Mestrado em Engenharia Informática
Estágio
Relatório Final

Cybersecurity from Social Media

Ricardo Simões
rfsimoes@student.dei.uc.pt

Orientador DEI:
Professor Doutor Tiago Cruz

Orientador DEI:
Francisco Nina Rente

Juri Arguente:
Professor Doutor Ernesto Jorge Fernandes Costa

Juri Vogal:
Professor Doutor Vasco Pereira

Data: 1 de Setembro de 2016

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Resumo

Este relatório documenta o trabalho realizado pelo aluno Ricardo Filipe Reis Simões, no âmbito de Estágio em Engenharia de Software do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Neste estágio, intitulado *Cyber Security From Social Media*, o objetivo foi criar um conjunto de coletores automatizados de informação proveniente dos média sociais, por forma a alimentar a plataforma de *Cyber Security Intelligence* da Dognædis, o Portolan. Os coletores estão munidos de capacidades de pré-processamento, como extração de expressões regulares e análise textual, esta última feita por uma ferramenta externa à empresa. Foi também trabalho do estagiário a criação de uma interface de utilizador para configuração desses coletores e visualização dos dados extraídos. Neste relatório encontra-se descrita cada fase do projeto, desde o planeamento à especificação da implementação, passando pelo estudo do estado da arte com a análise de ferramentas concorrentes ferramentas de suporte ao desenvolvimento e pela especificação de requisitos que se pretende que aplicação final cumpra. Termina com as conclusões sobre o trabalho realizado e desenvolvimentos futuros.

Agradecimentos

À Dognædis a oportunidade de estágio e pela simpatia demonstrada durante este período de estágio. Quero agradecer ainda toda a ajuda disponibilizada e conhecimentos transmitidos.
Ao André Pinheiro pela ajuda na revisão deste documento.
Ao meus orientadores Francisco Nina Rente e Tiago Cruz pelas preciosas revisões deste documento, pelos seus conselhos e pelas suas rigorosas orientações.
A todos os meus amigos que me acompanharam durante este percurso.
Por fim e mais importante, quero agradecer à minha família por sempre acreditarem em mim e proporcionarem a minha formação académica.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

Anonymous *Anonymous* é uma rede internacional de entidades activistas e hacktivistas..

Autenticação Básica Autenticação básica é um método de autenticação em que o nome de utilizador e a palavra passe são passadas através da rede de forma não encriptada.

Autenticação *Digest* Autenticação *diget* é um método de autenticação em que é aplicada uma *hash* a um conjunto de argumentos, incluindo nome de utilizador e a palavra passe.

Bot Aplicação cujo objetivo é realizar operações de forma repetitiva, substituindo um humano. O âmbito deste estáfio engloba vários *bots*, desde os *crawlers* aos *bots* que recolhem dados de servidores IRC, incluindo ainda os responsáveis pela extração de padrões da informação.

Botnet uma *botnet* consiste numa rede de computadores interligados que executam tarefas em simultâneo e de forma repetitiva. No contexto da Segurança Cibernética, as *botnets* são muitas das vezes utilizadas com objetivos criminosos, com por exemplo ataques DDOS. Ainda neste contexto, esta rede é geralmente formada por computadores de utilizadores comuns, que foram infetados com vírus que permitem aos criminosos controlá-los remotamente.

Callback *Callback* é uma rotina que é passada como argumento noutra rotina, sendo esperado que esta última execute a primeira a qualquer momento no tempo que lhe seja conveniente..

CSS CSS é uma linguagem usada para descrever a apresentação de um documento HTML (ou XML).

Dark Web Subconjunto da *Deep Web*. Consequetemente também não é indexado pelos motores de busca padrão e que não se consegue aceder da forma tradicional. Encontra-se constantemente associada a atividades criminosas, como por exemplo tráfico de droga e pornografia infantil.

DDOS Um ataque distribuído de negação de serviço (em inglês, *distributed denial of service (DDOS)*) representa uma tentativa de tornar um serviço indisponível, sobrecarregando-o com tráfego proveniente de várias fontes. , 11

Deep Web Conteúdo da *World Wide Web* que não é indexado pelos motores de busca padrão.

Drill-down *Drill-down* consiste no processo de visualizar informação de forma mais detalhada, com foco em algo específico.

Expressão Regular Uma expressão regular ou *regex* é uma sequência de caracteres que define um padrão de procura. São usadas para encontrar caracteres,palavras ou padrões de caracteres. Para criar uma expressão regular é necessário usar uma linguagem formal específica que irá depois ser interpretada e usada para examinar texto e identificar partes como as que já foram mencionadas..

Framework Uma *framework* é um ambiente de *software* reutilizável que atua como uma plataforma facilitadora do desenvolvimento de aplicações, produtos ou soluções..

Hacktivismo Hacktivismo é o ato de piratear ou invadir um sistema computacional com finalidades políticas ou sociais..

Hacktivista Indivíduo que realiza actos de hacktivismo..

Handler Rotina que executa assincronamente como resposta a um evento. Um evento pode ser o pressionar de uma tecla, clique do rato, receção de dados, entre várias outras coisas..

If-Modified-Since O cabeçalho If-Modified-Since usado nos pedidos HTTP(s) é usado como um método condicional: se a página pedida não tiver sido modificada desde a data definida neste campo, a mesma não será devolvida, sendo devolvido erro 304..

Imageboard Tipo de fórum em que o conteúdo das mensagens é maioritariamente imagens.

JSON JSON é uma sintaxe usada para armazenamento e troca de dados..

Message Broker Um *Message Broker* é um programa intermediário que traduz uma mensagem do formato do remetente para o formato do recetor..

Modal O Modal é uma caixa de diálogo ou pop-up que é exibida sobre da página atual..

Polling Fazer *polling* refere-se a pedir ativamente o estado de um dispositivo externo por parte de um programa cliente..

Proxy Um *Proxy* é uma máquina que atua como um intermediário para os pedidos que um cliente faz a um servidor.

Token de Acesso Um *Token* de Acesso é uma credencial, representada sob a forma de uma cadeia de caracteres, usada para ceder a recursos protegidos. Geralmente tem associados um conjunto de permissões de acesso e tempo de expiração..

Trigger Um *trigger* é código que é executado automaticamente ou disparado quando certos eventos ocorrem numa tabela ou vista de uma base de dados..

WHATWG A WHATWG é uma comunidade de pessoas interessadas na evolução da Web, em particular HTML e APIs.

XPath XPath é usado para navegar através dos elementos e atributos de um documento XML.

Tabela de Acrónimos

API Application Programming Interface

BSD Berkeley Software Distribution

CSS Cascading Style Sheets

CPU Central Processing Unit

DDOS Distributed Denial of Service

FTP File Transfer Protocol

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

IP Internet Protocol

IRC Internet Relay Chat

JSON JavaScript Object Notation

ORM Object-Relational Mapping

MIT Massachusetts Institute of Technology

OSINT Open Source Intelligence

REST Representational State Transfer

SVG Scalable Vector Graphics

URL Uniform Resource Locator

WHATWG Web Hypertext Application Technology Working Group

XHTML EXtensible HyperText Markup Language

XML eXtensible Markup Language

Capítulo 1

Introdução

Este documento constitui o relatório de Estágio em Engenharia de Software do Departamento de Engenharia Informática da Universidade de Coimbra, levado a cabo pelo aluno Ricardo Filipe Reis Simões. Este estágio teve a duração prevista de dois semestres. No primeiro foram feitos o planeamento, o estudo do estado da arte, a especificação dos requisitos e o desenho da arquitetura a desenvolver. O segundo semestre consistiu na implementação do sistema desenhado.
Posto isto, este documento serve também para reunir todo o trabalho elaborado ao longo do estágio.

1.1 Âmbito

O termo OSINT (sigla para *Open Source Intelligence*, em português pode ser traduzido para Inteligência a partir de Fontes Abertas) refere-se à capacidade de perceber informação extraída de fontes públicas, como por exemplo jornais, revistas e a internet e guardá-la como conhecimento para ser aplicado em determinados ambientes ou contextos[1] e para ser usado para traçar cursos de ação.Contudo, não se obtém esta capacidade diretamente da extração da informação. O processo envolve também o processamento, análise e apresentação da informação. O resultado final será bastante útil, pois permitirá identificar potenciais ameaças que possam estar presentes nos média sociais. Um exemplo seria um simples *post* na página de Facebook de um grupo de hacktivistas anunciando algum tipo de ataque cibernético contra as infraestruturas de uma empresa. Além da utilização imediata da informação recolhida esta pode mais tarde ser usada para fazer análises preditivas, isto é, tentar prever acontecimentos futuros com base no passado.
Com o crescimento massivo das média sociais torna-se essencial a sua utilização como fonte de informação. É imperativo o desenvolvimento de sistemas que permitam efetuar todo o processo referido com base em informação proveniente deste tipo de fontes, para apoio à decisão orientada à Segurança Cibernética. A este tipo de inteligência chama-se *Cyber Security Intelligence* (em português, Inteligência de Segurança Cibernética) e é o centro do âmbito deste estágio.

1.2 Objetivos

Este estágio encontra-se integrado no projeto Portolan, que é um produto desenvolvido pela Dognædis. O Portolan é uma plataforma de Inteligência de Segurança Cibernética focada nas necessidades de resposta a incidentes. Este possui um conjunto de *bots* que recolhem informação das mais diversas fontes, classificando-a de acordo com uma taxonomia específica definida por um consórcio europeu de equipas de resposta a Incidentes. Esta informação é posteriormente tratada de modo a ser enriquecida. Após o enriquecimento é analisada de acordo com os parâmetros definidos pela organização, podendo despoletar alertas, alimentar outros sistemas ou simplesmente ser guardada para pesquisas futuras. A informação armazenada é depois utilizada pelas equipas de resposta e seus operadores no trabalho diário. Originalmente o Portolan incidiu essencialmente sobre informação de ativos de redes, procurando detetar ameaças e correlacionar informação transiente, cuja perda, pode dificultar a investigação pelas equipas, em fases mais adiantadas do incidente. De momento, a plataforma encontra-se a ser expandida, não só para suportar um conjunto cada vez mais vasto de fontes externas de informação de ameaças mas também para realizar a ponte entre a informação existente fora do perímetro da organização para a informação gerada pelos seus ativos. A expansão do Portolan tem sido feita não só ao nível da correlação entre informação interna e externa, mas também através do alargamento da informação recolhida. Com este último objetivo em mente têm sido implementados mecanismos de deteção de novas vulnerabilidades, fugas de informação ou no âmbito do presente trabalho, recolha e tratamento de informação proveniente dos média sociais.
Assim sendo, o objetivo do estágio passa por desenvolver um conjunto de coletores de informação, para o Portolan, de diferentes média sociais, nomeadamente *Facebook*, *Twitter*, *Reddit*, *4chan* e canais IRC para alimentarem a plataforma de Inteligência Cibernética da Dognædis. O processo começa com a recolha dos dados, através das Interfaces de Programação de Aplicações (cujo acrónimo é API e provém do inglês *Application Programming Interface*) das fontes, de *crawling* ou *bots* Internet Relay Chat (IRC), passando depois pela fase de processamento e terminando na base de dados do Portolan. Depois da extração é feito um processamento com vista à filtragem e retenção de informação que possa ser importante. Este processamento pode ser feito de duas formas distintas: extração de expressões regulares e processamento de linguagem natural (esta última feita com recurso a ferramentas externas à empresa). Esta filtragem é importante, pois são extraídas quantidades massivas de informação, sendo que nem toda é importante e apenas iria tirar o foco aos responsáveis por analisar a informação final.
Para ajudar na configuração dos *bots* será criada uma interface gráfica de utilizador de *BackOffice*. Ainda dentro das interfaces serão criados *widgets*, tabelas, matrizes e grafos. Acrescenta-se ainda uma interface para análise e classificação manual de eventos.
O Portolan permite assim agilizar e tornar mais rápido um processo que seria muito demorado se fosse realizado por pessoas.

1.3 Entidade Acolhedora

A Dognædis é uma *spinoff* da área de segurança de informação e foi criada por investigadores do CERT-IPN e Universidade de Coimbra. A equipa criou, previ-

amente, no Instituto Pedro Nunes, o CERT-IPN, uma CSIRT(Computer Security Incident Response Team), equipa que esteve em atividade durante 5 anos. Devido ao aumento de sucesso e *feedback* positivo vindo tanto do sector privados como de organizações governamentais, a Dognædis lançou-se como companhia privada.
A Dognædis é certificada pela ISO/IEC 27001 - Sistema de Gestão da Segurança de Informação, cujo objetivo é estabelecer um esquema eficaz de lidar com informação, consolidando um conjunto de práticas eficazes de gestão de segurança de informação. Importa mencionar que a ISO/IEC 27001 é atualmente a maior referência nacional e internacional no que diz respeito à Gestão de Secugrança de Informação e todo o espetro de atividade da Dognædis está certificado.
Lançou um produto inovador, o CodeV, que tem a capacidade detetar problemas de segurança em *software* e que recebeu uma distinção por parte da Gartner.
A empresa tem clientes nos sectores Financeiro, Telecomunicações, Governo e Defesa. Disponibiliza ainda um conjunto de serviços: Suporte à Continuidade de Negócio, Auditoria e Consultoria de Segurança para Infraestruturas e *Software* e Desenho e Gestão de Redes.
A Dognædis está sedeada em Coimbra e tem uma subsidiária no Reino Unido. Em Março de 2016 passou a integrar o grupo Prosegur.

1.4 Estrutura do Documento

Este relatório de estágio encontra-se divido em sete capítulos distintos.
O primeiro capítulo representa a introdução ao projeto e à temática que lhe é adjacente. É apresentado o âmbito e objetivos do projeto para enquadrar o leitor com o mesmo. É ainda apresentada a entidade acolhedora.
No segundo capítulo é apresentado o planeamento das tarefas realizadas ao longo do primeiro semestre. É também referida a metodologia seguida bem como a análise de riscos e respetivo plano de mitigação.
O terceiro capítulo corresponde ao estudo do estado da arte. Começa por descrever o conceito de *Cyber Intelligence* e a *framework* OAuth 2, seguidos das várias fontes de informação que irão ser utilizadas. De seguida apresentam-se plataformas semelhantes ao sistema que foi desenvolvido e comparação com o *Portolan*. Posteriormente apresentou-se o levantamento feito sobre ferramentas utilizadas no desenvolvimento. Por fim foi feita um síntese e escolha final das ferramentas.
No quarto capítulo são especificados os requisitos funcionais, requisitos de qualidade e restrições do trabalho desenvolvido.
O quinto capítulo é apresentado o *Portolan* e o trabalho desenvolvido.
O sexto capítulo é onde se apresenta a validação dos requisitos funcionais e não funcionais.
O sétimo e último capítulo apresenta as conclusões e os desenvolvimentos futuros.

Capítulo 2

Planeamento

Neste capítulo será apresentado o planeamento do trabalho a realizar ao longo do ano, a metodologia de trabalho e a análise de riscos associados ao projeto. Serão ainda apresentados os métodos e abordagens usados na realização do planeamento.

2.1 Primeiro Semestre

A Tabela 2.1 apresenta o planeamento para o primeiro semestre efetuado no inicio do estágio. É acompanhada do diagrama de *Gantt* presente na Figura 2.1.

Fase	**Data de Inicio**	**Data de Fim**
Planeamento de Atividades	**15-09-2015**	**21-09-2015**
Estado da arte	**22-09-2015**	**16-11-2015**
Estudo de ferramentas semelhantes	22-09-2015	05-10-2015
Estudo sobre fontes de informação	06-10-2015	19-10-2015
Estudo sobre ferramentas a utilizar	20-10-2015	16-11-2015
Especificação de Requisitos	**17-11-2015**	**14-12-2015**
Requisitos Funcionais	17-11-2015	26-11-2015
Requisitos Não Funcionais	27-11-2015	10-12-2015
Restrições	11-12-2015	14-12-2015
Estudo do Portolan	**15-12-2015**	**18-12-2015**
Elaboração do relatório	**05-10-2015**	**22-01-2016**

Tabela 2.1: Planeamento do primeiro semestre

2.2 Segundo Semestre

Nesta secção serão apresentados o *Work Breakdown Structure*, a estimação de tarefas e a abordagem *Bottom-up* que serão usados para elaborar o planeamento

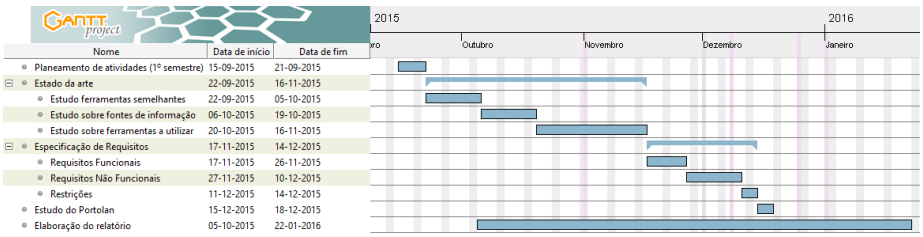

Figura 2.1: Diagrama de *Gantt* relativo ao primeiro semestre

do segundo semestre.
Será ainda apresentado o resultado final do planeamento.
O planeamento realizado nesta secção teve em conta a fase de transição do primeiro para o segundo semestre, com o objetivo de levar a solução a desenvolver ao encontro das necessidades de estágio.

2.2.1 Work Breakdown Structure

O *Work Breakdown Structure*[2] (WBS) consiste no processo de divisão das tarefas mais complexas de um projeto em tarefas mais simples, fáceis de gerir e de estimar o esforço necessário par a sua realização. No entanto existem regras a ser usadas para determinar as tarefas mais pequenas. Uma das regras usadas é a regra 8/80. Esta implica que nenhuma tarefa demore menos de 8 horas nem mais que 80 horas de trabalho.
Existem várias formas de apresentar um WBS, deste estruturas em forma de árvore a listas e tabelas.

2.2.2 Estimação das Tarefas

para estimar o esforço necessário para cada tarefa foi usado o método *Estimativa de Três Pontos*[3], em que a cada tarefa são atribuídos três valores de esforço:

Mais Provável Se tudo correr como esperado

Melhor Caso Se tudo correr bem

Pior Caso Se tudo correr mal

Obtidos estes valores para cada tarefa calcula-se o valor esperados de esforço através da seguinte fórmula:
(MelhorCaso + (4 * CasoMaisProvável) + PiorCaso) / 6

2.2.3 Estimativa de Baixo para Cima

para o desenvolvimento do planeamento foi seguida a abordagem *Estimativa de Baixo para Cima*[3]. esta abordagem consistem em criar o WBS, estimar cada tarefa usando a *Estimativa a Três Pontos* e somar as estimativas para se obter o esforço total.
Esta abordagem possui seis passos fundamentais:

1. Selecionar estimador/equipa de estimação (estagiário)

2. Obter todos os artefactos relevantes da aplicação a ser estimada (arquitetura de alto nível, requisitos e tecnologias adaptadas)
3. Criar o WBS
4. Estimar cada tarefa com a *Estimativa de Três Pontos*
5. Calcular o esforço estimado de todo o projeto
6. Refinar as estimativas, caso necessário

2.2.4 Processo de Estimação

Nesta secção será apresentado o processo de estimação do esforço necessário que começa na criação do WBS, cujo se apresenta abaixo sob a forma de uma lista. Este foi baseado na lista de requisitos funcionais iniciais, sendo que a lista final e mais detalhada de requisitos apresenta-se no anexo A.1.

1. Planeamento do Segundo Semestre
2. Definição do Modelo de Dados
3. Implementação do *bot* do Facebook
    - 3.1. Implementação do *crawler*
        - 3.1.1. Extrair Informação de um Utilizador
        - 3.1.2. Extrair Informação de Amizade
    - 3.2. Implementação da extração de informação através da API
4. Implementação do *bot* do Twitter
    - 4.1. Implementação do *crawler*
    - 4.2. Implementação da extração de informação através da REST API
    - 4.3. Implementação da extração de informação através da *Streaming* API
5. Implementação do *bot* do Reddit
    - 5.1. Monitorização de um *Subreddit*
    - 5.2. Monitorização de uma *Thread*
    - 5.3. Monitorização de um utilizador
6. Implementação do *bot* do 4Chan
    - 6.1. Monitorização de um *Board*
    - 6.2. Monitorização de uma *Thread*
7. Implementação do *bot* do IRC
    - 7.1. Monitorização das mensagens trocadas no canal
    - 7.2. Extração da informação sobre um utilizador
    - 7.3. Monitorização das ações de um canal
8. Implementação do *bot* extrator de expressões regulares

9. Implementação do *bot* de análise de texto
10. Implementação da interface de configuração dos *bots*
    10.1. Gerar Interface de Criação
    10.2. Gerar Interface de Edição
11. Implementação da interface de Classificação Manual de Dados
12. Validação dos Requisitos Não-Funcionais
    12.1. Validar Manutenência
    12.2. Validar Usabilidade
    12.3. Validar Segurança
    12.4. Validar Desempenho
    12.5. Validar Escalabilidade
13. Elaboração do documento final

De seguida foi feita a estimação de cada tarefa usando o já mencionado método *Estimação a Três Pontos*. O resultado apresenta-se na tabela 2.2.

2.2.5 Resultado Final

O resultado final é apresentado na tabela 2.3, acompanhada pela presente na figura 2.2.

