



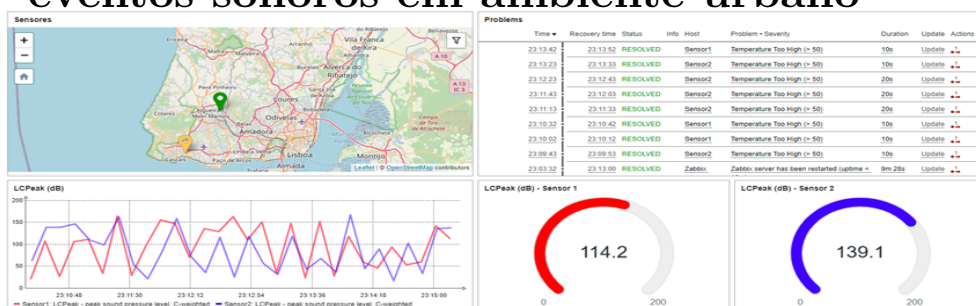
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA (ISEL)

DEPARTAMENTO DE ENGENHARIA ELETRÓNICA E DE
TELECOMUNICAÇÕES E COMPUTADORES (DEETC)

LEIRT

LICENCIATURA EM ENGENHARIA INFORMÁTICA, REDES E TELECOMUNICAÇÕES
UNIDADE CURRICULAR DE PROJETO

SoundDash — *Dashboard* agregador de eventos sonoros em ambiente urbano



Carlos Tavares (48725)

Frederico Martins (48709)

Orientadores

Professor Joel Paulo

Professor Carlos Gonçalves

Julho, 2023

Resumo

Este projeto aborda o paradigma da identificação de eventos sonoros, em tempo real, nas áreas urbanas.

A proposta deste projeto consiste na implementação de um sistema informático *online* que permita a análise, agregação e visualização de dados relativos aos eventos sonoros detetados por uma rede distribuída de estações de captura de som, colocadas junto a vias rodoviárias em diferentes localizações.

Deste modo, é possível obter o conhecimento mais rápido e preciso de ocorrências de eventos sonoros que traduzam situações potencialmente perigosas, como por exemplo tiros, acidentes rodoviários e explosões.

Tendo esta informação em mãos, as autoridades poderão ser alertadas de forma mais rápida, e provavelmente salvar vidas humanas que, em outra situação, estariam em risco, e é possível contribuir positivamente para a segurança pública e a gestão territorial.

A solução proposta foi desenvolvida utilizando a plataforma **Zabbix**, uma plataforma de gestão e monitorização de rede e aplicações de código aberto. Nesta plataforma, foram utilizados os recursos necessários para agregar, apresentar e interpretar dados de eventos sonoros detetados por sensores, de forma clara e intuitiva. Estes sensores assumem-se previamente desenvolvidos e implementados por um outro projeto em curso, estando já em funcionamento e prontos para fornecer os dados para o sistema. Com a criação de *Dashboards* personalizados, foi permitida a visualização de gráficos temporais e espectrais e outras informações relevantes por parte dos utilizadores, nomeadamente realizar estatísticas dos dados sonoros, obtendo, assim, uma visão clara e abrangente da situação atual de uma determinada localização.

Palavras-chave Segurança Pública, Gestão Territorial, Análise de Eventos Acústicos, Análise em Tempo Real, Bases de Dados, *Dashboard*, Análise Estatística de Dados Sonoros.

Abstract

This project addresses the importance of identifying sound events, in real time, in urban areas.

The purpose of this project is to implement an online computer system that allows the analysis, aggregation and visualization of data related to sound events detected by a distributed network of sound capture stations, placed next to roadways in different locations.

In this way, it's possible to obtain faster and more accurate knowledge of occurrences of sound events that translate potentially dangerous situations, such as shots, road accidents and explosions.

Having this information in hands, the authorities will be able to be alerted more quickly, and probably save human lives that, in another situation, would be at risk, and it's possible to contribute positively to public safety and territorial management.

The proposed solution was developed using the **Zabbix** platform, a network management and monitoring platform and open source applications. On this platform, the necessary resources were used to aggregate, present and interpret data from sound events detected by sensors, in a clear and intuitive way. These sensors are assumed to be previously developed and implemented by another ongoing project, being already in operation and ready to provide data to the system. With the creation of customized *Dashboards*, it was possible for users to visualize temporal and spectral graphs and other relevant information, namely to carry out sound data statistics, thus obtaining a clear and comprehensive view of the current situation occurring in a certain location.

Keywords Public Security, Territorial Management, Acoustic Events Analysis, Real-Time Analysis, Databases, Dashboard, Statistical Analysis of Sound Data.

Agradecimentos

Agradecemos a todos que nos apoiaram nessa jornada, incluindo os nossos orientadores Carlos Gonçalves e Joel Paulo, que sempre nos ajudaram e deram o suporte necessário desde o início da realização deste projeto.

Também damos o nosso obrigado aos nossos colegas de turma, familiares e amigos mais próximos, cujo incentivo e suporte também foram fundamentais, não só para a realização deste projeto tão ambicioso e que vai tão ao encontro dos objetivos que precisamos de alcançar nos dias de hoje para um mundo melhor e mais seguro, mas também na jornada da nossa vida desde o início até à fase atual.

Prefácio

Após um grande esforço que realizámos no ensino secundário para conseguirmos alcançar o objetivo almejado por muitos, que é entrar em uma faculdade pública, ao longo da nossa jornada académica na faculdade, enfrentámos inúmeros desafios e obstáculos que nunca tínhamos enfrentado antes. Por outro lado, estas dificuldades, por vezes extremas, foram, sem dúvida, fundamentais para o nosso crescimento e desenvolvimento não só a nível académico, como também a nível pessoal.

Após esta fase da nossa vida académica que foi tão crucial, temos a honra de ter chegado a esta fase final do curso e poder apresentar, em público, o projeto que escolhemos como ponto culminante da nossa formação académica.

No início deste semestre, foram-nos apresentadas, pelo coordenador do curso, várias propostas de projetos que se baseavam nas diferentes áreas que abrangem o nosso curso, e tivemos então a oportunidade de analisar, com cautela, cada uma das propostas, e com base nas áreas em que nos identificamos mais, escolher o nosso projeto. A partir dessa experiência, surgiu a motivação para o desenvolvimento deste projeto.

A escolha desse projeto em particular foi motivada não apenas pela paixão que temos pela área da Informática e de desenvolvimento de *software*, mas também pelo impacto significativo que ele pode ter na sociedade. Conscientes da importância da segurança pública e da gestão do território, propusemos-nos a desenvolver um sistema inovador e abrangente que contribua para um mundo mais seguro.

Esperamos que este projeto contribua o máximo possível para a área na qual o nosso projeto se enquadra, demonstrando o potencial do mundo da Informática para enfrentar problemas reais, que seja uma porta para futuras descobertas e realizações revolucionárias e que possamos, juntos, construir um futuro promissor no mundo da Informática e ter um impacto positivo na sociedade atual.

Índice

Lista de Figuras	xv
Lista de Tabelas	xvii
Glossário	xix
1 Introdução	1
1.1 Enquadramento	1
1.2 Contributos do Projeto	2
1.3 Organização do Relatório	2
2 Trabalho Relacionado	5
2.1 Sensores de Recolha de Sons	5
2.1.1 <i>Libelium Noise Level Sensor</i>	5
2.1.2 <i>Real-Time Noise Monitoring for Smart Cities</i>	7
2.1.3 <i>FI-Sonic</i>	8
2.2 Estado Atual de Conhecimento	10
2.2.1 Zabbix	12
2.2.2 <i>Dashboard</i>	13
2.2.3 Sensor	14
3 Análise de Requisitos	15
3.1 Casos de Utilização	16
3.2 Requisitos	17
3.2.1 Requisitos Funcionais	17
3.2.2 Requisitos Não Funcionais	18

4	Implementação	21
4.1	Ambiente de Execução	22
4.1.1	Instalação das Ferramentas Necessárias	23
4.1.2	Simulação dos Sensores	23
4.2	<i>Hosts</i> , <i>Items</i> , <i>Triggers</i> e <i>Templates</i>	24
4.2.1	<i>Hosts</i>	24
4.2.2	<i>Items</i>	25
4.2.3	<i>Triggers</i>	27
4.2.4	<i>Templates</i>	30
4.2.5	Relação Entre <i>Templates</i> , <i>Hosts</i> , <i>Items</i> e <i>Triggers</i> . . .	31
4.3	Gráficos	33
4.4	<i>Widgets</i>	34
4.5	Análise e Apresentação de Resultados e Estatísticas	35
5	Resultados Obtidos	37
5.1	Apresentação do <i>Dashboard</i>	37
5.2	Apresentação do Mapa Geográfico	38
5.3	Apresentação dos Gráficos	40
5.4	Apresentação de Widgets	40
5.5	Apresentação de Alertas Sobre Situações Anômalas	41
5.5.1	Falha de Conectividade com um Sensor	41
5.5.2	Vagas de Calor	42
5.5.3	Tiros	44
6	Conclusões e Trabalho Futuro	47
6.1	Conclusões	47
6.2	Trabalho Futuro	48
	Referências Bibliográficas	51
A	Estudo dos Níveis Sonoros	55
A.1	Índices Sonoros	55
A.2	Eventos Sonoros	57
A.2.1	Tiros	57
A.2.2	Gritos Humanos	58

B	Instalação do Zabbix	61
B.1	Instalação da Base de Dados MySQL	62
B.2	Instalação do Zabbix	63
C	Simulação dos Sensores	67
D	Utilizando o Docker Desktop	73
D.1	Definição do Docker Desktop	73
D.2	Ambiente de Execução	73
D.2.1	Instalação do Docker Desktop	74
D.2.2	Configuração do Docker Desktop	74

Lista de Figuras

2.1	<i>Libelium Noise Level Sensor</i> - Sensores para Monitorização dos Níveis Sonoros	6
2.2	<i>Libelium Noise Level Sensor</i> - Sistema de <i>Dashboard</i>	7
2.3	<i>Real-Time Noise Monitoring for Smart Cities</i> - Sensores Utilizados	8
2.4	<i>FI-Sonic Plat</i> - Plataforma <i>online</i> de análise e visualização de resultados do sistema <i>FI-Sonic</i>	9
2.5	<i>FI-Sonic Network</i> - Rede de sensores do sistema <i>FI-Sonic</i> . . .	10
3.1	Interação entre os sensores, o sistema de <i>Dashboard</i> e a base de dados	15
4.1	Interação entre os sensores e o Zabbix	22
4.2	Novo <i>host</i>	25
4.3	Novo <i>item</i>	26
4.4	Pré-processamento do valor de um determinado item	27
4.5	Novo <i>trigger</i>	28
4.6	Novo template	31
4.7	Relação entre <i>hosts</i> , <i>items</i> , <i>triggers</i> e <i>templates</i> no Zabbix . .	32
4.8	Processo de criação de um gráfico no Zabbix	33
4.9	Configuração do período de tempo de um gráfico no Zabbix .	34
4.10	Criação de um <i>widget</i> no Zabbix	35
5.1	Apresentação do <i>Dashboard</i>	38
5.2	Apresentação do mapa do <i>Dashboard</i> com dois sensores operacionais	38
5.3	Apresentação do mapa do <i>Dashboard</i> com dois sensores operacionais e um deles indicando um alerta	39

5.4	Apresentação do mapa do <i>Dashboard</i> com um sensor operacional e outro inoperacional	39
5.5	Apresentação de um gráfico no Zabbix	40
5.6	Apresentação de um widget no Zabbix	41
5.7	Apresentação do alerta de falha de conectividade mostrando a respetiva expressão	42
5.8	Apresentação do alerta de falha de conectividade na secção dos Problemas do Zabbix	42
5.9	Demonstração do alerta de vaga de calor	43
5.10	Desencadeamento do alerta de vaga de calor	44
5.11	Criação de um <i>trigger</i> para o caso de um tiro	44
5.12	Desencadeamento de um alerta referente a um tiro	45

Lista de Tabelas

A.1	Níveis sonoros de um tiro	58
A.2	Níveis sonoros de um grito humano	59

Glossário

A-weighted *Ponderado na Curva A.* 56

C-weighted *Ponderado na Curva C.* 44, 56

CSS *Cascading Style Sheets.* 34

HTTP *Hyper Text Transfer Protocol.* 14, 23, 27, 67, 71

ICMP *Internet Control Message Protocol.* 12

Impul *Impulsivity [0.0 - 1.0].* 56, 58, 59

IP *Internet Protocol.* 11, 25, 27, 40

IPMI *Intelligent Platform Management Interface.* 12

ISEL *Instituto Superior de Engenharia de Lisboa.* 5, 8

JSON *JavaScript Object Notation.* 2, 17, 23, 27, 34, 67, 71

LAE *Sound Exposure Level, Slow, A-weighted.* 56, 58, 59

LAeq *Equivalent Continuous Sound Pressure Level.* 6, 27, 55, 58, 59

LAFmax *Maximum sound pressure level, Fast, A-weighted.* 40, 44, 56, 58, 59

LAFmin *Minimum sound pressure level, Fast, A-weighted.* 56, 58, 59

LASmax *Minimum sound pressure level, Slow, A-weighted.* 56, 58, 59

LASmin *Minimum sound pressure level, Slow, A-weighted.* 56, 58, 59

LCPeak *Peak sound pressure level, C-weighted.* 56, 58, 59

LFmax *Maximum sound pressure level, Fast, no weighted.* 44, 56, 58, 59

LFmin *Minimum sound pressure level, Fast, no weighted.* 58, 59

LowFreq *Low frequency sounds [0.0 - 1.0].* 56, 58, 59

PHP *Personal Home Page.* 34

RESTful *Representational State Transfer.* 2

SEL *Sound Exposure Level, Slow, no weighted.* 56, 58, 59

SMS *Short Message Service.* 49

SNMP *Simple Network Management Protocol.* 11–13

Tonal *Tone [0.0 - 1.0].* 57–59

URL *Uniform Resource Locator.* 27

Capítulo 1

Introdução

Nos dias de hoje, existe uma grande necessidade de descobrir novas ferramentas e soluções para ajudar a contribuir para o aumento da segurança pública, nomeadamente, através da deteção e identificação, em tempo real, de eventos sonoros ocorridos nas vias rodoviárias como por exemplo acidentes rodoviários, distúrbios civis, explosões ou fenómenos naturais. Desta forma, poderão ser evitadas situações adicionais de perigo para os cidadãos resultantes da falta de informação sobre eventos anómalos.

Para contribuir para o cumprimento deste objetivo, pretende-se, com a realização deste projeto, implementar um sistema que permita a análise e interpretação, em tempo real, de eventos sonoros captados por vários sensores agregados em uma rede e distribuídos em diferentes vias rodoviárias.

Esta rede de sensores envia periodicamente dados para um sistema de *Dashboard*, que permite a visualização e interpretação desses dados através da elaboração de gráficos e estatísticas.

O foco principal deste projeto consiste na implementação do sistema de *Dashboard* e assume que os sistemas de recolha de sons já estão implementados e recolhem os dados sonoros que o sistema de *Dashboard* utiliza para exercer as suas funcionalidades.

