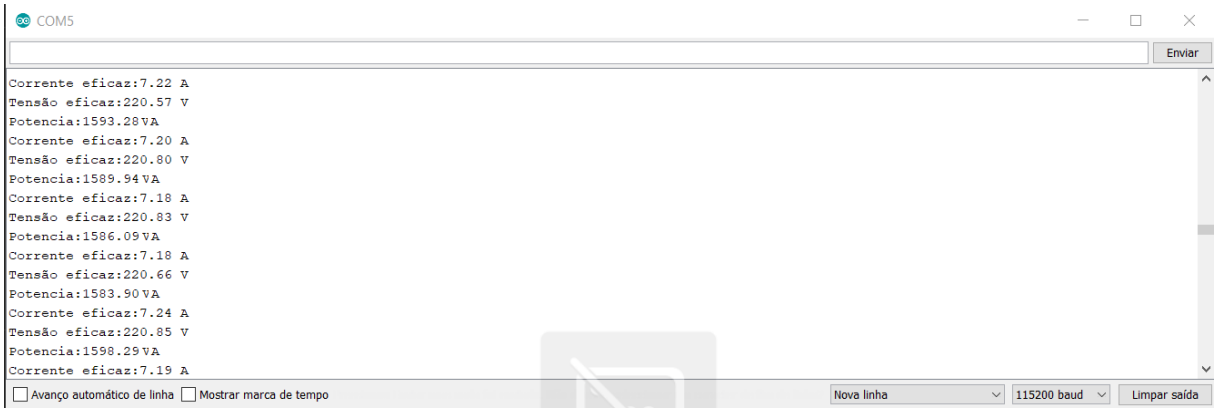


Figura 120. Valores obtidos quando um aquecedor está ligado à tomada (Estado:ON).

É possível verificar na Figura 121 que logo após a tomada ser desativada ainda existe ainda alguma tensão e corrente residual que diminui rapidamente para zero, no entanto, o valor de tensão nunca é nulo devido ao campo eletromagnético existente à volta. Visto se tratar de uma tensão muito baixa na ordem dos milivolts esta pode ser desprezada e em nada interfere com a potência aparente uma vez que a corrente é nula.

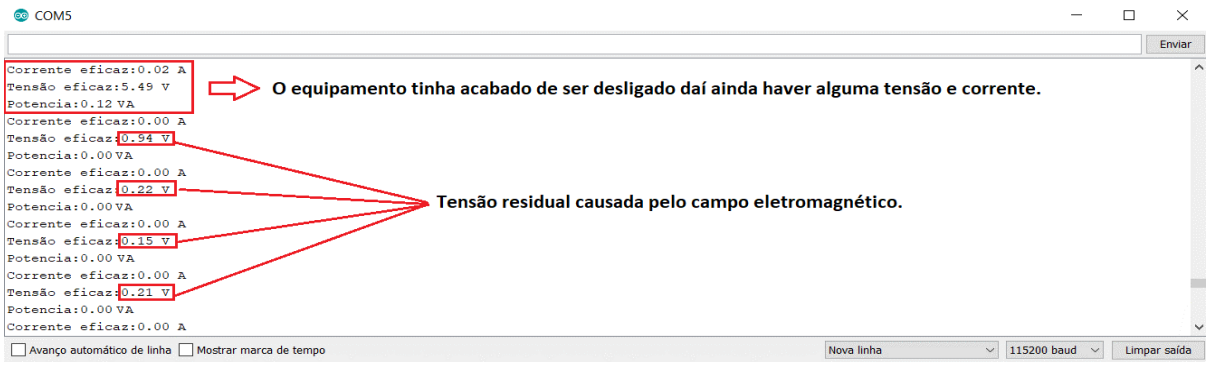


Figura 121. Valores obtidos quando um aquecedor está ligado à tomada (Estado:OFF).