Tarefas	Melhor Caso	Caso Mais Provável	Pior Caso	Caso Expectado
1	2	3	4	3,00
2	1	2	3	2,00
3.1.1	10	13	16	15,00
3.1.2	1	2	4	2,17
3.2	2	3	4	3,00
4.1	2	3	4	3,00
4.2	3	5	7	5,00
4.3	1	2	3	2,00
5.1	3	4	5	4,00
5.2	0,5	1	2	1,08
5.3	1	2	3	2,00
6.1	2	3	4	3,00
6.2	0,5	1	2	1,08
7.1	2	3	4	3,00
7.2	0,5	1	2	1,08
7.3	0,5	1	2	1,08
8	2	3	5	3,17
9	2	3	5	3,17
10.1	5	7	10	7,17
10.2	1	2	5	2,33
11	1	3	5	3,00
12.1	1	2	3	2,00
12.2	1	2	3	2,00
12.3	1	2	3	2,00
12.4	3	5	8	5,17
12.5	3	5	8	5,17
13	10	15	20	15,00
SOMA	**62**	**98**	**144**	**102**

Tabela 2.2: Tabela do esforço geral do segundo semestre

Tarefa	Data de início	Data de fim	Duração
1	08-02-2016	11-02-2016	3
2	12-02-2016	15-02-2016	2
3	16-02-2016	10-03-2016	18
3.1	16-02-2016	07-03-2016	15
3.1.1	16-02-2016	03-03-2016	13
3.1.2	04-03-2016	07-03-2016	2
3.2	08-03-2016	10-03-2016	3
4	11-03-2016	24-03-2016	10
4.1	11-03-2016	15-03-2016	3
4.2	16-03-2016	22-03-2016	5
4.3	23-03-2016	24-03-2016	2
5	28-03-2016	05-04-2016	7
5.1	28-03-2016	31-03-2016	4
5.2	01-04-2016	01-04-2016	1
5.3	04-04-2016	05-04-2016	2
6	06-04-2016	11-04-2016	4
6.1	06-04-2016	08-04-2016	3
6.2	11-04-2016	11-04-2016	1
7	12-04-2016	18-04-2016	5
7.1	12-04-2016	14-04-2016	3
7.2	15-04-2016	15-04-2016	1
7.3	18-04-2016	18-04-2016	1
8	19-04-2016	21-04-2016	3
9	22-04-2016	27-04-2016	3
10	29-04-2016	11-05-2016	9
10.1	29-04-2016	09-05-2016	7
10.2	10-05-2016	11-05-2016	2
11	12-05-2016	16-05-2016	3
12	17-05-2016	09-06-2016	17
12.1	17-05-2016	18-05-2016	2
12.2	19-05-2016	20-05-2016	2
12.3	23-05-2016	24-05-2016	2
12.4	27-05-2016	02-06-2016	5
12.5	03-06-2016	09-06-2016	5
13	13-06-2016	01-07-2016	15

Tabela 2.3: Planeamento do Segundo Semestre

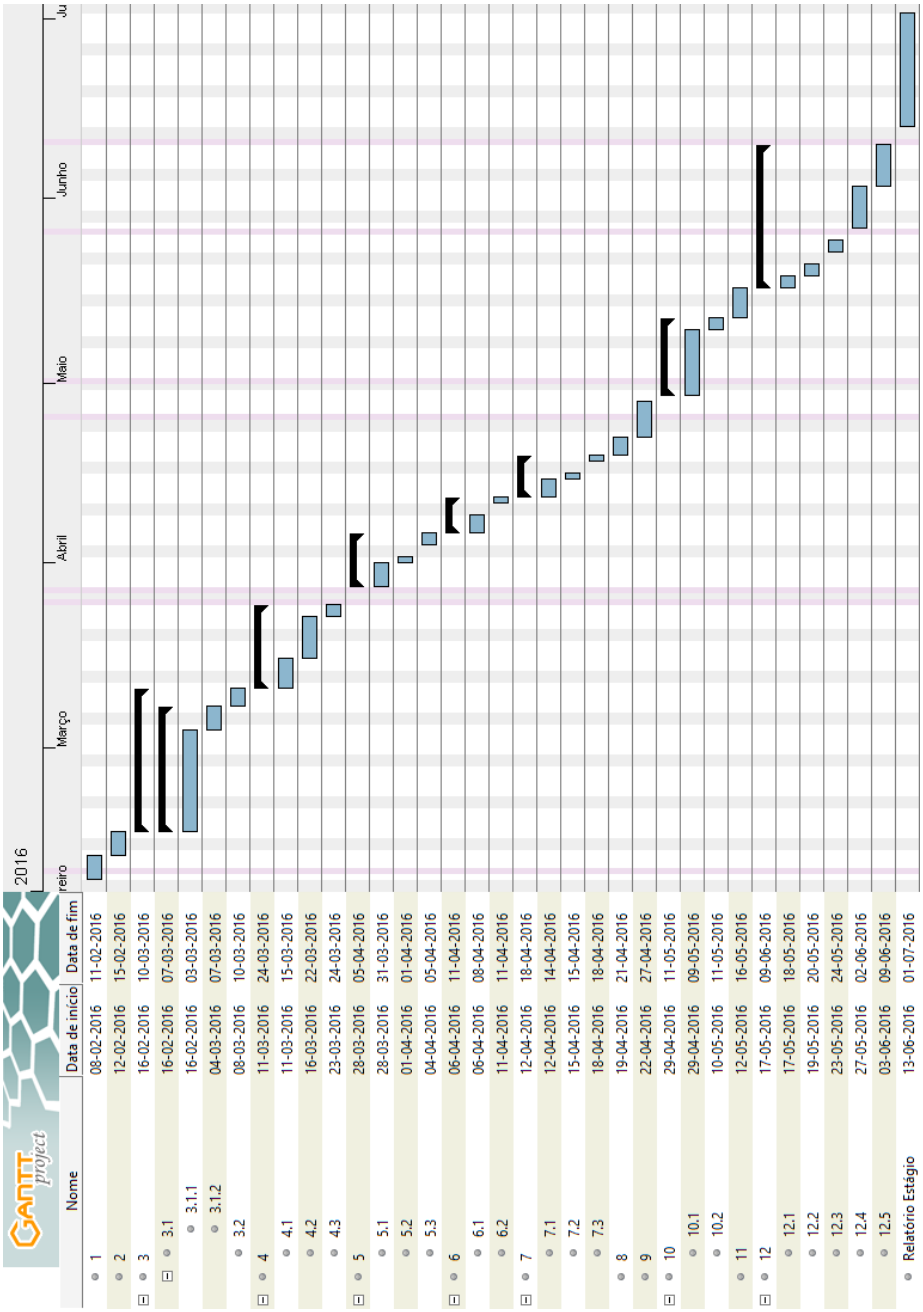

Figura 2.2: Diagrama de Gantt do segundo semestre

2.3 Esforço Real do Segundo Semestre

Esta secção servirá para apresentar o esforço real do segundo semestre. Inicialmente apresenta-se o WBS referente às tarefas desenvolvidas, seguido da apresentação do esforço real de cada tarefa.
Devido a alguns atrasos no desenvolvimento e na obtenção dos acessos à ferramenta externa de análise de texto a utilizar o planeamento anterior não foi cumprido. Para colmatar a até então impossibilidade de cumprimento deste requisito foram adicionados alguns requisitos adicionais com vista à não perca de valor do produto.
O WBS final apresenta-se abaixo em forma de lista.

1. Planeamento do Segundo Semestre
2. Definição do Modelo de Dados
3. Implementação do *bot* do Facebook
   - 3.1. Implementação do *crawler*
     - 3.1.1. Extrair Informação de um Utilizador
     - 3.1.2. Extrair Informação de Amizade
   - 3.2. Implementação da extração de informação através da API
4. Implementação do *bot* do Twitter
   - 4.1. Implementação do *crawler*
   - 4.2. Implementação da extração de informação através da REST API
   - 4.3. Implementação da extração de informação através da *Streaming* API
5. Implementação do *bot* do Reddit
   - 5.1. Monitorização de um *Subreddit*
   - 5.2. Monitorização de uma *Thread*
   - 5.3. Monitorização de um utilizador
6. Implementação do *bot* do 4Chan
   - 6.1. Monitorização de um *Board*
   - 6.2. Monitorização de uma *Thread*
7. Implementação do *bot* do IRC
   - 7.1. Monitorização das mensagens trocadas no canal
   - 7.2. Extração da informação sobre um utilizador
   - 7.3. Monitorização das ações de um canal
8. Implementação do *bot* extrator de expressões regulares
9. Implementação do *bot* de análise de texto
10. Implementação da interface de configuração dos *bots*
    - 10.1. Gerar Interface de Criação

    10.2. Gerar Interface de Edição

11. Implementação da interface de Classificação Manual de Dados
12. Alteração das atuais interfaces para suportar a apresentação dos novos tipos de dados
    12.1. Alterar *widgets*
    12.2. Alterar tabela
    12.3. Alterar matriz
13. Implementação de uma interface para visualização da informação sob a forma de um grafo
    13.1. Criação de um *bot* para criar a estrutura a ser usada para a construção visual do grafo
    13.2. Criação da parte de visualização do grafo
14. Validação dos Requisitos Não-Funcionais
    14.1. Validar Manutenência
    14.2. Validar Usabilidade
    14.3. Validar Segurança
    14.4. Validar Desempenho
    14.5. Validar Escalabilidade
15. Elaboração do documento final

O esforço final é apresentado na tabela 2.4 e na figura 2.3.

Tarefas	Data de início	Data de fim	Duração
1	08-02-2016	11-02-2016	3
2	12-02-2016	15-02-2016	2
3	16-02-2016	22-03-2016	26
3.1	16-02-2016	17-03-2016	23
3.1.1	16-02-2016	14-03-2016	20
3.1.2	15-03-2016	17-03-2016	3
3.2	18-03-2016	22-03-2016	3
4	23-03-2016	12-04-2016	14
4.1	23-03-2016	30-03-2016	5
4.2	31-03-2016	07-04-2016	6
4.3	08-04-2016	12-04-2016	3
5	13-04-2016	22-04-2016	8
5.1	13-04-2016	19-04-2016	5
5.2	20-04-2016	20-04-2016	1
5.3	21-04-2016	22-04-2016	2
6	26-04-2016	29-04-2016	4
6.1	26-04-2016	28-04-2016	3
6.2	29-04-2016	29-04-2016	1
7	02-05-2016	09-05-2016	6
7.1	02-05-2016	04-05-2016	3
7.2	05-05-2016	06-05-2016	2
7.3	09-05-2016	09-05-2016	1
8	10-05-2016	12-05-2016	3
9	13-05-2016	17-05-2016	3
10	12-05-2016	24-05-2016	9
10.1	12-05-2016	20-05-2016	7
10.2	23-05-2016	24-05-2016	2
11	25-05-2016	30-05-2016	3
12	31-05-2016	07-06-2016	6
12.1	31-05-2016	01-06-2016	2
12.2	02-06-2016	06-06-2016	3
12.3	07-06-2016	07-06-2016	1
13	08-06-2016	23-06-2016	11
13.1	08-06-2016	13-06-2016	3
13.2	14-06-2016	23-06-2016	8
14	24-06-2016	15-07-2016	16
14.1	24-06-2016	27-06-2016	2
14.2	28-06-2016	29-06-2016	2
14.3	30-06-2016	01-07-2016	2
14.4	04-07-2016	08-07-2016	5
14.5	11-07-2016	15-07-2016	5
Relatório Estágio	18-07-2016	29-08-2016	30

Tabela 2.4: Esforço Final do Segundo Semestre

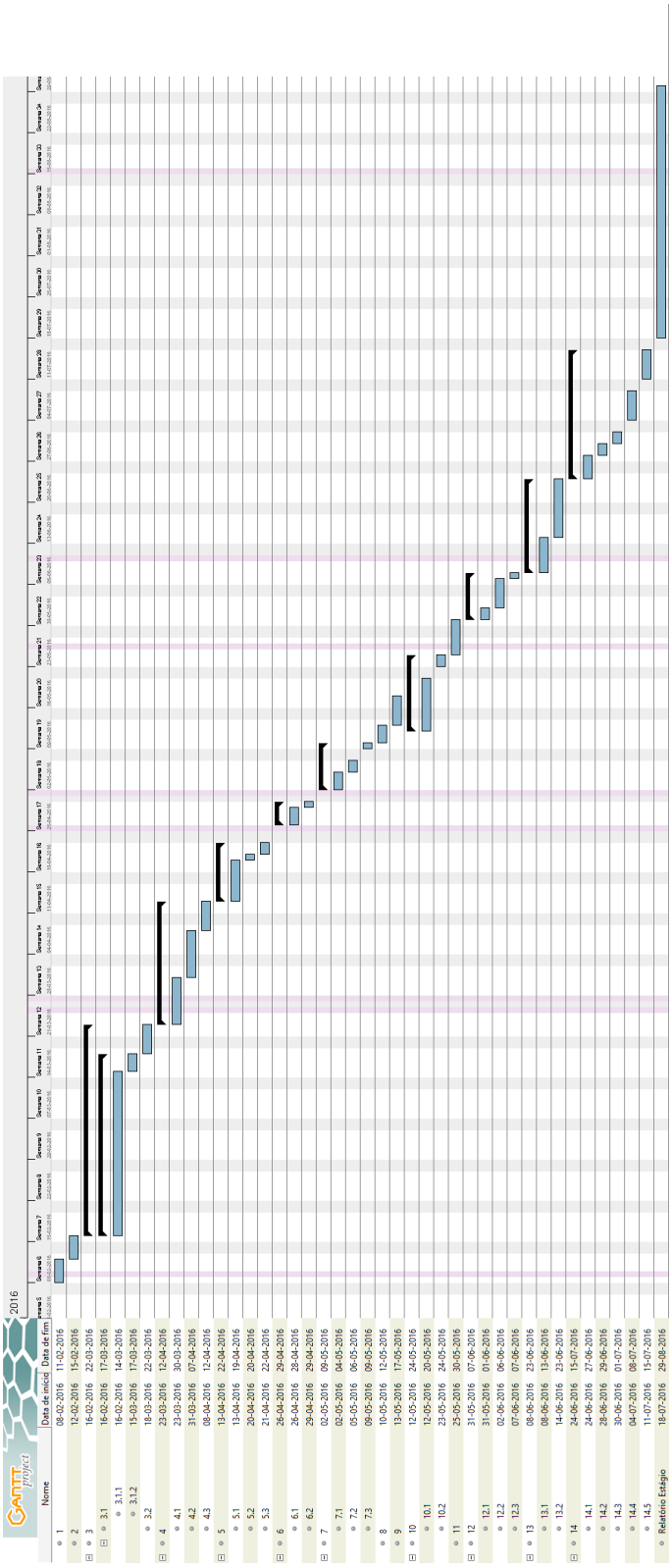

Figura 2.3: Diagrama de Gantt Final do Segundo Semestre

2.4 Metodologia

O estagiário não seguiu nenhuma metodologia de desenvolvimento formalmente comprovada. Deste modo usou como guia principal a lista de requisitos. As macro-tarefas foram decompostas em tarefas mais simples. Depois de obtida a lista completa de funcionalidades (já decompostas) a implementar foram definidos vários graus de prioridade. A priorização foi dividida em três níveis de acordo com o grau de importância identificada e aplicada a cada uma das funcionalidades em conjunto com orientador de estágio. Do mais prioritário para o menos temos os níveis **A**, **B** e **C**. Depois de definidas as prioridades dos vários requisitos a implementação iniciou-se pelos mais prioritários e terminou nos menos prioritários.
Apesar de não ter sido seguida nenhuma metodologia foi importado um conceito usado no Scrum: *Definition of Done*[4] (em português, Definição de Feito). Este último é definido como uma lista de passos que devem ser cumpridos para se considerar uma funcionalidade como terminada.
Os passos considerados foram:

- Implementação do código completa
- Realização de testes

  Testes Unitários Refere-se à prática de testar certas funções e áreas (ou unidades) do código.

  Testes de Integração Refere-se ao tipo de teste em que várias unidades são combinadas e testadas em grupo.

  Testes de Sistema Técnica de testes de caixa negra. É utilizada para avaliar o sistema como um todo, em que as funcionalidades são testada numa perspetiva de ponta a ponta. Testes executados em condições reais de utilização (ou similares).

  Testes de Aceitação Técnica de testes usada para determinar se uma aplicação cumpre ou não os requisitos. Permite ainda determinar se o cliente aceita ou não o sistema.

2.5 Análise de Riscos

Qualquer projeto de desenvolvimento de *software* tem riscos associados, por isso é de interesse realizar um estudo com o objetivo de identificar esses riscos e criar respetivos planos de mitigação.
Este projeto centra-se na coleta de informação de fontes públicas, em especifico os média sociais, apresentando-se estas também como uma fonte de riscos para o desenrolar do projeto. Alguns exemplos de riscos relacionados com as fontes são: alteração da estrutura do site, alteração de termos de serviço, alteração da informação disponibilizada, alteração nas APIs de acesso ou até mesmo o encerramento da fonte.
Depois de realizado o levantamento dos riscos é necessário proceder à sua classificação segundo três níveis: impacto, probabilidade e janela temporal.

Impacto O efeito no sucesso do projeto se o risco ocorrer.

Alto O sucesso do projeto fica comprometido

Médio O sucesso do projeto é possível, mas com grande custo temporal e financeiro

Baixo O sucesso é atingido sem grande dificuldade

Probabilidade Probabilidade do risco provocar o impacto definido.

Alta Probabilidade superior a 70

Média Probabilidade entre 40% e 70% (inclusive)

Baixa Probabilidade inferior a 40

Janela Temporal Tempo que vai desde a identificação do risco até ser necessário lidar com o risco.

Longa Janela temporal superior a 3 meses

Média Janela temporal entre 1 a 3 meses (inclusive)

Curta Janela temporal inferior a 1 mês.

2.5.1 Identificação e Classificação dos Riscos

Depois do levantamento de riscos e escolha de atributos e respetivos níveis de classificação irá ser realizada a classificação dos mesmos, que será apresentada nas tabelas 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11, 2.12 e 2.13.

Ientificador	A
Risco	As tecnologias a utilizar possuem elevada curva de aprendizagem (APIs, ferramentas de acessos HTTP(s), de *parsing* e de criação de *bots* IRC)
Impacto	Médio
Probabilidade	Média
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento
Mitigação	Ajuste do planeamento para ir de encontro ao tempo necessário para aprendizagem das tecnologias

Tabela 2.5: As tecnologias a utilizar possuem elevada curva de aprendizagem

Identificador	B
Risco	A plataforma Portolan possui elevada curva de aprendizagem
Impacto	Baixo
Probabilidade	Alta
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento
Mitigação	Ajuste do planeamento para ir de encontro ao tempo necessário para aprendizagem do Portolan e suporte por parte dos responsáveis pelo desenvolvimento do mesmo

Tabela 2.6: A plataforma Portolan possui elevada curva de aprendizagem

Identificador	C
Risco	Estimativas optimistas do tempo necessário para realização das tarefas
Impacto	Médio
Probabilidade	Alta
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento
Mitigação	Ajuste do planeamento de acordo com a priorização dos requisitos

Tabela 2.7: Estimativas otimistas do tempo necessário para realização das tarefas

Identificador	D
Risco	Alteração dos requisitos
Impacto	Alto
Probabilidade	Média
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento. Alterações à arquitetura.
Mitigação	Ajuste do planeamento de acordo com a priorização dos requisitos.

Tabela 2.8: Alteração dos requisitos

Identificador	E
Risco	Alterações à arquitetura
Impacto	Alto
Probabilidade	Baixo
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento. Alterações ao código previamente escrito.
Mitigação	Ajuste do planeamento de acordo com a priorização dos requisitos.

Tabela 2.9: Alterações à arquitetura

Identificador	F
Risco	Alteração da estrutura da página web
Impacto	Alto
Probabilidade	Média
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento. Alterações aos *crawlers* previamente escritos.
Mitigação	Ajuste do planeamento de acordo com a priorização dos requisitos.

Tabela 2.10: Alteração da estrutura da página web

Identificador	G
Risco	Alteração das APIs de acesso
Impacto	Alto
Probabilidade	Baixa
Janela Temporal	Média
Consequências	Atraso ou incumprimento do planeamento. Alterações aos módulos que fazem acesso a APIs
Mitigação	Ajuste do planeamento de acordo com a priorização dos requisitos.

Tabela 2.11: Alteração das APIs de acesso

Identificador	H
Risco	Alteração da informação disponibilizada pelos serviços
Impacto	Alto
Probabilidade	Baixa
Janela Temporal	Média
Consequências	A utilização dos serviços como fontes de informação viáveis pode ficar comprometida
Mitigação	Remoção do serviço e avaliação de possiveis fontes que possam conter informação semelhante

Tabela 2.12: Alteração da informação disponibilizada pelos serviços

Identificador	I
Risco	Alteração dos Termos de Serviço das fontes de informação
Impacto	Alto
Probabilidade	Baixa
Janela Temporal	Média
Consequências	A utilização dos serviços como fontes de informação viáveis pode ficar comprometida ou poderá levar a ajustes no código previamente escrito
Mitigação	Remoção do serviço e avaliação de possiveis fontes que possam conter informação semelhante. Ajuste do planeamento de acordo com a priorização dos requisitos.

Tabela 2.13: Alteração dos Termos de Serviço das fontes de informação

2.5.2 Matriz de exposição aos riscos

Por forma a priorizar os riscos, a aumentar a visibilidade e a assistir na tomada de decisões foi criada uma matriz (impacto x probabilidade) que está presente em 2.14 acompanhada da respetiva legenda apresentada na tabela 2.15.

		Impacto		
		Baixo	Médio	Alto
Probabilidade	Alta	B	C	
	Média		A	D,F
	Baixa			E,G,H,I

Tabela 2.14: Matriz de exposição aos riscos

Cor	Exposição
	Baixa
	Média
	Elevada
	Crítica

Tabela 2.15: Legenda da matriz 2.14

Depois da análise da matriz de exposição é apresentado na tabela 2.16 a priorização ordenada dos vários riscos identificados.

Identificador	Exposição
C	Elevada
D	Elevada
F	Elevada
A	Média
B	Média
E	Média
G	Média
H	Média
I	Média

Tabela 2.16: Tabela de priorização dos riscos

Capítulo 3

Estado da Arte

Neste capítulo são apresentados os resultados do estudo do estado da arte realizado durante a primeira fase do estágio. Inicialmente apresentam-se as fontes de informação a utilizar. Posteriormente serão apresentadas aplicações que sejam iguais ou semelhantes ao que se pretende desenvolver neste estágio. Foram estudadas ainda um conjunto de ferramentas a utilizar para desenvolvimento deste projeto.

3.1 Cyber Intelligence

Antes de se perceber o que é a *Cyber Intelligence*, primeiro tem de se perceber o que é a Inteligência. Segundo o Departamento de Defesa dos Estados Unidos, a Inteligência pode ser definida da seguinte forma:

1. O produto resultante da coleção, processamento, integração, avaliação, análise e interpretação de informação disponível relativamente a nações estrangeiras, forças ou elementos hostis ou potencialmente hostis, ou áreas de operações atuais ou potenciais.
2. As atividades que resultam no produto.
3. As organizações envolvidas em tais atividades.

De uma forma mais simples e generalizada pode-se definir que a Inteligência é tanto o produto final como as atividades de coleção, processamento, integração, avaliação, análise e interpretação de informação, que lhe deram origem. Pode-se também inferir que o propósito da Inteligência é oferecer cursos de ação para melhorar a tomada de decisão, através da previsão de atividades hostis e de intenções. A *Cyber Intelligence* não é nada mais que o conceito de Inteligência aplicado ao mundo cibernético.

3.2 OAuth 2

Nesta secção será apresentada a OAuth 2[5], servindo a informação apresentada para que se compreenda a forma de autenticação de alguns dos *bot* implementados nas respetivas APIs.
A OAuth 2 é uma *framework* de autorização e permite que aplicações de terceiros

obtenham acesso limitado a um serviço HTTP, como o Facebook por exemplo.
A informação que se segue é apresentada do ponto de vista do estagiário, isto é, do ponto de vista de quem desenvolve uma aplicação.
Existem quatro intervenientes no fluxo de funcionamento da OAuth 2: Proprietário dos Recursos, Servidor dos Recursos, Cliente e Servidor de Autorização.
O **Proprietário dos Recursos**, também referido como utilizador final, é quem autoriza o acesso a um recurso protegido, definindo, geralmente, um conjunto de limitações a essa autorização.
O **Servidor dos Recursos** é onde se encontram alojados os recursos protegidos, e quem aceita e valida os pedidos a esses mesmos recursos através da utilização de *tokens* de acesso.
O **Servidor de Autorização** fornece os *tokens* de acesso ao cliente depois do **Proprietário dos Recursos** se ter autenticado com sucesso e obtido a autorização. Juntamente com o **Servidor dos Recursos** representam a API.
O **Cliente** é a aplicação que faz pedidos de recursos protegidos em nome do **Proprietário dos Recursos** e com a sua autorização.
Os passos apresentados na figura 3.1 passam a ser explicados abaixo:

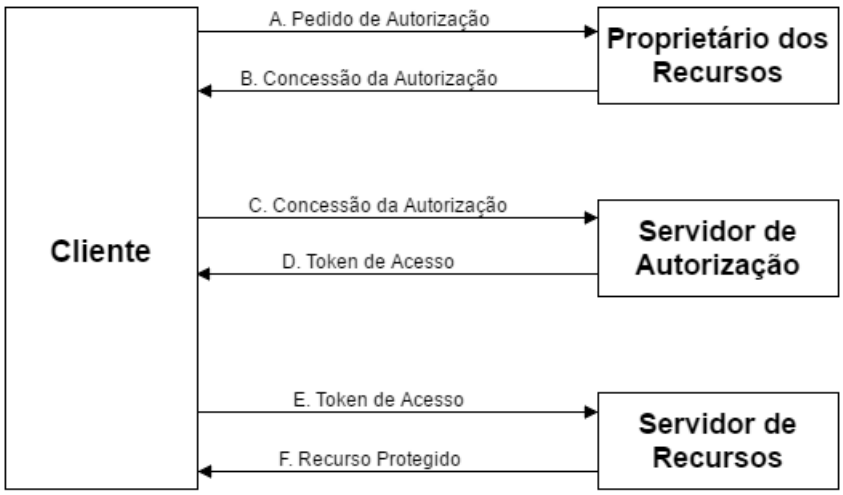

Figura 3.1: Fluxo OAuth 2

A O **Cliente** pede autorização ao **Proprietário** para acerder aos seus recursos.

B Se o **Proprietário dos Recursos** autorizar a aplicação recebe uma concessão de autorização.

C O **Cliente** pede um *token* de acesso ao **Servidor de Autorização** através da sua autenticação e da apresentação da concessão de autorização.

D Se o **Cliente** se autenticar com sucesso e apresentar uma concessão de autorização válida, o **Servidor de Autorização** emite um **token** de acesso.

E O **Cliente** faz um pedido para aceder a recursos protegidos do **Servidor de Recursos** e usa o *token* de acesso para se autenticar.

F Novamente, perante a validade do *token* de acesso, o **Servidor de Recursos** serve o pedido da aplicação.

O fluxo representado é apenas uma ideia genérica do seu funcionamento, podendo variar de acordo com o tipo de concessão de autorização a ser usado.

