INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




SERENITY Home Automation

FILIPE ANDRÉ AMORIM DA SILVA
Outubro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

SERENITY

Home Automation

Filipe André Amorim Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Computacionais

Orientador: Nuno Filipe Escudeiro

Porto, Outubro 2016

Resumo

O projeto apresentado neste relatório foi desenvolvido no âmbito da Unidade Curricular de Tese/Dissertação (TMDEI), do Mestrado em Engenharia Informática (MEI), na área de especialização em Sistemas Computacionais, do Instituto Superior de Engenharia do Porto (ISEP), como requisito para a obtenção do grau de mestre em Engenharia Informática. Este trabalho foi proposto pelo próprio autor deste documento, e tem como objetivo o desenvolvimento de um produto de automação de casas ou edifícios no geral.

Este relatório descreve a estrutura e o planeamento, do desenvolvimento e avaliação deste trabalho, que tem como foco principal, a criação de um sistema que permita monitorizar e controlar aparelhos pertencentes a casas ou edifícios, de forma a proporcionar segurança e comodidade ao cliente alvo, ou seja, os seus proprietários.

É também apresentada uma extensa análise de valor, que aborda o processo de transformação de projeto em produto, definindo como este apresentará valor para os possíveis clientes, e quais as estratégias de negócio a seguir para ter sucesso no mercado.

O desejo de tornar uma ideia, realidade, e o interesse pessoal no projeto e nas áreas técnicas que este aborda, criaram a motivação necessária para prosseguir com o seu desenvolvimento e evolução, sempre com a missão em vista, de tornar as atividades do dia-a-dia mais acessíveis e eficientes.

Palavras-chave: Automação de Casas, Eficiência, Acessibilidade, Segurança, Comunicação

Abstract

The project presented in this report was developed within the course of Thesis/Dissertation (TMDEI), from the Masters in Computer Engineering (MEI), in the area of specialization in Computer Systems, from the Porto's Superior Institute of Engineering, as a requirement to achieve the master's degree in Computer Engineering. This work was proposed by the writer himself, and it has as a main objective the development of a home automation product.

This report describes the structure and the planning, for the development and evaluation of this project, which main focus is the creation of a system that allows the monitoring and the control of appliances within houses or buildings, in order to provide security and commodity to the target customer, namely, their owners.

It is also presented an extensive value analysis that addresses the process of transforming a project into a product, by defining how this will present value to the possible customers and the business strategies to follow to be successful in the market.

The desire for turning an idea into reality and the personal interest in the project and in the technical fields that it addresses, generated the necessary motivation to proceed with the project's development and evolution, always with the mission in mind, of making day-to-day activities more accessible and efficient.

Keywords: Home Automation, Efficiency, Accessibility, Security, Android, Communication

Agradecimentos

Gostaria de dirigir os meus sinceros agradecimentos ao meu orientador, por me ter acompanhado ao longo da escrita deste relatório.

À minha família, por ter apoiado a ideia deste projeto e ajudado indiretamente no seu financiamento.

Ao meu colega e amigo Filipe Couto, por me ter ajudado a resolver vários problemas ao longo deste trabalho e por se mostrar sempre disponível para ajudar.

E, por fim, à minha companheira Maria João, pelo seu apoio e compreensão durante todo este trabalho.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

6LoWPAN *IPv6 over Low power Wireless Personal Area Networks*

AES *Advanced Encryption Standard*

AHP *Analytic Hierarchy Process*

ANJE Associação Nacional de Jovens Empresários

API *Application Programming Interface*

BLE *Bluetooth Low Energy*

GHz *Gigahertz*

GND *Ground*

GPIO *General Purpose Input/Output*

GSM *Global System for Mobile communications*

HTTP *Hypertext Transfer Protocol*

I^2C *Inter-Integrated Circuit*

IEEE *Institute of Electrical and Electronics Engineers*

ISEP Instituto Superior de Engenharia do Porto

JSON *JavaScript Object Notation*

Kbps Quilobit por Segundo

MAC *Media Access Control*

Mbps Megabit por Segundo

MEI Mestrado em Engenharia Informática

MHz *Megahertz*

MISO *Master In Slave Out*

MOSI *Master Out Slave In*

NET Novas Empresas e Tecnologias, S.A.

PV	Proposta de Valor
RDBMS	*Relational Database Management System*
REST	*Representational State Transfer*
RX	*Receive*
SCK	*Serial Clock*
SCL	*Serial Clock*
SDA	*Serial Data*
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language*
SS	*Slave Select*
SSL	*Secure Sockets Layer*
TMDEI	Tese / Dissertação / Estágio
TX	*Transmit*
USB	*Universal Serial Bus*
WWW	*World Wide Web*

1 Introdução

1.1 Contexto

Este projeto originou da necessidade de controlar um portão elétrico, que dá acesso a uma fábrica de um negócio de família. Os comandos utilizados para o propósito eram de curto alcance e havia a necessidade de controlo a partir de qualquer parte do mundo.

Para isto foi criado um pequeno e simples protótipo que foi instalado em Abril de 2015, o qual esteve em funcionamento durante um ano, com menos de 5 paragens relatadas.

A constante utilização deste protótipo permitiu descobrir alguns problemas no seu uso a longo prazo e, igualmente importante, permitiu expandir os horizontes desta ideia e começar a pensar em algo que se pudesse tornar num produto e que conseguisse abranger um mercado maior.

1.2 Problema

Tendo em conta o enquadramento dado no ponto anterior, foi então possível definir um problema mais geral no que toca ao controlo de aparelhos de casas ou edifícios de trabalho.

O problema principal pode então ser dividido em duas características: segurança e acessibilidade.

Segurança:

Cada vez mais as pessoas sentem a necessidade de aumentar a sua segurança, não tanto de uma forma física mas sim psicológica através da capacidade de controlo e supervisão de certos elementos. Quando estas estão em casa, principalmente a dormir, ou quando vão trabalhar, ter a possibilidade de controlar ou monitorizar todas as entradas possíveis, a temperatura ou o movimento dentro de certas divisões, dá-lhes uma certa serenidade pois poderão agir no preciso momento em que alguma situação ocorre (e.g. Invasão de

propriedade através da abertura de uma porta; Incêndio numa certa divisão; Movimento no quarto do bebé quando este está a dormir; Alguma janela que ficou aberta por esquecimento ou foi aberta quando não era suposto.).

Acessibilidade:

Tudo o que aumente a acessibilidade das pessoas e lhes facilite as tarefas do dia-a-dia será sempre bem aceite por estas. Quer seja para poderem calendarizar certas ações como a rega automática do jardim ou o ligar e desligar de luzes de presença, quer seja para poderem controlar no momento certos aparelhos elétricos como um portão ou as persianas de um quarto, todas estas possibilidades fornecem eficiência e poupam tempo precioso às pessoas, permitindo-lhes ter um jardim bonito ou abrir o portão de casa, sem esforço e estando em qualquer parte do mundo.

1.3 Análise de Valor

Os contributos deste projeto estão aliados à criação de uma primeira fase de um produto, cujo objetivo é dar às pessoas paz de espírito sobre as suas propriedades, assim como total controlo e capacidade de monitorização sobre as mesmas, quer estejam presentes ou em qualquer outro lugar.

1.4 Abordagem Preconizada

Um sistema como o que se pretende criar vai precisar de um gestor interno, um dispositivo capaz de receber instruções por parte de uma pessoa, agir sobre elas e comunicar com os dispositivos locais ou remotos, para que estes atuem sobre os componentes da casa. A este dispositivo dar-se-á o nome de controlador/*HUB*.

O **controlador/*HUB*** deverá ser compacto, capaz de suportar um sistema operativo com base em UNIX, capaz de se ligar a redes *Wireless* 802.11 e ter entradas e saídas digitais para interação com componentes com fios. Um exemplo seria o *Raspberry Pi 2* Modelo B.

Os **Dispositivos Remotos (sem fios)** que funcionarão como atuadores/sensores serão essencialmente constituídos por 3 partes:

- Um **emissor-recetor** que permite a comunicação bidirecional com o *HUB* (o qual também possuirá um igual). Para isto poderá ser utilizado um circuito que permita a comunicação através da utilização de um protocolo específico.
- Um ou mais **componentes eletrónicos**, como por exemplo sensores de movimento, sensores de temperatura, relés (atuadores elétricos), entre outros.

- Um **dispositivo embebido** (*embedded device*), que seja pequeno, que necessite de pouca energia e que seja capaz de se acoplar ao emissor-recetor e aos componentes eletrónicos. Um exemplo seria o *Arduino Leonardo Pro Micro*.
- Uma **bateria/pilha** capaz de manter o dispositivo remoto ligado, no mínimo, por um ou dois anos.

Os **Dispositivos Locais (com fios)** serão apenas componentes eletrónicos, como os utilizados nos Dispositivos Remotos, pois serão ligados diretamente ao *HUB*, o qual tratará da sua gestão.

Para a interação das pessoas com o sistema em si será criada uma **Aplicação *Android*** que permitirá ter controlo total sobre os dispositivos, incluindo configuração, gestão de notificações, agendamento e criação de cenários. A aplicação móvel comunicará com o *HUB* através da internet ou, da rede local onde este estiver inserido.

Em relação às escolhas feitas para cada uma destas partes do projeto, estas serão discutidas aprofundadamente no capítulo 3 de Avaliação de Abordagens.

2 Contexto e Estado da Arte

2.1 Conceitos de Negócio

Este subcapítulo tem a intenção de fazer uma breve introdução a alguns conceitos inerentes ao projeto em questão para que a leitura deste relatório e o enquadramento do projeto possam ser feitos de uma forma mais percetível.

2.1.1 Automação de Casas/Edifícios

A automação de casas, que pode também abranger edifícios no geral, tem como conceito base a centralização de atividades maioritariamente diárias e rotineiras, que podem passar pelo controlo de luzes, eletrodomésticos, dispositivos de segurança e outros sistemas, de forma a proporcionar às pessoas, conforto, facilidade de acesso, eficiência e segurança, onde quer que estas estejam.

A centralização é atingida quando se consegue o controlo dos variados mecanismos elétricos, utilizando um único ponto de acesso, como um computador ou um *smartphone*. Através deste ponto, é possível fazer diretamente o controlo e a gestão dos mecanismos (e.g. Controlar a cor e o estado das lâmpadas *hue* da marca *Philips.*) ou então, utilizar um aparelho intermédio que é capaz de receber os pedidos das pessoas e redirecioná-los para os mecanismos corretos, assim como proporcionar uma panóplia de funcionalidades, como o agendamento e os alertas (e.g. Gestão de diferentes aparelhos através do produto *Samsung Smart Things*.).

Este último exemplo será a abordagem que este projeto irá seguir.

2.1.2 Protocolos de Comunicação

Seja através de um *smartphone* ou de um dispositivo controlador/gestor, para se poder comunicar com os dispositivos remotos, é necessário que ambos suportem o mesmo protocolo de comunicação.

Um protocolo de comunicação permite uniformizar a forma como uma máquina comunica com outra. Funciona como uma linguagem bem estruturada que, quando conhecida por ambas, permite que as mensagens transmitidas sejam corretamente recebidas e interpretadas.

Exemplos destes protocolos vão desde os mais conhecidos como o *Wi-Fi* e o *Bluetooth* até outros menos difundidos pela sociedade em geral, como o *ZigBee*, o *6LowPan*, entre outros.

2.1.3 Polling

Polling é um processo de obtenção de dados ou do estado de um sistema, por parte de uma identidade, onde os pedidos são feitos de uma forma cíclica. O ciclo de um pedido corresponde ao tempo de espera até ao próximo pedido.

A utilização de *polling* é muito controversa visto que o ciclo de um pedido vai ser sempre inversamente proporcional ao esforço necessário (e.g. Ocupação de tempo de processamento; Gastos em tráfego.). Ou seja, quanto menos tempo se puder esperar para obter os dados de um sistema, maior sobrecarga se irá colocar sobre este.

2.2 Intervenientes e Requisitos

Neste projeto, os requisitos, funcionais e não funcionais, serão distribuídos pelos três principais componentes, que são: a Aplicação *Android*, o Controlador/*HUB* e os Dispositivos Remotos. Os Dispositivos fisicamente ligados ao *HUB* terão requisitos associados que serão da responsabilidade deste.

Cada componente terá pelo menos um interveniente associado, que será responsável pelas ações a ele atribuídas.

2.2.1 Intervenientes

Os principais **intervenientes** nos componentes referidos são os seguintes:

- **Utilizador:** Alguém a quem foi permitido o acesso a todas/certas funcionalidades da aplicação *Android* de forma a conseguir utilizar o produto. Se o Controlador/HUB estiver instalado numa casa então, normalmente, será alguém que a habita mas, se estiver num edifício de uma empresa poderá ser alguém que lá trabalha.
- **Administrador:** Conta de utilizador permanente, que tem acesso total ao sistema e que deve ser utilizada apenas para questões administrativas.
- **Sistema do Controlador/*HUB*:** Ator completamente autónomo, capaz de manter os seus serviços a funcionar corretamente caso ocorra uma anomalia. Funciona como

gestor principal de todo o produto e serve de intermediário entre os pedidos do Utilizador e as funcionalidades dos Dispositivos.

- **Sistema dos Dispositivos Remotos:** Ator completamente autónomo, cujo objetivo é disponibilizar a leitura de um aspeto ambiental ou do estado do Dispositivo ou, o controlo sobre um mecanismo elétrico, quando pedido pelo *HUB*.

2.2.2 Requisitos Funcionais

Para melhor explicar e demonstrar os requisitos funcionais, serão utilizados diagramas de casos de uso para cada um dos componentes.

Aplicação *Android*:

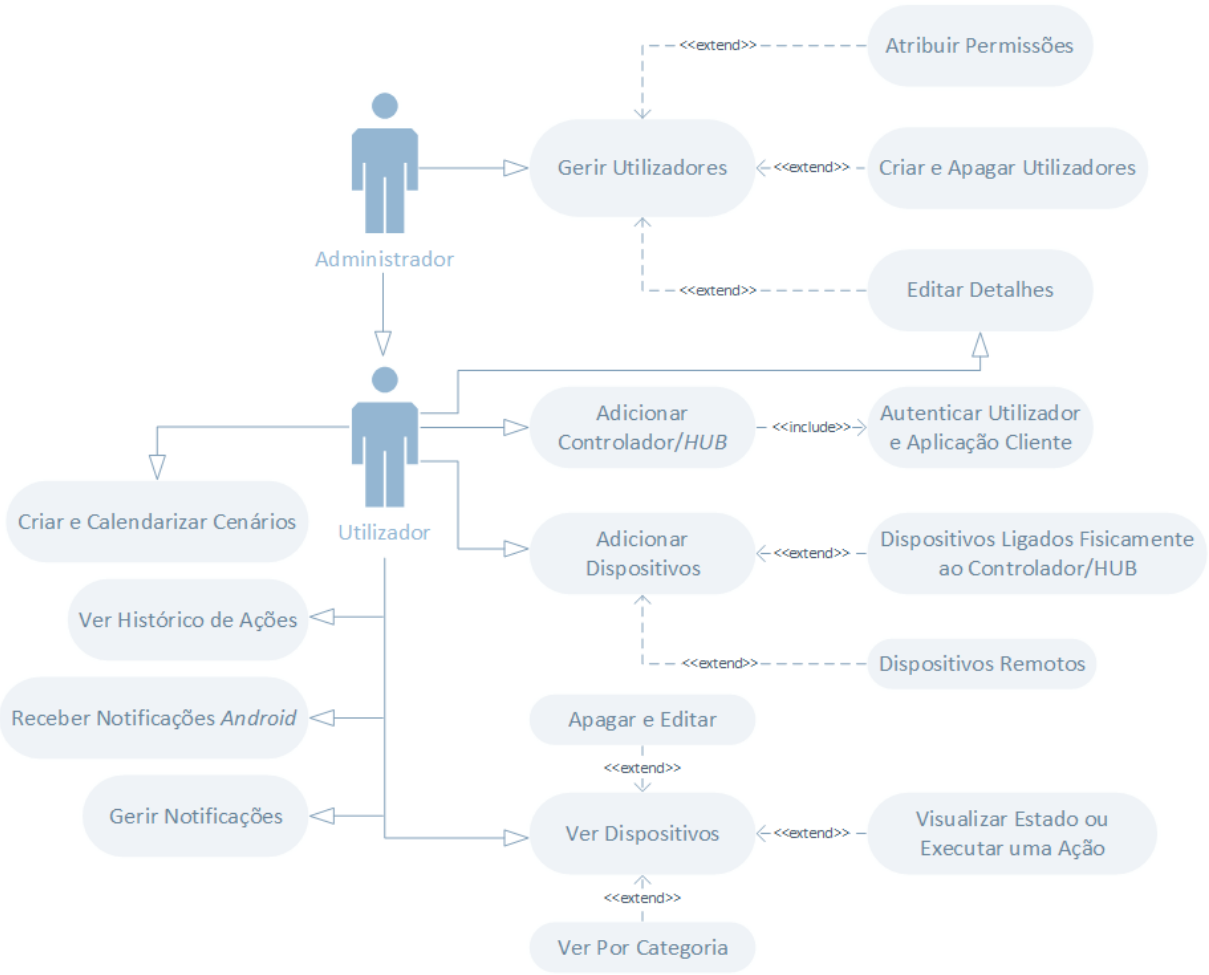


Figura 1 – Casos de Uso da Aplicação *Android*

Controlador/*HUB*:

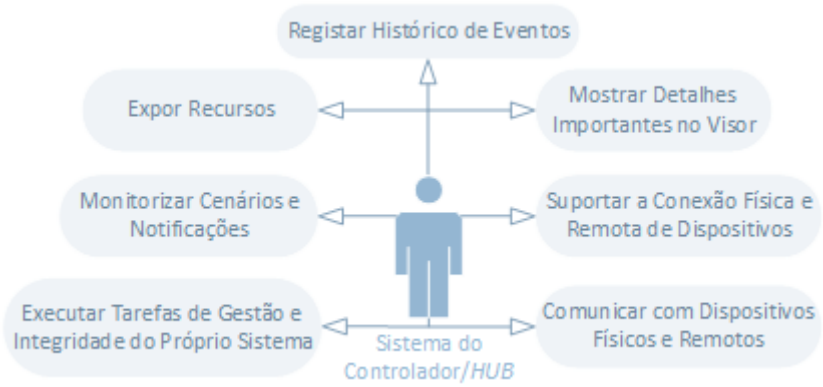


Figura 2 – Casos de Uso do Controlador/*HUB*

Dispositivos Remotos:

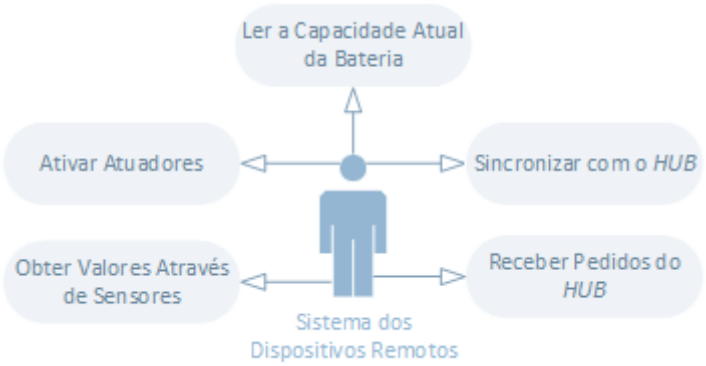


Figura 3 – Casos de Uso do Sistema dos Dispositivos Remotos

2.2.3 Requisitos Não Funcionais

Os requisitos não funcionais são uma parte fundamental para a sobrevivência de um projeto e, especificamente para este, serão apresentados os mais importantes relativamente a cada um dos componentes.

Aplicação *Android*:

- **Atenção a Falhas (*Fault Awareness*)** – A aplicação deve estar ciente de problemas, como a perda de ligação com o Controlador/*HUB*, e agir em conformidade com estes, dando também feedback ao utilizador caso seja necessário.
- **Usabilidade (*Usability*)** – A aplicação deve ser de fácil utilização e de rápida aprendizagem.
- **Tempo de Resposta (*Response time*)** – O tempo de resposta da aplicação pode ser afetado por 4 pontos: a própria aplicação, o dispositivo onde está a ser executada, a

comunicação com o exterior e o Controlador/*HUB*. Portanto, o seu desempenho será sempre subjetivo. No entanto, este deverá ser o mais baixo possível para dar ao utilizador uma experiência suave. Para quantificar o tempo ideal, é possível utilizar como base a lei de *Akscyn* (Akscyn et al. 1988), para sistemas de Hipertexto, que, traduzindo de (Eastgate 1999), diz o seguinte: "Sistemas de Hipertexto devem demorar cerca de ¼ de segundo para avançarem de um ponto para outro. Se este tempo for maior, as pessoas podem-se distrair; Se for muito menor, as pessoas podem não se aperceber que o conteúdo apresentado mudou."

Controlador/*HUB*:

- **Atenção a Falhas e Recuperação do Sistema (*Fault Tolerance and System Recovery*)** – Existe a necessidade de ter protocolos em execução para atender ao maior número de situações errôneas possíveis (e.g. Falha de energia; Falha de internet; Corrupção da base de dados; Más práticas de desligar o sistema por parte do utilizador.), para que o sistema consiga recuperar no menor tempo possível.

- **Custo/Preço do Produto (*Price*)** – Este é um fator importante a considerar quando se desenha um produto. Como o objetivo deste projeto é a criação de um produto com um preço competitivo, quanto mais baixo este poder ser, melhor.

Dispositivos Remotos:

- **Durabilidade (*Durability*)** – Neste caso, refere-se à durabilidade da bateria que acompanhará estes dispositivos. Esta deverá conseguir durar um período entre um a dois anos para que a pessoa não necessite de a substituir com muita frequência, proporcionando-se assim confiança no sistema e despreocupação com estes dispositivos.

- **Custo/Preço do Produto (*Price*)** – Pelas mesmas razões apresentadas no Controlador/*HUB*.

2.3 Restrições Existentes

A inexistência de um investidor no projeto impossibilita a obtenção de produtos concorrentes para teste e análise e, coloca restrições na compra dos componentes necessários para o desenvolvimento deste projeto.

2.4 Análise de Valor

Neste subcapítulo, a análise de valor é abordada com detalhe, utilizando para isso algumas questões (Silva 2016) pré-definidas.

2.4.1 Proposta de Valor

Pergunta:

Explique a necessidade de uma Proposta de Valor (PV) bem definida num negócio. Justifique a sua resposta com base na definição de PV, definição de valor e valor percebido.

Resposta:

Uma proposta de valor (Nicola 2016b) indica um conjunto de produtos ou serviços de uma empresa, que representam valor para o cliente. Esta deve-se focar na qualidade, no custo e na rapidez com que o valor é apresentado ao mercado, o qual deve ser específico e de tamanho apropriado para os recursos da empresa.

A proposta de valor identifica bem o cliente para o qual quer proporcionar valor (Nicola 2016a) e faz uma boa representação do último, transmitindo-o com clareza e de uma forma não técnica. Ao garantir a excecionalidade do valor conseguirá aumentar as probabilidades de captar o interesse dos clientes e assim conseguir a permanência do produto/serviço no mercado.

Porém, é necessário ter em atenção como o valor é percebido (Nicola 2016a) pelas várias entidades associadas à PV. A perceção de valor feita por parte do consumidor é uma avaliação da utilidade do produto ou serviço tendo por base uma relação entre os benefícios obtidos e os sacrifícios necessários.

Diferentes clientes podem ter uma perceção de valor diferente sobre um mesmo produto ou serviço pois associam-lhe diferentes benefícios e sacríficos. É necessário ter isto em conta para tentar encontrar uma certa harmonia entre o que as diferentes entidades valorizam mais e menos.

Em suma, uma proposta de valor bem definida, apesar de requerer esforço, pensamento e várias tentativas ao longo do seu processo de criação, demonstrar-se-á bastante útil pois limita os erros e consegue definir com rigor o valor que está a propor e a quem o está a dirigir.

2.4.2 Benefícios ou Sacrifícios para os Possíveis Clientes

Pergunta:

De acordo com o tema da sua tese, seja um produto/serviço ou mesmo uma componente, qual o valor (benefícios/sacrifícios) para os possíveis clientes que o (a) possam utilizar? Justifique convenientemente a sua resposta enquadrando os vários benefícios/sacrifícios numa perspetiva longitudinal de valor.

Resposta:

O valor que se pretende que o cliente obtenha do produto aqui proposto tem por base uma troca entre os benefícios que vai receber e os sacrifícios que vai fazer. (Nicola 2016a)

Os principais benefícios podem ser vistos das seguintes formas:

- Do ponto de vista geral do produto: Utilidade, Durabilidade, Adaptabilidade, Confiança e Eficiência, reduzindo o Tempo e Esforços gastos em tarefas manuais.
- Em termos de funcionalidades: Monitorização e Controlo, Mobilidade e Acessibilidade Global.
- Sobre a relação criada: Serviço de Suporte e Ajuda.
- Do ponto de vista psicológico: Serenidade, Segurança e Conveniência.

O principal sacrifício a ser feito pelo cliente será o preço do produto que este terá de pagar. No entanto, como o valor percebido é diferente para todos, poderão existir características no produto que o cliente pode considerar como sacrifícios, como por exemplo, o facto de este não interagir com outros produtos concorrentes, semelhantes.

Numa perspetiva longitudinal de valor, o produto, estando no mercado, passará por quatro fases de perceção de valor por parte do cliente:

- **Antes da Compra:** Neste momento, o cliente tem sempre um **desejo** do que pretende. Por exemplo, poderá desejar que este produto consiga controlar absolutamente tudo o que este tiver em sua casa. No entanto, tem noção de que isso não corresponde com a realidade e, por isso, **espera** que este consiga controlar pelo menos uma certa variedade de aparelhos.
- **No Momento da Compra:** Quando o cliente decide adquirir o produto, vai estar em contacto com três tipos de valores diferentes. O **valor da transação**, que neste caso equivale a um sacrifício que é o preço, ou seja, o dinheiro que o cliente paga pelo produto que vai comprar. O **valor de aquisição**, que é o valor que o cliente está a receber em forma de produto e que pode ser equiparado a alguns dos benefícios do mesmo, como por exemplo as suas funcionalidades e algumas características como a utilidade e a confiança. Por último, o **valor de troca,** que é a conjunção do valor da própria moeda com a influência que a sua posição no mercado tem sobre o produto.
- **Após a Compra:** Este momento constata todo o período de utilização do produto adquirido e pode ser dividido, em termos de valor, de três formas. O **valor recebido**: corresponde ao produto em si e às suas funcionalidades. O **valor de desempenho**: refere-se a quão bem o produto resolveu o problema para o qual foi comprado. O **valor entregue**: corresponde a todos os benefícios que foram adquiridos ao longo da

utilização do produto, como por exemplo, a durabilidade, a eficiência, o serviço de suporte e as vantagens ao nível psicológico.

- Após a Utilização ou Em Fase de Disposição: Nesta última fase, reflete-se sobre o **valor de redenção**, ou seja, se os sacrifícios feitos foram bem empregues. Neste caso, se a utilização do produto compensou o dinheiro que foi gasto.

2.4.3 Possíveis Cenários de Negócio

Pergunta:

Quais os possíveis cenários de negócio que poderia ter?

Resposta:

Existem quatro tipos de cenários, que se devem considerar, que podem ocorrer numa situação de negócio e que dão origem a resultados bastante distintos. Estes cenários são identificados por **negociação sem entendimento** (situação perde-perde), por **negociação distributiva** (situação ganha-perde), por **negociação integrativa** (situação ganha-ganha) e por **negociação de compromisso** (situação onde é definido um meio-termo entre ambas as partes, capaz de satisfazer minimamente as duas ofertas iniciais).

Para melhor entendimento da negociação distributiva e da negociação integrativa, estas podem ser comparadas através da seguinte tabela. (Nicola 2016c; Carnevale & Pruitt 1992)

Tabela 1 – Comparação entre Negociação Distributiva e Integrativa

Negociação Distributiva	**Negociação Integrativa**
Uma parte ganha, outra perde.	Ambas as partes saem a ganhar.
Assume-se que não existe uma solução que satisfaça ambas as partes e por isso se o resultado for abaixo do ponto de resistência, a negociação é terminada.	Existência de várias soluções para um problema, com o foco de gerar sempre uma situação de ganha-ganha.
Não existe uma relação prévia, nem a intenção de criar uma.	Desenvolvimento de relações de longo termo.

No caso do produto que será fruto deste projeto, dar-se-á mais enfâse a negociações integrativas pois, será do interesse de todos criar uma relação em que ambas as partes sintam que saem a ganhar de igual forma. Quer seja a negociação que se tem com o cliente ao definir um preço de produto adequado ao mercado alvo e ao garantir uma relação duradoura com este, quer sejam as negociações que se têm com os fornecedores, onde é do nosso interesse continuar a receber matéria-prima a preços aceitáveis e, onde é do interesse destes manter

um cliente estável e em quem possam confiar para manter uma relação que persista ao longo do tempo.

