



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Ferramenta Centralizadora de Monitorização

JORGE RAFAEL DE MELIM FARIA MARTINS, N42175

Relatório final realizado no âmbito de Projecto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020-2021

Orientadores : Engenheiro Artur Ferreira
Engenheiro Hugo José (ClaraNet)

Setembro, 2021



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área Departamental de Engenharia de Electrónica e Telecomunicações e de
Computadores**

Ferramenta Centralizadora de Monitorização

JORGE RAFAEL DE MELIM FARIA MARTINS, N42175

Relatório final realizado no âmbito de Projecto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020-2021

Setembro, 2021

Agradecimentos

Começo por agradecer ao Professor Artur Ferreira, pela sua orientação e acompanhamento ao longo do desenvolvimento do projeto. Um agradecimento especial aos Engenheiros Hugo José, Mariana Marques e Bruno Roque por todo o acompanhamento, disponibilidade, apoio, reuniões semanais e feedback que foram cruciais no desenvolvimento do projeto. Agradeço igualmente à ClaraNet, pela oportunidade do desenvolvimento deste projeto, integrando o mesmo num contexto empresarial.

Agradeço profundamente à minha família, nomeadamente à minha mãe, irmãos e irmã, por todo o apoio incondicional por períodos adversos e por percorrerem esta aventura comigo e ainda um agradecimento muito especial ao meu pai, por tudo o que me ensinou. Agradeço a todos os meus amigos que me apoiaram e ajudaram, tirando parte do seu tempo para providenciarem feedback. E ainda um agradecimento especial à Katia Pestana, por todo o seu acompanhamento, opiniões e apoio infinito, que refletiram num melhor eu e em grandes conquistas.

Resumo

A necessidade de monitorização de diferentes aspectos do funcionamento de componentes informáticos torna as ferramentas de monitorização imprescindíveis a todos que requerem o uso das mesmas.

Essa mesma alta necessidade exige com que hajam múltiplas instâncias de várias ferramentas, fazendo com que a informação proveniente das ferramentas seja distribuída e dispersa através de vários ecrãs e visualizações, criando o problema de dispersão e desagregação de informação.

Este projeto visa a solução do problema apresentado através do desenvolvimento de uma aplicação web single page que centralize toda a informação distribuída num único ponto, através do desenvolvimento de componentes *Back-End*, *Front-End* e base de dados, apresentando a informação proveniente de instâncias *Zabbix* numa *dashboard* eficaz e intuitiva.

Foi desenvolvida uma ferramenta de utilização simples e eficiente, foram utilizadas as frame-works *Spring-boot* (Componente Back-end) e *React* (Componente Front-end). A web-app encontra-se estruturada através do fundamento de dois tipos de utilizadores, "*Admin Users*" e "*Standard Users*". Dado que o objectivo fulcral é a obtenção de eventos provenientes de várias instâncias de ferramentas de monitorização, um *Admin User* controla toda a plataforma e os eventos recebidos no *feed* final de um utilizador.

Palavras-chave: Monitorização, Instâncias, Informação, Aplicação web, Zabbix

Abstract

The necessity of monitoring different functional aspects of IT components makes monitoring tools essential for everyone that require to use them.

That high demand develops into the existence of multiple instances from various tools, causing the information provided by them to be widespread through several computer screens, resulting in a widespread and disjunction of information.

This project develops a solution for the problem at hand through a single page web application that centralizes the widespread information on a single point, through the development of *Back-End*, *Front-End* and Data Base components, showing the information provided by *Zabbix* instances on a single effective and intuitive *dashboard*.

A simple and effective tool was developed, with frame-works such as *Spring-boot* (Back-End Component) and *React* (Front-End Component) were used. The web-app itself was fundamentally structured behind the concept of two types of users, "*Admin Users*" and "*Standard Users*". Given that the main objective is the retrieval of events from multiple monitoring tools instances, a *Admin User* controls the whole platform and events received at a user's final feed.

Keywords: Monitoring, Instances, Information, Web application, Zabbix

Índice

Lista de Figuras	xiii
Lista de Tabelas	xv
Lista de Listagens	xvii
1 Introdução	1
1.1 Problema	2
1.2 Organização do documento	2
2 Formulação do Problema	3
2.1 Ferramentas de Monitorização	3
2.1.1 <i>Zabbix</i>	4
2.2 Desafio apresentado	5
2.3 Terminologia e conceitos	5
3 Abordagem	9
3.1 Reuniões ClaraNet	9
3.2 Solução proposta	10
4 Implementação	13
4.1 Base de dados	14

4.1.1	<i>User</i>	14
4.1.2	<i>Connection</i>	15
4.2	Componente <i>Back-End</i>	15
4.2.1	<i>Java Persistence API</i>	15
4.2.2	Interface geral para conectores	17
4.2.3	Exemplo de pedido à API <i>Zabbix</i>	18
4.2.4	Conector <i>Zabbix</i>	19
4.2.5	<i>Request</i> e <i>RequestBuilder</i>	21
4.2.6	Atributos <i>Zabbix</i>	22
4.2.7	<i>End-points</i>	24
4.2.8	Controladores	27
4.2.8.1	Controlador de Eventos	27
4.2.8.2	<i>Zabbix Controller</i>	29
4.2.9	Autenticação de utilizadores	30
4.3	Componente <i>Front-End</i>	31
4.3.1	Events Page	31
4.3.2	<i>Setup Connections</i>	34
4.3.2.1	<i>Manage Connections</i>	35
4.3.2.2	<i>Configure</i>	37
4.3.3	<i>Configure Users</i>	38
5	Conclusão	41
5.1	Trabalho Futuro	42
	Referências	45

Lista de Figuras

2.1	Dashboard <i>Zabbix</i>	6
3.1	Estrutura do Projeto	10
4.1	Modelo EA	14
4.2	Página principal Events	32
4.3	Página Setup Connections	34
4.4	Configuração de conexões do sistema	35
4.5	Registo de nova conexão	36
4.6	Configuração de conexões do utilizador Admin	37
4.7	Configuração de utilizadores	38
4.8	Registo de utilizadores	39

Lista de Tabelas

4.1	Propriedades do objecto Event	22
4.2	Propriedades do objecto Trigger	23
4.3	Propriedades do objecto Host	24
4.4	Controladores <i>Back-End</i>	27

Lista de Listagens

4.1	<i>Implementação UserRepository</i>	16
4.2	<i>Exemplo de definição de query</i>	16
4.3	Código fonte para Interface Conector.	17
4.4	Exemplo de pedido JSON - Event.get.	18
4.5	Código fonte conector Zabbix - Construtor.	19
4.6	Código fonte conector Zabbix - Método Init.	19
4.7	Código fonte conector Zabbix - Método Destroy.	19
4.8	Código fonte conector Zabbix - Método Call.	20
4.9	Código fonte conector Zabbix - Método Login.	21
4.10	Exemplo de <i>Request</i>	21
4.11	Request de Events	28
4.12	Construção de Instância <i>Zabbix</i>	30



Introdução

As ferramentas de monitorização provam ser indispensáveis num planeta em rápida expansão tecnológica e, conseqüentemente, na monitorização de ditas tecnologias. Estas por sua vez permitem monitorizar diferentes componentes tais como servidores, máquinas virtuais, serviços de *cloud* e *networks* em inúmeros aspectos tais como utilização de rede, carga de CPU e consumo de espaço em disco.

Em dados serviços, estas informações, que são fornecidas em tempo real ou temporariamente mediante a necessidade dos utilizadores, são cruciais para a sua finalidade. Sem o fornecimento das mesmas, seria virtualmente impossível o controlo simples e eficaz de uma infraestrutura complexa de vários servidores, por exemplo. Contudo, existem ao dispor de um fornecedor de serviços de monitorização várias ferramentas. Todas apresentam as suas vantagens e desvantagens, porém, acabam por realizar o seu propósito final, ou seja, a monitorização de sistemas. Todavia, a disponibilidade de várias ferramentas, conforma-se devidamente noutro problema.

Este projeto foi realizado em parceria com a empresa ClaraNet. A ClaraNet é uma empresa que fornece serviços *ISP*, que se foca maioritariamente em soluções de *Cloud*, *Cibersegurança* e *Workplace*. Contudo, devido à sua natureza com operações relacionadas com a *Internet*, apresenta igualmente soluções de monitorização.

1.1 Problema

A necessidade de monitorização de grandes quantidades de equipamentos provenientes de diferentes pontos e organizações, resulta na utilização de várias ferramentas de monitorização e ainda de múltiplas instâncias das mesmas. Este facto culmina-se numa enorme quantidade de informação distribuída, por sua vez, em vários monitores para permitir a visualização das mesmas.

Como consequência, trabalhadores de empresas como a ClaraNet necessitam de um elevado número de ecrãs na sua área de trabalho de forma a monitorizar eficientemente um elevado número de alarmes despoletados pelas ferramentas de monitorização. Ademais da necessidade de uma parede de ecrãs que disponibiliza uma *overview* de várias instâncias de monitorização, com a finalidade de realçar eventos de maior severidade para trabalhadores que actuam sobre esses mesmos eventos.

O facto de ser apresentada uma grande quantidade de informação dispersa por vários monitores poderá dificultar a capacidade e eficiência de monitorização por parte de trabalhadores que estejam a deparar-se com as mesmas.

1.2 Organização do documento

A organização deste documento divide-se em cinco capítulos. No segundo capítulo, será aprofundado o desafio apresentado pela ClaraNet, juntamente com uma breve introdução às ferramentas de monitorização e o seu uso. No terceiro capítulo apresenta-se a solução desenvolvida para o problema em questão. No quarto aprofunda-se a implementação da solução apresentada e por fim, no quinto capítulo descrevem-se as conclusões finais.

2

Formulação do Problema

Neste capítulo será realizada uma análise em maior profundidade ao problema apresentado. Na secção 2.1 é realizada uma apresentação às ferramentas de monitorização, juntamente com um destaque sobre a ferramenta *Zabbix* na secção 2.1.1. Seguidamente, na secção 2.2, é detalhado o desafio apresentado e finalmente, na secção 2.3, são apresentados alguns conceitos chave para uma melhor compreensão das terminologias utilizadas no desenvolvimento do projeto.

Para primeiro entender o problema, é necessária uma contextualização sobre as ferramentas de monitorização e como estas funcionam.

2.1 Ferramentas de Monitorização

Tal como previamente mencionando, as ferramentas de monitorização têm como intuito possibilitar a monitorização de diferentes componentes tais como servidores, máquinas virtuais, serviços de *cloud* e *networks*. Estas conseguem fornecer várias informações tais como métricas de monitorização, utilização de redes, *workload* de um *CPU* e consumo de espaço em disco.

Neste projeto temos três ferramentas como referência, especificadas por parte da ClaraNet: o *Zabbix*¹[9], o *Nagios*²[5] e o *Nimsoft*³[6]. Foram seleccionadas estas ferramentas como referência, não obstante da quantidade disponível no mercado, devido ao facto de serem as ferramentas que apresentaram maior utilidade por parte da ClaraNet. Estas ferramentas disponibilizam API's de forma a que seja possível existir uma interação entre aplicações desenvolvidas por terceiros com as mesmas.

