

Universidade da Beira Interior

Departamento de Informática



Departamento de
Informática

IoT Messaging for Tidal Gauge Data

Elaborado por:

Vasco Moreira Senra

Orientador:

Professor Doutor Paul Crocker

3 de julho de 2023

Agradecimentos

Gostaria de expressar a minha profunda gratidão a todos aqueles que tornaram possível a realização deste projeto de licenciatura.

Em primeiro lugar, gostaria de agradecer ao meu orientador de projeto, Professor Doutor Paul Crocker e ao Doutor Fernando Geraldes pelas suas orientações, encorajamentos e apoios constantes ao longo de todo o processo. As suas ideias e sugestões foram fundamentais para o desenvolvimento do meu trabalho.

Além disso, gostaria de agradecer aos professores e colegas de curso que me inspiraram e apoiaram durante toda a minha jornada académica. As suas contribuições e críticas construtivas foram essenciais para o meu crescimento pessoal e profissional.

Por fim, gostaria de agradecer à minha família e amigos, que estiveram sempre presentes com o seu amor, incentivo e apoio em cada passo da minha jornada. O seu apoio incondicional ajudou-me a superar os desafios e a alcançar este objetivo tão importante na minha vida.

A todos vocês, a minha sincera gratidão e apreço.

Conteúdo

Conteúdo	iii
Lista de Figuras	v
1 Introdução	1
1.1 Enquadramento	1
1.2 Motivação	1
1.3 Objetivos	2
1.4 Organização do Documento	2
2 Estado da Arte	3
2.1 Introdução	3
2.2 Estado da Arte	3
2.2.1 Sistema Anterior	3
2.2.1.1 Problemas Encontrados	5
2.2.2 Dados	5
2.2.3 MQTT	7
2.2.3.1 QoS	8
2.2.3.2 Clean Session Flag	8
2.2.4 Router	9
2.3 Conclusões	9
3 Tecnologias e Ferramentas Utilizadas	11
3.1 Introdução	11
3.2 Tecnologias e Ferramentas Utilizadas	11
3.2.1 VirtualBox	11
3.2.2 Ubuntu	11
3.2.3 OpenVPN	12
3.2.4 LucidChart	12
3.2.5 Overleaf	12
3.2.6 SQLite3	12
3.2.7 ErdPlus	13
3.2.8 Eclipse	13

3.2.9	Marégrafos	13
3.2.10	<i>Router</i>	14
3.2.11	<i>MQTT</i>	14
3.3	Conclusões	14
4	Implementação	15
4.1	Introdução	15
4.2	Gestão de trabalho	15
4.3	Implementação	16
4.3.1	Vista geral da nova arquitetura	16
4.3.2	Base de Dados	17
4.3.3	Gestor para o SEGAL	19
4.3.4	Marégrafos	20
4.3.5	<i>Router</i>	20
4.3.6	Publisher no router	20
4.3.7	Segurança	22
4.3.8	Cliente para o SEGAL	22
4.3.9	Publisher SEGAL	23
4.3.10	Cliente Global	24
4.3.11	<i>Script</i> para testes	25
4.4	Conclusões	26
5	Testes e simulações	27
5.1	Introdução	27
5.1.1	Comportamento Normal	27
5.1.2	Falha no Marégrafo	29
5.1.3	Falha no Segal	30
5.1.4	Falha no Cliente	31
5.1.5	Teste do SSL	32
5.2	Conclusão	32
6	Conclusões e Trabalho Futuro	33
6.1	Conclusões Principais	33
6.2	Trabalho Futuro	33
Bibliografia		35

Lista de Figuras

2.1	Esta imagem representa de forma simplificada o sistema anterior.	4
2.2	Esta imagem é um excerto real de como os dados são armazenados.	6
2.3	Esta imagem é o diagrama do modelo do Protocolo MQTT [1].	7
2.4	Esta imagem mostra um <i>Router</i> RUT950.	9
4.1	Esta imagem representa de forma simplificada o sistema.	17
4.2	Esta imagem é uma representação da estrutura da Base de Dados.	18
5.1	Esta imagem mostra a simulação acima.	28
5.2	Esta imagem mostra a simulação acima.	29
5.3	Esta imagem mostra a simulação acima.	30
5.4	Esta imagem mostra a simulação acima.	31
5.5	Esta imagem mostra a simulação acima.	32

Listas de Excertos de Código

4.1	Script para criação da base de dados.	18
4.2	Dados usados para testes	19
4.3	Exemplo de código para adicionar um marégrafo na base de dados.	19
4.4	Este excerto de código verifica a conexão com o Broker e lê a linha.	21
4.5	Trecho de código responsável por escrever no ficheiro	23
4.6	Trecho de código que faz a preparação da nova mensagem.	24
4.7	Trecho de código que faz a separação da mensagem.	24
4.8	Este script é usado para simular a escrita dos marégrafos no ficheiro "Send.txt".	25

Acrónimos

SEGAL	<i>Space & Earth Geodetic Analysis Laboratory</i>
IPMA	Instituto Português do Mar e da Atmosfera
MQTT	<i>Message Queuing Telemetry Transport</i>
UBI	Universidade da Beira Interior
UC	Unidade Curricular
IoT	<i>Internet of Things</i>
QoS	<i>Quality of Service</i>
IDE	<i>Integrated Development Environment</i>

Glossário

Dat A extensão DAT representa um arquivo que contém dados. Ele adota este formato porque não está associado a nenhum programa em particular para realizar a sua abertura. 5

Broker *Broker* é um componente de *software* que atua como intermediário entre diferentes sistemas, aplicações ou serviços. O objetivo principal de um *broker* é facilitar a comunicação e a integração entre esses sistemas, permitindo a troca de informações de maneira eficiente e segura.. 2, 9, 14, 16, 17, 20–22, 24, 29–32

Hardwares Parte física de computadores e outros sistemas microeletrônicos. 3, 11, 17

Log Em computação, *log* de dados é uma expressão utilizada para descrever o processo de registo de eventos relevantes num sistema computacional. 3, 5

MQTT É um protocolo de transporte de mensagens de formato Cliente/Servidor, que possibilita a comunicação entre máquinas. 1, 2, 14

Plaintext São quaisquer dados legíveis num formato que pode ser visto ou utilizado sem a necessidade de uma chave de descriptografia ou dispositivo de descriptografia. 3, 5, 16

Publisher O publisher é responsável por criar e enviar as mensagens para o broker.. 7

Software É o conjunto de componentes lógicos de um computador ou sistema de processamento de dados; programa, rotina ou conjunto de instruções que controlam o funcionamento de um computador; suporte lógico.. 11, 12, 14, 32

String Na programação de computadores, uma cadeia de caracteres ou *string* é uma sequência de caracteres, geralmente utilizada para representar palavras, frases ou textos de um programa. 5

Capítulo

1

Introdução

1.1 Enquadramento

Este projeto foi realizado no contexto da Unidade Curricular (UC) Projeto da Universidade da Beira Interior (UBI).

1.2 Motivação

Este projeto está preocupado com a transmissão confiável e segura de dados sobre a amplitude das marés, oriundos das estações de monitorização do nível das águas. Estas estações possuem sensores que registam continuamente parâmetros sobre os níveis da água, tais como a altura, posição, velocidade, entre outros. Esses dados são importantes para muitas atividades, como navegação, habitat, modelação meteorológica e assim por diante.

Neste projeto estamos interessados no armazenamento destes dados e posterior transmissão segura aos utilizadores finais que, no nosso caso, é principalmente ao Instituto Português do Mar e da Atmosfera (IPMA). No entanto, o sistema a desenvolver tem que ser flexível para contemplar outros institutos e empresas. O sistema atual é baseado num sistema não confiável, já que o envio pode falhar devido a vários motivos como, por exemplo, falha de energia ou problemas de conexão. Neste projeto será desenhada e aplicada uma nova arquitetura baseada em *MQTT* que será abordada futuramente neste relatório.

1.3 Objetivos

Como mencionado anteriormente, o objetivo deste trabalho é implementar uma nova arquitetura baseada em *MQTT*. Para atingir este objetivo principal foram definidos vários objetivos parciais:

1. Desenho e Criação dum sistema baseado em *MQTT* para a transmissão de dados dum conjunto de Maregráfios;
2. Criação de um cliente para o marégrafo/hóspede/*router* do mesmo;
3. Criação da base de dados;
4. Criação do gestor para a base de dados;
5. Criação de um *Broker* para o SEGAL;
6. Criação de um cliente final para o IPMA;

1.4 Organização do Documento

De modo a refletir o trabalho que foi feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento;
2. O segundo capítulo – **Estado da Arte** – descreve a arquitetura anterior e os problemas encontrados nela;
3. O terceiro capítulo – **Tecnologias e Ferramentas Utilizadas** – descreve as tecnologias utilizadas durante o desenvolvimento do projeto;
4. O quarto capítulo – **Implementação** – apresenta como as tecnologias utilizadas foram implementadas de forma a obter o resultado final;
5. O quinto capítulo – **Conclusões e Trabalho Futuro** – descreve as conclusões obtidas durante e após a realização do trabalho e o que foi alcançado ou não, segundo os objetivos da proposta do projeto e, ainda, como poderá ser melhorado no futuro.