5.Conclusões e sugestões de trabalho futuro

5.1 Conclusões

Esta dissertação tinha como objetivo o desenvolvimento de um sistema de controlo e monitorização residencial que permitisse melhorar a qualidade de vida das pessoas, aumentar a eficiência energética e integrar um sistema de carregamento de veículo elétrico. Para tal, foi efetuado um estudo inicial de forma a especificar uma arquitetura que cumprisse com as necessidades e funcionalidades desejadas. A arquitetura escolhida é constituída por vários dispositivos Bluetooth Low Energy (BLE) e/ou Wi-Fi com diferentes funções, como nós sensores, nós atuadores e um *hub/gateway*. O protocolo escolhido para comunicação entre as entidades da plataforma IoT foi o MQTT. Após a definição da arquitetura foram analisadas as opções disponíveis no mercado para realizar a monitorização e controlo dos equipamentos da *smart home* de forma eficiente. Os dispositivos escolhidos foram: NodeMCU ESP8266 12E como nó sensor/atuador Wi-Fi; Firebeetle ESP32 como nó sensor BLE; Raspberry Pi 4 como *hub/gateway;* DHT11 como sensor de humidade e temperatura; ZMPT101B como sensor de tensão; DFRobot Gravity 20 A como sensor de corrente; Display OLED 0.91 como auxiliar de depuração; módulo relé de 1 canal como atuador; e kit Texas Instruments TMDSDOCK28335 como interface com o sistema de carregamento da bateria.

Na NodeMCU foi implementado um cliente MQTT que envia/recebe dados (e.g., dados adquiridos pelo sensor DHT11) e na ESP32 foi implementado um servidor BLE que envia dados. No *hub/gateway* foram implementadas diversas funcionalidades: um *broker* MQTT que faz a gestão das mensagens trocadas na rede; um *gateway* BLE/Wi-Fi, que permite a comunicação entre dispositivos que utilizam estes protocolos; uma base de dados local; um servidor HTTP e uma Web API, que permitem consultar a base de dados através de aplicações Web e Android; um *script* que insere automaticamente dados relevantes na base de dados; e um algoritmo que gere a corrente disponível na casa.

Foi ainda desenvolvida uma aplicação Android que controla e monitoriza equipamentos através de uma interface gráfica. Nesta aplicação é possível ligar e desligar dispositivos, consultar a energia/corrente que está a ser consumida, gerir o sistema de carregamento de um veículo elétrico, entre outros.

Foram realizados testes para avaliar o atraso e a fiabilidade no envio de pacotes para diferentes tipos de nós sensores (BLE e Wi-Fi). Nesses testes verificou-se que os valores atraso obtidos permitem que o sistema reaja de forma rápida a alterações de corrente prevenido o disparo do disjuntor.

Por último foi desenvolvido um protótipo de uma tomada inteligente que envia periodicamente os valores de corrente e tensão do equipamento que está ligado permitindo também modificar o seu estado (ON/OFF).

Um dos grandes desafios iniciais foi o facto dos endereços IP dos dispositivos mudarem ao longo do tempo na rede. Como solução foi utilizado o protocolo MQTT e atribuído um IP estático ao *hub/gateway*. Outro aspeto que dificultou o desenvolvimento foram as diferentes linguagens de programação necessárias: Java; PHP; Python; C/C++; e o facto de nunca ter desenvolvido uma aplicação Android.

Posto isto e analisados os objetivos propostos no início do projeto conclui-se que estes foram cumpridos com sucesso, resultando num sistema prático, funcional e de baixo custo.

5.2 Sugestões de trabalho futuro

Apesar de o sistema desenvolvido ser funcional e os resultados serem bastante satisfatórios, é sempre possível o seu aperfeiçoamento. Como trabalho futuro propõe-se:

- Implementação e análise do sistema num contexto real sendo os valores dos sensores adquiridos de forma periódica ou quando for detetada uma variação superior a um *threshold* pré-definido.
- Implementação de mecanismos de segurança na rede;
- Melhoramentos a nível do sistema de gestão de energia, com algoritmos mais complexos e eficazes;
- Aplicação Android com mais recursos e funcionalidades;
- Comandos de voz através do *hub/gateway;*
- Maior diversidade de sensores e atuadores.
- Conexão a uma *cloud* IoT*.*

Referências

[1] V. Govindraj, M. Sathiyanarayanan, e B. Abubakar, «Customary homes to smart homes using Internet of Things (IoT) and mobile application», em *Proceedings of the 2017 International Conference On Smart Technology for Smart Nation, SmartTechCon 2017*, 2018, pp. 1059–1063.

[2] L. Liu, Y. Liu, L. Wang, A. Zomaya, e S. Hu, «Economical and Balanced Energy Usage in the Smart Home Infrastructure: A Tutorial and New Results», *IEEE Trans. Emerg. Top. Comput.*, vol. 3, n. 4, pp. 556–570, Dez. 2015.