Todavia, devido a restrições de tempo e pelo desafio apresentado ser originalmente um projeto previsto para pelo menos dois alunos, foi realizado um ajuste de requisitos. Neste contexto, definiu-se como objetivo o desenvolvimento de uma interface que permita à implementação futura de outras ferramentas de monitorização, sendo realizada a implementação para a ferramenta de monitorização *Zabbix*.

2.1.1 *Zabbix*

Utilizou-se o *Zabbix* como a ferramenta implementada por indicação da ClaraNet, sendo a ferramenta de monitorização de uso (actualmente) quase exclusivo por parte da empresa. A ferramenta *Zabbix* permite vários tipos de monitorização:

- Verificações simples que permitem aferir a disponibilidade e respostas de serviços simples como *Simple Mail Transfer Protocol* (SMTP) e *Hypertext Transfer Protocol* (HTTP) sem a instalação de software no *host* monitorizado.
- Um agente *Zabbix* poderá ser instalado em *hosts* com sistemas operativos do tipo Linux e Windows para monitorizar estatísticas tais como uso de CPU, uso de redes, espaço de memória em disco, entre outros.
- Como alternativa à instalação de agentes nos *hosts*, o *Zabbix* também suporta monitorização através de verificações SNMP, TCP e ICMP, suportando assim uma grande variedade de mecanismos de notificação "*near-real-time*".

Em termos de funcionamento, o *Zabbix* pode ser subdividido em servidor *Zabbix* e agente *Zabbix*. Os agentes serão instalados em várias máquinas, tal como previamente mencionado, e estes terão uma ligação com um dado servidor *Zabbix* que estará a receber informação proveniente de agentes configurados. Existem outros componentes

¹<https://www.zabbix.com/>

²<https://www.nagios.org/>

³<https://support.nimsoft.com/>

Zabbix, tais como *Proxies* e *Senders*, contudo não será relevante estar a descrever pois não apresentam relevância para o projeto desenvolvido.

No desenvolvimento desta web-app, utilizou-se a documentação referente à versão 4.2 da ferramenta *Zabbix*⁴[10].

2.2 Desafio apresentado

Foi lançado um desafio para o desenvolvimento de uma aplicação web que soluciona a distribuição da informação proveniente das ferramentas de monitorização, centralizando essa informação numa única aplicação. Definiu-se como requisitos principais:

- Possibilitar a configuração de múltiplos conectores para ferramentas.
- Implementação de alarmes e listas agregadoras dos mesmos.
- Existência de perfis de acesso diferenciados para diversos tipos de utilizadores.
- Configurações a nível de utilizadores.
- Implementação de Dashboards sobre a aplicação.
- Obrigatoriedade de funcionamento com a ferramenta *Zabbix*

Devido à dimensão do projeto, foi definido por parte da ClaraNet que o objectivo fundamental do projeto seria possibilitar a conexão da *Web App* a múltiplas instâncias de servidores *Zabbix*. Tal deve-se ao facto da ferramenta de monitorização principal utilizada pela empresa ser precisamente o *Zabbix*.

2.3 Terminologia e conceitos

Existem algumas palavras chave que permitem obter uma melhor compreensão sobre o projecto em geral, nomeadamente:

Conector - Designa-se como conector a ligação entre a web-app com uma instância de uma ferramenta de monitorização. Por exemplo, é necessário a criação de um objecto não estático que estabeleça a conexão com o *end-point* `http://195.22.17.158/zabbix/` de forma a permitir a interação entre a web-app com esse servidor *Zabbix*.

⁴<https://www.zabbix.com/documentation/4.2/manual/api/reference>

Alarmes - As ferramentas de monitorização funcionam na base de alarmes, também designados por eventos. Estes são despoletados através de *triggers* criados manualmente. Por exemplo, imaginemos que estamos a monitorizar um servidor e adicionamos um *trigger* para despoletar um alarme quando o mesmo tiver ocupado 80% da sua capacidade de armazenamento. Ao ser atingido esse valor, a ferramenta envia um alarme a avisar que o mesmo ocorreu. É possível realizar diferentes ações sobre estes alarmes, tais como adicionar texto complementar ou realizar um *acknowledge*.

Dashboards analíticos - Os dashboards analíticos são desenvolvidos na componente *Front-end* da web-app e têm como intuito permitir a visualização simples e eficaz sobre os componentes monitorizados. Esta deverá ser adaptável ao gosto do utilizador permitindo ser realçado diversos componentes mediante as necessidades do utilizador. Apresenta-se um exemplo de um dashboard analítico através da Figura 2.1, proveniente da ferramenta *Zabbix*.

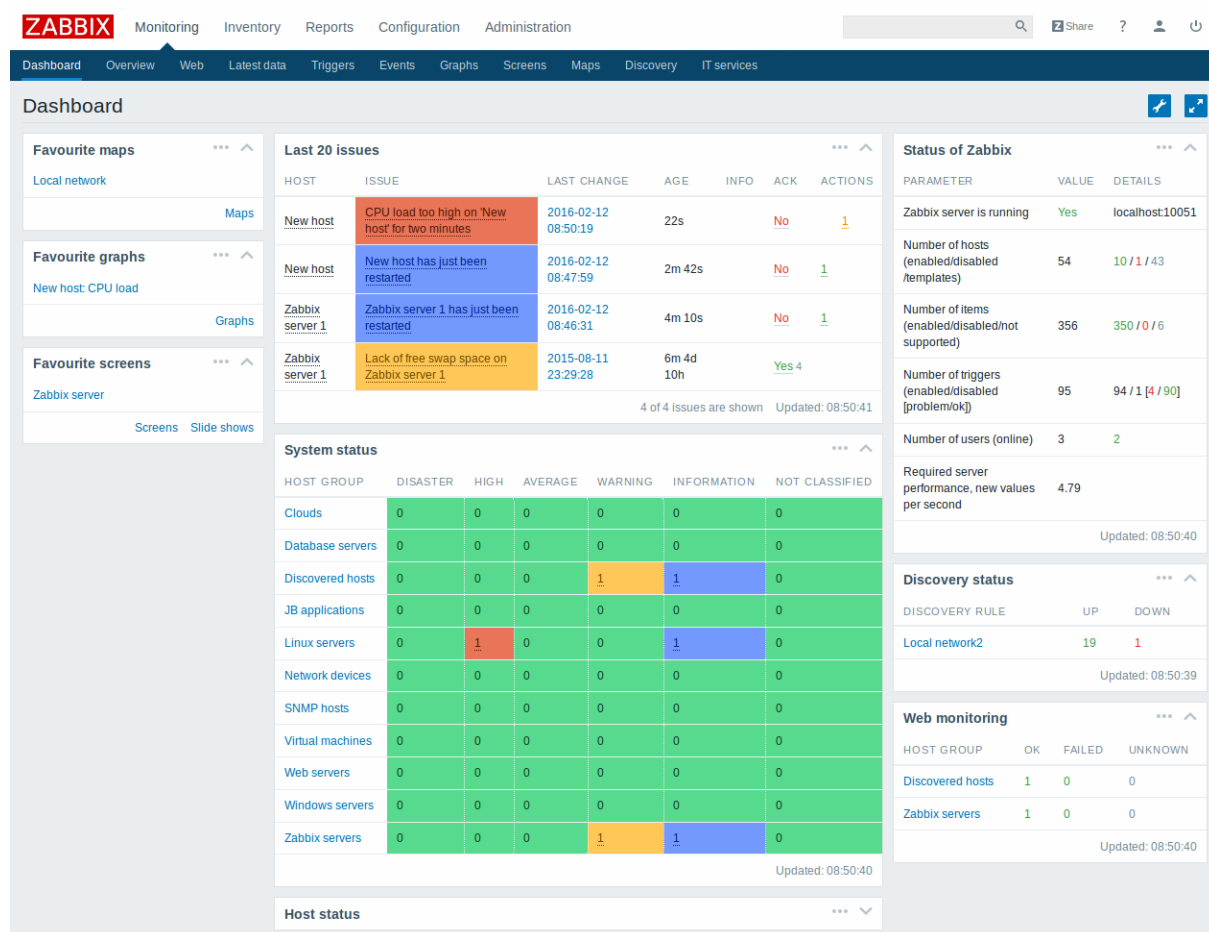


Figura 2.1: Dashboard *Zabbix*

Como podemos observar, são apresentados vários dados informativos distribuídos por vários componentes, os quais são alteráveis mediante as necessidades do utilizador. Mediante o foco principal de monitorização, poderá ser de maior interesse monitorizar aspectos referentes à quantidade de tráfego a entrar ou sair de por exemplo um servidor. Numa outra situação em que estejam a ser monitorizadas uma maior quantidade máquinas virtuais, poderá porventura ser de maior interesse ao utilizador ter um componente que apresente a utilização de CPU de cada máquina virtual.

Equipa 24/7 - Em termos estruturais, a ClaraNet encontra-se subdividida numa grande quantidade de equipas, tendo cada uma as suas prioridades e responsabilidades. As equipas 24/7, também conhecidas por equipas de primeira linha, são as que têm um primeiro contacto com todos os alarmes accionados pelos servidores que alojam servidores Zabbix. Estes tratam de tentar encontrar soluções para alarmes disparados de forma a que estes não cheguem às outras equipas, filtrando um *workload* proveniente dessa parte para que as outras equipas possam focar-se em outras áreas da empresa. Contudo, isto nem sempre é possível e por vezes apresentam-se alarmes de dificuldades superiores, cabendo à responsabilidade da equipa 24/7 de encaminhar estas situações à equipa correspondente através de plataformas internas à empresa. Por exemplo, alarmes referentes a situações de redes deverão ser encaminhados para uma equipa de redes, alarmes referentes a situações tenham a ver com máquinas virtuais deverão ser encaminhados para uma equipa de virtualização e *storage*. É de mencionar que esta divisão de equipas presentes na empresa, foi um conceito de alta relevância no desenho da solução desenvolvida. O foco destas equipas deverá ser unicamente os eventos que apareçam no seu ecrã e, por tal, não deverão preocupar-me com a manutenção e administração da plataforma. Será então necessária a administração da plataforma, proveniente de utilizadores com outras responsabilidades.

3

Abordagem

Neste capítulo é referenciada a abordagem tomada para o desenvolvimento da aplicação web. Na secção 3.1 é feito um levantamento das reuniões semanais realizadas com a equipa da ClaraNet que acompanhou o projeto e na secção 3.2 é apresentada a solução proposta.

3.1 Reuniões ClaraNet

Numa primeira fase, foram realizadas reuniões com a ClaraNet com o intuito de ser feita a apresentação da empresa, às ferramentas de monitorização e dos membros da empresa que iriam acompanhar no desenvolvimento deste projeto. Foi definido que seriam realizadas reuniões semanais curtas de forma a que houvesse um maior acompanhamento, especialmente por parte do Engenheiro Hugo José, e esclarecimento de dúvidas quanto ao que seria necessário desenvolver.