2.4.4 Modelo de Negócio de Canvas

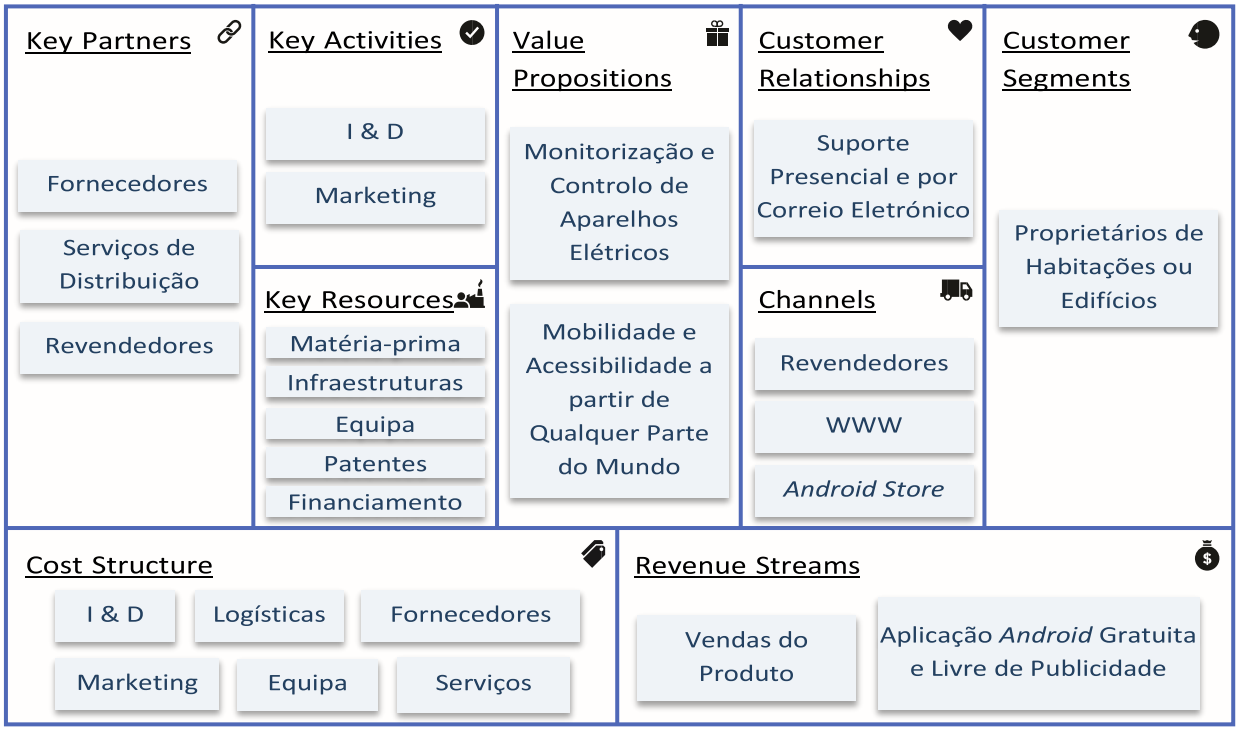
Pergunta:

Utilize o modelo de negócio de *Canvas* para descrever a sua ideia de negócio.

Resposta:

A ideia de negócio, do projeto aqui abordado, pode ser visualizada através do *canvas* apresentado na seguinte tabela (Strategyzer 2016) e compreendida com mais detalhe (Nicola 2016c) nas explicações dadas posteriormente para cada um dos seus campos.

Tabela 2 – Modelo de Negócio *Canvas*



Customer Segments – Para quem se está a criar valor.

Proprietários que necessitem de monitorizar ou controlar elementos de uma ou mais habitações/edifícios, de uma forma móvel e remota através de dispositivos *Android*.

Value Propositions – Propostas de valor que vão atrair clientes.

HUB (controlador), Dispositivos Sensoriais e de Atuação, Aplicação *Android*:

- Monitorização e Controlo de aparelhos elétricos, de um ou mais edifícios.
- Mobilidade e Acessibilidade a partir de qualquer parte do mundo utilizando a aplicação *Android*.

Channels – Forma como o valor chega aos clientes.

Revendedores: Utilização de grandes superfícies como *Leroy Merlin*, *Brico Depot* ou *AKI* para a venda do produto.

WWW: Utilização de um *website* para exposição e venda do produto.

Loja de Aplicações *Android*: Para descarregamento gratuito da aplicação para o dispositivo *Android*.

Customer Relationships – Tipo de relação que se cria com os clientes.

Suporte Presencial e por Correio Eletrónico: Utilização de correio eletrónico e posicionamento de pessoas qualificadas nos postos de venda para informações e aconselhamento acerca do produto.

Revenue Streams – Forma como o modelo de negócio obtém valor.

Vendas do Produto: Através do *website* ou dos revendedores. O pagamento do mesmo poderá ser feito através de sistemas de multibanco (*website* e revendedores), transferência bancária (*website*) e dinheiro (revendedores).

Key Activities – Atividades que devem ser feitas de forma constante para que o modelo de negócio funcione.

Investigação e Desenvolvimento:

- De dispositivos, com capacidade sensorial ou de atuação, tendo em vista, sempre que possível, a economia de energia.
- Do dispositivo controlador e das suas capacidades de comunicação.
- Da aplicação móvel, com o foco na facilidade de interação com o utilizador.

Marketing:

- Análise de mercado
- Publicidade

Key Resources – Ativos indispensáveis para fazer o modelo de negócio funcionar.

Componentes Eletrónicos Baratos: É necessário um fornecimento de componentes eletrónicos base, para a construção dos dispositivos controladores e dos dispositivos

sensoriais e de atuação, a um preço razoável, para que o preço final do produto consiga ser competitivo.

Estabelecimentos de Montagem: Locais onde o produto é montado e embalado, ficando assim pronto para ser distribuído aos revendedores.

Equipa Multidisciplinar: Equipa de trabalho constituída por especialistas das diferentes áreas necessárias (e.g. Software, *Web-Design*, Design de Produto, Eletrónica, Marketing e Recursos Humanos) para o bom funcionamento do modelo de negócio.

Proteção de Propriedade Industrial: Utilização de patentes para a proteção de invenções e obtenção de exclusividade na exploração económica dos produtos ou processos protegidos.

Apoios Financeiros: Utilização de *crowdfunding* (e.g. *Kickstarter*, *Indiegogo*, *Patreon*), (pré-) incubadoras (e.g. Associação Nacional de Jovens Empresários (ANJE); Novas Empresas e Tecnologias, S.A. (NET).) ou investidores particulares, para alavancar o projeto até ao ponto de auto sustentabilidade.

Key Partnerships – Rede de fornecedores e parceiros essenciais que garantem o funcionamento do modelo de negócio.

Fornecedores: São necessários fornecedores de componentes eletrónicos para apoiar os estabelecimentos de montagem e por isso serão estrategicamente escolhidos tendo em conta as suas posições geográficas.

Serviços de Distribuição: Para possibilitar o transporte dos produtos desde os estabelecimentos de montagem até aos revendedores ou até ao cliente final.

Revendedores: Os revendedores são mais do que um canal de entrega de valor. A parceria com estes permite obter feedback constante assim como aumentar a exposição do produto através das suas técnicas de marketing.

Cost Structure – Principais custos provenientes do modelo de negócio.

Investigação e Desenvolvimento

Marketing

Logísticas

Equipa de Trabalho

Fornecedores

Serviços de Montagem e Distribuição

2.4.5 Análise, Modelação e Quantificação da Criação de Valor

Pergunta:

De que forma pode analisar/modelar e/ou quantificar a criação de valor? Justifique convenientemente a sua resposta com base na análise de modelos conceptuais e/ou métodos quantitativos.

Resposta:

A criação de valor pode ser analisada e quantificada através da utilização de métodos analíticos de tratamento de informação. Estes métodos auxiliam na tomada de decisão e na escolha de hipóteses mais apropriadas.

No contexto deste trabalho, os métodos a serem abordados são: a Teoria dos Jogos e a Análise Hierárquica (*Analytic Hierarchy Process* (*AHP*)). (Nicola 2016d)

A **Teoria de Jogos** serve fundamentalmente para otimizar resultados tendo em conta um certo conjunto de condições. Tem muita utilidade em jogos onde existem oponentes, como por exemplo o *Bridge*, assim como para lidar com estratégias de tomadas de decisão em ambientes empresariais, como por exemplo decidir o preço de um produto.

O dilema do prisioneiro é um exemplo muito conhecido para explicar a teoria de jogos e pode ser adaptado a qualquer situação (e.g. económica, de negócio). Utilizando o caso da decisão do preço de um produto por parte de duas empresas que competem num mesmo mercado, é possível gerar quatro hipóteses.

Tabela 3 – Dilema do Prisioneiro Aplicado à Competitividade de Preços

	Preço Baixo	**Preço Alto**
Preço Baixo	Menor lucro mas, competitividade mantida.	A de preço baixo terá mais lucro que a de preço alto.
Preço Alto	A de preço baixo terá mais lucro que a de preço alto.	Maior lucro e competitividade mantida.

Tendo em conta a tabela previamente apresentada consegue-se perceber em termos de lucro quem fica a ganhar e quem fica a perder tendo por base as condições de definição do preço (baixo ou alto).

Será sempre mais proveitoso colocarem ambas um preço alto, mas manterão a competitividade se colocaram ambas um preço baixo.

No caso de uma das empresas colocar um preço baixo e a outra um preço alto então a primeira, teoricamente, terá mais proveito pois conseguirá atrair mais clientes.

Outra particularidade deste jogo é a repetição, ou seja, se ambas as empresas voltarem a ter outra situação de decisão de preços, desta vez terão em conta as opções tomadas por ambas, em encontros anteriores. (Dixit & Nalebuff 2008)

Suponha-se que, após vários encontros, ambas as empresas detêm um histórico de colocarem um preço alto, apesar de não manterem nenhum tipo de cooperação, e sentem-se confortáveis com esta estratégia. Neste caso, pode-se dizer que neste jogo entre ambas as empresas foi encontrado o **Equilíbrio de *Nash***, ou seja, não existe nenhum incentivo que faça mudar os seus comportamentos. (Porto Editora 2016)

A **Análise Hierárquica**, ou **AHP**, foca-se mais em encontrar a alternativa mais adequada de entre várias soluções, ao contrastar os elementos de cada uma, com os objetivos/requisitos que se pretendem atingir. Para isto são utilizados processos para estruturar o problema, quantificar os elementos e avaliar as soluções. A quantificação dos elementos é feita através da atribuição de um nível de importância, de 1 a 9.

O processo de estruturação serve para por em perspetiva todos os aspetos do problema e consiste nos seguintes quatro passos:

1. **Problema:** Definir bem o problema a solucionar. Por exemplo: Para qual dos sistemas operativos móveis deverá ser desenvolvida a aplicação que se enquadra no projeto *Serenity Home Automation*?
2. **Fatores ou Critérios:** Definir os elementos mais importantes e diferenciadores que se devem considerar neste processo. Por exemplo:
    a. Tempo de Existência;
    b. Proprietário/Código Aberto;
    c. Tamanho do Mercado;
    d. Variedade de dispositivos;
    e. Facilidade e Rapidez de Desenvolvimento.
3. **Processo de Tomada de Decisão:** Utilizar os critérios previamente definidos para criar pares de comparação e assim ajudar a tomar uma decisão. Por exemplo:
    a. Proprietário/Código Aberto vs. Facilidade e Rapidez de Desenvolvimento;
    b. Variedade de dispositivos vs. Facilidade e Rapidez de Desenvolvimento;
    c. Tempo de Existência vs. Tamanho do Mercado.
4. **Alternativas:** Tendo em conta a quantificação dos critérios e os resultados provenientes das comparações entre os mesmos, é possível associar uma

percentagem a cada uma das alternativas, indicando assim a que mais se adequa para resolver o problema indicado. Por exemplo, utilizando dados fictícios:

a. *IOS* = 27%;

b. *Android* = 60%;

c. Windows Phone = 13%;

A solução mais adequada ao problema exemplo indicado nesta análise, seria a utilização do sistema operativo *Android* visto ser o que possui uma percentagem maior, ou seja, é aquele que mais se identifica com os critérios mais importantes. De notar que esta percentagem seria obtida através da utilização de cálculos de matrizes de comparação par a par. (Nicola 2016d)

2.5 Estado da Arte

2.5.1 Soluções e Abordagens Existentes

O produto que será fruto deste projeto não será uma completa novidade para o mercado e como tal, já existem diversas soluções, com diferentes características que apontam para um mesmo objetivo, que é proporcionar automação completa ou parcial, de residências ou edifícios no geral.

De seguida é apresentada uma lista de soluções juntamente com as características mais importantes de cada uma.

- ***Vera*:** É provavelmente o sistema de automação de casas com maior popularidade. Utiliza o protocolo de comunicação *Z-Wave* e é capaz de interagir com mais de 1500 produtos compatíveis.
- ***Archos Smart Home*:** Utiliza o protocolo de comunicação *Bluetooth Low Energy* e garante o seu funcionamento com os seus 5 dispositivos remotos proprietários.
- ***Samsung Smart Things*:** Permite controlar dispositivos que estejam ligados à Rede Local, por *ZigBee*, *Z-Wave* e através de *Cloud-to-Cloud*. Suporta oficialmente mais de 200 dispositivos.
- ***OpenSprinkler*:** Permite o controlo e automação de sistemas de irrigação através de ligações físicas com válvulas de água.
- ***SESAME*:** Permite o controlo e automação de quase qualquer tipo de fechadura através de um *design* patenteado. Realiza as suas comunicações através de *Bluetooth Low Energy*.

2.5.2 Tecnologias Relevantes

As tecnologias mais relevantes para este projeto, que serão abordadas com mais detalhe no próximo capítulo, são:

- *Node.js*: é um ambiente de desenvolvimento de servidores *web* em *Javascript*.
- *MySQL*: é um sistema de gestão de bases de dados relacionais (*RDBMS*), de código aberto, muito popular. É uma parte central do *LAMP/WAMP*, o qual permite o desenvolvimento de *websites*.
- *MongoDB*: é uma base de dados não relacional orientada a documentos, que se tem tornado cada vez mais popular devido à sua capacidade de escalabilidade.
- *SQLite*: é um *RDBMS*, baseado em ficheiros e que não precisa de um servidor para ser utilizado.
- *Bluetooth Low Energy* (*BLE*): é um protocolo de comunicação utilizado maioritariamente em dispositivos móveis de pequena dimensão e de contacto próximo com o utilizador, onde o consumo energético tem de ser reduzido para aumentar o seu período de utilização entre recargas.
- *ZigBee*: é um protocolo de comunicação orientado à automação de casas, com baixos consumos de energia, bem estabelecido no mercado.
- *6LoWPAN*: é um protocolo de comunicação baseado em IPv6, com baixos consumos de energia, e com grande potencial na área de automação de casas.
- *RFM69 (LowPowerLab)*: é um protocolo de comunicação que define a estrutura dos pacotes de mensagens enviados através de rádio frequências, com baixos consumos de energia e longo alcance.

3 Avaliação de Abordagens

Neste capítulo serão avaliadas e selecionadas as abordagens mais apropriadas para cada um dos componentes do projeto tendo em conta certas grandezas específicas para cada um destes.

3.1 Controlador/*HUB*

3.1.1 Plataforma Base

Tendo em conta o preço, a acessibilidade e a extensa comunidade de suporte, o *Raspberry Pi* 2 Modelo B será a plataforma base que constituirá o *HUB*. Ao contrário de outras plataformas como o *BeagleBone Black* ou o *Banana Pi*, que têm funcionalidades semelhantes, mas cuja comunidade é mais reduzida e os preços são mais altos.

Em suma, um *Raspberry Pi* é uma plataforma de desenvolvimento em forma de um mini computador, com capacidades de processamento bastante mais reduzidas em comparação com um computador dito normal, mas que consegue ser bastante útil e versátil tendo em conta o seu tamanho, a possibilidade de instalação de praticamente qualquer sistema operativo *Linux*, a possibilidade de interação com componentes eletrónicos externos como sensores, botões, mini ecrãs, entre outros, e tudo isto consumindo uma quantidade de energia bastante reduzida.

Figura 4 – Figura Representativa de um *Raspberry Pi* 2 Modelo B (raspberrypi.org)

3.1.2 Sistema Operativo

De entre os vários sistemas operativos *Linux* capazes de serem instalados num *Raspberry Pi*, o melhor para este projeto será o *Raspbian* pois é o sistema operativo oficial da fundação *Raspberry*. Isto significa que terá uma comunidade igualmente grande o que facilita a resolução de problemas e faz com que a evolução deste seja contínua e melhorada. Esta grande exposição faz com que sejam criadas muitas ferramentas para aproveitar todo o potencial deste computador, as quais serão úteis para o desenvolvimento de projetos como o apresentado neste relatório.

3.1.3 Ambiente de Desenvolvimento

O ambiente de desenvolvimento escolhido para este componente será *Node.js*, utilizando *Javascript*, pelas seguintes razões (Anon n.d.):

- Foi desenhado de raiz para suportar ações assíncronas e ativadas por eventos;
- É de rápido processamento, visto que o seu núcleo (*V8 Javascript Runtime Engine*) foi escrito em C;
- Consegue suportar ligações concorrentes e não bloqueantes, mesmo executando sobre apenas uma *Thread*, pois utiliza um sistema de *Callbacks*;
- Possui um grande sistema de módulos, o *npm*, o que facilita o desenvolvimento através da utilização de bibliotecas já preparadas para certos fins.

3.1.4 Base de Dados

Em termos de bases de dados, tendo em conta a curva de aprendizagem necessária e o posicionamento no ranking mundial (http://db-engines.com/en/ranking), foram equacionadas as seguintes:

- *MySQL* – Existe conhecimento e experiência por parte do autor. Esta BD encontra-se na 2º posição do ranking mundial.
- *MongoDB* – *Não* existe conhecimento nem experiência por parte do autor. Esta BD encontra-se na 4º posição do ranking mundial e na 1º posição no ranking de bases de dados orientadas a documentos.
- *SQLite* – Existe conhecimento e experiência por parte do autor. Esta BD encontra-se na 9º posição do ranking mundial.

Logo à partida encontram-se duas grandes diferenças, duas delas são relacionais (*MySQL* e *SQLite*) e uma é orientada a documentos, ou seja, não relacional (*MongoDB*). Para se

perceberem as diferenças entre estes dois conceitos e, aproveitando as análises de (Serra 2015) e de (Gangji 2013), pode-se construir a seguinte tabela de comparação:

Tabela 4 – Comparação entre Bases de Dados Relacionais e Orientadas a Documentos

	Relacional	**Orientada a Documentos**
Representação de Informação	Tabelas, Colunas e Linhas.	Coleções de Documentos *JSON*.
Pesquisas	Utilização de SQL (Linguagem de Pesquisa Estruturada) através da formulação de um comando que é interpretado pelo sistema de base de dados.	Pesquisas através de objetos. Não existe uma linguagem para ser interpretada. Curva de aprendizagem maior, mas no fim é mais intuitivo.
Relações	Existência de operações *JOIN* que permitem pesquisas através de múltiplas tabelas.	Não suporta *JOINs* nem relações entre tabelas (chaves estrangeiras). Suporte de tipos de dados multidimensionais que permitem colocar documentos dentro de outros.
Transações	Atómicas, o processo tem de ser levado até ao fim ou então revertido ao início. Possibilidade de conter múltiplas operações dentro de uma transação e de as reverter (*roll back*) como se de apenas uma operação se tratasse.	Não suporta transações. Suporta operações atómicas.
Definição do Esquema	Necessidade de definição de tabelas e colunas antes de guardar informação. Cada linha da tabela tem de conter as mesmas colunas.	Precisamente o contrário, não há necessidade de definir um esquema definitivo. É possível adicionar novos campos aos documentos *JSON* sem primeiro fazer alterações. É utilizada uma abordagem *schema-on-read*, que permite uma não vinculação a uma estrutura predeterminada.

	Relacional	**Orientada a Documentos**
Normalização	Baixa flexibilidade se forem seguidos os padrões de normalização.	Tendo em conta o tipo de Relações e Transações, é necessário otimizar o esquema tendo em conta a forma como a aplicação vai aceder aos dados.
Desempenho	Baixo desempenho se for utilizado um mapeamento do tipo objeto-relacional (que tem por objetivo reduzir o distanciamento da programação orientada a objetos) e se os dados não forem indexados corretamente.	Tendo em conta os sacrifícios feitos nas Relações e nas Transações é possível obter um melhor desempenho. Continua a ser necessário indexar os dados mas, só a partir de um certo volume é que se nota a diferença.
Escalabilidade	Escala bem, para cima (*scale-up*), num único servidor, no entanto, quando a sua capacidade é atingida, é necessário escalar para fora (*scale-out*) e é aqui que a complexidade associada começa a causar problemas.	Escala bem para fora, de forma horizontal, aproveitando-se da possibilidade de distribuição e replicação de dados por um conjunto de servidores. É de baixa complexidade e possibilita a alteração de servidores sem provocar a inatividade do sistema.

Após a análise destes dois tipos de bases de dados, optou-se por utilizar uma que seja orientada a documentos, neste caso, a *MongoDB*, pelas seguintes razões:

- A sua integração com o ambiente de desenvolvimento será mais facilitada, tendo em conta a técnica de pesquisas através de objetos;
- O desenvolvimento será menos rígido por causa da liberdade de definição do esquema;
- Tendo em conta que se consegue obter um melhor desempenho através dos sacrifícios apresentados nas Relações e nas Transações, pode-se considerar este tipo de base de dados, como sendo mais leve, o que o torna mais apropriado para o tipo de plataforma base que será utilizada;
- Por último, como o conhecimento sobre bases de dados não relacionais é cada vez mais procurado em temos de mercado de trabalho, considera-se relevante escolher uma que esteja bem posicionada.

3.2 Plataforma Base dos Dispositivos Remotos

De entre variadíssimos sistemas de desenvolvimento, existem uns que permitem colocar um sistema operativo, normalmente utilizados para tarefas complexas, outros que têm o seu próprio sistema operativo e que são focados em tarefas mais específicas, e ainda aqueles que não utilizam um sistema operativo e que simplesmente executam consecutivamente um bloco de código.

Tendo em conta a simplicidade destes dispositivos, cujo objetivo é receber um comando e executar uma ação, que pode ser a leitura de um sensor ou a ativação de um atuador utilizando pontos de entrada e saída, optar-se-á por escolher o último sistema referido.

Dentro destes sistemas, a comunidade que mais se sobressai é a relativa a *Arduinos*. O *Arduino* é uma plataforma aberta e como tal existem imensas variedades de placas com diferentes propósitos e enquadramentos.

Visto que, para este projeto, se necessita que os Dispositivos Remotos sejam pequenos e que consumam pouca energia, os protótipos serão realizados tendo por base o *Arduino Pro Mini* e o *Arduino Pro Micro*.

3.3 Emissor-Recetor

Este não é propriamente um componente em si, mas fará parte de dois e terá um papel muito importante neste projeto. O emissor-recetor encontrar-se-á tanto no *HUB* como nos Dispositivos Remotos e servirá de meio de comunicação entre os dois. Para que isto seja possível, tem de existir uma linguagem em comum, ou seja, um protocolo de comunicação utilizado por ambos, para que estes consigam perceber o que cada um transmite.

Para um projeto como este, de automação de edifícios, os protocolos mais propícios a serem escolhidos são: *Wi-Fi*, *Bluetooth Low Energy* (*BLE*), *ZigBee*, *Z-Wave*, *Low Power Wireless Area Networks* (*6LoWPAN*), *RFM69* e *LoRaWAN*. (Rathnayaka et al. 2011)

A consideração de várias variáveis como, a integração no contexto existente, a complexidade da rede e a capacidade de esta se estender, o custo monetário e a distância de comunicação máxima, permitirá escolher o protocolo mais indicado às necessidades do projeto.

Estas necessidades estipulam que o Emissor-Recetor deverá ser capaz de comunicar a distâncias superiores a 100 metros, não deverá utilizar a frequência de 2.4 GHz, deverá ser de baixo custo e ter um consumo mínimo de energia. Isto permitirá que o projeto consiga abranger o espaço ocupado em média por uma casa ou até uma pequena-média empresa, evitar conflitos de comunicação sobre uma frequência que já é bastante utilizada neste contexto por parte de redes *Wi-Fi* e *Bluetooth*, garantir um preço competitivo e uma longa duração dos Dispositivos Remotos.

Posto isto, construiu-se a seguinte tabela de comparação, que permite avaliar de uma forma uniforme, todas as variáveis necessárias.

Tabela 5 – Comparação de Protocolos de Comunicação

Protocolo	**Hardware**	**Distância**	**Frequência**	**Velocidade**	**Custo**
Wi-Fi	ESP8266	350 m	2.4 GHz	54 Mbps	9 €
BLE	nRF8001	70 m	2.4 GHz	1 Mbps	30 €
ZigBee	XBee Series 2	1.6k m	2.4 GHz	250 Kbps	40 €
Z Wave	Z-Uno	40 m	EU = 868 MHz US = 908 MHz	40 Kbps	60 €
6LoWPAN	-	-	-	-	-
RFM69	RFM69HCW	500 m	EU = 433/868 MHz US = 915 MHz	300 Kbps	12 €
LoRaWAN	RFM96W	2 km	EU = 433/868 MHz US = 915 MHz	50 Kbps	17 €

O *Wi-Fi* está muito bem estabelecido, é bastante acessível e possui uma grande simplicidade na gestão da rede. Sendo um protocolo que é utilizado para fornecer internet às pessoas, faz com que a sua presença crie uma pegada enorme sobre a frequência 2.4 GHz.

O *BLE* é uma grande evolução dos primórdios desta tecnologia porém, sempre foi desenhada para comunicações de curto alcance, o que limita a sua utilização neste contexto. (Link Labs 2016)

O *ZigBee*, o *Z-Wave* e o *6LoWPan* são protocolos otimizados para um contexto de automação, o que lhes garante bons mecanismos de eficiência de energia, de transmissão de mensagens e de definição de topologias de rede complexas.

Tendo em conta que o *6LoWPAN* é um protocolo relativamente recente, a sua disponibilidade em termos de plataformas físicas e de recursos de aprendizagem, é bastante escassa, daí a falta de dados.

O *ZigBee* e o *6LoWPan* são ambos construídos sobre a camada *MAC IEEE 802.15.4*, o que lhes fornece um bom suporte para manterem uma simplicidade e flexibilidade próximas do *Wi-Fi*. (Sarto 2016) Já o *Z-Wave,* sendo um protocolo proprietário, possui uma estrutura própria, tornando mais difícil a sua integração e desenvolvimento.

Os protocolos *RFM69* e *LoRaWAN* são opções de baixo custo similares e têm a vantagem de funcionarem sobre uma frequência abaixo do GHz, o que lhes permite operar melhor num ambiente com obstáculos. O *LoRaWAN* tem a particularidade de possuir uma modulação especial que lhe permite alcançar distâncias muito maiores que o *RFM69*. A desvantagem destas duas tecnologias é o facto de não possuírem mecanismos para a utilização de topologias de rede complexas, tendo estes de ser implementados pelo programador.

Considerando todos estes pontos, concluiu-se que o protocolo mais adequado seria o *RFM69*, devido ao seu baixo custo, ao seu funcionamento numa frequência não congestionada e ao facto de conseguir uma boa distância de comunicação tendo em conta as necessidades do projeto.

4 Desenho da Solução

4.1 Conceptual

Um desenho conceptual da solução proporciona uma vista de alto nível sobre como todos os componentes do projeto interagem uns com os outros. É com este propósito que se apresenta a seguinte figura, que demonstra tais aspetos, os quais serão posteriormente detalhados.

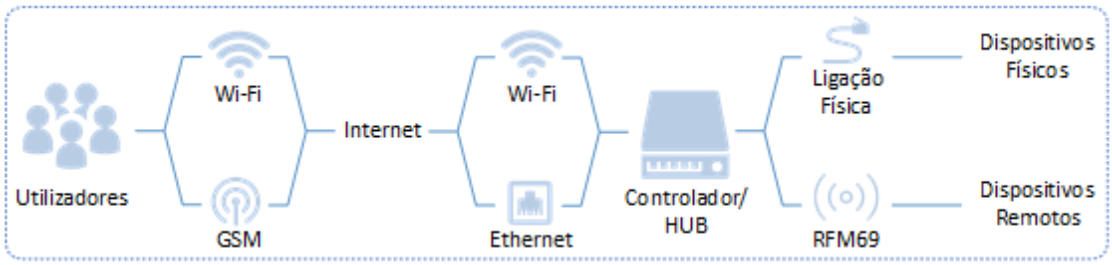


Figura 5 – Modelo Conceptual da Solução

O modelo apresentado traduz-se da seguinte forma:

- Os utilizadores interagem e obtêm feedback através da aplicação *Android*, que pode executar em qualquer dispositivo que suporte este sistema operativo;
- Estes dispositivos *Android*, para poderem comunicar com o *HUB*, precisam de uma ligação à internet, utilizando *Wi-Fi* ou *GSM*, visto que este expõe os seus recursos através da mesma, podendo estar ligado via *Wi-Fi* ou *Ethernet*;
- Assim que um pedido chega ao *HUB*, se for relacionado com a gestão de um dispositivo, é redirecionado adequadamente tendo em conta se o Dispositivo é físico ou remoto;
- Para finalizar o ciclo de um pedido, é enviada uma resposta do Dispositivo para o *HUB*, o qual faz o tratamento necessário, para que depois seja entregue o feedback ao utilizador através da aplicação móvel.