1.1 Enquadramento

O projeto descrito neste relatório encontra-se inserido num contexto mais amplo, onde existe um segundo projeto responsável pela implementação dos sensores. Esse projeto utiliza um conjunto de sensores, incluindo microfo-

nes, sendo responsável por recolher os dados relativos aos sons detetados, efetuar um pré-processamento e disponibilizar os mesmos para consumo, por exemplo, por um sistema *Dashboard*. A informação deste segundo sistema é disponibilizada na forma de um objeto em formato `JSON`.

No contexto da realização do nosso projeto, dado que o sistema de sensores ainda não estava totalmente desenvolvido, o mesmo foi simulado como um serviço `RESTful` implementado na linguagem `Python` e executado numa máquina virtual.

1.2 Contributos do Projeto

Este projeto contribui para a disponibilização de um sistema *Dashboard*, suportado na plataforma `Zabbix`, com a capacidade de interpretar eventos sonoros em tempo real, permitindo o envio de notificações imediatas aos utilizadores registados de quaisquer anomalias ocorridas nos locais onde os sensores estão localizados. Através da disponibilização dessas informações de forma rápida, eficaz e eficiente, contribui-se para o aumento da segurança pública.

O sistema proposto irá fornecer uma visão clara e abrangente dos eventos sonoros que ocorrem nas diferentes vias rodoviárias, facilitando a tomada de decisões por parte de entidades governamentais e auxiliando na implementação de medidas preventivas.

Para além disso, a implementação deste projeto resultou na criação e disponibilização de *widgets* customizados para a plataforma `Zabbix`, através de um tutorial fornecido pela documentação oficial da versão mais recente (6.4) do `Zabbix` [1]. Os ficheiros de configuração necessários para a elaboração de *widgets* na versão mais recente do `Zabbix` estão disponibilizados num repositório `git` criado para este efeito [2].

1.3 Organização do Relatório

Este relatório, para além deste capítulo de Introdução, está organizado do seguinte modo: no capítulo **2 – Trabalho Relacionado**, é realizada uma breve apresentação dos estudos, trabalhos e projetos que foram realizados em fases anteriores e cujo tema está, direta ou indiretamente, relacionado com o do nosso projeto.

No capítulo **3 – Análise de Requisitos**, são descritos os requisitos do sistema a ser implementado, e são abordadas as necessidades identificadas.

No capítulo **4 – Implementação**, é apresentada a proposta para a resolução do problema. Também são descritos os passos que foram seguidos para a correta implementação dessa proposta.

No capítulo **5 – Resultados Obtidos**, são apresentados os resultados obtidos na implementação da proposta descrita.

E, por fim, no capítulo **6 – Conclusões e Trabalho Futuro**, são descritas as conclusões que foram tiradas na implementação do projeto, sendo igualmente apresentadas as linhas de continuação do trabalho que falta para ser realizado numa fase futura por outros grupos de trabalho.

Capítulo 2

Trabalho Relacionado

Neste capítulo são apresentados trabalhos relacionados com o tema tratado no nosso projeto. Na secção **2.1 – Sensores de Recolha de Sons** são apresentados alguns projetos semelhantes àqueles que estão a ser desenvolvidos no contexto do projeto que está a ser desenvolvido pelos nossos colegas, ou seja, que utilizam sensores com funcionalidades semelhantes às dos que são utilizados neste projeto. Na secção **2.2 – Estado Atual de Conhecimento**, é apresentado o estado atual do nosso conhecimento sobre os principais tópicos abordados por este projeto.

2.1 Sensores de Recolha de Sons

As seguintes secções irão apresentar projetos que têm como objetivo contribuir para a segurança pública, combatendo a poluição sonora. Na secção **2.1.1 – *Libelium Noise Level Sensor***, é descrito o projeto *Libelium Noise Level Sensor*, desenvolvido com sensores *Smart Cities PRO*. Na secção **2.1.2 – *Real-Time Noise Monitoring for Smart Cities***, é descrito um projeto desenvolvido pela empresa *Sonitus Systems* e que é implementado nas cidades inteligentes. Na secção **2.1.3 – *FI-Sonic***, é apresentado um projeto desenvolvido no ISEL, que nos serviu de exemplo para a realização do nosso projeto ao longo do semestre.

2.1.1 *Libelium Noise Level Sensor*

Este projeto [3] utiliza sensores que detetam os níveis de ruído instalados nas paredes das casas e fornecem dados que podem ser utilizados para tomar

medidas de controlo de ruído.

Os testes foram realizados usando o parâmetro *Equivalent Continuous Sound Pressure Level* (LAeq), que permite calcular o nível médio de pressão sonora durante um determinado período de tempo.

Os sensores utilizados contêm a mesma funcionalidade dos utilizados no nosso projeto, conseguindo, no entanto, obter dados mais detalhados, como a direção do som e valores processados do ruído, tais como a frequência, o nível de ruído e o nível de pressão sonora. Para além disso, este produto é instalado nas paredes do lado de fora das casas. Já no nosso projeto, os sensores são instalados nas vias rodoviárias, apesar de também ser possível instalar os mesmos em outros locais.

É possível observar uma ilustração dos sensores utilizados neste projeto na Figura 2.1.



Figura 2.1: *Libelium Noise Level Sensor* - Sensores para Monitorização dos Níveis Sonoros

Na Figura 2.2, é apresentado o sistema *Dashboard* que é implementado neste projeto.

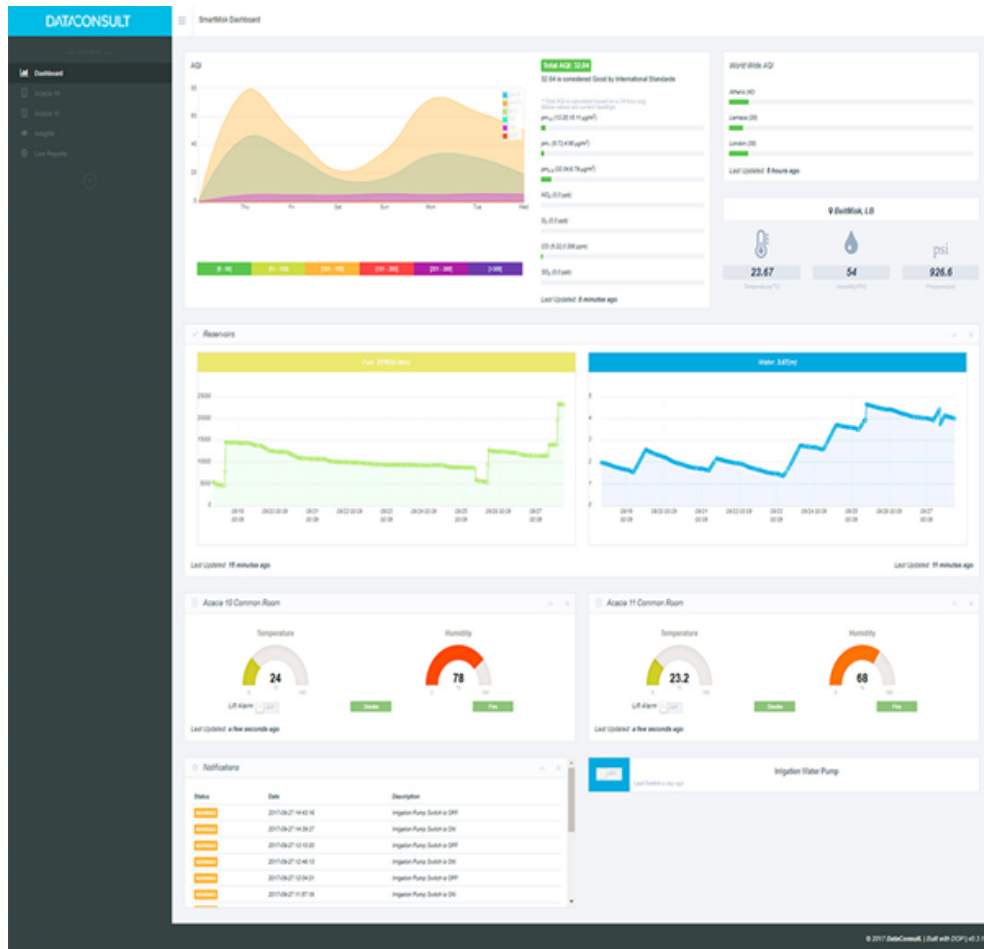


Figura 2.2: *Libelium Noise Level Sensor - Sistema de Dashboard*

2.1.2 *Real-Time Noise Monitoring for Smart Cities*

Este projeto [4] consiste em detetar e monitorizar, em tempo real, o ruído onde os sensores estão instalados, e implementa um sistema de *Dashboard* centralizado para visualizar e analisar os dados relativos ao ruído detetado. Para isso, é utilizado o projeto *Dynamap*, que visa implementar mapas de ruído urbano existentes através da produção de mapas de ruído dinâmicos, utilizando dados agregados de sensores de baixo custo, previamente implementados por outros projetos.

Para além disso, trata-se de um projeto enquadrado no conceito de Cidades Inteligentes (*Smart Cities*), e permite que os cidadãos utilizem uma aplicação nos seus *smartphones* para capturar medições de níveis sonoros, o

que não acontece no nosso projeto.

É possível observar uma ilustração deste projeto na Figura 2.3.



Figura 2.3: *Real-Time Noise Monitoring for Smart Cities* - Sensores Utilizados

2.1.3 *FI-Sonic*

O projeto *FI-Sonic* [5] é um projeto que foi desenvolvido no ISEL e que foi analisado e abordado diversas vezes ao longo do desenvolvimento do nosso projeto, dadas as muitas semelhanças que estes dois projetos possuem.

Este projeto, tal como o apresentado na secção **2.1.2 – *Real-Time Noise Monitoring for Smart Cities***, surge enquadrado no conceito de Cidades Inteligentes (*Smart Cities*), e realiza a monitorização de ruído em contínuo e em tempo real, e a análise de eventos sonoros através do processamento de sinais de áudio captados por uma rede de microfones.

Também apresenta um *Dashboard* onde estão disponibilizados mapas de ruído dinâmicos, que permitem a visualização da evolução dos níveis de ruído ao longo do tempo. A Figura 2.4 ilustra um exemplo de um *Dashboard* implementado neste projeto.



Figura 2.4: *FI-Sonic Plat* - Plataforma *online* de análise e visualização de resultados do sistema *FI-Sonic*

Outro aspecto importante é que, através da utilização de microfones de captação multidirecional, para além de identificar diversos tipos de sons, como por exemplo sons de veículos ligeiros e pesados, buzinas e disparos de armas de fogo, é possível localizar as fontes sonoras. No nosso projeto, os sensores não são capazes de detetar a localização de um determinado evento sonoro. A Figura 2.5 mostra a topologia da rede de sensores utilizada neste projeto, em que os dados capturados pelos mesmos são enviados para um servidor central (*“Central Processing”*) para agregação, visualização e análise desses dados através do uso de um sistema *Dashboard*.

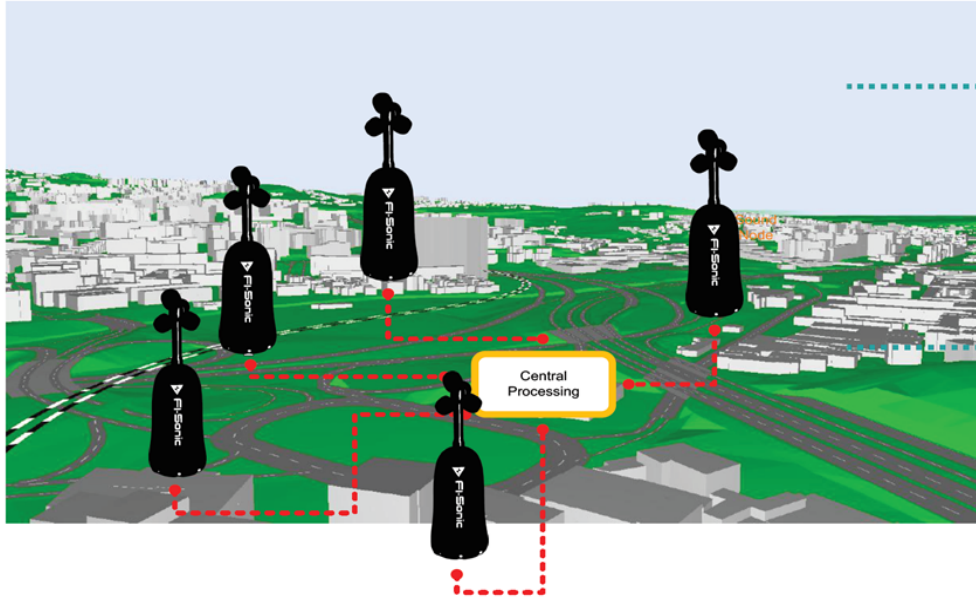


Figura 2.5: *FI-Sonic Network* - Rede de sensores do sistema *FI-Sonic*

Enquanto este projeto realiza apenas a monitorização do ruído e a agregação dos respetivos dados em mapas de ruído dinâmicos, o nosso projeto utiliza um sistema de *Dashboard* concretizado em uma plataforma especializada para gestão e monitorização de rede e aplicações de código aberto, que fornece recursos mais avançados de agregação, visualização e interpretação dos dados dos eventos sonoros. Esta plataforma também é capaz de interagir com diferentes entidades, como por exemplo bases de dados.

2.2 Estado Atual de Conhecimento

Foi importante, numa primeira fase, obter um entendimento sólido sobre o funcionamento das ferramentas de monitorização que foram colocadas em hipótese para a concretização do sistema de *Dashboard* neste projeto: OpenNMS [6] e Zabbix [7].

Ambas as plataformas seriam opções válidas para concretizar o sistema de *Dashboard* a ser implementado. Ambas são aptas para monitorizar e gerir grandes infraestruturas de rede, fornecendo recursos avançados para tal. No entanto, após termos analisado estas duas plataformas [8] [9], considerámos que a plataforma Zabbix seria a mais adequada para ser usada, dado que tem

a vantagem de ser gratuita, ao contrário do **OpenNMS**, e fornece uma interface mais intuitiva, simplificada e personalizável para os seus utilizadores, o que facilita a aprendizagem dos mesmos. Para além disso, a plataforma **Zabbix** é mais flexível e escalável, o que a torna mais adequada para monitorizar grandes infraestruturas de rede. A configuração inicial do **Zabbix** é mais simples comparativamente com a do **OpenNMS**. O **Zabbix**, ao contrário do **OpenNMS**, suporta a monitorização de dispositivos através dos seus endereços IP, o que facilita a comunicação do sistema com os sensores, podendo assim notificar de imediato aos seus utilizadores de que um sensor está inoperacional, quando não conseguir estabelecer conexão com esse sensor. O **Zabbix** utiliza o protocolo **SNMP**, o que facilita a gestão e monitorização de rede.

Durante o desenvolvimento do projeto através da utilização do **Zabbix**, foram adquiridos conhecimentos sobre a configuração e personalização do **Zabbix**, incluindo a criação e configuração de *hosts*, itens, *triggers* e *templates*, bem como a geração de gráficos personalizados. Mais detalhes sobre estes conceitos são apresentados no subcapítulo **4.2 – Hosts, Items, Triggers e Templates**.