Seguidamente, foi necessário chegar a um acordo quanto às tecnologias a serem utilizadas no desenvolvimento do projeto. Neste caso foi necessário chegar a um consenso, nomeadamente na definição de tecnologias do *Back-end*, devido ao facto de este ser um projeto que poderá ser usado e desenvolvido posteriormente pela ClaraNet, tornou-se necessário utilizar uma linguagem que esteja em conformidade com os trabalhadores da empresa.

Numa terceira fase, requisitaram-se duas máquinas virtuais, uma para alocação da web-app e outra para a alocação de um servidor *Zabbix*. A aquisição destas mesmas

máquinas acabou por ser algo demorada, principalmente por questões de segurança da empresa. Numa primeira fase foram entregues as máquinas, sendo a única camada de proteção existente, as credenciais de acesso às próprias máquinas. Contudo, isto apresentava vulnerabilidades tais que poderiam apresentar perigo para a ClaraNet. Consequentemente, o acesso foi retirado por um curto espaço de tempo, sendo adicionado posteriormente como medida de segurança, o acesso restrito ao endereço *IP* onde o projeto estaria a ser desenvolvido. Note-se, que estas medidas de segurança foram realizadas para que apenas os *IPs* que se encontravam *white-listed* do lado da empresa, tivessem a possibilidade de realizar pedidos ao servidor *Zabbix* e interagir com o mesmo.

Por fim, ao longo do mês de agosto, continuaram a ser realizadas as reuniões semanais. Por esta altura, a solução começou a ganhar forma e por tal começaram a surgir ideias que foram alterando o desenho inicial previsto da web-app. Estas alterações foram realizadas com o intuito de ser entregue um protótipo de um produto final, sempre com os requisitos e *input* proveniente pela equipa que acompanhou o processo em mente, para o qual, porventura, poderia continuar a ser desenvolvido pela empresa.

3.2 Solução proposta

Formulou-se a ideia do desenvolvimento de uma single page web-app, representada pela Figura 3.1.

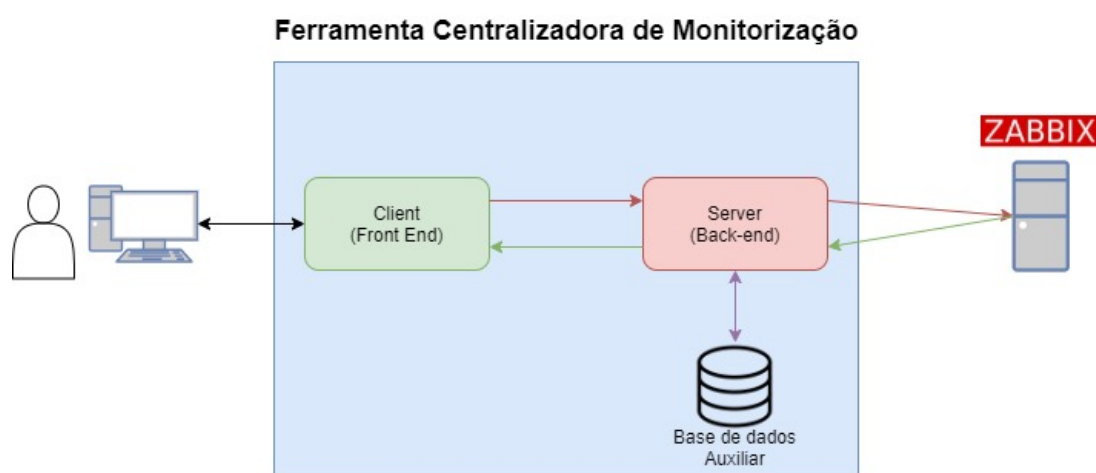


Figura 3.1: Estrutura do Projeto

Tal como representado pela Figura 3.1, foi desenvolvida uma componente *Front-end* e uma componente *Back-end* com o auxílio de uma base de dados relacional. Para o *front-end* decidiu-se utilizar a biblioteca *JavaScript*, *React*¹[7]. Utilizou-se esta biblioteca pela sua simplicidade e forte capacidade de construir uma componente *client-side* e igualmente por utilizar a linguagem *JavaScript*, sendo uma linguagem à qual a ClaraNet é familiar. Em termos de *back-end*, utilizou-se a *framework Spring-boot*²[8], desenvolvida em *Java*. Originalmente, foi proposto que a componente *server-side* fosse desenvolvida na linguagem *Kotlin*, contudo, devido ao facto de ser uma linguagem que é pouco familiar à ClaraNet, foi acordado que o desenvolvimento deveria ser realizado em *Java*. Por fim, foi desenvolvida uma base de dados auxiliar desenvolvida em *MariaDB*³[4], que estará associada maioritariamente com o tratamento de utilizadores, autenticação, conexões e associação de utilizadores com conexões.

A utilização da web-app e enquadramento da mesma na empresa, deriva de dois tipos de utilizadores: *Standard Users*, os quais seriam utilizados pelas equipas 24/7 incorporadas na ClaraNet e responsáveis pela monitorização de eventos. E *Admin Users*, utilizados por utilizadores que tenham acesso a informações que permitam à configuração de conexões de servidores *Zabbix* e porventura a outras ferramentas de monitorização.

O funcionamento da web-app irá basear-se na interação entre estes dois tipos de utilizadores em que um *Standard User* (ou *Admin User*, pois estes também têm acesso às listas de eventos provenientes de conexões associadas ao seu utilizador), apenas terá acesso às conexões que forem configuradas e associadas ao seu utilizador por parte de um *admin user*. Para ser adicionada uma nova conexão⁴ teria que ser requisitado por parte de um *standard user* a um *admin user* para que a mesma fosse configurada e associada a um utilizador. Decidiu-se realizar esta divisão de utilizadores, de forma a que os *Standards Users* apenas realizem o *login* na plataforma e observem os alarmes provenientes das conexões associadas ao próprio. É descartada então, a necessidade por parte destes utilizadores, do conhecimento de dados tais como o endereço IP de onde se encontre alocada a ferramenta ou das credenciais que permitem o acesso à ferramenta. Ficam então os *Admin Users* encarregados de adicionar conexões ao sistema, fornecendo um endereço IP e as credenciais para aceder à ferramenta e ainda associar conexões configuradas aos utilizadores registados na plataforma web-app.

¹<https://reactjs.org/>

²<https://spring.io/projects/spring-boot>

³<https://mariadb.org/>

⁴Entende-se por conexão, uma instanciação de uma ligação *Zabbix* com a web-app.

4

Implementação

Este capítulo descreve todas as implementações realizadas nos diferentes componentes desenvolvidos ou seja a componente *Back-End*, *Front-End* e base de dados. Tendo em conta todos os conceitos, as terminologias e palavras-chave previamente apresentadas e explicadas, apresentam-se as implementações realizadas, começando pela base de dados descrita pela secção 4.1, a componente *Back-End* detalhada na secção 4.2 e finalmente na secção 4.3 é abordada a componente *Front-End*.

4.1 Base de dados

Na Figura 4.1 apresenta-se o seguinte modelo EA cujo representa a base de dados desenvolvida.

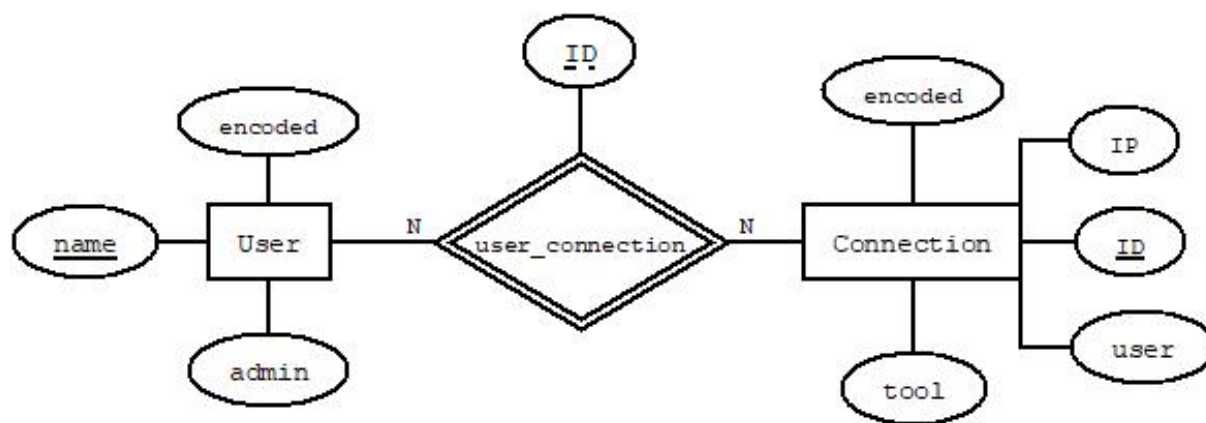


Figura 4.1: Modelo EA

Tal como previamente descrito na secção 2.2, a web-app funciona sobre o conceito de definição de tipo de *User* (*Standard* ou *Admin*), as conexões presentes no sistema e a associação entre ambos. Foram então desenvolvidas duas entidades, *User* e *Connection*, interligadas por uma relação *User_Connection* que permite associar múltiplos utilizadores a múltiplas conexões e vice-versa. Observe-se que nesta associação utilizou-se uma chave fraca denominada *ID*. Isto ocorre devido à forma como são realizadas as transacções entre a componente *Back-End* com a base de dados e será descrita na secção 4.2. Nas subsecções seguintes, são descritos os atributos de cada entidade e algum contexto sobre os mesmos.

4.1.1 User

Name - Chave primária da entidade, referencia o *Username* do utilizador. Sendo a chave primária, admite-se que apenas podem existir utilizadores com um *Username* único.

Encoded - Composto pela codificação em Base64 entre *Username:Password*, utilizado como forma de não guardar a *password* de um utilizador em *plain text*.

Admin - Atributo constituído por um valor *bit*, em que o valor 0 representa que o utilizador é um *Standard User* e o valor 1 representa que o utilizador é um *Admin User*.

4.1.2 *Connection*

ID - Chave primária da entidade, referencia um identificador único atribuído por um *Admin User* a uma conexão. A utilização deste identificador único será demonstrada nas secções seguintes.

IP - Endereço *IP* onde se encontra alocada a ferramenta de monitorização.

User - Utilizador associado à ferramenta de monitorização que permite realizar o login na ferramenta.

Encoded - Composto pela codificação em Base64 entre *Username:Password*, contudo neste caso, estas credenciais são referentes à ferramenta de monitorização. Utilizado como forma de não guardar a *password* de um utilizador em *plain text*.

Tool - Atributo que permite uma percepção facilitada de qual é a ferramenta estabelecida para esta conexão. Embora este atributo, nesta fase da solução desenvolvida, não seja utilizado, a sua ideia seria uma forma de simplificar o desenvolvimento de código na componente *Back-End* para distinguir qual a ferramenta associada à conexão.