4.2 Arquitetural

O modelo arquitetural tem por objetivo demonstrar quais são os componentes de baixo nível e como estes interagem para proporcionar comunicação e persistência entre a aplicação *Android* e o *HUB*.

Numa primeira fase do projeto, considerou-se estabelecer todas as comunicações entre a aplicação *Android* e o Controlador/*HUB* através de uma *API REST* exposta pelo último. Esta é uma comunicação *half-duplex* onde apenas a aplicação contacta o *HUB*, quando e sempre que necessitar de obter alguma informação ou efetuar algum tipo de tarefa. Para que a informação da aplicação se conseguisse manter o mais atualizada possível, seria então necessário recorrer à técnica de *Polling* abordada na secção 2.1.3.

Para oferecer persistência ao projeto, é utilizada uma base de dados *MongoDB* no *HUB,* onde é guardada toda a informação necessária para o funcionamento do sistema, e uma base de dados *SQLite* na aplicação *Android*, onde são guardadas apenas algumas informações que permitam o correto funcionamento da mesma.

É também possível ver uma comparação detalhada e justificada sobre a escolha da base dados a ser utilizada pelo *HUB*, no capítulo 3 de Avaliação de Abordagens.

Para congregar todas estas tecnologias, e outras, na plataforma do *HUB* é utilizado o ambiente de desenvolvimento *Node.js*, o qual suporta uma panóplia de tecnologias escritas em linguagem *Javascript*.

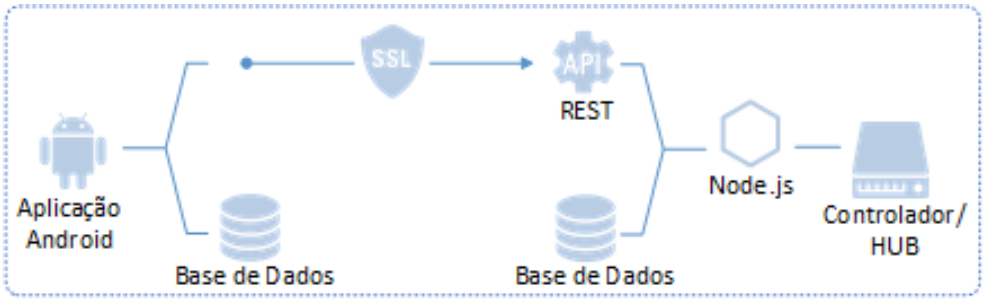


Figura 6 – Primeira Fase do Modelo Arquitetural da Solução

Numa segunda fase do projeto, devido ao fraco desempenho do *Polling* e das restrições apresentadas por uma ligação *half-duplex*, decidiu-se então fazer uma alteração. Os motivos da alteração serão detalhados na secção 5.1.2.4 das Conclusões e Alterações sobre a API REST.

A nova arquitetura consiste agora na limitação do consumo do serviço *REST* apenas para fins de autenticação e introduz um novo tipo de comunicação *full-duplex*, baseado em *Socket.IO*, que permite que a aplicação interaja com o *HUB* e que este interaja de forma independente com as aplicações que estabeleceram uma ligação.

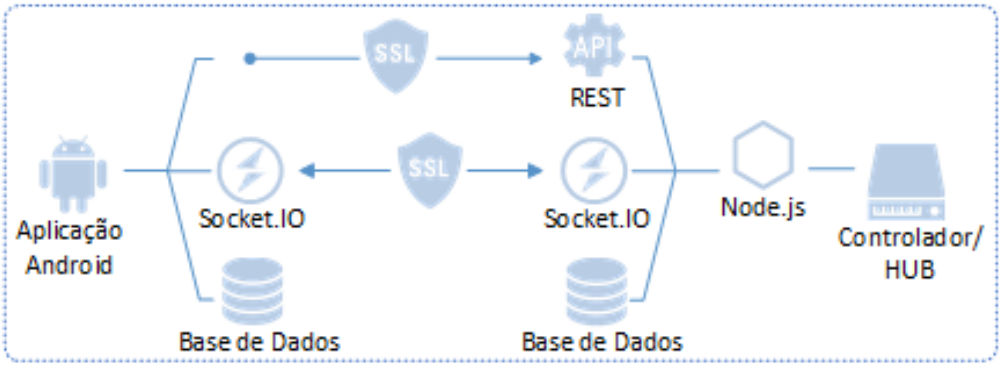


Figura 7 – Segunda Fase do Modelo Arquitetural da Solução

4.3 Detalhado

Nesta secção do desenho da solução serão apresentados processos mais específicos de cada componente e, devidamente detalhados.

4.3.1 Autenticação e Autorização

A maioria dos recursos fornecidos pelo *HUB* estarão protegidos por um sistema de autenticação/autorização.

Este sistema utilizará a especificação *OAuth2* e será baseado em *Tokens*. Na fase de autenticação (*Login*), o utilizador fará uma troca do seu nome de utilizador e palava-passe por um *Token* de Acesso (*Access Token*) e outro de Atualização (*Refresh Token*), tornando este passo o único em que a sua palavra-passe é enviada.

Juntamente com cada pedido a um recurso protegido, será enviado o *Access Token* para que seja utilizado no processo de Autorização.

Para aumentar a segurança, este *Access Token* terá um tempo de vida limitado portanto, quando este expirar, a aplicação cliente será notificada pelo *HUB* para que utilize o *Refresh Token*, para poder requisitar novos *Access e Refresh Tokens*, e então proceder com o pedido original utilizando o novo *Access Token*.

O seguinte diagrama de sequência representa, de uma forma simplificada, o processo previamente detalhado.

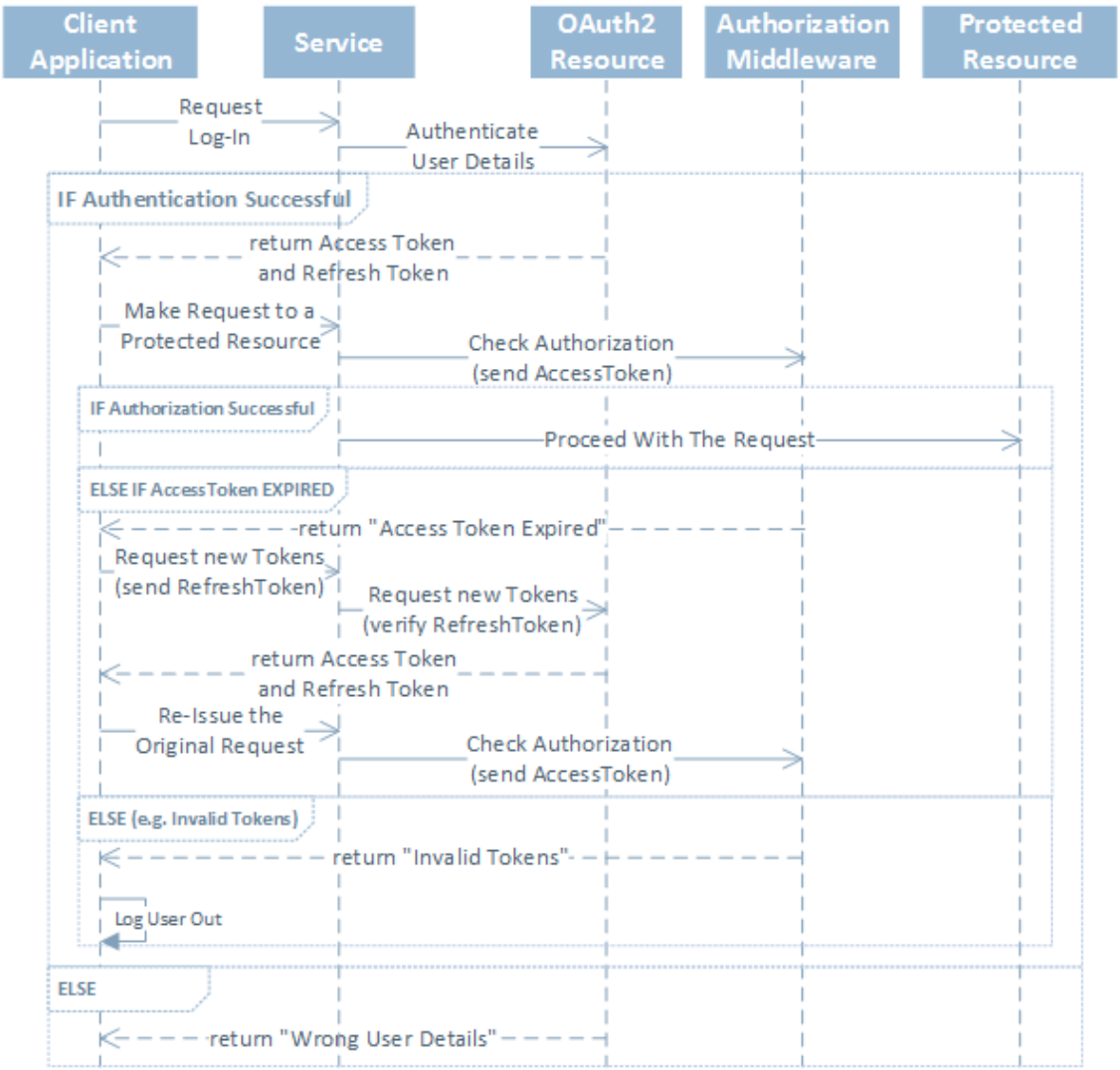


Figura 8 – Diagrama de Sequência da Primeira Fase do Processo de Autenticação/Autorização

Tal como mencionado no Desenho Arquitetural da Solução, na segunda fase manteve-se o processo de autenticação, mas como o resto do sistema de comunicação foi alterado, o processo de autorização também teve de ser reconstruído.

O novo sistema de comunicação é baseado em eventos e trabalha sobre ligações, como vai ser possível verificar com mais detalhe na secção 5.1.2.5, portanto, a abordagem da autorização vai ser consideravelmente diferente porque vão ser autorizadas ligações em vez de pedidos a recursos.

O diagrama de sequência presente na Figura 9 representa as alterações feitas de acordo com esta nova fase.

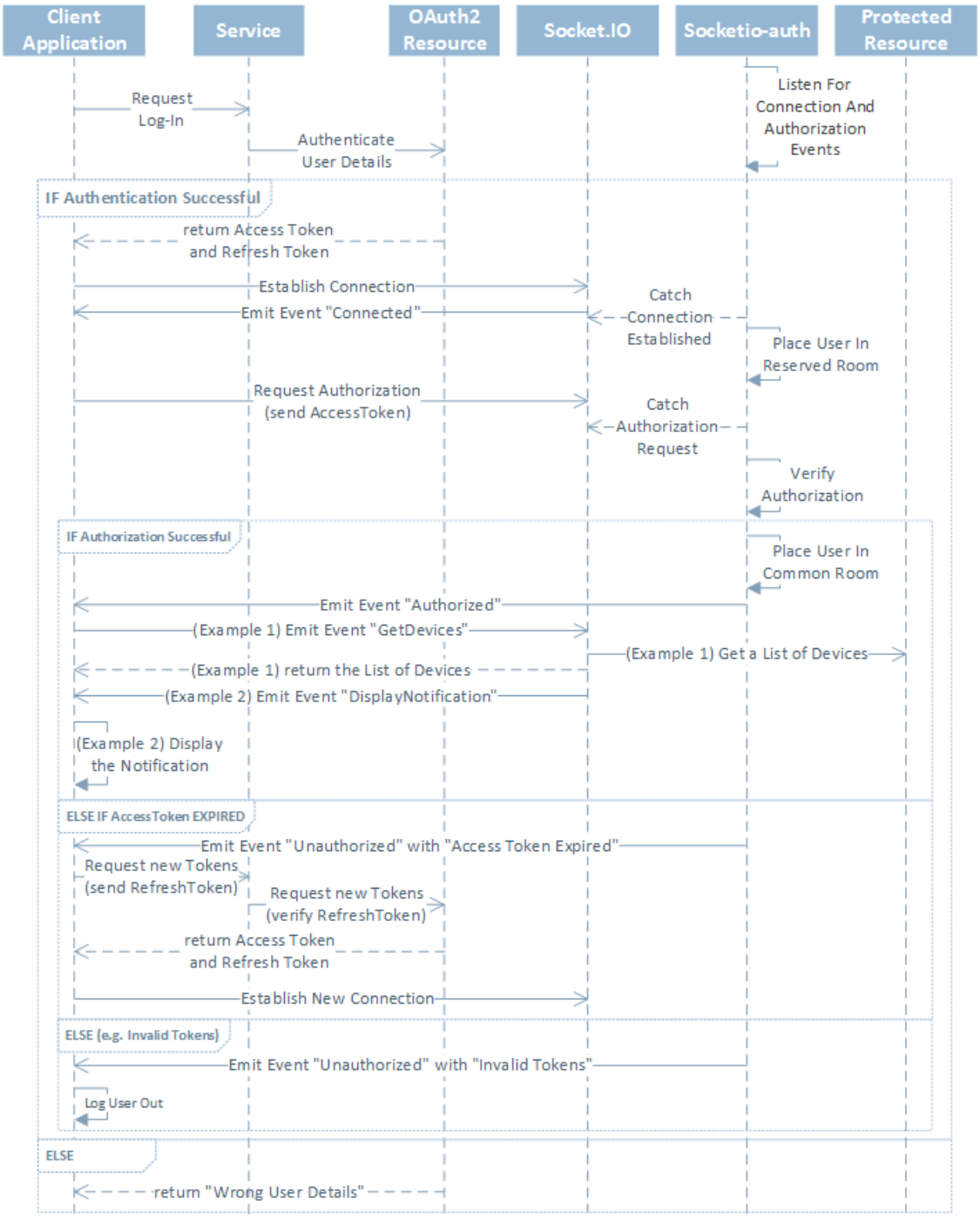


Figura 9 – Diagrama de Sequência da Segunda Fase do Processo de Autenticação/Autorização

O seguinte processo de autorização de uma ligação é possível graças à biblioteca *Socketio-auth* que gere parte deste fluxo de eventos.

Quando um utilizador é autenticado com sucesso, utilizando a aplicação cliente, é criada uma ligação com o *HUB* utilizando a biblioteca *Socket.IO*.

Assim que a ligação é estabelecida, o utilizador é colocado numa sala reservada onde os únicos privilégios que tem são fechar a ligação ou requisitar autorização, utilizando o *Token* de Acesso, para poder ser movido para a sala comum, onde obtém acesso aos restantes privilégios para usufruir dos recursos protegidos do sistema.

Para que não existam ligações não autorizadas permanentemente no sistema a ocupar recursos, a estadia de um utilizador na sala reservada tem um tempo limite, onde, aquando do seu término, a ligação é terminada.

A mesma técnica da primeira fase, relativamente à aplicação de um tempo de vida ao *Token* de Acesso, também é utilizada nesta fase. Quando o utilizador está na sala reservada e tenta obter autorização utilizando um *Token* de Acesso expirado, a ligação é terminada e a aplicação cliente é notificada para que possa requisitar novos *Tokens*, utilizando o *Refresh Token*, e depois abrir uma nova ligação para poder utilizar o novo *Token* de Acesso e obter autorização.

4.3.2 Proteção Contra Aplicações Cliente Não Autorizadas

Para evitar que sejam feitos pedidos aos recursos protegidos do *HUB,* através de aplicações que não as desenvolvidas no âmbito deste projeto, este possuirá um registo de todas as aplicações autorizadas (e.g. Aplicação *Android*, Aplicação *Web*).

Tendo em conta esta camada de segurança, para além dos dados do utilizador, são também enviados o identificador e a chave da aplicação, de onde provém o pedido de Autenticação.

4.4 Base de Dados

Este projeto é composto por duas base de dados, sendo que uma está presente no Controlador/*HUB* e outra na Aplicação *Android*. Não se optou por utilizar a *Cloud* para a persistência deste projeto, visto que, se pretende que exista a possibilidade de utilização do mesmo, por parte de um futuro cliente, sem que exista acesso à Internet.

4.4.1 Controlador/HUB

Tendo em conta que é uma base de dados em *MongoDB*, o diagrama apresentado representa não só as tabelas, mas também os modelos utilizados no *Node.js*. A falta das tradicionais ligações entre tabelas que representariam as chaves estrangeiras deve-se ao facto destas não serem tipicamente usadas neste tipo de base de dados.

Esta base de dados contempla informações sobre:

- ***User***: As contas de utilizador;
- ***Client***: As credenciais de autenticação das aplicações cliente permitidas;

- ***AccessToken*** e ***RefreshToken***: Os *Tokens* para o processo de autorização e de renovação da autenticação;
- ***ControlUnit*** e ***GPIOPin***: As possíveis unidades de controlo (e.g. *Raspberry Pi*, *BeagleBone*), que servem de plataforma base do *HUB*, e o respetivo mapeamento de portas de entrada e saída;
- ***InputType***: Os tipos de dispositivo suportados, do tipo *Input* (e.g. Sensor de Temperatura, Botão ou Sensor de Contacto);
- ***Trigger***: As diferentes formas suportadas para a ativação de um dispositivo do tipo *Output*;
- ***Category***: As categorias a que um dispositivo se pode associar;
- ***Device***: Os dispositivos, remotos ou locais.
- ***LinkedDevice***: Contém registados os dispositivos remotos que completaram o processo de vinculação com o *HUB*, e que, portanto, se apresentam disponíveis para serem geridos pela aplicação *Android*.
- ***History***: Registo de eventos relacionados com a alteração do estado de dispositivos.
- ***Notification***: Conjunto de regras relacionadas com um Dispositivo que permitem que o *HUB* decida se e quando deve de emitir um evento para que a aplicação cliente exiba uma notificação.
- ***UserNotification***: Indica os utilizadores que estão subscritos a uma certa notificação.

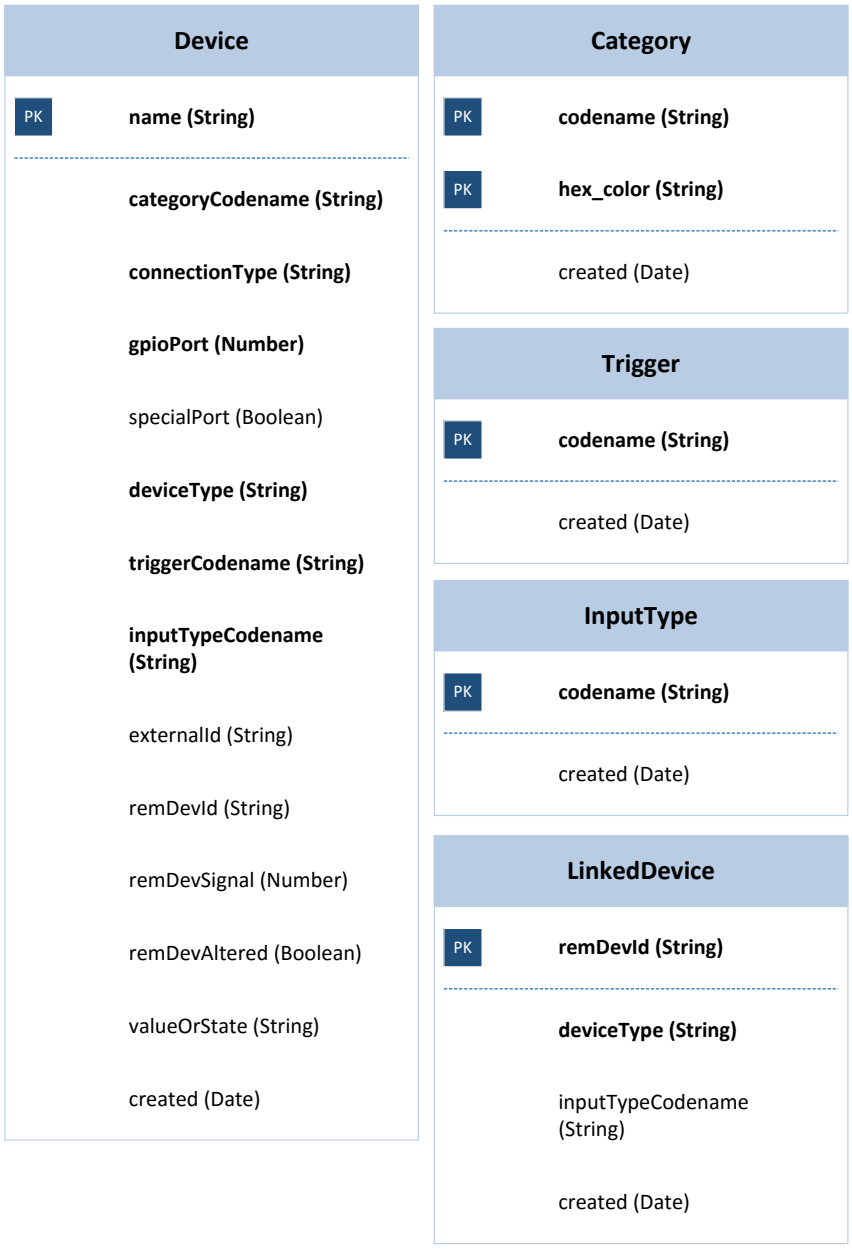


Figura 10 – Modelo de Base de Dados do *HUB* que Suporta a Gestão de Dispositivos

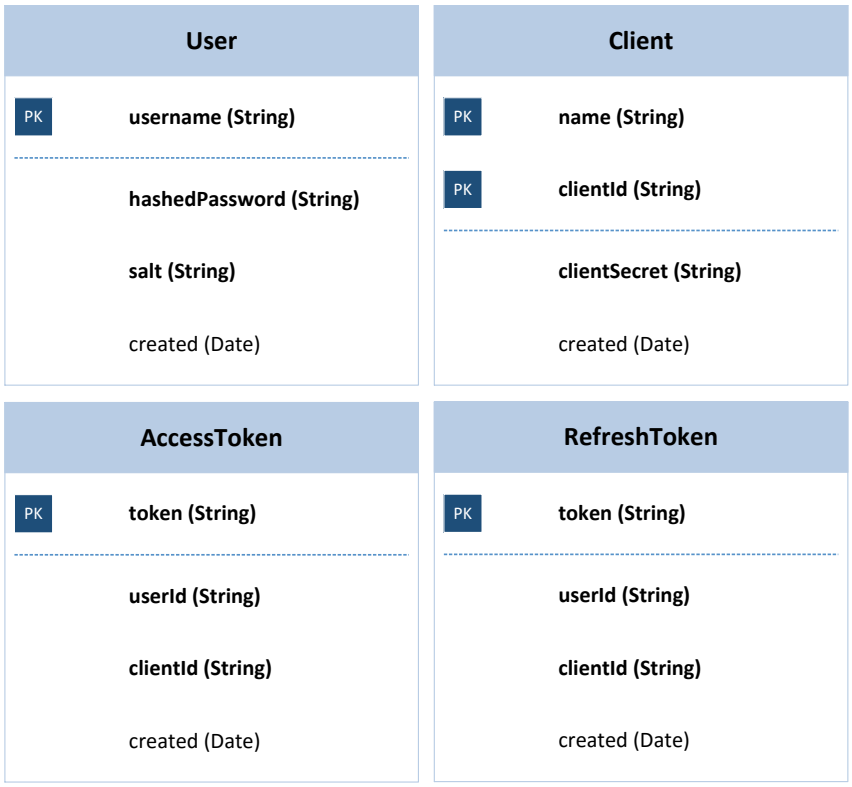


Figura 11 – Modelo de Base de Dados do *HUB* que Suporta o Sistema de Autenticação e Autorização

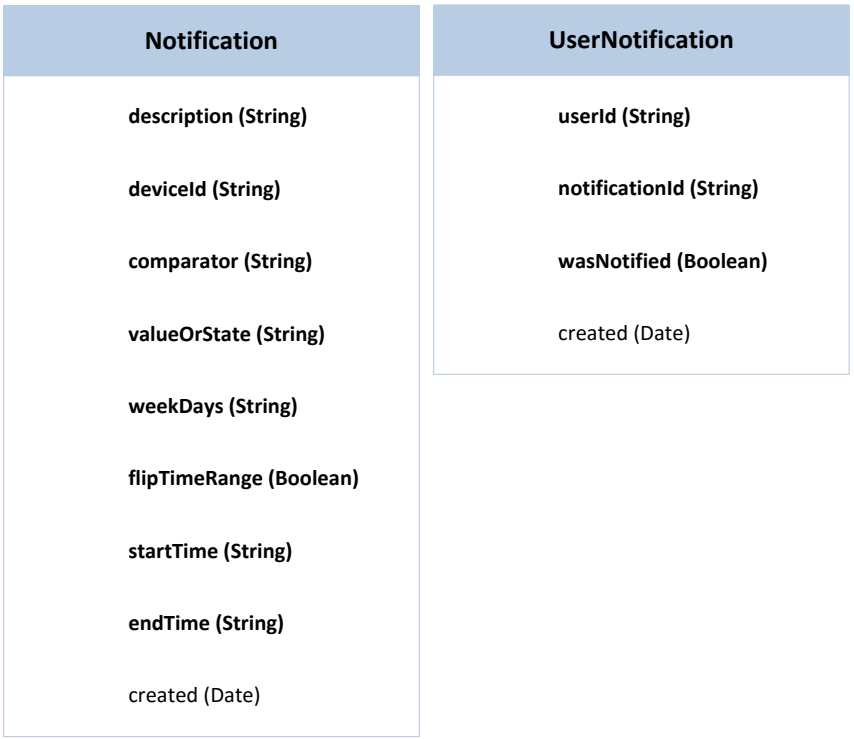


Figura 12 – Modelo de Base de Dados do *HUB* que Suporta a Gestão de Notificações

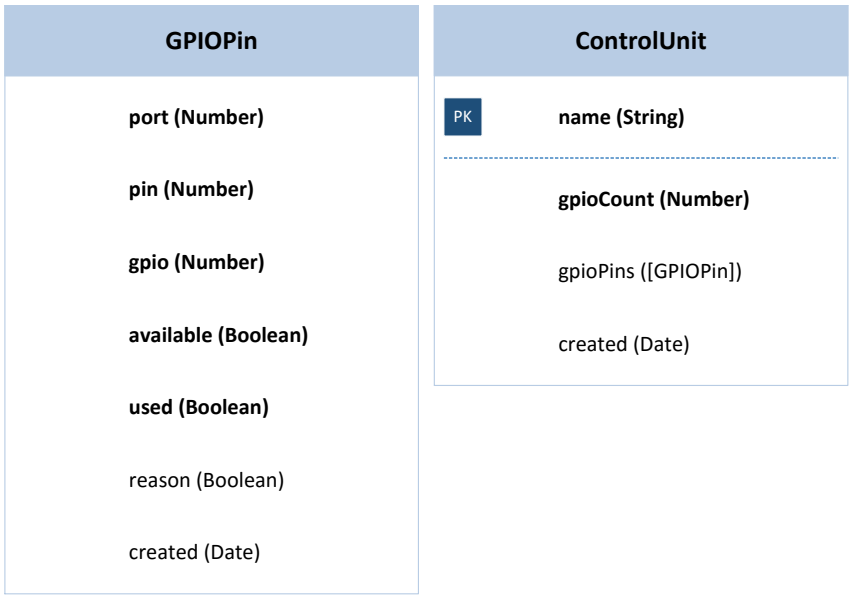


Figura 13 – Modelo de Base de Dados do *HUB* que Representa a sua Estrutura Física

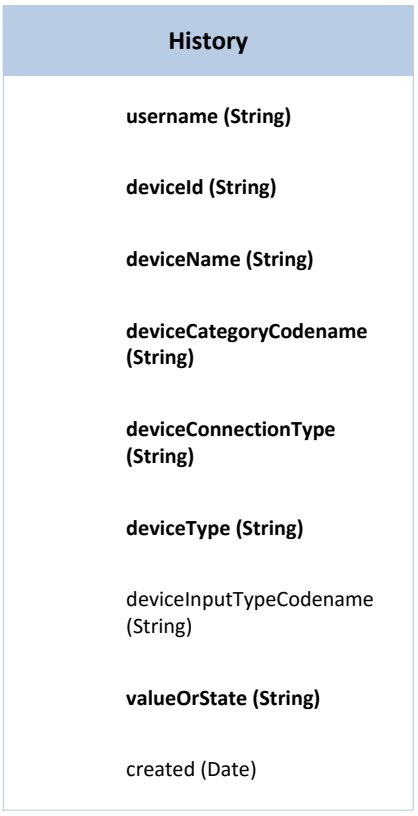


Figura 14 – Modelo de Base de Dados do *HUB* que Suporta a Gestão do Histórico

4.4.2 Aplicação Android

O seguinte diagrama representa a base de dados da aplicação *Android*. Esta contempla informações sobre os *HUBs* que estão a ser controlados/monitorizados, os dados das respetivas autenticações, categorias, diferentes formas de ativação, tipos de dispositivo, portas de entrada e saída e ainda o histórico.