Para além disso, foi necessário adquirir conhecimentos sobre o funcionamento dos sensores de captura de som, para que pudéssemos simular o comportamento dos mesmos.

Também foram estudados e determinados os parâmetros relativos a sons típicos que traduzam situações alarmantes. Com base nesse estudo, é possível criar alertas no **Zabbix** para notificar os utilizadores quando esses eventos anómalos ocorrerem, como por exemplo tiros, acidentes rodoviários e explosões. Os resultados obtidos através desse estudo estão especificados no apêndice **A – Estudo dos Níveis Sonoros**.

Outro tópico que estudámos foi sobre algumas bases de dados que poderiam ser utilizadas para interagir, externamente, com o sistema de *Dashboard*. Foram estudadas as bases de dados **MySQL**, **Elasticsearch**, **MongoDB** e **PostgreSQL**, e realizámos uma comparação entre elas [10] [11] [12] [13]. Também foram estudadas as principais diferenças entre modelo de dados relacional e não relacional, e tentámos entender qual destes dois modelos de dados mais ia ao encontro ao que se pedia da base de dados escolhida para interagir com o sistema *Dashboard*. Apesar de estes últimos estudos não terem sido diretamente essenciais para a implementação do trabalho nesta fase

atual, e a ligação a uma base de dados ter ficado para uma fase futura, foi uma parte importante para a compreensão das bases de dados em geral.

Nas seguintes secções, são apresentados alguns conceitos que é importante compreender, pois são a base do nosso projeto. Na secção **2.2.1 – Zabbix**, é apresentada uma definição sobre o que é a plataforma **Zabbix**. Na secção **2.2.2 – Dashboard**, é apresentada uma definição para o termo *Dashboard*. Por fim, na secção **2.2.3 – Sensor**, é apresentada uma definição para um sensor.

2.2.1 Zabbix

O **Zabbix** é uma plataforma amplamente utilizada por empresas e organizações em todo o mundo e que foi utilizada neste projeto para implementar o sistema de *Dashboard*.

Trata-se de uma plataforma de gestão de rede e aplicações de código aberto, desenvolvida como um *software*, que é capaz de efetuar a monitorização de diversos parâmetros de uma rede e também servidores, máquinas virtuais, aplicações, bases de dados, entre outros.

Os principais recursos e funcionalidades do **Zabbix**, que se enquadram no contexto deste projeto, são:

Monitorização abrangente, distribuída e escalável O **Zabbix** é capaz de monitorizar uma grande variedade de dispositivos, tais como servidores, aplicações e serviços de rede. Para isso, a plataforma suporta diversos protocolos de monitorização como ICMP, SNMP, IPMI e agentes nativos. Para além disso, a plataforma é projetada para lidar com ambientes complexos e em escala empresarial, permitindo a monitorização de diversos locais e milhares de dispositivos de forma simultânea. A escalabilidade, no contexto do nosso projeto, irá permitir que, a qualquer momento, seja possível adicionar ou remover um sensor do sistema *Dashboard* de modo rápido e eficiente.

Agregação de dados de diversas fontes Para este trabalho, pretende-se agregar dados relativos ao ruído detetado por vários sensores distribuídos em uma rede. Para isso, o **Zabbix** necessita de agregar dados de desempenho de várias fontes em tempo real. Isso é possível por meio de agentes que são previamente instalados e que são monitorizados através da utilização

de diversos protocolos de monitorização. O protocolo **SNMP**, por exemplo, é utilizado para agregar informações de dispositivos de rede. A monitorização é baseada em *logs*, que são usados para extrair informações relevantes para análise e identificação de problemas. É possível também utilizar expressões regulares para filtrar e extrair dados específicos dos *logs*. Para além disso, é disponibilizada uma grande variedade de integrações e *plugins* que permitem agregar dados de várias fontes e serviços.

Envio de Alertas e Notificações O Zabbix permite definir condições de alerta e notificações personalizadas. Os utilizadores poderão receber alertas de acordo com a configuração feita na plataforma e também com as suas preferências. Quando ocorrem eventos críticos ou violações de métricas definidas, o sistema pode enviar alertas por vários meios tais como e-mail, **SMS** ou **Telegram** para garantir que as equipas relevantes sejam informadas de imediato.

Visualização de dados O Zabbix oferece uma interface gráfica intuitiva para criar painéis personalizados. É possível criar gráficos, mapas, tabelas e indicadores personalizados para visualizar métricas e estatísticas de forma clara e fácil de entender.

Análise e relatórios O Zabbix fornece recursos avançados de análise e relatórios. É possível gerar relatórios automatizados com base em métricas específicas, criar tendências e previsões de desempenho, além de realizar análises detalhadas para solucionar problemas e otimizar a infraestrutura. Estes relatórios contêm diversas informações como por exemplo o registo de desempenho de um determinado componente, e permitem a análise dos dados previamente captados e agregados pela plataforma. Os mesmos podem ser gerados através da integração do Zabbix com ferramentas de análise e relatórios.

2.2.2 *Dashboard*

Um sistema de *Dashboard* consiste em uma ferramenta que permite visualizar e monitorizar dados e métricas importantes de dados vindos de várias origens, de forma centralizada.

O mesmo permite a representação visual de informações relevantes organizadas em painéis ou telas fornecendo uma visão rápida e fácil dos dados, permitindo também fazer pesquisas sobre os dados guardados em memória.

2.2.3 Sensor

Um sensor, no contexto deste trabalho, é um dispositivo composto por vários tipos de estações de monitorização, não necessariamente todos relacionados ao som, mas também relacionados com outros parâmetros, como por exemplo a temperatura e a velocidade do vento.

Todos eles estão conectados a uma rede de sensores e utilizam o protocolo HTTP para comunicação e processamento de pedidos feitos ao sistema *Dashboard*. Os dados enviados como resposta a um pedido HTTP consistem na agregação de todos os valores detectados pelos sensores durante um determinado período, bem como os dados processados internamente sobre esses dados.

Capítulo 3

Análise de Requisitos

Neste capítulo vamos realizar uma análise dos requisitos necessários para que possamos ter condições para implementar uma proposta válida que consiga resolver, com sucesso, o problema inicialmente levantado, atender às necessidades dos utilizadores e aos objetivos deste projeto.

A Figura 3.1 apresenta o cenário que pretendemos alcançar na realização deste projeto.

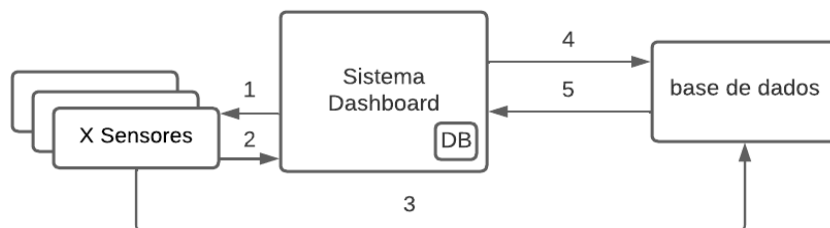


Figura 3.1: Interação entre os sensores, o sistema de *Dashboard* e a base de dados

De acordo com o que ilustra a Figura 3.1, apresentada acima, o sistema de *Dashboard* faz, periodicamente, pedidos aos sensores para enviarem dados (1), os sensores, por sua vez, respondem enviando os mesmos que ainda não tenham sido enviados (2). Os sensores também enviam dados periodicamente para a base de dados para que os mesmos sejam armazenados de forma mais segura e eficiente (3). O sistema de *Dashboard* faz pedidos à base de dados para obter os dados nela armazenados, para poder realizar a sua análise (4 e 5).

No nosso trabalho, foi implementado o sistema de *Dashboard*, as comunicações com os sensores (1 e 2) e a apresentação dos dados através da utilização de *widgets*.

Este capítulo começa com a secção **3.1 – Casos de Utilização**, são apresentados vários casos de utilização deste projeto, ou seja, como será utilizado este projeto. Por fim, na secção **3.2 – Requisitos**, são apresentados os requisitos, funcionais e não funcionais, necessários para o alcance dos objetivos deste projeto.

3.1 Casos de Utilização

Nesta secção, apresentam-se alguns dos vários cenários de uso deste projeto e das interações entre os utilizadores registados na plataforma e o sistema de *Dashboard*.

Criação de uma conta na plataforma É possível criar uma nova conta na plataforma correspondente ao sistema de *Dashboard*, criando um nome de utilizador e uma palavra-passe para tal. Quando isso for feito, será automaticamente realizado *login* na plataforma.

Efetuar *login* e *logout* Qualquer pessoa, para aceder à plataforma, deve estar registada na mesma e entrar com uma conta previamente criada através de *login*. A mesma poderá sair da plataforma através de *logout*.

Gestão de *widgets* Estando o utilizador registado na plataforma, poderá criar os seus próprios *widgets*, para ver os valores dos parâmetros de uma forma customizada. Também pode, a qualquer momento, remover *widgets* previamente criados.

Personalização dos dados apresentados Os utilizadores também poderão personalizar os *widgets* previamente criados do seu sistema de *Dashboard* para apresentar os conjuntos de dados de acordo com as suas necessidades e preferências.

Gestão da rede de sensores São oferecidos recursos aos utilizadores para adicionar ou remover sensores à plataforma correspondente ao sistema de *Dashboard*.

Agregação e análise de eventos sonoros em tempo real Os utilizadores também têm recursos para criar *hosts* na plataforma que permitam a agregação dos vários sons detetados pelos mesmos, e podem criar alertas (*triggers*) que serão gerados na sequência da análise dos mesmos.

3.2 Requisitos

Neste capítulo são apresentados os requisitos, tanto do sistema de *Dashboard* como os sensores, que são aspetos necessários para o correto funcionamento do mesmo. Na secção **3.2.1 – Requisitos Funcionais**, são apresentados os requisitos funcionais e na secção **3.2.2 – Requisitos Não Funcionais**, são apresentados os requisitos não funcionais.

3.2.1 Requisitos Funcionais

Os requisitos funcionais são descrições precisas e detalhadas das funcionalidades que o sistema deve oferecer. Os mesmos definem as ações que o sistema deve ser capaz de executar e como o mesmo deve responder a diferentes pedidos dos utilizadores. De seguida são apresentados os requisitos funcionais dos sensores e do sistema de *Dashboard*.

Sensores

Envio de dados processados em formato JSON Os sensores devem ser capazes de enviar, periodicamente, dados correspondentes aos eventos sonoros para o sistema *Dashboard*, através de objetos em formato JSON, para que o sistema possa utilizar esses dados para exercer as suas funcionalidades. O conteúdo de cada um dos objetos enviados é correspondente ao objeto “data” apresentado, no apêndice **C – Simulação dos Sensores**, mais especificamente na listagem de código **1**.

Sistema de *Dashboard*

Apresentação de gráficos das informações recolhidas pelos sensores

O sistema deve ser capaz de exibir visualmente os dados correspondentes aos eventos sonoros captados, em tempo real, pelos sensores, em forma de gráficos. Os gráficos apresentados podem ser, por exemplo, gráficos temporais, espectros, apresentação de eventos anómalos e gráficos customizados, como por exemplo *widgets*.

Armazenamento dos dados em bases de dados O sistema deve ser capaz de armazenar os dados dos eventos sonoros em uma base de dados, seja interna ou externa. Os sensores também poderão enviar os dados diretamente para a base de dados em vez de enviar para o sistema.

Identificação do tipo de evento sonoro O sistema deve ser capaz de identificar, em tempo real, certos eventos sonoros que traduzam situações alarmantes, como tiros, gritos humanos e acidentes rodoviários. O sistema deve apresentar, no sistema, qual tipo de evento sonoro anômalo ocorreu.

Apresentação da localização geográfica dos sensores Para além de identificar eventos sonoros em tempo real, o sistema deve apresentar, em um mapa, a localização geográfica dos sensores que estão a ser monitorizados pelo mesmo, para que os utilizadores possam, de imediato, ser informados não apenas de eventos anómalos, mas também do local exato da ocorrência dos mesmos.

3.2.2 Requisitos Não Funcionais

Os requisitos não funcionais são características do sistema que não se relacionam diretamente com as funcionalidades específicas, mas sim com propriedades gerais que este tem de possuir para exercer corretamente as suas funcionalidades. Os mesmos descrevem as restrições e os critérios de qualidade que o sistema deve atender. De seguida são apresentados os requisitos não funcionais dos sensores e do sistema de *Dashboard*.

Sensores

Confiabilidade O sistema deve ser confiável, garantindo a disponibilidade e integridade dos dados sonoros, ou seja, garantindo que as informações por ele fornecidas são credíveis.

Sistema de *Dashboard*

Segurança O sistema deve garantir a segurança dos dados, utilizando autenticação e controlo de acesso adequados.

Desempenho O sistema deve ter um desempenho adequado, sendo capaz de processar e exibir os gráficos das informações sonoras de forma rápida e eficiente.

Escalabilidade O sistema deve ser escalável para poder lidar com uma vasta quantidade de sensores, e também ser possível adicionar e remover sensores do sistema a qualquer momento.

Utilidade O sistema deve ser intuitivo e o mais prático possível para sua utilização, proporcionando uma experiência agradável para os utilizadores.

Capítulo 4

Implementação

A proposta apresentada para resolver este problema consiste na implementação de um sistema *Dashboard* que interage com os sensores e com uma base de dados externa. Para o auxílio da nossa implementação e da realização destes passos, recorreremos à documentação oficial da versão mais recente (6.4) da plataforma **Zabbix** [14].

Para resolver o problema levantado e para implementar a proposta descrita no capítulo anterior, foram seguidos uma série de passos sequenciais que serão descritos em cada uma das secções abaixo. Na secção **4.1 – Ambiente de Execução**, é descrito como, numa fase inicial, foi preparado e montado o ambiente base para a realização do projeto. Na secção **4.2 – Hosts, Items, Triggers e Templates**, é explicado em que consistem os *hosts*, *items*, *triggers* e *templates*, como estes quatro componentes do **Zabbix** funcionam e como interagem entre si. Na secção **4.3 – Gráficos**, é explicado como, numa fase seguinte, foram criados gráficos no **Zabbix** capazes de mostrar, em tempo real, a variação de um ou vários parâmetros dos níveis sonoros detetados em tempo real. Na secção **4.4 – Widgets**, é explicado como foram criados *widgets*, que são um complemento para a elaboração de gráficos para mostrar o valor exato de um determinado parâmetro dos níveis sonoros em tempo real. Por fim, na secção **4.5 – Análise e Apresentação de Resultados e Estatísticas**, é mostrado como foi tirado partido dos estudos feitos anteriormente e da elaboração dos gráficos, para apresentar resultados e estatísticas no **Zabbix**.

4.1 Ambiente de Execução

Para iniciar esta implementação, foi preparado e montado o ambiente utilizado para a realização deste projeto.

Numa fase inicial, havia-nos sido proposta a utilização da aplicação **Docker Desktop** [15] para montar o ambiente de execução para a concretização do sistema de *Dashboard*.

No entanto, devido a incompatibilidades entre o ambiente **Docker** e o ambiente **VirtualBox** (necessário no contexto de outra Unidade Curricular que estava a ser frequentada neste semestre), foi utilizada a plataforma de virtualização **VirtualBox** para este fim [16]. Esta plataforma consiste em um *software* de virtualização de código aberto que fornece várias funcionalidades, sendo que uma dessas funcionalidades é permitir a criação e execução de máquinas virtuais de diversos sistemas operativos.