Capítulo

2

Estado da Arte

2.1 Introdução

Neste capítulo será explicada a solução e/ou arquitetura existente anteriormente, bem como defeitos e/ou problemas encontradas nela.

2.2 Estado da Arte

2.2.1 Sistema Anterior

De forma geral, o sistema anterior está constituído por quatro *Hardwares* sendo eles um marégrafo, um *router*, um servidor intermédio (localizado no centro *Space & Earth Geodetic Analysis Laboratory* (SEGAL)) e o servidor do cliente final, que é neste caso o do IPMA.

De forma simplificada, podemos dividir o sistema anterior em duas etapas distintas que chamarei de etapa "Marégrafo-SEGAL" e "SEGAL-IPMA".

Na primeira etapa ou etapa "Marégrafo-SEGAL", o marégrafo recolhe dados a cada cinco segundos que são armazenados no *router*, dentro de um ficheiro em *Plaintext* que é explicado melhor na secção 2.2.2.

O *router* envia esse ficheiro para o servidor intermédio a cada vinte segundos, apagando esses dados em seguida para receber os próximos.

Na segunda etapa ou etapa "SEGAL-IPMA", o SEGAL recebe os dados e adiciona-os a um ficheiro que chamaremos de *Log* com um nome estruturado da seguinte forma "dd-mm-aaaa" e elimina os dados recebidos.

Posteriormente, envia para o servidor do IPMA o próprio ficheiro onde, por sua vez, o IPMA os tratará e analisará como bem entender.

Para melhor contextualização, a imagem 2.1 representa, de forma simplificada, o que foi descrito em cima.

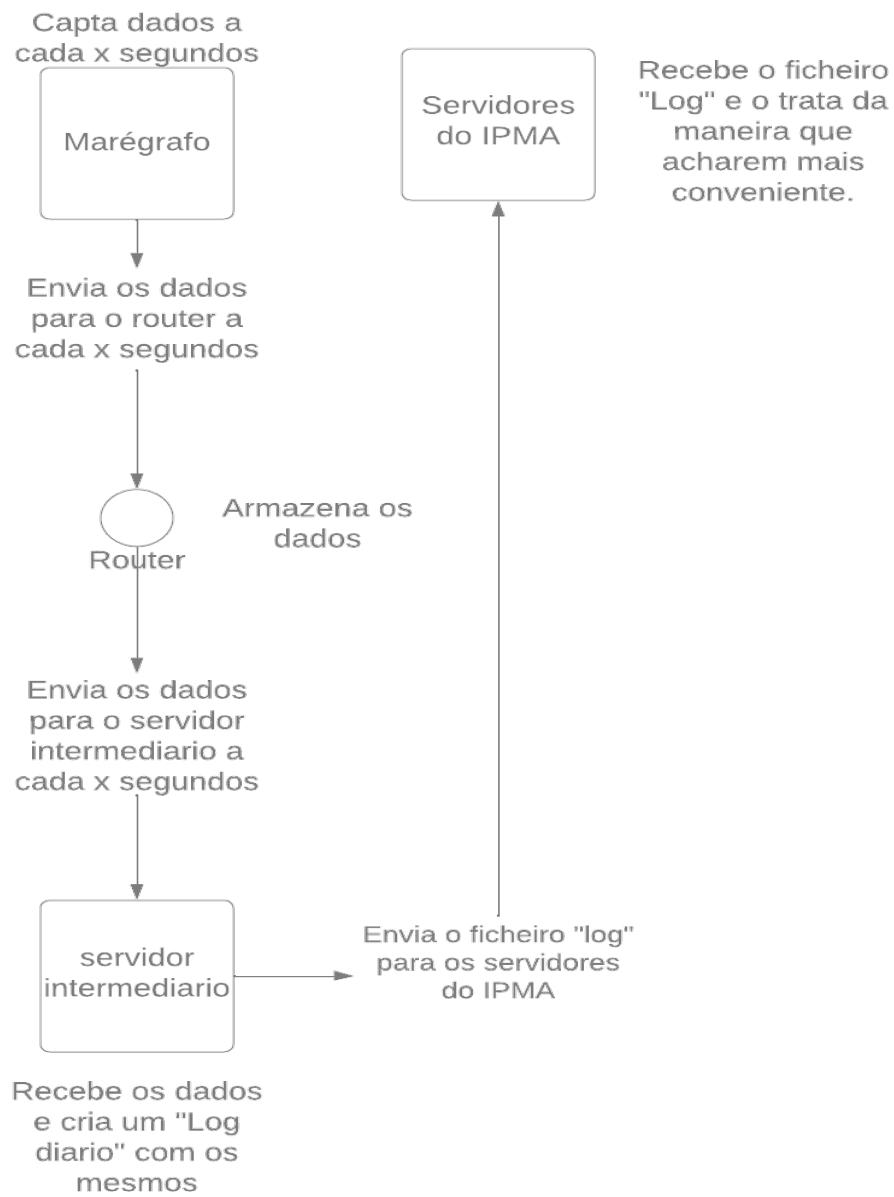


Figura 2.1: Esta imagem representa de forma simplificada o sistema anterior.

2.2.1.1 Problemas Encontrados

Este sistema não era perfeito e possuía alguns problemas que serão descritos em seguida.

Um dos principais problemas e o maior motivo deste projeto era a instabilidade e inconsistência no momento do envio de dados, apesar de não existir perda dos mesmos. Isto é devido a ser sempre enviado um ficheiro inteiro e aos dados não serem apagados, caso não se conseguisse enviar para o servidor do SEGAL, . No entanto, os dados poderiam vir com atrasos de alguns segundos, minutos ou até mesmo horas devido a falhas de comunicação entre os dispositivos sem existir consciência que os mesmos falharam no envio ou que não se encontravam disponíveis.

Outro dos problemas é a redundância dos dados, já que o servidor intermediário envia sempre o ficheiro *Log*. Isto causa um excesso e sobreposição de dados enviados, visto que enviamos sempre o mesmo ficheiro só que com mais entradas de dados para o IPMA.

Adicionalmente, encontrei alguns problemas de segurança em relação aos dados que, apesar de não serem sensíveis, estão em *Plaintext* em que apenas se mudou o tipo de ficheiro para *.Dat*.

Para além disso, relacionado com a segurança, não existe controlo sobre estes dados o que permite a sua adulteração em qualquer momento, sem o conhecimento de ninguém.

Por último, todo o sistema carece de documentação. A informação de como o sistema funciona e/ou deve funcionar tem de ser transmitida através de desenvolvedor para desenvolvedor o que origina uma dependência do desenvolvedor anterior ou uma perda de tempo desnecessária para conhecimento e entendimento do sistema completo.

2.2.2 Dados

Os dados que o marégrafo recolhe são armazenados num *Plaintext*, como foi descrito anteriormente, onde estes são organizados na forma de uma *String* com os dados separados por "," e com a seguinte estrutura.

O primeiro dado é a data da recolha dos dados pelo marégrafo que é organizada da seguinte forma "YYYY-MM-DD HH:MM:SS".

O segundo dado é o nome dado ao marégrafo pelo SEGAL que, por diversos motivos, pode ser diferente do nome dado ao marégrafo pelo IPMA.

O terceiro dado é referente a códigos de erro sendo, por defeito, o valor "0000", representando que não houve erros.

O quarto dado representa o número do registo a que a *String* se refere. Este possui um erro que o torna não sequencial, podendo estar, às vezes, no

registro número quarenta e sete mil e, por algum motivo, voltar para o trinta mil. Apesar de ser conhecido este erro, o mesmo não pode ser corrigido, pois o software responsável por isso não é desenvolvido pelo SEGAL.

O último dado refere-se à altura da maré que vem em metros e no formato "X.XXX". Quando o marégrafo, por algum motivo, não consegue captar a altura da maré esta é apresentada como "///".