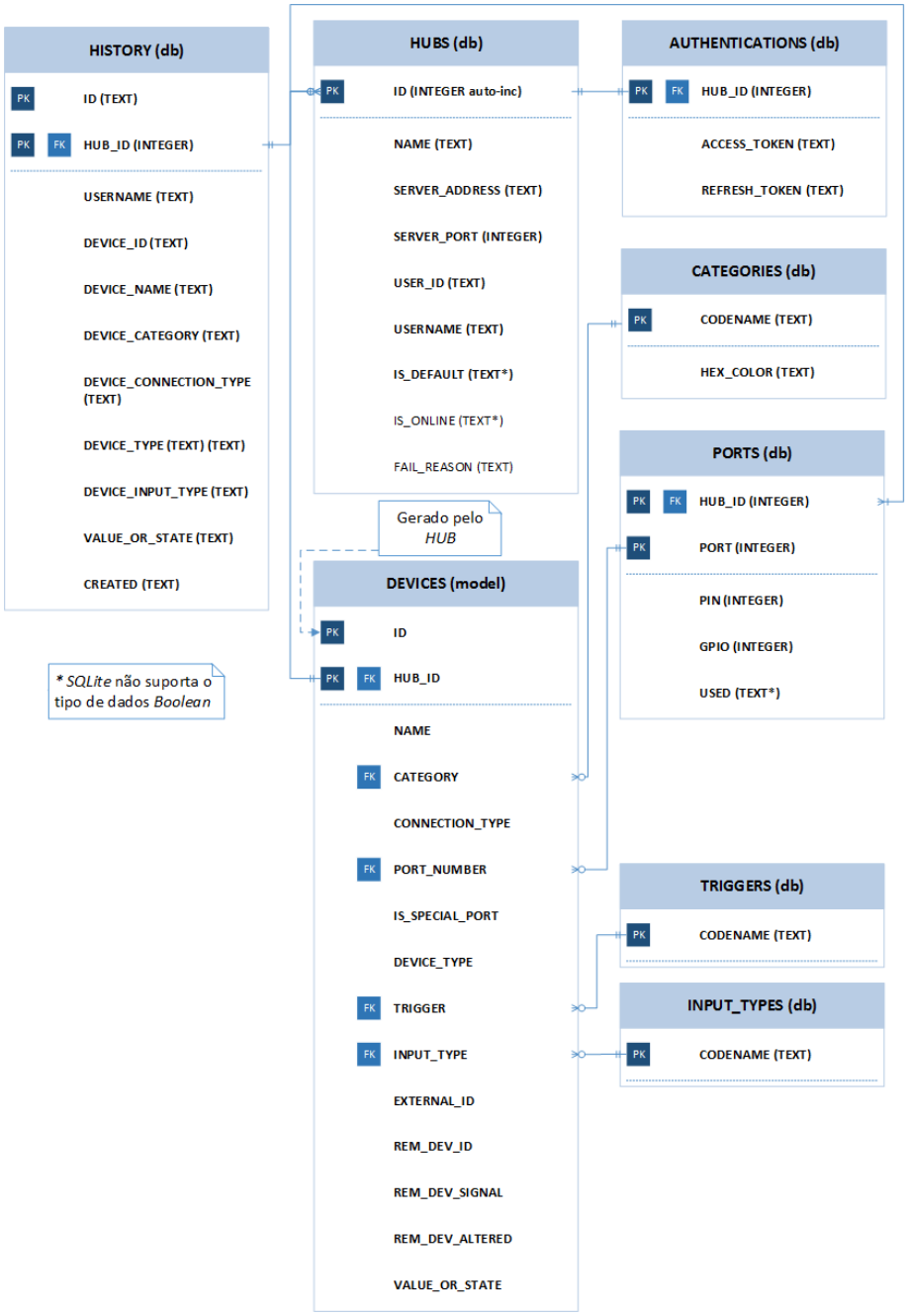


Figura 15 – Modelo de Base de Dados da Aplicação *Android*

Como se pode ver, ao lado do nome da tabela *DEVICES* está escrito "*model*". Isto significa que na realidade esta tabela não pertence à base de dados, mas sim aos modelos. Devido à necessidade de constante atualização da informação dos Dispositivos para que a aplicação se mantenha sincronizada, seria inútil que esta tabela existisse na base de dados.

5 Construção da Solução

O desenvolvimento deste projeto foi maioritariamente organizado e registado, utilizando o sistema de controlo de versões do *Bitbucket* e o sistema de criação de *issues* que o acompanha.

Neste foram criados dois repositórios: um para a componente *Android* e outro para a componente *Node.js* em *Linux*.

A partir daqui, o planeamento das tarefas necessárias para cada uma das componentes foi registado através da criação de uma *issue* por tarefa, a qual seria posteriormente executada e documentada através do lançamento de *commits* incrementais, à medida que a mesma era desenvolvida.

A utilização de um sistema de controlo de versões trouxe vantagens ao processo de desenvolvimento e teste, pois facilitou o controlo entre o que estava a ser desenvolvido e o que estava a ser testado em produção do lado de um suposto cliente.

Ao longo deste capítulo será apresentado o processo de investigação e desenvolvimento correspondente a cada uma das partes que compõe este projeto.

5.1 Controlador/*HUB*

5.1.1 Interação do Node.js com Sensores e Atuadores

Após a instalação do sistema operativo e da preparação do ambiente de desenvolvimento em *Node.js*, começou-se por testar o controlo dos pinos de entrada e saída (mais conhecidos como *GPIO* de *General Purpose Input Output*) do *Raspberry Pi*, utilizando a biblioteca *ONOFF*, para posterior controlo de sensores ou dispositivos elétricos.

Os referidos pinos têm ser instanciados em forma de um objeto, para poderem ser controlados, e para isto, é necessário indicar o tipo de funcionalidade que este pino terá: *Input* ou *Output*.

Se for do tipo ***Input***, significa que o componente eletrónico que estiver ligado a este vai poder fornecer um valor. No entanto, este valor pode ser obtido através de um pedido feito pelo sistema ou através de um observador colocado sobre o referido pino.

No caso de um sensor de temperatura ou de distância, para obter o seu valor a uma dada altura, é apenas necessário fazer um pedido visto que a medição é feita no momento.

No caso de um sensor de contacto, um botão ou um sensor de movimento, já é necessário colocar um observador sobre cada um destes, para ser possível detetar uma alteração no seu estado. Este tipo de sensores não fornecem uma medição, mas sim um estado, que se pode considerar binário, visto só existirem dois estados possíveis (e.g. aberto ou fechado, pressionado ou não pressionado, deteção de movimento ou não). Para além disto, e a principal razão por se utilizarem observadores, é o facto do estado destes sensores apenas ser alterado por ações externas, seja um portão a abrir, uma pessoa a carregar numa campainha ou a passar numa divisão onde se encontra um sensor de movimento.

Se for do tipo ***Output***, significa que o componente (e.g. Atuador ligado a uma luz) vai poder receber um sinal binário de forma a poder agir de acordo com este (e.g. Se receber o sinal 1, a luz liga-se, e se receber o sinal 0, a luz desliga-se). Também será possível a qualquer momento enviar um pedido para saber o estado de um pino deste tipo. Por exemplo, se quiser saber se a luz está ligada, como esta está a ser controlada por um atuador, vou então querer saber o estado deste. Como este está a ser controlado por um pino do tipo *Output,* o valor obtido após o pedido será 1 se a luz estiver ligada ou 0 se não estiver.

Sensores de Temperatura Modelo DS18B20

Estes sensores de temperatura foram os escolhidos para serem utilizados neste projeto visto serem os mais populares na indústria da automação e por existirem em formatos resistentes ou não à água.

Muito semelhante à antiga estrutura de rede que utilizava um único barramento para comunicação entre os diferentes computadores, os quais tinham uma placa de rede com um endereço *MAC* que os identificava de forma única, estes sensores funcionam relativamente da mesma forma.

Cada um vem de fábrica com um endereço único que os identifica e, para serem utilizados no *Raspberry Pi*, têm de estar ligados em série a um único pino.

A razão para esta topologia, deve-se ao facto de estes sensores funcionarem sobre o protocolo *One-Wire*, o qual permite a comunicação com qualquer tipo de dispositivo que utilize este protocolo, sobre uma única linha de dados.

Para se utilizar este protocolo, é necessário reservar um dos pinos, nas configurações do sistema operativo. Portanto, quando se envia um pedido de leitura por parte do sistema para este pino, é também enviado o endereço do sensor ao qual se quer requisitar o valor.

Tendo em conta que estes sensores são um caso particular, para o seu controlo é utilizada uma biblioteca específica, a *DS18X20*.

5.1.2 Criação do Servidor principal em Node.js

O objetivo principal deste servidor é permitir, não só o acesso controlado e abstraído aos pinos do *Raspberry Pi*, mas também fornecer um conjunto de outras funcionalidades que permitam apoiar o que será um sistema de automação.

5.1.2.1 Preparação da Base de Dados

Como foi definido no capítulo de Avaliação de Abordagens, a base de dados escolhida foi a *MongoDB* e como tal teve de ser instalada juntamente com uma biblioteca (*Mongoose*) para ser facilmente manipulada através do *Node.js*.

Utilizando o modelo da base de dados do *HUB,* apresentado anteriormente, foi criado um módulo onde, utilizando o *Mongoose*, se definiu a ligação à base de dados, os eventos de conexão e os *schemas* de cada coleção.

Um ***schema*** em *MongoDB* tem o formato de um objeto *JSON*, define os campos que devem existir em cada documento (linha) de uma coleção (tabela) e serve de base para a criação do respetivo modelo de dados. No entanto, uma coleção não obriga a que os documentos sigam um *schema*, permitindo assim que estes tenham diferentes campos.

```
var Category = new Schema({
    codename:   {type: String,  required: true, unique: true},
    hex_color:  {type: String,  required: true, unique: true},
    created:    {type: Date,    default: Date.now}
});

module.exports.CategoryModel = mongoose.model('Category', Category);
```

Figura 16 – Exemplo da Criação de um *Schema* e da Exportação do seu Modelo de Dados

Tendo o módulo base sido criado, este poderia agora ser utilizado pelo módulo que define a camada de acesso a dados, onde estão definidas as funções necessárias para interação com a base de dados.

5.1.2.2 Abstração da Manipulação de *Inputs* e *Outputs* (*GPIO*)

Para controlar a manipulação dos pinos, foi criado um módulo onde foram definidas algumas funções a serem utilizadas mais tarde pelos serviços expostos.

As funções são as seguintes:

- **onOffAsync (gpioId, stateString, callback)** – O objetivo desta função é alterar o estado de um pino, definido como sendo do tipo *Output*, utilizando o parâmetro *gpioId* para o identificar e o parâmetro *stateString* para o ativar ou desativar consoante receba *on* ou *off*. Com esta função pretende-se mimicar a ação de ligar ou desligar algum dispositivo. O parâmetro *callback* aponta para uma função que tem de ser executada assim que haja confirmação de sucesso ou ocorra um erro.

- **quickPulseAsync (gpioId, callback)** – O objetivo desta função é simular a ação física de carregar num botão de um comando para, por exemplo, controlar um portão que ao receber o mesmo sinal efetua a ação de parar, mover-se para abrir ou mover-se para fechar.

  Este controlo seria feito utilizando um atuador que serviria de intermediário entre o *Raspberry Pi* e o controlador do portão (que poderia ser um botão que estivesse dentro do edifício).

  Desta forma, para simular o referido sinal, esta função utiliza o parâmetro *gpioId* para identificar o pino onde está ligado o atuador, enviando-lhe de seguida um sinal de ativação e passado um segundo, um sinal de desativação.

  A utilização deste intervalo com mudanças de estado tem o propósito de deixar a corrente passar pelo atuador, completando o circuito, e fazendo com que o sinal chegue ao mecanismo que controla o motor do portão.

  O parâmetro *callback* aponta para uma função que tem de ser executada assim que haja confirmação de sucesso ou ocorra um erro.

- **getGpioValue (gpioId)** – Esta função devolve o estado do dispositivo que se encontra ligado ao pino passado no parâmetro *gpioId*, independentemente deste ser do tipo *Input* ou *Output*. No entanto, o estado real deste deve ser passível de ser identificado de uma forma binária (e.g. 1 para Ligado, 0 para Desligado, 1 para Aberto, 0 para Fechado).

- **getTempSensorsAsync (callback)** – Esta função utiliza a *callback* que recebe por parâmetro, para devolver um conjunto de objetos *JSON* que representam o endereço do sensor de temperatura e o seu valor correspondente em graus Celcius.

  Para obter esta informação é utilizada a biblioteca *DS18X20*, a qual faz uma pesquisa no barramento do pino registado como sendo *One-Wire*, para procurar os dispositivos que lá estão ligados, cujos endereços começam pelo número 28. Este número identifica o grupo correspondente aos sensores de temperatura que operam sobre o protocolo *One-Wire*.

- **watchBinaryInputDevice (gpioId, callback)** – Tal como foi explicado na secção 5.1.1, para ser possível detetar uma alteração num dispositivo do tipo *Input* que só é afetado por ações externas, é necessário colocar um observador sobre este. O papel desta função é fazer precisamente isto, utilizando o parâmetro *gpioId* para identificar o pino onde o dispositivo está ligado.

- **unwatchBinaryInputDevice (gpioId)** – Quando se deixa de utilizar um dispositivo ao qual tenha sido previamente associado um observador, é necessário remover o mesmo para que este não produza informações erróneas.

As funções *onOffAsync*, *quickPulseAsync*, *watch* e *unwatch* utilizam a biblioteca *ONOFF* para cumprirem o seu propósito. No entanto, como esta biblioteca não permite a leitura do estado de um pino se não existir um objeto instanciado sobre o mesmo e, visto que, a instanciação de um objeto destes faz com que o estado do pino seja automaticamente colocado a 0, para se conseguir obter o valor correto na função *getGpioValue*, foi necessário utilizar a biblioteca *RPIO*.

A razão pela qual a biblioteca *RPIO* não é utilizada em todo o projeto em vez da *ONOFF* deve-se ao facto das suas funções serem apenas síncronas, o que a torna praticamente inútil num ambiente assíncrono como o do *Node.js*.

5.1.2.3 Criação da API REST

Numa primeira fase do projeto considerou-se expor os serviços que o HUB forneceria através de uma *API REST*. Esta API tinha o objetivo de proporcionar pontos de acesso, para que a aplicação cliente pudesse fazer a gestão de *HUBs* e Dispositivos sempre que necessário, sobre um sistema de Autenticação e Autorização como detalhado no capítulo 4.3.1.

Os pontos de acesso desta API serão de seguida apresentados e detalhados.

Ponto de Acesso para Autenticação

POST **/oauth/token**

Para criar o sistema de autenticação, foi utilizada a biblioteca *Passport* que dispõe de múltiplas estratégias para suportar diversos tipos de autenticação. Estas estratégias estão pré-configuradas para receberem os parâmetros necessários para o tipo de autenticação escolhido.

Tendo em conta que o objetivo deste projeto é fazer a autenticação utilizando as credenciais da aplicação cliente (identificador e segredo) e as credenciais dos utilizadores (nome e palavra-passe) e depois utilizar um conjunto de *Tokens* que permite autorizar os pedidos e re-autenticar um utilizador, foram então utilizadas as seguintes estratégias:

- **(Autenticação) Passport-Oauth2-Client-Password:** Esta estratégia, juntamente com a utilização de um servidor de autorização em conformidade com as especificações

OAuth 2.0, permite criar dois pontos de troca de credenciais, utilizando o mesmo ponto de acesso.

- **Cliente + Utilizador:** Enviando os dados da aplicação cliente e do utilizador, se ambos estiverem corretos, são devolvidos dois *Tokens*, um de Acesso e outro de Atualização como referido no capítulo 4.3.1.

```
// Create OAuth 2.0 server
var server = oauth2orize.createServer();

// Exchange clientId & secret, username & password for access and refresh tokens
server.exchange(oauth2orize.exchange.password(function(client, username, password, scope, done){
//       ...
}));
```

Figura 17 – Demonstração do Código da Troca das Credenciais da Aplicação Cliente e do Utilizador Por *Tokens*

- **Cliente + *Token* de Atualização:** Enviando os dados da aplicação cliente e o *Token* de Atualização é possível re-autenticar um utilizador, cujo *Token* de Acesso entretanto expirou, sem a necessidade de enviar a sua palavra-passe.

```
// Exchange clientId & secret and refreshToken for new access and refresh tokens
server.exchange(oauth2orize.exchange.refreshToken(function(client, refreshToken, scope, done){
//       ...
}));
```

Figura 18 - Demonstração do Código da Troca das Credenciais da Aplicação Cliente e do *Token* de Atualização por Novos *Tokens* de Acesso e de Atualização

- **(Autorização) Passport-Http-Bearer:** Esta estratégia não é acedida diretamente por um ponto de acesso, mas de uma forma subtil é capaz de proteger os pontos de acesso necessários da *API*, tendo em conta que é invocada através da utilização de um sistema de *middleware* que se interpõe entre a chegada de um pedido e o seu destino.

  Os pontos de acesso que requerem autorização são definidos no servidor com o formato "/api/*" e é feita uma indicação de que todos os pedidos que tentem aceder a estes pontos vão ter de primeiro passar pelo referido *middleware*, onde é implementada esta estratégia para confirmar a validade do *Token* de Acesso enviado, relativamente ao utilizador e à aplicação cliente de onde provém o pedido.

  A Figura 19 demonstra um esqueleto do código de autorização e da definição do *middleware*.

```
var express     = require('express');
var app         = express();

//Authorization
app.all('/api/*', isAuthenticated);

//------------------------------------------------------------------//
// CUSTOM AUTHORIZATION ROUTE MIDDLEWARE FUNCTION
//------------------------------------------------------------------//

function isAuthenticated(req, res, next){
    passport.authenticate('bearer', function(err, user, info){
    //        ...
    })(req, res, next);
}
```

Figura 19 – Demonstração do Código de Autorização e Definição do *Middleware*

Pontos de Acesso para Gestão de Dispositivos

POST **/api/devices** **Adicionar Dispositivo**

Este ponto de acesso recebe os dados de um dispositivo e adiciona-o à base de dados.

POST **/api/devices/{deviceId}/output/{triggerCommand}** **Alterar Estado De Dispositivo**

Este ponto de acesso verifica a existência de um dispositivo e altera o seu estado consoante o tipo de comando que recebe. Por exemplo, se o *triggerCommand* for 1 e se o dispositivo for uma luz, então ela vai ser ligada.

GET **/api/devices?categoryToFilter** **Obter Uma Lista De Dispositivos**

Este ponto de acesso devolve uma lista de dispositivos filtrada ou não pela categoria recebida.

GET **/api/devices/{deviceId}** **Obter Um Dispositivo Através Do Id**

Este ponto de acesso verifica se existe um dispositivo com o id recebido, e se existir, então envia-o.

PUT **/api/devices/{deviceId}** **Editar Um Dispositivo Através Do Id**

Este ponto de acesso verifica a existência de um dispositivo através do id recebido, e se existir, então utiliza os dados recebidos para fazer uma atualização do registo que se encontra na base de dados.

DELETE **/api/devices/{deviceId}** **Remover Um Dispositivo Através Do Id**

Este ponto de acesso remove um dispositivo da base de dados, caso este exista.

Pontos de Acesso para Gestão de Sensores de Temperatura

GET	/api/tempsensors	Obter Uma Lista De Sensores De Temperatura

Este ponto de acesso utiliza a função **getTempSensorsAsync** mencionada no capítulo 5.1.2.2 para devolver uma lista de sensores de temperatura com a respetiva medição associada em graus Celcius.

GET	/api/tempsensors/{externalId}	Obter A Medição de Um Sensor de Temperatura

Este ponto de acesso utiliza uma variação da função utilizada no ponto anterior para devolver a medição associada ao sensor de temperatura, cujo endereço é igual ao recebido no *externalId*.

Ponto de Acesso para Gestão das Categorias

GET	/api/categories	Obter Uma Lista De Categorias

Este ponto de acesso devolve uma lista de categorias. As categorias são atribuídas pelo utilizador a um dispositivo, aquando da sua criação, para que a sua identificação seja mais percetível.

Ponto de Acesso para Gestão dos Tipos de Ativação

GET	/api/triggers	Obter Uma Lista Dos Tipos de Ativação

Este ponto de acesso devolve uma lista de tipos de ativação. Os tipos de ativação permitem que, na criação de um dispositivo do tipo *Output*, o utilizador possa escolher se o quer controlar com um botão do tipo Ligar/Desligar ou com um Controlo Deslizante, para simular, por exemplo, o carregar de um botão de um comando para ativar um portão.

Ponto de Acesso para Gestão dos Dispositivos Suportados do Tipo Input

GET	/api/inputtypes	Obter Uma Lista Dos Dispositivos Suportados Do Tipo *Input*

Este ponto de acesso devolve uma lista dos dispositivos suportados que são do tipo *Input*. Esta lista é utilizada para, aquando da criação de um dispositivo do tipo *Input*, o utilizador possa saber que dispositivos são atualmente suportados (e.g. Botão, Sensor de Temperatura, Sensor de Contacto).

Ponto de Acesso para Gestão dos Pinos GPIO Disponíveis

GET	/api/gpiopins	Obter Uma Lista Dos Pinos GPIO Disponíveis

Este ponto de acesso devolve uma lista dos pinos *GPIO* disponíveis sobre os quais o utilizador pode adicionar dispositivos. Para tornar mais fácil a ligação física dos dispositivos e para suportar futuros desenvolvimentos relativamente ao aspeto físico do *HUB*, estes pinos estão associados a números de portas, de uma forma muito semelhante às portas *Lan* que existem num *Router*.

5.1.2.4 Conclusões e Alterações sobre a API REST

Tendo-se atingido um bom patamar em que o projeto (*HUB* e aplicação *Android*) já possuía um sistema de autenticação e autorização de pedidos e permitia a gestão de múltiplos *HUBs* e dispositivos com fios, decidiu-se então fazer testes de integração para verificar a rapidez da comunicação e a estabilidade do sistema em geral.

As seguintes conclusões foram retiradas após estes testes:

- A rapidez de controlo de um Dispositivo era aceitável mas podia ser melhorada;
- Devido à utilização de *Polling*, o tempo de atualização dos dispositivos nos clientes tinha de ser superior ao desejado visto que, quanto mais frequentes fossem as atualizações, mais recursos do *HUB* se usariam e maior seria o consumo de dados por parte dos Clientes;
- Reparou-se também que a utilização de uma *API REST* iria limitar futuras funcionalidades (e.g. Notificações), devido ao facto de não ser possível criar uma comunicação bidirecional.

Tendo em conta estas conclusões decidiu-se fazer uma restruturação do sistema de comunicação onde se deixou de expor a maioria dos recursos através da *API REST*, à exceção do ponto de acesso para a autenticação, passando-se a utilizar a biblioteca *Socket.IO* para tudo o resto.

5.1.2.5 Estruturação da Comunicação Utilizando Socket.IO

Esta biblioteca possui uma implementação baseada em eventos, permite comunicação bidirecional e é independente do tipo de ambiente onde for utilizada, visto ser construída sobre uma camada de comunicação em tempo real, o *Engine.IO*, que faz a manutenção das ligações, escolhendo o transporte correto dependendo daquilo que o cliente suportar (e.g. *AJAX long-polling*, *WebSockets*, etc).

Ao deixar apenas o sistema de autenticação a ser exposto pela *API REST*, significa que passou a ser necessário um novo sistema de autorização, como foi apresentado na secção 4.3.1, e novos pontos de acesso utilizando o mecanismo de eventos do *Socket.IO*.

Com este propósito foram então implementados os seguintes eventos:

Eventos com Funcionalidades Iguais às do Serviço REST

PONTOS DE ACESSO EM *REST*	EVENTOS EM *SOCKET.IO*
/api/tempsensors	GetTempSensors
/api/tempsensors/{externalId}	(Deixou de ser necessário porque já não se utiliza *Polling*)
/api/categories	GetCategories
/api/triggers	GetTriggers
/api/inputtypes	GetInputTypes
/api/gpiopins	GetGpioPins

Eventos para Gestão de Dispositivos

SOCKET **AddDevice** **Adicionar Dispositivo**

Este evento recebe os dados de um dispositivo e adiciona-o à base de dados. No final, o *HUB* notifica todos os clientes que estejam no momento a visualizar a lista de Dispositivos, emitindo o evento ***UpdateDevices***, o qual está configurado nos mesmos para que, quando recebido, requisitem uma nova lista de Dispositivos.

SOCKET **TriggerDeviceById** **Alterar O Estado De Um Dispositivo**

Este evento verifica a existência de um dispositivo e altera o seu estado consoante o tipo de comando que recebe. Por exemplo, se o comando for 1 e se o dispositivo for uma luz, então ela vai ser ligada.

Se correr tudo bem, é emitido o evento ***UpdateSpecificDevice*** para todos os clientes que estejam no momento a visualizar a lista de Dispositivos, à exceção do que emitiu o ***TriggerDeviceById***, onde é enviado o identificador do Dispositivo e o seu novo estado, para que estes consigam atualizar apenas o elemento da lista associado a este Dispositivo.

Se existir alguma Notificação cujas regras vão de encontro com o novo estado do Dispositivo, é emitido o evento ***DisplayNotification*** para todos os clientes que no momento têm uma ligação ativa e que estão subscritos à respetiva notificação.

SOCKET **GetDevices** **Obter Uma Lista De Dispositivos**

Este evento devolve uma lista de Dispositivos filtrada ou não pela categoria recebida.

SOCKET **GetDeviceById** **Obter Um Dispositivo Através Do Id**

Este evento verifica se existe um Dispositivo com o id recebido, e se existir, então envia-o.

SOCKET **GetLinkedDevices** **Obter Uma Lista de Dispositivos Remotos Sincronizados**

Este evento devolve uma lista de dispositivos remotos que completaram o processo de sincronização com o *HUB*.

SOCKET **EditDevice** **Editar Um Dispositivo Através Do Id**

Este ponto de acesso verifica a existência de um dispositivo através do id recebido e se existir então utiliza os dados recebidos para fazer uma atualização do registo que se encontra na base de dados. Da mesma forma que o ***AddDevice***, é também emitido o evento ***UpdateDevices***.

Se o Dispositivo tiver alguma Notificação a ele associada, e se uma das seguintes propriedades tiver sido alterada, então considera-se que a integridade da Notificação é posta em risco e procede-se à sua remoção.

Propriedades da tabela *Device*: *connectionType, deviceType, triggerCodename, inputTypeCodename*.

SOCKET **DeleteDevice** **Remover Um Dispositivo Através Do Id**

Este ponto de acesso remove um dispositivo da base de dados caso este exista e, tal como o ***AddDevice*** e o ***EditDevice***, é emitido o evento ***UpdateDevices***.

Evento para Gestão do Histórico

SOCKET **GetHistory** **Obter Uma Lista De Registos do Histórico**

Este evento devolve uma lista de registos do Histórico, que estejam dentro de um intervalo de duas datas recebidas.

Eventos para Gestão de Notificações

SOCKET **AddNotification** **Adicionar Notificação**

Este evento recebe os dados de uma Notificação e adiciona-a à base de dados. No final, o *HUB* notifica todos os clientes que estejam no momento a visualizar a lista de Notificações, emitindo o evento ***UpdateNotifications***, o qual está configurado nos mesmos para que, quando recebido, requisitem uma nova lista de Notificações.

SOCKET **GetNotificationsForUser** **Obter Uma Lista De Notificações**

Este evento devolve uma lista de Notificações e outros detalhes relacionados, tendo em conta um Utilizador específico.

SOCKET **SetUserNotificationStatus** (De-) Subscrever um Utilizador a uma Notificação

Este evento cria um registo na tabela da base de dados que relaciona um Utilizador com uma Notificação, caso se trate de uma subscrição, ou então elimina o registo, caso se trate de uma remoção da subscrição.

SOCKET **DeleteNotificationById** Remover Uma Notificação Através Do Id

Este ponto de acesso remove os registos da tabela que relaciona os Utilizadores com a Notificação identificada pelo Id recebido, depois remove o registo da própria Notificação e, tal como o ***AddNotification***, é emitido o evento ***UpdateNotifications***.

