UNIVERSIDADE
BEIRA INTERIOR

Bigdata Correlation and Alert Notification

Afonso Pereira Figueiredo

Relatório do Projeto de Estágio
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Hugo Pedro Martins Carriço Proença
Orientador da Emvenci: Doutor Alexandre Miguel Coelho Aniceto
Coorientador da Emvenci: Engenheiro Diogo Henriques Silva

setembro de 2022

Declaração de Integridade

Eu, Afonso Pereira Figueiredo, que abaixo assino, estudante com o número de inscrição M10787 de/o Engenharia Informática da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 25/9/2022

Agradecimentos

Este projeto apenas foi conseguido com a ajuda de muitas pessoas e, por isso, quero primeiramente agradecer à minha família por todo o apoio e ajuda que me forneceram.
Quero agradecer ao Doutor Alexandre Aniceto pela disponibilidade e pela oportunidade que me permitiu crescer tanto ao longo deste percurso, ao meu orientador, professor Doutor Hugo Proença, por todo o apoio e disponibilidade fornecidos ao longo deste projeto, à Emvenci e a todos os elementos da empresa por me receberem de braços abertos e por todo o apoio que me prestaram durante o decorrer do estágio.
Quero por fim agradecer a todos aqueles que de algum modo estiveram envolvidos neste projeto.

Resumo

O presente relatório tem como objetivo expor o trabalho realizado durante o estágio na empresa Emvenci. Este incide no desenvolvimento de uma aplicação capaz de detetar problemas através da análise de *log files* e, posteriormente, enviar um alerta de modo a notificar os utilizadores do sucedido.
O documento em questão começa por descrever e apresentar a empresa, expor os motivos que levaram à necessidade de encontrar uma resolução para este problema e os objetivos da aplicação desenvolvida.
Seguidamente, é apresentada a plataforma desenvolvida pela empresa, a solução bigdata que se pretende utilizar e são expostos os requisitos e funcionalidades a atingir durante o desenvolvimento do projeto.
Posteriormente, são retratados os requisitos e funcionalidades da aplicação e a organização da base de dados, são expostas as ferramentas e tecnologias que se pretende utilizar durante todo o desenvolvimento do projeto, são apresentadas as tarefas a realizar de modo a atingir todos os objetivos desejados e é descrito o modo como a aplicação foi implementada e algumas das técnicas e métodos utilizados para alcançar este feito.
Por fim, são apresentados todos os testes realizados após o desenvolvimento da aplicação.

Palavras-chave

OpenSearch, Go, Análise de Ficheiros *Log*

Abstract

This report aims to expose the work done during the internship at Emvenci. It focuses on the development of an application capable of detecting problems through the analysis of log files and, subsequently, send an alert to notify users of such events.
The document in question begins by describing and presenting the company, exposing the motives that led to the need to find a solution to this problem and the objectives of the application developed.
Next, the platform developed by the company is presented as well as the bigdata solution that is intended to be used and the requirements and functionalities to be achieved during the development of the project.
Afterwards, the requirements and functionalities of the application and the organization of the database are portrayed, the tools and technologies that are intended to be used throughout the development of the project are exposed, the tasks performed to achieve all the desired objectives are presented and the way the application was implemented and some of the techniques and methods used to achieve this feat are described.
Finally, all the tests performed after the development of the application are presented.

Keywords

OpenSearch, Go, Log File Analysis

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

NoSQL *Not Only Structured Query Language*

HTTP *HyperText Transfer Protocol*

JSON *JavaScript Object Notation*

MOSH *Mobile Shell*

RGPD Regulamento Geral sobre a Proteção de Dados

REST *Representational State Transfer*

SaaS *Software as a Service*

API *Application Programming Interface*

AWS *Amazon Web Services*

FTP *File Transfer Protocol*

RDP *Remote Desktop Protocol*

SES *Simple Email Service*

SSH *Secure Shell*

URL *Uniform Resource Locator*

VNC *Virtual Network Computing*

Capítulo 1

Introdução

O trabalho descrito no presente relatório tem como objetivo expor a aplicação desenvolvida durante o estágio realizado na empresa Emvenci . Este primeiro capítulo descreve a empresa, os motivos que levaram à necessidade de desenvolver este tipo de *software*, os objetivos a atingir com o *software* desenvolvido e por fim a estrutura do relatório.

1.1 Emvenci

A Emvenci é uma empresa que opera a nível global na área de segurança fornecendo consultoria, serviços de auditoria e automação de processos. A empresa foi fundada em 2013 com o intuito de ajudar os clientes a mitigar os riscos e a aproveitar oportunidades. Esta desenvolve uma plataforma dividida em diferentes módulos capazes de sensibilizar os funcionários de uma empresa em questões de cibersegurança, garantir o bom funcionamento de políticas, gerir ameaças, ajudar em questões de privacidade e reportar e visualizar incidentes.

1.2 Motivação

A pandemia de COVID-19 forçou as empresas a moverem-se num sentido mais digital, apesar de esta mudança trazer vantagens muito positivas, abriu também muitas portas a ciberataques. Durante um ciberataque, muitas das plataformas existentes criam *logs* sobre ações realizadas ao longo do acontecimento. Uma análise detalhada destes dados pode levar à prevenção destes ataques. Por vezes, esta análise, quando efetuada por trabalhadores, está limitada pela sua eficiência, sendo que poderão não conseguir analisar todos os dados produzidos, levando a que a deteção destes problemas possa ser tardia e, consequentemente, levar a falhas nos serviços prestados pela empresa.
Por todas as razões listadas, o *software* a desenvolver tem como objetivo assumir controlo desta tarefa lenta e entediante.

1.3 Objetivos

O *software* desenvolvido durante o presente estágio tem como objetivo monitorizar e alertar o utilizador sobre problemas, definidos por este, que são possíveis de deduzir através da análise de *logs*. Este processo de análise é feito segundo regras de correlação ou de desvios do padrão normal, sendo estas criadas pelo utilizador seguindo padrões predefinidos. A aplicação irá comunicar com a solução bigdata por forma a obter os *logs* que necessita de processar. O processamento é feito segundo as definições da regra. Caso este leve à deteção

de um padrão indesejado, os alertas associados à regra em questão são ativados por forma a alertar o utilizador.
Numa fase mais inicial, por forma a adquirir as competências necessária para o desenvolvimento do projeto a realizar, uma pequena aplicação web foi desenvolvida. Posteriormente, de modo a ganhar algum conhecimento sobre a plataforma desenvolvida pela Emvenci, foi necessário corrigir um pequeno conjunto de *bugs* e implementar uma simples *feature* na plataforma.

1.4 Organização do Documento

Este documento encontra-se organizado da seguinte forma:

1. O primeiro capítulo - **Introdução** - apresenta a empresa, a motivação, os objetivos e a organização do documento;
2. O segundo capítulo - **Estado da Arte** - descreve a plataforma Cybersecurity Cloud, a solução bigdata e a aplicação desenvolvida;
3. O terceiro capítulo - **Engenharia de Software** - retrata os requisitos funcionais e não funcionais da aplicação, os casos de uso e a estrutura da base de dados;
4. O quarto capítulo - **Tecnologias Utilizadas** - expõe as tecnologias utilizadas durante o desenvolvimento do projeto;
5. O quinto capítulo - **Planeamento** - descreve as tarefas realizadas, tal como o seu plano de execução;
6. O sexto capítulo - **Implementação** - apresenta a estrutura da aplicação e o modo como esta foi implementada;
7. O sétimo capítulo - **Testes** - retrata todos os testes realizados após o desenvolvimento da aplicação;
8. O oitavo capítulo - **Conclusões e Trabalho Futuro** - apresenta as conclusões sobre o estágio.

Capítulo 2

Estado da Arte

2.1 Introdução

Neste capítulo é feita uma pequena introdução à plataforma disponibilizada pela Emvenci, à solução bigdata que se pretende utilizar, à aplicação desenvolvida e são ainda apresentados os requisitos e funcionalidades que se deseja implementar na mesma.

2.2 Plataforma Cybersecurity Cloud

A Emvenci desenvolve a plataforma Cybersecurity Cloud [1] como um *Software as a Service* (SaaS), esta oferece diferentes módulos capazes de ajudar os utilizadores em variadas questões de segurança.
A Plataforma segue o conceito de *multitenancy*, em que existe apenas uma instância de *software* utilizado por todos os *tenants*. Os *tenants* podem usufruir dos diferentes módulos após adquirem a subscrição relativa ao módulo desejado.
A plataforma disponibiliza os seguintes módulos:

- ***eLearning***
  O módulo eLearning fornece conteúdo *online* capaz de informar os utilizadores acerca de problemas de segurança;

- ***Phish Simulator***
  O módulo Phish Simulator permite criar campanhas de phishing seguras, de modo a testar e sensibilizar os utilizadores sobre este tipo de ataques;

- ***Privacy Manager***
  O módulo Privacy Manager possibilita aos utilizadores guardar e partilhar informação de forma segura, cumprindo com as regras de Regulamento Geral sobre a Proteção de Dados (RGPD)

- ***Policy Manager***
  O módulo Policy Manager permite aos utilizadores gerir as suas políticas, armazenadas na plataforma.

- ***Threat & Vulnerability Manager***
  O módulo Threat & Vulnerability Manager fornece a possibilidade de identificar vulnerabilidades através de uma análise ao servidor do cliente e, posteriormente, visualizar os problemas encontrados;

- **Log Analytics**
  O módulo Log Analytics permite ao utilizador ver informação que foi processada de modo a ser facilmente analisada.