Na secção **4.1.1 – Instalação das Ferramentas Necessárias**, serão descritos os passos que foram realizados para instalar as ferramentas necessárias para o correto funcionamento do sistema de *Dashboard*. Na secção **4.1.2 – Simulação dos Sensores**, será descrito como foram simulados os dois sensores nas máquinas virtuais da plataforma **VirtualBox**.

A Figura 4.1, apresenta um diagrama que ilustra a interação entre as máquinas virtuais que simulam o comportamento dos sensores e a máquina virtual aonde está instalada a plataforma **Zabbix**.

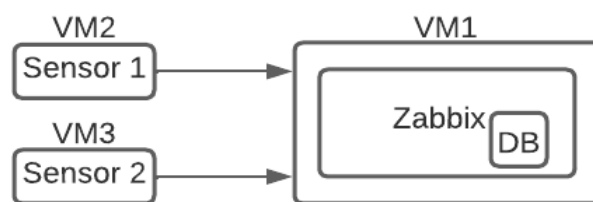


Figura 4.1: Interação entre os sensores e o **Zabbix**

No apêndice **D – Utilizando o Docker Desktop**, estão ilustrados os passos que seriam necessários para montar o ambiente de execução utilizando a aplicação **Docker Desktop**, em vez da plataforma **VirtualBox**, para implementar corretamente o sistema de *Dashboard* e os sensores.

4.1.1 Instalação das Ferramentas Necessárias

Numa primeira fase, foram realizadas as configurações necessárias para montar e preparar o ambiente de desenvolvimento e implementação do sistema *Dashboard*. Isso implica a instalação da plataforma **Zabbix**, juntamente com todas as suas dependências internas.

Esta plataforma depende internamente da base de dados **MySQL** para exercer as suas funcionalidades, e para isso foi necessário, primeiro, instalar os ficheiros de instalação dessa mesma base de dados na máquina virtual, e só depois disso foi possível instalar o **Zabbix** de forma correta e segura.

Mais detalhes sobre os passos que foram seguidos para este fim estão descritos no apêndice **B – Instalação do Zabbix**.

4.1.2 Simulação dos Sensores

Nesta fase do trabalho, os sensores ainda não estavam implementados pelo grupo responsável pela realização do outro projeto que está enquadrado no mesmo trabalho que o nosso projeto.

Por isso, para que fosse possível simular o comportamento dos sensores a enviar os dados referentes aos níveis sonoros para o **Zabbix**, foram criadas mais duas máquinas virtuais na plataforma **VirtualBox**, cada uma a simular o comportamento de um sensor, da mesma forma que foi criada uma máquina para a instalação do **Zabbix**.

Cada uma destas duas máquinas virtuais foi configurada para criar objetos em formato **JSON** que contêm informação sobre os níveis sonoros, e assim que as mesmas forem iniciadas, estarem aptas para responder aos pedidos realizados pelo **Zabbix**, para enviar desses dados. Para isso, foi adicionado a cada uma das máquinas um ficheiro **Python** que fosse capaz de enviar dados, em formato de objetos **JSON**, para o **Zabbix**, através de pedidos **HTTP** feitos pelo **Zabbix**.

No apêndice **C – Simulação dos Sensores**, estão especificados mais detalhes sobre como foram configuradas as máquinas virtuais para simularem o comportamento dos sensores desta forma.

4.2 *Hosts, Items, Triggers e Templates*

Para que se possa compreender corretamente como funciona esta proposta de implementação do nosso projeto e como chegámos aos resultados pretendidos, é necessário obter uma correta compreensão sobre os principais componentes da plataforma Zabbix, isto é, os *hosts*, explicados na secção 4.2.1 – ***Hosts***, os *items*, explicados na secção 4.2.2 – ***Items***, os *triggers*, explicados na secção 4.2.3 – ***Triggers*** e os *templates*, explicados na secção 4.2.4 – ***Templates***.

É necessário compreender como estes componentes funcionam, em que consistem e como os mesmos interagem entre si (4.2.5 – ***Relação Entre Templates, Hosts, Items e Triggers***).

4.2.1 *Hosts*

Na plataforma Zabbix, um *host* representa um dispositivo, servidor, máquina virtual ou aplicação a ser monitorizada. No contexto deste projeto, um *host* representa um sensor simulado em uma máquina virtual.

Cada *host* possui um ou vários itens associados ao mesmo, que representam métricas e parâmetros específicos que se pretende monitorar.

Para criar um *host*, no menu **Data Collection**, clicar em *Hosts*, e selecionar, no canto superior direito, a opção **Create Host**. Especificar o nome do *host* a ser criado e o grupo de *hosts* ao qual este pertence. A Figura 4.2 ilustra o processo de criação de um *host*.

New host ? ✕

Host IPMI Tags Macros Inventory Encryption Value mapping

* Host name

Visible name

Templates

* Host groups

Interfaces	Type	IP address	DNS name	Connect to	Port	Default
^ SNMP		<input type="text" value="127.0.0.1"/>	<input type="text"/>	<input checked="" type="radio"/> IP <input type="radio"/> DNS	<input type="text" value="161"/>	<input checked="" type="radio"/> Remove

* SNMP version

* SNMP community

Max repetition count

☒ Use combined requests

Add

Description

Monitored by proxy

Enabled ☒

Figura 4.2: Novo *host*

Como é ilustrado na Figura 4.2, para este caso, como um *host* representa um sensor, será necessário especificar um endereço IP para a interface à qual o *host* pertence, pois o Zabbix irá comunicar com a máquina virtual que simula o sensor através desse endereço IP. O tipo de interface do *host* pode ser tanto *Agent* como *SNMP*, dependendo dos casos.

Também poderão ser adicionados outros parâmetros opcionais como o conjunto de *templates* aos quais o mesmo pertence, o nome visível (*Visible Name*) e a descrição do *host*.

4.2.2 Items

Os items representam as métricas monitorizadas em um *host*. Podem ser itens simples, como valores numéricos ou textuais, ou itens mais complexos, como monitorização de logs, disponibilidade de serviços, latência de rede, entre outros. Cada item possui uma chave de item única, que é usada para identificá-lo no Zabbix. Esta chave poderá ser utilizada, por exemplo, para criar uma expressão de um *trigger* para desencadear um alerta, o que será melhor explicado na secção 4.2.3 – *Triggers*.

Para criar um item, é necessário seleccionar um determinado *host* e sele-

cionar a opção *Create item*. Cada item tem associado um nome (*Name*), que é único, e uma chave (*Key*) que será usada para referenciar o item. Também é possível atualizar o período de atualização do valor do item (pode ser, por exemplo, alterado para 10 segundos). A Figura 4.3 ilustra o processo de criação de um item.

The screenshot shows the 'Create item' form in Zabbix. The form is titled 'Item' and has tabs for 'Tags' and 'Preprocessing'. The fields are as follows:

- Name:** Item 1
- Type:** Zabbix agent
- Key:** icmpping
- Type of information:** Numeric (unsigned)
- Host interface:** 172.16.16.5:10050
- Units:**
- Update interval:** 10s
- Custom intervals:**

Type	Interval	Period	Action
Flexible	Scheduling	50s	1-7,00:00-24:00
- History storage period:** Do not keep history
- Trend storage period:** Do not keep trends
- Value mapping:** type here to search
- Populates host inventory field:** -None-
- Description:**
- Enabled:** ☒
- Buttons:** Add, Test, Cancel

Figura 4.3: Novo *item*

Como a Figura 4.3 ilustra, para criar um item é necessário especificar os seus parâmetros mais importantes, que são o seu nome (*Name*), o tipo do item (*Type*), a sua chave (*Key*), que é única e será utilizada para referenciá-lo na implementação de outros componentes do Zabbix e o intervalo de atualização do seu valor atual na plataforma (*Update interval*). Os parâmetros *History storage period* e *Trend storage period*, apesar de obrigatórios, não são muito relevantes, por isso mantivemos ambos como *Storage period*.

Para o contexto desta implementação, os items criados foram configura-

dos com o tipo “*HTTP Agent*”. Isto significa que o seu valor será obtido através de um pedido HTTP realizado pela plataforma através do comando `curl`, usando o URL especificado na criação do item (“URL”), que no exemplo ilustrado na Figura 4.3 é `http://{HOST.IP}:4002/GeradorJson`, sendo que “HOST.IP” é uma *tag* que corresponde ao endereço IP do *host* ao qual o item está referenciado.

Para processar o valor de um determinado item, é necessário aceder ao conteúdo da propriedade do objeto que o “sensor” retorna como resposta ao pedido HTTP feito. Como o objeto tem o formato JSON, é necessário especificar essa propriedade através de notação de objetos, de forma a pré-processar o seu valor. Para isso, é necessário, em cada item, aceder ao menu **Tags -> Preprocessing** e especificar a propriedade da qual se pretende extrair o seu valor atual, tal como ilustrado na Figura 4.4.

Preprocessing steps	Name	Parameters
1:	JSONPath	\$.body.DataRecord.noise_levels.LAeq.first()

Add

Type of information: Numeric (float)

Update Clone Test Delete Cancel

Figura 4.4: Pré-processamento do valor de um determinado item

No exemplo da Figura 4.4, é retornado o primeiro valor do *array* correspondente ao *LAeq*, que faz parte do objeto enviado pelo sensor e cujo conteúdo está apresentado no apêndice **C – Simulação dos Sensores**, mais especificamente na listagem de código **1**.

4.2.3 *Triggers*

Os *triggers* são condições ou regras definidas para desencadeamento de um alerta ou notificação quando ocorrem eventos que traduzem uma situação de alarme, ou violações de métricas específicas. Os *triggers* são baseados nos valores dos itens monitorizados. Para que se possa implementar um *trigger* corretamente, é necessário definir uma expressão. Por exemplo, um *trigger* pode ser configurado para desencadear um alerta quando a carga da CPU de um servidor excede um determinado limite. A Figura 4.5 ilustra o processo

de criação de um *trigger*.

The screenshot shows the Zabbix web interface for creating a new trigger. The breadcrumb navigation at the top reads: [All hosts](#) / [Sensor1](#) / **Enabled** / [ZBX](#) / [SNMP](#) / [Items](#) 6 / **Triggers** 4 / [Graphs](#) / [Discovery rules](#) / [Web scenarios](#). Below this, there are tabs for **Trigger**, [Tags](#), and [Dependencies](#). The main form contains the following fields and options:

- * Name:** Trigger 1
- Event name:** Trigger 1
- Operational data:** (empty text area)
- Severity:** Not classified, Information, Warning, Average, **High**, Disaster
- * Expression:** `last(/Sensor1/gettemp)>0` (with an **Add** button to the right)
- Expression constructor:** (empty text area)
- OK event generation:** Expression, Recovery expression, None
- PROBLEM event generation mode:** Single, Multiple
- OK event closes:** All problems, All problems if tag values match
- Allow manual close:** ☐
- Menu entry name:** Trigger URL
- Menu entry URL:** (empty text area)
- Description:** New Trigger To Be Created
- Enabled:** ☒
- Buttons:** Add, Cancel

Figura 4.5: Novo *trigger*

No contexto da realização deste projeto, a criação de alertas é essencial, dado que é necessário notificar aos utilizadores registados de que ocorreu alguma situação anormal, neste caso nas localidades onde os sensores estão instalados. Neste caso, pretende-se gerar um alerta sempre que os valores dos parâmetros dos níveis sonoros detetados em tempo real, forem semelhantes ou próximos dos valores dos parâmetros dos níveis sonoros de um som que traduza uma situação alarmante, que foram previamente determinados através do estudo realizado dos níveis sonoros desses sons.

Para isso, no Zabbix basta seleccionar um dos **hosts**, e clicar na opção **Create trigger** no canto superior direito. Depois, será possível atribuir ao *trigger* a ser criado um nome, o nome do evento (opcional), e uma expressão.

Níveis de severidade

Os alertas no Zabbix devem ter um nível de severidade definido. Tal como a Figura 4.5 ilustra, as opções para nível de severidade são:

Not classified Significa que o alerta não apresenta nenhuma informação relevante nem que apresente nenhuma situação preocupante.

Information É um alerta que serve apenas para informação que possa ser relevante para um utilizador, mas não apresente necessariamente nenhuma situação preocupante.

Warning É um alerta que apresenta uma informação que representa uma situação potencialmente perigosa, de grau médio.

Average É um alerta que apresenta uma informação que representa uma situação potencialmente perigosa, de grau alto.

High É um alerta que apresenta uma informação que representa uma situação extremamente perigosa, de grau muito alto.

Disaster É um alerta que apresenta uma informação que representa uma situação potencialmente desastrosa, como por exemplo a inoperacionalidade de um dos seus componentes.

Expressões

A expressão de um *trigger* define a condição de desencadeamento de um determinado alerta, ou seja, define em que condições o *trigger* deve ser acionado. A expressão envolve vários parâmetros e operações lógicas para determinar a lógica do *trigger*.

Um exemplo de expressão de trigger no Zabbix é a seguinte:

```
{<hostname>:<item.key>.last()} < <valor>
```

Nesta expressão, **hostname** representa o nome do host monitorizado, **item.key** é a chave do item que será verificado e **valor** é o valor de referência que deve ser comparado. A função **last()** retorna o valor mais

recente registrado para o item. Neste caso, o alerta é desencadeado se o valor mais recente registrado para o item for menor do que o valor registrado na variável `valor`.

Por exemplo, para criar um *trigger* que seja acionado quando a temperatura de um servidor ultrapassar 80 graus Celsius, a expressão seria:

```
{ Server : system . temperature . last ( ) } > 80
```

As expressões de *triggers* no **Zabbix** são altamente flexíveis e permitem combinações complexas de métricas, operadores e funções para atender às necessidades específicas de monitorização. As mesmas desempenham um papel fundamental no desencadeamento de alertas e, no contexto deste projeto, na identificação de certos eventos sonoros em tempo real.

Para além da função `last()`, que retorna o valor do último elemento de um *array*, existem outras funções que são amplamente utilizadas em expressões de *triggers* no **Zabbix**:

avg() Calcula a média dos elementos de um determinado *array*.

sum() Calcula a soma dos elementos de um determinado *array*.

first() Retorna o valor do primeiro elemento de um *array*.

max() Retorna o elemento com o maior valor em um *array*.

min() Retorna o elemento com o menor valor em um *array*.

Mais expressões se encontram disponibilizadas na documentação oficial do **Zabbix** [17].