Evento para Controlo do Envio de Atualizações do Estado de Dispositivos

SOCKET **WantSpecificDeviceUpdates** (Não) Receber Alterações ao Estado de Dispositivos

Este evento permite que as aplicações cliente indiquem se querem receber ou não atualizações sobre a mudança do estado dos dispositivos. Com esta técnica consegue-se controlar quando e a quem devem ser emitidos os eventos ***UpdateSpecificDevice***.

Evento para Manter a Ligação Ativa

SOCKET **Pong** Resposta para Manter a Ligação Ativa

Este evento é recebido em resposta ao evento ***Ping*** enviado pelo *HUB* para cada cliente, de 20 em 20 segundos, para que a ligação destes se mantenha sempre ativa. Ao contrário dos outros eventos, este é o único que não precisa que o utilizador concretize com sucesso o processo de autenticação da ligação.

5.1.3 Utilização de Ferramentas Auxiliares para Análise

Para tornar mais eficiente o processo de administração e resolução de problemas relacionados com o sistema físico do *HUB* e com os serviços que nele executam, foram instaladas duas aplicações, *Munin* e *Monit*.

5.1.3.1 Munin

Munin é uma ferramenta que permite a monitorização da utilização de recursos, pertencentes a um sistema *Unix*, através da apresentação de diversos gráficos, numa aplicação web. No contexto deste projeto, o papel principal desta ferramenta foi o de diagnosticar problemas relacionados com o cartão de memória, tendo em conta que foi detetado um baixo desempenho nas funções do *HUB* durante uma fase inicial do seu desenvolvimento, assim como múltiplas situações de corrupção dos dados deste.

O cartão de memória utilizado era da marca *Toshiba*, e tinha uma classificação de velocidade, número 4, com 4.4 MB/s de velocidade mínima de escrita.

Ao analisar os gráficos de latência (Figura 20), relativos às operações de leitura e escrita do cartão de memória, reparou-se que a média de tempo que o processador tinha de esperar até ao término de uma operação do cartão de memória, era de 3.57 segundos. Ou seja, quando era necessário, por exemplo, fazer uma leitura à base de dados, durante este tempo todo, o processador tinha de estar parado, afetando assim, o desempenho deste projeto.

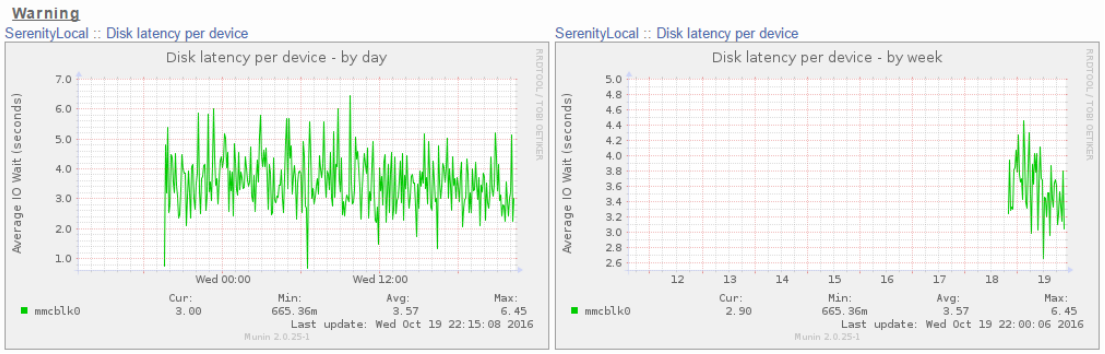


Figura 20 – Gráfico Diário e Semanal da Latência do Cartão de Memória Utilizando a Ferramenta *Munin*

Visto que a velocidade do cartão de memória aparentava ser a causa deste problema, decidiu-se trocar para um outro, da marca *Samsung*, cuja classificação de velocidade era número 10, com 48 MB/s de velocidade mínima de escrita.

Após esta troca, voltou-se a analisar o gráfico semanal (Figura 21), verificando-se uma média de tempo de 226 milissegundos, conseguindo-se assim, uma melhoria bastante significativa no desempenho do projeto e na manutenção da integridade por parte do cartão de memória.

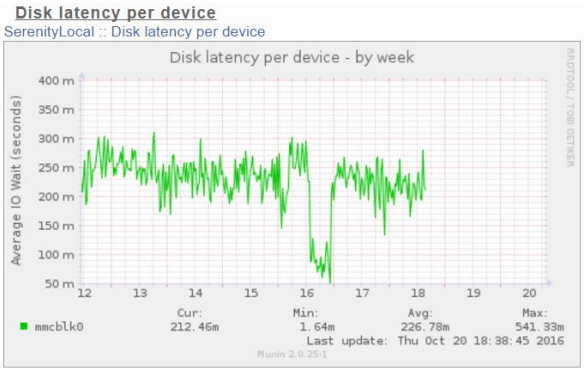


Figura 21 - Gráfico Semanal da Latência do Novo Cartão de Memória Utilizando a Ferramenta *Munin*

5.1.3.2 Monit

Monit é uma utilidade para gerir e monitorizar sistemas *Unix*. No contexto deste projeto, o papel principal desta ferramenta, é monitorizar alguns serviços do *HUB* assim como a aplicação em *Node.js*. Caso ocorra algum problema (e.g. Crash do serviço *Node.js*) com os

processos associados a estes, o *Monit* é responsável por criar novos processos, mantendo assim, o *HUB*, com o mínimo de tempo de inatividade possível.

Monit permite também, de uma forma remota, visualizar e alterar o estado de cada processo através de uma aplicação web, a qual se mantém protegida utilizando *SSL* e um sistema de autenticação.

De seguida são apresentadas duas capturas da aplicação web, que demonstram o estado dos serviços no ambiente de desenvolvimento (Figura 22) e no ambiente de teste em produção (Figura 23).

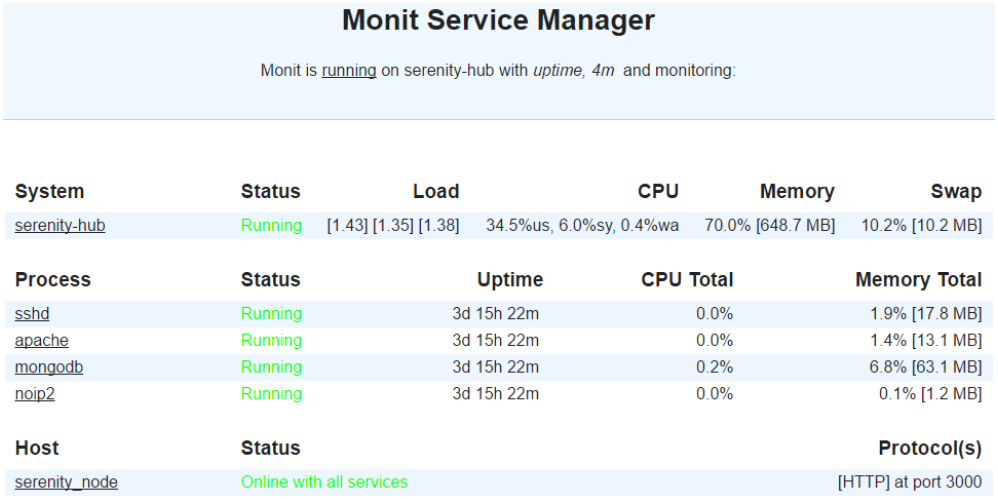


Figura 22 – Demonstração do Estado dos Serviços do *HUB* no Ambiente de Desenvolvimento

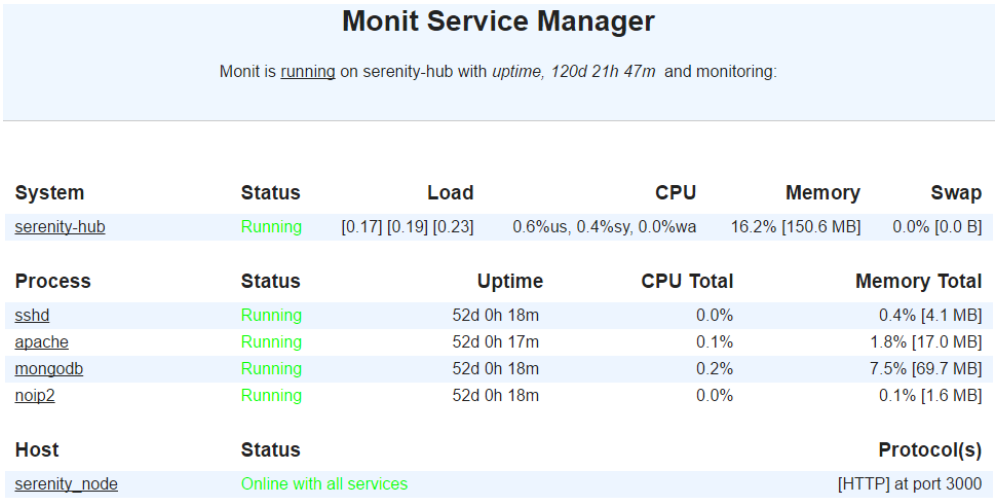


Figura 23 – Demonstração do Estado dos Serviços do *HUB* no Ambiente de Teste em Produção

O ambiente de desenvolvimento e o de produção têm implementadas diferentes versões do projeto, as quais estão bastante espaçadas temporalmente.

Em desenvolvimento estão a ser geridos 7 dispositivos locais e 2 remotos e, em média estão ligados 2 utilizadores para teste. Em produção estão a ser geridos 2 dispositivos locais e, em média estão ligados 3 utilizadores.

Tendo contextualizado ambos os ambientes, é possível agora fazer uma análise das duas figuras:

- A versão mais recente, que tem mais funcionalidades e mais dispositivos a executar no sistema de desenvolvimento, está a exigir bastante mais recursos em comparação com a versão que está a executar no sistema de produção;
- Em ambos os ambientes, todos os serviços estão a executar corretamente, porém, no ambiente de produção é possível verificar que, apesar do *HUB* estar a funcionar há 120 dias, há 52 dias houve a necessidade de reiniciar os seus serviços.

5.2 Desenvolvimento de Dispositivos Remotos

5.2.1 Hardware Utilizado ao longo do Desenvolvimento

Numa fase inicial, utilizou-se o *Arduino Pro Mini* (versão não oficial), o qual operava a 3.3V e 8MHz e cujo armazenamento era de 14KBs. Para além disto, para ser possível fazer a transferência do código de um computador para este, era necessário utilizar um adaptador conhecido como *FTDI232* que possui uma entrada *Mini-USB* e expõe alguns pinos para conexão com o *Arduino*.

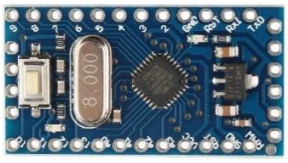


Figura 24 – Demonstração de um *Arduino Pro Mini* (dx.com)

Figura 25 – Demonstração de um Adaptador *FTDI232* (dx.com)

Ao longo do desenvolvimento dos Dispositivos Remotos, optou-se por criar apenas um programa, que através da alteração de umas propriedades, se conseguisse adaptar a qualquer tipo de Dispositivo que fosse suportado pelo projeto. Esta decisão levou ao encontro de um problema que foi a falta de memória, ou seja, o programa criado já ocupava pouco mais do que os 14KBs existentes no *Arduino Pro Mini*, não sendo possível fazer a transferência do código.

Para resolver este problema, decidiu-se substituir esta versão do *Arduino* pelo *Pro Micro* (versão não oficial), o qual opera de igual forma, mas possui o dobro do espaço de armazenamento, 28KBs. Outra vantagem que esta versão trouxe, foi o facto de incluir uma porta *Micro-USB* e a tecnologia necessária para a interação direta com um computador, o que tornou o processo de desenvolvimento e teste bastante mais fluído.

Figura 26 – Demonstração de um *Arduino Pro Micro* (photobucket.com)

5.2.2 Interação do Arduino com Sensores e Atuadores

De uma forma muito semelhante ao *Raspberry Pi*, o *Arduino* também possui os conceitos de *GPIOs*/pinos que podem ser utilizados como sendo do tipo *Input* ou *Output*.

Para a utilização dos sensores de temperatura DS18B20 também foi necessário recorrer a uma biblioteca que trabalhasse com o protocolo *One-Wire*, e outra mais específica (*DallasTemperature*) que utilizava o acesso ao barramento, concedido pela biblioteca anterior, para gerir o acesso a estes sensores.

5.2.3 Interação e Comunicação via Rádio

Tal como ficou decidido na secção 3.3, o sistema de comunicação utilizado pelos Dispositivos Remotos tem por base o rádio *RFM69HCW*.

A conexão entre o *Arduino* e este rádio realiza-se através do barramento *SPI* (*Serial Peripheral Interface*). Este tipo de barramento possui uma interface de comunicação série síncrona, onde os dados são enviados bit a bit de forma sequencial, permite transferências de dados de forma bidirecional através de duas linhas (*MOSI* e *MISO*), e utiliza uma arquitetura de *master-slave* onde existe apenas um *master*, neste caso o *Arduino*, e vários *slaves*, neste caso apenas o rádio, identificados através das respetivas linhas, de *slave select* (*SS*). Estas linhas servem também para que o *master* indique ao *slave* que deve acordar para receber ou enviar dados. (Grusin 2014)

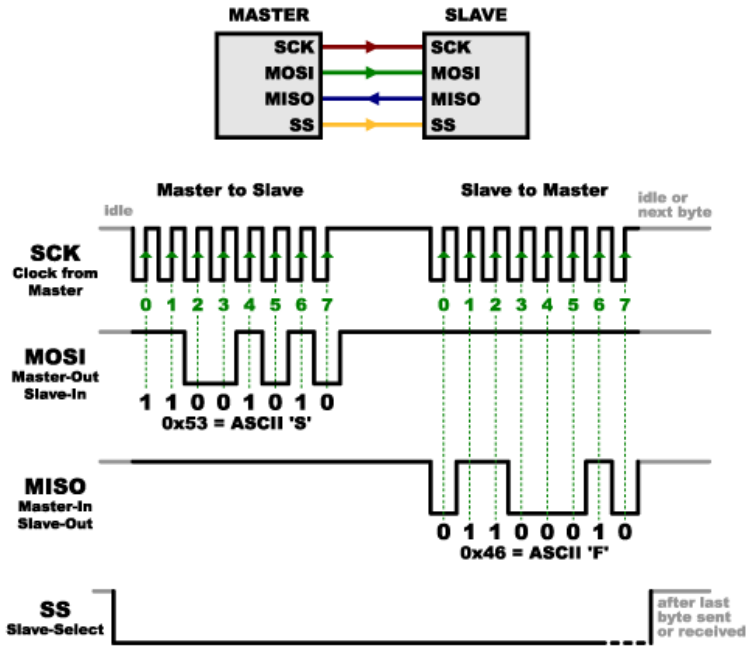


Figura 27 – Exemplificação da Comunicação *SPI* Entre um Dispositivo *Master* e um *Slave* (sparkfun.com)

Para abstrair a utilização deste barramento e ser possível controlar o rádio a um alto nível, recorreu-se à biblioteca *RFM69* para *Arduino*, fornecida pela *LowPowerLab*.

Tendo em conta que este equipamento vai estar a comunicar dentro de uma rede, é necessário definir alguns detalhes no código para que tal seja possível.

- **Frequência**: Visto que este projeto está a ser desenvolvido dentro da União Europeia, foi necessário verificar quais as frequências legais sobre as quais se pode transmitir e adquirir os rádios com a frequência correta, que neste caso é de 433MHz. Esta frequência tem de ser a mesma para todos os Dispositivos Remotos porque não é possível comunicar em frequências diferentes.

- **Chave de Encriptação**: A comunicação através destes rádios é encriptada utilizando o *standard AES-128* (*Advanced Encryption Standard*) implementado diretamente no *hardware*. Sendo este tipo de encriptação de 128 bits, é então necessário definir uma chave de 16 bytes/caracteres, que deve ser conhecida por todos os Dispositivos Remotos de uma mesma rede.

- **Identificador da Rede**: Para que os Dispositivos saibam a que rede pertencem, todos os que pertencerem a uma mesma rede devem conter o mesmo identificador definido. As redes podem ser identificadas desde o número 0 ao 255 e cada uma está isolada da outra.

- **Identificador do Dispositivo**: Para que dentro de uma rede seja possível enviar mensagens de um Dispositivo X para um Dispositivo Y é necessário que cada um deles tenha definido o seu próprio identificador único. Os Dispositivos podem ser

identificados desde o número 0 ao 254, podendo existir até 255 dispositivos dentro de cada rede. O endereço 255 está reservado para efeitos de *Broadcast* de maneira que, se um Dispositivo quiser contactar com todos os outros ao mesmo tempo, pode identificar o destinatário utilizando este endereço.

Apesar de estes detalhes estarem a ser definidos diretamente no código, numa situação futura espera-se que, à exceção da Frequência, estes possam ser atribuídos automaticamente pelo *HUB*.

Para o *Arduino* interagir com o rádio utiliza-se então a biblioteca *RFM69* para instanciar um objeto, indicando os pinos aos quais o rádio está ligado ao *Arduino*. Através deste, é possível inicializar o rádio utilizando a Frequência, o Endereço do Dispositivo e o Endereço de Rede, definir que a comunicação tem de ser encriptada utilizando a Chave de Encriptação e por fim, iniciar a transferência e receção de mensagens de forma segura e corretamente direcionada entre Dispositivos.

Na secção 5.2.4 será explicada com mais detalhe a estrutura de execução do programa de um Dispositivo Remoto, onde estará incluída esta parte da interação com o Rádio.

5.2.4 Estruturação do Programa de um Dispositivo Remoto

Fluxo de Execução do *Arduino*

Um *Arduino* não utiliza um Sistema Operativo para controlar as suas operações portanto, o seu fluxo de execução tem por base duas funções:

- *Setup*: Esta função é executada uma única vez, quando o *Arduino* é ligado, e tem como objetivo a inicialização de variáveis, a definição dos tipos dos pinos a serem utilizados e a execução de alguma lógica que pode estar associada a bibliotecas;

- *Loop*: Após o *Setup* ter terminado, esta função vai ser executada consecutivamente, permitindo um controlo constante e ativo do ambiente proporcionado por esta plataforma.

Definição de Propriedades e Variáveis Globais

Como foi mencionado na secção 5.2.1, através da utilização/alteração de umas propriedades definidas no início do programa, este deve ser capaz de funcionar corretamente ao ser colocado num Dispositivo Remoto que utilize um dos seguintes periféricos suportados:

- Atuador (Relé)
- Botão
- Sensor de Contacto
- Sensor de Temperatura

Estas propriedades representam de uma forma técnica o periférico que vai ser utilizado, servindo assim de fator de decisão durante o funcionamento do programa. A Figura 28 demonstra a representação de um sensor de temperatura.

```
//*************************************************************************************************
//***************************** IMPORTANT REMOTE DEVICE SETTINGS ***********************************
//*************************************************************************************************
//+++++++++++++++++++++ CHOOSE THE TYPE OF DEVICE ACCORDING TO ITS FUNCTION ++++++++++++++++++++++
//*************************************************************************************************
//#define DEVICE_TYPE "OUTPUT" // RELAY, LED ...
#define DEVICE_TYPE "INPUT"
String deviceType;
//*************************************************************************************************
//++++++++++++++++++++++++ IF THE DEVICE TYPE IS INPUT CHOOSE IT'S TYPE ++++++++++++++++++++++++++
//*************************************************************************************************
//#define INPUT_TYPE "null"
//#define INPUT_TYPE "button"
//#define INPUT_TYPE "contactsensor"
#define INPUT_TYPE "temperaturesensor"
String inputType;
//*************************************************************************************************
```

Figura 28 – Propriedades de Representação do Periférico utilizado pelo Dispositivo Remoto

Para suportar os sensores de temperatura, as bibliotecas necessárias têm de ser também instanciadas como variáveis globais (Figura 29) para poderem ser utilizadas durante a função de *Setup* e durante o processamento do *Loop*. Para além disso, tem de ser reservado um pino (nº 5) no *Arduino,* onde estará associado o barramento *One-Wire*, tal como acontece no *Raspberry Pi*.

```
#define ONE_WIRE_BUS 5
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
```

Figura 29 – Definição das Variáveis Globais para o uso de Sensores de Temperatura no Arduino

Os restantes periféricos podem partilhar o mesmo pino (nº 4), visto que, o modo de funcionamento deste será definido dinamicamente durante a função de *Setup*.

```
#define PERIPHERAL_PIN 4
// to check the button or similar peripheral state
boolean inputHigh = false;
```

Figura 30 – Definição das Variáveis Globais para o uso dos Restantes Periféricos

Relativamente ao rádio, como explicado na secção 5.2.3, também é criado um objeto global deste, para que possa ser posteriormente utilizado no *Setup* e no *Loop*.

```
#define IS_RFM69HCW    true // Is High Power Version?
#define RFM69_CS       10   // Chip/Slave Select Pin
#define RFM69_IRQ      3    // Interrupt Pin
#define RFM69_IRQN     0    // Interrupt Number
#define RFM69_RST      9    // Pin to Reset Radio

RFM69 radio = RFM69(RFM69_CS, RFM69_IRQ, IS_RFM69HCW, RFM69_IRQN);
```

Figura 31 – Definição das Variáveis Necessárias para a Utilização do Rádio

Definição do *Setup*

Nesta parte do programa, são feitas as seguintes tarefas:

- *Reset* do rádio, para que esteja num estado neutro;
- Inicialização do rádio;
- No caso dos sensores de temperatura é inicializado o barramento;
- No caso dos outros periféricos é definido o modo (*Input* ou *Output*) do pino a que estão associados.

Definição do *Loop*

Esta parte do programa é executada consecutivamente de uma forma cíclica, segundo o diagrama de fluxo da Figura 32.

Este fluxo de eventos começa com o processo de vinculação do Dispositivo Remoto com o *HUB*. Apenas quando esta vinculação é concluída com sucesso é que o Dispositivo fica inteiramente funcional.

Portanto, tendo sido concluída com sucesso, passa-se ao processo de verificação de receção de mensagens. Se existir uma mensagem recebida, é enviada a confirmação de receção, a mensagem é processada e a ordem nela contida é executada.

O passo seguinte é executado independentemente de se ter recebido uma mensagem ou não e tem o objetivo de observar alterações ao estado dos Dispositivos do tipo *Input*, excluindo os sensores de temperatura, tal como é feito no *Raspberry Pi*, na secção 5.1.2.2. Caso seja detetada uma alteração é enviada uma mensagem a informar o *HUB*.

Tendo em conta que o processo de vinculação já foi concluído uma vez, a partir deste ponto, o fluxo volta sempre à verificação de receção de mensagens.

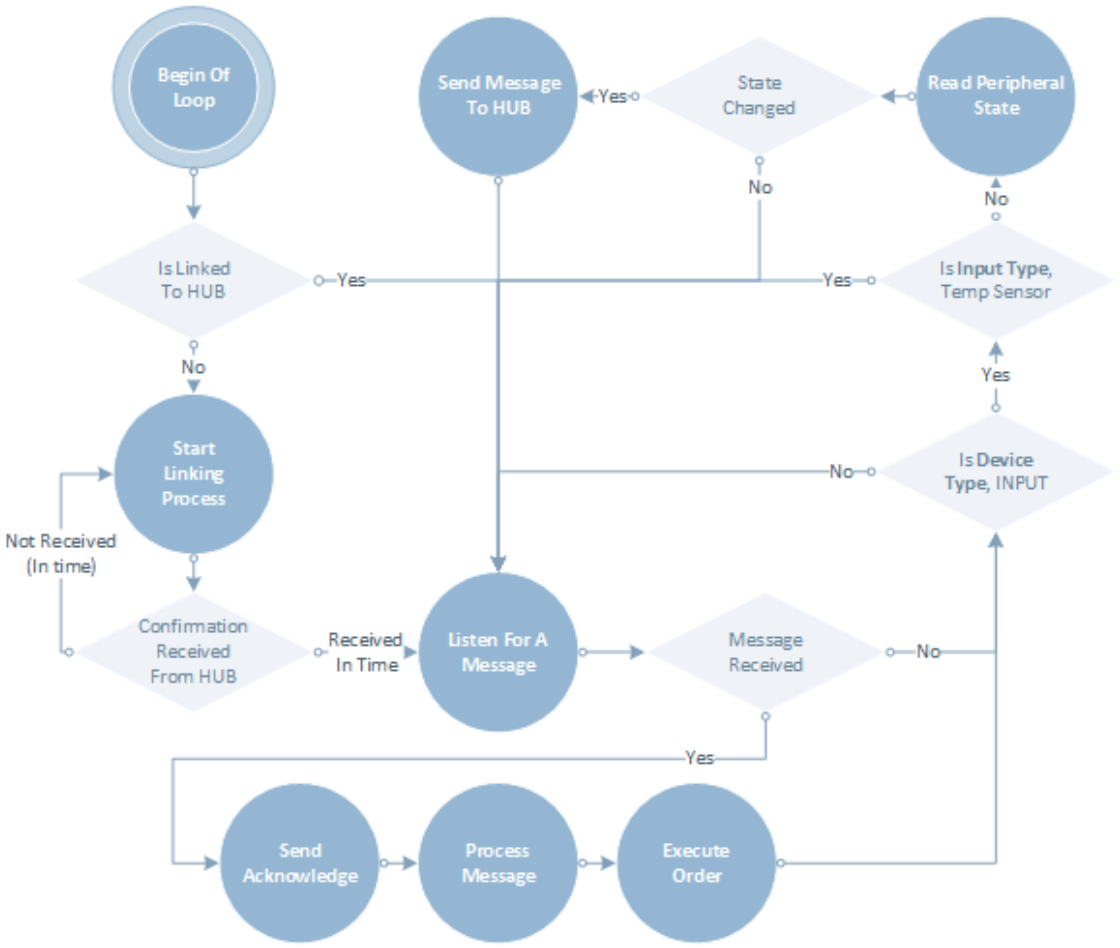


Figura 32 – Fluxo de Eventos da Função Loop dos Dispositivos Remotos

Processo de Vinculação

O processo de vinculação é uma forma de um Dispositivo Remoto se dar a conhecer ao *HUB* e garantir que este sabe da sua presença na rede.

Este não é propriamente um processo de segurança, pois esse está maioritariamente assegurado pela Chave de Encriptação, mas antes um processo de gestão para que as aplicações cliente possam visualizar de forma imediata os Dispositivos Remotos existentes na rede.

5.3 Integração do HUB com os Dispositivos Remotos

Para que fosse possível gerir a rede dos Dispositivos Remotos e controlá-los através da aplicação cliente, foi necessário integrar no *HUB* o mesmo tipo de rádio utilizado pelos Dispositivos Remotos.

5.3.1 Interação entre o Raspberry Pi e o Rádio RFM69

A interação entre estes dois componentes foi realizada através do ambiente de desenvolvimento utilizado no *Raspberry Pi*, o *Node.js*, portanto, a ideia mais correta seria utilizar uma biblioteca que funcionasse sobre este e que fosse capaz de comunicar diretamente com o rádio.

Tal biblioteca foi encontrada, a *RFM69* fornecida por *dconstructing* na plataforma *GitHub*, no entanto, após a execução de alguns testes verificou-se que a mesma não estava a funcionar corretamente, pois não estava a conseguir enviar mensagens para outro rádio.

A Figura 33 representa uma parte das mensagens de teste e demonstra a tentativa falhada de enviar uma mensagem, com a justificação de que esta tinha excedido o tempo limite.

```
switched to transmit mode
register write 5a => 1011101 (was 1010101 )
register write 5c => 1111100 (was 1110000 )
transmit initiated
send timed out <Buffer 80 0e 11 64 40 48 65 6c 6c 6f 20 57 6f 72 6c 64> <Buffer 00 00 0e 0e 0e 0e 0e 0e 0e 0e 0e 0e 0e 0e 0e 0e>
Error sending to mote [Error: Send timed out]
```

Figura 33 – Falha no Envio de Mensagem Utilizando a Biblioteca *RFM69* para *Node.js*

Da mesma forma que o *Arduino*, a comunicação entre o *Raspberry Pi* e o rádio é feita sobre o barramento *SPI*, tal como foi explicado na secção 5.2.3. Visto que o objetivo desta biblioteca é abstrair o tratamento deste barramento, esta utiliza uma outra biblioteca, a *Pi-SPI* fornecida por *natevw* na mesma plataforma, a qual realiza as interações de baixo nível entre o *Raspberry Pi* e o rádio.

Tendo conhecimento desta biblioteca, foi iniciada uma comunicação, no dia 16 de Junho de 2016, com o seu autor e com outra pessoa que a estava a testar na mesma altura, para que fosse possível verificar se existia algum problema com esta. Após várias trocas de mensagens, ao longo de duas semanas, chegou-se à conclusão de que esta estava a funcionar corretamente, direcionando-se assim, o problema, de volta para a biblioteca *RFM69*.