Para referência visual, a imagem 2.2 contém uma representação fidedigna da forma como os dados são armazenados.

```
"2023-02-28 16:56:40",ECPM,000,46685,1.166
"2023-02-28 16:56:45",ECPM,000,46686,1.275
"2023-02-28 16:56:50",ECPM,000,46687,0.989
"2023-02-28 16:56:55",ECPM,000,46688,1.35
"2023-02-28 16:57:00",ECPM,000,46689,1.225
"2023-02-28 16:57:05",ECPM,000,46690,1.121
"2023-02-28 16:57:10",ECPM,000,46691,///
"2023-02-28 16:57:15",ECPM,000,46692,1.295
"2023-02-28 16:57:20",ECPM,000,46693,1.094
"2023-02-28 16:57:25",ECPM,000,46694,///
"2023-02-28 16:57:30",ECPM,000,46695,1.393
"2023-02-28 16:57:35",ECPM,000,46696,1.242
"2023-02-28 16:57:40",ECPM,000,46697,///
"2023-02-28 16:57:45",ECPM,000,46698,1.365
"2023-02-28 16:57:50",ECPM,000,46699,1.254
"2023-02-28 16:57:55",ECPM,000,46700,///
"2023-02-28 16:58:00",ECPM,000,46701,1.321
"2023-02-28 16:58:05",ECPM,000,46702,1.404
"2023-02-28 16:58:10",ECPM,000,46703,1.322
"2023-02-28 16:58:15",ECPM,000,46704,///
"2023-02-28 16:58:20",ECPM,000,46705,1.32
"2023-02-28 16:58:25",ECPM,000,46706,1.175
"2023-02-28 16:58:30",ECPM,000,46707,///
"2023-02-28 16:58:35",ECPM,000,46708,1.45
"2023-02-28 16:58:40",ECPM,000,46709,1.21
"2023-02-28 16:58:45",ECPM,000,46710,///
"2023-02-28 16:58:50",ECPM,000,46711,1.287
"2023-02-28 16:58:55",ECPM,000,46712,1.336
"2023-02-28 16:59:00",ECPM,000,46713,///
"2023-02-28 16:59:05",ECPM,000,46714,1.174
"2023-02-28 16:59:10",ECPM,000,46715,1.489
"2023-02-28 16:59:15",ECPM,000,46716,1.212
"2023-02-28 16:59:20",ECPM,000,46717,///
"2023-02-28 16:59:25",ECPM,000,46718,1.089
"2023-02-28 16:59:30",ECPM,000,46719,1.452
"2023-02-28 16:59:35",ECPM,000,46720,1.248
"2023-02-28 16:59:40",ECPM,000,46721,1.118
root@RUT955-1109015169:/mnt/mmcblk0p1/1/data#
```

Figura 2.2: Esta imagem é um excerto real de como os dados são armazenados.

2.2.3 MQTT

Message Queuing Telemetry Transport (MQTT) é um protocolo de mensagens de Publicação-Subscrição leve que é bastante popular em aplicações de gênero *Internet of Things* (IoT). Este foi projetado para ser simples, eficiente e confiável, tornando-o adequado para dispositivos IoT que têm capacidade de processamento e largura de banda limitadas, que é exatamente o que ocorre neste projeto.

No MQTT, existem dois tipos de entidades: Publicadores (*Publisher*) e Clientes. Os Publicadores usam brokers para enviar dados como mensagens. As mensagens são organizadas em tópicos e podem ser enviadas de forma assíncrona, o que significa que o remetente não precisa esperar uma resposta do destinatário. Os clientes são aplicações que se inscrevem em tópicos específicos para receber as mensagens.

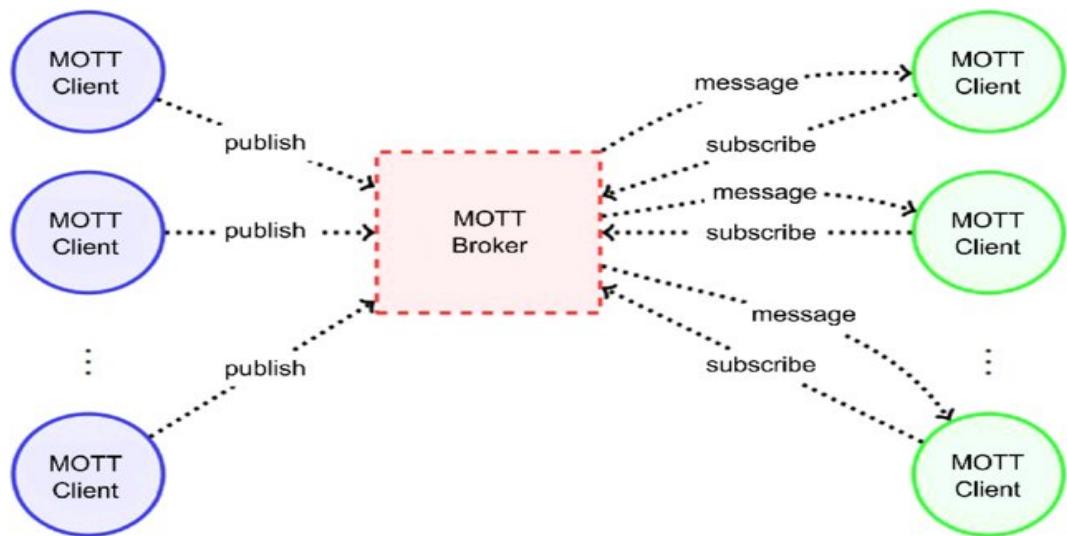


Figura 2.3: Esta imagem é o diagrama do modelo do Protocolo MQTT [1].

2.2.3.1 QoS

Quality of Service (QoS) e *Clean Session Flag* são dois recursos importantes do protocolo MQTT.

QoS define o nível de garantia de entrega de mensagens. Existem três níveis de QoS: QoS 0, QoS 1 e QoS 2.

- QoS 0 (*At most once delivery*): garante que a mensagem seja entregue no máximo uma vez, sem confirmação de entrega;
- QoS 1 (*At least once delivery*): garante que a mensagem seja entregue pelo menos uma vez, com confirmação de entrega;
- QoS 2 (*Exactly once delivery*): garante que a mensagem seja entregue exatamente uma vez, com confirmação de entrega.

O nível de QoS escolhido deve ser baseado na importância da mensagem e nas limitações de recursos do dispositivo. Por esse motivo e derivado à natureza do projeto, foi escolhido o QoS 2, pois precisamos garantir uma entrega e não há necessidade de entregar várias mensagens.

2.2.3.2 Clean Session Flag

A "*Clean Session Flag*" é uma opção que define se o cliente MQTT deve manter as informações de sessão armazenadas no servidor ou não.

Caso esta *flag* for definida como "verdadeira", o cliente inicia uma nova sessão e o servidor limpa todas as informações da sessão anterior, contudo, se esta *flag* for definida como "falsa", o cliente reconecta-se ao servidor com as informações da sessão anterior.

Neste projeto usamos a *flag* como "falsa" para manter a informação das mensagens que já foram recebidas e as que não foram. Combinando esta opção com o QoS podemos garantir que todas as mensagens serão entregues.

2.2.4 Router

Para a realização do sistema MQTT é preciso um router para a transferência de dados e para ser o *Broker* do nosso sistema MQTT.

Neste projeto são usados os *routers* RUT950 da empresa *Teltonika*.

Em resumo, o Teltonika RUT950 é um *router* confiável e versátil, bem adequado para uso em ambientes exigentes. Para além destas características, ele já vem com o protocolo MQTT previamente instalado facilitando ,assim, o desenvolvimento do projeto. A imagem 2.4 mostra um desses *routers*.



Figura 2.4: Esta imagem mostra um *Router* RUT950.

2.3 Conclusões

Através desta análise foi possível entender o sistema anterior, o estado em que se encontrava e os problemas e/ou defeitos do mesmo, bem como, os aspectos a melhorar com a nova solução.

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

Neste capítulo serão explicadas todas as tecnologias e ferramentas utilizadas para o desenvolvimento do projeto em questão.

3.2 Tecnologias e Ferramentas Utilizadas

3.2.1 *VirtualBox*

O *VirtualBox* é um *Software* de virtualização. Este permite a instalação e utilização de um sistema operacional dentro de outro como, por exemplo, dois ou mais computadores independentes, mas que compartilhem fisicamente os mesmos *Hardwares*. Mais informações podem ser encontradas no site da oracle [2] entre eles os próprios softwares e documentação.

Neste trabalho, foi utilizado este programa para ser possível usar o Ubuntu e simular os *Hardwares* necessários, o *router*, o servidor do SEGAL e o servidor do IPMA.

3.2.2 *Ubuntu*

O *Ubuntu* é um sistema operativo completo e disponível gratuitamente. O seu *download* pode ser realizado através do site do *ubuntu* [3].

Neste trabalho, foi utilizado este sistema operacional para acessar e configurar o *router*, simular e configurar clientes e, ainda, simular os servidores nos quais o *Software* desenvolvido irá ser instalado.

3.2.3 *OpenVPN*

O *OpenVPN* é um *Software* para criar redes privadas virtuais, através de túneis criptografados entre computadores. Neste trabalho, foi utilizado esta ferramenta para ser possível desenvolver o projeto fora do gabinete do SEGAL permitindo, assim, a conexão aos routers usados. Mais informações podem ser encontradas no site do *OpenVPN* [4] entre eles o próprio *Software* e documentação.

3.2.4 *LucidChart*

O *Lucidchart* é uma ferramenta online de criação de diagramas e fluxogramas que permite aos utilizadores criar diagramas e gráficos profissionais. Com o *Lucidchart*, os utilizadores podem facilmente criar fluxogramas, mapas mentais, organogramas, *wireframes*, diagramas de rede e muitos outros tipos de visualizações. Esse site pode ser consultado no site do LucidChart [5]. Neste projeto, foi usado para criar os diagramas 2.1 e 4.1.

3.2.5 *Overleaf*

O *Overleaf* é um editor online de *LaTeX* que permite aos utilizadores criar e colaborar em documentos *LaTeX*. Com o *Overleaf*, os utilizadores podem criar documentos com texto formatado, fórmulas matemáticas, figuras e referências bibliográficas. Este editor pode ser acedido pelo site do *Overleaf*[6]. Este foi usado para redigir este relatório.