4.2 Componente *Back-End*

A componente *Back-End* é responsável de criar uma ponte entre a componente *Front-End* com as *APIs* das ferramentas e ainda com a base de dados. Esta funciona figurativamente como um "*Man-in-the-Middle*", no sentido em que é responsável de encaminhar e processar pedidos provenientes da componente *Front-End*, comunicando com uma *API* ou base de dados para devolver uma resposta.

4.2.1 *Java Persistence API*

Com o intuito de facilitar as transacções entre a componente *Back-End* com a base de dados relacional, foi utilizada a *Java Persistence API*¹[2]. Para tal, utilizou-se a *framework hibernate*², que permite o mapeamento dos atributos entre a base de dados relacional

¹https://pt.wikipedia.org/wiki/Java_Persistence_API

²<https://pt.wikipedia.org/wiki/Hibernate>

com um objeto desenvolvido na componente *Back-End*. Como exemplo, temos o objecto *User* (presente no path: `src/main/java/G27/Central/DB/User.java`), onde através de anotações indica-se que aquele objeto representa uma entidade e indica-se o nome da tabela.

Seguidamente são realizados os construtores, variáveis que representam os atributos da entidade e métodos auxiliares. Após construída a representação em forma de objecto de uma entidade, é necessário desenvolver um repositório para realizar as transacções entre a componente *Back-End* e a base de dados.

Apresenta-se a Listagem 4.1 que representa a implementação do repositório para a entidade *User*.

```

1 public interface UserRepository extends CrudRepository<User, String> {
2     User findByEncoded(String encoded);
3     User findByName(String name);
4     User save(User user);
5     int deleteUserByName(String name);
6 }

```

Listagem 4.1: Implementação *UserRepository*

A *CrudRepository*³ é uma interface que disponibiliza operações *CRUD*⁴ genéricas, mais uma vez simplificando o processo de realização de transacções. Tal como o nome dos métodos indicam, é possível realizar uma pesquisa pelos atributos *Encoded* e *Name*, inserir um novo utilizador através do método *save* e realizar uma eliminação através do atributo *Name*. Contudo, esta metodologia de transacções apresenta as suas limitações e por vezes é necessário formular *queries*, quer seja pelo tipo de objeto que seja devolvido ou por *queries* de maior complexidade. Apresenta-se a Listagem 4.2 que representa um exemplo de uma *query* desenvolvida no repositório *User_Connection*.

```

1 @Query("select id from User_Connection where username=:username and conID
2     =:conID")
3 long queryByUserAndCon(String username, String conID);

```

Listagem 4.2: Exemplo de definição de *query*

Tal como podemos observar, desenvolveu-se uma *query* que permitisse obter o *ID* de uma *User_Connection* que tivesse o *username* e *connectionID* especificada.

³<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

⁴<https://www.auhd.site/upfiles/elibrary/Azal2020-01-22-12-28-11-76901.pdf>

4.2.2 Interface geral para conectores

Desenvolveu-se uma interface que define o que um conector terá que implementar obrigatoriamente. Podemos observar a definição da mesma através do seguinte código representado pela Listagem 4.3.

```
1 public interface Connector {  
2     void init();  
3     void destroy();  
4     boolean login(String user, String password);  
5     JSONObject call(Request request);  
6 }
```

Listagem 4.3: Código fonte para Interface Conector.

Definiram-se quatro funções básicas as quais irão permitir realizar as operações críticas para estabelecer e manter um conector com as API's das ferramentas de monitorização. O método **init** estabelece uma conexão inicial e porventura manterá a mesma. Para terminar esta conexão, deverá ser utilizado o método **destroy**. O método **login** será utilizado para realizar a autenticação às API's, recebendo um *username* e *password*, retornando um booleano de forma a sinalizar o resultado da autenticação. Por fim, o método **call** será utilizado para realizar os pedidos às API's das ferramentas, retornando um *JSONObject*, sendo este a resposta proveniente da API. A implementação destes métodos será exemplificada na secção 4.2.4.

4.2.3 Exemplo de pedido à API *Zabbix*

Apresenta-se na Listagem 4.4 um exemplo de um pedido à API *Zabbix*, neste caso um pedido de obtenção de eventos associados a um *trigger*.

```
1  {
2      "jsonrpc": "2.0",
3      "method": "event.get",
4      "params": {
5          "output": "extend",
6          "select_acknowledges": "extend",
7          "selectTags": "extend",
8          "selectSuppressionData": "extend",
9          "objectids": "13926",
10         "sortfield": ["clock", "eventid"],
11         "sortorder": "DESC"
12     },
13     "auth": "038e1d7b1735c6a5436ee9eae095879e",
14     "id": 1
15 }
```

Listagem 4.4: Exemplo de pedido JSON - Event.get.

Um pedido JSON à ferramenta *Zabbix* necessita obrigatoriamente dos seguintes parâmetros:

- **jsonrpc** - Identifica a versão do protocolo JSON-RPC⁵[3] utilizada pela API. É necessário enviar o valor **2.0** pois esta é a versão implementada pela API *Zabbix*.
- **method** - Indica o método a ser invocado na API. Neste caso está a ser realizada uma operação **event.get** que obtem determinados eventos mediante os parâmetros especificados.
- **params** - Parâmetros que serão passados ao método a ser invocado na API.
- **auth** - Referente ao *token* recebido por parte do utilizador após realizar o *login* na API.
- **id** - Identificador arbitrário do pedido.

⁵<https://www.jsonrpc.org/specification>

4.2.4 Conector *Zabbix*

Apresenta-se na Listagem 4.5 a implementação do conector *Zabbix*, através da interface geral desenvolvida, juntamente com o construtor do objecto *ZabbixConnector*.

Construtor

```
1    public ZabbixConnector(String url){
2        try {
3            uri = new URI(url.trim());
4        } catch (URISyntaxException e) {
5            throw new RuntimeException("invalid url", e);
6        }
7    }
```

Listagem 4.5: Código fonte conector *Zabbix* - Construtor.

Com o propósito de serem realizados pedidos HTTP à API do *Zabbix*, recorreu-se às bibliotecas *Apache HTTP Client*. Demonstra-se a utilização destas bibliotecas nos métodos seguintes através da Listagem 4.6 e 4.7.

Init e Destroy

```
1    @Override
2    public void init() {
3        if (httpClient == null) {
4            httpClient = HttpClients.custom().build();
5        }
6    }
```

Listagem 4.6: Código fonte conector *Zabbix* - Método Init.

```
1    public void destroy() {
2        if (httpClient != null) {
3            try {
4                httpClient.close();
5            } catch (Exception e) {
6                logger.error("close httpclient error!", e);
7            }
8        }
9    }
```

Listagem 4.7: Código fonte conector *Zabbix* - Método Destroy.

Nas duas listagens apresentadas podemos observar como é inicializada e terminada a instância do objecto que permite a realização de pedidos HTTP com a API Zabbix. Neste caso, é utilizado uma variável denominada por `httpClient` do tipo `CloseableHttpClient`, que será inicializada no método `Init` e terminada no método `Destroy`.

Call

```
1      public JSONObject call(Request request) {
2          if (request.getAuth() == null) {
3              request.setAuth(this.auth);
4          }
5
6          try {
7              HttpRequest httpRequest = org.apache.http.client.methods.
RequestBuilder.post().setUri(uri)
8                  .addHeader("Content-Type", "application/json")
9                  .setEntity(new StringEntity(JSON.toJSONString(request),
ContentType.APPLICATION_JSON)).build();
10             CloseableHttpResponse response = httpClient.execute(httpRequest
);
11             HttpEntity entity = response.getEntity();
12             byte[] data = EntityUtils.toByteArray(entity);
13             return (JSONObject) JSON.parse(data);
14         } catch (IOException e) {
15             throw new RuntimeException("ZabbixApi call exception!", e);
16         }
17     }
18 }
```

Listagem 4.8: Código fonte conector Zabbix - Método Call.

Tal como previamente mencionado, o método `call` será responsável por todos os pedidos realizados entre a web-App com a API, neste caso do *Zabbix*. Este recebe um *Request* previamente já formatado, envia o pedido em formato HTTP e devolve a resposta no formato `JSONObject`.

Login

```

1      public boolean login(String user , String password) {
2          this.auth = null;
3          Request request = RequestBuilder.newBuilder().paramEntry("user" ,
4              user).paramEntry("password" , password)
5              .method("user.login").build();
6          JSONObject response = call(request);
7          String auth = response.getString("result");
8          if (auth != null && !auth.isEmpty()) {
9              this.auth = auth;
10             return true;
11         }
12         return false;
13     }

```

Listagem 4.9: Código fonte conector Zabbix - Método Login.

Demonstra-se a utilização do método **call** através do método **login**. É então formulado um Request através dos parâmetros *user* e *password* recebidos, para posteriormente ser chamado o método **call**, para realizar o pedido de *login*.

4.2.5 Request e RequestBuilder

Após ter sido adquirido o conhecimento da estrutura de um pedido em formato *JSON* para a API do *Zabbix*, descrito na secção 4.2.3, desenvolveram-se duas classes que se encarregam de construir o pedido num formato correcto e dinamicamente, de forma a evitar repetições de código extensivas. Apresenta-se na Listagem 4.10 um exemplo da formulação de um pedido à API do *Zabbix* através da componente *Back-End* desenvolvida.

```

1      Request request = RequestBuilder.newBuilder()
2          .method("event.get").paramEntry("output" ,"extend")
3          .paramEntry("select_acknowledges" ,"extend")
4          .paramEntry("selectTags" ,"extend")
5          .paramEntry("selectSuppressionData" ,"extend")
6          .build();

```

Listagem 4.10: Exemplo de Request

Como podemos observar, podemos tomar partido dos objectos *Request* e *RequestBuilder* de forma a indicar por exemplo o tipo de método a ser invocado na API do *Zabbix*,

sendo neste caso um *"event.get"*, que permite obter os eventos presentes na instância do servidor *Zabbix*. Seguidamente, através dos métodos *paramEntry* do *RequestBuilder*, podemos adicionar múltiplos parâmetros que filtram/apresentam a resposta proveniente do *Zabbix*. Por fim, invocamos o método *Build* que retorna uma instância do tipo *Request*. Este *Request* construído será o *Request* que posteriormente será utilizado no método *Call* de um conector.

4.2.6 Atributos *Zabbix*

Apresentam-se uma série atributos associados aos três objectos principais adquiridos pela API do *Zabbix*, os *events*, os *triggers* e os *hosts*. Estes atributos são os principais campos posteriormente extraídos pela parte da componente *Front-End* e que permite a construção dos componentes do mesmo.