4.2.4 Templates

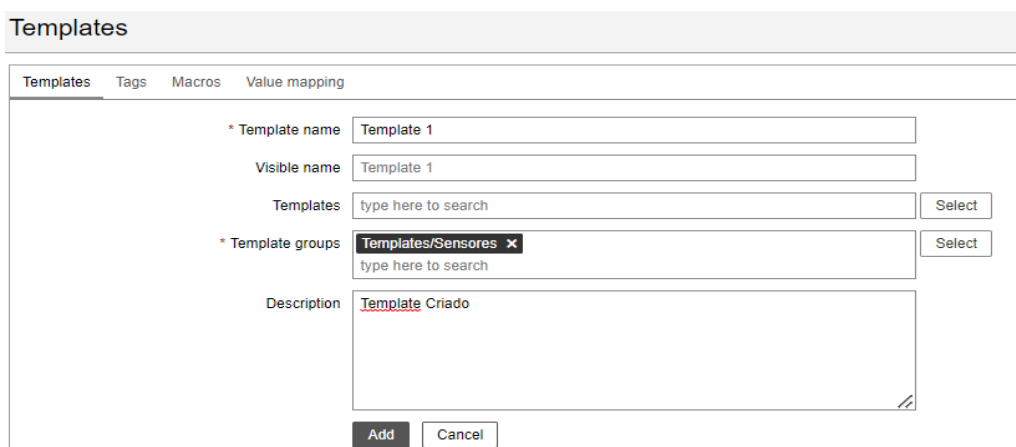
Os *templates* são modelos predefinidos que contêm configurações reutilizáveis para *hosts*, itens, *triggers* e outras entidades do **Zabbix**. Através da utilização dos mesmos, é possível criar configurações padronizadas e aplicá-las a vários *hosts* de forma consistente.

Por exemplo, um *template* de servidor web pode conter itens e triggers para monitorizar métricas específicas de um servidor web, como tempo de resposta das páginas, status dos serviços, entre outros.

Criar um **template** é uma prática bastante eficiente na plataforma Zabbix para lidar com vários *hosts* que tenham *items*, *triggers* ou outros parâmetros em comum, poupando-se tempo e recursos, pois deixa de haver a necessidade de ter sempre de criar os mesmos *items* e alertas para cada *host*.

Ao criar um **template**, todos os seus *items*, alertas, gráficos, entre outros, estarão automaticamente associados a todos os *hosts* que pertencerem a esse **template**. Por isso, para adicionar mais um sensor ao sistema, basta adicionar mais um *host* a esse **template**.

A Figura 4.6 ilustra o processo de criação de um *template*.



The screenshot shows the 'Templates' section of the Zabbix web interface. It features a tabbed interface with 'Templates', 'Tags', 'Macros', and 'Value mapping'. The 'Templates' tab is active. The form includes the following fields and controls:

- * Template name:** A text input field containing 'Template 1'.
- Visible name:** A text input field containing 'Template 1'.
- Templates:** A search input field with the placeholder 'type here to search' and a 'Select' button.
- * Template groups:** A dropdown menu showing 'Templates/Sensores' with a close icon (X) and a search input field with the placeholder 'type here to search', accompanied by a 'Select' button.
- Description:** A large text area containing the text 'Template Criado'.
- Buttons:** 'Add' and 'Cancel' buttons at the bottom.

Figura 4.6: Novo **template**

4.2.5 Relação Entre *Templates*, *Hosts*, *Items* e *Triggers*

A Figura 4.7 descreve a relação entre os quatro componentes apresentados nas secções anteriores.

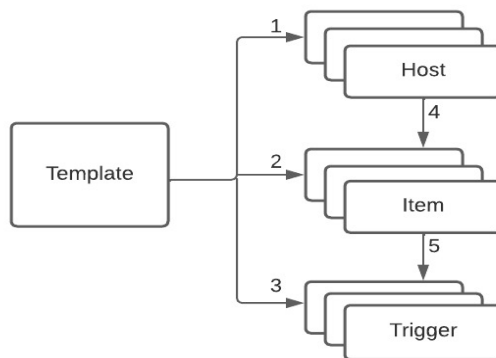


Figura 4.7: Relação entre *hosts*, *items*, *triggers* e *templates* no Zabbix

Tal como se pode observar, um *template* pode ser aplicado a um ou vários *hosts* (1) para fornecer uma configuração pré-definida, automatizada e simplificada para os mesmos, isto porque basta adicionar um *host* ao *template* para que as configurações atuais do *template*, nomeadamente os seus *items* e *triggers* (4 e 5), sejam automaticamente aplicadas a esse *host*.

Para além de ser aplicado a um ou vários *hosts*, um *template* também pode ser aplicado a um ou vários *items* (2), fornecendo também uma configuração simplificada a esse item através da aplicação automática das suas configurações atuais, nomeadamente dos *triggers* pertencentes a esse item. Também pode ser aplicado isoladamente a um ou vários *triggers* (3).

Um *host* possui vários *items* (4), cada item agrega dados específicos do *host* ao qual pertence e pode possuir vários *triggers* (5). Os *triggers* são configurados para gerar alertas na sequência da análise dos dados agregados pelos *items*, feita através de expressões personalizadas.

Esta estrutura de relacionamento entre estes quatro componentes permite ao Zabbix agregar, armazenar, avaliar e interpretar eventos com base nas configurações definidas para cada um dos *hosts* existentes. A flexibilidade e a escalabilidade do Zabbix são ampliadas por meio da utilização de *templates*, que simplificam a configuração e a manutenção de múltiplos *hosts*.

4.3 Gráficos

Após ter o Zabbix e a rede de sensores a funcionarem através das máquinas virtuais, é necessário configurar o Zabbix de forma que possa mostrar em gráficos os dados enviados pelas máquinas. A Figura 4.8 ilustra o processo de criação de um gráfico no Zabbix.



Figura 4.8: Processo de criação de um gráfico no Zabbix

Para elaborar um gráfico conforme ilustrado na Figura 4.8, é necessário referenciar um *host* e um item associado a esse *host*, para que seja possível efetuar a monitorização, em tempo real, do valor desse item.

É necessário também configurar o tempo de atualização do valor na opção *Refresh Interval* para o tempo desejado.

A Figura 4.9 ilustra os passos para outra configuração necessária para a correta elaboração dos gráficos, que consiste no ajuste do período de tempo de apresentação dos gráficos.

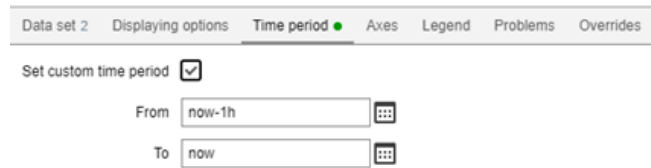


Figura 4.9: Configuração do período de tempo de um gráfico no Zabbix

Tal como a Figura 4.9 mostra, esta configuração consiste em personalizar o tempo desde o qual são apresentados os valores, ou seja, se pretendemos, por exemplo, mostrar a variação do valor de um item nos últimos 5 minutos, nas últimas 24 horas, nos últimos 7 dias ou nos últimos 6 meses. Para isso, clica-se na secção “Time period”, ativa-se a opção “Set custom time period” e configura-se o período de visualização de dados desejados, desde o tempo especificado em “From” até o tempo especificado em “To”.

Para a realização deste passo, seguimos os passos descritos na documentação oficial do Zabbix [7]. Estes passos serão descritos nas próximas secções.

4.4 *Widgets*

No contexto desta parte da implementação desta proposta, um *widget* é uma ferramenta customizada que permite a exibição de informações como valores de parâmetros específicos, de forma visualmente atraente e personalizada.

A criação desta ferramenta implica o desenvolvimento e utilização de um conjunto de ficheiros de código aberto nas linguagens de programação Javascript, CSS, PHP e JSON.

Para esta fase criámos apenas um tipo de *widget*, apresentado no próximo capítulo, mas numa fase futura podem ser alteradas as configurações de forma que se possa ter vários tipos de *widgets*.

Este passo foi essencial, numa fase inicial, para adquirir experiência a trabalhar com a plataforma Zabbix, mas também é uma prática útil para apresentar, de forma customizada, os valores dos itens recolhidos pela plataforma Zabbix, em tempo real, sendo um bom complemento à elaboração de gráficos e estatísticas. A Figura 4.10 ilustra o processo de utilização de um *widget* previamente criado.

Add widget ? ×

Type: Gauge chart Show header: ☒

Name: Widget Example

Refresh interval: Default (1 minute)

* Item: Sensor1: Impulsividade [0.0 - 1.0] × Select

Advanced configuration: ☒

Color:

* Min: 0

* Max: 70

Units: Auto value

Description:

Add Cancel

Figura 4.10: Criação de um *widget* no Zabbix

4.5 Análise e Apresentação de Resultados e Estatísticas

Após a implementação dos gráficos e a determinação dos parâmetros relativos a sons considerados anómalos através do estudo feito sobre os níveis sonoros que indiquem situações alarmantes, os mesmos serão utilizados para obtenção de resultados. Poderemos obter os mesmos com base em parâmetros como, por exemplo, a temperatura detetada no local.

Para além disso, também poderemos obter resultados mais amplos, como por exemplo a média da temperatura daquele local nas últimas 24 horas, ou o nível de ruído nas últimas 12 horas. E utilizando esses resultados, é necessário efetuar uma análise dos mesmos para que possamos constatar se naquele local onde o sensor está instalado ocorreu alguma situação alarmante ou não.

Para isso, existe a necessidade de criar alertas para poder notificar aos utilizadores que de facto existe uma anomalia, e para isso teremos de determinar os parâmetros relativos a sons típicos que traduzem situações alarmantes, o que será especificado na próxima etapa.

Exemplo: Se, por exemplo, a temperatura for superior a 50 graus, pode ser indicado um alerta de vaga de calor, ou se for inferior a 20 graus negativos, um alerta de vaga de frio.

Capítulo 5

Resultados Obtidos

Neste capítulo, são apresentados os resultados que foram obtidos com a realização do projeto, para uma melhor compreensão da nossa implementação. Para além disso, serão apresentados e analisados os resultados que foram obtidos dos estudos que realizámos ao longo da realização deste projeto.

Primeiro, na secção **5.1 – Apresentação do *Dashboard***, será ilustrado o sistema de *Dashboard*. Na secção **5.2 – Apresentação do Mapa Geográfico**, serão apresentados três exemplos de apresentação do mapa geográfico apresentado no sistema de *Dashboard*. Na secção **5.3 – Apresentação dos Gráficos**, serão apresentados alguns dos gráficos que foram elaborados para cada um dos parâmetros dos níveis sonoros detetados pelos sensores. Na secção **5.4 – Apresentação de Widgets**, será ilustrado um exemplo de um widget elaborado neste trabalho. Por fim, na secção **5.5 – Apresentação de Alertas Sobre Situações Anómalas**, serão apresentados os resultados da criação de alertas no Zabbix para indicar situações anormais.

5.1 Apresentação do *Dashboard*

A Figura 5.1 ilustra o *Dashboard* implementado, onde é possível observar um mapa com a localização dos sensores, gráficos com os valores dos parâmetros dos níveis sonoros e um resumo dos problemas ocorridos em cada um dos sensores.

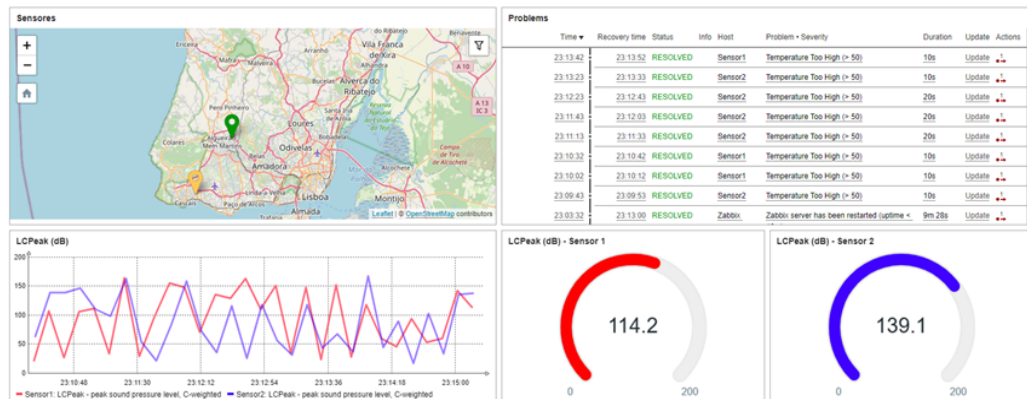


Figura 5.1: Apresentação do *Dashboard*

É possível observar o estado do sensor através de um código de cores, onde a cor verde indica ausência de problemas, a cor amarela indica uma situação anómala e a cor vermelha indica que o sensor está inoperacional. Nas seguintes secções são descritos cada um dos elementos que constituem o *Dashboard*.

5.2 Apresentação do Mapa Geográfico

No primeiro caso, ilustrado na Figura 5.2, os dois sensores estão operacionais, sinalizando que está tudo a correr dentro da normalidade e que não existe nenhuma anomalia.

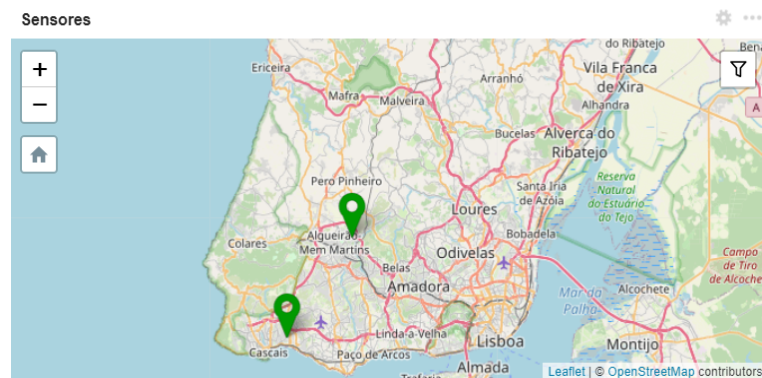


Figura 5.2: Apresentação do mapa do *Dashboard* com dois sensores operacionais

No segundo caso, ilustrado na Figura 5.3, verifica-se que um dos sensores apresenta uma anomalia, representado pela cor amarela. Esta anomalia pode indicar um problema interno do sensor ou que um dos parâmetros medidos pelo sensor está fora da gama normal de funcionamento, como por exemplo a temperatura medida do sensor ser superior a um determinado valor.

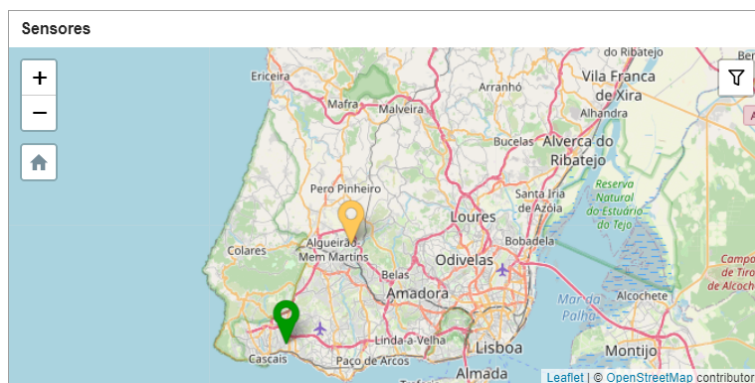


Figura 5.3: Apresentação do mapa do *Dashboard* com dois sensores operacionais e um deles indicando um alerta

No terceiro caso, ilustrado na Figura 5.4, verifica-se que um dos sensores está inoperacional, indicado pela sua cor vermelha.