3.2.1 Concessão de Autorização

A **Concessão de Autorização** é uma credencial que representa a autorização dada pelo **Proprietário dos Recursos** dada ao **Cliente** para este usar para obter o *token* de acesso. Importa referir que este é um ponto importante pois os *bots* a desenvolver não podem envolver qualquer tipo de interação humana nem interfaces gráficas para a obtenção desta credencial e consequentemente do *token* de acesso.
Existem quatro tipos:

Concessão com Código de Autorização Fluxo baseado em redirecionamentos e interação humana com o *browser*. Não cumpre as duas premissas mencionadas anteriormente logo não pode ser utilizado.

Concessão Implícita Fluxo otimizado para clientes implementados num *browser*. Não cumpre as duas premissas mencionadas anteriormente logo também não pode ser utilizado.

Concessão com as Credenciais do Proprietário dos Recursos Neste fluxo as credenciais do **Proprietário dos Recursos** (nome de utilizador e palavra passe) são usadas diretamente para obter o *token* de acesso. Cumpre as duas premissas mencionadas anteriormente logo pode ser utilizado.

Concessão com as Credenciais do Cliente Fluxo em que as credenciais do **Cliente** podem ser usadas como **Concessão de Autorização**. Contudo, o acesso é limitado aos recursos do próprio **Cliente**. Apesar de cumprir as premissas mencionadas anteriormente não pode ser utilizado pois não permite extrair dados de terceiros.

3.2.2 Registo do Cliente

Antes do **Cliente** iniciar o fluxo OAuth 2 mencionado em 3.1 é necessário que este registe a sua aplicação no Servidor de Autorização. Este registo é feito tipicamente através do preenchimento de um formulário na zona de *developers* do serviço, em que se fornece, entre outras informações, o nome da aplicação cliente e o URI de redirecionamento. Dependendo do tipo de concessão utilizado este último é particularmente importante porque será usado para redirecionar o utilizador, depois deste autorizar a aplicação, para a parte da mesma onde serão geridos os *tokens* de acesso.
Depois do registo estar completo, a aplicação terá associados o Identificador do Cliente e o Segredo do Cliente.

3.2.3 Concessão com as Credenciais do Proprietário dos Recursos

Na secção 3.2.1 explica-se a razão de utilização da Concessão com as Credenciais do Proprietário dos Recursos. O fluxo para este tipo de concessão apresenta-se na figura 3.2. Os passos apresentados na figura 3.2 passam a ser explicados abaixo:

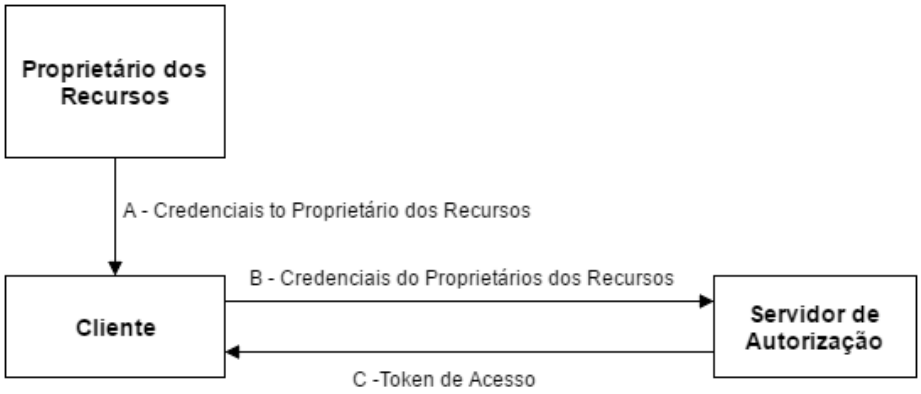

Figura 3.2: Fluxo da Concessão com as Credenciais do Proprietário dos Recursos

A O **Proprietário dos Recursos** fornece ao **Cliente** o seu nome de utilizador e palavra-passe.

B O **Cliente** pede um *token* de acesso ao **Servidor de Autorização** usando as credenciais recebidas. Quando faz o pedido o **Cliente** autentica-se no **Servidor de Autorização**.

C O **Servidor de Autorização** autentica o **Cliente** e valida as credenciais. Se forem válidas cria um *token* de acesso.

Este é o fluxo usado para aceder às APIs que assentam em OAuth 2.

3.2.4 Twitter e OAuth 1.0a

O API do Twitter assenta na OAuth 1.0a[6] que foi tornada obsoleta pela OAuth 2. Contudo, a autenticação nesta API baseia-se na Concessão com as Credenciais do Cliente usada na OAuth 2. O fluxo da mesma apresenta-se na figura 3.3.

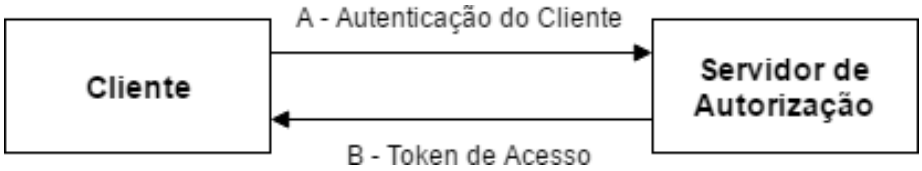

Figura 3.3: Fluxo da Concessão com as Credenciais do Cliente

3.3 Fontes de Informação

Nesta subsecção apresentam-se as fontes de informação que foram propostas para o estágio: *Facebook*, *Twitter*, *Reddit*, *4chan* e canais IRC. Estas são as fontes das quais os *bots* extrairão informação. É apresentada uma breve descrição de cada uma das fontes, bem como o meio de acesso primário à sua informação e algumas limitações.
Devido ao elevado número de utilizadores, à acessibilidade e ao alcance dos média

sociais como por exemplo o Facebook e o Twitter, grupos terroristas têm usado cada vez mais para promover os seus objetivos, espalhar a sua mensagem e até organizar ataques.

3.3.1 Facebook

O *Facebook*[7] é atualmente a maior rede social, com cerca de mais 1500 milhões de utilizadores[8]. É assim imperativo a sua utilização como fonte de informação de referência. Disponibiliza a *Graph API*[9] que é o meio de acesso primário para aplicações escreverem e lerem para o grafo socialdo *Facebook*. O grafo social é uma estrutura em forma de grafo que mantém as relações entre pessoas e as coisas em que elas estão interessadas, tudo isto possível através das páginas, do mecanismos de "Gostos"do *Facebook*, entre outros vários tipos de ligações existentes nesta rede social. Uma representação exemplo encontra-se na figura 3.4[10]. Contudo o *Facebook* tem orquestrado um conjunto muito sofisticado de controlos de privacidade online para ajudar a proteger os seus utilizadores, o que leva a que a *Graph API* apenas retorne conteúdo que o utilizador autenticado consiga visualizar através da rede social. O acesso à graph API é feito usando a *framework* de autorização *OAuth 2*[5].

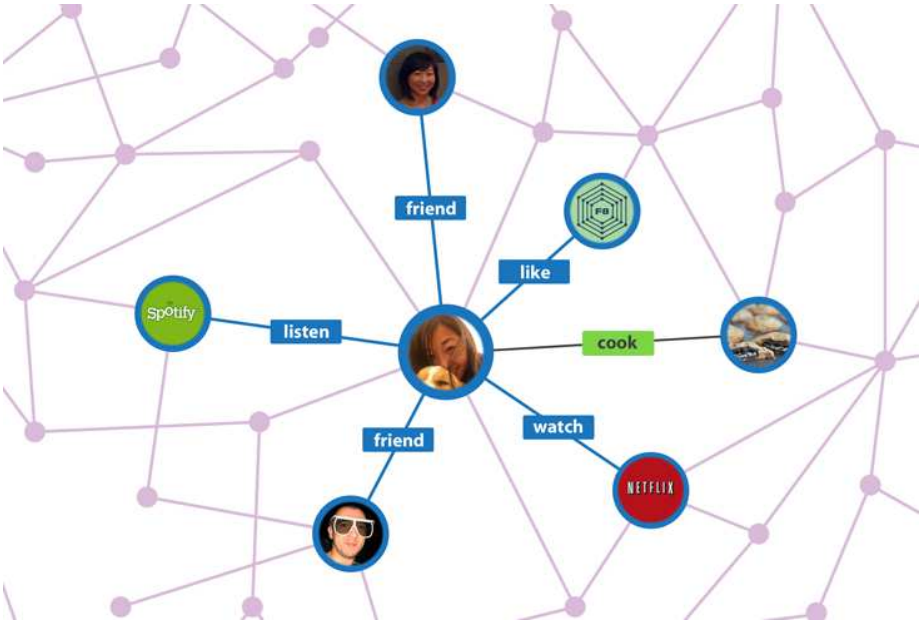

Figura 3.4: Exemplo para o Grafo Social do Facebook

3.3.2 Twitter

O *Twitter*[11] é uma rede social e um serviço de *microblogging* com cerca de 320 milhões de utilizadores mensais[12] e que permite aos utilizadores escrever e ler pequenas mensagens (até 140 caracteres) chamados *Tweets*. À semelhança do *Facebook* é uma fonte rica de informação, em que a mesma aparece e fica disponível

para consumo quase em tempo real. Os *Tweets* e o mecanismo "seguir"do *Twitter* conecta as pessoas de várias formas, desde simples diálogos entre elas a grafos que as ligam aos seus gostos. Apenas utilizadores registados podem escrever, no entanto o conteúdo encontra-se disponível para leitura por utilizadores não registados.
Disponibiliza uma API RESTful[13] para acessos programáticos de leitura e escrita de *Tweets*; as repostas são no formato JSON (JavaScript Object Notation)[14]. Caso um utilizador pretenda monitorizar ou processar *Tweets* em tempo real, o *Twitter* possui a *Streaming* API[15], que permite que as mensagens sejam empurradas para a aplicação, sem haver necessidade de fazer *polling* constante a um *endpoint* REST[16].
O *Twitter* permite, na maioria das situações, 15 pedidos por cada 15 minutos. Existem ainda outras situações em que é permitido que se façam 180 pedidos por cada 15 minutos. Existe ainda uma minoria de casos em que são possíveis 30, 60 e 300 pedidos por cada 15 minutos.
O *Twitter* também não é exceção e também é muito comum verificar que é bastante utilizado, por exemplo, para anunciar as chamadas "ops". Os perfis de *Twitter* usados para estas operações ("ops"ou apenas "op") geralmente tem no nome de perfil a palavra *Operation*, e são usados na maioria das vezes para anunciar operações, que consistem em ataques DDoS (Distributed Denial-of-Service) a sites. Na figura 3.5[17] apresentam-se exemplos de *Tweets* a anunciar esses ataques.

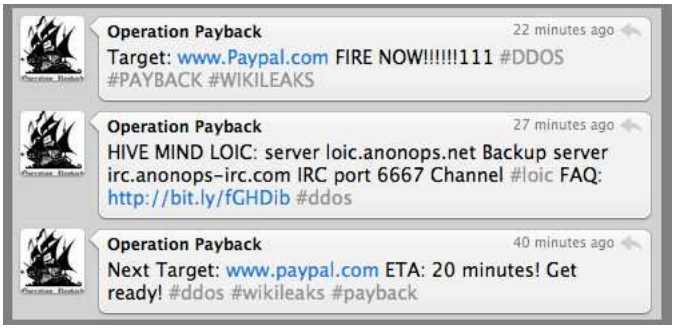

Figura 3.5: Exemplo de um Tweet sobre um ataque DDoS

3.3.3 Reddit

O *Reddit*[18] é um site de entretenimento, notícias e rede social com cerca de 235 milhões de utilizadores mensais[19] onde utilizadores registados podem submeter conteúdo (*thread*), como textos ou hiperligações para outros *sites*. Essas *threads* poderão receber votos, para cima ou para baixo, que servirão para as organizar dentro do *site*. Todos as *threads* são organizados por áreas de interesse chamadas Subreddits. Cada Subreddit possui seis separadores:

Populares Onde são mostradas as *threads* que recentemente têm muita atividade (comentários ou votos "para cima").

Novos Mostra as últimas *threads* criadas, ordenadas por ordem crescente de criação.

Subindo São as *threads* que estão a apresentar bastante atividade no momento.

Controversos *Threads* que bastantes "votos para cima"e "para baixo".

Topo É onde se encontram as *threads* com mais "votos para cima"durante um período especificado.

Gilded *Threads* e comentários que receberem *gold*. O Reddit permite que os utilizadores façam donativos aos utilizadores.

Este esquema de organização dos Subreddits permite uma maior flexibilidade aquando da extração da informação, pois além de permitir que se tenha acesso às *threads* que vão sendo criadas, também permite que o utilizador apenas se foque nas mais populares.
Disponibiliza uma API assente em REST e JSON. O *Reddit* impõe o cumprimento de certas regras de utilização, em que de destacam o limite do número de pedidos por minuto. São permitidos 30 pedidos por minuto, podendo este número aumentar para 60, caso os clientes se conectem usando *OAuth2*.
Como não é possível monitorizar todo o conteúdo do *Reddit* a solução passa por monitorizar certos *Subreddits* que se apresentem como uma boa fonte de informação. Um exemplo seria o *Subreddit netsec*[1] onde são criadas *threads* sobre segurança de informação, como for exemplo informações sobre vazamentos de palavras-passe ou base de dados. A figura 3.6 [20] apresenta um exemplo.

Figura 3.6: Exemplo de uma Thread no Reddit com informação sobre um vazamento de números de telefone

3.3.4 4chan

4chan[2] é um *imageboard* com cerca de 22 milhões de utilizadores mensais[21] em que os utilizadores postam anonimamente. Tal como no *Reddit* os seus conteúdos encontram-se organizados em áreas de interesse, neste caso denominadas *boards*.
O *4chan* disponibiliza uma API RESTful[22] e devolve informação no formato JSON. Nas regras da API informa-se que se deve realizar no máximo um pedido por segundo.
O site já foi ligado a subculturas e activismo da Internet, mais especificamente ao grupo activista *Anonymous*. Os utilizadores do *4chan* têm tido parte ativa na coordenação de ataques contra sites ou utilizadores da Internet ou criação de ameaças de violência[23].

3.3.5 Internet Relay Chat Servers/Channels

Internet Relay Chat (IRC)[24] é um protocolo da camada de aplicação que facilita a comunicação na forma de texto. Permite também a troca de ficheiros e conversas privada e em grupo.
O IRC funciona como um sistema cliente servidor, em que existe um servidor ao

[1] `https://www.reddit.com/r/netsec`
[2] `http://www.4chan.org/`

qual se ligam os vários clientes e todas as mensagem enviadas pelos segundos têm que passar pelo primeiro. Servidores também se podem ligar entre si e tudo o que se ligar a um servidor, que também não o seja, é obrigatoriamente um cliente. Os clientes distinguem-se através de uma alcunha única.
Dentro dos servidores existem canais com um ou mais utilizadores. Sempre que um utilizador pretende entrar num canal, caso este não exista é automaticamente criado. Se um canal ficar vazio é também automaticamente apagado.
Os servidores IRC também se apresentam como uma fonte importante de informação em particular para deteções de operações de uma *botnet*. A título de exemplo, pode-se ver na figura 3.5 que no segundo Tweet é mencionado o seguinte servidor IRC: *loic.anonops.net*, juntamente com *HIVE MIND LOIC*. LOIC (*Low Orbit Ion Cannon*) é uma aplicação desenvolvida para lançar ataques DDoS e possui uma funcionalidade *hive mind* (em português, mente de colmeia) que permite que a cópia do LOIC que o utilizador tem instalada no seu computador seja apontada para um canal IRC, permitindo que outra entidade controle a aplicação, criando-se assim um *botnet* criminosa[25].

3.4 Estudo de Ferramentas Semelhantes

Nesta secção são identificadas ferramentas que contêm um conjunto de funcionalidades semelhantes ao que se pretende desenvolver. Estas apresentam-se como um ponto de partida e meio de comparação para este projeto. Estas ferramentas fazem, entre outras coisas, recolha e análise de informação com origem nos média sociais. As ferramentas abordadas são as seguintes: *The Cyveillance Cyber Threat Center* da Cyveillance[26], *Threat Intelligence Platform* da DigitalStakeout[27], *Web Intelligence Engine* da Recorded Future[28], *Cyber Intelligence Feeds* da SenseCy[29] e *ZeroFOX Enterprise* da ZeroFOX[30].

3.4.1 The Cyveillance Cyber Threat Center

Cyveillance é uma empresa envolvida na indústria da inteligência cibernética. Usa uma plataforma de tecnologia proprietária, *The Cyveillance Cyber Threat Center*, e analistas para identificar riscos antecipadamente para prevenção e mitigação de ameaças. Esta plataforma combina pesquisas na web, monitorização de redes sociais e canais IRC, entre outras fontes, para dar conhecimento sobre ameaças fora da rede do utilizador. Centra-se na descoberta de informação técnica relacionada com as ameaças, como nomes de domínio e endereços de IP, atividades de *phishing*, *malware*.
Além de permitir pesquisas baseadas em texto e em imagens, o utilizador também pode configurar filtros específicos para certas ameaças, pesquisas personalizadas e alertas. Inclui um conjunto de *dashboards* gráficos e relatórios diários de ameaças. Na última versão do produto as pesquisas baseadas em imagem permitem procurar na web *trademarks*, contas e produtos falsos.
Na figura 3.7[31] apresenta-se um diagrama representativo do ciclo de geração de inteligência usado por pela Cyveillance.
Há alguns meses foi adquirida pela LookingGlass Cyber Solutions, Inc[3].

[3] `https://www.lookingglasscyber.com/`

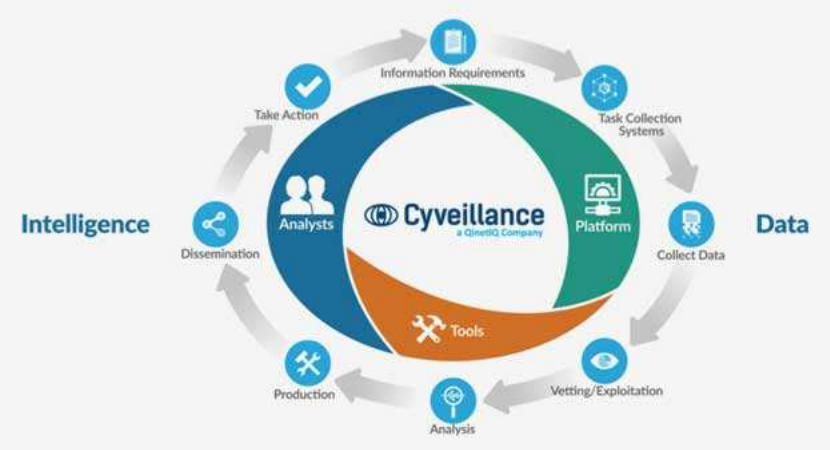

Figura 3.7: Representação das várias fases de geração de inteligência da plataforma *The Cyveillance Cyber Threat Center*

3.4.2 Threat Intelligence Platform

A DigitalStakeout oferece uma plataforma de inteligência cibernética baseada na *cloud*, a *Threat Intelligence Platform*[32] que analisa a web em tempo real devolvendo os conhecimentos necessários para gerir os riscos cibernéticos e atenuar as ameaças. Permite agregar informação das redes sociais por parâmetros à escolha do utilizador, como *keyword*, *hashtag*, localização, meta-dados, entre outros.
As descobertas são automaticamente classificadas e ligadas a pessoas, lugares, coisas ou outras entidades. Inclui ainda um conjunto de ferramentas com o propósito de ajudar a analisar e visualizar os resultados.

3.4.3 Web Intelligence Engine

A *Recorded Future* possui o motor de inteligência patenteado, o *Web Intelligence Engine*[33], que analisa continuamente a Web para dar aos seus cliente uma visão sobre ameaças emergentes. Este motor usa processamento de linguagem natural e outras técnicas de análise. Possui coletores automatizados de informação para mais de 700,000 fontes em sete línguas diferentes (que incluem blogs, média sociais, fóruns e sites governamentais) e uma interface gráfica com *dashboards* para análise e visualização das ameaças, bem como um sistema de alertas. Permite filtragem e categorização das ameaças relevantes para organização ou localização geográfica.
Permite ainda subscrever gratuitamente a *Recorded Future Cyber Daily* para receber diariamente os melhores resultados sobre ameaças via email.

3.4.4 Cyber Intelligence Feeds

SenseCy é uma empresa que atua também na área da OSINT, em particular *Cyber Intelligence* e que oferece um serviço denominado *Cyber Intelligence Feeds*[34]. À semelhança das anteriores especializa-se na coleção, produção e análise de informação *open-source* nomeadamente informação proveniente de blogs, fóruns, redes sociais e ainda *Deep Web* e *DarkNet*. Oferece um conjunto de relatórios cobrindo vários tópicos: relatórios técnicos, perfis de grupos hacktivistas, relatórios regionais

ou relatórios dedicados a tópicos específicos. Também recorre à ajuda de analistas para assegurar que os seus clientes recebam apenas informação importante e precisa.

3.4.5 ZeroFOX Enterprise

A ZeroFOX oferece uma plataforma de segurança contra os riscos nos média sociais, com o objetivo de capacitar as organizações de uma melhor postura relativamente à segurança de informação, mitigação de fraudes e prevenção de ameaças. Essa plataforma, de nome *ZeroFOX Platform*[35], encontra-se em execução na *cloud* e foi desenhada para monitorizar objetos sociais, como contas de utilizador, *keywords*, *hashtags*, endereços IP, domínios, localização geográfica e detetar ameaças de segurança de informação. Além dos coletores de dados possui também três motores de *machine-learning*: *Profile Analysis Engine*, *Link Analysis Engine* e *Content Analysis Engine* de forma a determinar a natureza maliciosa do conteúdo de um *post* feito no Facebook, por exemplo.
Também possui *dashboards* interativos com funcionalidades de *drill-down*.
A ZeroFOX disponibiliza na sua página um diagrama ilustrativo do funcionamento da sua plataforma e que se apresenta abaixo, na figura 3.8[36].

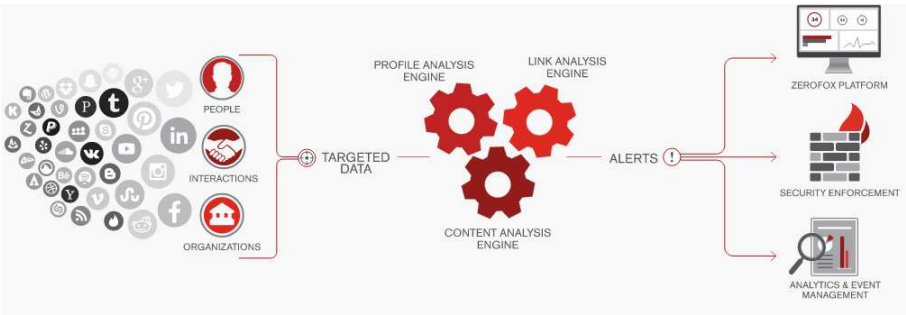

Figura 3.8: Esquema ilustrativo do funcionamento da plataforma ZeroFOX Platform

3.4.6 Síntese

Nesta secção serão apresentadas na tabela 3.1 as funcionalidades de cada uma das ferramentas existentes relativamente ao que se pretende desenvolver.
Para avaliar será usado o seguinte método:

- ✓- A ferramenta apresenta a característica
- ✗- A ferramenta não apresenta a característica
- N/A - Não há informação que permita concluir se a ferramenta apresenta ou não a característica

Como se pode verificar todas as ferramentas fazem análise de informação proveniente das redes sociais e a maioria efetua pesquisas por padrões de caracteres. Dentro desses padrões encontram-se na maioria *keywords*, *hashtags* endereços IP e domínios. Menor ainda é o número dos que referem que fazem algum tipo de análise textual.

Funcionalidades / Plataformas	Pesquisa por padrões	Análise de Texto	Dashboards	Agregação de Info	Grafo	Classificação de dados
Portolan	✓	✓	✓	✓	✓	✓
The Cyveillance Cyber Threat Center	✓	N/A	✓	N/A	N/A	N/A
Threat Intelligence Platform	✓	N/A	✓	✓	N/A	N/A
Web Intelligence Engine	✓	✓	✓	N/A	N/A	✓
Cyber Intelligence Feeds	✓	N/A	N/A	N/A	N/A	N/A
ZeroFOX Enterprise	✓	✓	✓	N/A	N/A	N/A

Tabela 3.1: Síntese das funcionalidades

3.5 Ferramentas de Desenvolvimento

Além da utilização das APIs das redes sociais referidas anteriormente pode ser necessário recorrer a técnicas de *crawling*, acessos a APIs, bem como ferramentas de criação de *bots* para os canais IRC. Abaixo apresentam-se algumas ferramentas consideradas para utilização no desenvolvimento das funcionalidades previstas.