Após esta investigação, o problema foi registado na página da biblioteca, no dia 7 de Julho de 2016 e, até ao momento da escrita deste relatório, ainda não obteve resposta por parte do seu autor. Este registo pode ser visualizado através do seguinte *URL*:

https://github.com/dconstructing/rfm69/issues/8

Uma vez que esta biblioteca era a única existente para *Node.js*, capaz de controlar o rádio utilizando o barramento *SPI* e, tendo em conta que a única maneira de o controlar é através deste barramento, foi necessário encontrar uma solução que adaptasse a ligação entre o *Raspberry Pi* e o rádio.

Tendo obtido conhecimento na criação dos Dispositivos Remotos, sobre como interagir com um rádio, utilizando um *Arduino*, decidiu-se então aplicar o mesmo na criação de um adaptador capaz de interagir com o *Raspberry Pi*. Para isto, foi necessário pesquisar sobre

como realizar a comunicação entre o *Raspberry Pi* e o *Arduino*, tendo surgido duas alternativas.

A **primeira alternativa** que surgiu foi a utilização do barramento *I²C* (*Inter-Integrated Circuit*). Este barramento, em comparação com o *SPI*, utiliza uma arquitetura de *multi-master* e *multi-slave*, permite uma ligação física mais simples (2 linhas apenas), permite transferências de dados de forma bidirecional através de uma única linha (*SDA*) e possui um limite na sua velocidade, sendo assim mais lento que o *SPI*. (Oudjida et al. 2009)

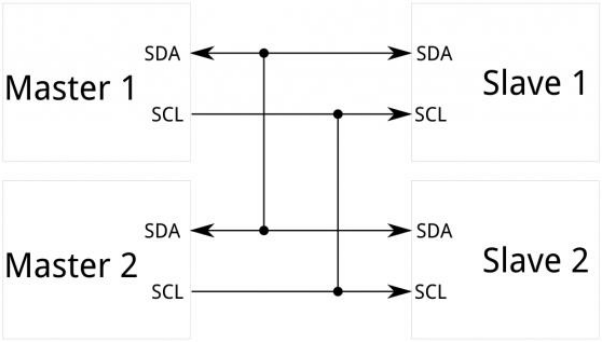


Figura 34 – Exemplificação da Comunicação *I²C multi-master* e *multi-slave* (sparkfun.com)

Considerando estes aspetos, foi dada uma oportunidade a esta alternativa, tendo-se chegado a um ponto em que se descobriu que não seria possível que o *Raspberry Pi* fizesse ambos os papéis de *master* e *slave*, visto não existir nenhuma biblioteca para *Node.js* que suportasse o registo dos eventos de escuta que um *slave* precisaria.

A necessidade para que este fizesse ambos os papéis deve-se ao facto de que, apesar de ser bidirecional, a comunicação do barramento *I²C* tem de ser sempre iniciada por um *master*. Ou seja, um *slave* não pode enviar nada para um *master* a não ser que este o requisite.

"*The I2C bus "Master" is the component that initializes a transfer, generates the SCL clock, and ends the transfer sequence. The Master can be the Transmitter or Receiver depending on whether it is writing to or reading from a device. The I2C bus "Slave" is the device addressed by the Master and can be a Transmitter or Receiver depending on whether the Master is writing or reading.*" (Myers 2007)

Para continuar com esta estratégia teria de se implementar a técnica de *Polling*, do *Raspberry Pi* para o *Arduino*, o que traria limitações na comunicação em tempo real, de forma similar ao que foi concluído na secção 5.1.2.4. Por esta razão, esta foi descartada.

A **segunda alternativa** que surgiu foi a utilização do barramento *USB* (*Universal Serial Bus*). Em comparação com os já conhecidos, *SPI* e *I²C*, este possui uma interface de comunicação série assíncrona, onde a sincronização da transferência de dados já não é suportada por um sinal de relógio (*SCK* no *SPI* e *SCL* no *I²C*), mas sim através da especificação direta na estrutura de uma mensagem, dos *bits* de início e fim da mesma. A comunicação é bidirecional, permitindo que

ambos os componentes enviem e recebam mensagens simultaneamente, e a velocidade de transferência de dados é superior à dos anteriores.

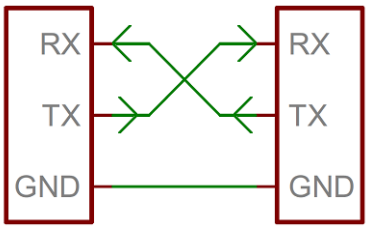


Figura 35 – Exemplificação da Comunicação *USB* Entre Dois Componentes (sparkfun.com)

Para a implementação desta estratégia, do lado do *Node.js*, foi utilizada a biblioteca *Node-SerialPort* fornecida por *EmergingTechnologyAdvisors* na plataforma *GitHub*, a qual permite o envio de mensagens para um dispositivo específico neste barramento e, uma escuta ativa para a receção de mensagens provenientes do referido dispositivo.

Esta escuta, após a receção da mensagem, permite aplicar um processo de triagem ao seu conteúdo, onde este é analisado para que possa ser corretamente executado.

Do lado do *Arduino*, que funciona como adaptador, não foi necessária uma biblioteca extra, visto que o acesso à comunicação *USB* já é disponibilizado sobre a forma do objeto *Serial*. Este objeto permite fazer o envio de mensagens e verificar se alguma mensagem foi recebida.

Após alguns testes, concluiu-se que esta seria uma alternativa viável para a comunicação entre o *Raspberry Pi* e o *Arduino*, pois garante uma baixa latência na transferência de dados e permite que ambos tenham a oportunidade de enviar mensagens quando quiserem.

Adotando esta estratégia fica-se então com a arquitetura de comunicação representada pela Figura 36.

Figura 36 – Representação da Arquitetura de Comunicação para a Interação entre o *HUB* e o rádio

Posto isto, avançou-se com o desenvolvimento de um módulo para *Node.js*, para abstrair a interação com o adaptador, a fim de este ser facilmente integrado com o sistema de comunicação e persistência do *HUB*. Foi também desenvolvido o programa para executar no *Arduino* do adaptador, com o objetivo de fornecer uma interface de manipulação sobre o rádio, ao módulo do *Node.js*.

5.3.2 Estruturação do Módulo Node.js de Interação com o Adaptador

Tal como foi mencionado anteriormente, foi criado um módulo à volta da biblioteca *Node-SerialPort*, para ser possível utilizar o *Node.js* para interagir com o adaptador através do barramento *USB.*

Este módulo indica à biblioteca o identificador da ligação *USB* que o adaptador está a utilizar, para que esta possa abrir uma ligação entre estes dois componentes. Esta ligação é constantemente verificada para que, caso o adaptador reinicie, a ligação possa também ser reiniciada.

Tendo em conta que é o módulo de comunicações que recebe os pedidos de um utilizador e os tem de redirecionar, caso seja necessário contactar com um dispositivo remoto, este terá de interagir com o módulo aqui apresentado. Esta interação é feita através de um conjunto de funções (Figura 37) que permite a correta estruturação da mensagem a enviar, tendo em conta a informação que se quer passar para o dispositivo remoto.

```
//------------------------------------------------------------------//
// DESCRIPTION: Write to the serial port
//------------------------------------------------------------------//
function writeToSerial(packet, callback){
    serialPort.write(packet, function (err) {
        callback(err);
    });
}

//------------------------------------------------------------------//
// DESCRIPTION: Send a write request
//------------------------------------------------------------------//
function sendWrite(remDevId, state, callback){
    var packet = "{W|" + remDevId + "|" + state + "}";
    writeToSerial(packet, callback);
}
module.exports.sendWrite = sendWrite;

//------------------------------------------------------------------//
// DESCRIPTION: Send a read request
//------------------------------------------------------------------//
function sendRead(remDevId, callback){
    var packet = "{R|" + remDevId + "}";
    writeToSerial(packet, callback);
}
module.exports.sendRead = sendRead;
```

Figura 37 – Funções do Módulo de Interação com o Adaptador Acedidas pelo Módulo de Comunicações

Para que o módulo de comunicações possa receber uma resposta a uma mensagem enviada, durante um tempo limitado é registada uma *callback* no módulo de interação com o adaptador, para que este a possa utilizar para, posteriormente, fornecer a resposta.

Para ser possível receber mensagens vindas do adaptador, esta biblioteca permite associar uma função de *callback* ao evento de receção de dados.

```
serialPort.on('data', function (data) {
    // ...
});
```

Figura 38 – Definição do Evento de Receção de Dados e Associação de *Callback*

Nesta função é feita uma análise à mensagem recebida para verificar se a sua estrutura está de acordo com a definida na secção 5.3.4, de maneira a que seja possível separar a mensagem em diferentes partes, que vão indicar a ação a realizar, o identificador do dispositivo remoto, entre outros.

Através da ação a realizar, é possível saber o tipo de tarefa a executar e, caso tenha sido definida uma *callback* por parte do módulo de comunicações, esta será utilizada para passar a informação necessária.

As ações admitidas nesta função são de seguida apresentadas, juntamente com os respetivos diagramas, que representam as suas sequências de eventos:

- *Input Event* – Indica o estado atual de um dispositivo remoto após este ter sido alterado devido a uma interação física com o mesmo;

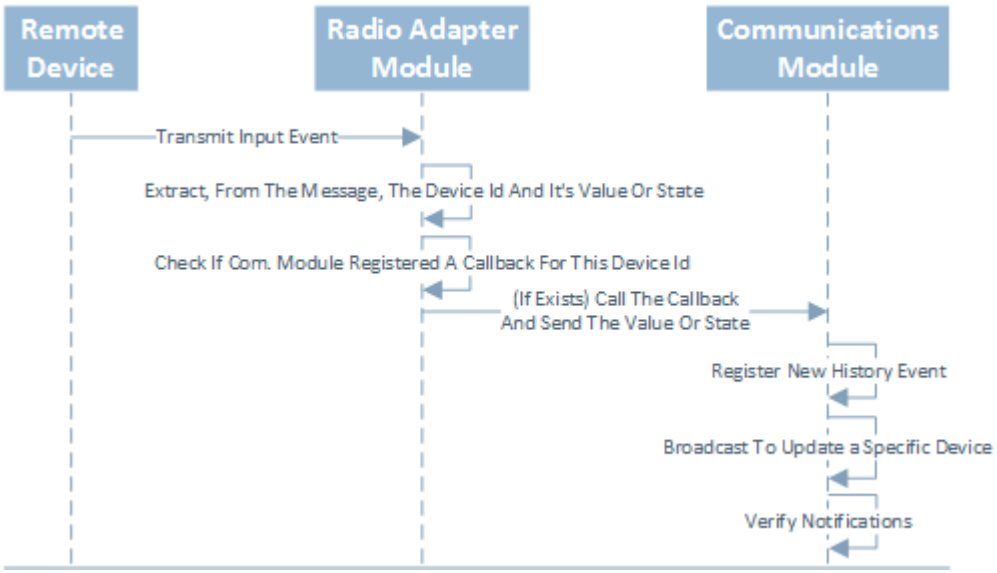


Figura 39 – Diagrama de Sequência da Receção de um Evento Externo Ocorrido num Dispositivo Remoto

- *Acknowledge* – Informa que uma mensagem chegou a um dispositivo remoto;

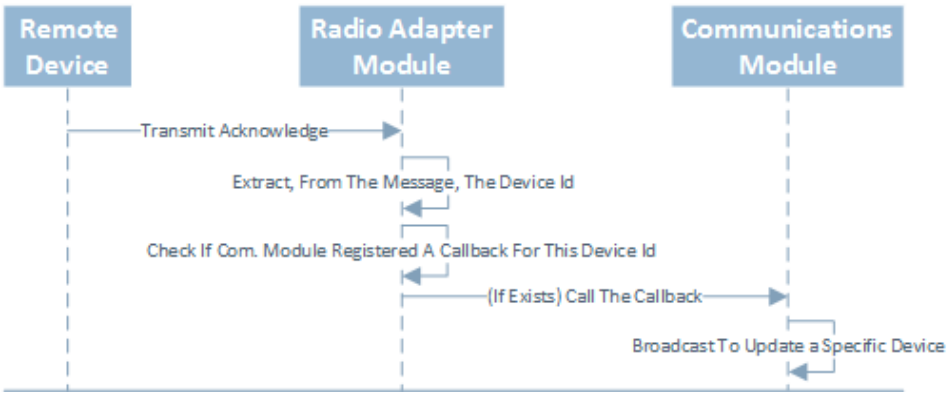


Figura 40 – Diagrama de Sequência da Receção de uma Confirmação vinda de um Dispositivo Remoto

- *Read* – Fornece a leitura do estado de um dispositivo remoto, e é recebida em reposta a um pedido de leitura;

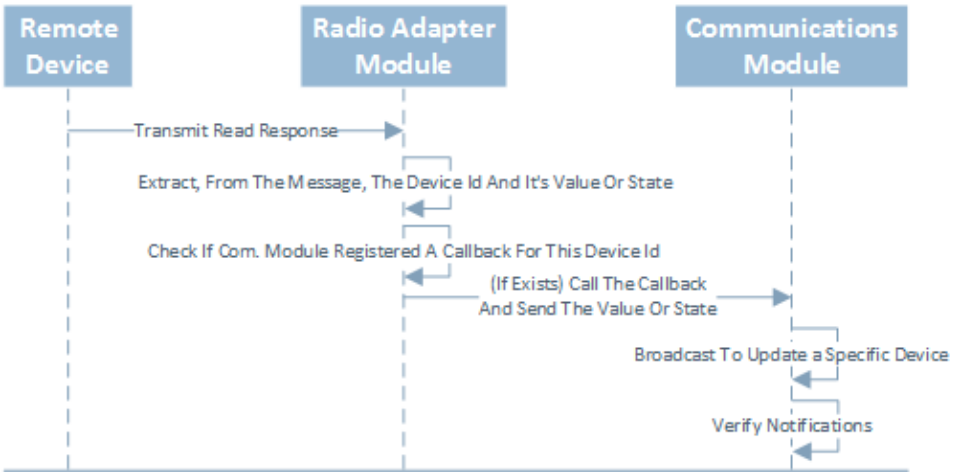


Figura 41 – Diagrama de Sequência da Receção da Resposta a uma Leitura de um Dispositivo Remoto

- *Link Request* – Recebe as informações de um dispositivo remoto que está a inicializar o processo de vinculação com o *HUB*.

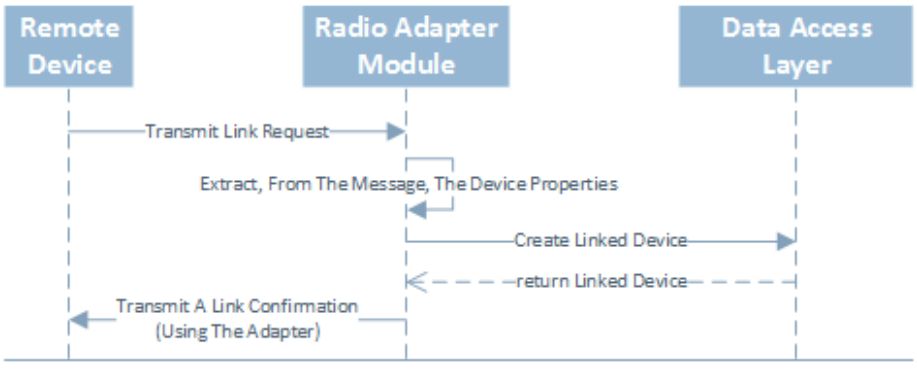


Figura 42 – Diagrama de Sequência da Receção de um Pedido de Vinculação de um Dispositivo Remoto

5.3.3 Estruturação do Programa do Adaptador

A estrutura do programa do adaptador utiliza a mesma lógica de inicialização do rádio e segue os mesmos princípios base do programa dos Dispositivos Remotos, apresentado na secção 5.2.4. Portanto, a parte mais importante a abordar relativamente ao adaptador é a da definição do *Loop*, representada pelo diagrama de fluxo da Figura 43.

Neste, o fluxo de eventos começa com uma leitura do *buffer* de mensagens do barramento *USB*, para verificar a existência de uma mensagem proveniente do *HUB*.

Se for encontrada uma mensagem, esta é analisada para se obter algumas informações, como a ação requisitada e o identificador do destinatário, depois é convertida para um formato requerido pelo rádio e, por fim, é transmitida.

De seguida, ou caso não se tenha encontrado uma mensagem, passa-se ao processo de verificação de receção de mensagens via rádio. Se existir uma mensagem recebida, é enviada a confirmação de receção e a mensagem é redirecionada diretamente para o *HUB* através do barramento *USB*.

Independentemente de se ter recebido uma mensagem via rádio ou não, a partir deste ponto o fluxo volta sempre à leitura do *buffer* de mensagens do barramento *USB*.

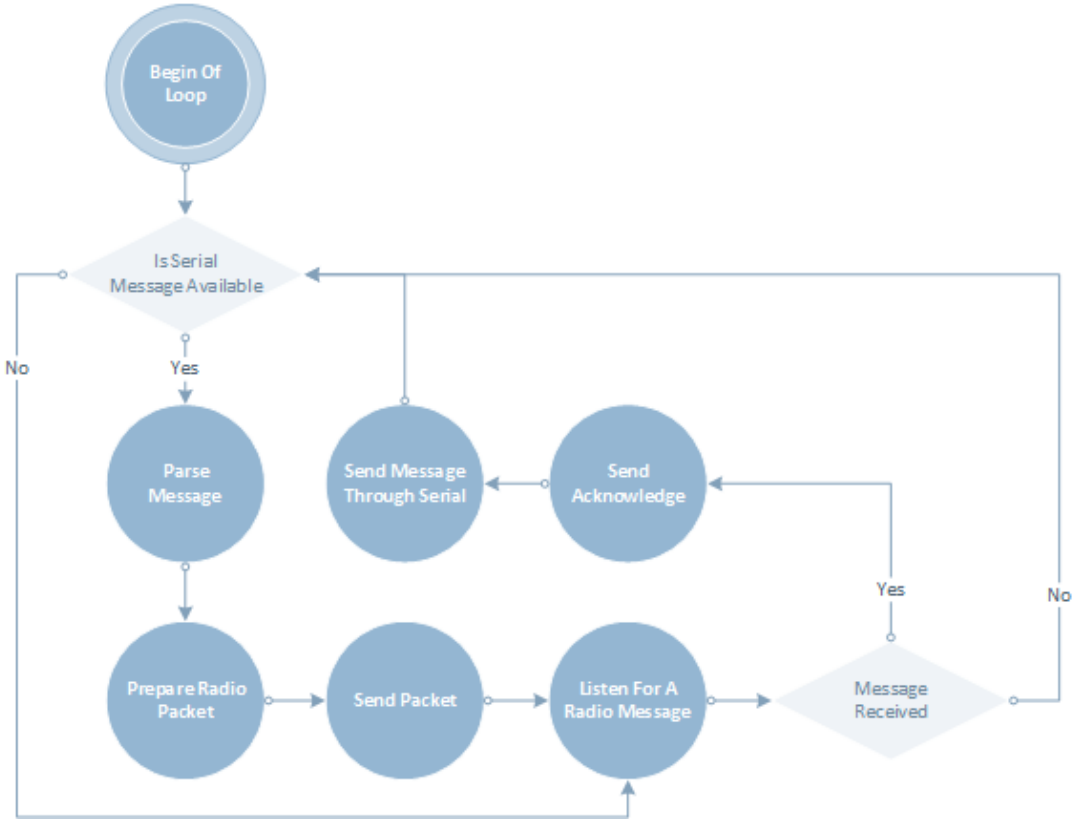


Figura 43 – Fluxo de Eventos da Função Loop do Adaptador

5.3.4 Definição da Estrutura da Mensagem

Para uniformizar a análise das mensagens enviadas e recebidas via rádio, tanto no *HUB,* como no adaptador e nos Dispositivos Remotos, foi definida a estrutura apresentada pela Figura 44.

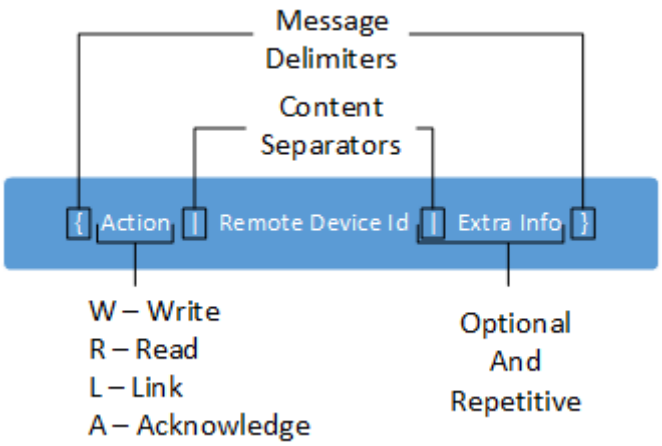


Figura 44 – Estrutura de uma Mensagem Transmitida via Rádio

Alguns exemplos de mensagens enviadas do *HUB* para um Dispositivo Remoto:

- {W|002|1} – Se o Dispositivo for do tipo *Output*, por exemplo, um atuador a controlar uma luz, ao receber esta mensagem vai interpretar que tem de a ligar.
- {R|002} – Ao receber esta mensagem, o Dispositivo irá enviar de volta o estado ou o valor associado a este, tendo em conta o seu tipo. Se for um sensor de temperatura, irá enviar a medição atual.
- {L|002} – Ao receber esta mensagem, o Dispositivo obtém a confirmação de que o processo de vinculação foi terminado com sucesso.

Alguns exemplos de mensagens enviadas de um Dispositivo Remoto para o *HUB*:

- {L|002|INPUT|temperaturesensor} – Ao receber esta mensagem, o *HUB* inicia o processo de vinculação do Dispositivo Remoto com o identificador número 2, do tipo *Input*, mais especificamente, um sensor de temperatura.
- {R|002|25.3} – Esta é uma mensagem recebida em resposta a um pedido de leitura a um sensor de temperatura, cuja medição foi de 25.3 graus Celcius.

5.4 Aplicação Android

A aplicação *Android* é, neste momento, a única interface que uma pessoa pode utilizar para fazer a gestão do *HUB* e dos Dispositivos Remotos. Esta foi desenvolvida para suportar dispositivos *Android* a partir da versão 4.0 (*Ice_Cream_Sandwich*) até à mais recente 7.0 (*Nougat*), o que cobre cerca de 97.4% dos dispositivos ativos na *Google Play Store*. Foi

também incluído suporte para a Língua Portuguesa, visto que este projeto foi desenvolvido e testado em Portugal, para além da Língua Inglesa que está definida por padrão.

Ao longo deste capítulo serão descritas as funcionalidades da aplicação, utilizando imagens provenientes de um dispositivo *Android* com a versão 5.0 (*Lollipop*).

5.4.1 Serviço de Comunicações

Persistência de Ligações

Para que esta aplicação pudesse oferecer uma experiência de comunicações em tempo real, esta teria de manter, constantemente ativas, as ligações com os *HUBs* geridos, mesmo quando a aplicação não estivesse a ser utilizada pela pessoa. Portanto, para que este objetivo fosse concretizado, foi necessário criar um serviço (*Real Time Service*) que executasse em segundo plano, ficando responsável pela gestão das comunicações via *Socket.IO* e, servindo também de interface para o acesso aos recursos de cada *HUB*, por parte das *activities*.

Interface para *Activities*

Uma *activity* tem acesso a este serviço através da criação de um *binder* associado a este, colocando assim os seus recursos públicos à disposição desta. Este *binder* fica ativo apenas durante o tempo em que a *activity* estiver visível, visto que, só aí é que o utilizador poderá realizar uma ação que necessite do acesso aos recursos de um *HUB*.

Representação das Ligações

As ligações são representadas através de um objeto *Real Time Communication*, o qual permite a associação de um *HUB* com um *socket*, necessário para comunicações via *Socket.IO*. Este objeto possui toda a lógica necessária para a escuta de eventos provenientes do *HUB* e para a emissão de eventos, de forma assíncrona, por parte da aplicação.

Os eventos que podem ser emitidos são os apresentados na secção 5.1.2.5 e os que podem ser recebidos são os seguintes:

- *Authenticated*
- *Unauthorized*
- *Ping*
- *UpdateDevices*
- *UpdateSpecificDevice*
- *UpdateNotifications*
- *DisplayNotification*

Controlo e Feedback sobre o Estado de uma Ligação

Assim que um *socket* é conectado e devidamente autorizado, é feito um *broadcast* dentro da aplicação para avisar as *activities* interessadas de que foi estabelecida uma ligação com o

HUB. O mesmo acontece quando uma ligação é perdida ou incapaz de ser estabelecida. Neste caso, as activities recebem esta informação e bloqueiam o uso de qualquer recurso relacionado com o *HUB* em questão, mostrando uma mensagem de feedback (Figura 45) e permitindo apenas o redireccionamento para a página de gestão dos *HUBs*.

Figura 45 – Mensagem de Feedback sobre a Indisponibilidade de um *HUB*

As interações com este serviço de comunicações são representadas pelo diagrama de sequência presente na Figura 46.

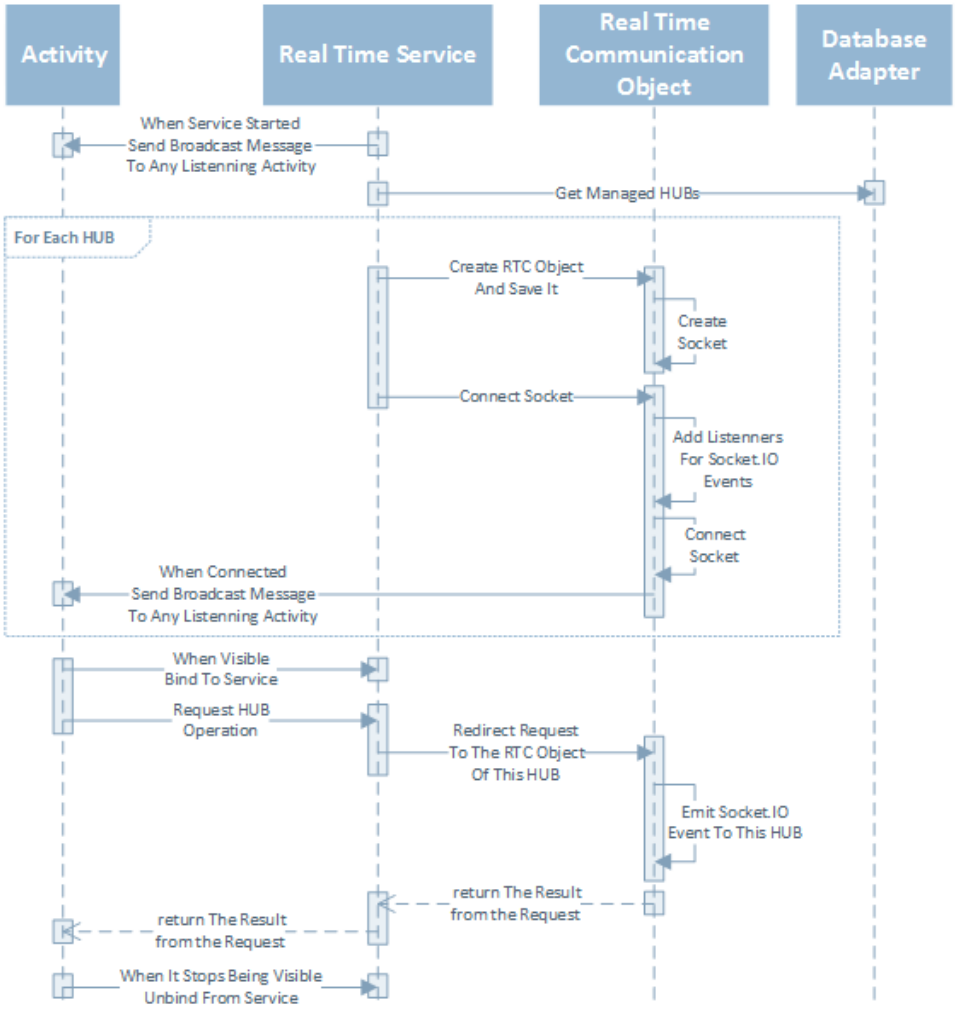


Figura 46 – Digrama de Sequência das Interações com o Serviço de Comunicações da Aplicação Android

5.4.2 Autenticação Utilizando a API REST

Tal como foi mencionado na secção 5.1.2.4, o *HUB* expõe os pontos de acesso para o processo de autenticação, utilizando a *API REST*.

Para que a aplicação cliente pudesse realizar a autenticação, foi criada a classe *REST Service Gateway*, onde reside a lógica de acesso a esta *API*. Esta é a base de comunicações necessária para a gestão de *HUBs,* e será integrada no processo apresentado na secção 5.4.3.

Nesta classe, as comunicações com a *API,* são efetuadas através de ligações *HTTP* criadas a partir da classe *Http URL Connection,* pertencente ao *Java*. Esta permite definir um tempo máximo de ligação e de espera por uma resposta, e definir o tipo de comunicação a efetuar consoante a nomenclatura *REST*. Tendo a ligação preparada, são definidos os parâmetros a enviar e é iniciada a comunicação. Quando a resposta chega, esta é capturada e posteriormente processada.