2.3 Objetivos e funcionalidades da aplicação

O *software* desenvolvido tem como principal objetivo analisar dados provenientes da solução bigdata, por forma a encontrar padrões anómalos. Estes padrões anómalos são definidos por um utilizador através de regras, sendo que a informação contida nestas é o que possibilita a procura e análise da informação desejada. A análise da informação é feita segundo o tipo de regra definido, sendo que cada tipo de regra processa a informação de maneira diferente. Após este processo de análise, caso tenha sido encontrado algum padrão anómalo pelo qual se pretende procurar, os alertas associados a esta regra são ativados por forma a enviar um aviso para o utilizador acerca da anomalia ou regra que fez com que estes alertas fossem executados.
Após o processo de execução de uma regra, deve ser criado um *log*, relativo à execução atual, na solução bigdata. Em caso de falhas durante a execução de uma regra, deve também ser criado um *log* acerca da ocorrência.

2.3.1 Solução Bigdata

A solução bigdata consiste numa base de dados *Not Only Structured Query Language* (NoSQL), onde a informação é guardada como *JavaScript Object Notation* (JSON). A informação é organizada em índices e cada um destes contém um conjunto de documentos JSON. Esta é ainda indexada antes de ser armazenada. Indexação é o mecanismo que permite a este tipo de base de dados organizar e dar respostas de uma forma rápida. Este mecanismo consiste numa estrutura que mapeia o conteúdo para o documento JSON.
A comunicação com este tipo de base de dados é feita através de uma *Representational State Transfer* (REST) *Application Programming Interface* (API), onde o pedido é feito via *HyperText Transfer Protocol* (HTTP), contendo as informações sobre a operação a realizar no corpo do pedido, em formato JSON.

Este tipo de base de dados é utilizado devido à sua capacidade de resposta em tempo real. Visto que é necessário correr as regras com uma certa frequência, é necessário que a base de dados tenha capacidade de dar resposta a todos os pedidos com eficiência.
O OpenSearch [2] foi a ferramenta escolhida para desempenhar esta função. Apesar de esta ferramenta fornecer outras funcionalidades, apenas é destacada a solução bigdata, pois neste projeto não se pretende fazer uso de nenhuma outra.

2.3.2 Regras

As regras definem padrões ou anomalias que se pretendem encontrar. Estas devem correr segundo a sua frequência por forma a analisar os eventos mais recentes. Para que tudo isto

aconteça, inicialmente, durante o processo de criação de regras é necessário definir algumas variaveis:

1. ***Query***
   A *query* define a informação a retornar pela solução bigdata.

2. **Frequência de execução**
   A Frequência de execução remete para a frequência com que a regra deve correr.

3. **Tipo de Regra**
   O tipo de regra define o modo como a informação será processada.

4. **Alertas**
   Os alertas representam os métodos que se pretende utilizar de modo a alertar o utilizador caso o tipo de regra encontre o padrão que pretende.

5. **Servidor**
   O servidor determina onde se encontra a informação que se pretende processar. Este deverá alojar a solução bigdata2.3.1 que se pretende utilizar.

2.3.3 Tipos de Regras

Os tipos de regras definem como a informação será processada. Cada regra tem de necessariamente pertencer a um tipo de regra. Estes tipos de regras que serão apresentados estarão implementados por predefinição, apesar disto, será possível definir novos tipos de regras, sendo que cada novo tipo terá de seguir padrões e implementar as estruturas predefinidas.

- ***Any***
  Este tipo de regra não faz processamento, ou seja, toda a informação retornada pela solução bigdata corresponde ao que se pretende encontrar.

- ***Blacklist***
  O tipo de regra *Blacklist* verifica que a informação retornada não contem valores presentes na lista de valores proibidos. Esta regra requer uma lista de valores que se pretende negar e a chave correspondente ao parâmetro que queremos analisar.

- ***Whitelist***
  Este tipo de regra analisa os valores retornados e verifica que todos eles pertencem à lista de valores permitidos. Muito semelhante ao tipo de regra anterior, esta requer também uma lista de valores mas estes são agora permitidos, e uma chave também referente ao parâmetro que pretendemos analisar.

- ***Change***
  O tipo de regra *Change* verifica se um parâmetro foi alterado relativamente à última ocorrência. Para esta regra será apenas necessário definir o parâmetro e o período de tempo que se pretende analisar.

- ***Frequency***
  Este tipo de regra verifica se o número de ocorrências da *query* é superior ao limite definido, sendo que o número de ocorrências é analisado segundo um determinado período de tempo. Este tipo de regra requer a definição à priori do limite de ocorrências e o período de tempo que se pretende analisar.

- ***Spike***
  O tipo de regra *Spike* analisa o volume de dados que ocorreu durante um determinado período de tempo e garante que este volume não sofreu aumentos ou decréscimos de uma determinada quantidade, em relação ao ultimo período de tempo. O tipo de regra Spike requer o período de tempo que se pretende analisar e o aumento ou decréscimo pelo qual pretendemos procurar.

- ***Flatline***
  Este tipo de regra verifica se o número de eventos é inferior ao limite definido durante um determinado período de tempo, sendo necessário definir à priori o limite e o período de tempo.

2.3.4 Alertas

Os alertas são um mecanismo que permite notificar os utilizadores sobre anomalias detetadas pelas regras, devido a isto, os alertas só são acionados pelas regras quando estas encontram as anomalias pelas quais pretendem procurar.
Pretende-se implementar os seguintes alertas:

- **Cybersecurity Cloud Tasks**
  O alerta Cybersecurity Cloud Tasks comunica com a plataforma da Emvenci afim de criar uma *Task* na mesma, esta ferramenta disponibilizada pela plataforma tem como objetivo notificar o utilizador acerca deste evento.

- **Cybersecurity Cloud Incidents**
  Semelhante ao anterior, este alerta também faz uso da plataforma por forma a criar este tipo de evento na mesma, a única diferença é que este tipo de evento consiste em criar um incidente que será atribuído a uma equipa por forma a ser resolvido.

- **Cybersecurity Cloud Requests**
  O alerta Cybersecurity Cloud Requests, como os anteriores, fará uso da plataforma, de modo a notificar o utilizador, também este consiste em em criar um evento que será atribuído a uma equipa por forma a ser resolvido, a única diferença relativamente ao anterior passa pelo modo como ambos são tratados pela plataforma.

- ***Amazon Web Services* (AWS) *Simple Email Service* (SES)**
  O alerta AWS SES tem o propósito de enviar um email ao utilizador, fazendo uso do serviço de emails fornecido pela AWS.

Os três primeiros alertas estão diretamente relacionados com a plataforma Cybersecurity Cloud, deste modo, todos eles farão uso da API da plataforma por forma a acionar o alerta.

Mais uma vez, tal como nos tipos de regras, pretende-se desenvolver esta funcionalidade, de modo a ser possível criar novos alertas seguindo padrões predefinidos.

2.4 Conclusão

Este capítulo dá a conhecer o projeto desenvolvido de uma forma mais detalhada.
A solução bigdata que se pretende utilizar permite dar respostas em tempo real, isto é um fator importantíssimo devido ao facto de que o projeto em questão vai constantemente pedir dados á solução bigdata.
Apesar de o projeto em questão não estar diretamente relacionado com a plataforma desenvolvida pela Emvenci, este fará uso da plataforma através dos alertas, sendo que três destes utilizam ferramentas presentes na mesma.

Capítulo 3

Engenharia de Software

3.1 Introdução

Este capítulo apresenta a engenharia de *software* da aplicação. Inicialmente, nas secções 3.2 e 3.3, são apresentados os requisitos funcionais e não funcionais do sistema, seguidamente, as secções 3.4 e 3.5 descrevem os diagramas de casos de uso e os diagramas de sequência. Por fim, a secção 3.6 ilustra a estrutura das bases de dados.

3.2 Requisitos funcionais

Os requisitos funcionais representam as funcionalidades que um sistema fornece. Estes estão descritos na tabela 3.1.

1	O utilizador deverá poder criar, editar, visualizar e eliminar regras;
2	O utilizador deverá poder criar, editar, visualizar e eliminar alertas;
3	O utilizador deverá poder criar, editar, visualizar e eliminar servidores;
4	A aplicação deverá correr as regras automaticamente após a sua inicialização;
5	A aplicação deverá automaticamente adicionar e correr novas regras inseridas pelo utilizador;
6	A aplicação deverá automaticamente parar e remover regras apagadas pelo utilizador;
7	A aplicação deverá automaticamente parar e atualizar regras alteradas pelo utilizador;
8	A aplicação deverá automaticamente adicionar novos servidores inseridos pelo utilizador;
9	A aplicação deverá automaticamente remover servidores apagados pelo utilizador;
10	A aplicação deverá automaticamente atualizar servidores alterados pelo utilizador;
11	A aplicação deverá automaticamente adicionar novos alertas inseridos pelo utilizador;
12	A aplicação deverá automaticamente remover alertas apagados pelo utilizador;
13	A aplicação deverá automaticamente atualizar alertas alterados pelo utilizador;
14	O sistema deverá automaticamente tentar recuperar em caso de falhas durante a execução de uma regra;
15	A comunicação entre a aplicação e a plataforma Cybersecurity Cloud deverá ser feita de forma segura;

Tabela 3.1: Requisitos Funcionais.

3.3 Requisitos não funcionais

Os requisitos não funcionais referem-se a características gerais de um sistema por forma a garantir o seu sucesso. Estes são apresentados na tabela 3.2.

1	O utilizador deverá ter uma conta válida na plataforma Cybersecurity Cloud para usufruir do sistema;
2	O utilizador deverá ter a subscrição necessária para usufruir do sistema;
3	A aplicação deverá ter acesso à internet;
4	A aplicação deverá ser intuitiva e de simples utilização;

Tabela 3.2: Requisitos Não Funcionais.

3.4 Diagramas de casos de uso

Os diagramas de casos de uso ilustram todas as possíveis interações entre o sistema e o utilizador. Neste caso as interações são feitas com a plataforma Cybersecurity Cloud [1] que posteriormente comunica alterações necessárias à aplicação.