3.5.1 Web Crawling e Acesso a APIs

A extração de informação das fontes pode ser realizada de duas formas: *web crawling* ou acedendo às suas APIs. Caso seja por *crawling* é ainda necessária uma biblioteca de *parsing* de HTML. Caso seja acesso às APIs será necessária uma biblioteca de OAuth.
As ferramentas de acesso analisadas, que tanto poderão ser utilizadas no *crawler* como no acesso às APIs foram: Scrapy[37], urllib[38], urllib2[39], Requests[40] e Mechanize[41].
Quanto às ferramentas de *parsing* HTML analisadas apresentam-se as seguintes: Scrapy[37], HTMLParser[42], lxml[43], html5lib[44] e Beautiful Soup[45].
O Scrapy aparece como uma ferramenta de acesso e como uma ferramenta de *parsing*, pois possui ambas as funcionalidades. A análise de cada uma das partes será feita em separado, juntamente com as ferramentas do mesmo tipo.
Por fim as ferramentas de OAuth analisadas foram: OAuthLib[46] e Requests-OAuthLib[47].

3.5.1.1 Ferramentas com funcionalidades completas de web crawling

Considerou-se uma ferramenta completa algo que permite o acesso a um URL e efetuar o *parsing* de HTML.

Scrapy *Framework open source* de *web crawling* e *web scraping* de alto nível usada para retirar informação estruturada de páginas *web* . É escrita em *Python*[48] e tem suporte integrado para seleção e extração de informação de fontes HTML[49]/XML[50] com utilização de expressões XPath[51]. Abaixo apresentam-se os conceitos básicos principais desta *framework*:

- **Spider** Spiders são classes que definem como será feito o *scraping* de um determinado site (ou grupo de sites), incluindo como proceder na presença de *links* (por exemplo, pode-se definir se a aplicação segue ou não o link) e como extrair a informação das páginas, isto é, como fazer o *parsing* do conteúdo.
- **Selector** Mecanismo do Scrapy para extrair a informação de uma fonte HTML. Chamam-se *Selectors* (em português, seletores) porque selecionam cer-

tas partes do documento HTML através da especificação de expressões XPath ou CSS. Estes *Selectors* são construidos sobre a biblioteca *lxml*, fazendo com que sejam ambos semelhantes relativamente à velocidade e precisão de *parsing*.

Item A classe *Item* é usada para definir o formato da informação de saída. Objetos desta classes são simples *containers* que serão preenchidos com a informação resultante do *parsing*. Têm um funcionamento como os dicionários do Python, mas fornecem proteção acrescida contra campos não declarados.

Item Pipeline Todos os *Items* extraídos por uma *Spider* são enviados para o *Item Pipeline*, onde serão processados sequencialmente por vários componentes. Cada um destes componentes é uma classe que implementa um único método: recebem um *Item* e executa uma ação sobre ele, decidindo também este continua no *pipeline* ou é descartado.

O Scrapy permite que o utilizador ajuste um conjunto de configurações, permitindo aumentar a eficiência e o desempenho dos *crawlers*. É possível modificar o número de pedidos em simultâneo e o nível de verbosidade dos *logs*, desativar a utilização de *cookies*, de novas tentativas de acesso e de redirecionamentos, mudar o tempo de espera por um novo pedido, entre outros.
Está ao abrigo da licença BSD.
Cada parte do Scrapy será analisada juntamente com as outras ferramentas analisadas nas secções 3.5.1.2 e 3.5.1.4.

3.5.1.2 Ferramentas de Acesso

As ferramentas de acesso permitem que se façam pedidos a um URL, à semelhança do que é feito por um *browser*. Dentro das ferramentas de acesso encontram-se as seguintes:

urllib Módulo da biblioteca padrão do *Python* que fornece uma interface de comunicação simples para acessos a recursos identificados por um URL. Pode ser usado para acessos HTTP, HTTPS, FTP e ficheiros locais. Permite também a passagem de parâmetros no URL, mas é necessário proceder-se ao *encoding* dos mesmos antes que se faça o pedido (inclui uma funcionalidade para o efeito). Inclui ainda a possibilidade de utilização de *proxies*.

urllib2 Módulo da biblioteca padrão do *Python* que fornece uma interface de comunicação atualizada para acesso a recursos identificados por um URL. Pode ser usado para acessos HTTP, HTTPS, FTP e ficheiros locais. Suporta redirecionamento de URL, *cookies*, *proxies*, autenticação básica e autenticação *digest*. Além do mencionado permite ao utilizador especificar os *headers* para os pedidos. Permite também a passagem de parâmetros no URL, mas necessita da função de *encoding* da biblioteca *urllib*, daí que parte das vezes seja necessário usar ambas as bibliotecas em simultâneo.

Requests Biblioteca escrita em *Python* semelhante às urllib e urllib2, mas que fornece uma interface de mais alto nível que estas últimas. À semelhança das anteriores também permite a passagem de parâmetros no URL, mas não necessita que se proceda explicitamente ao *encoding*. Dentro das funcionalidades que oferece encontram-se: personalização dos *headers* dos pedidos,

sessões, *cookies*, *proxies*, redirecionamento, histórico, e autenticações básica e *digest*. Outra funcionalidade que também possui e que poderá ser útil é a *Streaming Requests*, que permite iterar facilmente sobre APIs de *streaming*, tal como a *Twitter Streaming API*.
Encontra-se ao abrigo da licença Apache2.

Mechanize Um módulo de *Python* muito útil para navegar na *web* programaticamente, simulando um *browser*. O Mechanize guarda e envia *cookies* automaticamente, segue *links* e preenche e submete formulários. Mantém também o registo dos *sites* visitados como histórico, permitindo "retroceder", "avançar"e "recarregar". Está ao abrigo da licença BSD.

Síntese e Escolha
Abaixo apresenta-se a tabela 3.2 que servirá para resumir de forma visual quais das funcionalidades necessárias são suportadas pela ferramentas apresentadas. Para avaliar será usado o seguinte método:

- ✓- A ferramenta suporta a funcionalidade
- ✗- A ferramenta não suporta a funcionalidade

	urllib	**urllib2**	**Requests**	**Mechanize**	**Scrapy**
Acesso HTTP	✓	✓	✓	✓	✓
Acesso HTTPS	✓	✓	✓	✓	✓
Autenticação Básica	✗	✓	✓	✓	✓
Autenticação Digest	✗	✓	✓	✓	✗
Proxies	✗	✓	✓	✓	✓
OAuth	✗	✗	✗	✗	✗
Pedidos de Streaming	✗	✗	✓	✗	✗

Tabela 3.2: Tabela com funcionalidades necessárias (na primeira coluna) e ferramentas (primeira linha)

Através da análise da tabela verificamos que a ferramenta que se encontra mais próxima do pretendido é a ferramenta *Requests*. No entanto, e tal como todas as outras, não suporta OAuth. Importa ainda relembrar que esta funcionalidade é necessária para realizar a interação com as APIs das fontes de informação. Para o efeito foi utilizada uma biblioteca adicional. A escolha da mesma encontra-se documentada na secção 3.5.1.3.
A acrescentar ao facto de ser a que mais se adequa, a escolha da biblioteca *Requests* permitiu não aumentar o número de dependências externas da aplicação, pois é atualmente utilizada no Portolan.

3.5.1.3 Bibliotecas OAuth

Dentro das bibliotecas que permitem implementar a lógica OAuth destacam-se as seguintes: *OAuthLib* e *Requests-OAuthLib*.

OAuthLib Biblioteca que implementa a lógica OAuth sem assumir a utilização de uma biblioteca de pedidos HTTP(S) específica, sendo por isso muito genérica. Foi desenvolvida seguindo as especificações mais recentes de OAuth 1.0 e OAuth 2.0.

Requests-OAuthLib Combinação das bibliotecas *Requests* e *OAuthLib*.

A escolha pendeu para a segunda, pois além de ser implementada usando *Requests*(que foi a biblioteca escolhida para a realização de pedidos HTTP(s)), a sua documentação encontra-se melhor reproduzida e munida de um maior número de exemplos.

3.5.1.4 Parsers de HTML

Um *parser* de HTML permite fazer uma análise automatizada do mesmo. Possui duas funcionalidades distintas[52]:

Percorrer o conteúdo HTML Permite que se navegue pela estrutura de um documento HTML, extraindo ou modicando o seu conteúdo.

Limpar o conteúdo HTML Permite corrigir a falta de *tags* de fecho e inserir indentação.

Das duas funcionalidades, apenas a primeira será utilizada. Mais especificamente, apenas se pretende extrair conteúdo. As ferramentas de *parsing* de HTML são apresentadas abaixo:

HTMLParser Módulo da biblioteca padrão do *Python* que permite fazer *parsing* de um documento HTML ou XHTML[53]. Objetos desta classe incluem um conjunto de métodos que são chamados automaticamente para *tags* específicas, como por exemplo os métodos *handle_starttag* e *handle_endtag* que tratam das *tags* de início e fim, respetivamente. Outro método importante é o *handle_data(data)* que é chamada para processar dados de *tags* de texto, *<script>* e *<style>*. Esta biblioteca não cria nenhuma estrutura aquando do *parsing* do conteúdo que lhe é fornecido, como algumas das outras bibliotecas que serão referidas mais abaixo.

lxml Biblioteca *Python* baseada em libxml2 e libxslt (duas bibliotecas de C) e ElementTree (criada por Fredrik Lundh) que permite ler informação de formatos HTML, XHTML e XML e constrói uma árvore de *parsing*. Esta árvore pode ser atravessada de duas maneiras distintas: XPath ou seletores CSS. Possui dois *parsers*, um para HTML e outro para XML, sendo este último usado também para XHTML. Apesar de suportar HTML mal formado não garante que o resultado final apresente o conteúdo original. O *parsing* pode ser feito de um ficheiro local ou de um URL HTTP ou FTP.
Permite também fazer algo semelhante ao HTMLParser, com uma funcionalidade chamada *target parser interface*, que consiste em passar um objeto de destino ao *parser*. Esse objeto de destino é uma classe com métodos bem definidos que são executados para *tags* específicas, como por exemplo, *tags* de inicio e de fim ou informação textual.
Encontra-se ao abrigo da licença BSD.

html5lib html5lib é uma biblioteca de *parsing* de HTML escrita em Python que é bastante influenciada pelos *browsers* atuais e baseada na especificação WHATWG HTML5[54]. À semelhança do lxml também é construída uma árvore de *parsing*, em que podem ser escolhidas árvores de três tipos diferentes: a *etree* que é a árvore padrão, *dom* que constrói uma árvore baseada em

xml.dom.minidom e a *lxml.etree* que consiste na implementação da *ElementTree* por parte da biblioteca lxml. Depois de construída a árvore é possível fazer a travessia da mesma usando os *Tree Walkers* em que os nodos são devolvidos como tuplos com a seguinte estrutura:(nodo atual, índice do nodo atual relativamente ao seu pai, nós ancestrais, *flag*).
Está ao abrigo da licença MIT.

Beautiful Soup É uma biblioteca escrita em *Python* que permite retirar informação de documentos HTML/XML. O utilizador pode escolher o *parser* que pretender dentro dos suportados: HTMLParser, lxml e html5lib. Também à semelhança de algumas das ferramentas anteriores o BeautifulSoup transforma o conteúdo HTML numa árvore constituída por objetos Python próprio desta biblioteca, sendo os três principais: Tag, NavigableString, BeautifulSoup. Um objeto do tipo Tag corresponde a uma simples *tag* de um documento XML ou HTML. NavigableString corresponde ao texto que se encontra dentro de uma *tag*. Um objeto do tipo BeautifulSoup representa o conteúdo como um todo.
Esta biblioteca não possui suporte para expressões XPath, mas possui para seletores CSS. Para navegar através da árvore basta indicar o nome da *tag* que se pretende: por exemplo para a *tag* <title> basta a linha de código *BeautifulSoup.title* ou através de expressões regulares.
Está ao abrigo da licença MIT.

Como já foi mencionado ao longo da descrição de algumas das bibliotecas de *parsing*, o conteúdo HTML é estruturado sob a forma de uma árvore. Na figura 3.9[55] apresenta-se um exemplo abstrato a qualquer uma das implementações feitas pelas ferramentas mencionadas.

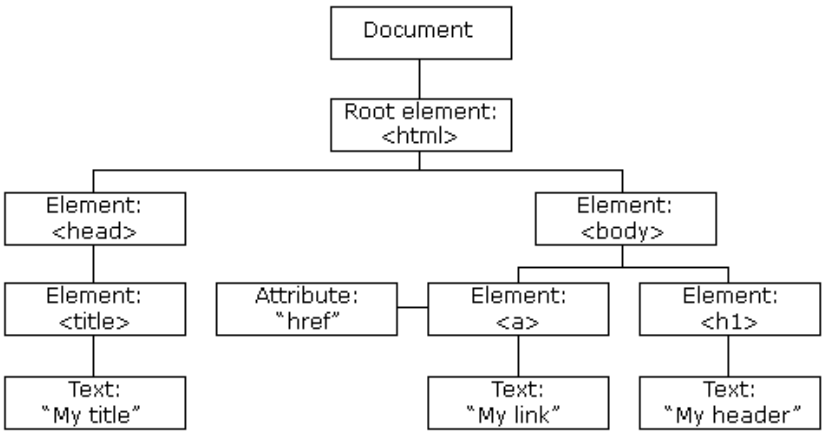

Figura 3.9: HTML sob a forma de uma estrutura em árvore

A figura 3.9 representa código HTML presente em 3.5.1.4.

```
<html>
        <head>
                My title
        </head>
        <body>
                <a href="">
                        My Link
                </a>
                <h1>
                        My header
                </h1>
        </body>
</html>
```

Snippet 3.1: Código que origina o diagrama presente em 3.9

Síntese e Escolha

Todas as ferramentas permitem que se extraia informação presente em determinadas *tags* de uma documento HTML. A diferença residirá em aspetos como velocidade de implementação, documentação, exemplos e suporte disponíveis, e acima de tudo desempenho e memória consumida.

Para testar o desempenho das bibliotecas o estagiário implementou uma pequena porção de código que iria ser necessário para um dos *crawlers*, em cada uma das quatro bibliotecas mencionadas. Este código foi escrito, em todas as ferramentas, para o mesmo documento HTML e foram realizados testes preliminares para garantir que a informação final era igual em todos os casos.

Depois de escrito o código foram realizadas trinta execuções para cada biblioteca e contabilizados o tempos de execução e memória consumida. No final calculou-se a média e o desvio padrão. Os resultados apresentam-se na figura 3.11 e **??** . Nas figuras as barras e os valores representam as médias. Já as linhas a vermelho representam o desvio padrão.

Pela análise das figuras verificamos que a biblioteca **lxml** apresenta o melhor desempenho e o segundo melhor consumo de memória, apenas ficando atrás para a biblioteca **htmlparser**.

O protótipo desenvolvido permitiu também averiguar o tempo de implementação que seria necessário para cada biblioteca. Ficou claro que as bibliotecas **lxml** e **BeautifulSoup** permitem uma maior velocidade de implementação, pois apresentam suporte a XPath e seletores CSS, no caso da primeira, ou só suporte a seletores CSS, no caso da segunda.

Importa ainda referir que esta velocidade de implementação é ainda encurtada pela ajuda fornecida pelos *browsers*, ou pelo menos por alguns deles, como é o caso do Firefox, Iceweasel e Chrome. Os *browsers* mencionados permitem, quer seja usando a consola de *developer* nativa ou através de extensões (como o *Firebug* e o *Firepath*), extrair diretamente o XPath ou o seletor CSS completo para a informação que pretendemos retirar. Além disso permitem ainda que o utilizador teste os seus próprios XPath e seletores CSS para verificar se está, realmente, a extrair a informação que pretende.

Concluindo, a escolha recaiu sobre a biblioteca **lxml**, pois apresenta o melhor desempenho e permite uma implementação mais rápida. Em conjunto com o **lxml** poderá

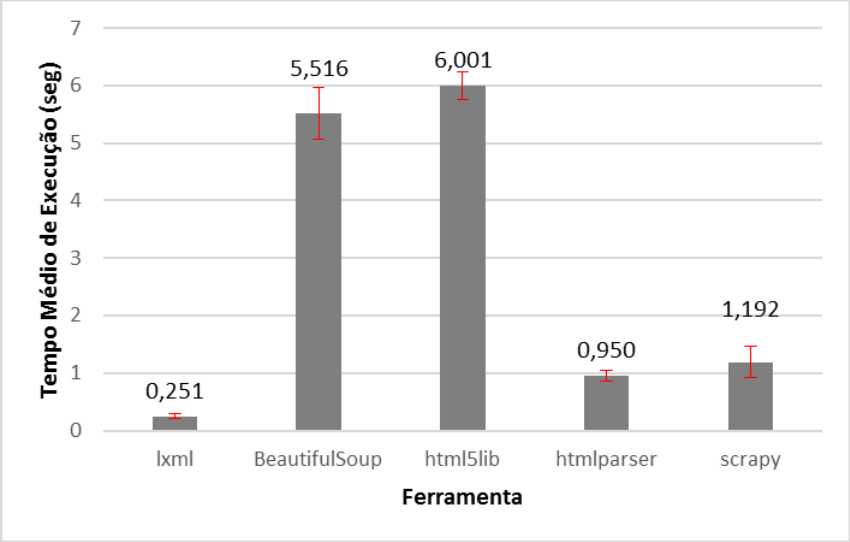

Figura 3.10: Tempo médio de execução, em segundos, das bibliotecas de *parsing* mencionadas

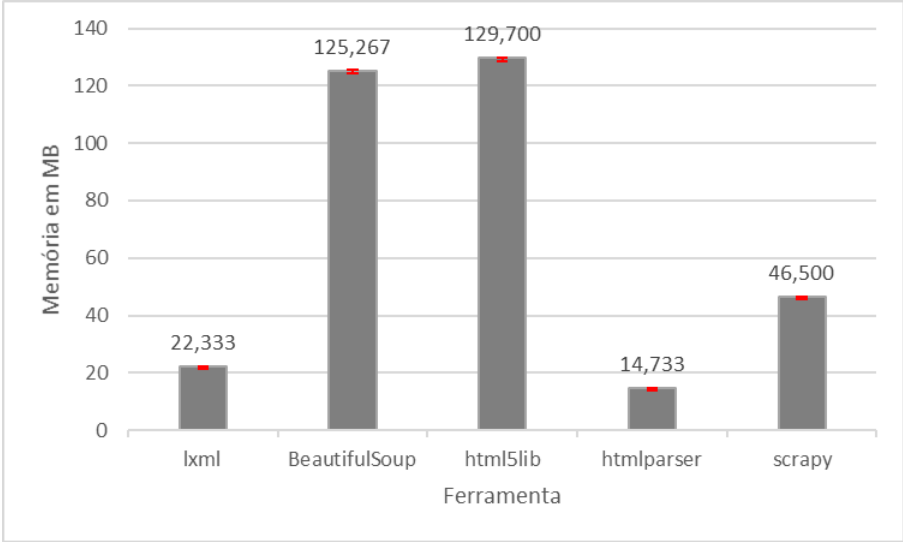

Figura 3.11: Memória Consumida, em Megabyte, das bibliotecas de *parsing* mencionadas

ser usado XPath ou CSS. A diferença entre estas duas encontra-se na sintaxe. Como o aluno já possui experiência prévia foi escolhido o XPatha.

3.5.2 Ferramentas de criação de bots IRC

A criação de *bots* para canais IRC necessita de ferramentas específicas que estejam preparadas para lidar com os protocolos adjacentes.

Em baixo apresentam-se algumas ferramentas que possam ser usadas para extração de informação de canais IRC. As ferramentas que foram estudadas são: Sopel[56], bosnobot[57], minchat[58] e ircLogBot[59].

Sopel *Framework* escrita em Python para criação de *bots* para IRC. É baseado em módulos que podem conter uma ou mais funções que por sua vez contêm um determinado número de atributos que as fará serem executadas. Dentro destes atributos estão a utilização de comandos, a correspondências com expressões regulares e eventos. Sopel também permite ao utilizador criar os seus módulos de forma simples. Está ao abrigo da licença *Eiffel Forum License*, versão 2.

bosnobot *Bot* IRC escrito em Python e assente na implementação do protocolo IRC feita pela *framework* Twisted: *twisted.words.protocols.irc*. Esta é uma *framework* web escrita em Python e orientada a eventos.
O bosnobot é um *bot* que facilmente se configura e se coloca em execução, para isso basta definir o *nickname* do *bot* e *password* que servirão para o identificar perante um servidor, os canais ao qual se vai ligar, servidor e porto. A juntar ao mencionado é necessário também indicar a classe Python que irá ser o usada para o tratamento das mensagens.

ircLogBot *Bot* escrito em Python e baseado na *framework* Twisted, em particular do módulo *twisted.words.protocols.irc*, que faz o *log* de todas as mensagens de um canal de um servidor. O servidor encontra-se definido no código e o canal ao qual o utilizador se pretende ligar será passado como argumento quando executarmos o *bot*. Importa referir que esta é uma implementação minimalista e apenas deverá servir para exemplo de utilização da *framework*. Entre outras, importa referir a classe *LogBot*, classe responsável por fazer o *log* de eventos. Essa classe possui um conjunto de métodos predefinidos para um conjunto de eventos, como por exemplo: entrada de num novo membro no canal, receção de uma mensagem e alteração de *nickname*.

Síntese e Escolha
De forma resumida a biblioteca **Sopel** funciona através de expressões regulares. Apenas os resultados que correspondem com o definido é que são devolvidos. Estas são definidos no código do *bot*, o que torna a configuração do mesmo complicada, pois sempre que seja necessário adicionar uma nova expressão regular é necessário alterar diretamente no código.
Tendo em conta que será desenvolvido um *bot* para procura de padrões (baseados em expressões regulares) na informação extraída por todos os outros *bots*, esta biblioteca não será usada. Isto porque iria apenas permitir implementar um *bot* semelhante ao de extração de expressões regulares, mas limitado às fontes IRC, com o acréscimo de ser mais dificilmente configurável.
O **bosnobot** já não é atualizada há sete anos. Consequentemente as suas dependências externas são também antigas: Python 2.5.1 (a versão 2 vai em 2.7.12) e Twisted 2.4 (a versão mais recente é a 16.3). Este *bot* é muito semelhante ao **ircLogBot**, pois são ambos baseados no módulo*twisted.words.protocols.irc*. Importa ainda salientar que a versão da *framework* **Twisted** usada por esta biblioteca não suporta o comando *whois*, usado em canais IRC para obter informação acerca de um determinado utilizador que esteja nesse mesmo canal.
O **ircLogBot** é implementação é minimalista e foi retirada do site oficial da *framework*. Suporta o comando *whois*, ao contrário do **bosnobot**. Neste caso não

será usada nenhuma biblioteca diretamente, mas sim como base de criação do *bot* final. Essas bibliotecas serão as que assentam na *framework* **Twisted**, em particular o bot ircLogBot.

Capítulo 4

Requisitos

Neste capítulo é apresentada a especificação de requisitos da plataforma desenvolvida. Esta é uma fase importante que antecede o *design* de um *software* e que tenta criar uma ponte entre este e as necessidades do cliente. Como já foi mencionado o objetivo do presente estágio foi desenvolver um conjunto de *bots* para extrair informação das mais diversas fontes sociais, bem como implementar uma interface gráfica de utilizador que permita visualizar os dados extraídos e configurar os próprios *bots*. A interface de visualização inclui uma *dashboard* com vários tipos de *widgets*, visualização dos dados sobre a forma de uma tabela , matriz ou grafo. Inclui também um interface que permite a classificação manual de eventos. Importa referir que o *dashboard*, a tabela e matriz já faziam parte do Portolan, mas foi necessária a sua alteração para ir de encontro à nova informação extraída. Uma lista completa dos requisitos encontra-se em A.