Utilizando este sistema, a *REST Service Gateway* é capaz de fazer a autenticação de um *HUB*, utilizando as credenciais de um utilizador (Figura 47), ou o *Refresh Token* associado aos detalhes de uma prévia autenticação (Figura 48).

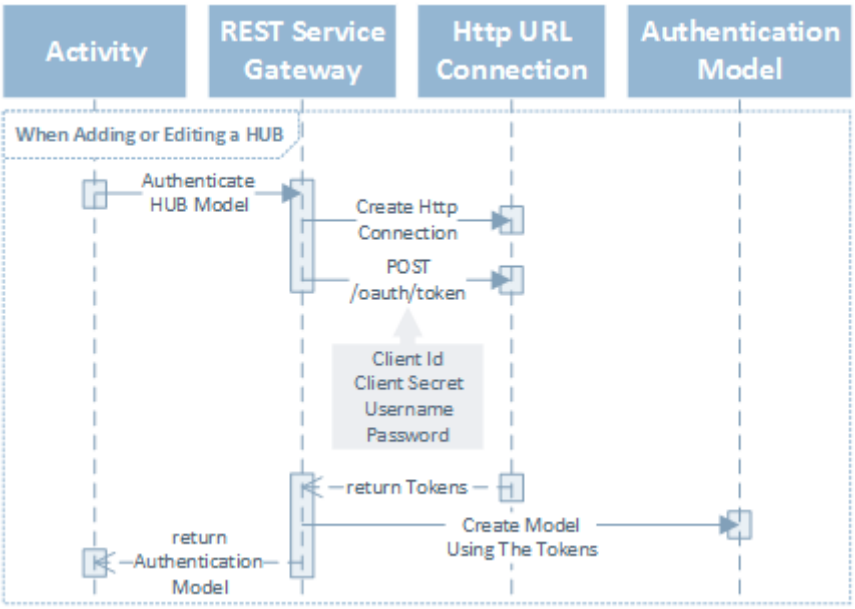


Figura 47 – Autenticação de um *HUB* Através da *API REST* Utilizando as Credenciais do Utilizador

Tendo em conta que o serviço de comunicações da secção 5.4.1 é que trata da gestão das ligações dos *sockets*, apenas este tem a necessidade de utilizar o segundo método de autenticação.

Ou seja, no processo de autorização de um *socket*, que acontece após a sua conexão, vai ser necessário utilizar o *Access Token*, no entanto, se este já tiver expirado antes de se proceder a uma nova tentativa de autorização, terão de se requisitar novos *Tokens*, realizando-se assim a autenticação através do *Refresh Token*.

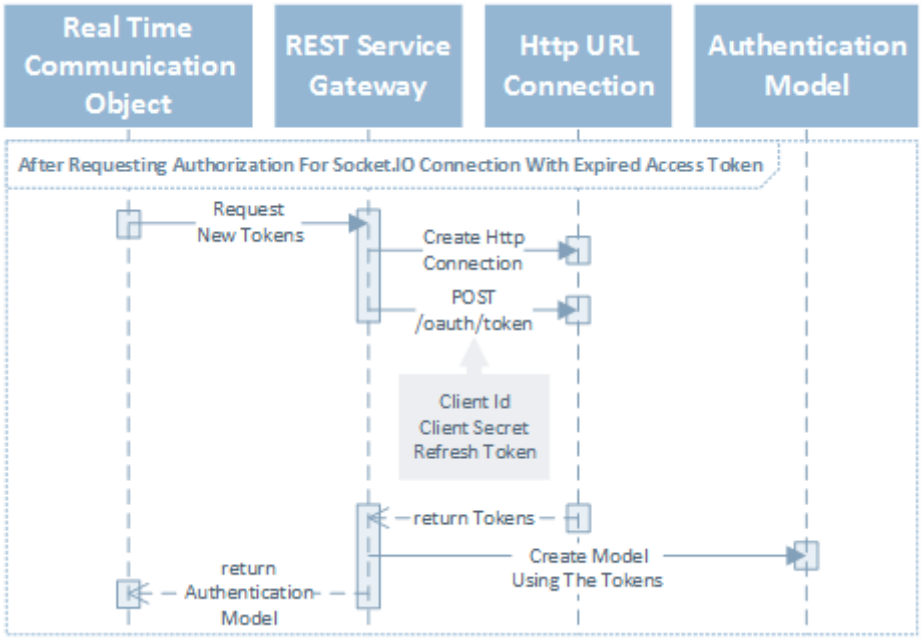


Figura 48 – Autenticação de um *HUB* Através da *API REST* Utilizando o *Refresh Token*

5.4.3 Gestão de Controladores/HUBs

Figura 49 – Lista de *HUBs*

Para permitir a gestão de múltiplas casas ou edifícios, a aplicação possibilita a adição de vários *HUBs*.

A página dos *HUBs* (Figura 49) apresenta os detalhes chave de cada *HUB*, assim como o estado da sua ligação. A partir desta página, utilizando o botão que se encontra à esquerda de cada *HUB*, é possível aceder a um menu de opções (Figura 50), para fazer a sua edição ou remoção.

Figura 50 – Menu de Opções de um *HUB*

Para se escolher o *HUB* que se quer aceder/visualizar, ao longo da aplicação, este deve ser selecionado clicando na estrela cinzenta que passará a amarela. Se apenas tiver sido adicionado um *HUB* à aplicação, este será automáticamente escolhido.

Figura 51 – Adicionar *HUB*

Para adicionar um *HUB* (Figura 51), o utilizador terá de fornecer um nome descritivo, o endereço IP e a porta que o identificam, e os detalhes de uma conta de acesso.

Tendo o formulário preenchido e avançando com a ação de adicionar o *HUB*, é iniciado o processo de autenticação, utilizando a *API REST*, onde serão validados os dados inseridos.

Se tudo estiver correto, o *HUB* e os *Tokens* recebidos após a autenticação são registados na base de dados local da aplicação.

Figura 52 – Editar *HUB*

Ao editar um HUB (Figura 52), os dados deste são carregados para o formulário, à exceção da palavra-passe, a qual terá de ser novamente introduzida devido à política da aplicação de não guardar palavras-passe para segurança do utilizador.

Tendo sido feitas as alterações necessárias, ao confirmar a edição, é iniciado o processo de autenticação tal como é feito quando se adiciona um *HUB*.

Sempre que um *HUB* for adicionado, editado ou removido com sucesso, o serviço de comunicações é notificado para que possa fazer a gestão adequada do objeto *Real Time Communication* associado a este.

O processo de cada uma destas funcionalidades pode ser revisto no diagrama de sequência presente na Figura 53.

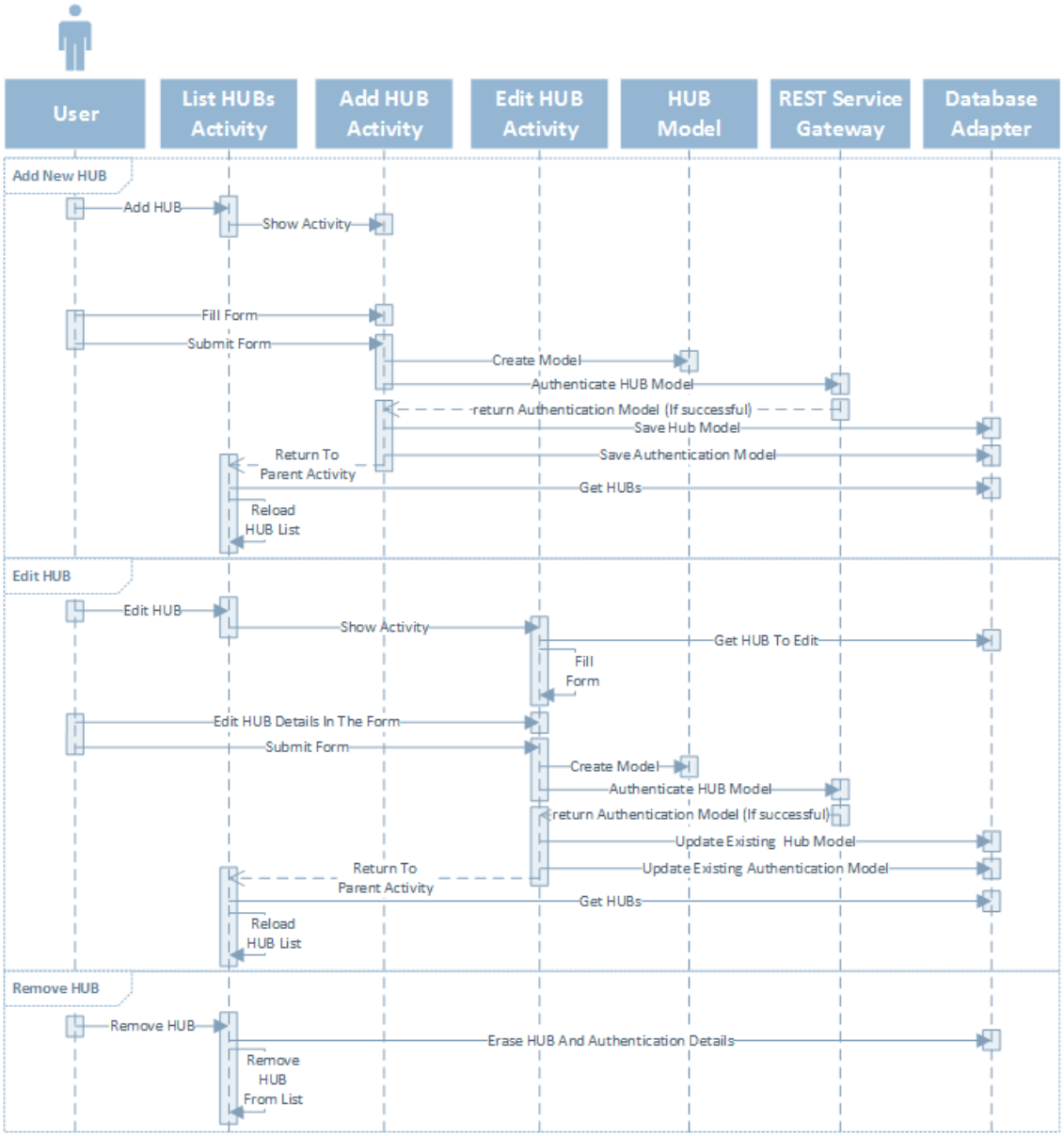


Figura 53 – Diagrama de Sequência das Ações Relacionadas com *HUBs*

5.4.4 Gestão de Dispositivos

Figura 54 – Lista de Dispositivos

A página dos dispositivos (Figura 54), assim que o utilizador tiver adicionado um HUB, será a primeira a ser vista quando a aplicação é aberta.

Esta serve como centro de controlo e monitorização, onde o utilizador pode interagir com os dispositivos criados e visualizar os seus estados.

Para que o utilizador saiba qual o *HUB* que está a gerir, este é sempre apresentado no topo da lista dos dispositivos.

Tal como nos *HUBs*, também é possível aceder a um menu de opções (Figura 55) em cada um dos dispositivos, para fazer a sua edição ou remoção.

Figura 55 – Menu de Opções de um Dispositivo

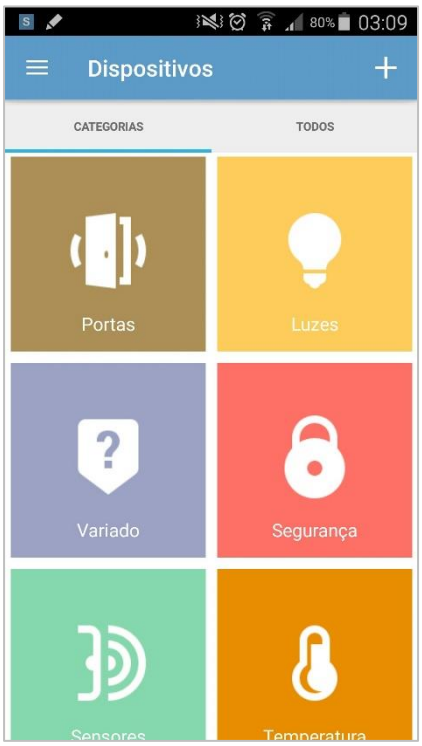


Figura 56 – Lista das Categorias dos Dispositivos

Para facilitar a gestão dos dispositivos, o utilizador pode aceder à secção das categorias (Figura 56) e selecionar a que pretende, para depois ser redirecionado para uma página onde serão apresentados apenas os dispositivos dessa categoria.

Por exemplo, se for escolhida a categoria Temperatura, será apresentada uma lista (Figura 57) com os dois dispositivos existentes que estão associados a esta, e que neste caso são sensores de temperatura.

Figura 57 – Lista de Dispositivos da Categoria Temperatura

Para adicionar um dispositivo, o utilizador terá de introduzir um nome descritivo e escolher a categoria que mais se adequa.

De seguida terá de escolher o tipo de ligação que o dispositivo utiliza, Local ou Remota e, a partir daí, poderá seguir um dos caminhos apresentados na Figura 58, para finalizar o preenchimento do formulário.

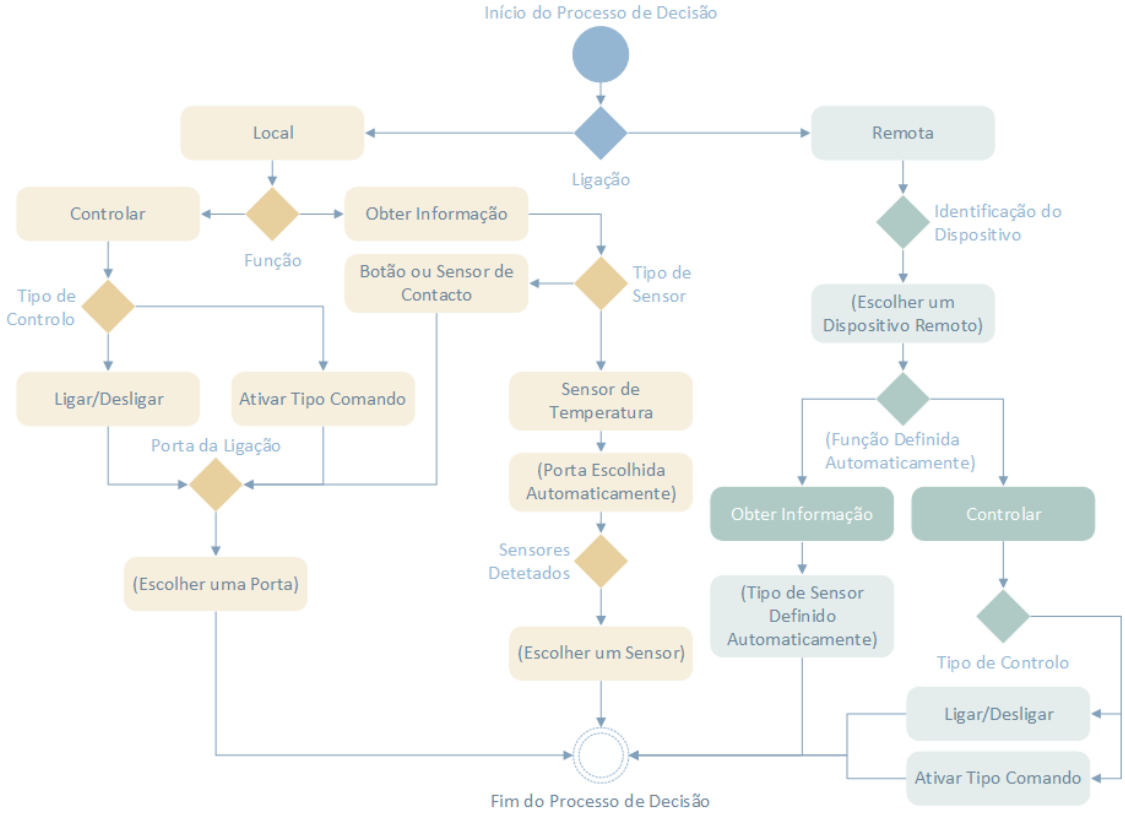


Figura 58 – Diagrama de Atividades do Preenchimento do Formulário de Adição de um Dispositivo

A informação necessária para adicionar um dispositivo local é mais extensa, visto que, o *HUB* não tem conhecimento sobre o que está fisicamente ligado a ele, contrariamente aos dispositivos remotos, que fornecem alguma informação ao *HUB* no seu processo de vinculação.

As Figuras Figura 59 e Figura 61 demonstram a adição de um candeeiro como dispositivo local e remoto, respetivamente, e as Figuras Figura 60 e Figura 62 demonstram a adição de um sensor de temperatura.

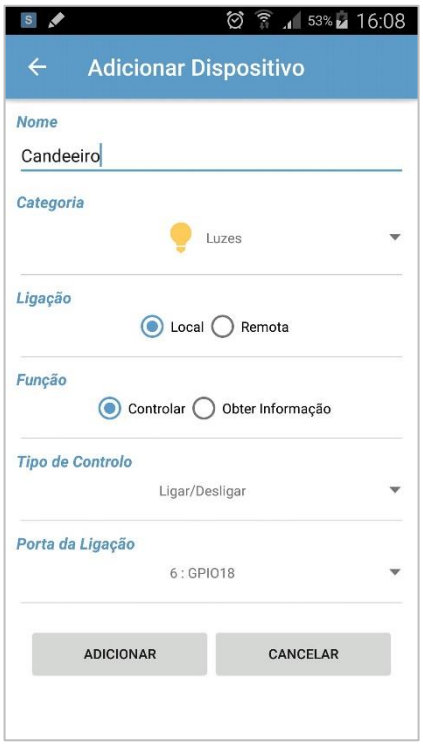


Figura 59 – Adicionar um Candeeiro como Dispositivo Local

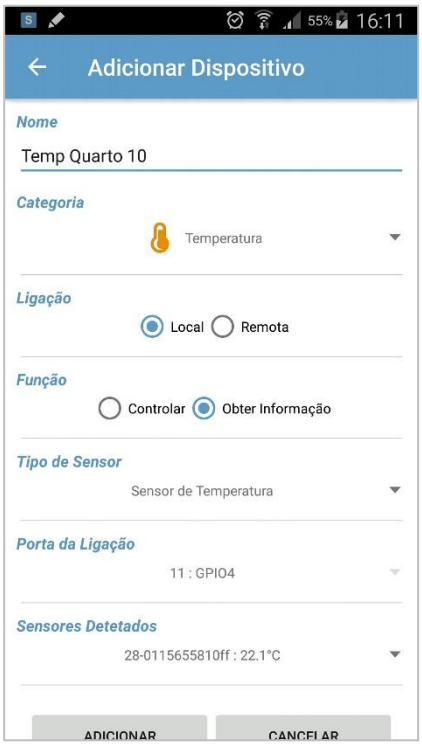


Figura 60 – Adicionar um Sensor de Temperatura como Dispositivo Local

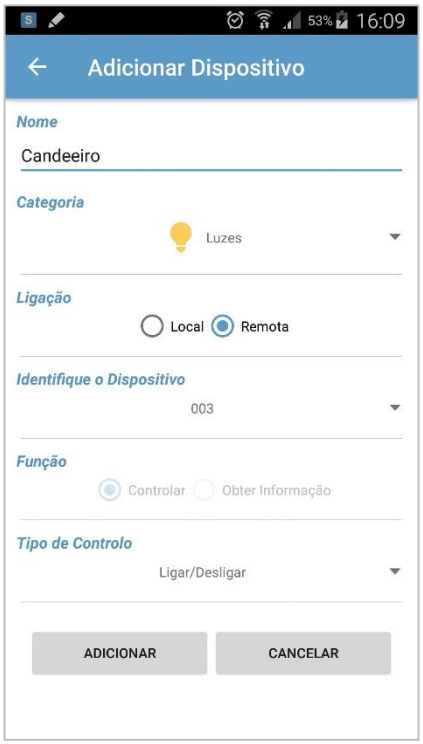


Figura 61 – Adicionar um Candeeiro como Dispositivo Remoto

Figura 62 – Adicionar um Sensor de Temperatura como Dispositivo Remoto

Ao finalizar o preenchimento do formulário e procedendo-se com a adição de um dispositivo, este é adicionado à base de dados do *HUB*, utilizando o serviço de comunicações da secção 5.4.1, e o utilizador é redirecionado para a página dos dispositivos onde o poderá ver juntamente com os outros.

Se houverem outros utilizadores a visualizar a página de dispositivos, durante a adição de um novo, esta é automaticamente atualizada assim que a adição terminar. A edição de um dispositivo segue a mesma lógica da adição.

Todo este processo pode ser visto no diagrama de sequência apresentado na Figura 63.

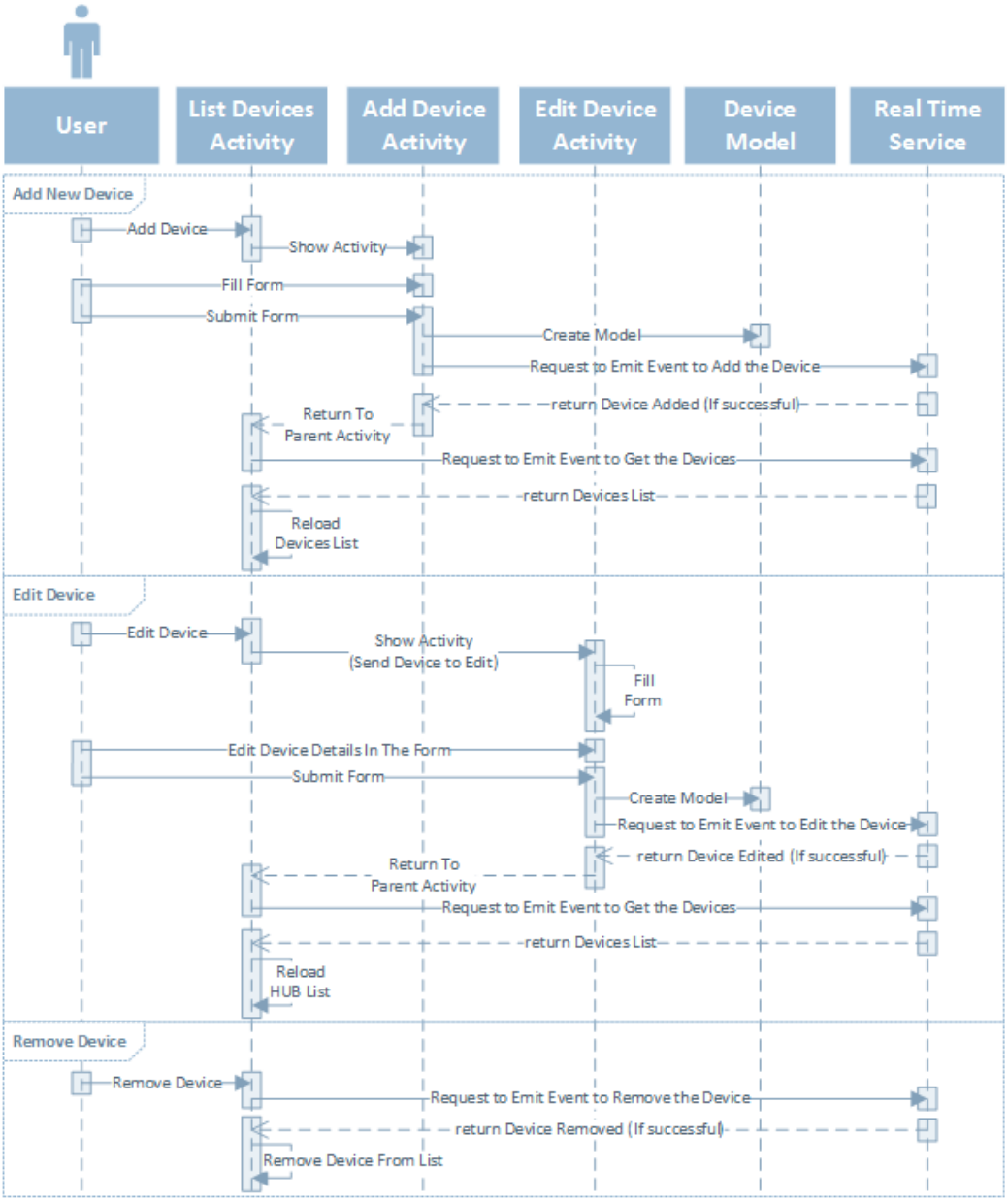


Figura 63 – Diagrama de Sequência das Ações Relacionadas com a Gestão de Dispositivos

5.4.5 Histórico de Eventos

Figura 64 – Histórico de Eventos

Num ambiente de automação e controlo de dispositivos físicos é muito importante ter informação sobre quando uma ação foi realizada, o que aconteceu e quem esteve envolvido.

Para responder a esta necessidade, a aplicação permite a visualização de um histórico de eventos (Figura 64), separados pelos dias de ocorrência.

Um evento é registado sempre que ocorrer uma alteração ao estado de um dispositivo. No entanto, se for um sensor de temperatura ou similar, pelo facto do seu valor poder flutuar bastante, apenas é registado um evento quando o seu valor ativar uma notificação.

Quando um utilizador acede à página do histórico pela primeira vez são carregados os eventos dos últimos 7 dias. Para visualizar eventos mais antigos é utilizada a técnica de *Lazy Loading*, para que a aplicação carregue mais eventos sempre que o utilizador atingir o fim da lista.

Tratando-se de um histórico, os eventos nunca vão ser alterados, portanto, para aumentar a eficiência da aplicação e reduzir os custos de comunicação, sempre que se carregarem eventos, estes são guardados na base de dados local. Considerando este aspeto, quando um utilizador aceder à página do histórico e já tiver eventos guardados serão apenas requisitados os eventos a partir da data mais recente guardada, até à data que este está a aceder. Este processo é retratado pela Figura 65.

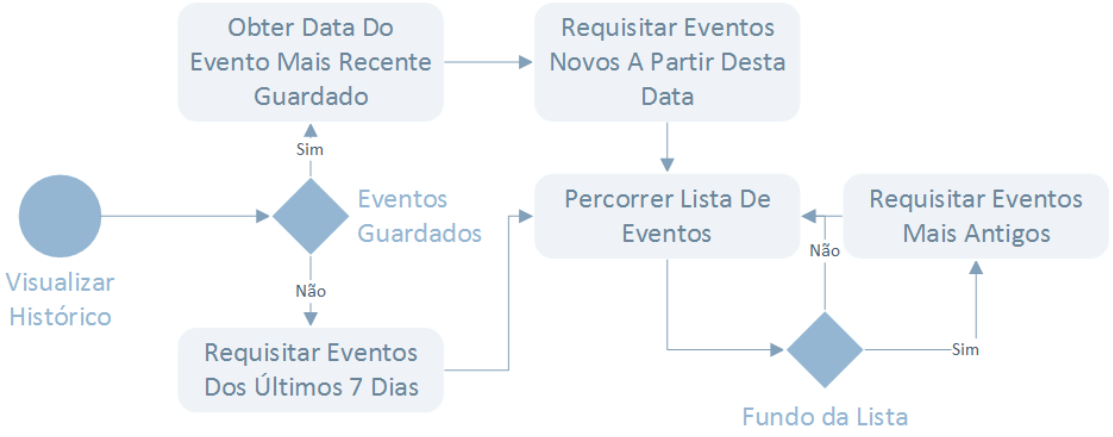


Figura 65 – Diagrama de Atividades da Visualização e Carregamento de Eventos do Histórico

5.4.6 Gestão de Notificações

Figura 66 – Lista de Notificações

Para tirar o máximo partido deste projeto, foi implementado um sistema de notificações, para que o utilizador pudesse ser alertado quando uma certa situação ocorresse.

Este sistema utiliza o construtor de notificações do Android para as apresentar (Figura 67) ao utilizador. Mais à frente detalhar-se-á esta funcionalidade.

Figura 67 – Exibição de Notificações

Através da página de notificações (Figura 66), todos os utilizadores podem ver os alertas criados e decidir, se querem ou não, estar subscritos a cada um deles.

Figura 68 – Adicionar uma Notificação (Exemplo 1)

Para criar uma notificação, o utilizador deve introduzir uma descrição e escolher o dispositivo sobre o qual quer criar o alerta.

Tendo o dispositivo sido escolhido, poderá então selecionar o estado do mesmo, os dias e as horas em que se quer verificar. Desta forma, cria-se uma regra que o *HUB* analisará, sempre que houver uma alteração do estado de um dispositivo, para poder decidir se deve notificar os utilizadores subscritos ou não.

Ao escolher o intervalo de verificação, se o utilizador escolher uma hora de início maior que a de fim, este será dividido em dois intervalos, como se pode ver na Figura 68.

Ao criar uma notificação, há a possibilidade de subscrever todos os utilizadores a ela, ou então apenas o utilizador que a está a criar.

As figuras Figura 69 e Figura 70 representam outros exemplos de como adicionar uma notificação utilizando outras definições e outros dispositivos, sendo também possível reparar na adaptação do formulário relativamente ao estado do dispositivo, tendo em conta o dispositivo selecionado.