Event

Tabela 4.1: Propriedades do objecto Event

Propriedades	Tipo	Descrição
EventId	String	ID único de um evento
Object	Integer	Tipo de objecto associado ao evento
Object ID	String	ID único do objecto associado ao evento
Acknowledged	Integer	Sinaliza se o evento foi Acknowledged
Clock	Timestamp	Regista quando o evento foi criado
Value	Integer	Estado do objecto associado ao evento
Severity	Integer	Severidade do evento
Suppressed	Integer	Verifica se o evento encontra-se suprimido

Um evento poderá estar associado a vários tipos de objectos, neste projeto contudo, apenas será de interesse, numa primeira fase, os eventos criados por *triggers*, ou seja de uma forma geral, o campo **Object** terá o valor **0**, que corresponde devidamente a um evento criado por um *trigger*. Uma propriedade importante proveniente do Event é o campo **Acknowledged**, o qual permite saber se algum utilizador realmente já reconheceu a existência desse mesmo evento e porventura realizou algo sobre esse evento. O campo **Value**, no contexto do evento ter sido accionado por um *trigger*, poderá ter os valores **0** e **1**, representando um "Ok" ou "Problem", respetivamente. Em relação

ao campo **Severity**, poderão ser recebidos no total 6 valores diferentes: **0** que representa severidade não classificada; **1** que indica que se trata apenas de uma informação; **2** representando um aviso (severidade baixa); **3** indicando uma severidade média; **4** representando severidade alta; Por fim o valor **5**, alertando para uma severidade catastrófica.

Trigger

Tabela 4.2: Propriedades do objecto Trigger

Propriedades	Tipo	Descrição
Expression	String	Expressão reduzida do Trigger
Name	String	Nome do Trigger
Description	String	Descrição do Trigger
Comments	String	Comentários associados a um Trigger

O campo **Expression** permite que sejam criados *triggers* sobre vários componentes presentes nos *hosts* a que estejam a ser monitorizados. Apresentam-se dois exemplos de expressões que poderão ser utilizadas como trigger:

1. "expression": "{server:system.cpu.load.avg(1h)}/{server:system.cpu.load.avg(1h,1d)>2}"
2. "expression": "{www.zabbix.com:vfs.file.cksum[/etc/passwd].diff()=1}"

Na primeira expressão, está a ser comparada a carga atual de CPU de um servidor com a carga de CPU do mesmo servidor, no mesmo horário do dia anterior. Esta expressão será verdadeira se a carga da última hora for duas vezes superior à carga do dia anterior, relativamente à hora.

Na segunda expressão, é realizada uma simples verificação se o ficheiro na directoria "/etc/passwd" foi alterado. Esta expressão retorna verdadeiro quando o último valor da verificação "checksum" do ficheiro for diferente da verificação anterior.

Apesar da utilidade proveniente por parte dos *triggers* em que seria possível, por exemplo, criar um *trigger* que ao ser disparado correria um *script*, nesta fase de desenvolvimento da web-app apenas é relevante obter a propriedade *Comments*. Esta propriedade permite mostrar informação que porventura poderá ser considerada importante por quem desenvolveu o *trigger*. Devido a tal, decidiu-se que seria um atributo relevante a apresentar na componente *Front-End*.

Host

Tabela 4.3: Propriedades do objecto Host

Propriedades	Tipo	Descrição
Name	String	Nome do Host
Maintenance_status	String	Apresenta se o host encontra-se em manutenção ou não
Description	String	Descrição do Host

Um ponto relevante por parte dos hosts, para além de ser relevante apresentar o nome dos mesmos, é necessário saber se estes encontram-se em manutenção ou não. Em termos de contexto do mundo real por parte da empresa, saber se um *host* se encontra devidamente em manutenção ou não, é uma informação vital para inferir a importância de um alarme disparado. Isto é, se certos *triggers* foram accionados resultando na aparição de um evento, contudo estes provêm de um *host* que se encontra em manutenção, então este evento poderá não ter importância. O facto de um evento apresentar relevância ou não, consoante o estado de um *host*, provém pelo facto que se um *host* se encontrar em manutenção, então este estará potencialmente a ser sujeito a testes que resultam na criação de eventos, que poderão não ser relevantes exatamente pelo facto de ser apenas um teste.

4.2.7 *End-points*

Apresentam-se todos os *End-Points* definidos e possíveis de utilizar, juntamente com uma breve descrição da utilização dos mesmos:

Admin User Endpoints:

STOREDCONNECTIONS_PATH-

End-Point: /storedCon

Métodos disponíveis: GET, POST, DELETE

Descrição: Permite a obtenção de todas as conexões, registar conexões e apagar conexões sobre a base de dados.

CONNECTIONS_PATH-

End-Point: /connections

Métodos disponíveis: GET

Descrição: Permite a obtenção de todas as conexões instanciadas no sistema

Nota: Este *End-Point* não é utilizado apesar de implementado, contudo poderá ser útil justificando a sua descrição neste relatório.

USERS_PATH-

End-Point: /users

Métodos disponíveis: GET, DELETE

Descrição: Permite a obtenção de todos os utilizadores inscritos na plataforma e remoção dos mesmos, através da base de dados.

REGISTER-

End-Point: /register

Métodos disponíveis: POST

Descrição: Permite realizar a inscrição de um utilizador na plataforma, através da base de dados

CONNECTIONS_USER_PATH-

End-Point: /connections/user

Métodos disponíveis: GET,POST,DELETE

Descrição: Permite a obtenção de todas as conexões associadas a um utilizador na plataforma, juntamente com a associação ou desassociação entre um utilizador e uma conexão

Nota: O método *get* encontra-se disponível para *Standard Users* e o campo *user* representa o username de um dado utilizador.

Standard User Endpoints:**Login-**

End-Point: /login

Métodos disponíveis: POST

Descrição: Permite a autenticação de um utilizador na plataforma através das credenciais fornecidas.

HOST_PATH_ID-

End-Point: /iid/host/eid

Métodos disponíveis: GET

Descrição: Permite a obtenção das informações sobre um *host* mediante os identificadores fornecidos

Nota: O campo iid presente no *End-Point* representa o identificador único associado a uma instância *Zabbix* e o campo eid referencia-se ao ObjectID de um dado evento.

TRIGGER_PATH-

End-Point: /iid/trigger/eid

Métodos disponíveis: GET

Descrição: Permite a obtenção das informações sobre um *trigger* mediante os identificadores fornecidos

Nota: O campo iid presente no *End-Point* representa o identificador único associado a uma instância *Zabbix* e o campo eid referencia-se ao ObjectID de um dado evento.

EVENTS_PATH-

End-Point: /user/event

Métodos disponíveis: POST

Descrição: Permite a obtenção de todos os eventos das conexões associadas a um dado utilizador

Nota: O campo user presente no *End-Point* representa um dado utilizador da plataforma. Não é necessário indicar os ID's das instâncias associadas a um utilizador pois os métodos implementados que executam ao ser realizado um pedido neste *End-Point* obtêm esses mesmos ID's.

ACK_PATH-

End-Point: /iid/user/event/ack

Métodos disponíveis: POST

Descrição: Permite realizar a operação de *acknowledge* sobre um dado evento

Nota: O campo iid presente no *End-Point* representa o identificador único associado a uma instância *Zabbix* e o campo user representa um dado utilizador da plataforma. Neste caso, é necessário indicar a instância sobre qual realizar a operação.

4.2.8 Controladores

De forma a suportar os pedidos provenientes da componente *Front-End* para a componente *Back-End* foram implementados controladores que reagem ao ser efectuado um pedido a um designado *End-Point*, mediante o tipo de pedido. Apenas serão apresentados dois exemplos neste relatório, contudo, apresenta-se uma tabela indicando o uso de cada controlador.

Tabela 4.4: Controladores *Back-End*

Nome	Descrição
ConnectorController	Controlador onde são gerenciadas as instâncias às ferramentas
EventController	Controlador onde são realizadas operações relevantes aos eventos
HostController	Controlador onde são realizadas operações relevantes aos hosts
TriggerController	Controlador onde são realizadas operações relevantes aos triggers
UserController	Controlador onde são realizadas operações relevantes aos users, permitidas aos <i>Standard Users</i>
AdminConsController	Controlador onde são realizadas operações administrativas relacionadas com conexões
AdminUserController	Controlador onde são realizadas operações administrativas relacionadas aos utilizadores, juntamente com a associação ou remoção de conexões dos utilizadores.

Note-se que no *build* entregue para a defesa final do trabalho, apresentam-se outros controladores. Estes controladores foram deixados e entregues propositadamente, por apresentarem funcionalidades que outrora foram utilizados, contudo ao desenvolver do projeto deixaram de ser utilizados. Todavia, não se descarta da eventualidade da utilização desses mesmos controladores, no caso da continuação do desenvolvimento da web-app.

4.2.8.1 Controlador de Eventos

Presente na classe *EventController*, este controlador é um dos controladores *core* da web-app, exatamente por ser responsável de obter os eventos provenientes das várias instâncias associadas a um utilizador. Descreve-se seguidamente troços de código considerados importantes na designação do funcionamento dos métodos *getSpecificEvents* e *ackEvent*.

GetSpecificEvents-

Salienta-se o seguinte troço de código, representado pela Listagem 4.11, a qual representa a formulação de um pedido de eventos à API *Zabbix*.

```

1  RequestBuilder aux = RequestBuilder.newBuilder().method("event.get")
2      .paramEntry("output","extend")
3      .paramEntry("select_acknowledges","extend")
4      .paramEntry("selectTags","extend")
5      .paramEntry("selectSurpressionData","extend")
6      .paramEntry("value",1).paramEntry("sortfield","clock")
7      .paramEntry("sortorder","DESC");
8
9  if(body.containsKey("severities")){
10     if(!body.getJSONArray("severities").isEmpty())
11         aux.paramEntry("severities",body.getJSONArray("severities"));
12 }
13 if(body.containsKey("acknowledged")){
14     JSONArray acks = body.getJSONArray("acknowledged");
15     if(acks.getBoolean(0) && !acks.getBoolean(1))
16         aux.paramEntry("acknowledged",true);
17 }
```

Listagem 4.11: Request de Events

Neste pedido, destaca-se o facto de serem pedidos os eventos por ordem cronológica descendente. Ou seja, por *default* todos os eventos recebidos e que posteriormente serão apresentados, estão ordenados pelo evento mais recente ao mais antigo.

É de referir que o atributo *Value*, mencionado na tabela 4.1, tem uma importância significativa neste pedido. Observe-se que um dos parâmetros adicionados é devidamente o *Value* e o valor que se pede são os eventos que apresentam esse mesmo atributo a 1. Isto é relevante devido ao facto que mesmo sendo resolvido um evento (o atributo *Value* passa a ter um valor 0), este evento continua presente na base de dados do *Zabbix*. Ao não ser adicionado este parâmetro, estaríamos a receber eventos que já haveriam de estar resolvidos, criando ruído visual para os eventos que realmente importam.

Sublinha-se a importância do sistema de construção de *Request* dinâmicos, pois como será descrito posteriormente, será possível filtrar eventos na componente *Front-End*, resultando em pedidos que necessitam de adaptar-se mediante o pedido feito por parte de um utilizador. O mesmo é exemplificado entre as linhas 9 e 16 das listagem apresentada

Após ser realizada toda a construção de um *Request* neste método, é realizada uma operação que permite obter as conexões associadas ao utilizador que realizou este pedido e seguidamente é construído o objecto JSON que será a resposta desta componente servidora para a componente cliente.