3.2.6 *SQLlite3*

O *SQLite* é uma biblioteca de *Software* que fornece um sistema de gestão de base de dados relacional implementado como um mecanismo de base de dados SQL transacional, sem servidor, sem configuração e autocontido. É uma escolha popular para sistemas de bases de dados embutidas, aplicações móveis e aplicações da web de pequena escala, pois é leve, eficiente e fácil de usar. É possível consultar algumas informações extras e fazer o *download* do mesmo pelo site do *sqlite*[7].

Graças a essas características é usado para guardar informação sobre as instituições a que poderíamos dar a informação dos marégrafos e respetivas informações destes.

3.2.7 *ErdPlus*

O *ERDPlus* é uma ferramenta online usada para criar e projetar Diagramas de Entidade-Relacionamento (ERDs). Um ERD é uma representação visual das entidades (tabelas ou objetos) num banco de dados, seus atributos (propriedades ou campos) e os relacionamentos entre eles. Estas ferramentas podem ser acedidas através do site do *ErdPlus* [8].

Neste projeto foi usado para criar o esquema 4.2 que depois foi usado para criar e desenvolver a base de dados.

3.2.8 *Eclipse*

O *Eclipse* é um *Integrated Development Environment* (IDE) popularmente usado para desenvolvimento de software. Ele fornece um conjunto abrangente de ferramentas, *plugins* e recursos para facilitar a criação, edição, depuração e compilação de diversos tipos de aplicações. Este IDE pode ser consultado no site do *Eclipse*[9] bem como a sua documentação e várias bibliotecas para o mesmo.

O Eclipse é conhecido principalmente por ser usado no desenvolvimento de aplicações Java, embora também seja compatível com várias outras linguagens de programação, como C/C++, Python, PHP, Ruby, entre outras. Neste projeto foi usado para escrever todo o programa em Java.

3.2.9 **Marégrafos**

Os marégrafos são instrumentos usados para medir o nível do mar em relação a um ponto de referência conhecido. Esses instrumentos são instalados em locais costeiros e registam as variações no nível do mar ao longo do tempo.

Os marégrafos consistem numa boia ou tubo vertical selado, dentro do qual há um dispositivo de medição (geralmente um sensor de pressão) que regista as mudanças na pressão causadas pelas variações do nível do mar. Essas medições são então registadas e podem ser utilizadas para análise de padrões de maré, estudos climáticos, previsões de maré e monitorização de eventos como tempestades e tsunamis.

Assim, estes instrumentos são os responsáveis pela captação e escrita dos dados.

3.2.10 Router

O *Router* é um dispositivo de rede que encaminha pacotes de dados entre diferentes redes. Geralmente é usado para conectar redes locais (LANs) a uma rede de área ampla (WAN), como a Internet. É usado como *Broker* para este projeto e é o responsável pelo envio das mensagens para o SEGAL.

3.2.11 MQTT

O MQTT é um protocolo leve de mensagens, projetado para dispositivos com recursos limitados de rede, como sensores e dispositivos de IoT. Desse modo, é projetado para ser eficiente em termos de consumo de banda larga e energia. Mais informações podem ser encontradas no site do MQTT[10] entre eles os próprios softwares e documentação.

Embora seja incomum ter um *router* específico com suporte nativo ao MQTT, é possível configurar um *router* para atuar como um intermediário ou ponte entre dispositivos MQTT e um *Broker* MQTT externo. Neste projeto, os *routers* já vinham com este suporte facilitando em parte a implementação do sistema em geral.

Existem vários *Software* para implementar o *MQTT* tanto no *Broker* tanto como nos clientes. Entre eles temos o *Mosquitto* [11] que foi usado para realizar a simulação 5.1.5, mas existem muitos outros como por exemplo *FlashMQ* [12] e *HiveMQ* [13], cada um destes com suas próprias bibliotecas e para especificações diferentes.

3.3 Conclusões

Estas foram as ferramentas e tecnologias utilizadas para o desenvolvimento do projeto. Para além disso, foram usadas outras ferramentas não mencionadas anteriormente por privação de relevância (monitor, teclado, rato, *email*, entre outras), já que se tratam de auxílios básicos para a concretização de grande parte dos projetos tecnológicos.

Capítulo

4

Implementação

4.1 Introdução

Neste capítulo será apresentada a implementação realizada, detalhando os procedimentos efetuados.

4.2 Gestão de trabalho

Na implementação do trabalho, a divisão de tarefas foi feita por partes e pela seguinte ordem:

- Análise de todos os requisitos do projeto;
- Criação da Base de Dados;
- Criação do Gestor de Base de Dados para o SEGAL;
- Criação do Cliente para os Marégrafos;
- Criação do Cliente para o SEGAL;
- Criação de um Cliente Global;
- Criação de *script* para testes;
- Testes e correções;
- Implementação do sistema em situações reais.

Durante todas estas etapas, este relatório foi escrito e atualizado continuamente.

4.3 Implementação

4.3.1 Vista geral da nova arquitetura

Nesta secção, o projeto inteiro é descrito de forma rápida e simplificada. Inicialmente, cada marégrafo possui algumas informações necessárias para realizar uma conexão ao *Broker*. Obrigatoriamente, tem de possuir um *URL* e, preferencialmente, deve ser incluído um *user* e uma *password* para se conseguir conectar (nos testes foi avaliado com e sem estes parâmetros, contudo, na versão final ficou sem, por comodismo). Tendo isto estabelecido, é possível a conexão através de MQTT entre um *Broker* e um cliente.

O Publicador necessita ainda de um tópico onde vai escrever a mensagem que, por praticidade, foi definido como "Marégrafo" em todas as conexões entre os marégrafos e o SEGAL. O *publisher* do SEGAL escreve as mensagens recebidas em vários tópicos diferentes que são usadas para controlar que cliente recebe quais mensagens.

Para controlar estes dados, as informações são guardadas numa base de dados juntamente com informações de instituições e, ainda, quais destas têm acesso ao marégrafo.

Essa base de dados pode ser editada através do gestor, um programa desenvolvido em Java com esse propósito.

Os clientes no SEGAL conectam-se à base de dados para buscar os dados necessários para realizar essas conexões. Como é necessário guardar a informação recebida pelo MQTT, os clientes têm uma função que guarda esses dados num *Plaintext* que recebe o nome do Marégrafo e extenção ".log".

Para melhor compreensão, consultar a imagem 4.1.

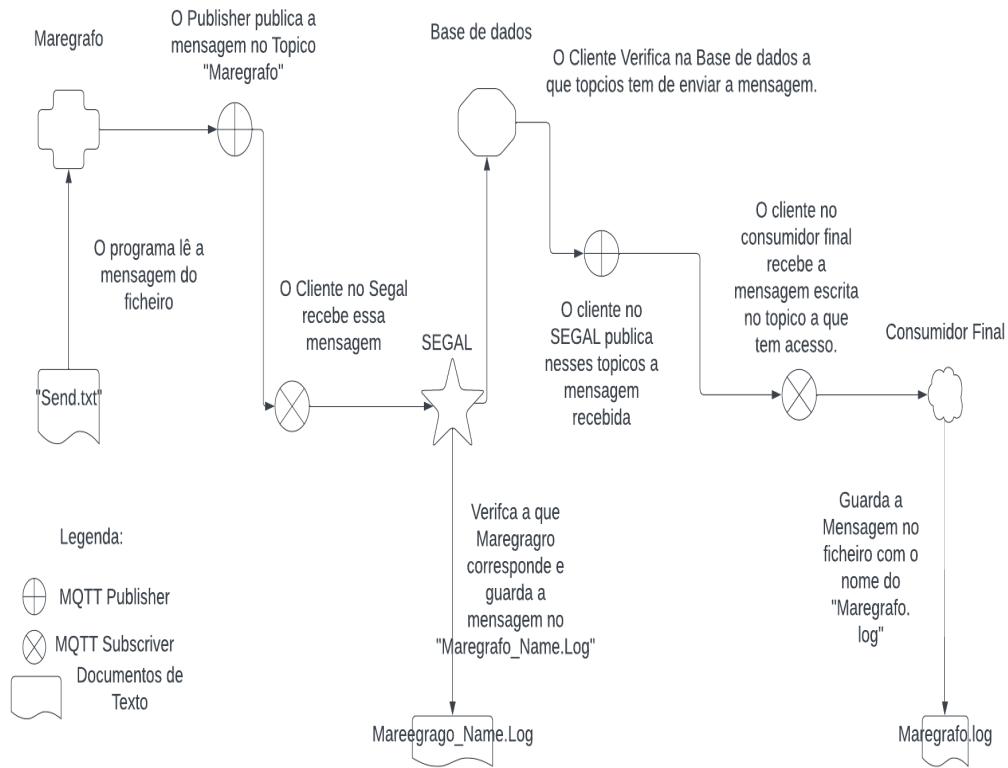


Figura 4.1: Esta imagem representa de forma simplificada o sistema.

Nota: Para a realização deste projeto foram considerados alguns adendos para ser possível a execução do mesmo. Entre eles, não foi obtida resposta para a possibilidade da instalação de um cliente nos servidores do IPMA. Além disso, não foi possível a confirmação de que os marégrafos possam escrever no ficheiro "Send.txt". Por último, não existia *Hardwares* suficiente para realizar todo o sistema e, por isso, o *Broker* do SEGAL é o mesmo que é usado para simular o marégrafo.