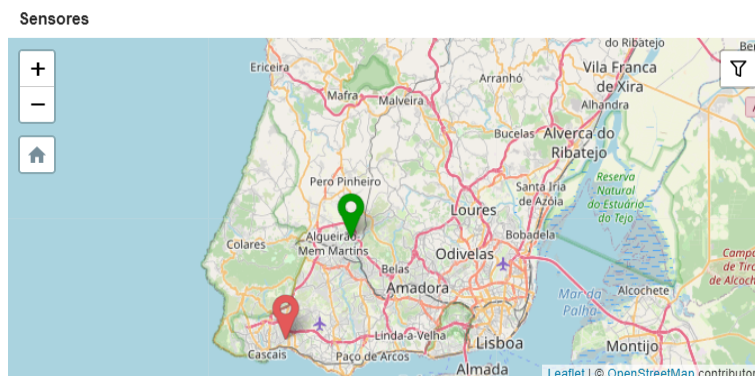


Figura 5.4: Apresentação do mapa do *Dashboard* com um sensor operacional e outro inoperacional

Para verificar que o sensor está operacional ou não, a plataforma Zabbix realiza um *ping* de x em x tempo para todos os *hosts* (sensores) e que pertencem a um *template* previamente criado. Cada *host* está associado a um

endereço IP. Se o Zabbix não conseguir ter conectividade com um determinado *host*, significa dizer que o respetivo sensor está inoperacional. Quando o *host* volta a estar operacional, não pesquisa os seus itens de forma imediata, pois esse comportamento pode sobrecarregar o *host* [18].

5.3 Apresentação dos Gráficos

A Figura 5.5 apresenta o gráfico que demonstra a evolução temporal do parâmetro LAFmax que foi definido através de um *template* que foi aplicado aos *hosts* que representam os diferentes sensores. Neste gráfico, a cor vermelha e a cor azul representam, respetivamente, os valores deste parâmetro para o Sensor 1 e para o Sensor 2.

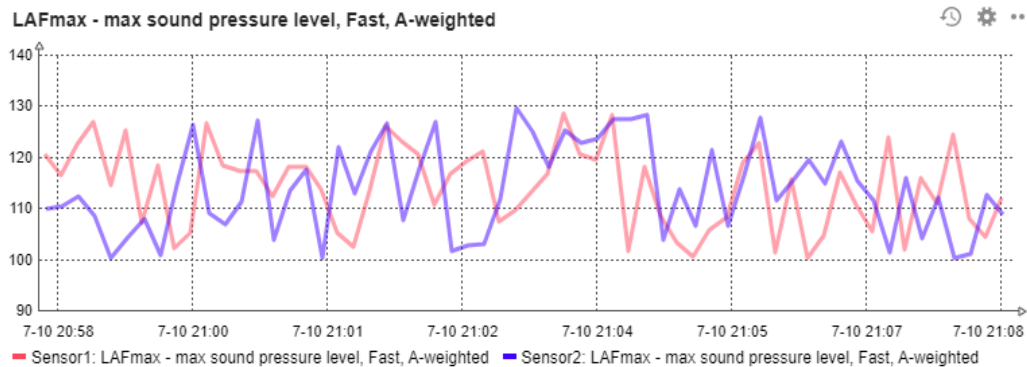


Figura 5.5: Apresentação de um gráfico no Zabbix

5.4 Apresentação de Widgets

A Figura 5.6 apresenta o aspeto visual de um *widget* do tipo *Gauge chart* que originalmente não vem incluído no Zabbix e que foi desenvolvido no contexto deste projeto. Tal como referido na secção 4.4 – **Widgets**, esta possibilidade permite apresentar os valores medidos numa forma gráfica mais personalizada.

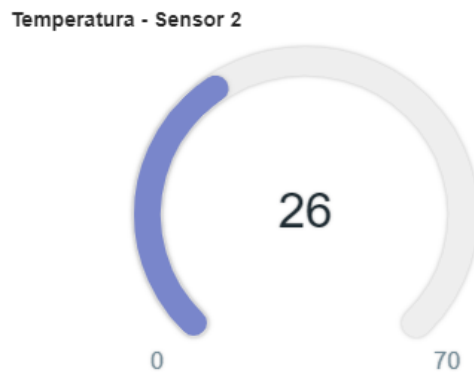


Figura 5.6: Apresentação de um `widget` no Zabbix

O exemplo mostrado na Figura 5.6 apresenta o valor mais recente da temperatura detetada pelo sensor.

5.5 Apresentação de Alertas Sobre Situações Anómalas

A informação presente nas Figuras 5.2, 5.3 e 5.4 apenas disponibiliza informação básica sobre o estado de um sensor, por exemplo saber se o sensor está ativo ou não. A utilização dos alertas (Quadro apresentado no canto superior direito da Figura 5.1) permite ter acesso ao histórico de todos os eventos e situações anómalas geradas por um *host* (sensor).

Na secção **5.5.1 – Falha de Conectividade com um Sensor**, são apresentados os resultados quando ocorre uma falha de conectividade com um determinado sensor, neste caso um *host* do Zabbix. Na secção **5.5.2 – Vagas de Calor**, são apresentados os resultados que demonstram a potencial ocorrência de uma vaga de calor. Na secção **5.5.3 – Tiros**, são apresentados os resultados que demonstram a potencial ocorrência de um tiro.

5.5.1 Falha de Conectividade com um Sensor

Quando ocorre a falha de conectividade com um sensor, pretende-se apresentar um alerta no Zabbix indicando essa situação. O objetivo é apresentar esse alerta através de um código de cores vermelho no mapa geográfico apresentado no *Dashboard*.

Para que isso possa acontecer, foi necessário definir um nível de severidade adequado para mostrar a inoperacionalidade de um sensor no mapa, pois este parâmetro acaba por definir o código de cores apresentado nesse mapa.

<input type="checkbox"/>	Severity	Name ▲	Operational data	Expression
<input type="checkbox"/>	Disaster	Ping Fail Template		<code>count(/Generate Data/icmpping,10s,,0)>0</code>

Figura 5.7: Apresentação do alerta de falha de conectividade mostrando a respetiva expressão

`count (/ GenerateData / icmpping , 10 s , , 0) > 0`

Na Figura 5.8, está ilustrada a forma como o alerta é apresentado na secção principal dos Problemas do Zabbix:

Problems						
Time ▼	Recovery time	Status	Info	Host	Problem • Severity	Duration
19:05:52		PROBLEM		Sensor1	Host Unreachable	8m 40s

Figura 5.8: Apresentação do alerta de falha de conectividade na secção dos Problemas do Zabbix

Na Figura 5.4, está ilustrado um exemplo de um sensor inoperacional e que apresenta o código de cores vermelho.

5.5.2 Vagas de Calor

A Figura 5.9 apresenta a forma como foi criado o *trigger* para indicar uma potencial ocorrência de uma vaga de calor em uma determinada localidade.

Triggers

All templates / Generate Data Items 5 Triggers 3 Graphs Dashboards Discovery rules Web scenarios

Trigger Tags Dependencies

* Name: Temperature Too High (> 50)

Event name: Temperature Too High (> 50)

Operational data:

Severity: Not classified Information **Warning** Average High Disaster

* Expression: last (/Generate Data/gettemp)>50 Add

Expression constructor

OK event generation: Expression Recovery expression None

PROBLEM event generation mode: Single Multiple

OK event closes: All problems All problems if tag values match

Allow manual close: ☐

Menu entry name: Trigger URL

Menu entry URL:

Description:

Enabled: ☒

Update Clone Delete Cancel

Figura 5.9: Demonstração do alerta de vaga de calor

A Figura 5.10 ilustra o desencadeamento de um alerta referente a uma potencial vaga de calor, mais em concreto a apresentação de uma temperatura maior do que 50 graus Celsius na localidade onde o respetivo sensor está instalado. A Figura 5.10 também mostra que o nível de severidade escolhido foi **Warning**, pois é o que gera o código de cores amarelo no mapa, que representa uma situação alarmante, e a expressão definida para desencandear o alerta é a seguinte:

```
last (/Generate Data/gettemp)>50
```

Esta expressão utiliza a chave referente ao item da temperatura pertencente ao *template* de nome “Generate Data”, que foi previamente criado para a implementação do sistema *Dashboard*.

Problems						
Time ▼	Recovery time	Status	Info	Host	Problem • Severity	Duration
14:58:12		PROBLEM		Sensor1	! Temperature Too High (> 50)	7s

Figura 5.10: Desencadeamento do alerta de vaga de calor

5.5.3 Tiros

A Figura 5.11 apresenta a forma como foi criado o *trigger* para indicar uma potencial ocorrência de um tiro em uma determinada localidade.

Triggers

All templates / Generate Data Items 8 Triggers 4 Graphs Dashboards Discovery rules Web scenarios

Trigger Tags Dependencies

* Name: Tiro

Event name: Tiro

Operational data:

Severity: Not classified Information **Warning** Average High Disaster

* Expression: last(/Generate Data/getimpulsivity)>=0.8 and last(/Generate Data/gettonality)<=0.2 and last(/Generate Data/getLCpeak)>140.0 and last(/Generate Data/getLAFmax)>=110.0 and last(/Generate Data/getLFmax)>=130.0

Expression constructor

Figura 5.11: Criação de um *trigger* para o caso de um tiro

Como é possível observar através da análise das expressões apresentadas na Figura 5.11, o alerta é desencadeado quando a impulsividade é maior ou igual do que 0.8 (relembrando que a mesma varia entre 0 e 1, tal como a tonalidade), quando a tonalidade é menor ou igual a 0.2, o nível de pressão sonora do ruído com frequências **C-weighted** é maior do que 140 dB, o **LAFmax** é maior ou igual a 110 dB e o **LFmax** é maior ou igual a 130 dB.

A Figura 5.12 ilustra um momento do desencadeamento de um alerta referente à potencial ocorrência de um tiro. O nome do alerta apresentado é “Tiro”.

Problems						
Time ▼	Recovery time	Status	Info	Host	Problem • Severity	Duration
<u>16:22:12</u>	<u>16:22:42</u>	RESOLVED		<u>Sensor1</u>	<u>Temperature Too High (> 50)</u>	<u>30s</u>
<u>16:21:42</u>	<u>16:22:02</u>	RESOLVED		<u>Sensor1</u>	<u>Temperature Too High (> 50)</u>	<u>20s</u>
<u>16:21:41</u>	<u>16:21:48</u>	RESOLVED		<u>Sensor1</u>	<u>Tiro</u>	<u>7s</u>

Figura 5.12: Desencadeamento de um alerta referente a um tiro

Capítulo 6

Conclusões e Trabalho Futuro

Este capítulo está dividido em duas seções. A seção **6.1 – Conclusões** apresenta as conclusões tiradas na realização deste projeto ao longo do semestre e todas as tarefas que conseguimos concluir até à fase atual. A seção **6.2 – Trabalho Futuro** apresenta as tarefas que inicialmente foram planejadas para serem realizadas nesta fase mas que não conseguimos realizar e que, por isso, serão realizadas em uma fase futura.

6.1 Conclusões

Ao longo do projeto, enfrentamos inúmeras dificuldades e desafios, mas acreditamos que cada obstáculo foi uma oportunidade para o nosso crescimento e desenvolvimento. Através de muito esforço, dedicação e persistência, conseguimos concluir, com êxito, algumas das tarefas propostas, apesar de não termos conseguido concluir tudo o que pretendíamos em uma fase inicial.

Uma das principais tarefas concluídas com sucesso foi a implementação de um sistema escalável, através da adição e remoção de sensores do mesmo de forma rápida e eficiente, de acordo com as necessidades, através da utilização de *templates* do Zabbix.

Também conseguimos criar e personalizar *widgets* na plataforma correspondente ao sistema *Dashboard*, por meio da criação de *hosts* e itens, para além de modificar ou remover os já existentes, podendo assim atender às necessidades dos utilizadores.

Conseguimos gerar gráficos no Zabbix, correspondentes à variação de um determinado parâmetro relativo aos níveis sonoros detetados por “sensores”

simulados em máquinas virtuais.

Para além disso, foi feita uma estimativa de valores dos parâmetros relativos a eventos sonoros que representam situações de alarme, nomeadamente as vagas de calor, tiros e gritos humanos. Para o caso dos tiros e gritos humanos, foi necessário realizar um estudo. Com base nesses parâmetros, foi possível criar alertas no **Zabbix** para que, numa fase posterior, os utilizadores possam ser imediatamente notificados sobre a ocorrência desses eventos sonoros.

Também foi configurada a plataforma **Zabbix** para gerar um alerta quando ocorrer uma falha de conectividade com um sensor.

No entanto, este estudo provavelmente não foi totalmente preciso, pelo que poderão ser necessários mais estudos para que se possa chegar a mais conclusões relativamente aos valores dos parâmetros que caracterizam estes tipos de eventos sonoros. Para além disso, existem muitos outros tipos de sons que precisam de ser estudados para que se possam criar mais alertas para notificar os utilizadores sobre mais eventos sonoros anómalos.

Para além desta, existem muitas outras tarefas que foram planeadas para serem realizadas nesta fase mas que, infelizmente, não conseguimos realizar mas que ficarão para ser completadas numa fase futura por outros grupos de projeto, tarefas essas especificadas detalhadamente na secção **6.2 – Trabalho Futuro**.

6.2 Trabalho Futuro

Criação de mais alertas no sistema *Dashboard* Apesar de já termos conseguido configurar o **Zabbix** para gerar alertas para indicar quatro situações (falha de conectividade com um *host*, vaga de calor, tiro, grito humano), existem muitos outros eventos que traduzem situações alarmantes, tais como acidentes rodoviários, explosões, fenómenos naturais, entre muitos outros, e que não foram identificados nesta fase do projeto através da geração de alertas, e é necessário completar esta tarefa de identificação de mais eventos sonoros para que se possa, de facto, contribuir melhor para o alcance do objetivo central deste projeto.

Ligação à base de dados escolhida Integrar o sistema com uma base de dados, escolhida para ser utilizada externamente à plataforma correspondente ao sistema de *Dashboard*, irá permitir o armazenamento dos dados numa base de dados adequada para o efeito, usufruindo dos benefícios que a mesma oferece.

Notificação de utilizadores registados de certos eventos sonoros Utilização e implementação de um mecanismo de notificação para os utilizadores registados no sistema, para que os mesmos sejam devidamente alertados pela plataforma, imediatamente após a ocorrência de eventos sonoros específicos, através de meios como e-mail, SMS ou notificações *push*.

Configuração das preferências do utilizador Permitir que os utilizadores configurem as suas preferências no sistema, nomeadamente as notificações que desejam receber, os filtros para os eventos sonoros do seu interesse, entre outras configurações.

Configuração das preferências dos sensores Implementar, no sistema, as preferências dos sensores pertencentes à rede. Isto envolve vários aspetos como ajustar intervalos de tempo para enviar dados.

Implementação de pedidos do utilizador à estação de som e à base de dados Permitir aos utilizadores realizar pedidos específicos relativos aos dados que pretendem obter, tanto à estação de som (sensores) como à base de dados. Isso pode incluir pedidos de dados sonoros históricos, como a média da pressão sonora dos últimos 7 dias ou do último mês, bem como outras informações sobre eventos sonoros registados.

Após a conclusão dessas tarefas, que são essenciais para a implementação do projeto na sua totalidade, diversas tarefas tais como as mencionadas abaixo podem ser consideradas para trabalhos futuros mais ambiciosos, tirando proveito deste projeto e agregando valor ao mesmo.