AckEvent-

Neste método é necessário realçar como é realizado um *acknowledge* na API *Zabbix*⁶. Nomeadamente o parâmetro *action*, que indica a operação a realizar neste pedido de *acknowledge*. Este parâmetro funciona como uma *bitmask*, significando que mediante as operações realizadas desta operação de *acknowledge* é necessário adicionar o seu peso. Por exemplo, se for necessário adicionar uma mensagem que acompanha a operação de *acknowledge*, então *action* ficará com valor 4, pois este é o valor associado à mensagem. Contudo, caso seja pretendido também alterar a severidade de um evento, representado pelo valor 8 em bits, o parâmetro *action* terá que ter o valor 12 para incorporar a mensagem juntamente com a alteração de severidade. Note-se que é possível realizar uma operação *Acknowledge* de um evento, sem que este seja devidamente *acknowledged*. Isto ocorre porque poderá ser útil na eventualidade de querer ser apenas adicionada uma mensagem relevante ao evento ou alterar a severidade do mesmo ou ambos.

4.2.8.2 *Zabbix Controller*

Com o objectivo de ser possível ter a conexão simultânea com várias instâncias *Zabbix*, foi criada uma estrutura de dados *HashMap*, onde a chave será providenciada por um ID das conexões registadas na base de dados e o valor associado, será uma instância *Zabbix* dessa mesma conexão. Estas conexões presentes na base de dados são criadas no momento de registo de uma conexão por parte de um *Admin user*. Desta forma, é simplificado o processo de obtenção da instância pretendida, sendo apenas necessário fornecer o ID atribuído à instância ao método *getZab*. Apresenta-se a Listagem 4.12 que demonstra a criação de uma instância *Zabbix*:

⁶<https://www.zabbix.com/documentation/4.2/manual/api/reference/event/acknowledge>

```

1 public class ConnectorController {
2     private static ZabbixConnector zab;
3     private static HashMap<String,ZabbixConnector> connectors = new HashMap
4     <>();
5
6     public static ZabbixConnector buildConZab(String id,String ip,String
7     user, String encoded){
8         String url = "http://" + ip + "/zabbix/api_jsonrpc.php";
9         zab = new ZabbixConnector(url);
10        zab.init();
11        connectors.put(id,zab);
12        String auxPass = Encoder.decoder(encoded);
13        String logPass = auxPass.split(":")[1];
14        try{
15            boolean login = zab.login(user, logPass);
16            System.out.println("Login result= " + login);
17        } catch (Exception e){throw new ResponseStatusException(HttpStatus.
18        BAD_REQUEST,"Couldnt initialize Connection",e);}
19        return zab;
20    }
21 }

```

Listagem 4.12: Construção de Instância *Zabbix*.

Este método auxiliar *buildConZab*, recebe todos campos após ser pedido as conexões associadas a um utilizador e posteriormente os detalhes associados dessas mesmas conexões para permitir a instanciação das mesmas. Note-se como é formulado o url. Todos os pedidos realizados à API *Zabbix* através de um url, necessitam de apresentar o formato formulado para que seja possível realizar pedidos.

4.2.9 Autenticação de utilizadores

A web-app utiliza *Basic HTTP Authentication Scheme*⁷[1], viabilizando uma autenticação simples, porém eficaz. Ao ser realizado o registo na web-app, será guardado o *username* juntamente com um campo *encoded* e ainda o valor representativo de o *user* ser um *Standard User*, representado por um 0, ou um *Admin User*, representado por um 1. Apresenta-se um exemplo de codificação de um utilizador "admin" com a palavra-passe "admin":

⁷<https://datatracker.ietf.org/doc/html/rfc7617>

Exemplo:

Basic Base64(admin:admin)

Resultado:

Basic YWRtaW46YWRtaW4=

O valor *username* será mantido no *Front-end* através da *LocalStorage*, até ser realizado o *logout*, o qual irá remover esse mesmo valor presente na *LocalStorage*. É de notar que para realizar qualquer tipo de operação na web-app na componente *Front-End*, é necessário estar devidamente autenticado.

4.3 Componente *Front-End*

No que toca à componente *Front-End*, decidiu-se criar uma *User Interface* simplificada, apresentando unicamente o que realmente é relevante e de uma forma intuitiva e de compreensão fácil. Isto resume o que foi desenvolvido nesta componente em que, em vez de ter tido como foco principal uma *interface* complexa em termos estéticos, que seria consumidora em termos de tempo de desenvolvimento, focou-se no bom funcionamento e num *design* intuitivo.

Existe uma grande distinção entre tipo de utilizadores. O mesmo aplica-se nesta componente, com uma distinção ainda mais significativa quanto às páginas e funcionalidades disponíveis para cada tipo de utilizador. Ao ser acedido o *link* inicial da aplicação, o utilizador será obrigado a realizar o *login* com as suas credenciais. Para obter credenciais de acesso à plataforma, é necessário requerer a um *Admin User*, para registar um utilizador, pois apenas estes terão permissões para o mesmo. Apresenta-se em detalhe todos os componentes desenvolvidos para a componente *Front-End*, vistos a partir de um *Admin User*, devido ao facto de este ter acesso a todas as páginas da plataforma.

4.3.1 Events Page

Após ser realizado o *login*, o utilizador (independentemente do tipo de utilizador) será reencaminhado para a página principal da single-page web-app, a página dos eventos. Os eventos que são apresentados nesta página, tratam-se de todos os eventos disparados nas instâncias de conexões associadas a um dado utilizador. Na eventualidade de um utilizador não ter conexões associadas, é apresentada uma mensagem que indica que não existem eventos a apresentar. Apresenta-se na Figura 4.2 a seguinte imagem com o intuito de facilitar a explicação dos componentes que constituem nesta página.

FCM Setup Connections Configure Users **Events** Logout Logged in as: admin

Filter Events:

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

Ack:
☐ Acknowledged Events
☐ Not Acknowledged Events

Refresh events:
☐ No Refresh
☐ 5 seconds
☐ 30 seconds
☐ 1 minute

Apply Filters

Provider	Host	Problem	Info	Time	Severity	Acknowledged
Zabbix1	Zabbix server	/etc/passwd has been changed on Zabbix server	!	24/08/2021 17:28:35	Warning	No ▾
Zabbix1	TESTE🔑	High ICMP ping response time	!	11/08/2021 18:29:5	Warning	Yes ▾
Zabbix1	TESTE🔑	High ICMP ping loss	!	11/08/2021 18:29:5	Average	Yes ▾
Zabbix1	TESTE🔑	Unavailable by ICMP ping	!	11/08/2021 18:28:6	High	Yes ▾
Zabbix1	dummytest	SMTP service is down on dummytest	!	18/05/2021 18:57:40	Average	Yes ▾
Zabbix1	dummytest	NNTP service is down on dummytest	!	18/05/2021 18:57:39	Average	Yes ▾
Zabbix1	dummytest	HTTPS service is down on dummytest	!	18/05/2021 18:57:38	Average	No ▾

Figura 4.2: Página principal Events

Como podemos observar, nesta página temos 3 componentes principais sendo estas a *NavigationBar*(1), o *Filter Events*(2) e uma tabela que contém os eventos (Componente *InfoBox.js*)(3). Salienta-se o facto desta página estar disponível para os dois tipos de utilizadores tal como mencionado, porém, um *Standard User* apenas terá acesso a esta página. O mesmo é refletido através das opções disponíveis na *NavigationBar*, onde apenas terá visível as opções "Events", "Logout" e a notificação do seu *username*.

Começando pela *NavigationBar*, esta encontra-se disponível em todas as páginas da web-app. Esta permite navegar pelas várias páginas disponíveis na web-app, principalmente no caso de se tratar de um *Admin User*. Este terá acesso às páginas de administração da plataforma que serão descritas posteriormente, à página que apresenta os eventos das conexões associadas ao seu utilizador, à página para realizar *logout* e uma informação a indicar o *username* de utilizador *logged in*.

Seguidamente, o componente *Filter Events*, representado pelo quadrado vermelho e o número dois, é responsável por dar a possibilidade de um utilizador filtrar eventos mediante o pretendido e ainda por dar a possibilidade de realizar pedidos à componente *Back-End* em parcelas de tempo definidas. Um utilizador poderá filtrar eventos

mediante a sua severidade ou eventos que se encontrem *Acknowledged* ou não. Após a selecção dos filtros a aplicar, o utilizador apenas necessita de carregar no botão *Apply Filters*, para os mesmos serem devidamente aplicados.

Por fim, o componente denominado *InfoBox*, que apresenta todos os eventos das conexões associadas a um utilizador. Este componente é relativamente mais complexo pelo facto de integrar outros quatro componentes, de forma a apresentar todas as informações relevantes a cada evento. Cada evento apresentado nesta tabela, é um componente *Event*. Existem diferenças entre cada evento apresentado, nomeadamente nas parcelas *Host*, *Info* e *Acknowledged*. Expõe-se agora cada parcela presente nesta tabela detalhadamente.

Provider- Representa o ID único de uma conexão registada na plataforma. Desta forma, torna-se simples identificar de qual conexão é que um evento está associado.

Host- Indica qual o *host* associado a um dado evento. Note-se que na Figura 4.2, os *hosts* "TESTE" apresentam uma chave de fendas a cor-de-laranja. Isto indica que o *host* encontra-se em manutenção e este símbolo é um dos componentes integrados no componente principal. Se for posto o cursor sobre este símbolo, o componente irá reagir e apresentar uma pequena mensagem que indica que o dado *host* encontra-se em manutenção.

Problem- Mensagem associada ao evento indicando o problema em questão.

Info- Comentários associados ao *trigger* de um evento. Esta informação poderá ser relevante para indicar por exemplo valores atingidos, permitindo uma melhor compreensão sobre o evento disparado. Constate-se que tal como os icons apresentados nos *hosts*, este *icon* representa um componente, sendo ele o componente *EventInfo*. Quando um evento apresenta comentários associados ao seu *trigger*, é apresentado o icon em azul e é possível carregar sobre o mesmo que originará num *pop-up* com um ou mais comentários. Caso este não apresente um comentário associado, é apresentado o icon a cinzento e se for posto o cursor sobre o símbolo, o componente irá reagir e apresentar uma pequena mensagem a indicar que não existe informação associada.

Time- *Timestamp* do evento quando foi disparado. Tem-se que os eventos são apresentados por ordem cronológica decrescente, ou seja, do mais recente ao mais antigo.

Severity- Simboliza a severidade do evento.

Acknowledged- Representa se um evento foi devidamente *Acknowledged* ou não e permite realizar operações relacionadas com *acknowledge*. É possível realizar operações tais como adicionar uma mensagem ou alterar a severidade de um evento sem estar devidamente a realizar *acknowledge* sobre um evento. Os botões por si são um componente individual ao componente *Event* e representam o componente *Acknowledge*. Ao carregar sobre o botão são apresentados dois botões, um para realizar a operação de *acknowledge* e outra para observar mensagens que tenham sido adicionadas ao evento através de uma operação *acknowledge*.