4.1 Requisitos Funcionais

Nesta secção vão ser apresentados os requisitos funcionais da plataforma, recolhidos e especificados com o cliente, neste caso a Dognædis.

Para complementar os casos de uso mencionados é apresentada uma descrição dos mesmos.

4.1.1 Adicionar Widget

Como utilizador da aplicação é possível adicionar novos *widgets* ao seu *dashboard*. Estes *widgets* darão ao utilizador vários dados estatísticos sobre a aplicação. Requisito já existente, mas que foi alterado. Os dados devolvidos dependerão dos parâmetros de procura que se encontram na figura 4.2.

4.1.2 Explorar Base de Dados: Tabela

Como utilizador é possível explorar a base de dados e apresentar os dados organizados numa tabela. Requisito já existente, mas que foi alterado, passando a criação da tabela para uma forma dinâmica. Esta forma dinâmica permite ao utilizador

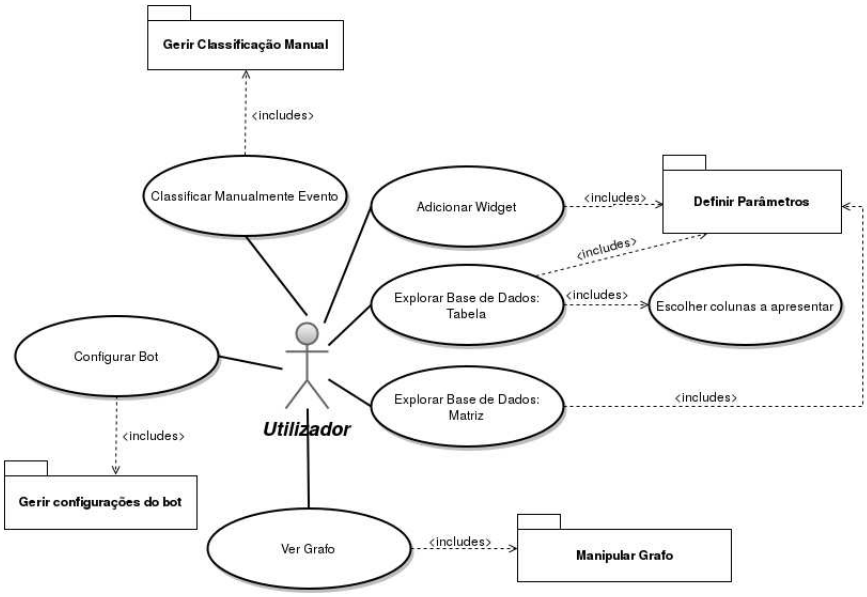

Figura 4.1: Diagrama de Casos de Uso para a Interface Gráfica de Utilizador

escolher as colunas que pretende. É ainda possível pesquisar colunas, selecionar todas e desselecionar todas. Os dados devolvidos dependerão dos parâmetros de procura que se encontram na figura 4.2.

4.1.3 Explorar Base de Dados: Matriz

Como utilizador é possível explorar a base de dados e apresentar os resultados numa matriz. Requisito já existente, mas que foi alterado. Os dados a apresentar dependerão dos parâmetros de procura, cujo diagrama de casos de uso se apresenta na figura 4.2.

4.1.4 Definir Parâmetros

Antes de visualizar a informação, o utilizador necessita de preencher alguns campos nas várias interfaces, campos esses que serão usados nas *queries* à base de dados.
Definir limite de resultados a apresentar
É possível ao utilizador limitar o número de resultados devolvido. Requisito já existente.

Definir Tipo de Estatística
É possível ao utilizador definir o tipo de estatística a utilizar. Dentro das estatísticas possíveis encontram-se a soma e a média. É ainda possível escolher qual o campo a usar para o cálculo da estatística.

Definir *Queries*
É possível ao utilizador definir as *queries* à base de dados. Para se escrever uma

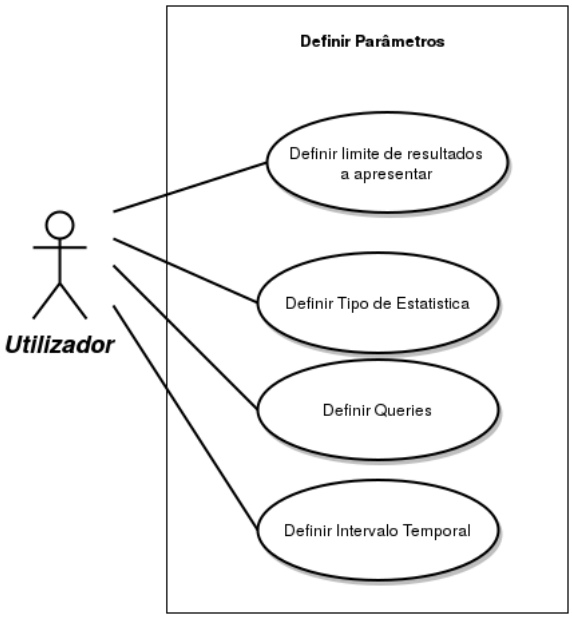

Figura 4.2: Diagrama de Casos de Uso para Definição dos Parâmetros de Pesquisa na Base de Dados

query usa-se uma notação simplificada de JavaScript que será depois interpretada para uma *query* de MongoDB. Requisito já existente.

Definir Intervalo Temporal
É possível ao utilizador definir um intervalo temporal para limitar a informação devolvida. Requisito já existente.

4.1.5 Classificar Manualmente Evento

Como utilizador da aplicação é possível classificar manualmente um evento. O utilizador deverá ser capaz de visualizar a informação contida no evento. O diagrama de casos de uso para *Gerir Classificação Manual* apresenta-se na figura 4.3.

4.1.6 Configurar Bot

Como utilizador da aplicação é possível configurar *bots* para as mais diversas fontes de informação. Os *bots* serão configuráveis com um conjunto de parâmetros que se encontram apresentados na figura 4.4. Importa referir que a figura mencionada apresenta o conjunto de todos os casos de uso possíveis para o conjunto de todos os *bots*, ou seja, nem todos os casos de uso são usados em todos os *bots*.

Adicionar Bot
Como utilizador da aplicação é possível adicionar um novo *bot* de recolha de informação e configurá-lo com vários parâmetros de recolha.

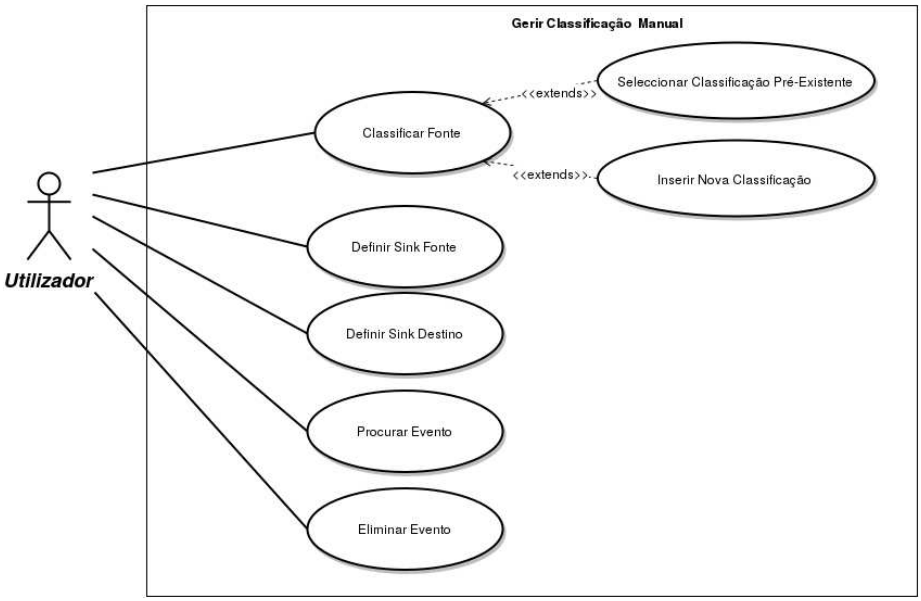

Figura 4.3: Diagrama de Casos de Uso para Gerir Classificação Manual

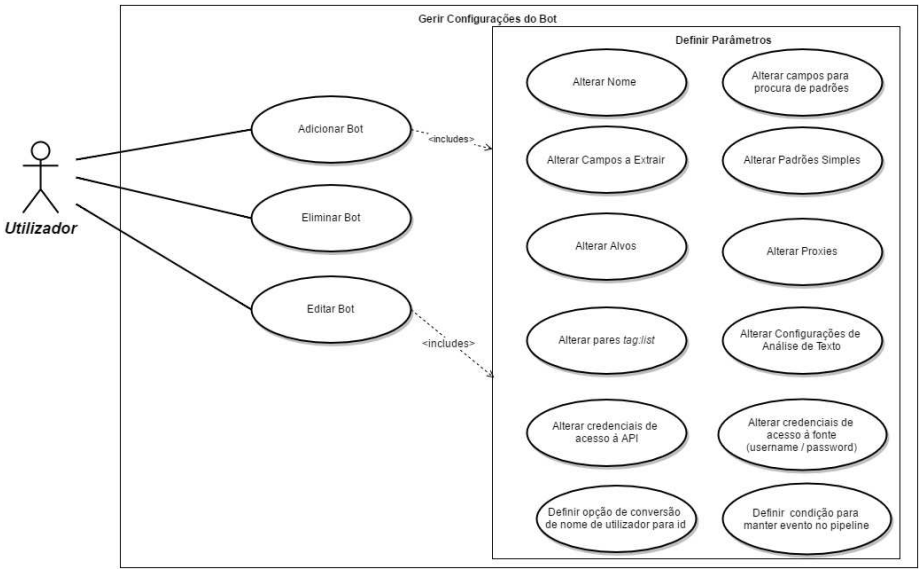

Figura 4.4: Diagrama de Casos de Uso para Gerir Configurações de um Bot

Editar Bot
Como utilizador da aplicação é possível editar as configurações de um *bot* .

Eliminar Bot
Como utilizador da aplicação é possível eliminar uma fonte, deixando esta de ser utilizada pela aplicação.

Definir Parâmetros do *bot*
Nesta secção serão definidos casos de uso mais detalhados sobre todas as configurações que são possíveis de realizar quando se adiciona um novo *bot* e que serão explicados de seguida.

- **Alterar Nome**
  Como utilizador é possível definir o nome de um *bot*, sendo a existência deste obrigatória.

- **Alterar Campos de Procura de Padrões**
  Como utilizador é possível definir em que campos determinadas expressões regulares deverão ser procuradas. Um exemplo de um campo seria o conteúdo de um *post*. Caso de uso especifico do *bot* de pesquisa de expressões regulares.

- **Alterar Campos a Extrair**
  Como utilizador é possível escolher que campos pretende extrair. Exclusivo aos *bots* do Facebook e Twitter. Alguns exemplos de campos que podem ser seleccionados serão a *timeline*, os *amigos* e seguidores.

- **Alterar Padrões Simples**
  Como utilizador é possível alterar as expressões regulares que serão usadas para análise da informação extraída. Dentro destes padrões tem-se tipos predefinidos como endereços IP, dominios, endereços MAC, entre outras que o utilizador pode definir.

- **Alterar pares *tag*/*list***
  Como utilizador deverá ser possível configurar pares *tag*/*list*, sendo a *list* uma lista com pelo menos uma *keyword*. As *keywords* da *list* serão usadas para classificar a informação segundo a *tag*.

- **Alterar Alvos**
  Como utilizador deverá ser possível definir os alvos dentro de uma determinada fonte. Para processar toda a informação que é colocada diariamente nas fontes mencionadas seria necessário um grande poder de processamento, e na maioria dos casos esse tipo de acesso nem é disponibilizado, daí a importância de se definir estes alvos dentro da própria fonte. Esses alvos são páginas de Facebook, perfis do Twitter, subreddits, *boards* do 4chan e canais IRC específicos.

- **Alterar Configurações de Análise de Texto**
  Como utilizador deverá ser possível alterar as configurações de análise textual. Exclusivo ao *bot* de análise de texto.

- **Alterar Proxies**
  Como utilizador da aplicação deverá ser possivel definir um conjunto de *proxies* a serem usados pelo *bot* aquando dos acessos.

- **Alterar credenciais de acesso à API**
  Como utilizador deverá ser possível alterar as credenciais a serem usadas para realizar o acesso às APIs das fontes de informação. Se o utilizador introduzir mais que um conjunto de credenciais, as mesmas serão usadas de forma rotativa.

- **Alterar credenciais de acesso à fonte (Nome de Utilizador / Palavra Passe)**
  Como utilizador deverá ser possível alterar as credenciais a serem usadas para realizar o *login* nas fontes de informação. Se o utilizador introduzir mais que um conjunto de credenciais de acesso, as mesmas serão usadas de forma rotativa.
- **Definir opção de conversão do nome d utilizador para id**
  Como utilizador deverá ser possível definir se se pretende que o *bot* converta todos os nomes de utilizador para ids. Esta opção degrada significativamente o desempenho do *bot* ou pode gastar muito rapidamente os limites de pedidos às APIs. O utilizador deve ser informado destes problemas.
- **Definir condição para manter evento na pipeline** Como utilizador deverá ser possível definir quando um evento deve ser mantido ou descartado do *pipeline*. No caso do *bot* de extração de expressões regulares o utilizador pode indicar que apenas pretende o evento na *pipeline* se alguma das expressões encontrar uma combinação. Já no *bot* de acesso à ferramenta de análise de texto da IBM deverá ser definido um valor limite mínimo que será usado para filtrar os resultados devolvido pela ferramenta.

4.1.7 Ver Grafo

Como utilizador da aplicação é possível visualizar as ligações entre as várias entidades. Entenda-se como entidade tudo o que possuir um identificador único (geralmente nome de utilizador ou id). As várias funcionalidades de manipulação do gráfico apresentam-se na figura 4.5.

Realçar Nodo e Ligação
Como utilizador deverá ser possível realçar um nodo quando se coloca o ponteiro do rato em cima do mesmo. Ao mesmo tempo as ligações ao nodos vizinhos e os próprios serão realçados. É ainda apresentada a informação sobre o nodo.
Como utilizador é também possível realçar uma ligação e os nodos das suas extremidades colocando o ponteiro do rato em cima da mesma. À semelhança do caso anterior é mostrada informação sobre a ligação.

Duplo Clique para Centrar Nodo
Como utilizador da aplicação é possível centrar um nodo fazendo duplo clique sobre ele.

Fazer *zoom*
Como utilizador é possível usar a roda do rato para fazer *zoom* ao grafo, seja aproximar ou afastar.

Arrastar Grafo e Nodos
Como utilizador é possível arrastar apenas um nodo ou o grafo como um todo.

Pesquisar Nodo
Como utilizador é possível pesquisar um nodo, sendo que este aparece realçado. Este requisito serve como base para a criação do grafo, sendo necessária a sua execução.

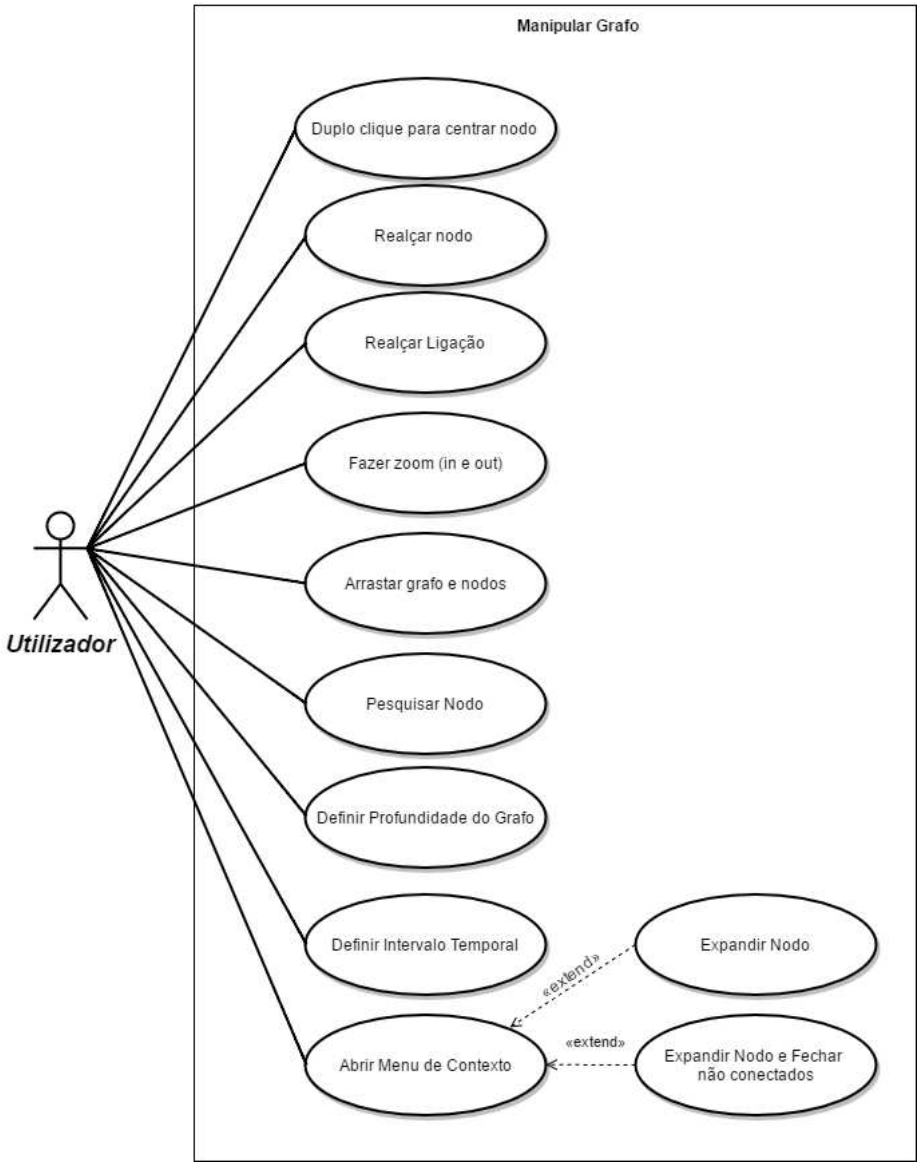

Figura 4.5: Diagrama de Casos de Uso para Manipular Grafo

Definir Profundidade do Grafo
Como utilizador é possível definir a profundidade do grafo a criar. Entenda-se como profundidade do grafo o número de ligações existentes entre o nodo selecionada como origem de criação do mesmo e outro qualquer nodo. Ou seja, se for definido um valor de profundidade dois, o grafo será criado com nodos que se possuam, no máximo, duas ligações entre eles próprios e o nodo raiz. É juntamente com o requisito *Pesquisar Nodo* a base de criação do grafo. No entanto, se não for preenchido pelo utilizador o grafo terá profundidade um.

Definir Intervalo Temporal
Como utilizador é possível definir o intervalo temporal para a criação do grafo. Faz também parte da base de criação do grafo, contudo não é necessário preencher este campo. Por defeito a janela temporal é de 24 horas podendo o utilizador definir outra janela que queira..

Abrir Menu de Contexto
Como utilizador é possível abrir um menu de contexto clicando com o botão direito do rato num nodo.

- **Expandir Nodo:** Como utilizador é possível expandir o nodo selecionado, mantendo na interface todos os restantes.
- **Expandir Nodo e Fechar não Conectados:** Como utilizador é possível expandir o nodo selecionado, fechando todos os que não estão diretamente ligados ao mesmo.

4.2 Requisitos Não Funcionais

Nesta secção serão apresentados os requisitos não funcionais da aplicação a desenvolver. Os requisitos não funcionais são características que o sistema deve possuir em adição às funcionalidades e que serão avaliados no final da implementação para verificar se a aplicação os cumpre. Os requisitos não funcionais apresentados e descritos são os seguintes: disponibilidade, manutenência, segurança, usabilidade e escalabilidade.

4.2.1 Manutenência

A Manutenência está relacionada com o custo da mudança e refere-se à facilidade com que um sistema de software pode acomodar mudanças.
Sendo as fontes a principal fonte de riscos, como foi referido na secção 2.5, é importante que todo o código relativo às mesmas seja de fácil alteração. Além disso, as mesmas poderão passar a fornecer novos tipos de informação ou novas formas de acesso que podem acrescentar valor ao produto e será importante garantir que expansões (e até reduções) sejam feitas da forma mais fácil e rápida possível. É importante também considerar a mudança ou descontinuação das bibliotecas que suportam o sistema. Como forma de garantia deste requisito, o código da aplicação será escrito de forma o mais genérica e modular possível. Por exemplo, cada tipo de acesso é realizado por um módulo específico para permitir que não seja necessário mudar o código em todos os *bots* que utilizam este módulo.
A juntar ao mencionado é requerida a existência de um modelo de dados próprio para o Portolan, que tem como objetivo a uniformização da informação. O mesmo já existe e foi atualizado para suportar os novos tipos de informação extraída. Isto permite que independentemente das ferramentas utilizadas todos os restantes módulos conseguem entender e processar a informação. Isto é, se mais tarde for necessário adicionar novos *bots* basta que aos mesmos seja dada a capacidade de processamento, conversão e interpretação da informação de acordo com o formato do modelo de dados.

4.2.2 Desempenho

O desempenho é medido através da velocidade de operação do sistema.
Existem dois tipos de requisitos de desempenho: tempo de resposta e *Throughput*. O primeiro consiste no tempo que o sistema demora a responder ao utilizador. O segundo indica a quantidade de trabalho que o sistema consegue realizar num dado intervalo de tempo.
O tempo de resposta ao utilizador deverá ser no máximo um segundo, à exceção das pesquisas na base de dados ou criação do grafo, facto que pode variar linearmente dependendo da pesquisa a realizar e do tamanho do grafo a gerar.
Já o *Throughput* varia bastante. Depende se o acesso é via API ou *crawler*. Depende também da utilização ou não de *proxies* e da largura de banda disponível. Se a API bloquear a aplicação porque atingiu os limites de pedidos vai também afetar este tempo. Algumas APIs forçam a existência de intervalos entre pedidos, o mesmo se passa nos *crawlers*. Assumindo a não existência destes fatores e que a informação é recebida pelos *bots* de forma instantânea a aplicação deverá ser capaz de processar vinte eventos por segundo, sendo este o número médio de eventos devolvido pelas APIs das várias fontes.

4.2.3 Segurança

Um sistema seguro tem a capacidade de resistir a tentativas de ataques externos ou internos. E sendo a Dognædis uma empresa na área da segurança de informação, torna-se essencial que a aplicação a desenvolver tenha em conta este requisito.
A solução a desenvolver assenta na expansão de uma plataforma cujos requisitos de segurança já foram validados. Contudo a solução a implementar vem trazer algo de novo que ainda não tinha sido necessário validar: a vulnerabilidade a injeções NoSQL.
Posto isto, o requisito de segurança da aplicação é não ser vulnerável a injeção NoSQL.

4.2.4 Usabilidade

Usabilidade é um requisito não funcional que avalia o quão fáceis as interfaces de utilizador são de usar. Esta avaliação é feita com base em cinco componentes:

Facilidade de aprendizagem Quão fácil é para utilizadores executarem tarefas básicas durante o primeiro contacto com a interface?

Eficiência Depois dos utilizadores aprenderem a usar a aplicação, com que eficiência conseguem realizar as tarefas?

Memorabilidade Se os utilizadores deixarem de usar a aplicação durante algum tempo, é fácil de voltar a conseguir realizar as tarefas de forma eficiente?

Erros Quantos erros é que os utilizadores fazem e quão graves são? Conseguem recuperar facilmente desses mesmos erros?

Satisfação Quão satisfatório é usar a aplicação?

Todas as componentes serão validadas com realização de testes de usabilidade. No entanto existem certas medidas que podem ser tomadas para facilitar a aprendizagem do utilizador ou até mesmo para o relembrar de certos aspetos da aplicação.

Outro ponto que deve ser tido em conta quanto à usabilidade é o tempo de resposta. Este deverá seguir os 3 limites de tempo de resposta enunciados por Jakob Nielsen[60]:

0.1 segundos Limite para que o utilizador sinta que o sistema reage instantaneamente;

1.0 segundos Limite para que o utilizador se comece a aperceber do atraso;

10 segundos Limite para que o utilizador se mantenha focado na aplicação. Se o atraso for superior a aplicação deverá mostrar *feedback* ao utilizador.