[3] S. Singh e N. Singh, «Internet of Things (IoT): Security challenges, business opportunities & reference architecture for E-commerce», em *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, 2015, pp. 1577–1581.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, e M. Ayyash, «Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications», *IEEE Commun. Surv. Tutorials*, vol. 17, n. 4, pp. 2347–2376, Out. 2015.

[5] L. Da Xu, W. He, e S. Li, «Internet of things in industries: A survey», *IEEE Transactions on Industrial Informatics*, vol. 10, n. 4. IEEE Computer Society, pp. 2233–2243, 01-Nov-2014.

[6] R. Khan, S. U. Khan, R. Zaheer, e S. Khan, «Future internet: The internet of things architecture, possible applications and key challenges», em *Proceedings - 10th International Conference on Frontiers of Information Technology, FIT 2012*, 2012, pp. 257–260.

[7] S. Krco, B. Pokric, e F. Carrez, «Designing IoT architecture(s): A European perspective», *2014 IEEE World Forum Internet Things, WF-IoT 2014*, pp. 79–84, 2014.

[8] H. Fernández-López, J. A. Afonso, J. H. Correia, e R. Simoes, «Towards the design of efficient nonbeacon-enabled ZigBee networks», *Comput. Networks*, 2012.

[9] J. A. Afonso, A. J. F. Maio, e R. Simoes, «Performance Evaluation of Bluetooth Low Energy for High Data Rate Body Area Networks», *Wirel. Pers. Commun.*, vol. 90, n. 1, pp. 121–141, 2016.

[10] A. Augustin, J. Yi, T. Clausen, e W. M. Townsley, «A study of Lora: Long range & low power networks for the internet of things», *Sensors (Switzerland)*, vol. 16, n. 9, Set. 2016.

[11] B. Vejlgaard, M. Lauridsen, H. Nguyen, I. Z. Kovacs, P. Mogensen, e M. Sorensen, «Coverage and Capacity Analysis of Sigfox, LoRa, GPRS, and NB-IoT», em *IEEE Vehicular Technology Conference*, 2017, vol. 2017-June.

[12] G. R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X. P. Costa, e B. Walke, «The IEEE 802.11 universe», *IEEE Commun. Mag.*, vol. 48, n. 1, pp. 62–70, Jan. 2010.

[13] E. Ferro e F. Potorti, «Bluetooth and wi-fi wireless protocols: a survey and a comparison», *IEEE Wirel. Commun.*, vol. 12, n. 1, pp. 12–26, Fev. 2005.

[14] C. Gomez, J. Oller, J. Paradells, C. Gomez, J. Oller, e J. Paradells, «Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology», *Sensors*, vol. 12, n. 9, pp. 11734–11753, Ago. 2012.

[15] R. Tei, H. Yamazawa, e T. Shimizu, «BLE power consumption estimation and its applications to smart manufacturing», em *2015 54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 2015, pp. 148–153.

[16] F. J. Dian, A. Yousefi, e S. Lim, «A practical study on Bluetooth Low Energy (BLE) throughput», em *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 768–771.

[17] R. K. Kodali e V. S. K. Gorantla, «Weather tracking system using MQTT and SQLite», em *2017 3rd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, 2017, pp. 205–208.

[18] J. Toldinas, B. Lozinskis, E. Baranauskas, e A. Dobrovolskis, «MQTT Quality of Service versus Energy Consumption», 2019, pp. 1–4.

[19] Y. Sasaki, T. Yokotani, e H. Mukai, «Comparison with Assured Transfer of Information Mechanisms in MQTT», em *2018 International Japan-Africa Conference on Electronics, Communications and Computations (JAC-ECC)*, 2018, pp. 95–98.

[20] Android Developers, «Download Android Studio and SDK tools». [Em linha]. Disponível em: https://developer.android.com/studio/index.html. [Acedido: 03-Mai-2019].

[21] Android Developers, «Platform Architecture». [Em linha]. Disponível em: https://developer.android.com/guide/platform. [Acedido: 25-Abr-2019].

[22] Android Developers, «Understand the Activity Lifecycle». [Em linha]. Disponível em:

https://developer.android.com/guide/components/activities/activity-lifecycle.html. [Acedido: 20-Abr-2019].

[23] Android Developers, «Fragments». [Em linha]. Disponível em: https://developer.android.com/guide/components/fragments. [Acedido: 25-Abr-2019].

[24] Android Developers, «Services overview». [Em linha]. Disponível em: https://developer.android.com/guide/components/services. [Acedido: 26-Abr-2019].

[25] R. K. Kodali e S. Yerroju, «Energy Efficient Home Automation Using IoT», em *2018 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, 2018, pp. 151–154.