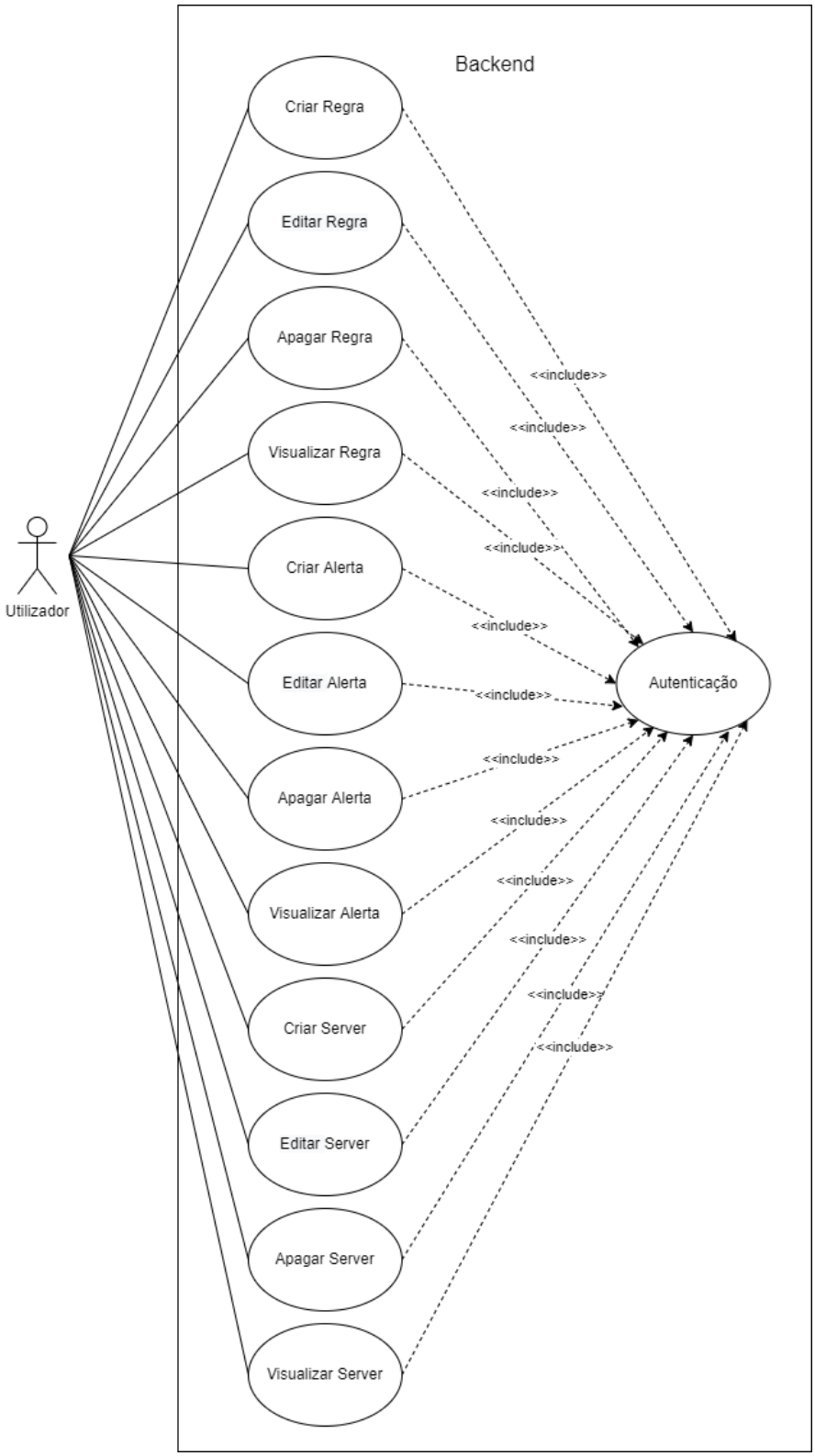

Figura 3.1: Diagrama de casos de uso de gestão de regras, servidores e alertas.

Todos os casos de uso referidos na figura 3.1, representam ações que permitem ao utilizador manipular os servidores referentes ao acesso à Solução Bigdata 2.3.1, os Alertas que definem o modo como notificar os utilizadores e as Regras que definem os padrões a procurar.

3.5 Diagramas de sequência

Os diagramas de sequência descrevem interações entre o utilizador, a aplicação e o *backend* da plataforma Cybersecurity Cloud [1]. Estes diagramas são apresentados no apêndice B

3.6 Modelação de Dados

A modelação de dados tem como objetivo esquematizar a estrutura e organização dos dados da aplicação. Na figura 3.2 é apresentado o diagrama Entidade-Associação que demonstra os diferentes objetos e as suas relações.

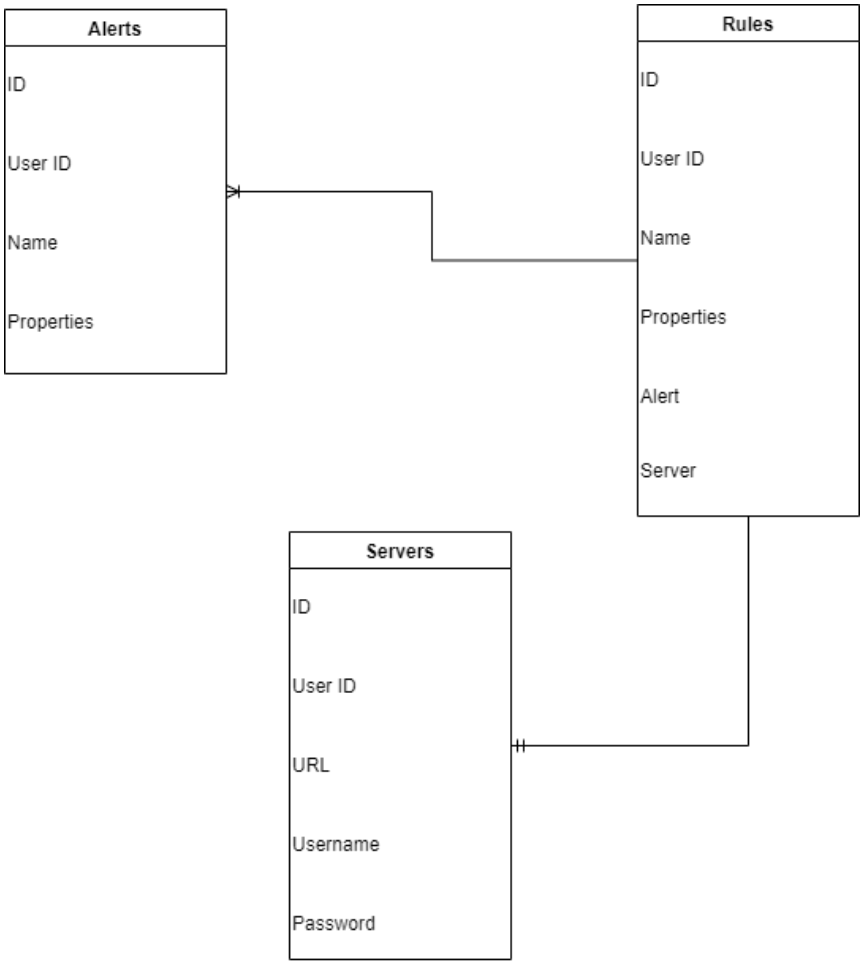

Figura 3.2: Diagrama Entidade-Associação da aplicação.

As regras representam objetos que contêm informações relacionadas com as regras executadas pela aplicação. Dentro destas conseguimos identificar o utilizador ao qual a regra pertence, o servidor e alertas associados, o nome da regra e as propriedades necessárias para a

aplicação correr a regra, sendo estas JSON.
Seguidamente, os alertas expõem a estrutura do objeto associado aos alertas que notificam o utilizador quando necessário. Estes contêm informações como o identificador do utilizador a quem pertence o alerta, o nome deste e as propriedades do alerta em formato JSON.
Por fim, os servidores simbolizam os objetos que possibilitam a ligação à solução Bigdata. Os servidores têm necessariamente de incluir um *username*, *password* e *Uniform Resource Locator* (URL) para possibilitar esta ligação e ainda o identificador do utilizador ao qual pertence o servidor.

3.7 Conclusões

Este capítulo dá a conhecer as funcionalidades do sistema e a forma como este está estruturado. São também apresentado com algum detalhe os procedimentos seguidos pela aplicação durante a sua execução e a estrutura da base de dados.

Capítulo 4

Tecnologias Utilizadas

4.1 Introdução

Neste capítulo são apresentadas e descritas todas as tecnologias utilizadas durante o desenvolvimento do projeto. Estas vão de encontro com o que todos os desenvolvedores da empresa utilizam, assim, a difusão e reutilização do código é facilitada e promovida.
Por forma a agilizar o desenvolvimento e criação de tarefas, a plataforma Jira 4.6 é uma das principais ferramentas em utilização. É aqui onde os *testers* reportam *bugs*, onde são criadas todas as tarefas e posteriormente atribuídas a um desenvolvedor.
O Visual Studio Code 4.3 é o editor de código fonte utilizado pelos desenvolvedores por forma a criar e alterar o código, posteriormente, o MobaXterm 4.5 é utilizado para aceder ao servidor remoto, a fim de testar o código produzido, garantindo o seu bom funcionamento. Por fim, quando o código está pronto a ser utilizado em produção, através do Git 4.4 as novas alteração são enviadas para o repositório, de modo a serem fundidas com o ambiente de produção.