4.3.2 *Setup Connections*

Esta página que encontra-se unicamente disponível para *Admin Users*, possibilita o controlo sobre todos os aspectos relativos às conexões da plataforma. Podemos verificar as conexões disponíveis no sistema, ou seja, as conexões que se encontram guardadas na base de dados e podemos igualmente adicionar ou remover conexões. É possível igualmente configurar todas as conexões associadas a um utilizador, verificando quais já se encontram devidamente associadas e adicionando ou removendo outras. Apresenta-se na figura 4.3, a representação da página descrita.

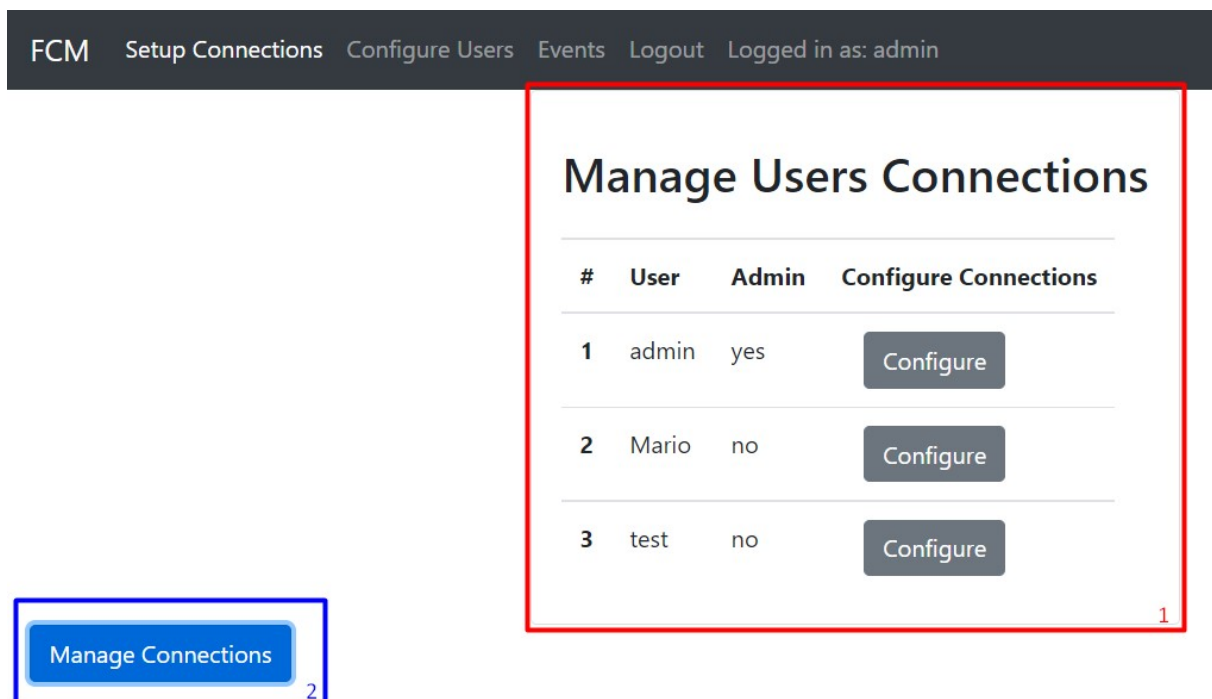


Figura 4.3: Página Setup Connections

O componente principal que constitui esta página é o componente *ConnectionManager*, englobando os componentes apresentados nesta página. Na divisão da página a vermelho(1), é apresentada uma tabela com os utilizadores registados na plataforma, juntamente com um botão "configure" que permite configurar as conexões associadas a cada utilizador. Na segunda divisão a azul(2), observa-se um botão azul a indicar "Manage Connections". Este botão permite observar todas as conexões configuradas na plataforma e ainda adicionar ou remover conexões. Exibe-se seguidamente, o que é apresentado após interagir com cada um dos botões.

4.3.2.1 *Manage Connections*

Na figura 4.4, podemos observar as configurações de conexões da plataforma.

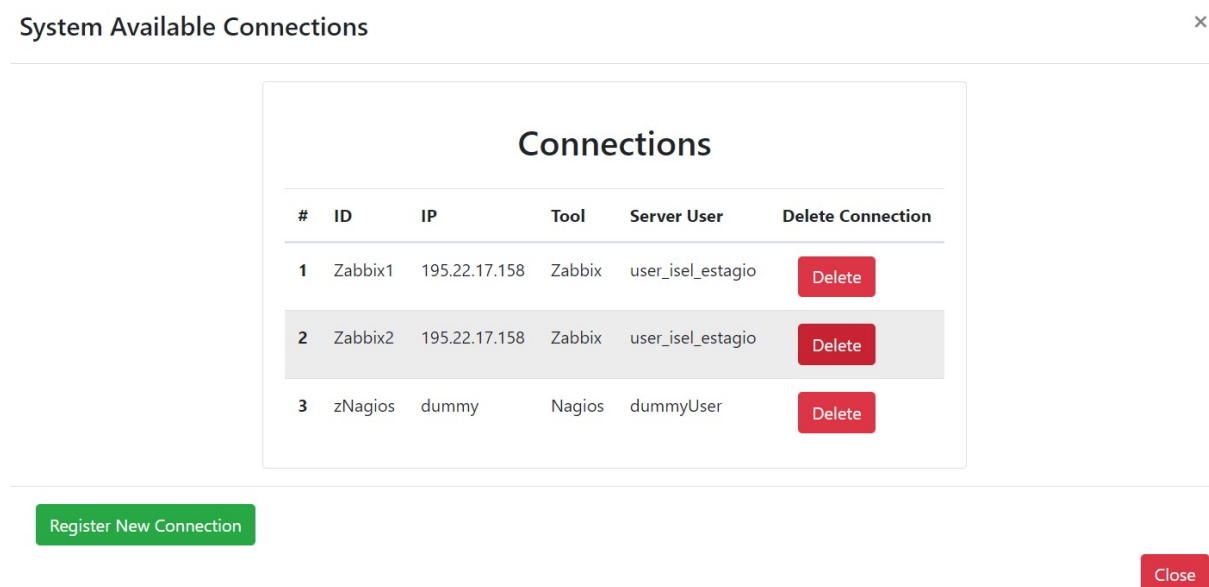


Figura 4.4: Configuração de conexões do sistema

É apresentada uma tabela com todas as conexões existentes na plataforma e a possibilidade de remover as mesmas e ainda a possibilidade de adicionar novas conexões. O componente associado é o *StoredConnections*. É de salientar que este componente recebe um parâmetro especial sendo ele o *remove*, que permite à tabela do componente reconhecer a necessidade de adicionar uma nova parcela, que neste caso é utilizada para remover conexões. Este componente *StoredConnections* é igualmente utilizado nas associações de conexões aos utilizadores, contudo sem a parcela "Delete Connection". Apresenta-se ainda o "pop-up" exibido após carregar sobre o botão "Register New Connection".

Register New Connection ×

ID

Enter ID

Give the Server an ID here

IP

Enter IP

Type Server's IP here

Server Username

Enter username

Type Server's username here

Server Password

Enter password

Type Server's password here

Select Monitoring tool

☐ Zabbix ☐ Nagios ☐ Nimsoft

Close

Confirm

Figura 4.5: Registo de nova conexão

Para registar uma nova conexão, apenas é necessário preencher o formulário representado pela figura 4.5. Primeiramente, é necessário atribuir um ID único à conexão que está a ser registada. No caso da tentativa de registo de uma conexão com um ID já existente, será apresentada uma mensagem de erro a indicar que já existe uma conexão com esse ID. Seguidamente é necessário fornecer o IP de onde se encontra alojada a ferramenta de monitorização e as credenciais de acesso à ferramenta. Por fim, é possível indicar qual é a ferramenta que esta conexão utiliza. Realça-se o facto desta última opção, nesta versão do trabalho, ser na realidade apenas demonstrativa, pois em termos de código na componente *Back-End* não é verificado esta seleção.

4.3.2.2 *Configure*

Na figura 4.6, podemos observar as configurações de conexões associadas ao utilizador admin.

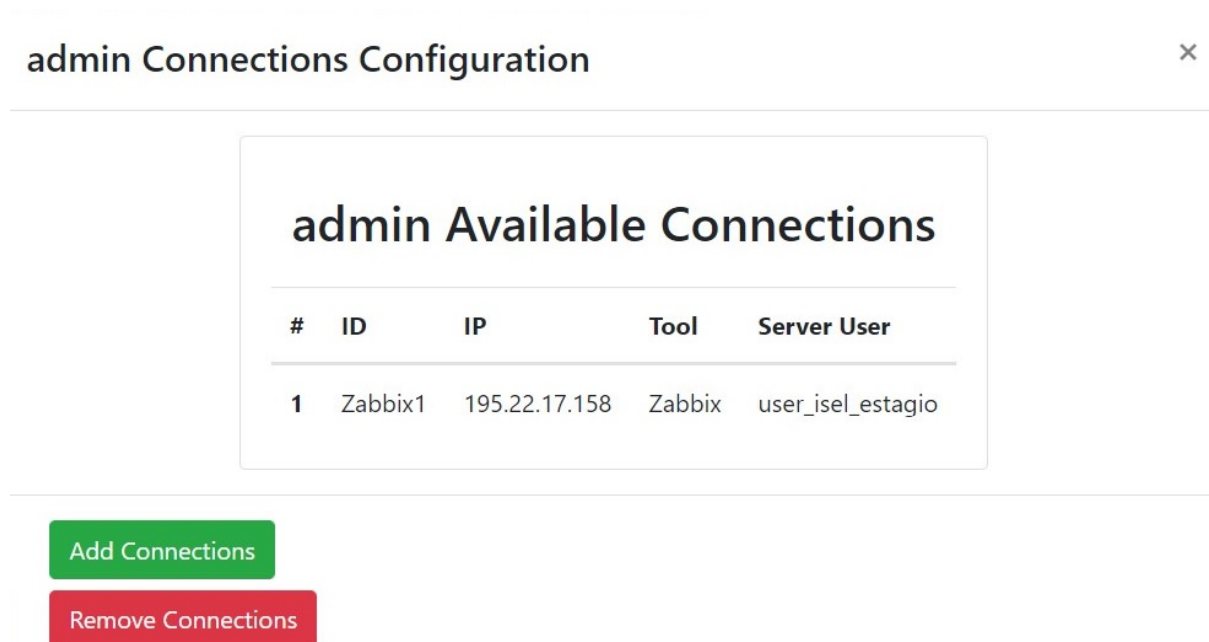


Figura 4.6: Configuração de conexões do utilizador Admin

Neste menu são apresentadas as conexões já estabelecidas com o utilizador juntamente com duas opções, uma para associar conexões presentes no sistema ao utilizador e outra para desassociar. Note-se que é possível associar e desassociar múltiplas conexões em simultâneo. O componente associado é o *ConfigureConnections*.

4.3.3 *Configure Users*

Na figura 4.7, é representada a página que permite a administração de utilizadores na plataforma.