[26] R. K. Kodali e S. Soratkal, «MQTT based home automation system using ESP8266», em *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, 2016, pp. 1–5.

[27] J. Joshi *et al.*, «Performance enhancement and IoT based monitoring for smart home», em *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 468–473.

[28] T. Dahoumane, M. Haddadi, e Z. Amokrane, «Web Services and GSM based Smart Home Control System», em *Proceedings of the 2018 International Conference on Applied Smart Systems, ICASS 2018*, 2019, pp. 24–25.

[29] S. Al-Sarawi, M. Anbar, K. Alieyan, e M. Alzubaidi, «Internet of Things (IoT) communication protocols: Review», em *ICIT 2017 - 8th International Conference on Information Technology, Proceedings*, 2017, pp. 685–690.

[30] Z. Jebroni, J. A. Afonso, e B. Tidhaf, «Home Energy Monitoring System Towards Smart Control of Energy Consumption», *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng. LNICST*, vol. 269, pp. 40–53, 2019.

[31] J. A. Afonso, F. Rodrigues, P. Pereira, H. Gonçalves, e J. L. Afonso, «Wireless monitoring and management of energy consumption and power quality events», *Lect. Notes Eng. Comput. Sci.*, vol. 2217, pp. 338–343, 2015.

[32] Texas Instruments, «C2000 Real-Time Control MCUs». [Em linha]. Disponível em: http://www.ti.com/microcontrollers/c2000-real-time-control-mcus/overview.html?DCMP=Delfino&HQS=delfino#. [Acedido: 16-Mai-2019].

[33] Texas Instruments, «TMDSDOCK28335 TMS320F28335 Experimenter Kit | TI.com». [Em linha]. Disponível em: http://www.ti.com/tool/TMDSDOCK28335. [Acedido: 18-Nov-2019].

[34] Texas Instruments, «TMS320F28335 Delfino™ 32-bit MCU with 150 MIPS, FPU, 512 KB Flash, EMIF, 12b ADC | TI.com». [Em linha]. Disponível em: http://www.ti.com/product/TMS320F28335. [Acedido: 16-Out-2019].

[35] Texas Instruments, «CCSTUDIO-C2000 Code Composer Studio (CCS) Integrated Development Environment (IDE) for C2000 Microcontrollers | TI.com». [Em linha]. Disponível em: http://www.ti.com/tool/CCSTUDIO-C2000. [Acedido: 16-Out-2019].

[36] WAVESHARE, «0.91 inch OLED Module User Manual». [Em linha]. Disponível em: https://www.waveshare.com/w/upload/1/10/0.91inch_OLED_Module_User_Manual_EN.pdf. [Acedido: 15-Mai-2019].

[37] «Módulo Relé 1 canal 5V compatível com arduino - botnroll.com». [Em linha]. Disponível em: https://www.botnroll.com/pt/digital/455-modulo-rele-8-canais-5v-em-linha.html?search_query=rele&results=109. [Acedido: 29-Out-2019].

[38] «Módulo Sensor de Voltagem». [Em linha]. Disponível em: https://www.electrofun.pt/sensores-arduino/modulo-sensor-de-voltagem. [Acedido: 29-Out-2019].

[39] DFROBOT, «Gravity: Analog AC Current Sensor (20A) - DFRobot». [Em linha]. Disponível em: https://www.dfrobot.com/product-1486.html. [Acedido: 29-Out-2019].

[40] Texas Instruments, «CD405xB CMOS Single 8-Channel Analog Multiplexer/Demultiplexer With Logic-Level Conversion». [Em linha]. Disponível em: http://www.ti.com/lit/ds/symlink/cd4051b.pdf. [Acedido: 30-Out-2019].

[41] E. F. Philhower, «ESP8266 core for Arduino». [Em linha]. Disponível em: https://github.com/esp8266/Arduino. [Acedido: 10-Dez-2018].

[42] Tzapu, «ESP8266 WiFi Connection manager with web captive portal». [Em linha]. Disponível em: https://github.com/tzapu/WiFiManager. [Acedido: 12-Fev-2019].

[43] B. Blanchon, «ArduinoJson: Efficient JSON serialization for embedded C++». [Em linha]. Disponível em: https://arduinojson.org/. [Acedido: 02-Abr-2019].