4.2 Go

Go [3] é uma linguagem de programação desenvolvida pela Google em 2007. Esta linguagem de programação é compilada e estática, esta inclui ainda mecanismos como *garbage collector*, *memory safety* e *structural typing*.
Go é uma linguagem fortemente influenciada por C, mas mantendo um alto nível de simplicidade, por outro lado, esta linguagem foi desenvolvida numa era onde a programação concorrente era um tópico essencial, devido a isto, o desenvolvimento da linguagem focou-se bastante neste tópico, fazendo desta uma das melhores linguagens nesta área, garantindo-lhe um rápido crescimento no mercado.
Esta linguagem é utilizada exclusivamente para o desenvolvimento do *backend* da plataforma Cybersecurity Cloud.

4.3 Visual Studio Code

O Visual Studio Code [4] é um editor de código fonte desenvolvido pela Microsoft. Este editor inclui ferramentas como *debugging*, *syntax highlighting*, *intelligent code completion*, *snippets*, *code refactoring* e *embedded Git*. Este editor fornece ainda a possibilidade de instalar extensões capazes de garantir apoio para qualquer linguagem de programação, fazendo deste um dos melhores e mais fiáveis editores do mercado.

4.4 Git

O Git [5] é um sistema de controlo de versões distribuído, esta ferramenta é gratuita e utilizada para coordenar o trabalho produzido pelos desenvolvedores de *software*. Esta foi criada por Linus Torvalds em 2005, deste então, Junio Hamano tem sido responsável por manter e desenvolver o *software*.

4.5 MobaXterm

O MobaXterm [6] é um terminal que fornece ferramentas remotas de acesso a uma rede, dentro desta ferramenta podemos encontrar tecnologias como *Secure Shell* (SSH), X11, *Remote Desktop Protocol* (RDP), *Virtual Network Computing* (VNC), *File Transfer Protocol* (FTP), *Mobile Shell* (MOSH) e comandos Unix como bash, ls, cat, sed, grep, awk e rsync. Esta aplicação é utilizada com o propósito de aceder facilmente e intuitivamente ao servidor remoto, a fim de testar o código desenvolvido.

4.6 Jira

O Jira [7] é uma plataforma desenvolvida pela Atlassian que permite o manuseamento de projetos e os seus problemas. Nesta plataforma os utilizadores podem criar histórias e reportar problemas, planear *sprints* e distribuir tarefas pela equipa de desenvolvedores. Esta plataforma permite ainda criar fluxos de trabalho por forma a conseguir uma fácil adaptação à equipa de desenvolvimento.

4.7 Clean Architecture

Clean architecture é uma filosofia de desenho de *software* introduzida por Robert Cecil Martin no livro "Clean Architecture: A Craftsman's Guide to *software* Structure and Design", Esta filosofia de desenho de *software*, representada na figura 4.1, é divida em quatro partes.

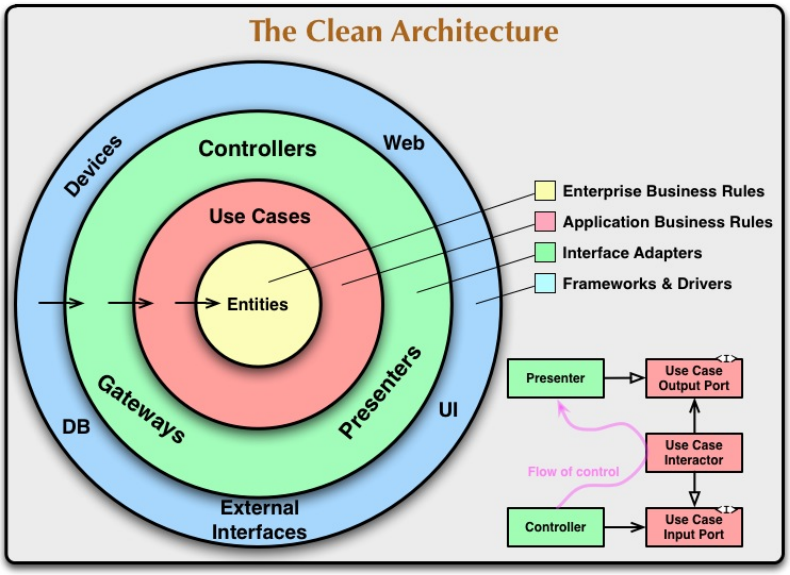

Figura 4.1: Estrutura da arquitetura Clean Architecture.

O anel interior, *Entities*, representa os objetos e estruturas de dados. Este anel diz respeito às regras e estruturas gerais utilizadas pelo *software*
O segundo anel, *Use Cases*, refere-se às regras da aplicação. Neste anel é possível encontrar a implementação de todos os casos de uso do sistema. Estes casos de uso trabalham com as entidades por forma a manipular a informação presente nestas, assim é possível atingir os objetivos dos casos de uso.
O terceiro anel, *Interface Adapters*, trata de converter a informação proveniente dos casos de uso para fontes externas como Bases de dados ou aplicações web e vice-versa
O quarto anel, *Frameworks and Drivers*, consiste em *frameworks* e ferramentas como as bases de dados, neste anel o *software* tem como objetivo aceder e fazer uso destas ferramentas externas.

4.8 Conclusão

Este capítulo dá a conhecer todas as tecnologias utilizadas durante o desenvolvimento deste projeto. Grande parte destas permitiram adquirir novos conhecimentos e capacidades que terão um impacto bastante positivo no desenvolvimento do projeto.

Capítulo 5

Planeamento

5.1 Introdução

Neste capítulo são apresentadas as tarefas realizadas durante o estágio, tal como a sua execução. Grande parte das tarefas são confidencias, devido a isto, é apenas apresentada a informação essencial.

5.2 Tarefas

As tarefas apresentadas vão de encontro com os objetivos da empresa para o estágio proposto, assim, os objetivos definidos são facilmente atingidos de uma forma estruturada.
De modo a demonstrar o planeamento proposto de uma forma cronologicamente organizada, foi criado o mapa de Gantt que se apresenta no Apêndice A. Este mapa foi desenvolvido com base nas tarefas propostas, sendo estas definidas pela seguinte ordem:

- Tarefa 1 - Onboarding, introdução e estudo da linguagem de programação Go e da arquitetura Clean Architecture;
- Tarefa 2 - Introdução à plataforma e todos os seus componentes;
- Tarefa 3 - Análise do estado da arte;
- Tarefa 4 - Planeamento e implementação das funcionalidades definidas;
- Tarefa 5 - Testes e melhorias do *software* desenvolvido;
- Tarefa 6 - Escrita do relatório de estágio.

5.3 Planificação do Trabalho

Tarefa 1 - Onboarding
Na primeira tarefa ”Onboarding”, o principal objetivo foi introduzir e integrar o estagiário na empresa de modo a garantir uma melhor adaptação a todas as tecnologias utilizadas e ao fluxo de trabalho da empresa. Para tal, foi feito um pequeno estudo sobre a linguagem de programação Go 4.2 e sobre a filosofia de desenho de *software* Clean Architecture 4.7. Por forma a praticar todos os conceitos e tecnologias estudadas foram definidos dois *milestones*:

- Desenvolver uma pequena aplicação capaz de dar resposta a pedidos provenientes dos utilizadores, tendo estes pedidos a intenção de executar operações de criação, leitura, atualização e eliminação de dados;

- A aplicação desenvolvida teve por base a estrutura Clean Architecture 4.7.

Tarefa 2 - Introdução à plataforma
Na segunda tarefa ”Introdução à plataforma”, o objetivo passou por compreender o funcionamento de todos os serviços fornecidos pela empresa, tal como estudar os módulos e componentes implementados na plataforma. Esta tarefa foi completada através de dois *milestones*.

- Corrigir *bugs* existentes na plataforma;
- Desenvolver uma nova funcionalidade para a plataforma.

As tarefas referidas anteriormente foram escolhidas com o propósito de estimular o estagiário a desenvolver conhecimento sobre os diferentes módulos da plataforma e tecnologias utilizadas pela empresa.

Tarefa 3 - Análise do estado da arte
A terceira tarefa ”Análise do estado da arte”, envolve a apresentação da plataforma Cybersecurity Cloud, da solução bigdata que se pretende utilizar e a definição dos requisitos e funcionalidades que se deseja implementar.

Tarefa 4 - Planeamento e implementação
A quarta tarefa ”Planeamento e implementação”, remete para o planeamento e implementação de todas as funcionalidades e requisitos definidos anteriormente.

Tarefa 5 - Testes e melhorias
A quinta tarefa ”Testes e melhorias”, está relacionada com os testes funcionais a realizar na aplicação desenvolvida, por forma a garantir a execução pretendida das funcionalidades implementadas. É também nesta fase onde se verifica o desempenho e se realizam os últimos ajustes e melhorias no *software* desenvolvido.

Tarefa 6 - Escrita do relatório de estágio
A sexta tarefa, ”Escrita do relatório de estágio”, remete para a escrita do relatório, sendo que esta tarefa será executada ao longo do decorrer do estágio.

5.4 Conclusão

A apresentação das tarefas a realizar durante qualquer estágio é essencial para compreender a planificação do mesmo. Deste modo é possível apresentar o fluxo de desenvolvimento que se pretende atingir, tal como os objetivos a alcançar em cada tarefa. O mapa de Gantt apresentado no Apêndice A permite visualizar a distribuição cronológica das tarefas pelas diferentes semanas relativas à duração do desenvolvimento do projeto.

Capítulo 6

Implementação

6.1 Introdução

Neste capítulo é descrita a arquitetura da aplicação, secção 6.2, seguidamente, na secção 6.3 é apresentada detalhadamente a implementação. Na implementação é descrito o modo de funcionamento da aplicação e dos seus vários componentes.

6.2 Arquitetura

A estrutura da aplicação foi definida com os colegas de trabalho segundo a arquitetura de software referida na secção 4.7. Devido a isto, seguindo os princípios estudados, a aplicação foi dividida de acordo com a seguinte figura 6.1.