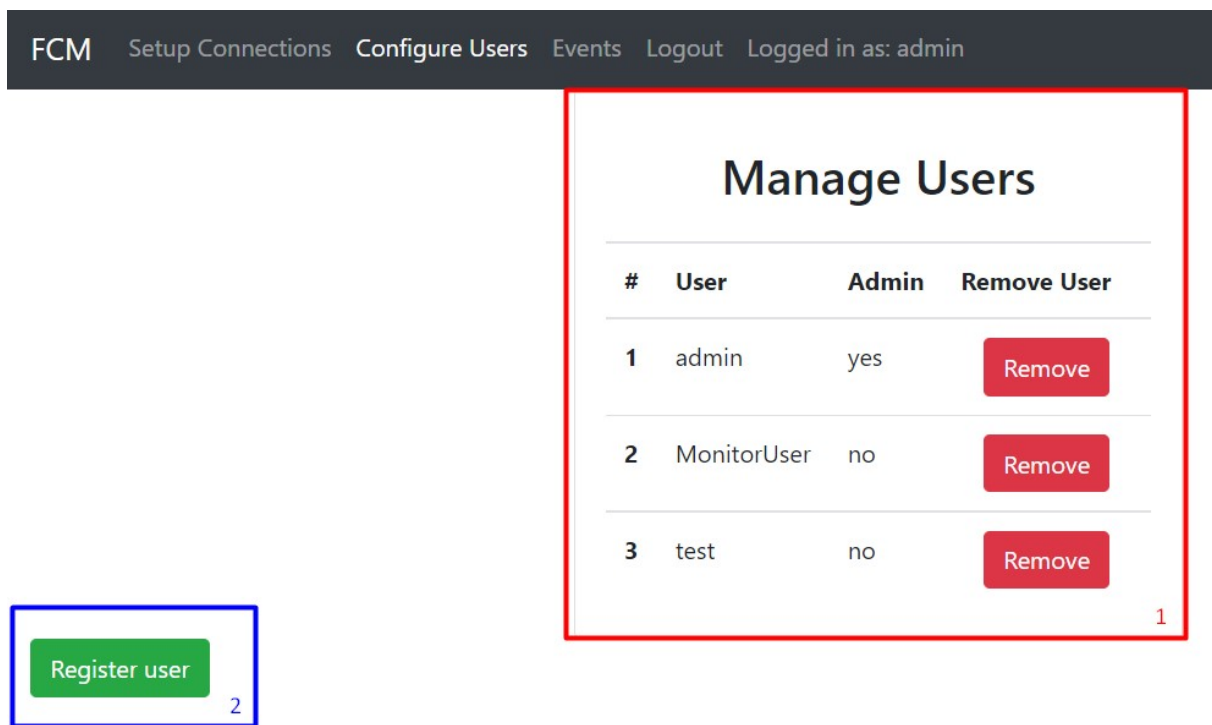
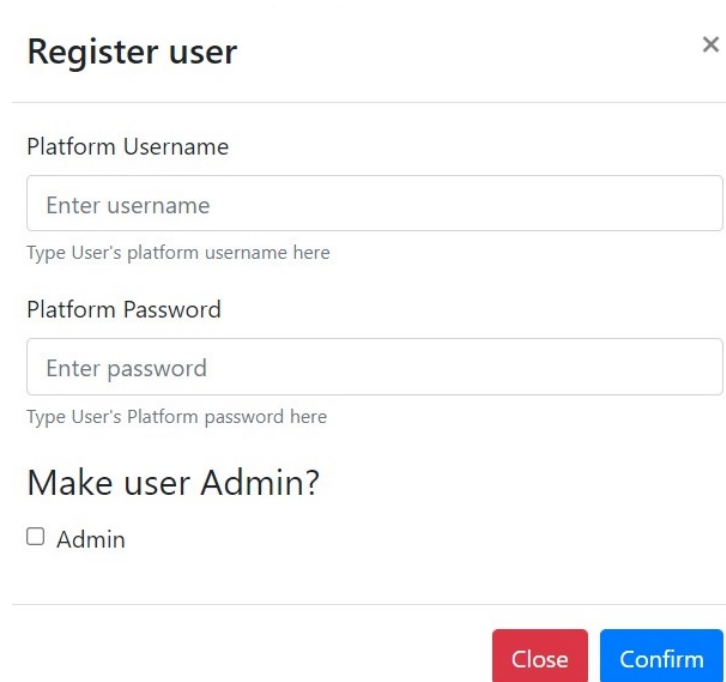


Figura 4.7: Configuração de utilizadores

Nesta página, são apresentados os utilizadores registados na plataforma, a possibilidade de remover esses mesmos utilizadores e ainda a possibilidade de registar novos. O componente que a figura representa é o *UserManager*. Apesar do formato da tabela(1) apresentada nesta página seja equivalente às tabelas previamente já apresentadas, esta encontra-se integrada dentro do próprio componente. Em termos de regras, um *Admin User* poderá remover qualquer utilizador sem restrições com a exceção de não poder remover a si próprio. Desta forma é garantida a existência de pelo menos um utilizador na plataforma. Apresenta-se seguidamente o formulário que surge ao interagir com o botão "Register User"(2).



The image shows a 'Register user' form with a title bar containing the text 'Register user' and a close button 'x'. The form contains two input fields: 'Platform Username' with a placeholder 'Enter username' and a hint 'Type User's platform username here', and 'Platform Password' with a placeholder 'Enter password' and a hint 'Type User's Platform password here'. Below these is a section titled 'Make user Admin?' with a checkbox labeled 'Admin'. At the bottom right are two buttons: 'Close' (red) and 'Confirm' (blue).

Figura 4.8: Registo de utilizadores

Como podemos observar, trata-se de um formulário simples em que são introduzidas as credenciais de acesso à plataforma, juntamente com a opção de tornar um utilizador num *Admin User* ou não. Refere-se ainda que existiu a possibilidade de existir uma espécie de "*Super Admin User*" que teria a possibilidade de criar *Admin Users*, contudo no final foi decidido que apenas seria necessário dois tipos de utilizadores por simplicidade.

Conclusão

Evidenciou-se uma imperfeição inerente às ferramentas de monitorização. A limitação da necessidade de um ecrã inteiro, de forma a conseguir interagir propriamente com uma instância de uma ferramenta de monitorização, causa a necessidade de múltiplos ecrãs de forma a monitorizar estas instâncias com eficácia. Esta limitação poderá provar-se avassaladora resultando em contratempos devido à necessidade da procura de um dado evento proveniente de uma dada instância. A solução desenvolvida, embora seja um protótipo do que poderá vir a ser, visa uma solução simples e eficaz para resolver esta situação. Observaram-se resultados positivos, onde a web-app desenvolvida faz o que foi definido como fundamental por parte da ClaraNet, ou seja, a obtenção de eventos provenientes de múltiplas instâncias *Zabbix*, definição de diferentes tipos de utilizadores e a permissão de um tipo de utilizador realizar operações sobre os outros.

As soluções dinâmicas que foram desenvolvidas neste projeto, permitem a reutilização do código para um melhoramento e continuação de desenvolvimento do projeto. O facto de ser desenvolvida uma componente *Front-End*, que embora simples, prova-se ser intuitiva, dinâmica e de compreensão fácil.

Todos os projectos apresentam aspectos menos conseguidos e este certamente não é excepção à regra. Devido à curva de aprendizagem associada às várias tecnologias associadas, certos aspectos programáticos poderiam ser melhorados, nomeadamente na componente *Front-End*. Existem alguns *bugs* conhecidos, por exemplo ao ser pedido para fazer refresh dos eventos de x em x tempo, se for seleccionada a opção "No

Refresh", é realizado um refrescamento a mais. Apesar do conhecimento da razão do acontecimento desse *bug*, no final não foi possível solucionar o mesmo, por exemplo.

Contudo, considera-se a web-app desenvolvida como algo muito benéfico para a área de monitorização, onde foi desenvolvido um projeto com a possibilidade de ser continuado internamente pela ClaraNet.

Aprendeu-se uma nova vertente associada à Engenharia Informática sob forma do mundo de monitorização, algo que previamente à realização deste projeto era desconhecido da minha parte. Compreendeu-se a importância das ferramentas de monitorização e os conceitos sobre as mesmas, juntamente com a utilização destas num contexto empresarial.

5.1 Trabalho Futuro

Apesar da grande quantidade de ideias formuladas e ponderadas, foi necessário realizar uma seleção deste leque de ideias, devido à escassez de um dos recursos mais valiosos no desenvolvimento de um projeto, sendo esse o tempo. A predominância sobre as ideias adviria das conceptualizações fundamentais que resultariam num bom conceito e desenho de uma ferramenta centralizadora de monitorização. Por tal, o que foi desenvolvido acaba por representar na realidade um protótipo/ideia de conceito sobre uma ferramenta desta natureza.

Uma vertente inicial, seria adicionar a capacidade à web-app de aceitar e conseguir processar pedidos provenientes e a outras ferramentas para além do *Zabbix*. Em termos da componente *Back-End*, a implementação seria simples, sendo necessário, numa primeira fase, aproveitar a interface conceptualizada para a integração de conexões de ferramentas. Seguidamente, seria necessário realizar pequenos ajustes ao sistema de construção de *Requests* descrito na secção 4.2.5 e possivelmente desenvolver uma classe responsável pela manutenção de todas as conexões independentemente da ferramenta. Finalmente, seria necessário definir *End-points* e desenvolver os métodos para que estes realizassem o pretendido. Em termos da componente *Front-End*, ou os ajustes seriam realizados no *Back-End* de forma a devolver uma resposta esperada, ou então o componente *InfoBox* teria que ser modificado.

Na componente *Front-End* seria para implementar mais opções de filtros de eventos, tais como filtrar por um *host* estar em manutenção ou não, filtrar por *host name* e filtrar por intervalo de tempo. Seria útil igualmente a possibilidade de ordenar os eventos mediante o gosto/necessidade do utilizador, ou seja, carregar por exemplo em cima de uma das parcelas presentes na tabela na página referente aos eventos, e ordenar

mediante a parcela carregada. Nas tabelas apresentadas, ficou por desenvolver paginação na eventualidade de haver por exemplo demasiados eventos ou um número elevado de utilizadores ou conexões no sistema, de forma a resolver a apresentação de demasiada informação de uma única vez.

Referências

- [1] *Http authentication scheme*. URL: <https://datatracker.ietf.org/doc/html/rfc7617>.
- [2] *What is java persistance api?* URL: https://pt.wikipedia.org/wiki/Java_Persistence_API.
- [3] *What is jsonrpc?* URL: <https://www.jsonrpc.org/specification/>.
- [4] *Mariadb*. URL: <https://mariadb.org/>.
- [5] *What is naggios?* URL: <https://www.nagios.org/>.
- [6] *What is nimsoft?* URL: <https://support.nimsoft.com/>.
- [7] *React*. URL: <https://reactjs.org/>.
- [8] *Spring-boot*. URL: <https://spring.io/projects/spring-boot/>.
- [9] *What is zabbix?* URL: <https://www.zabbix.com/>.
- [10] *Zabbix 4.2 documentation*. URL: <https://www.zabbix.com/documentation/4.2/manual/api/reference>.