4.3.2 Base de Dados

Como mencionado anteriormente, o projeto necessitava de uma base de dados para armazenar os dados dos marégrafos, das instituições e a relação entre eles. Assim sendo, foi criado uma base de dados em *Sqlite3* que cumprisse esses objetivos.

Do mesmo modo, existiam também algumas restrições e/ou situações que tinham de ser respeitadas, tais como o facto de cada instituição conseguir dar

um nome diferente a um marégrafo mas, ao mesmo tempo, cada marégrafo só conseguir ter um nome por instituição. Havia também o caso de que um marégrafo podia ser analisado por várias instituições diferentes, só por uma instituição, ou até por nenhuma, mas tinha de ser mantido na base de dados na mesma. Por fim, cada instituição tem de ter um tópico diferente para garantir a confidencialidade e o acesso apenas aos marégrafos supostos. O design da base de dados segue o esquema da imagem 4.2.

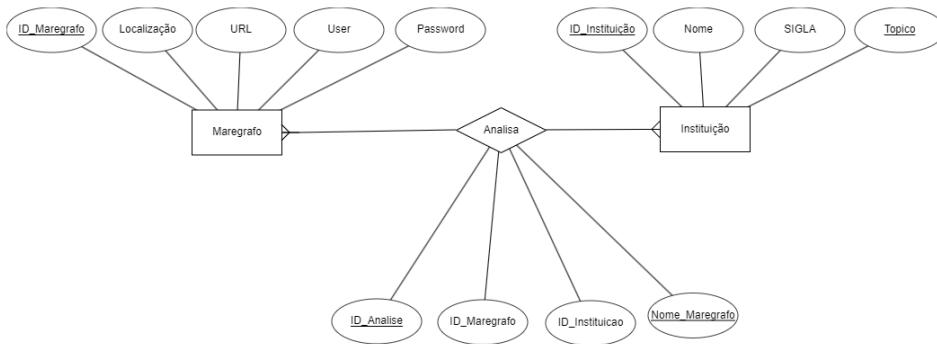


Figura 4.2: Esta imagem é uma representação da estrutura da Base de Dados.

Em seguida, essa estrutura foi respeitada e criada através do seguinte *script*.

```

CREATE TABLE Maregraph (
    ID_Maregraph INTEGER PRIMARY KEY,
    Location TEXT,
    URL TEXT,
    User TEXT,
    Password TEXT
);
CREATE TABLE Institution (
    ID_Institution INTEGER PRIMARY KEY,
    Name TEXT,
    Sigla TEXT,
    Topic TEXT,
    CONSTRAINT unique_name UNIQUE (Topic)
);
CREATE TABLE Analyze (
    ID_Analyze INTEGER PRIMARY KEY,
    ID_Maregraph INTEGER,
    ID_Institution INTEGER,
    Name TEXT,
    FOREIGN KEY (ID_Maregraph) REFERENCES Maregraph(ID_Maregraph),
    FOREIGN KEY (ID_Institution) REFERENCES Institution(ID_Institution),
    CONSTRAINT unique_name UNIQUE (ID_Maregraph, ID_Institution)
);
  
```

```
 );
```

Exerto de Código 4.1: Script para criação da base de dados.

Para ser possível realizar testes foram, ainda, adicionados os seguintes dados de teste com o seguinte *script*.

```
Insert into Maregraph(Location ,URL,User ,Password) Values ('Cascais ','  
10.0.7.45:1833 ',' ',' ');  
Insert into Maregraph(Location ,URL,User ,Password) Values ('Faro ','  
10.0.7.46:1833 ',' ',' ');  
Insert Into Institution(Name,Sigla ,Topic) Values ('(Space & Earth  
Geodetic Analysis Lab','SEGAL','Maregrafo');  
Insert Into Institution(Name,Sigla ,Topic) Values ('Instituto Portugues do  
Mar e da Atmosfera','IPMA','ipmatopics');  
Insert Into Institution(Name,Sigla ,Topic) Values ('Uk Met Office ','UKMET'  
'ukmettopics');  
Insert Into Analyze(ID_Maregraph, ID_Institution ,Name) Values(1,1,'Faro ')  
;  
Insert Into Analyze(ID_Maregraph, ID_Institution ,Name) Values(2,1,'  
Cascais');  
Insert Into Analyze(ID_Maregraph, ID_Institution ,Name) Values(1,2,'  
FaroIPMA');  
Insert Into Analyze(ID_Maregraph, ID_Institution ,Name) Values(2,2,'  
CascaisIPMA');  
Insert Into Analyze(ID_Maregraph, ID_Institution ,Name) Values(2,3,'  
FaroUkData');
```

Exerto de Código 4.2: Dados usados para testes

4.3.3 Gestor para o SEGAL

Após a criação da base de dados e de esta estar pronta para uso, era necessário um gestor para a mesma.

Assim sendo, para editar esta base de dados foi criado um programa em Java, para ser usado pelo SEGAL. Com este programa chamado "Gestor", o SEGAL consegue adicionar, remover ou editar instituições, marégrafos e/ou as suas conexões.

Este gestor, de forma resumida, consiste em vários menus com variadas opções. Através destes menus é possível editar toda a base de dados. Todo o programa foi desenvolvido para ser usado pelo *prompt* de comandos e sem interface gráfica, pois é visado para ser usado em servidores.

O seguinte segmento de código 4.3 é um excerto presente no gestor. Especificamente, é usado para conectar-se à base de dados e adicionar um novo marégrafo.

```
 ...
```

```

if (confirm) {
    try {
        Connection conn = DriverManager.getConnection(DBLocation);
        PreparedStatement pstmt = conn.prepareStatement("INSERT INTO
            Maregraph (Location, URL, User, Password) VALUES (?, ?, ?, ?)");
        pstmt.setString(1, Location);
        pstmt.setString(2, URL);
        pstmt.setString(3, User);
        pstmt.setString(4, Password);
        pstmt.executeUpdate();
        pstmt.close();
        conn.close();
        System.out.println("Sucesso : Adicionado com sucesso !");
    } catch (SQLException e) {
        System.out.println(e.getMessage()); // In case of error, display
                                         the error message
    }
} else {
    System.out.println("Cancelado : Cancelado com sucesso !");
}
...

```

Exerto de Código 4.3: Exemplo de código para adicionar um marégrafo na base de dados.

4.3.4 Marégrafos

Apesar de não ter muita relevância prática para o projeto, o marégrafo tem acesso a um ficheiro no *router*, onde escreve os dados lidos.

4.3.5 Router

O *router*, como explicado anteriormente, recebe os dados do marégrafo. Este *router* tem instalado o software MQTT e, ainda, o nosso cliente desenvolvido em Java.

4.3.6 Publisher no router

O programa instalado no router monitoriza um ficheiro chamado "Send.txt". Esse programa cria um cliente MQTT que tenta conectar-se ao *Broker*. Caso este se consiga conectar, o programa lê cada linha do ficheiro e envia-o. Se este for enviado com sucesso para o *Broker*, ele apaga essa linha do ficheiro.

Na hipótese do cliente não conseguir se conectar por algum motivo, o programa continua a supervisionar o ficheiro, mas apenas comparando as datas das mensagens. Caso a data da primeira linha do ficheiro (correspondente

à mensagem mais antiga que não foi enviada) tenha a diferença de seis horas para com a última data do ficheiro (correspondente à mensagem mais recente) este apaga a primeira linha do ficheiro. Isto acontece porque o IPMA pediu que, em caso de falha, se envie apenas as seis horas mais recentes. Quando é possível a reconexão, o programa envia estas linhas e volta a funcionar na normalidade.

O trecho de código 4.4 mostra como o cliente verifica a linha e se está conectado ao *Broker*.

```
...
// Reads the file line by line and sends each line to the function
// that sends the message
public static void readData() {
    while (true) { // Infinite loop to check the file
        try {
            Scanner scanner = new Scanner(file); // Create a scanner
            to read the file
            if (file.length() > 0) { // Check if the file is not
                empty (overkill, but used to save resources)
                while (scanner.hasNextLine()) { // Loop to read the
                    entire file
                    if (!Publisher.isConnected()) { // Check if the
                        client is connected
                        isConnected = false; // Set the boolean to
                        false to indicate not connected
                        connect(); // Function to connect the client
                    } else {
                        sendData(scanner.nextLine()); // Function to
                        send the message via MQTT
                        if (!scanner.hasNextLine()) { // Check if
                            the end of the file is reached
                            FileWriter writer = new FileWriter(file,
                                false); // Clear the contents of
                                the file
                            writer.close(); // Close the writer as
                                the file has been cleared
                        }
                    }
                }
            }
            scanner.close(); // Close the scanner that reads the
            file
        } catch (IOException e) {
            System.out.println(e.getMessage()); // In case of error,
            display the error message
        }
    }
}
```

...

Exerto de Código 4.4: Este excerto de código verifica a conexão com o Broker e lê a linha.