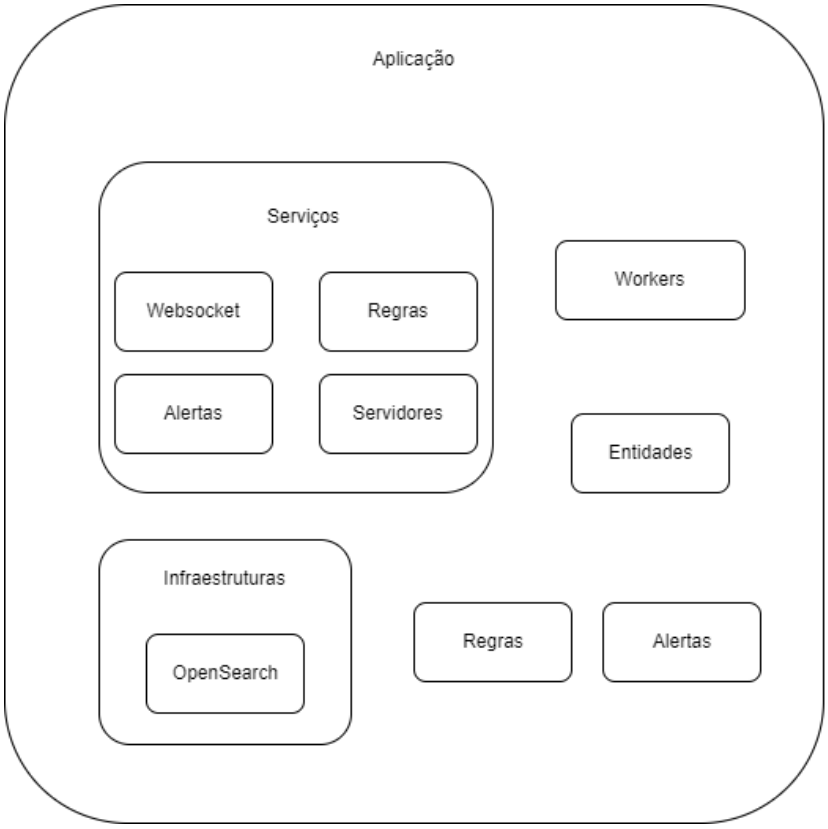

Figura 6.1: Arquitetura da aplicação.

- **Serviços** - Os serviços contêm toda a parte lógica relacionada com os casos de uso do sistema. Dentro do *websocket* podemos encontrar operações de comunicação como a receção de mensagens relacionadas com a atualização de regras, remoção de servidores

e outros.
Os restantes serviços, Regras, Alertas e Servidores, são utilizados pelo *websocket* de modo a inicializarem ou atualizarem informações relacionadas com estes.

- **Workers** - Estes representam operações cíclicas que estão sempre em funcionamento, como por exemplo a execução de regras. Após a inicialização das regras nos serviços, estas são enviadas ao *worker* para iniciarem o seu ciclo de execução.

- **Entidades** - As entidades contêm as estruturas básicas que são usadas pelos serviços. A informação proveniente do *websocket* é traduzida para estas estruturas durante a criação ou atualização de Regras, Alertas e Servidores.

- **Infraestruturas** - As infraestruturas representam objetos externos aos quais se pretende aceder. Neste caso, dentro do OpenSearch podemos encontrar operações de acesso à solução Bigdata 2.3.1.

- **Regras e Alertas** - As regras exteriores ilustram operações relacionadas com a execução das mesmas e processamento de informação. Os alertas contêm operações de comunicação, via *websocket*, para forçar a plataforma Cybersecurity Cloud [1] a lançar os alertas.

6.3 Implementação

A aplicação desenvolvida tem como objetivo ser integrada com um projeto já existente, o rGateway.
O rGateway é uma aplicação que corre do lado do cliente com o objetivo de sincronizar informações de pessoas registadas no sistema do cliente com as pessoas existentes no *backend* da plataforma Cybersecurity Cloud [1].
O rGateway faz uso de um *websocket* para permitir a comunicação com o *backend* da plataforma Cybersecurity Cloud [1]. O *websocket* é inicializado durante a execução do rGateway conforme as configurações estabelecidas. Visto que esta aplicação corre do lado do cliente, é necessário que o *websocket* seja um canal de comunicação seguro e confiável. A aplicação desenvolvida também necessita de comunicar com o *backend* da plataforma Cybersecurity Cloud [1], portanto foi feito uso deste mesmo *websocket* já implementado.

6.3.1 *Backend* da plataforma Cybersecurity Cloud

O *backend* da plataforma Cybersecurity Cloud [1] foi utilizado para armazenar todas as regras, servidores e alertas. Para tal foi necessário criar *endpoints* capazes de lidar com pedidos externos que interajam com as bases de dados de modo a criar regras, servidores e alertas. Estes pedidos necessitam de ser realizados por utilizadores da plataforma e conter informação válida que respeite as regras de validação. Quando existem alterações em regras, servidores ou alertas, a plataforma comunica estas alterações à aplicação mantendo-a atualizada.

O *backend* da plataforma Cybersecurity Cloud [1] contém ainda instruções para aceitar pedidos de execução de alertas provenientes do *websocket*. A informação proveniente do *websocket* permite acionar o alerta correspondente com informações sobre a anomalia detetada pela aplicação.
Todo o código desenvolvido no *backend* da plataforma Cybersecurity Cloud [1] seguiu também a arquitetura referida na secção 4.7.

6.3.2 Regras

As regras são o principal componente desta aplicação, visto que são estas que controlam e definem o principal fluxo do sistema. Tendo estas tamanha importância, é necessário que sejam fornecidas à aplicação de forma segura. Devido a isto, durante a inicialização da aplicação, o *backend* da plataforma Cybersecurity Cloud [1] fornece as regras via *websocket*.
Quando são feitas alterações às regras, do lado da plataforma Cybersecurity Cloud [1], esta envia todas as novas alterações, via *websocket*, por forma a que a aplicação force a paragem das regras alteradas e as atualize.
Após a criação ou atualização das regras recebidas, estas inicializam o seu ciclo de execução em *threads* de modo a garantir uma melhor performance e em caso de falhas, apenas as regras com problemas sofrem interrupções. No final de cada ciclo de execução é criado um *log*, na solução Bigdata 2.3.1, com informações sobre a data e duração da execução, um identificador da regra executada e as falhas que ocorreram durante o ciclo de execução.
As regras são definidas e guardadas na base de dados da plataforma Cybersecurity Cloud [1]. Estas têm uma representação semelhante ao excerto de código 6.1.

```
{
    "name":"example",
    "type":"Spike",
    "run_every":10,
    "time_frame":"5h",
    "limit_execution":"* * * * *",
    "properties": {
        "spike_height"2,
        "upward_spike":true,
        "downward:spike":false
    },
    "filter": [
        {
            "term": {
                "exampleTerm":"exampleValue"
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 6.1: Regra exemplo

Todas as regras necessitam obrigatoriamente de um nome, ”name”, um tipo de regra, ”type”, o intervalo entre cada ciclo de execução em minutos, ”run_every”, propriedades onde constam informações relativas ao tipo de regra, ”properties”, filtros relativos à *query* que vai ser executada pela solução Bigdata 2.3.1, ”filter”, tendo estes obrigatoriamente de seguir regras e padrões estabelecidos pela solução Bigdata 2.3.1, os alertas associados e o servidor de acesso à solução Bigdata 2.3.1.
Existem ainda parâmetros opcionais como o limite de execução, ”limit_execution”, e o período de tempo dos dados que se pretendem analisar,”time_frame”.
O limite de execução baseia-se numa expressão cron, um formato semelhante ao utilizado no *job scheduler* Cron [8], que permite limitar o ciclo de execução de uma regra. O período de tempo dos dados que se pretendem analisar é necessário em tipos de regras como *Spike*, *Frequency* e *Flatline*.

6.3.2.1 Processamento

A execução de uma regra requer primeiramente a recolha de informação presente na solução Bigdata 2.3.1.
Inicialmente a aplicação obtém a informação a processar através de *batches*, isto foi conseguido fazendo uso de uma ferramenta fornecida pelo OpenSearch [2], os Scrolls [9], estes contêm um id que permite à aplicação utilizar este id para obter o próximo *batch* de informação. Infelizmente esta técnica de processamento, *batch* a *batch*, poder-se-ia tornar lenta quando a quantidade de informação a processar fosse muito elevada. Devido a isto, para contornar este problema, optou-se por fazer uso de *threads*, onde cada *thread* processa um *batch* de informação. O OpenSearch [2] permite este tipo de operações através de Sliced Scrolls [9], que permitem utilizar várias operações de Scrolls para o mesmo pedido.

6.3.3 Alertas

Os alertas são utilizados pelas regras quando estas encontram anomalias que correspondem às definições de pesquisa estabelecidas na regra.
Estes são definidos e guardados na base de dados da plataforma Cybersecurity Cloud [1], devido a isto, são fornecidos pela plataforma via *websocket*. Os alertas recebidos são traduzidos para objetos utilizados pela aplicação de modo a serem chamados pelas regras. Quando são feitas alterações a um alerta do lado da plataforma Cybersecurity Cloud [1], esta comunica as alterações via *websocket* por forma a que a aplicação se mantenha atualizada.
Uma das principais características dos alertas é a sua capacidade de reutilização, todas as regras que utilizem o mesmo tipo de alerta fazem uso da única instância deste mesmo tipo de alerta.
Apesar da anomalia ser detetada pelas regras e o alerta executado na aplicação, a notificação que avisa o utilizador do sucedido é lançada pela plataforma Cybersecurity Cloud [1], existe um pedido por parte do alerta, via *websocket*, para forçar a plataforma a lançar a notificação para o utilizador.
A informação enviada via *websocket* depende do alerta e de como este está implementado,

de um modo geral, todos os alertas enviam uma referência relativa à regra que os chamou e uma breve descrição sobre o acontecimento, por forma a que a plataforma Cybersecurity Cloud [1] consiga notificar o utilizador sobre este acontecimento com algum detalhe.
Os alertas têm uma representação semelhante ao excerto de código 6.2.