Form ainda seguidas algumas linhas de guia quanto ao desenho da mesma. Todos os aspetos tiveram como guias as oito regras de ouro de desenho de interfaces[61], sendo que se destacam as seguintes:

Manter a coerência Devem ser usadas formas de interação e terminologias bem conhecidas e familiares a todos os utilizadores. As mesmas devem ser aplicadas de forma coerente a toda a interface.

Fornecer Usabilidade Universal Adicionar funcionalidades para utilizadores novatos, como explicações, mas também para utilizadores avançados, como por exemplo atalhos ou outros elementos que permitam um ritmo mais rápido de execução.

Oferecer informação de retorno informativa Para cada ação do utilizador, deverá haver informação de retorno enviada pelo sistema.

Evitar Erros Tentar desenhar o sistema de forma a utilizadores não possam cometer erros graves. Por exemplo, desativar todos itens que não são necessários ou apropriados para determinada circunstância e não permitir caracteres alfabéticos em campos numéricos.

4.2.5 Escalabilidade

Um sistema é escalável se conseguir suportar o aumento de tarefas a realizar.
A solução desenvolvida exige que o sistema deverá ser funcional independentemente do número de alvos definidos. E deverá ser capaz de o fazer de forma fiável, resistindo a falhas e mantendo níveis de desempenho semelhantes ao longo do tempo.
Importa ainda referir que existem componentes que se podem apresentar como *bottlenecks* do sistema, sendo esses os *bots* de filtragem de informação (*bots* de procura de expressões regulares, análise de texto e criação do grafo). Estes receberão informação de vários *bots* em simultâneo, tornando necessário um maior poder de processamento nestes casos.
De notar que todos os *bots* são multi processo, suportando o processamento de informação em simultâneo. O número de *processos* trabalhadores que cada *bot* possui é definido na interface de configuração. Isto leva a que haja várias configurações diferentes possíveis e caberá ao estagiário estudar qual a que produz melhores resultados.

4.3 Restrições

Uma restrição é um fator limitativo que vai obrigar a determinadas soluções durante o *design* da arquitetura. Abaixo apresentam-se as restrições existentes:

Linguagem de Programação A linguagem de programação utilizada será *Python*, pois o Portolan encontra-se desenvolvido nesta linguagem. A sua utilização tem a vantagem de facilitar a criação e integração dos novos módulos desenvolvidos.

Plataformas Suportadas A arquitetura a implementar tem de ser desenhada com a finalidade de funcionar na infra-estrutura técnica existente na empresa e não deve colocar qualquer barreira à integração no Portolan.

Licenças de Software Quaisquer ferramentas externas (*frameworks*, bibliotecas ou código fonte) usadas para a realização deste projeto não deve trazer custos para a empresa, exceto escolhas feitas por parte da última.

Ferramentas externas As ferramentas externas a utilizar deverão fazer parte de um desenvolvimento ativo que garanta a sua atualização e a correção de problemas.

Capítulo 5

Arquitetura e Trabalho Desenvolvido

Neste capítulo é apresentado o Portolan e a sua arquitetura. São ainda especificadas as implementações dos vários *bos* e da interface gráfica de utilizador.

5.1 Portolan

O Portolan é constituído por um conjunto de *bots* e por uma interface gráfica. A atual arquitetura dos *bots* contempla um conjunto de componentes: *nodes* (*bots*), *sinks* e Eventos. Os dois primeiros compõem os vários *Pipelines* do Portolan. A informação que flui no *Pipeline*, quer seja entre *nodes* ou entre *nodes* e *sinks* representa um Evento.
Já a parte gráfica é constituída por um base de dados relacional, um servidor web, a *framework* Django e pelas tecnologias de apresentação de informação HTML, CSS e JavaScript.

5.1.1 Nodes

Os *nodes* podem ser de três tipos: produtor, consumidor e nodo.
O **produtor** é responsável pela recolha de informação e enviá-la para a(s) *sink(s)* às quais se encontra conectado.
O **consumidor** lê informação de uma *sink* e processa de acordo com o configurado.
O **nodo** é tanto produtor como consumidor. Lê informação de uma *sink*, faz o processamento da mesma e envia para todas as *sinks* às quais se encontra ligado.

5.1.2 Sinks

As *sinks* representam os locais de armazenamento de informação. Existem três tipos de *sinks* no Portolan: *Logger*, Redis[62] e MongoDB[63].
O ***Logger*** faz o registo da informação para ficheiro.
Redis é uma tecnologia que permite o armazenamento de estruturas de dados em memória. É vulgarmente utilizado como base de dados, *cache* ou *Message Broker*. Esta última representa a sua utilização no Portolan para comunicação entre *nodes*.

MongoDB é uma base de dados orientada ao Documento e, além deste último, trabalha também sobre o conceito de Coleção. Uma Coleção é um conjunto de Documentos e é o equivalente a uma tabela de um Sistema de gestão de Base de Dados Relacional. Contudo não forçam um determinado esquema de dados. Documentos presentes numa coleção podem possuir campos diferentes. Um Documento não é nada mais que um conjunto de pares chave-valor e como já foi mencionado não possuem esquema fixo. Comparando novamente com um Sistema de gestão de Base de Dados Relacional, um Documento é o equivalente a uma Linha de uma Tabela e um Campo é o equivalente a uma Coluna de uma Tabela.

5.1.3 Pipeline

Como já foi mencionado os *nodes* e *sinks* são usados para construir os vários *pipelines* do Portolan.
Um *pipeline* consiste numa cadeia de elementos de processamento arranjados de forma a que a saída de cada elemento é a entrada do próximo elemento.
Na figura 5.1 apresenta-se um exemplo. Na figura o *Bot 1* é do tipo produtor e os

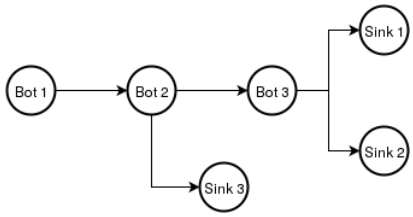

Figura 5.1: Pipeline Exemplo

Bot 2 e *Bot 3* são do tipo nodo. Importa ainda referir que ao se ligar dois *bots* é criada automaticamente uma *sink* intermédia de Redis.

5.1.4 Evento

Um Evento é o único tipo de dados trocado entre *nodes* ou entre *nodes* e *sinks*. Um Evento é representado pela classe *Event* e contém toda a informação a enviar num dicionário. Ou seja, em qualquer um dos *bots* desenvolvidos sempre que seja necessário enviar informação para o próximo *node* ou *sink* é necessário criar um Evento e preenche-lo com essa informação. Esse processo será descrito no ponto imediatamente abaixo.

Modelo de Dados
Como já foi mencionado, no MongoDB não há esquema de dados. Contudo, o Portolan força o seu próprio esquema através da *harmonization*. A *harmonization* é um ficheiro em formato JSON que contém todas as chaves permitidas nos Eventos do Portolan. Define ainda o tipo de dados que cada chave representa, uma descrição dos mesmos e se a sua existência é obrigatória. Caso o leitor pretenda, pode visualizar o conteúdo do ficheiro no anexo C.4.2. Todo o conteúdo dentro da chave *social* foi da autoria do estagiário e representa o leque total de informação que os *bots* podem extrair. Importa ainda salientar que foi o estagiário que deu suporte a chaves que representem dados do tipo *List*, pois o modelo de dados anterior não

necessitava, o que não acontece com o novo modelo.
Assim, a *harmonization* representa o modelo de dados. Sempre que a informação seja enviada para uma *sink* é criado um Evento. Esse Evento é preenchido com o conteúdo que o *bot* pretenda enviar e ao mesmo tempo é validado de acordo com o modelo. Ou seja, verifica se chaves marcadas como obrigatórias existem e verifica se o tipo de dados do valor associado a uma chave é o mesmo que o definido no modelo.
Importa referir que todos os *bots*, os já existentes ou os criados pelo estagiário têm de obedecer ao modelo de dados presente na *harmonization*, caso contrário não será possível enviar a informação. Isto leva a que seja necessário realizar o processamento dos dados a enviar, maioritariamente nos produtores, pois quando chega aos *nodes* já está de acordo com o modelo. Este processamento é maior no acesso às várias APIs, pois cada fonte usa o seu modelo de dados e devolve os dados de acordo com o mesmo. No caso dos *crawlers* a informação já é extraída seguindo o modelo de dados existente para que não seja necessário voltar a fazer processamento.
Importa ainda referir que o modelo de dados é utilizado como uma forma de prevenção contra ataques de injeção NoSQL.

5.1.5 Arquitetura e Diagrama de Classes (Bots)

Nesta secção serão apresentados a arquitetura e o diagrama de classes dos *bots* do Portolan. Na figura 5.2 apresenta-se o diagrama de classes dos *bots* do Portolan.

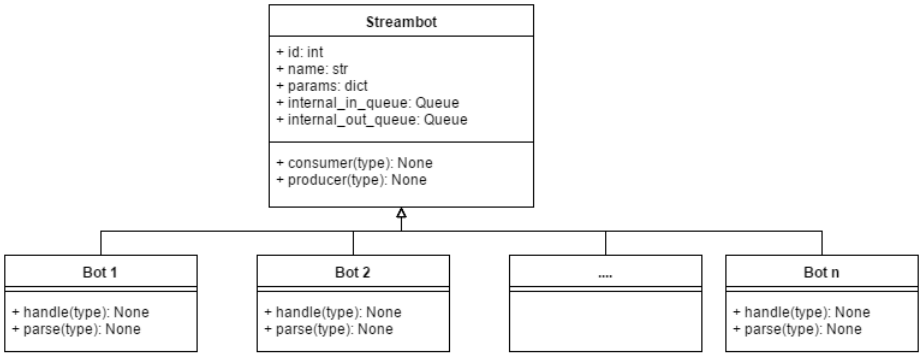

Figura 5.2: Diagrama de Classes dos bots do Portolan

Variáveis da classe StreamBot:

id Identificador do *bot*.

name Nome do *bot*.

params Dicionário com os parâmetros de configuração.

internal_in_queue Fila de entrada de dados.

internal_out_queue Fila de saída de dados.

Métodos da classe StreamBot:

consumer Método onde são lidos dados de entrada da *sink* definida e colocados na *internal_in_queue*. Neste método encontra-se em execução um processo que está constantemente à escuta na *sink* de entrada e sempre que lê algo coloca na fila. Este método é exclusivo aos *nodes* dos tipos consumidor e nodo.

producer Método onde são lidos dados da *internal_out_queue* e enviados para as várias *sinks* definidas. Neste método encontra-se em execução um processo que está constantemente à escuta na fila de saída. Sempre que lê algo envia para todas as *sinks*. Este método é exclusivo aos *nodes* dos tipos produtor e nodo.

Métodos das classes Bot:

parse Método presente em *nodes* do tipo produtor. Este método recebe um objecto do tipo *multiprocessing.Pool*, que representa uma *pool* de processos trabalhadores. O tamanho desta *pool* é definido na configuração de cada *bot*. Método onde são feitos os acessos às fontes de informação. No caso da informação recebida poder ser dividida e processada separadamente, esse processamento é feito pelos trabalhadores da *pool*. Por exemplo, no acesso à API do Facebook são recebidos quinze *posts*. A cada *post* será associado um trabalhador da *pool* para fazer o seu processamento.

handle Método presente em *nodes* dos tipos produtor e nodo. Este método é chamado sempre que são colocados dados na *internal_in_queue*. Existe um processo responsável por estar à escuta nesta fila. Esse processo tem ao seu dispôr uma *pool* de processos trabalhadores. Sempre que lê algo da fila de entrada, atribui um trabalhador da *pool* e executa a função *handle*.

Por fim apresenta-se a arquitetura de um *bot*, representada na figura 5.3. Esta é a arquitetura atual dos *bots* do Portolan. Da mesma, o estagiário trabalhou sobre duas componentes em particular: *Main Producer* e *Workers Pool*. No caso do *bot* ser do tipo produtor é implementado o *Main Producer* com a capacidade de extração de informação e as rotinas de processamento da informação a serem executadas pelos vários processos trabalhadores. Já no caso de ser do tipo produtor ou nodo o código implementado é executado pelos processos da *pool*.

5.1.6 Arquitetura da Aplicação Gráfica

Nesta secção irá ser apresentada a arquitetura do Portolan relativa à parte gráfica. A mesma foi desenvolvida com recurso à *framework* Django, HTML, CSS e JavaScript. Na figura 5.4(adaptado de: [64]) apresenta-se um diagrama com a arquitetura enunciada.

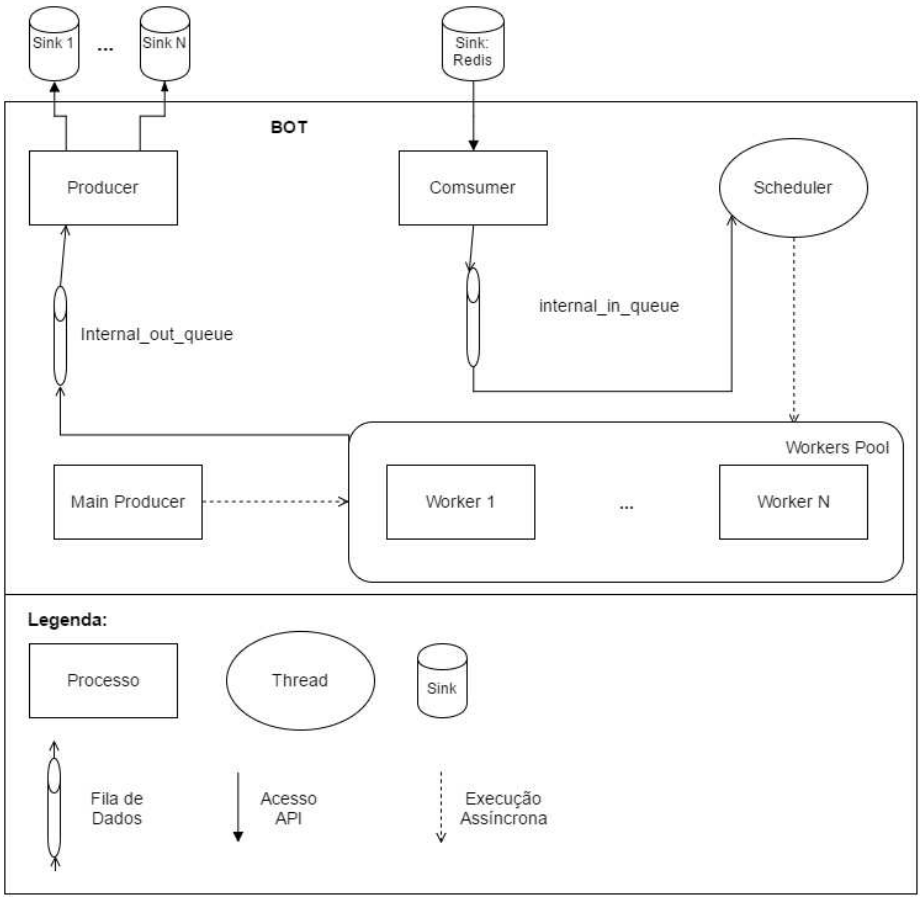

Figura 5.3: Arquitetura dos bots do Portolan

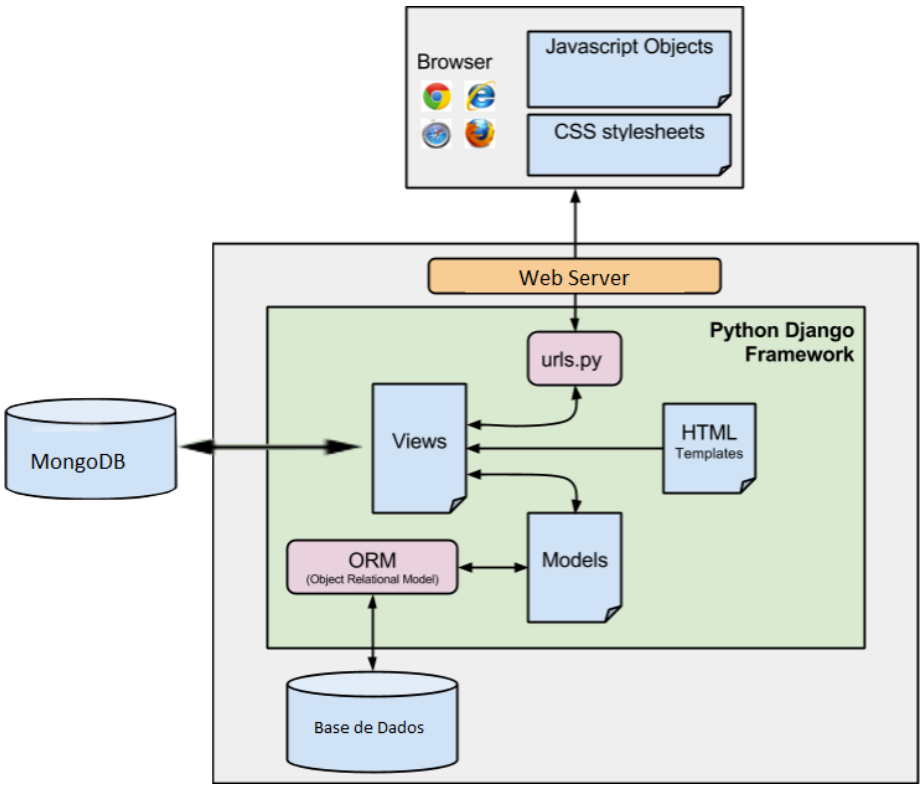

Figura 5.4: Arquitetura da aplicação gráfica do Portolan

A *framework* Django possui três camadas: *Models*, *Views* e *Templates*.

Models

Camada de abstração para estruturar e manipular a informação da aplicação. Um *Model* contém os campos e comportamentos da informação a aguardar. É representado por uma classe e mapeia uma tabela da base de dados em que cada atributo da classe representa uma coluna na tabela. A comunicação entre *Model* e Base de Dados é feita por intermédio do ORM que gera automaticamente o acesso à base de dados[65]. A título de exemplo, para criar a tabela **User** com as colunas **username** e **password** seria necessário o seguinte código:

```
from django.db import models

class User(models.Model):
    username = models.CharField(max_length=30)
    password = models.CharField(max_length=30)
```

O respetivo código SQL seria:

```
CREATE TABLE user (
    "id" serial NOT NULL PRIMARY KEY,
    "username" varchar(30) NOT NULL,
    "password" varchar(30) NOT NULL
);
```

View
O conceito de *vistas* permite encapsular a lógica responsável pelo processamento de um pedido de um utilizador e pela devolução da resposta. Interage diretamente com o ficheiro *url.py*. Este contém todos os URLs possíveis da aplicação e as *vistas* associadas aos mesmos. Sempre que um utilizador acede a determinado URL, o Django verifica qual a vista associada ao mesmo e executa[66].

Template
Esta camada fornece uma sintaxe amigável para renderizar a informação a ser apresentada ao utilizador. Um *template*, além das porções estáticas de HTML, permite que os utilizadores definam como o conteúdo dinâmico será inserido através do uso de uma sintaxe especial[67].

MongoDB e Base de Dados
O Portolan possui duas bases de dados: uma não relacional, **MongoDB** e outra relacional, representada pela componente **Base de Dados**. A primeira é, como já foi referido, onde fica armazenada a informação extraída. A segunda é utilizada para guardar informação sobre os utilizadores registados na aplicação e dados que estes tenham guardados como *widgets*, histórico de pesquisas à base de dados, entre outros.

5.2 Bots

Nesta secção irá ser apresentado todo o trabalho desenvolvido, desde *bots* de extração de informação aos *bots* de análise de expressões regulares e de texto e bot de construção do grafo para visualizar a informação.

5.2.1 Bot Facebook

Nesta secção será explicado o funcionamento do *bot* que extrai informação do Facebook, divido na parte de *crawling* e acesso a API.

5.2.1.1 Crawler

Nesta secção será especificada a implementação do *crawler* do Facebook. Importa referir que entre cada pedido a um URL é feito um *sleep* de dois segundos. O processo de *crawling* começa com o *login* na rede social.

Login
Para o *crawler* funcionar será necessário efetuar *login* na página web do Facebook, pois esta rede social não mostra a maior parte da informação a utilizadores não ligados. Importa ainda referir que o *crawler* apenas consegue extrair o que o utilizador conseguir visualizar através do acesso por meio do *browser*, por exemplo.
Assim sendo começa-se por se apresentar os passos necessários para realizar *login* nesta rede social, como se mostra na figura 5.5.

1º Passo Como podemos verificar o acesso é feito a *https://m.facebook.com* (Facebook Mobile) e não a *https://www.facebook.com*. Isto deve-se ao facto de a biblioteca *Requests* não conseguir lidar com o JavaScript da versão normal da página.

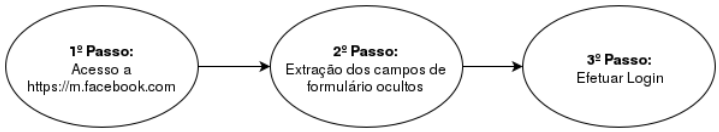

Figura 5.5: Passos para realizar login no Facebook programaticamente

2º Passo Depois de implementada a funcionalidade de *login* a mesma foi testada para validação. Essa validação falhou e após esta tentativa falhada de *login* foi necessário usar o *browser* para analisar os pedidos HTTP(s) realizados aquando do *login*. Para o efeito usou-se o módulo *Network* presente nas ferramentas de *developer* do *browser* (neste caso o Iceweasel, que é o *browser* nativo do sistema operativo Debian). O resultado apresenta-se na figura 5.6. Pela análise da figura 5.6 podemos retirar três pontos importantes para a

Figura 5.6: Informação sobre o pedido de login no Facebook

realização de login:

1. O *login* deve ser feito no endereço que se encontra no topo a azul.
2. Além do email e da *password* são enviados outros campos: campos de formulário ocultos
3. É usado o método POST

Depois de obtida esta informação é necessário extrair o valor dos campos de formulário ocultos. Para o efeito é necessário realizar sempre o 1º passo, porque os mesmos são gerados em cada acesso.

3º Passo Depois de extraídos os campos ocultos, acrescenta-se o email e a *password*. Estes são os parâmetros usados quando se faz um POST no URL supramencionado. A partir daqui o *bot* está ligado e pode começar a extrair informação, de acordo com o configurado (assumindo que o *login* foi realizado com sucesso.)

Paginação

A maior parte dos campos que se pretende extrair não apresentam todos os resultados de uma só vez, sendo necessário proceder à mudança de página. Na página

web a mudança dá-se clicando no botão "Ver Mais", no *bot* é necessário fazer um pedido ao URL que está associado ao botão. Em algumas situações a paginação causou alguns problemas ao nível do XPath: este era diferente para as a primeiras páginas, tendo sido necessário fazer duas implementações diferentes. Abaixo apresenta-se um exemplo dos diferentes XPath:

- **1ª página**
  ".//[@id='root']/div[3]/div[position()>1 and position()<last()]//a/@href"*
- **Restantes páginas**
  ".//[@id=ïrootÏ]/div[3]/div//a/@href"*

Noutras situações, como no caso da *timeline*, por vezes o botão "Ver Mais"não existe, sendo substituído por um botão com o ano.

Posts e Timeline
Um *post* é, em termos de conteúdo HTML, formado por duas ou três *tags *: cabeçalho (onde aparece o nome do criador do *post*), conteúdo(onde se encontra o conteudo textual) e pré-visualização (onde aparece a pré-visualização quando se coloca um *link* num *post*). A primeira encontra-se sempre presente enquanto que uma das restantes duas poderá não existir. Isto dificulta o trabalho do developer, pois é necessário validar se se trata do conteúdo do *post* ou da pré-visualização de um *link* colocado.
O próprio conteúdo também tem de ser analisado com vista à extração de *hashtags*, *tags* e *links*.

- **Hashtag**

```
<a href="/hashtag/dognaedis">#dognaedis<\a>
```

- **Tag**

```
<a href="/dognaedis">Dognaedis<\a>
```

- **Pré-visualização de um *link***

```
<a href="https://www.dognaedis.com/">Innovation in
    ↪ Security Technologies | DOGN\AE DIS<\a>
```

Antes de terminar este ponto importa referir que quando se faz a análise da *timeline* não é possível visualizar os comentários de um *post*. Para o efeito é preciso clicar no botão "Comentários"ou "História Completa". Novamente, no caso do *bot* terá que se fazer novo pedido ao URL associado a um dos botões. Também é necessário proceder da mesma forma para visualizar os "gostos"(sejam eles no *post*, nos comentários ou nas respostas aos comentários) e as respostas aos comentários.