Figura 69 - Adicionar uma Notificação (Exemplo 2)

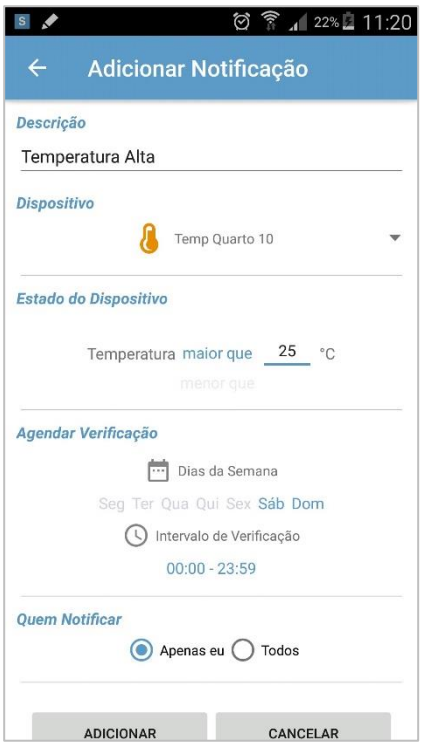


Figura 70 - Adicionar uma Notificação (Exemplo 3)

Tendo em conta que vários utilizadores podem estar subscritos a uma mesma notificação, decidiu-se não permitir a edição da mesma, visto que, poderia ter efeitos indesejáveis para os outros utilizadores.

Posto isto, os processos de adicionar uma notificação, de subscrever ou remover uma subscrição, e de remover uma notificação, podem ser analisados através do diagrama de sequência presente na Figura 71.

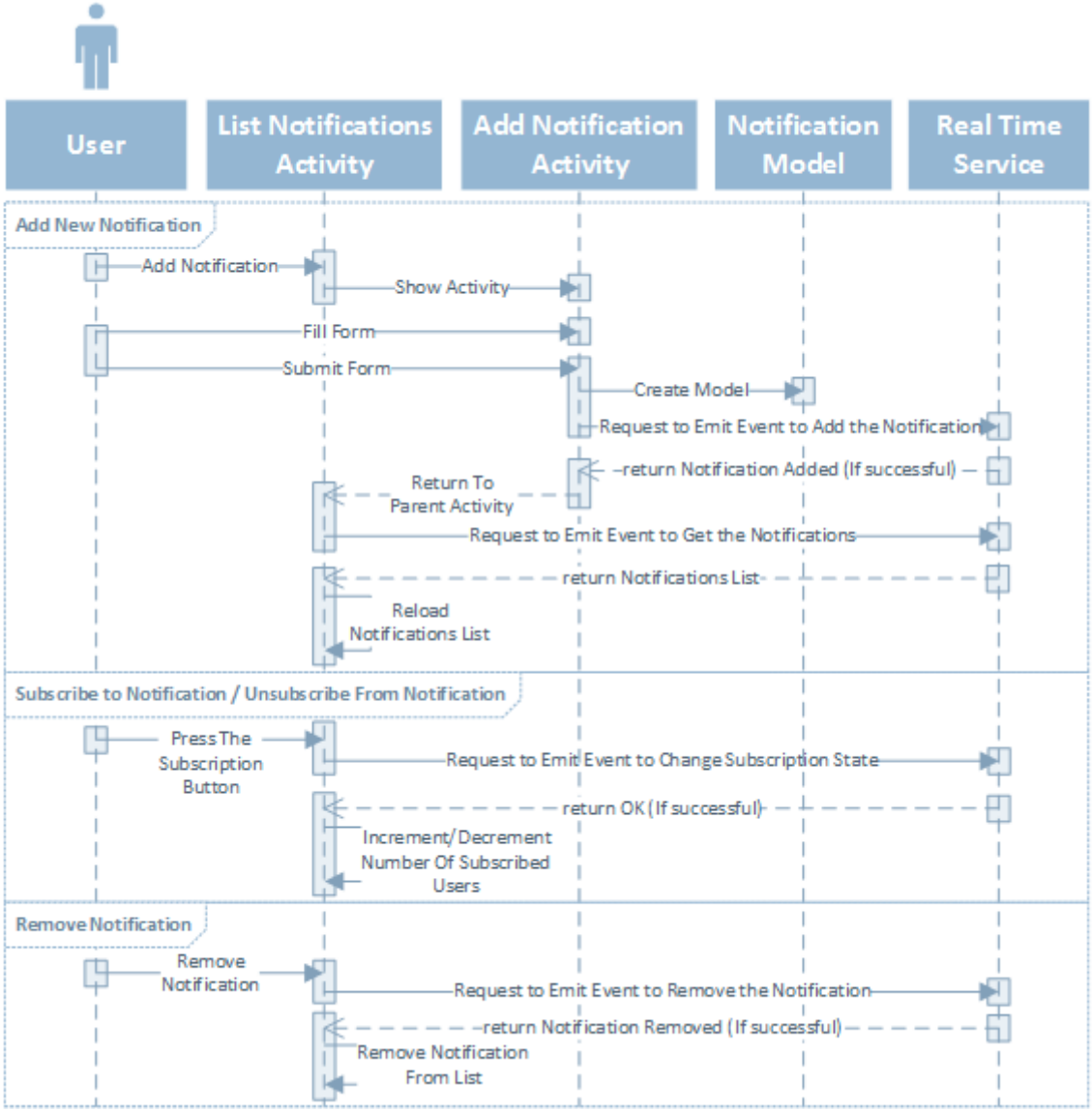


Figura 71 – Diagrama de Sequência das Ações Relacionadas com a Gestão de Notificações

Para exibir as notificações no sistema *Android* foi criada a classe *Android Notification Singleton*, a qual gere as notificações que tem de apresentar, agrupando-as pelo tipo de notificação (como se pode ver na Figura 67) e, posteriormente, apresentando-as utilizando a classe *Notification Compat*, pertencente ao *Android*.

O serviço de comunicações da secção 5.4.1 é crucial para esta funcionalidade, pois através dos objetos *Real Time Communication* permite que a aplicação receba os eventos emitidos pelo *HUB* para fazer a exibição de notificações, mesmo quando a aplicação não está a ser utilizada.

Para o utilizador descartar uma notificação recebida, este pode simplesmente deslizar a notificação para a direita, ou então carregando sobre ela para ser levado para a página do histórico de eventos da aplicação.

Este processo de exibição e manipulação de uma notificação pode ser analisado no diagrama de sequência presente na Figura 72.

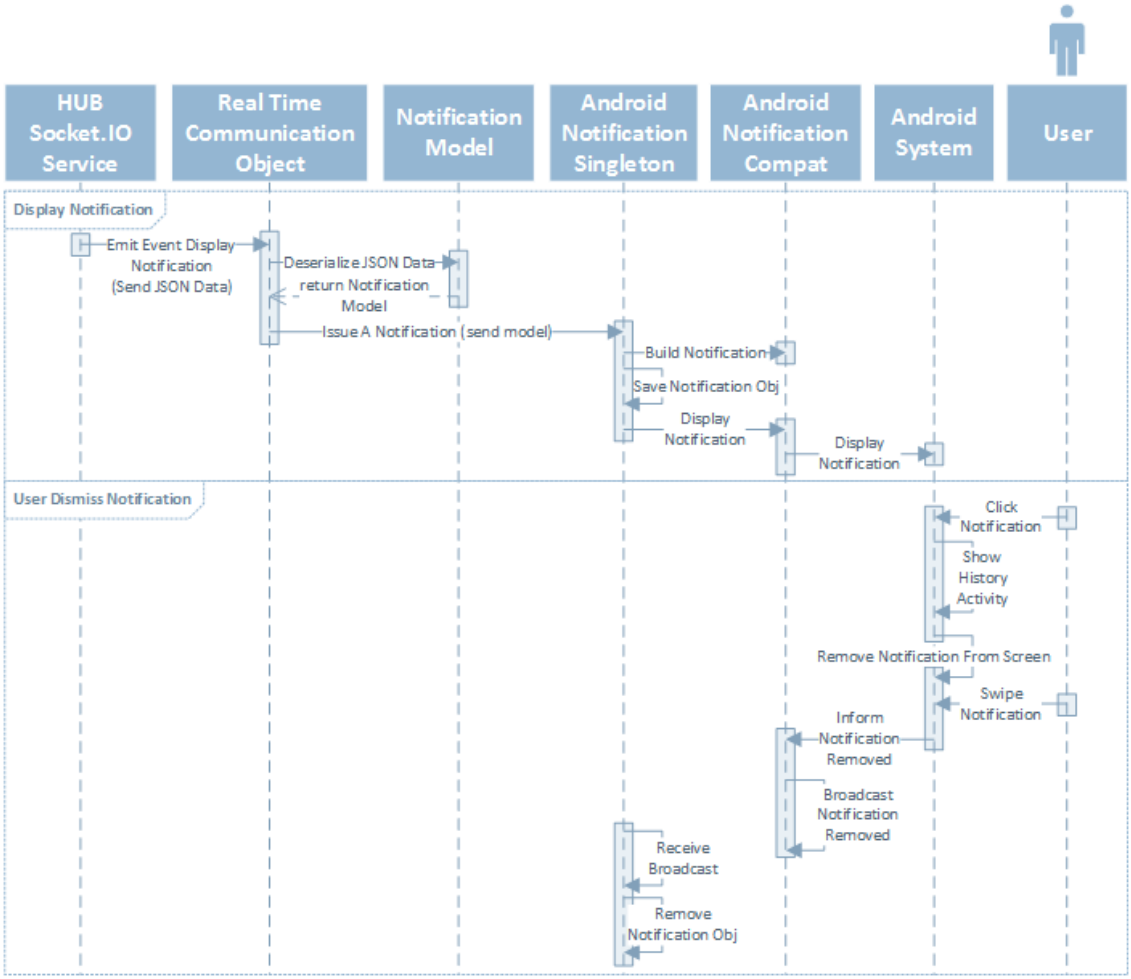


Figura 72 – Diagrama de Sequência do Processo de Exibição e Manipulação de uma Notificação

6 Avaliação da Solução Preconizada

6.1 Metodologias de Avaliação

Ao longo do desenvolvimento do projeto, quando se atingia uma versão estável do mesmo, esta era integrada e testada numa propriedade constituída por escritórios e uma fábrica. Nesta propriedade foi instalado um *HUB*, juntamente com dois dispositivos locais: um atuador para controlar o portão de entrada e um sensor de temperatura para monitorizar o quadro elétrico.

Durante a utilização do sistema por parte das pessoas que lá trabalham, foram recolhidas opiniões sobre o uso da aplicação móvel, e sobre a estabilidade do *HUB*.

A integração do projeto num contexto mais realista, permitiu obter informações vitais, que levaram ao melhoramento do funcionamento do *HUB* e da experiência de utilização da aplicação *Android*.

6.2 Grandezas Utilizadas na Avaliação

As grandezas utilizadas para avaliar a solução final são de seguida apresentadas, juntamente com algumas conclusões obtidas a partir de testes realizados, que têm como objetivo garantir a validade de cada uma.

- **Tempo de Resposta:** Recolha dos tempos necessários para a comunicação entre a aplicação e o *HUB* através dos diferentes casos de uso apresentados e, análise da média destes para ver se se encontram dentro de limites aceitáveis.

  Tendo em conta que o tempo de execução de uma ação não pode ser longo ao ponto do utilizador perder o foco sobre a ação que pretende realizar, foi definido na aplicação, um tempo máximo de 10 segundos para que este possa receber sempre feedback em tempo útil, mesmo que o seu resultado seja negativo.

- **Tempo de Recuperação:** Recolha dos tempos necessários que levam o *HUB* a voltar a um estado normal após uma falha (e.g. Internet, Energia, Falha Interna) e, utilização destes para implementação de melhorias nos processos de recuperação.

  Para testar esta grandeza, foram simuladas as seguintes situações:

  - Falha de Internet, desligando e ligando o *router* que fornece a ligação ao *HUB*;
  - Falha de Energia, desligando e ligando o transformador que fornece energia ao *HUB*;
  - Falha Interna, inserindo um erro no código do servidor *Node.js* para que fosse possível assistir à sua reinicialização.

  Após estes testes, verificou-se que a inicialização do *HUB* demora menos de um minuto a ser concluída e, confirmou-se a eficácia da ferramenta *Monit*, mencionada na secção 5.1.3.2, a qual rapidamente reinicializou o serviço parado. Desta forma garante-se que o tempo de recuperação do *HUB* será reduzido.

- **Distância de Comunicação entre o *HUB* e os Dispositivos Remotos:** Obtenção das distâncias máximas passíveis de serem realizadas, tanto no exterior como no interior, tendo sempre em conta a situação de teste e os obstáculos presentes nesta.

  Os dispositivos remotos não chegaram a ser instalados na propriedade que serviu de teste para este projeto, devido ao facto de não se ter conseguido obter uma versão do projeto, estável o suficiente, para ser utilizada num ambiente real.

  No entanto, esta propriedade ao possuir uma fábrica com paredes grossas e escritórios com paredes semelhantes às de uma habitação, tornou-se o candidato ideal para a realização dos testes de comunicação entre o *HUB* e os dispositivos remotos durante o seu desenvolvimento.

  Posto isto, colocou-se o *HUB* dentro de um escritório que fica a 50 metros da fábrica e utilizou-se um dispositivo remoto para testar a comunicação, colocando-o nas seguintes localizações: em frente à fábrica, 20 metros dentro da fábrica com muitas obstruções (Paredes, Maquinaria, etc.) pelo meio, dentro de um armazém parcialmente subterrâneo, e por fim, no exterior, a 70 metros do escritório.

  Com estes testes conseguiu-se um desempenho bastante aceitável, sendo que foi possível estabelecer comunicação em todas as localizações de teste, apenas com algumas falhas dentro da fábrica quando as obstruções se tornavam excessivas, dentro do armazém quando se testava em partes mais subterrâneas afastadas da entrada, e, durante o transporte do dispositivo de uma localização para a outra.

- **Satisfação do Cliente:** Realização de um inquérito aos utilizadores de teste para obtenção de *feedback* acerca dos vários componentes que compõem este projeto.

  Tendo em conta que foi feito um acompanhamento de perto, das diferentes versões que foram implementadas no ambiente de produção, foi possível obter de uma forma ativa as opiniões provenientes das pessoas que interagiram com o projeto.

  Através deste acompanhamento foi possível melhorar o projeto e perceber a sua importância e utilidade, para uma gestão mais dinâmica e remota da propriedade de teste.

6.3 Hipóteses a Testar

Para suportar a robustez do produto foram testadas as seguintes hipóteses.

Relativamente ao *HUB*:

- **Resiliência e Durabilidade:** Capacidade de manter a integridade da informação que detêm, após uma elevada quantidade de más práticas por parte do utilizador.

  Para testar esta hipótese foi necessário recorrer a uma má prática muito comum, que consiste em desligar um aparelho retirando o seu transformador da ficha, em vez de o desligar utilizando primeiro o botão adequado.

  Assim que este teste foi realizado pela primeira vez assistiu-se de imediato à corrupção da base de dados, de maneira que, quando o *HUB* era reiniciado, o servidor em *Node.js* não conseguia aceder aos dados.

  Para corrigir este problema, era necessário fazer uma limpeza à base de dados e criá-la novamente. Tendo em conta que esta não era de todo uma solução viável, foi investigada mais a fundo, a base de dados *MongoDB*, tendo-se encontrado a funcionalidade de *Journaling*. Esta permite manter cópias de segurança regulares, e garante a reparação e o restabelecimento dos dados, caso a integridade da base de dados se encontre comprometida. Após a ativação desta funcionalidade, nunca mais se encontrou este problema.

  Ao realizar repetidamente o teste aqui mencionado chegou-se a um ponto em que o sistema operativo que executa a partir de um cartão de memória, ficava infinitamente num ciclo de tentativa de inicialização, onde se podiam visualizar erros relativos a dados corrompidos e ao facto de o sistema não ter detetado um encerramento correto, e portanto, se encontrar num estado irreparável.

Estando neste estado, a única solução para voltar a utilizar o sistema era formatar o cartão de memória, reinstalando e configurando o sistema operativo, assim como todas as ferramentas necessárias para o projeto.

Tendo em conta que o cartão de memória era a variável mais provável de causar este problema, resolveu-se fazer a troca para um com melhores velocidades (analisadas na secção 5.1.3.1) e melhor reputação, o qual tem demonstrado ser mais robusto, tendo em conta que se tem mantido estável tanto em desenvolvimento como em produção, passando este teste sem problemas.

Não encontrando mais problemas, através da realização deste teste, pode-se concluir que esta hipótese se encontra validada.

Relativamente à aplicação *Android*:

- **Perda de Ligação:** Pretende-se assegurar que a aplicação executa um comportamento adequado quando existe uma falha de ligação à Internet, por parte do dispositivo móvel que a detém, ou então, quando não é possível estabelecer uma ligação com o(s) *HUB(s)* que esta está a monitorizar.

  Esta hipótese é garantida pela secção 5.4.1, quando se aborda o controlo e *feedback* sobre o estado de uma ligação.

Relativamente aos Dispositivos Remotos:

- **Limites de Comunicação:** Pretende-se verificar a percentagem de perda de mensagens, na comunicação entre o *HUB* e um dispositivo remoto, num ambiente livre de obstruções.

  Para testar esta hipótese foram realizados dois conjuntos de testes, em alturas diferentes de um mesmo dia, cada um contendo 500 interações, com um dispositivo remoto que possuía um atuador para controlar um candeeiro.

  Estes testes foram realizados utilizando a aplicação *Android* e monitorizando as mensagens apresentadas pelo servidor *Node.js*.

  No final das primeiras 500 interações obteve-se um resultado de 5.4% de falhas e concluiu-se que cada mensagem enviada pelo *HUB* para o dispositivo remoto demorou, em média, 2 segundos, a ser enviada e confirmada. Quando uma interação com um dispositivo remoto falha, o utilizador é devidamente notificado através da apresentação de uma mensagem de alerta na aplicação *Android*. As mensagens falhadas podem ser vistas na Tabela 6.

Tabela 6 – Primeiro Conjunto de Testes da Comunicação dos Dispositivos Remotos

Número do Teste	Ação Falhada
40	Failed to Send Off message
44	Failed to Send On message
77	Failed to Send On message
81	Failed to Send Off message
86	Failed to Send Off message
105	Failed to Send On message
116	Failed to Send On message
130	Failed to Send Off message
201	Failed to Send Off message
206	Failed to Send Off message
211	Failed to Send Off message
270	Failed to Send Off message
271	Failed to Send Off message
306	Failed to Send Off message
317	Failed to Send Off message
328	Failed to Send On message
336	Failed to Send Off message
347	Failed to Send Off message
348	Failed to Send Off message
362	Failed to Send On message
408	Failed to Send Off message
412	Failed to Send On message
432	Failed to Send Off message
439	Failed to Send Off message
445	Failed to Send Off message
485	Failed to Send Off message
486	Failed to Send Off message

No final do segundo conjunto de interações verificou-se o mesmo tempo médio de envio e confirmação das mensagens e obteve-se um resultado de 5.8% de falhas. As mensagens falhadas podem ser vistas na Tabela 7.

Tabela 7 - Segundo Conjunto de Testes da Comunicação dos Dispositivos Remotos

Número do Teste	Ação Falhada
521	Failed to Send On message
533	Failed to Send Off message
538	Failed to Send Off message
568	Failed to Send Off message
590	Failed to Send On message
613	Failed to Send On message
626	Failed to Send Off message
689	Failed to Send Off message

Número do Teste	Ação Falhada
704	Failed to Send Off message
710	Failed to Send On message
729	Failed to Send On message
747	Failed to Send Off message
813	Failed to Send On message
815	Failed to Send Off message
818	Failed to Send Off message
826	Failed to Send On message
833	Failed to Send On message
835	Failed to Send Off message
839	Failed to Send On message
881	Failed to Send Off message
888	Failed to Send Off message
913	Failed to Send Off message
922	Failed to Send Off message
939	Failed to Send On message
962	Failed to Send On message
965	Failed to Send Off message
976	Failed to Send Off message
977	Failed to Send Off message
983	Failed to Send On message

Apesar da média de falhas ser de 5.6%, e desta se poder considerar bastante aceitável, em futuros desenvolvimentos deste projeto será investida uma porção considerável de tempo para diminuir esta percentagem, de forma a garantir uma maior satisfação de um possível cliente.

- **Durabilidade da Bateria:** Tendo em conta o facto de que seria impensável esperar um ou dois anos para ver realmente quanto tempo a bateria durou, planeou-se fazer algumas medidas ao longo de certos períodos e, a partir dessas informações, fazer uma estimativa aproximada do tempo que a bateria de um Dispositivo Remoto iria durar.

  Visto que não chegou a ser possível implementar técnicas de eficiência energética aos dispositivos remotos através de uma gestão eficiente do tempo de funcionamento dos respetivos rádios, foi descartado o teste desta hipótese no contexto desta tese. No entanto, esta será replaneada numa situação de futuros desenvolvimentos deste projeto.

7 Conclusões

Neste capítulo serão descritas as conclusões finais da realização deste projeto, assim como algumas perspetivas de trabalho futuro.

Começa-se por fazer um resumo dos objetivos realizados no decorrer do projeto e das principais vantagens da sua utilização.

De seguida, são mencionadas algumas limitações dos vários componentes do projeto, finalizando-se com perspetivas e ideias que poderão vir a ser implementadas no mesmo.

7.1 Contribuições do Projeto

Na seguinte lista, apresentam-se os objetivos principais, atingidos:

- Construção de um gestor central (o *HUB*), capaz de controlar e monitorizar dispositivos locais e remotos, com a possibilidade de ser acedido através da internet ou de uma rede local.
- Suporte para a utilização de dispositivos locais e remotos com as funcionalidades: atuador, sensor de temperatura, botão e sensor de contacto.
- Construção de uma aplicação *Android*, que permite:
    - Fazer a gestão de múltiplos *HUBs*;
    - Fazer a gestão, o controlo, e a monitorização, de dispositivos locais e remotos;
    - A criação de notificações para alertas importantes, tendo por base o estado de um dispositivo;
    - A visualização do histórico de eventos.

Tendo em conta que, certas funcionalidades necessitaram de mais tempo do que o planeado, não foi possível implementar a gestão de utilizadores e permissões, nem a gestão de cenários, que estavam definidos nos objetivos iniciais.

Concluindo, conseguiu-se alcançar o objetivo principal da criação de um sistema de automação, que permite abordar aspetos de acessibilidade e segurança, e que poderia ser comercializado a um preço competitivo. Este projeto tornou-se também uma mais-valia para a propriedade onde foi testado, permitindo ao dono, controlar o acesso às instalações por entidades externas, estando este, em qualquer parte do mundo.

7.2 Limitações

A passagem da utilização de um serviço baseado em *REST* para um serviço baseado em *Sockets* trouxe a grande vantagem de se poder transmitir ao utilizador uma experiência de eventos em tempo real.

No entanto, trouxe também uma grande desvantagem notada ao longo do desenvolvimento, que consiste na instabilidade das ligações entre o *HUB* e a aplicação *Android*. Mais concretamente, o facto de estas ligações serem desligadas sem motivo aparente, e, o seu restabelecimento não ser imediato. Toda a investigação que foi feita aponta para que o problema seja causado pela forma como estas são geridas pelo sistema *Android*, porém, não foi encontrada uma solução capaz de o mitigar.

Relativamente aos dispositivos remotos, tal como foi explicado na secção 5.3.1, foi necessário criar um adaptador para que o *Raspberry Pi* pudesse interagir com o rádio *RFM69*. Apesar do adaptador ter solucionado o problema abordado nessa secção, este serve apenas como prova de conceito, visto que, cria um elo fraco na comunicação com os dispositivos remotos, ao manter parte do processo de envio e receção de mensagens e ao criar a necessidade de utilização de dois protocolos físicos para concretizar este processo.

7.3 Trabalho Futuro

Em relação ao trabalho futuro, a tarefa mais prioritária será a de eliminar as limitações. Para isto, pretende-se fazer uma pesquisa exaustiva sobre o problema da instabilidade das comunicações entre o *HUB* e a aplicação *Android*, de maneira a encontrar uma solução ou, em último caso, uma alternativa semelhante à tecnologia *Socket.IO*. Para além disto, pretende-se eliminar o adaptador, que permite a interação do *Raspberry Pi* com o rádio *RFM69*, mesmo que seja necessária a correção da biblioteca por parte do autor deste relatório, permitindo assim uma interação direta entre os dois componentes e uma maior fiabilidade nas comunicações com os dispositivos remotos.

Relativamente a novo conteúdo, pretende-se implementar as funcionalidades que não foram possíveis de abordar durante o período desta tese, juntamente com as seguintes ideias:

- Gestão de utilizadores e permissões (Planeada mas não implementada);
- Gestão de cenários, onde será possível planear o comportamento de dispositivos, tendo em conta o comportamento de outros (e.g Quando um dispositivo, que é um botão, for carregado, a campainha irá tocar) (Planeada mas não implementada);
- Utilização de *Geofencing* na criação de cenários, para que seja possível planear uma tarefa, tendo em conta o posicionamento global de um utilizador relativamente ao *HUB*;
- Agendamento de tarefas, onde será possível programar o que um dispositivo deve fazer e quando (e.g. Regar o jardim todos os sábados das 09:00 até às 09:30);
- Suporte de câmaras e novos tipos de sensores;
- Controlo de voz dos dispositivos utilizando a aplicação *Android*;
- Adição de um módulo *GSM* ao *HUB* para funcionar como alternativa à internet.

Para finalizar, existe também a intenção de alargar os horizontes deste projeto, para procurar outros contextos onde este possa ser aplicado (e.g. Na gestão dos lugares de parques de estacionamento). Sendo assim, a continuação do desenvolvimento deste projeto torna-se essencial para a criação de um novo produto capaz de se adaptar a diferentes contextos, com maior alcance de comunicação e a um preço acessível a todos.

Referências

Akscyn, R.M., McCracken, D.L. & Yoder, E.A., 1988. KMS: a distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7), pp.820–835. Available at: http://portal.acm.org/citation.cfm?doid=48511.48513.

Anon, Node.js – reasons to use, pros and cons, best practices! Available at: http://voidcanvas.com/describing-node-js/ [Accessed February 20, 2016].

Carnevale, P.J. & Pruitt, D.G., 1992. NEGOTIATION AND MEDIATION.

Dixit, A. & Nalebuff, B., 2008. Prisoners' Dilemma. The Concise Encyclopedia of Economics. Available at: http://www.econlib.org/library/Enc/PrisonersDilemma.html [Accessed February 12, 2016].

Eastgate, 1999. Akscyn's law. Available at: http://www.eastgate.com/HypertextNow/archives/Akscyn.html [Accessed February 14, 2016].

Gangji, A., 2013. MySQL vs. MongoDB: Looking At Relational and Non-Relational Databases. Available at: http://www.neonrain.com/blog/mysql-vs-mongodb-relational-and-non-relational-databases [Accessed February 19, 2016].

Grusin, M., 2014. Serial Peripheral Interface (SPI) - Learn.SFE. *Sparkfun*, pp.1–13. Available at: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi [Accessed January 1, 2016].

Link Labs, 2016. Top 7 M2M & IoT Wireless Technologies Explained. , pp.1–11.

Myers, P., 2007. Interfacing using Serial Protocols Using SPI and I2C. , pp.1–9. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.2531&rep=rep1&type=pdf.

Nicola, S., 2016a. Análise de valor de negócio, Aula 1.

Nicola, S., 2016b. Análise de valor de negócio, Aula 2.

Nicola, S., 2016c. Análise de valor de negócio, Aula 3.

Nicola, S., 2016d. Análise de valor de negócio, Aula 4.

Oudjida, A.K. et al., 2009. FPGA implementation of I2C & SPI protocols: A comparative study. In *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*. IEEE, pp. 507–510. Available at: http://ieeexplore.ieee.org/document/5410881/.

Porto Editora, 2016. Equilíbrio de Nash. *2003-2016*. Available at: http://www.infopedia.pt/$equilibrio-de-nash [Accessed February 12, 2016].

Rathnayaka, A.J.D., Potdar, V.M. & Kuruppu, S.J., 2011. Evaluation of wireless home automation technologies. *IEEE International Conference on Digital Ecosystems and Technologies*, (July 2011), pp.76–81.

Sarto, J., 2016. ZigBee VS. 6LoWPAN for Sensor Networks. Available at: https://www.lsr.com/white-papers/zigbee-vs-6lowpan-for-sensor-networks [Accessed February 20, 2016].

Serra, J., 2015. Relational databases vs Non-relational databases. Available at: http://www.jamesserra.com/archive/2015/08/relational-databases-vs-non-relational-databases/ [Accessed February 19, 2016].

Silva, N., 2016. Requisitos dos Outcomos da Entrega P1.1.

Strategyzer, 2016. Strategyzer | Business Model Canvas. Available at: http://businessmodelgeneration.com/canvas/bmc [Accessed February 13, 2016].