```
{
    "type":"Email",
    "alert_text":"ExampleText!",
    "properties": {
        "subject":"ExampleSubject",
        "from_email":"ExampleEmail",
        "from_name":"ExampleName",
        "email":"ExampleEmail",
        "content":"<p>ExampleContent</p>"
    }
}
```

Excerto de Código 6.2: Alerta exemplo

Um alerta contém três propriedades principais, o tipo, uma pequena descrição acerca do alerta e as propriedades referentes ao tipo de alerta em questão. As propriedades contêm a maior parte da informação a ser enviada para o utilizador por parte da plataforma Cybersecurity Cloud [1].

6.3.4 Comunicação com a solução Bigdata

A solução Bigdata 2.3.1 é uma plataforma externa à aplicação, devido a isto, é necessário que a plataforma Cybersecurity Cloud [1] forneça os servidores que contêm as credencias de autenticação e o URL que permite aceder à solução Bigdata 2.3.1.
Os servidores são enviados via *websocket* em formato JSON. Estes podem ser representados segundo o excerto de código 6.3.

```
{
    "id":1,
    "url":"ExampleURL",
    "username":"ExampleUsername",
    "password":"ExamplePassword"
}
```

Excerto de Código 6.3: Servidor exemplo

Visto que a comunicação com a solução Bigdata 2.3.1 é essencial para o bom funcionamento da aplicação, foi implementado um sistema que, em caso de falhas, refaz o pedido um determinado número de vezes.

6.4 Conclusão

Este capítulo descreve com algum detalhe o modo de organização e estrutura da aplicação, os métodos utilizados para integrar todas as funcionalidades e as técnicas utilizadas durante a

fase de processamento de informação, dando assim a conhecer a aplicação com mais detalhe e clareza.

Capítulo 7

Testes

7.1 Introdução

Este capítulo descreve todos os testes realizados de modo a garantir o bom funcionamento da aplicação.

7.2 Testes

Os testes foram conduzidos num ambiente onde a Solução Bigdata 2.3.1, o OpenSearch [2], continha dados automaticamente gerados. A Criação destes dados consistia em aceder à plataforma do OpenSearch [2] de modo a forçar esta a gerar novos dados sobre este acesso. Devido a isto, as regras foram criadas analisando os dados da Solução Bigdata 2.3.1 por forma a conseguir garantir que o resultado produzido pela aplicação coincidia com o resultado esperado.

Os testes realizados consistiram em criar regras para a aplicação executar e posteriormente analisar o resultado produzido por esta. A análise foi feita comparando o resultado da aplicação com informação obtida da solução Bigdata 2.3.1. Devido a isto, o volume de dados utilizado durante a fase de testes foi obrigatoriamente de baixa dimensão para possibilitar uma análise mais eficiente.
Para todas as regras foram realizados testes com variações em campos como filtros, chaves de comparação e listas de valores, de modo a possibilitar à aplicação encontrar o padrão pelo qual se procura ou a garantir que esta não o deve encontrar.

7.2.1 Testes com regras tipo *Any*

A regra do tipo *Any* implica que a aplicação, dentro dos filtros estabelecidos, vai lançar o alerta associado, caso seja retornada qualquer informação pela solução Bigdata 2.3.1. Para realizar testes a este tipo de regra foram utilizadas as propriedades da regra 7.1.
Os testes realizados com a primeira regra concluíram que quando existe informação retornada pela solução Bigdata 2.3.1, a aplicação procede à execução do alerta associado, levando ao envio de um email por parte da plataforma Cybersecurity Cloud [1], correspondendo assim com os resultados esperados.
A primeira fase de testes envolveu ainda a análise do campo que permite limitar a execução de uma regra através de uma expressão cron. Os vários testes realizados, com alterações neste campo, permitiram concluir que esta funcionalidade limita com sucesso a execução da regra segundo a expressão definida.

```
{
    "name":"Test",
    "type":"Any",
    "run_every":2,
    "time_frame":"1h",
    "limit_execution":"* 23 11 * *",
    "filter": [
        {
            "term": {
                "exampleTerm":"exampleValue"
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.1: Regra do tipo Any

7.2.2 Testes com regras tipo *Blacklist*

A regra do tipo *Blacklist* garante que não existem, dentro do campo especificado valores pertencentes à *Blacklist*. Este tipo de regra foi testado utilizando a regra 7.2.

```
{
    "name":"Test2",
    "type":"Blacklist",
    "run_every":2,
    "time_frame":"1h",
    "limit_execution":"* * * * *",
    "properties": {
        "blacklist":["exampleValue","exampleValue2"],
        "comparekey":"exampleKey"
    },
    "filter": [
        {
            "query": {
                "query_string": {
                    "query": "exampleKey: exampleValue"
                }
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.2: Regra do tipo Blacklist

Os testes realizados com a segunda regra mostraram que quando o campo ”exampleKey” contém valores que estão presentes na *Blacklist*, a aplicação procede à execução do alerta que leva a plataforma Cybersecurity Cloud [1] a enviar o email indicado, o que permite concluir

o bom funcionamento do tipo de regra atual.

7.2.3 Testes com regras tipo *Whitelist*

A regra do tipo *Whitelist* verifica que só existem valores, associados ao campo especificado, pertencentes à lista de valores permitidos. De forma a testar este tipo de regra foram utilizadas as propriedades da regra 7.3.

```
{
    "name":"Test3",
    "type":"Whitelist",
    "run_every":2,
    "time_frame":"1h",
    "limit_execution":"* * * * *",
    "properties": {
        "whitelist":["exampleValue","exampleValue2"],
        "comparekey":"exampleKey"
    },
    "filter": [
        {
            "query": {
                "query_string": {
                    "query": "exampleKey: exampleValue"
                }
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.3: Regra do tipo Whitelist

Os testes realizados com esta regra permitiram deduzir que caso o campo "exampleKey" não contenha valores presentes na *Whitelist*, a aplicação recorre à execução do alerta com o objetivo de levar a plataforma Cybersecurity Cloud [1] a enviar o email correspondente, permitindo constatar o bom funcionamento deste tipo de regra.

7.2.4 Testes com regras tipo *Spike*

A regra do tipo *Spike* tem como objetivo analisar o volume de dados de um intervalo de tempo e comparar com o volume de dados prévio garantindo que este não sofreu aumentos ou decréscimos segundo a quantidade definida. De modo a garantir o bom funcionamento deste tipo de regra foram utilizadas as propriedades da regra 7.4.
Na figura 7.1 conseguimos visualizar um gráfico construído com alguns dados retirados da aplicação.

```
{
    "name":"Test4",
    "type":"Spike",
    "run_every":2,
    "time_frame":"2m",
    "limit_execution":"* * * * *",
    "properties": {
        "spike_height":2,
        "upward_spike":true,
        "downward:spike":false
    },
    "filter": [
        {
            "term": {
                "exampleTerm":"exampleValue"
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.4: Regra do tipo Spike

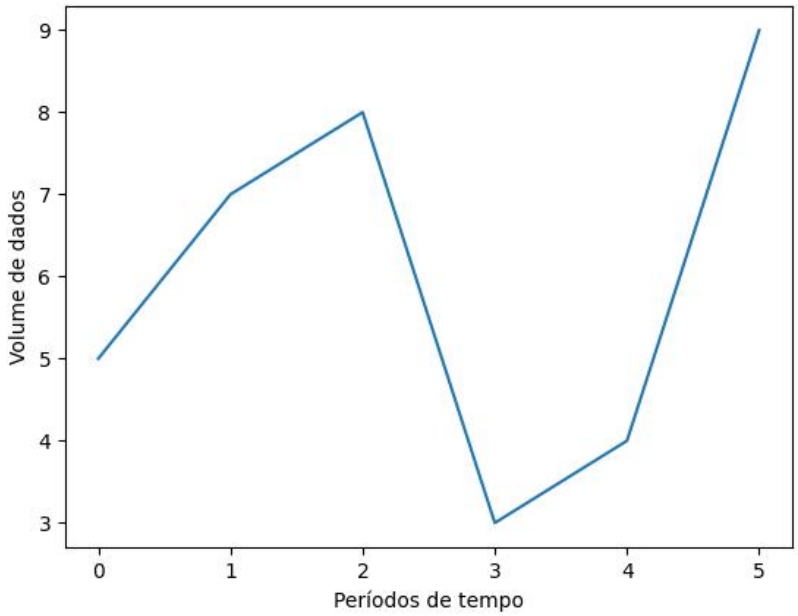

Figura 7.1: Volume de dados de uma regra *Spike* sobre vários períodos de tempo.

Segundo a figura 7.1 conseguimos verificar que entre o quarto e quinto período de tempo o volume de dados aumentou de quatro para nove, um aumento superiores a duas vezes o volume de dados do quarto período. Como a regra definida procura aumentos duas vezes superiores ao volume de dados prévio, a aplicação deteta esta anomalia e procede à execução do alerta, mais uma vez coincidindo com o resultado esperado.

7.2.5 Testes com regras tipo *Change*

A regra do tipo *Change* tem como objetivo analisar um campo e garantir que este não sofre alterações. Para testar este tipo de regra foram executadas as propriedades da regra 7.5.