Conversão do *Username* para Id
Um página ou perfil possui dois identificadores únicos (nome de utilizador e id) e um nome. Sempre que a aplicação se deparar com qualquer um dos casos (página ou

perfil) tenta preencher os três campos mencionados. O nome e o nome de utilizador são possíveis de extrair diretamente. Já para extrair o identificador é necessário primeiro fazer um pedido para essa página ou perfil. De seguida é extraído o id, sendo que este se encontra associado à foto de perfil. Há uma exceção: no caso desse perfil ou página não possuir nome de utilizador é possível extrair diretamente o id, no entanto não se consegue obter o nome de utilizador. Nesta situação o URL tem a seguinte forma: *.../profile.php?id=1234567890....*
O id é importante para relacionar pesquisas entre o *crawler* e a API, pois esta última não fornece os nomes de utilizador, apenas os ids. Contudo este processo afeta bastante o desempenho da aplicação, pois entre pedidos é feito um *sleep* para evitar que a aplicação seja bloqueada.
Exemplo: extração de mil "gostos"de um *post*. Assumindo que fazemos um *sleep* de dois segundos entre pedidos, a aplicação "dorme"no total 2000 segundos, o que equivale a aproximadamente 33 minutos. Para evitar esta degradação de desempenho o utilizador pode desativar a opção de conversão na configuração dos *bots*.

Datas
A tabela 5.1 apresenta os diferentes tipos de representação de datas usado pelo Facebook. Todas as datas são convertidas para objetos *datetime.datetime*. Os

String Tipo	Exemplo	Nota
Mês Dia at Hora(12):Minutos am|pm	Julho 31 at 4:20pm	Data de um post efetuado no actual ano
Mês(abreviado) Dia	Jul 31	Data de um comentário no actual ano
Mês Dia, Ano at Hora(12):Minutos am|pm	Julho 31, 2016 at 4:20pm	Data de um post efetuado em anos anteriores ao atual
Mês(abreviado) Dia, Ano	Jul 31, 2016	Data de um comentário efetuado em anos anteriores ao atual
1 min	1 min	Post ou comentário realizado há 1 minuto
X mins	2 mins	Post ou comentário realizado há mais de 1 minuto e menos de 60 minutos
1 hr	1 hr	Post ou comentário realizado há 1 hora
X hrs	2 hrs	Post ou comentário realizado há mais de 1 hora e menos de 24 horas
Just Now	Just Now	Post ou comentário realizado há menos de 1 minuto

Tabela 5.1: Tabela com os diferentes tipos de representação de datas usado pelo Facebook

primeiros quatro casos são convertidos usando o módulo *datetime.strptime*[1]. Este método faz análise de uma data sob a forma de uma cadeia de caracteres de acordo com o formato definido. Importa referir que com este método é necessário adicionar manualmente o ano (sempre o atual) à data, apenas no caso de não existir. Caso contrário o ano associado ao objeto será o ano de 1900.
Nos restantes casos elimina-se tudo o que não são números, converte-se para inteiro o restante e depois remove-se esse tempo à hora atual.

Requests e JavaScript
Como já foi mencionado não foi possível realizar *login* na página web do Facebook, pois a biblioteca Requests não consegue lidar com o JavaScript da mesma. Este foi um ponto não tido em conta na análise inicial feita sobre ferramentas do género, o que levou a que mais tarde fosse feito um novo levantamento para se tentar solucionar este problema.
A ferramenta analisada para tentar solucionar o problema encontrado foi a biblioteca Selenium[68], muito utilizada para automação do *browser*. A mesma permite ao utilizador escolher um conjunto de *browsers* com cabeça (com interface), como Firefox ou Chrome, mas também sem cabeça, o PhantomJS. Os primeiros foram logo descartados, pois implicam a utilização de interface gráfica, o que a máquina onde o Portolan se encontra não possui. Já a segunda não possui interface. Contudo comporta-se como um *browser* levando a mais tempo necessário para carregar o conteúdo da página que a biblioteca Requests (como o leitor vai poder verificar pela análise da figura 5.7). Consequentemente afeta o desempenho da aplicação sendo por isso descartada.
Uma outra solução passou pela utilização da versão *mobile* do Facebook, pois a biblioteca Requests não apresentou qualquer problema neste caso. Esta solução foi a escolhida, pois permite manter os mesmo níveis de desempenho sem aumentar as dependências tecnológicas da aplicação.
Na figura 5.7 apresentam-se as médias para cada ferramenta do tempo necessário até que fosse possível realizar *login* no Facebook. Para cada teste foram realizados trinta pedidos ao mesmo URL (versão *mobile* do Facebook para a biblioteca Requests e versão normal para as restantes) e guardou-se o tempo demorado até ser possível extrair os campos ocultos e se efetuar o *login*. Importa referir que as linhas a vermelho, sobre as barras, representam o desvio padrão.

5.2.1.2 API

O acesso via API é realizado para extrair *posts* feitos por uma página e informação da mesma, ao contrário do *crawler* que é usado para os perfis pessoais. Este acesso é feito através de um pedido a um simples URL. Esse URL obrigatoriamente tem de conter os parâmetros *access_token* e *fields*. Para se entender o funcionamento dos mesmos basta aceder ao Explorador da Graph API [2]. Através desta plataforma é possível gerar os *tokens* de acesso, bem como testar os vários *fields* possíveis. Permite ainda exportar em código para várias linguagens diferentes os acessos que vão sendo feitos. A título de exemplo mostra-se código necessário para extrair informação sobre o próprio utilizador, em particular o nome e o id:

[1] https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior
[2] https://developers.facebook.com/tools/explorer

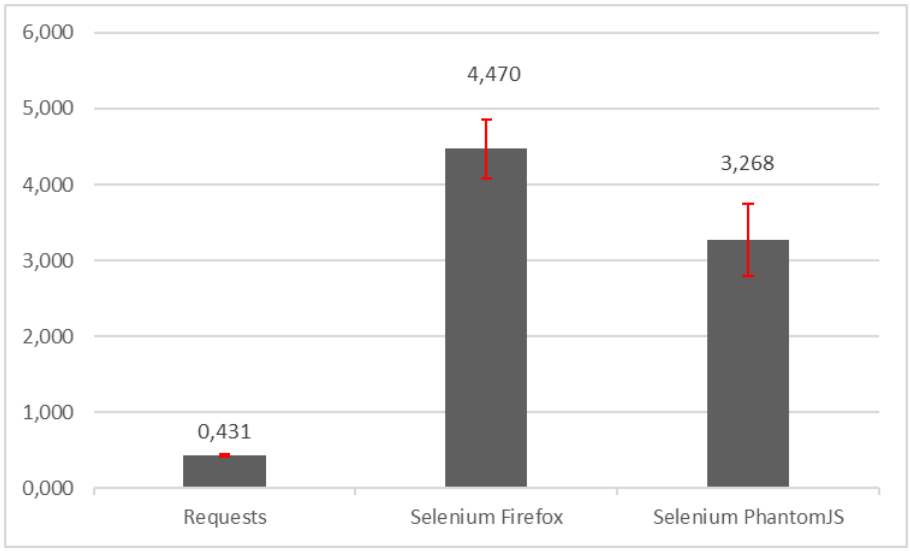

Figura 5.7: Tempo médio de acesso até que fosse possível realizar login

```
import requests
response = requests.get("https://graph.facebook.com/v2.7/me?
    ↪ fields=id&name&access_token=<access_token>")
```

Paginação

À semelhança do *crawler* também na API é necessário proceder à paginação do conteúdo. A API não devolve o conteúdo todo de uma só vez. Este é divido em várias páginas, sendo que cada uma possui uma quantidade máxima de dados (por exemplo, cada pedido devolve no máximo 25 *posts* de um página). É então necessário fazer um novo pedido para cada página.

Quando se faz um pedido à Graph API a resposta é em formato JSON. Um dos campos do conteúdo recebido possui o nome *paging*. Dentro deste campo encontra-se um novo documento, constituído por dois campos: *previous* (anterior) e *next* (próximo). Estes campos são usados para navegar entre as várias páginas da informação e contêm o URL para as páginas respetivas. Basta, então, realizar um novo pedido no URL que se encontra no conteúdo destes campos. Em baixo apresenta-se uma amostra do conteúdo de uma resposta com os campos de paginação.

```
{
  "data": [
     ... Endpoint data is here
  ],
  "paging": {
    "previous": "https://graph.facebook.com/me/feed?limit=25&
        ↪ since=1364849754",
    "next": "https://graph.facebook.com/me/feed?limit=25&until
        ↪ =1364587774"
  }
}
```

5.2.2 Bot 4Chan

Nesta secção apresenta-se o *bot* desenvolvido para extrair informação do 4Chan. O meio de acesso a esta fonte de informação é via a sua API.
Das regras da API destacam-se as seguintes:

1. Não realizar mais que um pedido por segundo
2. Usar o campo *If-Modified-Since* no cabeçalho dos pedidos

Apresentadas as regras mais relevantes, o *bot* foi desenvolvido tendo em conta as mesmas para evitar que o mesmo fosse bloqueado. A primeira é cumprida fazendo *sleep(1)*(dormir 1 segundo) antes de todos os pedidos. A segunda é cumprida definindo o campo *If-Modified-Since* em todos os pedidos com a data da ultima execução do *bot*.
Além das classes transversais a todos os *bots* apresenta ainda as que se encontram na figura 5.8.

Datas

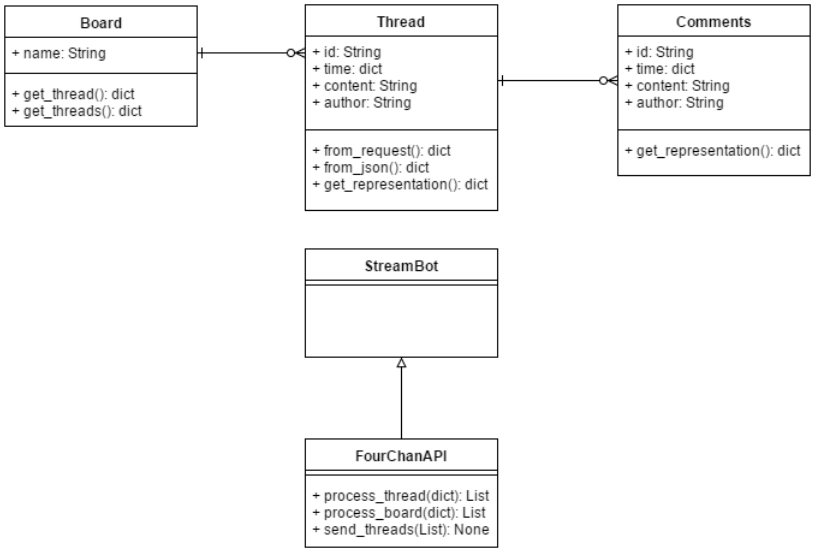

Figura 5.8: Diagrama de Classes do 4Chan

À semelhança de outros *bots* neste também é necessário converter as datas para um objecto *datetime.datetime*, sendo que a data aparece no seguinte formato: "Mês/-Dia/AnoDiaDaSemanaHoras:Minutos:Segundos".

Acesso
O acesso é feito através de um pedido HTTP ou HTTPS simples, não sendo necessárias quaisquer credenciais, pois a API é pública.

Monitorização
É possível monitorizar todo um *Board* ou apenas uma *Thread*. Depois de extraída a informação de cada *Thread* é associado um processo da *pool* para fazer o seu processamento.

Relembra-se que um *Board* representa uma área de interesse (por exemplo, desporto) constituída por várias *Threads*. Uma *Thread* é constituída pelos autor, titulo, descrição, data de publicação e comentários.

Paginação

À semelhança de outros *bots* também este faz uso da paginação da API à qual acede. Para aceder a uma página de um *Board* do 4Chan basta realizar um pedido ao seguinte URL:
http://a.4cdn.org/<board>/<pagenumber>.json, em que os campos entre <> são variáveis. Este pedido devolve, em formato JSON, todas as *threads* da página especificada, acompanhadas de no máximo três comentários. Por esse motivo este pedido serve apenas para extrair os números das *threads* existentes numa determinada página de um *board*. Posteriormente é realizado um novo pedido, para cada *thread* extraída, no seguinte URL:
http://a.4cdn.org/<board>/thread/<threadnumber>.json. Este novo pedido devolve o conteúdo da *thread* e os respetivos comentários. Por fim, processa-se e valida-se a informação de acordo com o modelo de dados e envia-se para o próximo *node* ou *sink*.

5.2.3 Bot IRC

Este *bot* foi desenvolvido para extrair informação dos vários canais IRC para o qual se encontra definido. Este *bot* é distinto dos restantes na medida em que o acesso não é a uma API nem via *crawling*, mas sim usando o protocolo IRC, implementado pela *framework* Twisted, que foi a biblioteca escolhida como base para implementação deste *bot* (ver secção 3.5.2).
Também este *bot* apresenta classes extra às comuns a todos os outros, sendo apresentadas na figura 5.9.

LogBotFactory

Classe responsável pela criação e lançamento dos *bots*. Descende da classe **ClientFactory** que é a responsável por fornecer uma interface, enquanto que a classe *LogBotFactory* implementa a mesma.
É criada uma instância desta classe para cada nova ligação. É responsável por receber os eventos relativos ao estado da ligação, o que permite fazer, entre outras coisas, reconectar em caso de quebra de ligação.

LogBot

Classe que descende de *IRCClient*, sendo que esta última fornece uma interface para o protocolo cliente do IRC. A classe *LogBot* é onde será implementa a interface mencionada, sendo que os métodos implementados são as funções de retorno para os vários eventos.

Reactor

O núcleo da Twisted é o ciclo de eventos do *reactor*. O *reactor* interaje com eventos de rede, do sistema de ficheiros e eventos agendados. Espera pelo acontecimento desses eventos e envia-os para os manipuladores respetivos.
O *reactor* possui um método de transporte que recebe o endereço de ligação (*hostname* ou IP), porto e uma instância da classe **LogBotFactory**. Esse método, com nome connectXXX (em que XXX pode ser TCP, UDP, entre outros), quando chamado fica responsável por registar as funções de retorno para ser notificado quando

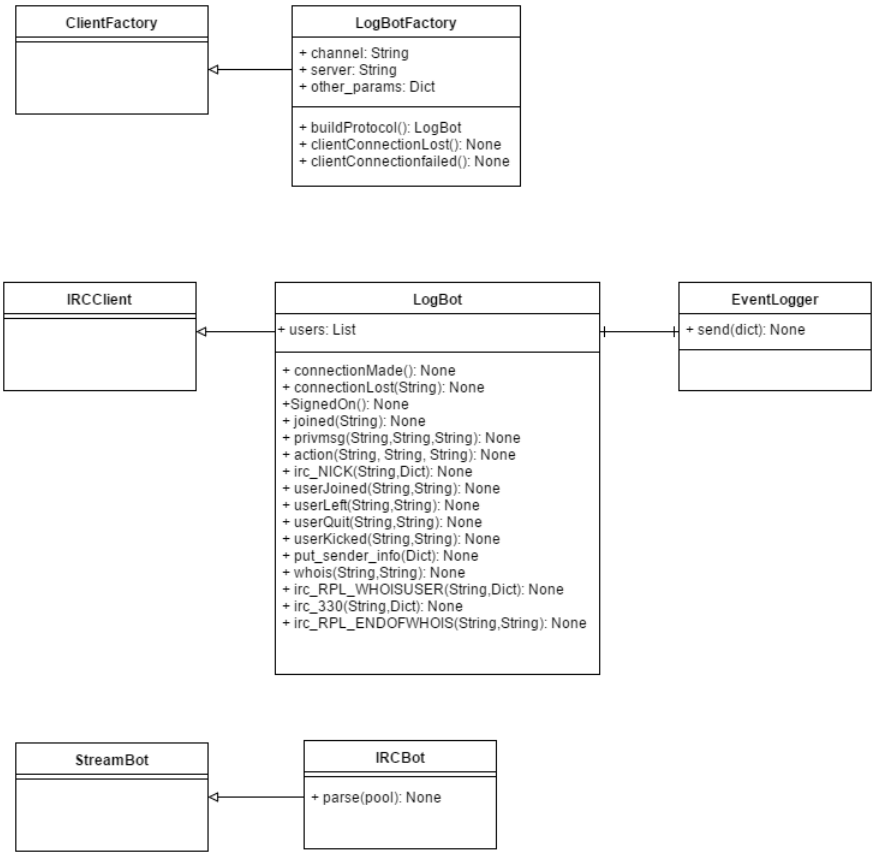

Figura 5.9: Diagrama de Classes do bot IRC

informação está disponível.
Depois de todos as funções de retorno estarem registadas, o *reactor* é iniciado com *reactor.run()*. Após estar em execução, o *reactor* irá fazer *polling* e enviar eventos até que seja parado.
Na figura 5.10(adaptada de: [69]) apresenta-se um esquema sobre o funcionamento do *reactor* para o caso de um utilizador enviar uma mensagem para o canal. O funcionamento é semelhante para todos os eventos, apenas muda a função de retorno.

Comando *WHOIS*

Como pode verificar no anexo A.4, o primeiro teste ao comando *WHOIS* falhou. Isto porque a função responsável por enviar o comando ao servidor devolve *None*. Quando se executa o método *whois* é enviado para o servidor IRC um comando com o mesmo nome. O método retorna de imediato depois de enviar o comando ao servidor. Neste momento este último ainda não enviou a resposta ao comando (possivelmente ainda nem recebeu o pedido). Para obter a resposta é necessário sobrepor alguns métodos na subclasse da classe IRCClient da Twisted. Muita da informação que é enviada pelo servidor pode ser apanhada usando métodos de retorno com prefixo *irc_*. Usando este caso como exemplo, uma das respostas ao

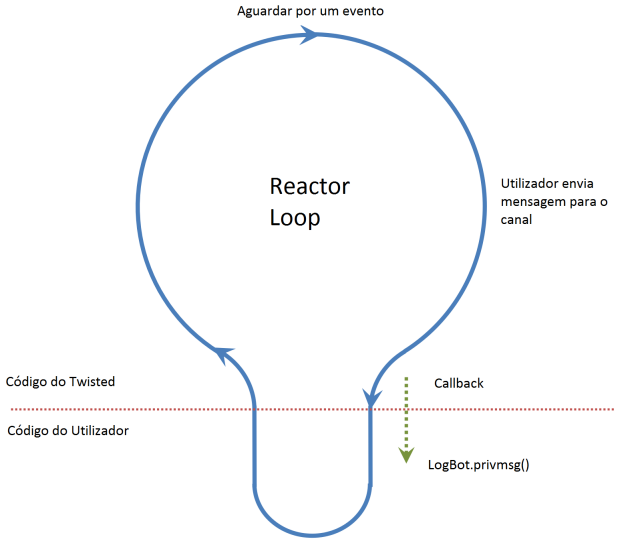

Figura 5.10: Esquema do funcionamento do Reactor

comando *WHOIS*, documentada no RFC 1459[3], tem o nome *RPL_WHOISUSER*. Para obter esta resposta é necessário sobrepor o método *irc_RPL_WHOISUSER*. Quando o cliente recebe a resposta do servidor, o método anterior é chamado com os parâmetros da resposta. Esses parâmetros são, transcrevendo a RFC: *«nick> <user> <host> * :<real name>"*.

5.2.4 Bot Reddit

Este *bot* foi desenvolvido para extrair informação do Reddit via API. para aceder à API o utilizador necessita de *tokens* de acesso e credenciais de *login*. É possível extrair informação de um *subreddit* e de uma *thread*. É ainda possível monitorizar a atividade de um utilizador.
Há semelhança dos *bots* anteriores também este possui um conjunto de classes extra que se apresentam na figura 5.11.

Monitorização de um Subreddit e de uma Thread
A monitorização inicia-se com a listagem dos ids de todas as *threads* existentes na primeira página de determinado subreddit. Obtida a lista (esta lista não contém os comentários), cada *thread* é preenchida com comentários e comentários ao comentários. O Reddit permite profundidade arbitrária para a árvore de comentários, mas no caso do Portolan foi limitada a dois níveis por questões de conveniência nas *queries* à base de dados. Contudo, não significa que a informação não seja extraída e guardada, mas sim que é reorganizada por forma a manter a profundidade mencionada.
A monitorização de uma *thread* é feita de forma semelhante à de um *subreddit*,

[3]https://tools.ietf.org/html/rfc1459

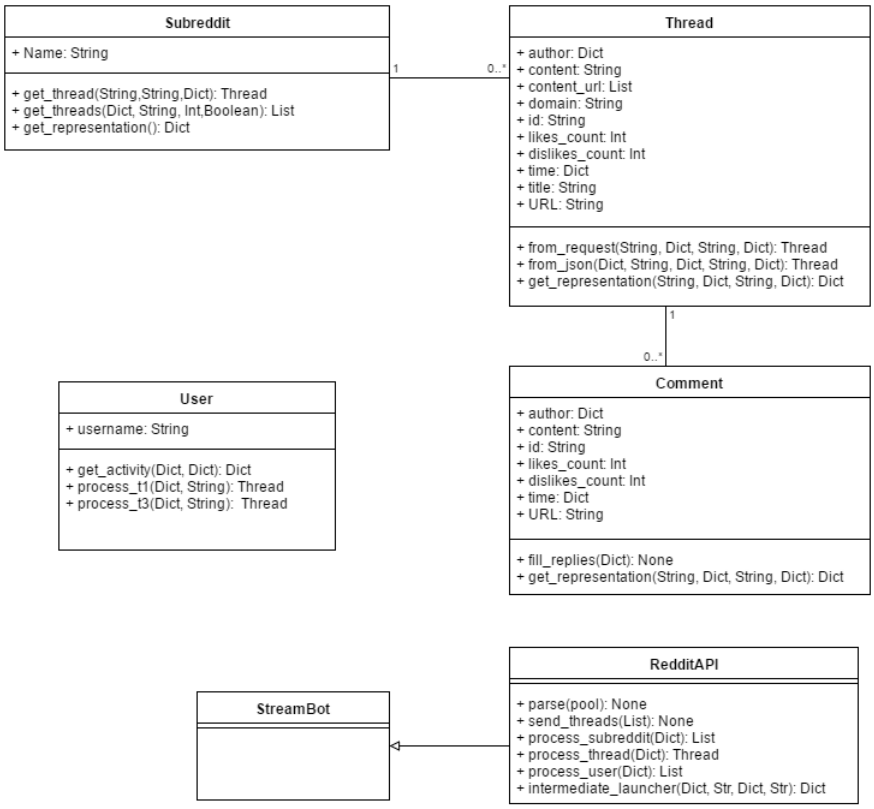

Figura 5.11: Diagrama de Classes do bot do Reddit

sendo que a parte inicial de listagem não é efetuada.

Monitorização de um Utilizador
Todas as *threads* onde um utilizador foi ativo (criou, comentou, gostou, entre outros) são extraídas e preenchidas da mesma forma que nos casos anteriores.

Paginação
À semelhança de outros *bots* também neste é necessário realizar a paginação, pois nem via API nem via página web se obtém todos os resultados de uma só vez. A mesma é realizada da seguinte forma:

1. Guarda-se o id da ultima *thread* da atual página.
2. Ao URL base que dá acesso ao conteúdo da primeira página acrescenta-se o seguinte: *?after=<thread_id>*. Ou seja, é acrescentado o parâmetro *after* ao pedido.
3. Realiza-se novo pedido.

A partir deste ponto, o processo é o mesmo.

Limite de Pedidos
A API tem limite de pedidos por minuto. Sempre que se faz um pedido, a resposta contém os seguintes campos no cabeçalho:

X-Ratelimit-Used Número aproximado de pedidos usado no atual período.

X-Ratelimit-Remaining Número aproximado de pedidos restantes.

X-Ratelimit-Reset Número aproximado de segundos até restabelecer os valores iniciais.

Contudo o *bot* não monitoriza em cada pedido estes cabeçalhos. Quando se atinge o limite de pedidos é recebido o código HTTP 429 (Demasiados Pedidos). Só aqui é que se analisam os campos anteriores. Para não ficar com informação incompleta, o *bot* é colocado "a dormir"até que esteja novamente apto a extrair informação.