4.3.7 Segurança

As comunicações entre *Broker* e clientes podem ser realizadas através de SSL. O SSL (*Secure Sockets Layer*) é um protocolo de segurança padrão que estabelece uma conexão criptografada entre um cliente e um servidor para fins de autenticação e privacidade.

Para ser possível esta comunicação é preciso criar um arquivo para armazenar certificados duma Autoridade de Certificação (CA - Certificate Authority). Estas são entidades confiáveis que emitem certificados digitais para autenticar a identidade de organizações, servidores, sites e indivíduos.

Após a criação de uma CA criamos um certificado digital para o *Broker* assinado por esta. Isto é feito através da criação dum arquivo CRS (Certificate Request) que é assinado pela CA. Por fim criamos um certificado digital assinado pelo mesmo CA para o(s) cliente(s).

Após a criação destes fazemos *upload* do certificado digital do *Broker* e do certificado digital do CA para o *Broker*, que irá atuar como servidor, e especificamos a versão do SSL que desejamos utilizar.

Esse teste pode ser consultado no capítulo 5.1.5.

Curiosamente, foi mesmo necessário criar um certificado para o cliente para fazer autenticação do servidor e do cliente. Habitualmente, numa conexão cliente (*browser*) com um servidor da Internet apenas é necessário um certificado do servidor.

Esta comunicação foi testada com sucesso, mas devido a problemas no *Hardware* (*router* disponível) tornou-se inviável a utilização prática da mesma. Existia um *bug* que fazia **reset** aos dados configurados no *router*.

Para além dum mecanismo de autenticação via ssl/tls com certificados X.509, o MQTT possui também um sistema de *user* e *password* para garantir apenas a publicação de mensagens por pessoas de confiança.

4.3.8 Cliente para o SEGAL

Posteriormente ao *router* enviar as mensagens para o *Broker*, o SEGAL tem de receber as mesmas. Assim sendo, foi feito um programa para ser executado nos servidores do SEGAL. Este programa conecta-se à base de dados e, em seguida, vai buscar a quais *Brokers* este se pode conectar e as informações necessárias para isso. Após isso, este tenta conectar-se a todos os *Brokers*,

recebendo as mensagens deles e escrevendo-as em ficheiros com a seguinte estrutura de nome "NomeMarégrafo.Log". Ao mesmo tempo que este programa recebe as mensagens, ele analisa-as e vê para quais clientes de outras instituições tem de as enviar. O trecho de código seguinte mostra como é tratado a parte de escrever no ficheiro.

```
...
// Function to create the log file
// Checks if the directory and file exist
public static void CreateLogFile(Path path, String logName, String
    data) {
    try {
        File logFile = new File(path + "/" + logName + ".log"); // Create the file with .log format
        logFile.createNewFile(); // Create the file if not exist
        WriteOnLog(logFile, data); // Function to write the message to it
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage()); // In case of error, display the error message
    }
}

// Function to write to the file
public static void WriteOnLog(File logFile, String data) {
    try {
        FileWriter myWriter = new FileWriter(logFile, true); // Create the writer that doesn't overwrite existing content in the file
        myWriter.write("\n" + data); // Write the message
        myWriter.close(); // Close the writer
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage()); // In case of error, display the error message
    }
}
...
```

Exerto de Código 4.5: Trecho de código responsável por escrever no ficheiro

4.3.9 Publisher SEGAL

O publicador SEGAL, à medida que recebe as mensagens, analisa o nome do marégrafo de que as mesmas foram enviadas. Com essa informação, ele procura na base de dados a que marégrafo esse nome se refere. Assim, ele verifica também quais as instituições que têm acesso a esse marégrafo e a que tópico está associado a essa instituição (relembrando que cada instituição tem

um tópico único). Posto isto, ele reencaminha a mensagem com a estrutura "Nome do Marégrafo|Mensagem" para o tópico a que cada instituição é subscrita.

```
// Function to process the message and send it to the Publisher
// for publishing
public static void Send(String message, String Name_Maregraph) {
    getDBInfo(Name_Maregraph); // Retrieve information from the
                                database
    for (String Data : list) {
        String[] Parts = Data.split("\|"); // Split at the "|" character
        String Topic = Parts[0]; // Topic comes before "|"
        String Name = Parts[1]; // Maregraph Name comes after "|"
        String newMessage = Name + "|" + message; // Create a new
                                                    message with the Maregraph Name and the message
        Main.publisher.sendData(Topic, newMessage); // The publisher
                                                    publishes the new message to the topic
    }
}

// Function to retrieve information from the database
public static void getDBInfo(String Name_Maregraph) {
    list.clear(); // Clear the list
    ConnectionDB.getPublisher(Name_Maregraph); // Function that
                                                retrieves who to send the messages to from the database
}
```

Excerto de Código 4.6: Trecho de código que faz a preparação da nova mensagem.

4.3.10 Cliente Global

Desta forma, só falta os clientes finais receberem essa mensagem. Para isso, é instalado um cliente nos servidores dos mesmos que se conectará ao *Broker* do SEGAL. Ao receber a mensagem, este cliente separa-a em "nome do marégrafo" e na "mensagem" em si. Assim, o cliente sabe a que marégrafo corresponde essa mensagem e em que ficheiro a deve escrever.

No trecho de código seguinte é mostrado como é realizado essa separação.

```
// Function that is called whenever the client receives a message
// It checks if the topic is the one we want
// Writes the message to the Client log
@Override
public void messageArrived(String TopicS, MqttMessage message)
    throws Exception {
    if (TopicS.equals(Topic)) { // Check if the topic is the
                                intended one
```

```

String data = new String(message.getPayload(),
    StandardCharsets.UTF_8); // Receive the message
String [] Parts = data.split("\|"); // Split the message
String Name_Maregraph = Parts[0]; // Receive the name of the
                                maregraph
String Message = Parts[1]; // Receive the message itself
WriteLog.CreateLogFile(Client.path, Name_Maregraph, Message)
    ; // Write the message to the log
}
}

```

Exerto de Código 4.7: Trecho de código que faz a separação da mensagem.

4.3.11 *Script para testes*

Para testar todo o sistema foi gerado um *script* em Python que simula a escrita de um marégrafo no ficheiro "Send.txt". Esse *script* corre numa *virtual box* que simula um *router* e que, em seguida, é a responsável por enviar as mensagens simuladas do marégrafo para o resto do sistema.

Este *script* é representado no trecho de código 4.8.

```

Contador= 0
Path='Projeto/Logs/Maregrafo/'

def main():
    while True:
        global Path
        os.makedirs(Path, exist_ok=True)
        Date=datetime.datetime.now()
        File_name=Path+Date.strftime('ECXX5s'+'%Y%m%d') + '.dat'
        DataToSend>Data(Date.strftime("%Y-%m-%d %H:%M:%S"))
        with open(File_name, 'a') as File:
            File.write(DataToSend)
            File.close()
        with open(Path+"Send.txt", 'a') as File:
            File.write(DataToSend)
            File.close()
        time.sleep(5)

def Data(Date):
    Profundidade= round(random.uniform(0,2),3)
    global Contador
    Contador +=1
    Info = f'{Date} {Profundidade} 0 ' 'Teste' {Contador}\n'
    return Info

```

Exerto de Código 4.8: Este script é usado para simular a escrita dos marégrafos no ficheiro "Send.txt".

Como o *script* é usado apenas para testes internos realizados por mim, este não está comentado nem "traduzido", já que não é suposto ser usado por mais ninguém nem implementado no projeto em si .

4.4 Conclusões

Neste capítulo foi descrito o processo de implementação já concluído, juntamente com algumas imagens e excertos do código de suporte. Assim sendo, é importante relembrar que não está descrito todo o trabalho realizado e/ou todos os testes e que este não representa a totalidade do trabalho e dos problemas enfrentados na realização do mesmo.

Capítulo

5

Testes e simulações

5.1 Introdução

Nesta secção, é descrito de forma rápida e simplificada alguns testes que foram realizados e como o sistema em si se deve comportar. Também é abordado o comportamento esperado e obtido em algumas situações passíveis de ocorrer.

5.1.1 Comportamento Normal

Nesta simulação não ocorreram problemas (erros de conexão, erros de instalação, entre outros). Sendo assim, os dados são lidos pelo nosso cliente no marégrafo e enviados em tempo real através do protocolo MQTT e com a conexão por TCP para o servidor presente no SEGAL. Nesse momento, o cliente no SEGAL recebe os dados, escreve-os num ficheiro com o nome do marégrafo e envia-os para os clientes.

Na imagem 5.1 observa-se na *virtual box* o programa que envia os dados para o SEGAL e o *script* a executar para gerar os dados. Para além disso, vemos os dois ficheiros criados e com os dados mais recentes, sendo que cada um tem o nome que lhe foi atribuído na base de dados. Foi usado só um marégrafo, pois os resultados obtidos não variam independentemente se é um caso isolado ou vários ao mesmo tempo. Então, por facilitismo, só se simulou um marégrafo .