```
{
    "name":"Test5",
    "type":"Change",
    "run_every":2,
    "time_frame":"10m",
    "properties": {
        "compareKey":"exampleKey"
    },
    "filter": [
        {
            "query": {
                "query_string": {
                    "query": "exampleKey: exampleValue"
                }
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.5: Regra do tipo Change

Os testes realizados com a regra anterior concluíram que esta consegue detetar alterações no campo definido e acionar o alerta por forma a notificar o utilizador, garantindo o bom funcionamento da aplicação.

7.2.6 Testes com regras tipo *Frenquency*

O tipo de regra *Frenquency* analisa o volume de dados de um intervalo de tempo e garante que este nunca é superior ao limite definido. Para realizar testes a este tipo de regra foram utilizadas as propriedades da regra 7.6.

O gráfico representado na figura 7.2 ilustra informação retirada da aplicação com a qual se conseguiu definir o volume de dados obtidos durante cinco períodos de tempo e o limite máximo para o volume de dados, representado pela linha amarela.

```
{
    "name":"Test6",
    "type":"Frenquency",
    "run_every":2,
    "time_frame":"10m",
    "properties": {
        "num_events":6
    },
    "filter": [
        {
            "term": {
                "exampleTerm":"exampleValue"
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.6: Regra do tipo Frenquency

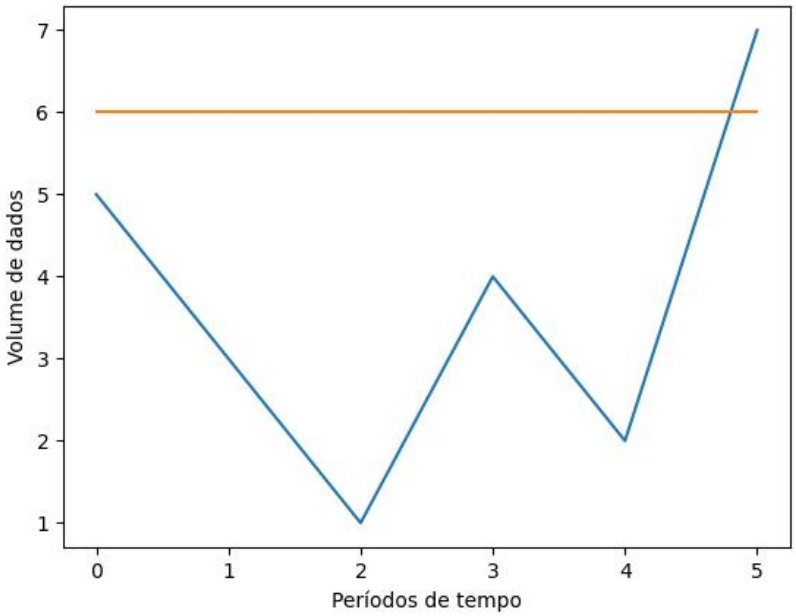

Figura 7.2: Volume de dados de uma regra *Frenquency* sobre vários períodos de tempo.

Os resultados obtidos pela aplicação demonstram que esta detetou uma anomalia no quinto período de tempo, o que levou a aplicação a recorrer à execução do alerta associado para notificar o utilizador. Após uma análise dos dados presentes na solução Bigdata 2.3.1 concluiu-se que este tipo de regra consegue detetar o tipo de anomalias pretendido com sucesso.

7.2.7 Testes com regras tipo *Flatline*

O tipo de regra *Flatline* analisa o volume de dados de um intervalo de tempo e garante que este nunca é inferior ao limite definido. Este tipo de regra foi testado utilizando a regra 7.7.