[44] I. Grokhotkov e M. Molinari, «OTA Update - ESP8266 Arduino Core». [Em linha]. Disponível em: http://arduino.esp8266.com/Arduino/versions/2.0.0/doc/ota_updates/ota_updates.html. [Acedido: 05-Set-2019].

[45] J. Gähwiler, «MQTT library for Arduino». [Em linha]. Disponível em:

https://github.com/256dpi/arduino-mqtt. [Acedido: 10-Mar-2019].

[46] Adafruit Industries, «Adafruit GFX graphics core library». [Em linha]. Disponível em: https://github.com/adafruit/Adafruit-GFX-Library. [Acedido: 16-Jul-2019].

[47] Adafruit Industries, «Arduino library for SSD1306 monochrome 128x64 and 128x32 OLEDs». [Em linha]. Disponível em: https://github.com/adafruit/Adafruit_SSD1306. [Acedido: 16-Jul-2019].

[48] Adafruit Industries, «Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors». [Em linha]. Disponível em: https://github.com/adafruit/DHT-sensor-library. [Acedido: 25-Ago-2019].

[49] G. Hudson, «GitHub - openenergymonitor/EmonLib: Electricity monitoring library - install in Arduino IDE's libraries folder then restart the IDE». [Em linha]. Disponível em: https://github.com/openenergymonitor/EmonLib. [Acedido: 30-Out-2019].

[50] «Gravity_Analog_AC_Current_Sensor__SKU_SEN0211_-DFRobot». [Em linha]. Disponível em: https://wiki.dfrobot.com/Gravity_Analog_AC_Current_Sensor__SKU_SEN0211_. [Acedido: 30-Out-2019].

[51] Espressif Systems, «Arduino core for the ESP32». [Em linha]. Disponível em: https://github.com/espressif/arduino-esp32. [Acedido: 20-Mar-2019].

[52] N. Kolban, «esp32-snippets/BLE C++ Guide». [Em linha]. Disponível em: https://github.com/nkolban/esp32-snippets/blob/master/Documentation/BLE C%2B%2B Guide.pdf. [Acedido: 12-Mar-2019].

[53] Nordic Semiconductor ASA, «nRF Connect for Mobile». [Em linha]. Disponível em: https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp. [Acedido: 10-Ago-2019].

[54] Eclipse Foundation, «Eclipse Mosquitto». [Em linha]. Disponível em: https://mosquitto.org/. [Acedido: 04-Jan-2019].

[55] MariaDB Foundation, «About MariaDB - MariaDB.org». [Em linha]. Disponível em: https://mariadb.org/about/. [Acedido: 02-Fev-2019].

[56] The Apache Software Foundation, «The Apache HTTP Server Project». [Em linha]. Disponível em: https://httpd.apache.org/. [Acedido: 22-Fev-2019].

[57] M. Jayaratne *et al.*, «phpMyAdmin». [Em linha]. Disponível em: https://www.phpmyadmin.net/.

[Acedido: 15-Mai-2019].

[58] R. Light, «paho-MQTT version 3.1.1 client class». [Em linha]. Disponível em: https://pypi.org/project/paho-mqtt/. [Acedido: 15-Jul-2019].

[59] M. Rodrigues e I. Naoki, «PyMySQL: Pure Python MySQL Client». [Em linha]. Disponível em: https://github.com/PyMySQL/PyMySQL. [Acedido: 15-Mai-2019].

[60] I. Harvey, «Python interface to Bluetooth LE on Linux». [Em linha]. Disponível em: https://github.com/IanHarvey/bluepy. [Acedido: 15-Jul-2019].

[61] V. Monteiro, J. P. Carmo, J. G. Pinto, e J. L. Afonso, «A flexible infrastructure for dynamic power control of electric vehicle battery chargers», *IEEE Trans. Veh. Technol.*, vol. 65, n. 6, pp. 4535–4547, Jun. 2016.

[62] P. Jahoda, «MPAndroidChart». [Em linha]. Disponível em: https://github.com/PhilJay/MPAndroidChart. [Acedido: 01-Out-2019].

[63] P. Singh, «gson: A Java serialization/deserialization library to convert Java Objects into JSON and back». [Em linha]. Disponível em: https://github.com/google/gson. [Acedido: 01-Out-2019].

[64] Eclipse Foundation, «Eclipse Paho Android Service». [Em linha]. Disponível em: https://www.eclipse.org/paho/clients/android/. [Acedido: 01-Out-2019].

[65] G. Combs, «Wireshark». [Em linha]. Disponível em: https://www.wireshark.org/. [Acedido: 17-Out-2019].