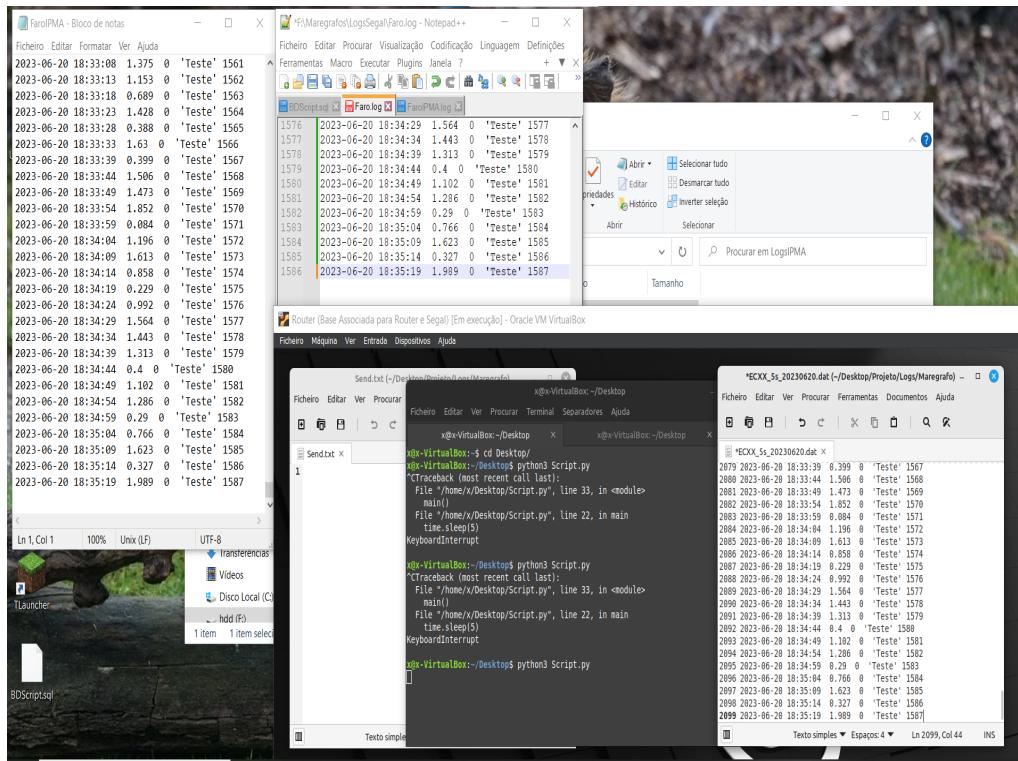


Figura 5.1: Esta imagem mostra a simulação acima.

5.1.2 Falha no Marégrafo

Nesta simulação, o marégrafo perdeu a conexão ao *Broker* durante algum tempo. A partir do momento em que a conexão foi perdida, uma *thread* começa a ser executada em segundo plano. Essa *thread* compara as datas da ultima e da primeira mensagem e, se essa diferença for superior a seis horas, a mensagem mais antiga será apagada e ignorada. Assim que a conexão for possível, essa *thread* é suspensa e todas as outras mensagens serão enviadas normalmente.

Na imagem 5.2 foi simulado este caso mas, em vez de seis horas, era um minuto de diferença, ou seja, ao fim de um minuto a *thread* apaga a mensagem mais antiga. Desta maneira, como podem ver onde refere as datas em que foram enviadas, estas diferem em um minuto. Para além disso, todas as outras mensagens foram enviadas, recebidas pelo SEGAL e enviadas para o IPMA sem problemas.

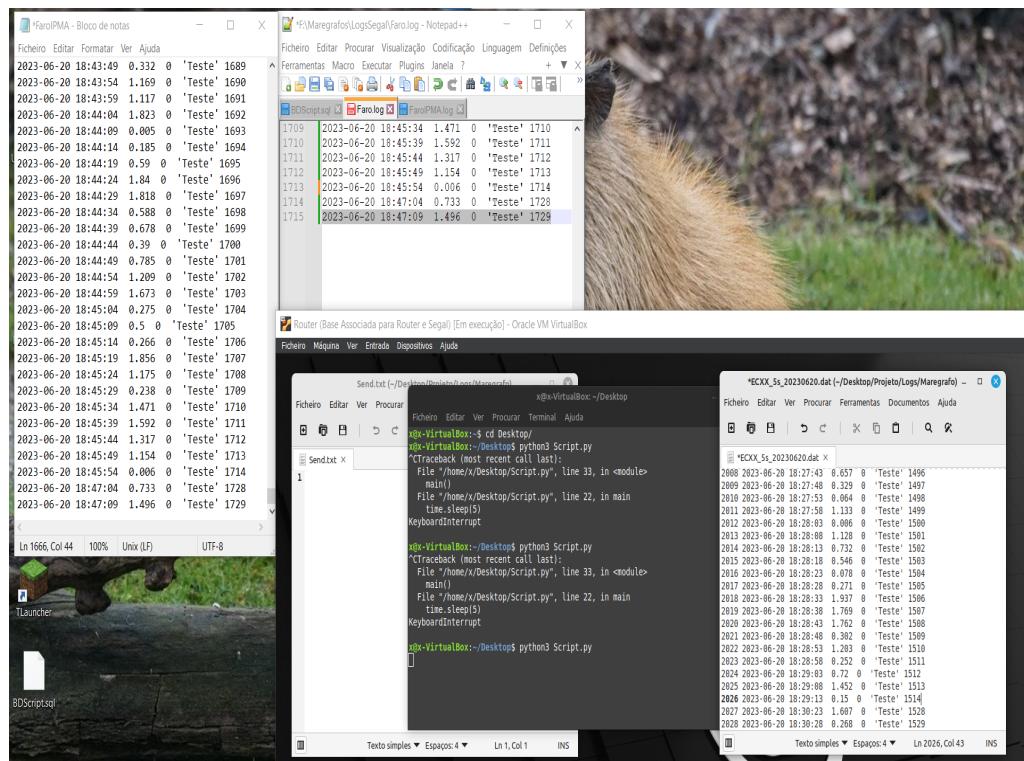


Figura 5.2: Esta imagem mostra a simulação acima.

5.1.3 Falha no Segal

Nesta simulação, o SEGAL perdeu a conexão ao *Broker* do marégrafo. Assim, o SEGAL fica a tentar a reconexão com o *Broker* do marégrafo. Quando esta volta a estar disponível, o SEGAL recebe as mensagens que foram enviadas pelo *Broker* para ele e, simultaneamente, o SEGAL envia essas mensagens para os outros clientes.

Na imagem 5.3 foi simulado este caso e como é visível, todas as mensagens foram enviadas mesmo que o cliente do SEGAL estivesse desconectado por cerca de dois minutos. Quando este se reconectou ao *Broker*, ele acabou por receber as mensagens.

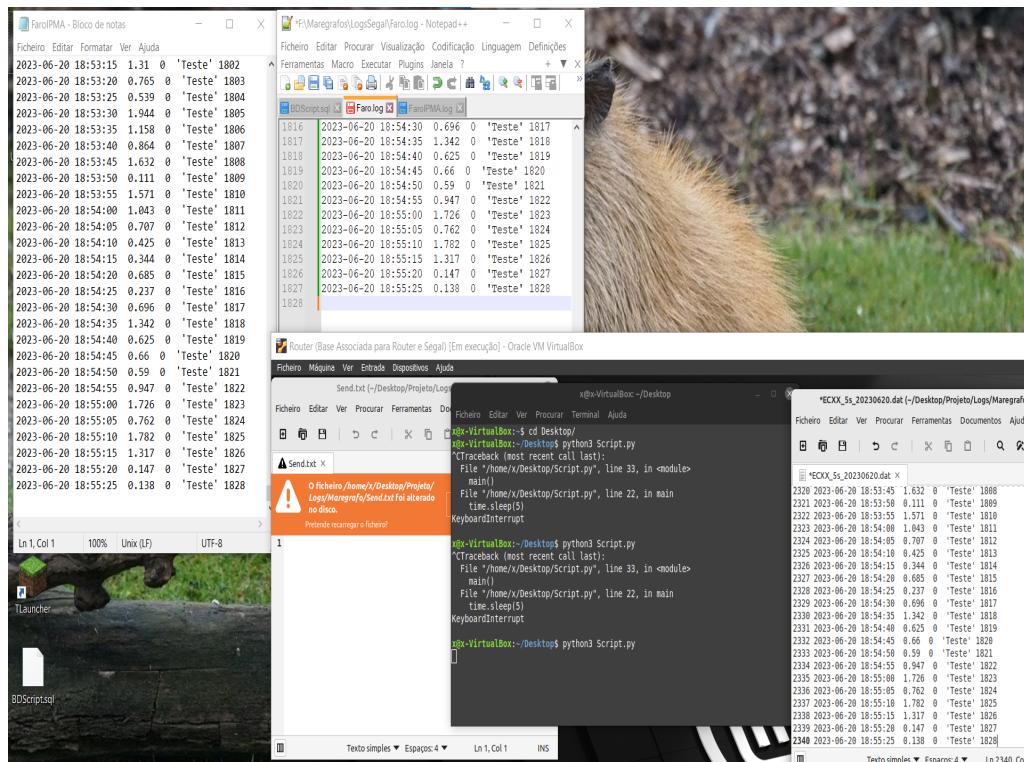


Figura 5.3: Esta imagem mostra a simulação acima.