```
{
    "name":"Test7",
    "type":"Flatline",
    "run_every":2,
    "time_frame":"2m",
    "properties": {
        "threshold":2
    },
    "filter": [
        {
            "term": {
                "exampleTerm":"exampleValue"
            }
        }
    ],
    "alerts":["email"],
    "server":1
}
```

Excerto de Código 7.7: Regra do tipo Flatline

A figura 7.3 representa dados retirados da aplicação ao longo de vários períodos de tempo. É possível observar o limite definido, sendo este representado pela linha amarela.

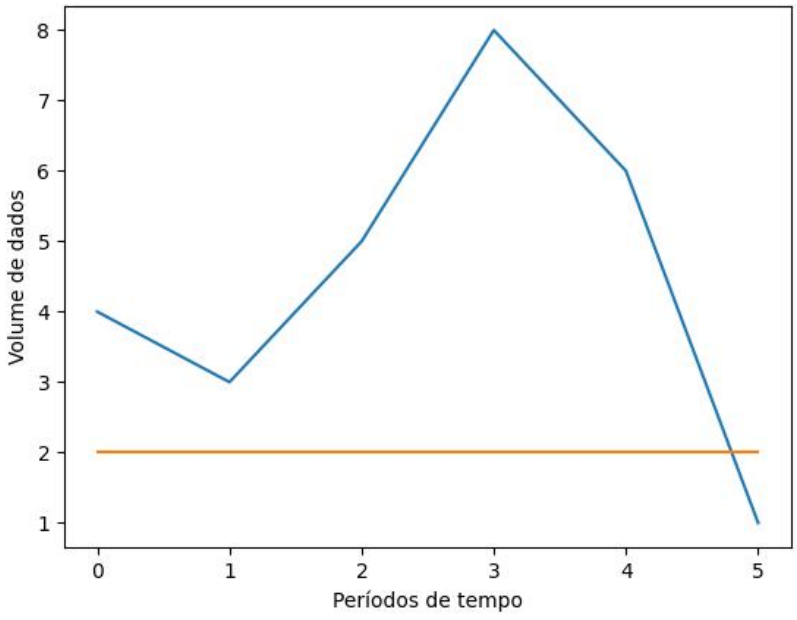

Figura 7.3: Volume de dados de uma regra *Flatline* sobre vários períodos de tempo.

Os testes realizados resultaram no envio de um email por parte da plataforma Cybersecurity Cloud [1] quando o volume de dados existente foi inferior ao limite definido. Este feito ocorreu no quinto período de tempo. Devido a isto, é possível concluir o bom funcionamento deste tipo de regra.

Capítulo 8

Conclusões e Trabalho Futuro

8.1 Conclusões Principais

Com este relatório, pretende-se dar a conhecer a empresa e a sua plataforma, os objetivos e funcionalidades do projeto e todas as tarefas realizadas durante a execução do mesmo.
O projeto em questão consistiu no desenvolvimento de uma aplicação capaz de detetar anomalias na informação fornecida, tendo esta origem em *logs* produzidos por uma plataforma. Através de regras e alertas definidos pelo utilizador, a aplicação é capaz de procurar pelos padrões anómalos pretendidos e notificar o utilizador sobre a anomalia detetada.

8.2 Trabalho Futuro

Inicialmente foram propostas funcionalidades que não vão de encontro com os princípios da linguagem GO. Pretende-se dar a possibilidade de criar tipos de regras que façam processamento de informação segundo o gosto do utilizador e alertas que enviem uma notificação conforme os requisitos do utilizador. Este tipo de funcionalidades requer a leitura de código que execute operações pretendidas pelo utilizador. A linguagem Go compila o código para ficheiros binários executáveis, o que dificulta a leitura de código externo, devido a isto, após alguma discussão com os colegas de trabalho decidiu-se deixar esta funcionalidade para outra altura.
Durante o relatório foram referidos quatro tipos de alertas, mas apenas foi implementado o alerta via email. Os restantes três requerem alguma discussão com os colegas de trabalho por forma a garantir uma implementação ideal. Neste momento, o próximo passo consiste em trabalhar com a equipa do *frontend* e designers para desenvolver a interface que irá permitir ao utilizador manusear regras, servidores e alertas com mais facilidade.

Bibliografia

[1] Emvenci. Cybersecurity cloud. [Online]. Available: https://www.cybersecurity.cloud 3, 10, 11, 20, 21, 22, 23, 25, 26, 27, 32, 39, 40, 41

[2] Amazon Web Services. Opensearch. [Online]. Available: https://opensearch.org 4, 22, 25, 39

[3] Google. Go. [Online]. Available: https://go.dev 13

[4] Microsoft. Visual studio code. [Online]. Available: https://code.visualstudio.com 13

[5] Linus Torvalds. Git. [Online]. Available: https://git-scm.com 14

[6] Mobatek. Mobaxterm. [Online]. Available: https://mobaxterm.mobatek.net 14

[7] Atlassian. Jira. [Online]. Available: https://www.atlassian.com/software/jira 14

[8] Michael Kerrisk. Cron. [Online]. Available: https://man7.org/linux/man-pages/man5/crontab.5.html 22

[9] Amazon Web Services. Scroll. [Online]. Available: https://opensearch.org/docs/1.2/opensearch/rest-api/scroll/ 22

Apêndice A

Mapa de Gantt

Este apêndice tem o objetivo de expor com mais detalhe o mapa de Gantt elaborado. Este encontra-se dividido em duas partes, as tarefas A.1 e a distribuição destas pelas diferentes semanas A.2. As tarefas aqui apresentadas vão de encontro com o que está definido no capítulo 5 nas secções 5.2 e 5.3

Tarefas	Início	FIM
Tarefa 1		
Desenvolvimento de uma pequena aplicação	11/1/21	11/15/21
Tarefa 2		
Correção de bugs	11/16/21	11/30/21
Desenvolvimento de uma funcionalidade	12/1/21	12/31/21
Tarefa 3		
Análise do estado da arte	1/1/22	1/28/22
Tarefa 4		
Planeamento de funcionalidades	1/10/22	2/3/22
Implementação das funcionalidades definidas	2/1/22	4/10/22
Tarefa 5		
Testes e melhorias	4/7/22	4/20/22

Figura A.1: Tarefas.

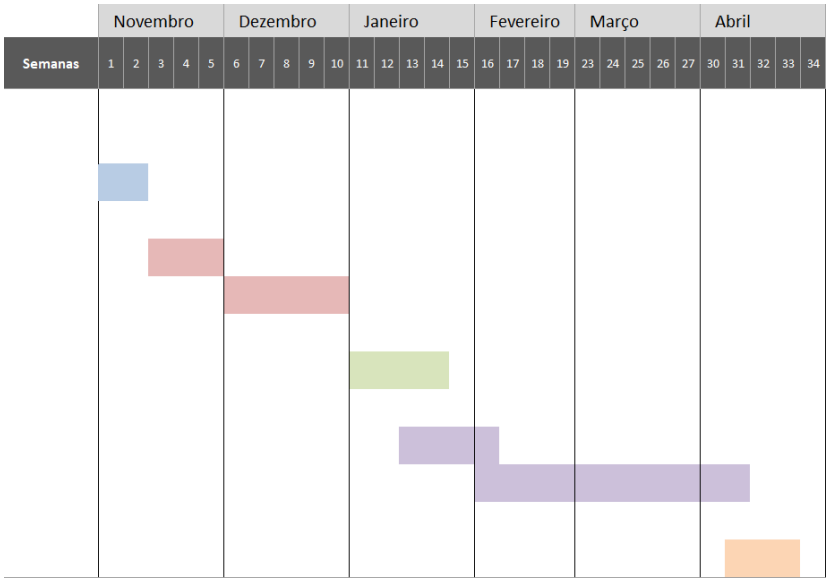

Figura A.2: Mapa de Gantt.

Apêndice B

Diagramas de sequência

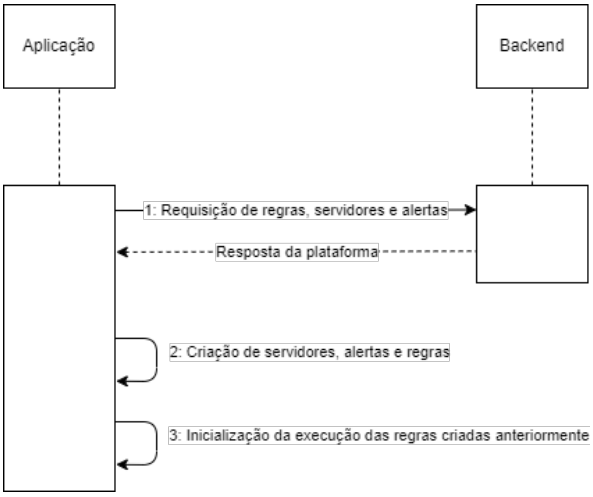

Figura B.1: Diagrama de sequência de inicialização da aplicação.

O diagrama de sequência de inicialização da aplicação, figura B.1 representa a sequência de instruções executadas pela plataforma durante o processo de inicialização. Inicialmente a aplicação realiza um pedido de modo a obter todas as regras, servidores e alertas guardados na plataforma Cybersecurity Cloud [1]. Com esta informação, a aplicação inicializa as componentes necessárias para o seu funcionamento. Por fim é inicializado o ciclo de execução das regras criadas.

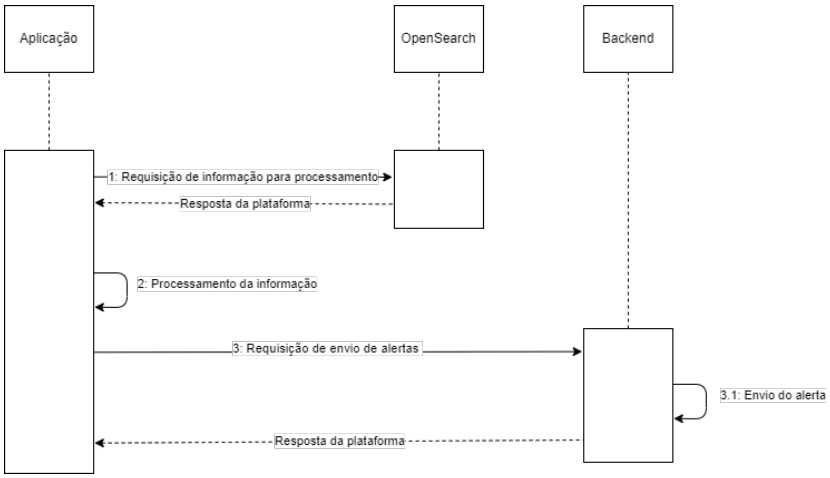

Figura B.2: Diagrama de sequência de execução de uma regra.

A figura B.2, demonstra o processo de execução de uma regra. Inicialmente, a aplicação faz um pedido ao OpenSearch [2] com o objetivo de obter a informação necessária para seguidamente processar. Caso sejam detetadas anomalias, a aplicação comunica com o *backend*

da plataforma Cybersecurity Cloud [1] por forma a que esta lance o alerta que notifica o utilizador do sucedido.

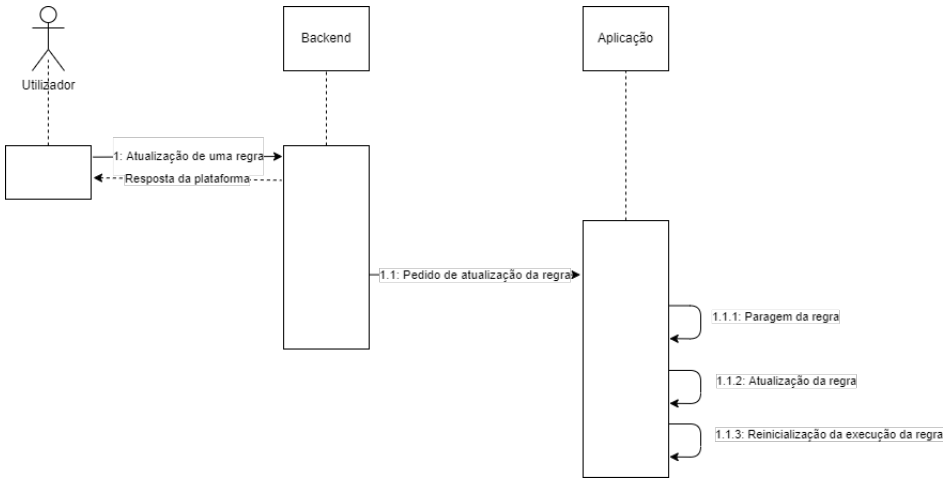

Figura B.3: Diagrama de sequência de atualização de uma regra.

O diagrama de sequência de atualização de uma regra, figura B.3, remete para o cenário em que o utilizador faz alterações em uma das regras no *backend* da plataforma Cybersecurity Cloud [1] e esta, por sua vez, comunica as novas alterações à aplicação com o objetivo de atualizar e reinicializar a execução desta regra.

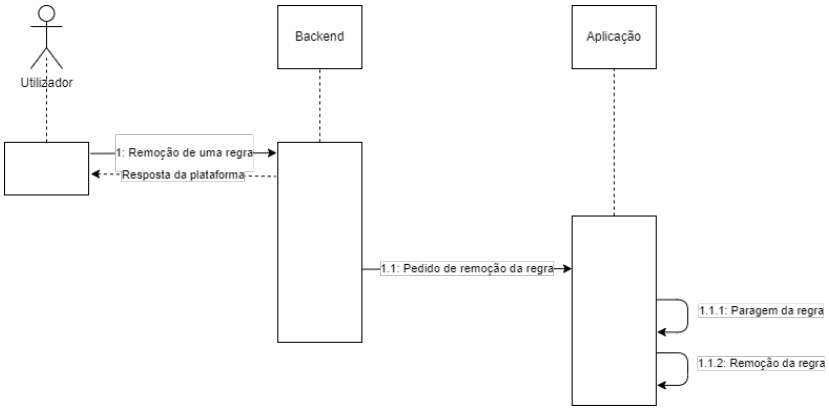

Figura B.4: Diagrama de sequência de remoção de uma regra.

O diagrama de sequência de remoção de uma regra, figura B.4, representa o processo onde o utilizador remove manualmente uma regra no *backend* da plataforma Cybersecurity Cloud [1] e esta comunica a alteração à aplicação com o intuito de parar e remover a regra.

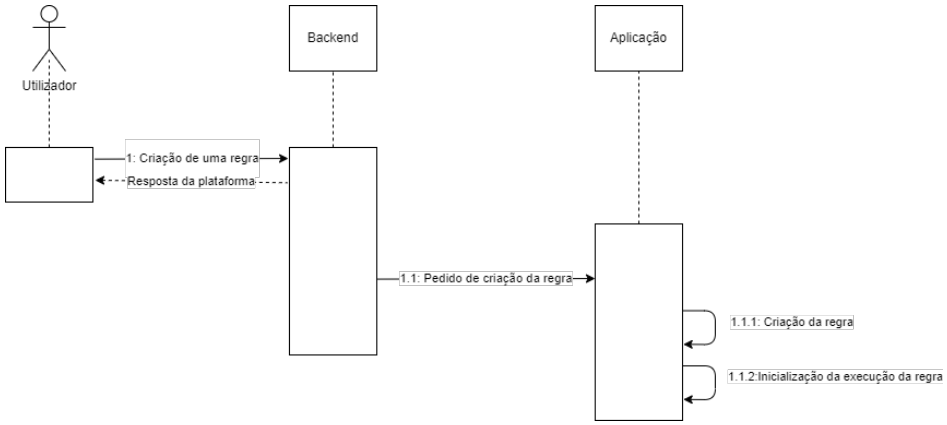

Figura B.5: Diagrama de sequência de criação de uma regra.

O diagrama de sequência de criação de uma regra, figura B.5, identifica o conjunto de ações em que o utilizador cria uma regra no *backend* da plataforma Cybersecurity Cloud [1], onde posteriormente é inicializado o pedido de criação desta nova regra na aplicação.