Expansão da rede de sensores Ampliar a cobertura da rede de sensores, adicionando mais dispositivos em áreas estratégicas, obtendo, assim, uma visão mais abrangente e precisa dos eventos sonoros ocorridos em uma determinada localidade.

Integração com sistemas de gestão de emergências Estabelecer integrações com sistemas de gestão de emergências já existentes, permitindo o envio automático de alertas e informações relevantes para as autoridades competentes em situações de eventos sonoros críticos, facilitando a tomada de decisões e ações rápidas.

Integração com sistemas de gestão urbana Integrar o sistema com outras soluções, previamente implementadas, para gestão urbana, como sistemas de monitorização de tráfego, câmaras de segurança e sistemas de controlo de ruído, com a finalidade de obter uma visão mais completa e integrada da situação urbana, permitindo uma tomada de decisão mais eficiente e coordenada.

Referências Bibliográficas

- [1] D. Lambert, “Zabbix Dashboard - Top Hosts Widget,” <https://www.youtube.com/watch?v=ntoiYJPkBso>, 2022.
- [2] F. Martins, “Zabbix Widgets - Configuration Files,” <https://github.com/OcireF/Zabbix-Widgets>, 2023.
- [3] Libelium, “New Sound Level Sensor to Control Noise Pollution,” <https://www.libelium.com/libeliumworld/new-sound-level-sensor-to-control-noise-pollution/>, 2016.
- [4] S. Systems, “Real-Time Noise Monitoring for Smart Cities,” <https://www.sonitussystems.com/insights/real-time-noise-monitoring-for-smart-cities/>, 2020.
- [5] A. A. Lab, “FI-Sonic,” <https://acusticaudiolab.isel.pt/>, 2015.
- [6] T. O. G. The Order of the Green Polo, “OpenNMS,” <https://www.opennms.com/>, 2023.
- [7] D. Lambert, “Zabbix,” <https://www.zabbix.com/>, 2023.
- [8] FinancesOnline, “Compare OpenNMS vs Zabbix,” <https://comparisons.financesonline.com/opennms-vs-zabbix>, 2023.
- [9] GetApp, “OpenNMS vs Zabbix Comparison,” <https://www.getapp.com/it-management-software/a/opennms/compare/zabbix-monitoring-solution/>, 2023.
- [10] IntelliPaat, “MongoDB vs SQL - Top Key Differences,” <https://intellipa.com/blog/mongodb-vs-sql/?US#no3>, 2023.

- [11] DB-Engines, “System Properties Comparison Elasticsearch vs MongoDB vs PostgreSQL,” <https://db-engines.com/en/system/Elasticsearch%3BMongoDB%3BPostgreSQL>, 2023.
- [12] EDUCBA, “PostgreSQL vs Elasticsearch,” <https://www.educba.com/postgresql-vs-elasticsearch/>, 2022.
- [13] D. Everything, “Difference between relational and non-relational database,” <https://blog.debugeverything.com/pt/diferenca-base-de-dados-relacional-e-nao-relacional/>, 2020.
- [14] D. Lambert, “Zabbix Documentation - 6.4 Version,” <https://www.zabbix.com/documentation/current/en>, 2023.
- [15] S. Hykes. (2021) Docker Desktop. <https://www.docker.com/products/docker-desktop/>.
- [16] Oracle, “Oracle VM VirtualBox,” <https://www.virtualbox.org/>, 2007.
- [17] D. Lambert, “Zabbix Documentation - 6.4 Version - Trigger Expressions,” <https://www.zabbix.com/documentation/current/en/manual/config/triggers/expression>, 2023.
- [18] A. Vladishev, “Zabbix 6.4 Version - Unreachable Host Definitions,” <https://www.zabbix.com/documentation/current/pt/manual/appendix/items/unreachability>, 2023.
- [19] E. Murphy, *Environmental Noise Pollution*. Elsevier, 2014.
- [20] C. M. B. da Silva, *Qualidade do ar e nível sonoro numa carreira de tiro - Estudo de Caso*, Julho 2015.
- [21] NTI, “Acústica Extrema no Campo de Tiro,” <https://www.nti-audio.com/pt/novidades/acustica-extrema-no-campo-de-tiro>, 2019.
- [22] Canonical, “Get Ubuntu Server | Download | Ubuntu,” <https://ubuntu.com/download/server#downloads>, 2023.
- [23] MySQL, “A Quick Guide to Using the MySQL APT Repository,” <https://dev.mysql.com/doc/mysql-apt-repo-quick-guide/en/>, 2023.

- [24] A. Vladishev, “Zabbix 6.4 Version - Server, Frontend, Agent,” https://www.zabbix.com/download?zabbix=6.4&os_distribution=ubuntu&os_version=22.04&components=server_frontend_agent&db=mysql&ws=apache, 2023.
- [25] —, “Zabbix 6.4 Version - Proxy,” https://www.zabbix.com/download?zabbix=6.4&os_distribution=ubuntu&os_version=22.04&components=proxy&db=mysql&ws=, 2023.
- [26] —, “Zabbix 6.4 Version - JavaGateway,” https://www.zabbix.com/download?zabbix=6.4&os_distribution=ubuntu&os_version=22.04&components=java_gateway&db=&ws=, 2023.
- [27] F. Martins. (2023) Gerador de Dados JSON em formato de ficheiro Python. https://github.com/OcireF/Sensor_simulator_code.
- [28] D. Lambert. (2023) Zabbix Docker Component Repositories. https://www.zabbix.com/container_images.

Apêndice A

Estudo dos Níveis Sonoros

Para determinar os valores dos parâmetros relativos a sons típicos que representam situações alarmantes, foi necessário realizar um estudo dos níveis sonoros de cada um dos sons que traduzem situações alarmantes. Por exemplo, sabemos que o som de um tiro tem uma determinada frequência, e caso o som detetado apresenta uma frequência com o valor aproximado dessa frequência, então provavelmente houve um tiro no local onde o sensor está instalado.

Para que possamos gerar estes alertas, é necessário determinar estes parâmetros, não apenas na frequência como outros como o nível de ruído, o nível de pressão sonora, entre outros. E, para ser possível determinar estes parâmetros, recorreremos a consulta de várias referências relativas a este tema, tanto bibliográficas como cibergráficas.

Neste capítulo, primeiro são apresentadas, na secção **A.1 – Índices Sonoros** breves descrições sobre o que consistem cada um dos parâmetros dos níveis sonoros. Na secção **A.2 – Eventos Sonoros**, são apresentados os resultados dos estudos realizados.

A.1 Índices Sonoros

Abaixo apresentam-se breves descrições sobre em que cada um dos parâmetros dos níveis sonoros consiste.

LAeq Corresponde ao nível médio contínuo de pressão sonora ao longo de um determinado período de tempo.

LCPeak Nível máximo de pressão sonora **C-weighted**, que leva em consideração todas as frequências audíveis.

LAE Nível de exposição sonora **A-weighted**, em um longo período de tempo.

SEL Nível de exposição sonora em um longo período de tempo.

LowFreq Nível de presença de sons de baixa frequência, que varia entre 0 e 1.

LAFmax Este parâmetro corresponde ao nível máximo de pressão sonora em um curto intervalo de tempo, **A-weighted**.

LAFmin Este parâmetro corresponde ao nível mínimo de pressão sonora em um curto intervalo de tempo, **A-weighted**.

LFmax Corresponde ao nível máximo de pressão sonora, num curto intervalo de tempo.

LAFmin Corresponde ao nível mínimo de pressão sonora, **A-weighted**, num curto intervalo de tempo.

LASmax Nível máximo de pressão sonora **A-weighted**, num longo período de tempo.

LASmin Nível mínimo de pressão sonora **A-weighted**, num longo período de tempo.

Impul A impulsividade é uma medida da característica abrupta e instantânea de um determinado som, que varia entre 0 e 1. Um tiro é um som extremamente impulsivo, com uma variação repentina e uma duração muito curta.

Tonal Esta característica é uma medida da presença de um tom puro em um som, que varia entre 0 e 1. Por exemplo, um tiro geralmente não possui um tom puro, mas pode apresentar uma tonalidade distinta em comparação com outros sons de fundo.

A.2 Eventos Sonoros

Para a realização do estudo dos níveis sonoros típicos de cada um dos sons típicos que representem alguma anomalia, recorreremos à consulta de outros projetos, previamente realizados, referentes aos respectivos tópicos [19] [20] [21]. Até à fase atual, concluímos o estudo dos níveis sonoros de um tiro, cujos resultados estão ilustrados na secção **A.2.1 – Tiros** e de um grito humano, cujos resultados estão ilustrados na secção **A.2.2 – Gritos Humanos**.

A.2.1 Tiros

Na tabela A.1, estão apresentados os resultados obtidos do estudo da estimativa de valores de parâmetros determinados relativos aos níveis sonoros que indicam a potencial ocorrência de um tiro.

Tabela A.1: Níveis sonoros de um tiro

Nome da Propriedade	Valor
L _{Aeq}	107.2 dB(A)
L _{AFmax}	118 dB
L _{AFmin}	57 dB
L _{Fmax}	135 dB
L _{Fmin}	95 dB
L _{ASmax}	125 dB
L _{ASmin}	90 dB
L _{AE}	100 dB
SEL	105 dB
Impul	0.8
Tonal	0.1
LC _{Peak}	151 dB
LowFreq	0.2

Os parâmetros que podemos considerar como prioridade para detetar um eventual tiro na nossa implementação são o **L_{AFmax}**, o **L_{Fmax}** e o **LC_{Peak}**, dado que um tiro produz um pico de som relativamente muito alto, em um intervalo de tempo muito curto, e também a **Impul**, dado que o tiro é um som muito impulsivo, e a **Tonal**, dado que o tiro geralmente não possui um tom puro.

A.2.2 Gritos Humanos

Na tabela A.2, estão apresentados os resultados obtidos da estimativa de valores de parâmetros determinados relativos aos níveis sonoros que indicam a potencial ocorrência de um grito humano.

Tabela A.2: Níveis sonoros de um grito humano

Nome da Propriedade	Valor
LAeq	80 dB(A)
LAFmax	90 dB
LAFmin	70 dB
LFmax	95 dB
LFmin	75 dB
LASmax	65 dB
LASmin	65 dB
LAE	75 dB
SEL	70 dB
Impul	0.8
Tonal	0.1
LCPeak	100 dB
LowFreq	0.2

Este tipo de som, tal como o tiro, apresentado na secção **A.2.1 – Tiros**, é um som muito impulsivo e com um tom não propriamente puro.

Por isso, os parâmetros a ter em conta são o **LAFmax**, **LFmax**, **Impul** e **Tonal**.

Para além disso, o grito humano é um tipo de som que apresenta bandas de frequência específicas. Por isso, os parâmetros das bandas de frequências, em 1/3 de octavo e em 1 octavo, também são úteis para ter em consideração.

Apêndice B

Instalação do Zabbix

Neste anexo estão descritos todos os passos que foram realizados para a correta instalação da plataforma **Zabbix**, a qual foi escolhida para a implementação do sistema *Dashboard*.

Esta plataforma foi instalada através de uma máquina virtual criada e gerida na plataforma de virtualização **Ubuntu**, como já referido. [16]

A máquina virtual escolhida tem o sistema operativo **Ubuntu** e a versão 22.04 LTS, e foi obtida através do site oficial do **Ubuntu**. [22]

Durante a instalação da máquina, não foi selecionado nenhum pacote adicional, sendo criado um único utilizador para trabalhar na mesma. Foram utilizadas as configurações predefinidas da máquina.

Mais tarde, para tornar mais fácil a configuração da máquina através de um terminal, foi estabelecida uma sessão **SSH** com a máquina virtual, utilizando a aplicação **Putty**. Para realizar a ligação por **SSH**, foi criada na máquina virtual uma regra de *Port Forwarding* para direccionar o tráfego da rede para um determinado porto, e esse porto foi utilizado no **Putty** para o estabelecimento da sessão **SSH**.

O **Zabbix** depende internamente da base de dados **MySQL** para exercer as suas funcionalidades. Por isso, é necessário, primeiro, instalar esta base de dados na máquina virtual. Os passos que foram seguidos para a instalação da mesma estão descritos na secção **B.1 – Instalação da Base de Dados MySQL**. Só depois da instalação da base de dados, deve ser instalado o **Zabbix** de forma correta, e para isso foram seguidos os passos descritos na secção **B.2 – Instalação do Zabbix**.

B.1 Instalação da Base de Dados MySQL

Para instalação desta base de dados, foi utilizado como guia o site oficial do MySQL. [23]

Primeiro, foi criada uma diretoria à parte para instalar, na máquina, todos os ficheiros referentes à base de dados MySQL. Apesar de este passo ser opcional, contribui para uma melhor organização dos ficheiros de instalação. Para fazer isso, foi executado na máquina o seguinte comando:

```
mkdir mySQLInstallFiles
```

De seguida, foi adicionado o repositório MySQL APT, utilizando os seguintes comandos abaixo.

Obter os pacotes do MySQL de uma página web:

```
wget https://dev.mysql.com/get/mysql-apt-config_0.8.25-1_all.deb
```

Instalar os pacotes do MySQL no software da máquina virtual:

```
sudo dpkg -i mysql-apt-config_0.8.25-1_all.deb
```

Atualizar a informação dos pacotes:

```
sudo apt-get update
```

Instalar o MySQL com APT:

```
sudo apt-get install mysql-server
```

Verificar o estado da base de dados MySQL, deve estar *ACTIVE (RUNNING)*:

```
systemctl status mysql
```

Para selecionar uma versão mais recente do MySQL, e depois atualizar novamente os pacotes:

```
sudo dpkg-reconfigure mysql-apt-config
```

```
sudo apt-get update
```

Outra opção válida seria instalar o MySQL do *source* com o repositório MySQL APT.

Atualizar os pacotes:

```
sudo apt-get update
```

Instalar os pacotes dos quais o processo de instalação depende:

```
sudo apt-get build-dep mysql-server
```

Instalar o código fonte dos componentes de MySQL, e construí-los (correr este comando na diretoria na qual pretende que os ficheiros de instalação estejam localizados):

```
sudo apt-get source -b mysql-server
```

Instalar um determinado pacote, normalmente não é necessário pois todos os pacotes necessários já estão automaticamente instalados:

```
sudo dpkg -i (nome do pacote).deb
```

Para ver os pacotes do repositório MySQL atualmente instalados:

```
sudo dpkg -l | grep mysql | grep ii
```

B.2 Instalação do Zabbix

Depois de instalar corretamente a base de dados MySQL na máquina virtual, poderá ser instalado, com segurança, a plataforma Zabbix. Para a sua instalação, foram seguidas as instruções do site oficial do Zabbix. [24] [25] [26]

Mais uma vez, apesar de ser um passo opcional, por uma questão de organização, foi criada uma diretoria à parte para os ficheiros de instalação do Zabbix.

```
mkdir mySQLInstallFiles
```

Agora, será necessário instalar o repositório do Zabbix. Para isso, temos de obter o repositório do arquivo da web, depois instalar esse repositório como um pacote Debian no sistema operativo da minha máquina virtual, e atualizar os pacotes disponíveis para instalação.

```
wget https://repo.zabbix.com/zabbix/6.4/ubuntu/pool/main/z/ \
zabbix-release/zabbix-release_6.4-1+ubuntu22.04_all.deb
```

```
sudo dpkg -i zabbix-release_6.4-1+ubuntu22.04_all.deb
```

```
sudo apt update
```

Agora, é necessário instalar os componentes Server, Frontend e Agent do Zabbix:

```
sudo apt install zabbix-server-mysql zabbix-frontend-php \
zabbix-apache-conf zabbix-sql-scripts zabbix-agent
```

Agora, temos de criar a base de dados inicial no Zabbix.