5.1.4 Falha no Cliente

Nesta simulação, é simulada a perda de conexão do cliente final com o *Broker* do SEGAL. Neste caso, quando a reconexão for possível, ele recebe todas as mensagens que o SEGAL recebeu.

Na imagem 5.4 foi simulado este caso e, como é visível, todas as mensagens foram enviadas mesmo o cliente do IPMA estivesse desconectado por cerca de três minutos, quando este se reconectou ao *Broker* do SEGAL ele acabou por receber as mensagens.

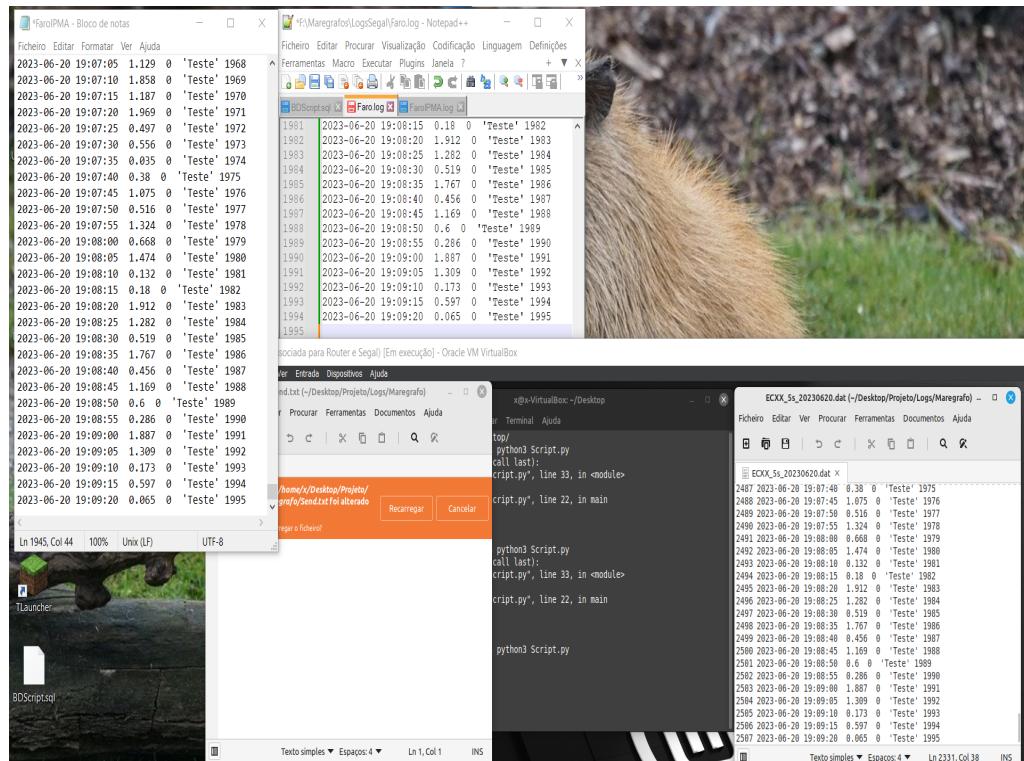
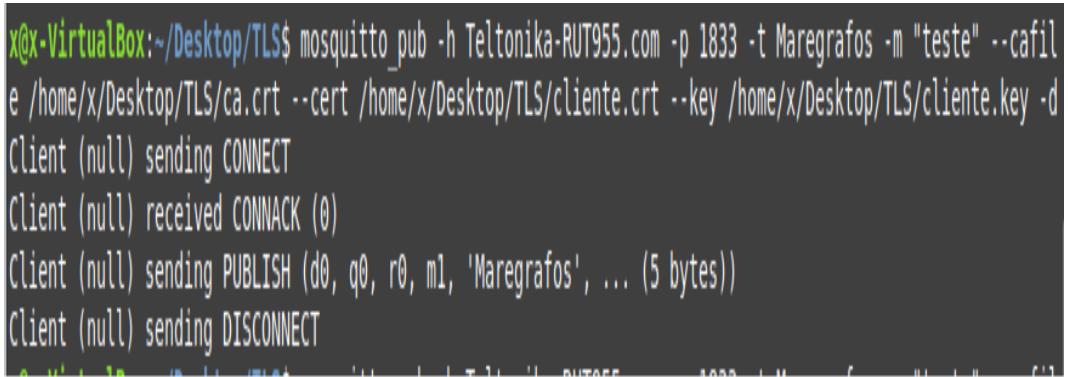


Figura 5.4: Esta imagem mostra a simulação acima.

5.1.5 Teste do SSL

Como mencionado no 4.3.7, foi testado a comunicação por SSL. Com os ficheiros criados e com auxílio do *mosquitto* (um *Software* de MQTT para *linux*) foi realizado o teste da comunicação.

Na imagem 5.1.5, podemos ver o envio de uma mensagem para um tópico no *Broker* e, ainda, o envio do CA, do cliente CRT e da chave do cliente. Vemos que foi realizada a comunicação e enviada a mensagem com sucesso.

A screenshot of a terminal window titled 'Terminal' showing the execution of a 'mosquitto_pub' command. The command is: 'mosquitto_pub -h Teltonika-RUT955.com -p 1833 -t Maregrafos -m "teste" --cafile /home/x/Desktop/TLS/ca.crt --cert /home/x/Desktop/TLS/cliente.crt --key /home/x/Desktop/TLS/cliente.key -d'. The output shows the client connecting to the broker, receiving a connection acknowledgement, publishing the message 'teste' to the topic 'Maregrafos', and then disconnecting.

```
x@x-VirtualBox:~/Desktop/TLS$ mosquitto_pub -h Teltonika-RUT955.com -p 1833 -t Maregrafos -m "teste" --cafile /home/x/Desktop/TLS/ca.crt --cert /home/x/Desktop/TLS/cliente.crt --key /home/x/Desktop/TLS/cliente.key -d
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'Maregrafos', ... (5 bytes))
Client (null) sending DISCONNECT
```

Figura 5.5: Esta imagem mostra a simulação acima.

5.2 Conclusão

Ao longo deste capítulo, exploramos a importância dos testes, bem como o suposto comportamento do programa desenvolvido. Os testes desempenham um papel crucial na garantia da qualidade do produto final, identificando e corrigindo erros, defeitos e problemas de desempenho antes que o *Software* seja implantado num ambiente de produção.

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Com a realização deste projeto e relatório, foi possível explorar conhecimentos de base de dados, sistemas distribuídos, engenharia de software, sistemas operativos, entre outros. Além disso, usufrui da oportunidade de consolidar os conhecimentos adquiridos ao longo desta licenciatura e que me ajudarão no futuro. Do mesmo modo, gostei de desenvolver este programa por ter uma utilidade real, ser algo diferente do que estou habituado a realizar e por visualizar muitas opções a desenvolver com este contexto.

6.2 Trabalho Futuro

Em atualizações futuras poderia ser adicionado um diretório na base de dados onde teríamos de colocar o ficheiro log em vez deste estar definido estaticamente no programa em si. Além disso, poderia se adicionar a mensagem na base de dados ao mesmo tempo que se adiciona no ficheiro de texto associado ao marégrafo.

Outro aprimoramento seria um sistema de controlo melhorado com sistema de avisos de perda de conexão e, ainda, um sistema de inteligência artificial que analisaria em tempo real os dados do marégrafo, reconheceria dados indicadores de catástrofes (tsunamis) e enviaria um alerta. Nesse contexto também poderia ser adicionada uma integração a dispositivos móveis.

Para além disso, pensei em adaptar este projeto a um contexto de sistema de regas para agricultores. Assim, seria usado o mesmo princípio na análise de poços de água, permitindo ao agricultor saber o seu estado sem necessidade de se deslocar até eles. Através disto, seria possível analisar se o poço

esvaziou, se o motor parou de bombear água ou se alguma coisa de diferente aconteceu. Isto seria viável através de medições de consumo energético, pressão de água e profundidade do poço.

Bibliografia

- [1] Khalid Aloufi and Omar Alhazmi. A hybrid iot security model of mqtt and uma. *Communications and Network*, 12:155–173, 11 2020.
- [2] Virtual Box. Oracle VM VirtualBox. Online, 2023.
- [3] Ubuntu. Enterprise Open Source and Linux | Ubuntu. Online, 2023.
- [4] OpenVPN. Business VPN For Secure Networking | OpenVPN. Online, 2023.
- [5] Lucid. Intelligent Diagramming | Lucidchart. Online, 2023.
- [6] OverLeaf. Overleaf, Online LaTeX Editor. Online, 2023.
- [7] SQLLite. SQLlite Home Page. Online, 2023.
- [8] ErdPlus. ErdPlus. Online, 2023.
- [9] Eclipse. The Community for Open Innovation and Collaboration | The Eclipse Foundation. Online, 2023.
- [10] MQTT. MQTT: The Standard for IoT Messaging. Online, 2023.
- [11] Mosquitto. Eclipse Mosquitto. Online, 2023.
- [12] FlashMQ. FlashMQ. Online, 2023.
- [13] HiveMQ. HiveMQ. Online, 2023.