Primeiro, foi necessário alterar a palavra-passe do utilizador root. Para isso, são necessários os seguintes passos:

```
systemctl stop mysql

sudo mysqld_safe --skip-grant-tables &
### abrir uma nova janela de comandos se necessário

mysql -uroot
(vai entrar dentro do menu mysql)

mysql> use mysql;

mysql> flush privileges;

mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';

### substituir 'MyNewPass' por uma palavra-passe à escolha
mysql> flush privileges;

mysql> quit; ### sair do menu mysql
```

Depois disto, foi criada a base de dados inicial:

```
mysql -uroot -p
### inserir a password escolhida anteriormente em
### (IDENTIFIED BY ...)

mysql> create database zabbix character set \
utf8mb4 collate utf8mb4_bin;

mysql> create user zabbix@localhost identified by 'password';

mysql> grant all privileges on zabbix.* to zabbix@localhost;

mysql> set global log_bin_trust_function_creators = 1;

mysql> quit;
```

Importar o esquema e os dados no servidor do Zabbix:

```
mysql> zcat /usr/share/zabbix-sql-scripts/mysql/server.sql.gz | \
mysql --default-character-set=utf8mb4 -uzabbix -p zabbix

### inserir a password escolhida no comando
### 'create user ... 'password''
```

Depois, desativar a opção de log_bin_trust_function_creators:

```
mysql -uroot -p
### password escolhida
```

```
mysql> set global log_bin_trust_function_creators = 0;
mysql> quit;
```

Configurar a base de dados para o servidor Zabbix editando o ficheiro `/etc/zabbix/zabbix_server.conf`:

```
sudo nano /etc/zabbix/zabbix_server.conf

### editar a seguinte linha no ficheiro
DBPassword=<password escolhida para o MySQL>

### Trocar o nome do host para 'localhost'
```

Iniciar o processo de execução servidor e do agente:

```
systemctl restart zabbix-server zabbix-agent apache2

systemctl enable zabbix-server zabbix-agent apache2
```

Agora será possível aceder ao site `http://localhost/zabbix`

Instalar a componente Proxy para o Zabbix: (criar a base de dados inicial tal como na instalação das componentes do servidor e do agente)

```
sudo apt update

sudo apt install zabbix-proxy-mysql zabbix-sql-scripts

mysql -uroot -p

## inserir a password escolhida ##

mysql> create database zabbix_proxy character \
set utf8mb4 collate utf8mb4_bin;

mysql> create user CMzabbix@localhost identified by \
'password';
### 'password' ou outra palavra-passe à escolha

mysql> grant all privileges on zabbix_proxy.* to CMzabbix@localhost;

mysql> set global log_bin_trust_function_creators = 1;

mysql> quit;
```

Importar os dados e esquema iniciais no servidor do Zabbix:

```
sudo cat /usr/share/zabbix-sql-scripts/mysql/proxy.sql | mysql \
--default-character-set=utf8mb4 -uCMzabbix -p zabbix_proxy
```

```
### inserir a password de create user ... 'password'
```

Desativar a opção de log_bin_trust_function_creators:

```
mysql -uroot -p
```

```
### inserir a password escolhida ###
```

```
mysql> set global log_bin_trust_function_creators = 0;
```

```
mysql> quit;
```

Instalar a componente Java Gateway para o Zabbix (opcional):

```
sudo apt update
```

```
sudo apt install zabbix-java-gateway
```

```
systemctl restart zabbix-java-gateway
```

```
systemctl enable zabbix-java-gateway
```

Apêndice C

Simulação dos Sensores

Para simular os sensores como descrito anteriormente, foram criadas duas máquinas virtuais, uma representando cada sensor. As máquinas foram criadas e configuradas da mesma forma que foi criada a máquina na qual foi instalada a plataforma Zabbix.

Depois de instalar as máquinas, foi criado, em cada uma das máquinas, um ficheiro Python que criava objetos em formato JSON. Neste ficheiro, são criadas várias rotas HTTP que são executadas quando um pedido HTTP é feito por parte de uma outra entidade, como por exemplo uma aplicação Web.

Uma dessas rotas, quando é executada, permite obter um objeto no formato JSON, cujo conteúdo representa o dos níveis sonoros detetados por um sensor real. Esta rota tem a seguinte implementação, apresentada na figura 1:

Código 1: Gerador de objetos JSON

```
@app.route('/GeradorJson', methods=['GET'])
def GeradorJson():

    data = {
        "DataRecord": {
            "STATION": {
                "ID": 100,
                "Geolocation": 10.4,
                "Direction": 10.1,
                "MicSensitivity": 1.1,
                "Height": 1.1,
                "Orientation": {
                    "Azimuth": 1.1,
                    "Elevation": 1.1,
                }
            }
        }
    }
```

```

},
"timeStamp": 10.0,
"noise_levels": {
  "LAeq": [],
  "LAeq_Lin": [],
  "LAFmax": [],
  "LAFmax_Lin": [],
  "LFmax": [],
  "LFmax_Lin": [],
  "LAFmin": [],
  "LAFmin_Lin": [],
  "LASmax": [],
  "LASmax_Lin": [],
  "LASmin": [],
  "LASmin_Lin": [],
  "LFmin": [],
  "LFmin_Lin": [],
  "LCPeak": [],
  "LCPeak_Lin": [],
  "LAE": [],
  "LAE_Lin": [],
  "SEL": [],
  "SEL_Lin": [],
  "Impul": [],
  "Tonal": [],
  "LowFreq": [],
  "Freq1_0OctBands": {
    "16": [],
    "31.5": [],
    "63": [],
    "125": [],
    "250": [],
    "500": [],
    "1000": [],
    "2000": [],
    "4000": [],
    "8000": [],
    "16000": []
  },
},
"Freq1_3OctBands": {
  "16": [],
  "20": [],
  "25": [],
  "31.5": [],
  "40": [],
  "50": [],
  "63": [],
  "80": [],
  "100": [],

```



```

        "125" : [] ,
        "160" : [] ,
        "200" : [] ,
        "250" : [] ,
        "315" : [] ,
        "400" : [] ,
        "500" : [] ,
        "630" : [] ,
        "800" : [] ,
        "1000" : [] ,
        "1250" : [] ,
        "1600" : [] ,
        "2000" : [] ,
        "2500" : [] ,
        "3150" : [] ,
        "4000" : [] ,
        "5000" : [] ,
        "6300" : [] ,
        "8000" : [] ,
        "10000" : [] ,
        "12500" : [] ,
        "16000" : [] ,
        "20000" : []
    }
},
"weather": {
    "Temperature" : [] ,
    "AirSpeed" : [] ,
    "AirDirection" : []
}
}

data [ "DataRecord" ] [ "STATION" ] [ "ID" ] =
    random.randint(0, 1000000)
data [ "DataRecord" ] [ "STATION" ] [ "Geolocation" ] =
    random.randint(0, 1000000) / 10
data [ "DataRecord" ] [ "STATION" ] [ "Direction" ] =
    random.randint(0, 10000) / 10
data [ "DataRecord" ] [ "STATION" ] [ "MicSensitivity" ] =
    random.randint(0, 10000) / 10
data [ "DataRecord" ] [ "STATION" ] [ "Height" ] =
    random.randint(0, 10000) / 10
data [ "DataRecord" ] [ "STATION" ] [ "Orientation" ] [ "Azimuth" ] =
    random.randint(0, 10000) / 10
data [ "DataRecord" ] [ "STATION" ] [ "Orientation" ] [ "Elevation" ] =
    random.randint(0, 10000) / 10

data [ "DataRecord" ] [ "timeStamp" ] = random.randint(0,10000) / 10

```

```

data [ "DataRecord" ] [ "noise_levels" ] [ "LAeq" ]
    .append(random.randint(350, 750) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LAFmax" ]
    .append(random.randint(1000, 1300) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LFmax" ]
    .append(random.randint(1000, 1500) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LAFmin" ]
    .append(random.randint(200, 600) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LASmax" ]
    .append(random.randint(1000, 1400) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LASmin" ]
    .append(random.randint(600, 1000) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LFmin" ]
    .append(random.randint(500, 1000) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LCPeak" ]
    .append(random.randint(100, 1700) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "LAE" ]
    .append(random.randint(300, 1100) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "SEL" ]
    .append(random.randint(400, 1100) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "Impul" ]
    .append(random.randint(0, 10) / 10)

data [ "DataRecord" ] [ "noise_levels" ] [ "Tonal" ]
    .append(random.randint(0, 10) / 10)

    data [ "DataRecord" ] [ "noise_levels" ] [ "LowFreq" ]
        .append(random.randint(0, 10) / 10)

data [ "DataRecord" ] [ "weather" ] [ "Temperature" ]
    .append(random.randint(0, 70))

data [ "DataRecord" ] [ "weather" ] [ "AirSpeed" ]
    .append(random.randint(0, 70))

data [ "DataRecord" ] [ "weather" ] [ "AirDirection" ]
    .append(random.randint(0, 70))

```

```
print("Pedido recebido!")
return data
```

Este código está disponibilizado num repositório *git*. [27]

Como é possível observar acima, a função cria um objeto que representa a estrutura de um objeto JSON.

Em seguida, a função gera dados aleatórios para alguns campos do objeto “data”. Por exemplo, ela gera valores aleatórios para a temperatura do campo “Temperature” dentro do campo “weather”.

Finalmente, a função exibe a mensagem “Pedido recebido!” na consola e retorna o objeto, que será convertido em formato JSON e enviado como resposta ao pedido HTTP GET, utilizando a rota `’/GeradorJSON’`.

Para o ficheiro ser executado automaticamente logo no arranque da máquina virtual, foram feitas várias configurações na mesma, descritas abaixo.

Editar o ficheiro “go.sh” (**sudo nano go.sh**), de modo a que o mesmo tenha o seguinte conteúdo listado na figura 2:

Código 2: Ficheiro go.sh

```
#!/bin/bash

sudo -H -u cmvm2 bash -c \
"/bin/python3 /home/cmvm2/pythonFiles/geradorJSON.py"
```

De seguida, foi editado o ficheiro “/lib/systemd/system/sensorScript.service” (**sudo nano /lib/systemd/system/sensorScript.service**), tendo o conteúdo listado na Figura 3:

Código 3: Ficheiro sensorScript.service

```
[Unit]
Description=Sensor_Dashboard_Script1

[Service]
ExecStart=/home/cmvm2/go.sh

[Install]
WantedBy=multi-user.target
```

Assume-se que **cmvm3** é o nome do utilizador criado na máquina virtual.

Atribuir as permissões de execução do ficheiro “go.sh” para o utilizador **cmvm3**:

```
chmod +x go.sh
```

O ficheiro **sensorScript.service** guarda os comandos que serão executados logo no arranque da máquina virtual.

Para iniciar o serviço de *script* de automação:

```
sudo systemctl start sensorScript.service
```

Para parar o serviço de *script* de automação:

```
sudo systemctl stop sensorScript.service
```

Para verificar o estado do serviço de *script* de automação:

```
sudo systemctl status sensorScript.service
```

Antes de editar os ficheiros **Python** presentes nas máquinas virtuais, deve-se parar o serviço de *script* automatizado, e depois de editar os mesmos deve-se iniciar novamente o serviço de *script* automatizado.

Apêndice D

Utilizando o Docker Desktop

Este capítulo começa com uma apresentação sobre em que consiste a aplicação **Docker Desktop**, apresentada na secção **D.1 – Definição do Docker Desktop**.

Por fim, são especificados os passos detalhados para montar o ambiente de execução para implementação do sistema *Dashboard* através desta aplicação, na secção **D.2 – Ambiente de Execução**.

D.1 Definição do Docker Desktop

O **Docker Desktop** é uma aplicação de *Desktop* que fornece uma forma fácil para criação, configuração e gestão de ambientes **Docker** em sistemas operativos **Windows** e **macOS**. Nesta aplicação, combina-se o uso do **Docker Engine**, o *runtime* do **Docker**, com uma interface gráfica intuitiva e recursos adicionais para simplificar o desenvolvimento e a execução de *containers*.

Torna-se, assim, possível a criação e execução de *containers Docker* na máquina local, oferecendo uma solução mais simplificada, eficiente e completa para desenvolver, testar e executar aplicações em *containers*, em vez de ter de se lidar com o trabalho e custos da criação de uma máquina remota e da configuração de ambientes complexos.

D.2 Ambiente de Execução

Na secção **D.2.1 – Instalação do Docker Desktop**, é descrito o primeiro passo necessário, que é instalar a aplicação no sistema operativo da minha

máquina local.

Na secção **D.2.2 – Configuração do Docker Desktop**, são descritas as configurações necessárias para que a aplicação possa concretizar o sistema *Dashboard* através do Zabbix e da simulação dos sensores a enviarem os dados para o Zabbix.

D.2.1 Instalação do Docker Desktop

Para instalar esta aplicação, acedemos ao *site* oficial do Docker [15] e descarregámos o ficheiro de instalação da aplicação correspondente ao sistema operativo da nossa máquina local.

D.2.2 Configuração do Docker Desktop

Para a simulação de um sensor, é necessário criar um *Dockerfile* para configurar o mesmo.

Um *Dockerfile* é um ficheiro de texto que contém instruções para a construção da imagem do Docker que especifica a imagem base, instala as dependências necessárias, copia os seus ficheiros e configura a máquina.

E temos de pegar as imagens do Docker utilizadas para o Zabbix, que são o Zabbix Server, o Zabbix Proxy, o Zabbix Frontend, o Zabbix Agent e o Zabbix Java. Os mesmos são pegos diretamente do site oficial do Zabbix como repositórios. [28]

Depois de criar o *Dockerfile* para cada sensor, é possível construir as imagens do Docker correspondentes. Para isso, é necessário abrir uma janela de comandos e navegar até à diretoria onde os ficheiros *Dockerfile* estão localizados.

O comando a ser executado, para cada *Dockerfile*, é o apresentado abaixo, substituindo “nome da imagem” pelo nome a ser dado à imagem do Docker.

```
docker build -t <nome da imagem>
```

Uma vez que as imagens *Dockerfile* estejam construídas, deve ser executado o seguinte comando para correr a imagem previamente criada por cada máquina virtual, sendo que “nome do container” corresponde ao nome do *container* a ser criado e “nome da imagem” corresponde ao nome da imagem do Docker.

```
docker run --name <nome do container> -d <nome da imagem>
```

Para criarmos dois sensores diferentes, é necessário executar os seguintes comandos:

```
docker run -d --name <nome do sensor> sensor  
docker run -d --name <nome do sensor> sensor
```

Desta forma, o ambiente já está montado para a correta implementação do sistema de *Dashboard* através do uso da aplicação **Docker Desktop**.