Acesso
Para aceder à API são necessários os seguintes passos:

1. Autenticação Básica HTTP:

```
client_auth = HTTPBasicAuth(<consumer_key>, <
    ↪ consumer_secret>)
```

2. Criar objecto com informação a enviar no pedido:

```
post_data={"grant_type":"password","username":<username>,
    ↪ "password":<password>}
```

3. Fazer pedido.

```
response = requests.post("https://www.reddit.com/api/v1/
    ↪ access_token", auth=client_auth, data=post_data)
```

4. Obter o *token* de acesso. Isto é feito concatenando o tipo de *token* com o *token* em si.

```
data = response.json()
token = data['token_type'] + "␣" + data['access_token']
```

5. Por fim adiciona-se o *token* aos cabeçalhos e faz-se o pedido, caso contrário o pedido não será realizado com sucesso.

```
headers.update({"Authorization": token})
data = requests.get(<url>, headers=headers)
```

5.2.5 Bot Twitter

Este *bot* foi desenvolvido para extrair informação do *Twitter*. Possui três formas distintas de acesso: API REST, *Streaming* API e *crawling*. Na figura 5.12 apresenta-se o diagrama de classes correspondente a este *bot*. Este poderá ser configurado para

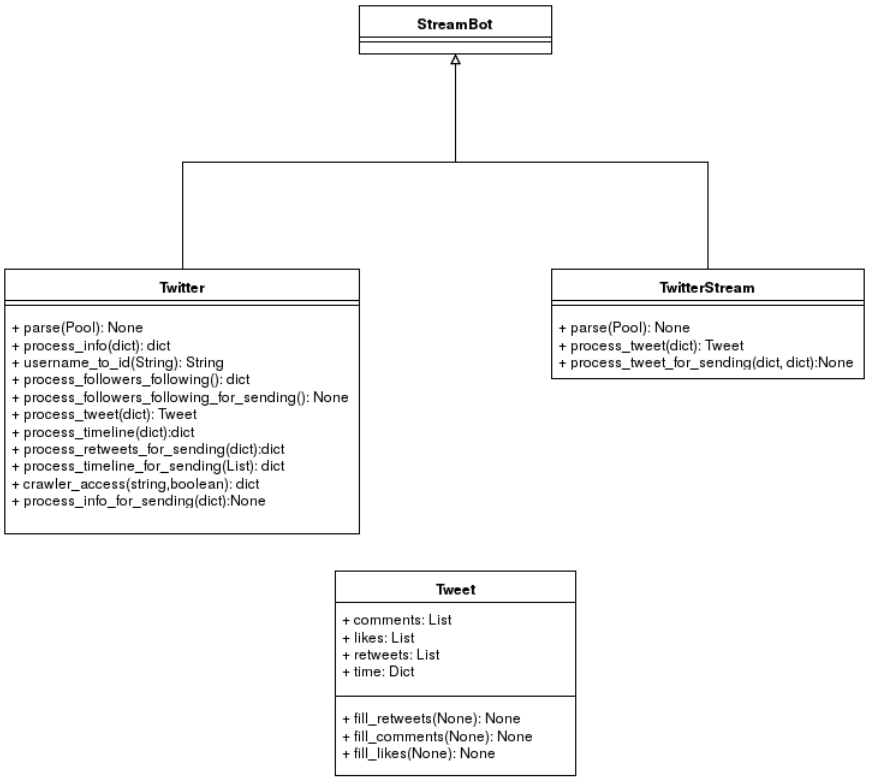

Figura 5.12: Diagrama de Classes do bot do Twitter

extrair a *timeline*, os seguidores, os seguindo e a informação de um perfil.

5.2.5.1 Crawler

O *crawler* foi desenvolvido para a extração dos seguidores e dos seguindo de um perfil e para extração dos comentários e gostos de um *tweet*. Para o efeito foi desenvolvida a capacidade de *login*, que é o começo de toda o processo de *crawling*.

Login
Assim sendo começa-se por se apresentar os passos necessários para realizar *login* nesta rede social, semelhante ao caso do Facebook, como se mostra na figura 5.13.

1º Passo Novamente o acesso é feito à página web para dispositivos móveis.

2º Passo Para se perceber quais eram os campos necessários usou-se o módulo *Network* presente nas ferramentas de *developer* do *browser*. O resultado

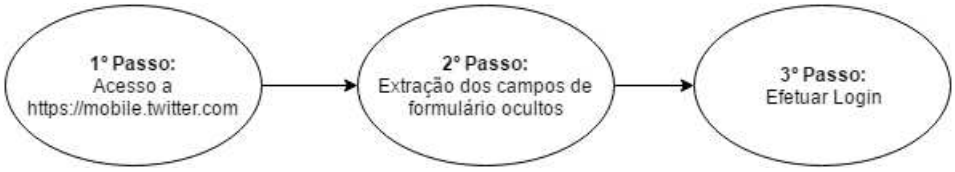

Figura 5.13: Passos para realizar login no Twitter programaticamente

Figura 5.14: Informação sobre o pedido de login no Twitter

apresenta-se na figura 5.14. Pela análise da figura 5.14 podemos retirar três pontos importantes para a realização de login:

1. O *login* deve ser feito no endereço que se encontra no topo a azul.
2. Além do email e da palavra passe são enviados outros campos: campos de formulário ocultos
3. É usado o método POST

Depois de obtida esta informação é necessário extrair o valor dos campos de formulário ocultos. Para o efeito é necessário realizar sempre o 1º passo, porque os mesmo são gerados em cada acesso.

3º Passo Depois de extraídos os campos ocultos, acrescenta-se o email e a palavra passe. Estes são os parâmetros usados quando se faz um POST no URL supramencionado. A partir daqui o *bot* está ligado e pode começar a extrair informação, de acordo com o configurado.

A extração dos seguidores, dos seguindo e das respostas a um Tweet é feita de forma simples. Basta aceder através da página web para se perceber a qual URL se faz os pedidos e e realiza-los via *Requests*. Cada situação tem o seu URL específico, basta mudar o id do perfil ou *Tweet* em questão.
Para extrair os gostos já é mais complexo, pois o conteúdo é gerado dinamicamente: é feito um pedido a um URL assincronamente usando Javascript e o conteúdo é apresentado num Modal. Foi necessário usar as ferramentas de developer do browser para se perceber como era feito o pedido. O resultado apresenta-se na figura 5.15. A análise à figura 5.15 permite verificar o URL usado e verifica-se que apenas basta mudar o valor do parâmetro *id* para se obter resultados para outro qualquer *Tweet*. A resposta ao pedido é em formato JSON, em que o segundo campo (*htmlusers*) contém uma porção de código HTML que vai ser acrescentada à pagina atual. É desta porção de código HTML que vai ser extraída a informação.

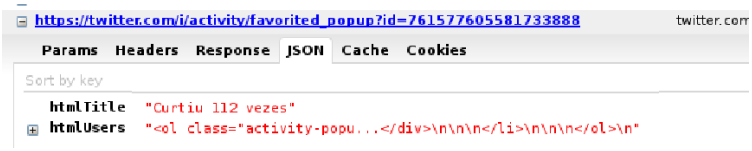

Figura 5.15: Informação sobre o pedido de visualização dos "gostos"de um Tweet

Conversão de Username para Id
À semelhança do bot do Facebook o *crawler* do Twitter também não consegue obter o id de um perfil. Mas a API deste último, ao contrário da do Facebook fornece o nome de utilizador, que permite relacionar a informação extraída pela API e pelo *crawler*. Para obter o id é realizada uma chamada à API sobre informação desse perfil e extrai-se o id.
Isto gasta muito rapidamente os limites de acesso à API, sendo dada a opção de desativação da mesma ao utilizador.

5.2.5.2 API REST

A API REST é usada para extrair os *tweets* criados por determinado perfil ou para obter informação sobre determinado perfil.

Acesso
Para aceder à API do Twiter é necessário realizar o fluxo OAuth 1. Contudo a biblioteca *requests_oauthlib* trata da maior parte do fluxo e deixa o *developer* focar-se noutras tarefas. Aceder ao recursos protegidos através da API usando esta biblioteca implica o seguinte código:

```
twitter = OAuth1Session(client_key, client_secret,
    ↪ resource_owner_key, resource_owner_secret)
data = twitter.get('https://api.twitter.com/...')
```

Limite de Pedidos
O funcionamento da API do Twitter é muito semelhante ao do Reddit. Na resposta aos pedidos são sempre enviados três campos no cabeçalho da resposta, específicos à monitorização do limites de acesso:

X-Rate-Limit-Limit Limite de pedidos ao *endpoint* usado (limites diferem de *endpoint* para *endpoint*).

X-Rate-Limit-Remaining Número aproximado de pedidos restantes.

X-Rate-Limit-Reset Número aproximado de segundos até restabelecer os valores iniciais.

Também à semelhança do *bot* do Reddit, não é feito qualquer tipo de análise destes campos até que se receba o código de erro HTTP, neste caso o número 429, a indicar que atingimos o limite de pedidos. Novamente, a solução implementada coloca o processo a "dormir"durante o tempo necessário para que seja possível voltar a realizar pedidos.

Paginação
A paginação é realizada de forma semelhante à da API do Reddit. Basta guardar o id do último *tweet* retornado e usá-lo como parâmetro no URL do próximo pedido. O URL ficaria assim: *https://api.twitter.com/...?max_id=<tweet_id - 1>*. Ou seja, é adicionado o parâmetro *max_id* ao URL. Na figura 5.16[70] apresenta-se um exemplo. Neste caso a API iria devolver cinco *tweets* (*count* = 5), em que o id

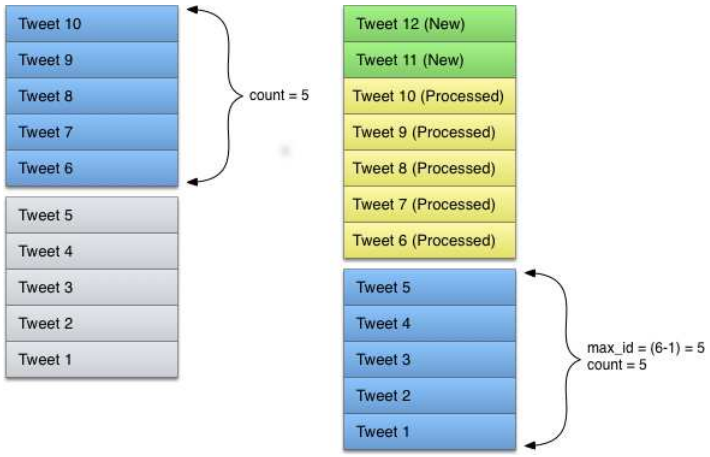

Figura 5.16: Informação sobre a paginação na API REST do Twitter

não poderia ser superior a 5.

5.2.5.3 Streaming API

A *Streaming* API possui dois parâmetros principais para a extração de informação: *track* e *follow*. Abaixo passa-se a explicar a utilidade de cada um:

track Lista de frases separadas por virgulas que será usada para determinar que *Tweets* serão devolvidos na *stream*. Uma frase pode ser um ou mais termos separados por espaços e corresponderá se todos os termos na frase estiverem presentes no *Tweet*. Virgulas é o equivalente lógico de um "Ou" e os espaços são o equivalente lógico do "E"[71]. Exemplos:

- "a dognædis" é o equivalente a "a" **E** "dognædis"
- "a,dognædi" é o equivalente a "a" **Ou** "dognædis"

follow Lista de ids de utilizadores separados por uma vírgula, indicando que *Tweets* de que utilizadores devem ser colocados na *stream*[72].

Estes parâmetros podem ser usados em simultâneo.
Como já foi mencionado, todos os *bots* possuem uma *pool* de processos trabalhadores. A cada *Tweet* recebido é associado um trabalhador da *pool* que vai fazer o processamento. para cada *Tweet* são também preenchidos os *retweets*, os comentários e os gostos, à semelhança da API REST.

Regras de Reconexão
Sempre que existem regras ou informação sobre boas práticas de utilização das APIs, o estagiário procurou segui-las e este caso não é excepção. O Twitter fornece um conjunto de medidas a serem tomadas quando for necessário efetuar reconexão, que caso não sejam seguidas levam a bloqueios de acesso durante determinado tempo. Se a ligação falhar um vez, o *bot* deve tentar reconectar imediatamente. Se esta tentativa falhar deve seguir as seguintes regras:

- Esperar linearmente para erros ao nível do TCP/IP, em que o intervalo entre tentativas deve aumentar 250 milissegundos a cada tentativa, até um máximo de 16 segundos.
- Esperar exponencialmente em caso de códigos de erro HTTP (4XX e 5XX). Começar com intervalo de 5 segundos e duplicar a cada tentativa, até um máximo de 320 segundos.
- Esperar exponencialmente em caso de códigos de erro HTTP número 420 e 429 (Demasiados Pedidos). Começar com 1 minutos de espera e duplicar a cada tentativa. Não tem máximo. Importa referir que por cada pedido que seja efetuado e seja recebido os códigos anteriormente mencionados, o tempo de espera até que o limite seja reiniciado aumenta.

5.2.6 Construção de Grafos

Com o objetivo de se criar outras formas de visualização da informação optou-se pela criação de uma estrutura em forma de grafo. Nesta secção irão ser discutidas as opções de implementação e implementação em si, deixando as questões visuais para a secção 5.3.5.
O Portolan possui uma base de dados principal, na qual toda a informação final (informação que já passou por todos os processos de filtragem e processamento) se encontra. O grafo irá apenas incidir sobre esta base de dados e não sobre toda a informação extraída. Logo é necessário realizar este processo imediatamente na altura da inserção nesta base de dados. Posto isto, a primeira solução foi a utilização de *triggers* ou algo semelhante que existisse no MongoDB. Contudo, não existe nada oficial e todas as soluções desenvolvidas por terceiros são direcionadas para a tecnologia JavaScript.
A solução passou por desenvolver um novo *bot* que fosse responsável pela construção do grafo, que deverá de ser colocado imediatamente antes da *sink* final (figura 5.17) (pode ser colocado antes, mas assim perder-se-ia o objetivo de criar apenas o grafo com a informação final) ou pode-se replicar a informação para a base de dados e para o *node* (figura 5.18). A solução escolhida passou por ir construindo o grafo ativamente ao invés de o apenas construir quando o utilizador fizesse um pedido. Isto permite que o tempo de resposta ao utilizador seja menor, pois o grafo já se encontra construído e pronto a visualizar.

A elipse dupla representa o conjunto de outros *nodes* do *pipeline* que não são relevantes para o que se pretende mostrar. Ambas as representações do *pipeline* são válidas, sendo que na versão sequencial se adiciona mais um ponto onde a informação se pode perder antes de chegar à *sink* final.
Importa ainda mencionar que, no caso de utilização do pipeline 5.18, a replicação de dados não se reflete na base de dados do grafo. Os dados são analisados e apenas a informação necessária à criação e distinção dos nodos é necessária, como o nome,

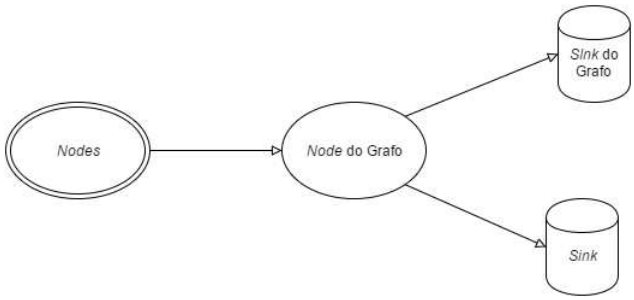

Figura 5.17: Pipeline sequencial com adição do *bot* de construção do grafo

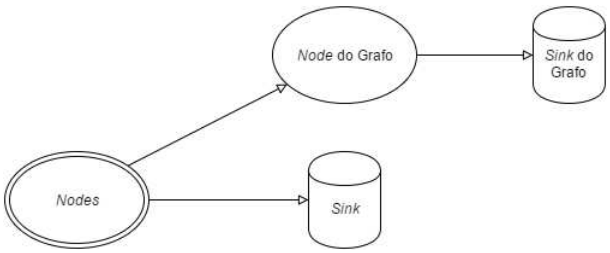

Figura 5.18: Pipeline com replicação da informação para *bot* de construção do grafo e sink final

o nome de utilizador e o id. Acrescenta-se ainda um campo para definir o tipo de nodo.

Criação do Grafo

Para criar a representação visual foi usada a biblioteca de JavaScript **D3js**[73]. Esta biblioteca necessita que a informação que lhe é fornecida esteja no seguinte formato:

```
data = {"nodes":[{name:"Peter",label:"Person",id:1,...},{name:
    ↪ "Michael",label:"Person",id:2,...},{name:"Neo4j",label:
    ↪ "Database",id:3,..},...],"links":[{source:0, target:1,
    ↪ type:"KNOWS", since:2010,...},{source:0, target:2, type
    ↪ :"FOUNDED",...},{source:1, target:2, type:"WORKS_ON"
    ↪ ,...},...]};
```

No contexto do Portolan e da informação extraída das várias fontes sociais, um nodo é tudo o que possua um nome, nome de utilizador ou id e terá um tipo associado (não é obrigatório, mas permite distinguir *posts*, comentários, perfis, entre outros). Alguns exemplos são os *posts*, os perfis e as páginas. As ligações terão sempre um tipo associado (novamente, não é obrigatório, apenas os campos *source* e *target* são necessários para se formar a ligação). Um exemplo seria a ligação entre *post* e perfil, em que esta teria o tipo "autor". Importa referir que os valores dos campos *source* e *target* representam o a posição dos nodos no *array nodes*, ou seja, a primeira ligação seria entre o Peter e o Michael.

Assim, para se criar esta estrutura inicial necessária para a construção do grafo são necessários os seguintes passos:

1. Começar no nó raiz (exemplo: *post*)

2. Extrair id, nome de utilizador e nome
3. Tentar adicionar nó
   (a) Verificar existência do nó
   (b) Se existir devolve o índice do mesmo no *array* de nodos
   (c) Se não existir, criar e devolver o índice do mesmo no *array* de nodos
4. Obter nodos filho
5. Para cada filho
   (a) Tentar adicionar nó
      i. Verificar existência do nó
      ii. Se existir devolve o índice do mesmo no *array* de nodos
      iii. Se não existir, criar e devolver o índice do mesmo no *array* de nodos
   (b) Tentar adicionar ligação entre pai e filho
      i. Verificar existência de ligação
      ii. Se existir, não faz nada
      iii. Se não existir, cria ligação

5.2.7 Bot Extrator de Expressões Regulares

Este *bot* foi desenvolvido para procurar as expressões regulares definidas, nos campos definidos. As cadeias de caracteres não necessitam de estar escritas de acordo com a sintaxe de uma expressão regular, podem ser simples palavras, no entanto serão sempre tratadas como se de uma expressão regular se tratasse.
Este *bot* serve como primeiro meio de filtragem da informação extraída podendo ser definido para apenas encaminhar a informação para o próximo *bot* se encontrar algo. O mesmo apresenta o diagrama de classes representado na figura 5.19. As

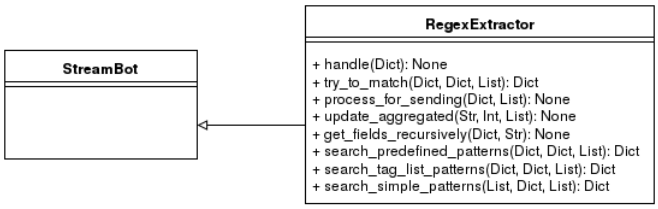

Figura 5.19: Diagrama de classes do Bot Extrator de Expressões Regulares

pesquisas podem passar por expressões regulares predefinidas (IPv4, IPv6, endereços MAC, domínios, entre outros), as quais podem ser alteradas pelo utilizador ou padrões simples (lista de expressões regulares ou simples palavras não escritas no formato de expressão regular). Outro forma será a pesquisa por listas de padrões, que se encontram associados a uma determinada palavra. Um exemplo para este último caso seria:

Segurança Exploit, Firewall, Malware, Virus, Worm, Trojan

O objetivo destes pares **palavra**:**lista de padrões** é classificar a informação segundo a **palavra** definida. Importa ainda referir que a **lista de padrões** representa de uma forma geral os **padrões simples**, sendo que estes últimos não estão associados a nenhuma **palavra**, nem serão usados para classificar automaticamente a informação.
No final de processar a informação recebida, apresentam-se os resultados por expressão regular e caminho-chave. A titulo de exemplo, abaixo apresenta-se a informação recebida por este nó, no formato JSON:

```
{
  ...
  "social":{
    "post":{
      "domain":"self.netsec",
      "author":{
        "username":"Dognaedis",
        "id":"",
        "name":""
      },
      "url":"https://www.reddit.com/...",
      "title":"... Dognaedis Security Thread ..."
    }
  }
  ...
}
```

De seguida apresenta-se um exemplo da configuração deste *bot*:

```
{
  ...
  "target":[
    {
      "selected_search_paths":[
        "social.post.title"
      ],
      "simple_patterns":[
        "Exploit",
        "Firewall",
        "Malware",
        "Virus",
        "Worm",
        "Trojan",
        "Security"
      ],
      "tag_list":{
        "Security":[
          "Exploit",
          "Firewall",
          "Malware",
          "Virus",
          "Worm",
          "Trojan",
          "Security"
        ]}}]...}
```

A representação final para esta situação seria:

```
{
  ...
  "regex_matches":{
    "total_matches":2,
    "social":{
      "post":{
        "title":{
          "total_matches":2,
          "simple_patterns":{
            "Security":1
          },
          "tag_list":{
            "Security":{
              "Security":1
            }
          }
        }
      }
    }
  }
  ...
}
```

Este formato permite visualizar onde foram encontradas as expressões regulares, mas dificulta a criação de *queries* à base de dados que façam agrupamento de campos, pois é difícil prever quais os campos selecionados para a pesquisa. Para o efeito criou-se outro tipo de estrutura que já agrega, apenas para o próprio Documento, todas as expressões regulares e respetivo número de ocorrências. Exemplo:

```
{
  ...
  "aggregated_regex_matches":[
    {
      "name":"Security",
      "count":2
    }
  ]
  ...
}
```

5.2.8 Bot IBM AlchemyLanguage

O AlchemyLanguage[74] é um dos serviços oferecidos pela plataforma da IBM[75], Watson[76]. Representa um conjunto de APIs na *cloud* que oferecem capacidades de análise de texto usando processamento de linguagem natural. Este conjunto de APIs permitem adicionar informação semântica à informação fornecida. As funcionalidades do serviço de análise de texto da API Alchemy são:

- Extração de Entidades
- Análise de Sentimento

- Análise de Emoção
- Extração de Palavras-Chave
- Marcação de Conceitos (*Concept Tagging*)
- Extração de Relações
- Extração de Taxonomias
- Extração do Autor
- Deteção da Língua
- Extração de Texto
- *Parsing* de Micro-formatos
- Deteção de *Feeds*
- Suporte para *Linked Data* (dados ligados entre si)

Esta API recebe como dados de entrada um URL para uma página *web* ou um documento de texto ou HTML. A informação é devolvida em formato JSON e possui um dos seguintes campos: relevância ou pontuação. O utilizador pode definir um limite para filtrar a informação e devolver apenas os resultados com relevância ou pontuação superior a esse limite.
A informação final permitirá, por exemplo, associar uma determinada entidade com um determinado sentimento (positivo, negativo, neutro). A associação entre uma entidade e um sentimento negativo poderá ser um indicativo que a mesma está ou poderá vir a ser um alvo de um qualquer tipo de ataque. Isto ajuda o analista a perceber em quais eventos se deve focar e quais precisam da sua atenção.
Este *bot* possui o diagrama de classes apresentado na figura 5.20.

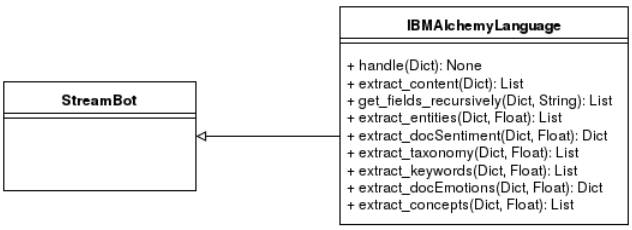

Figura 5.20: Diagrama de Classes do Bot IBM AlchemyLanguage

5.2.9 Pipeline Exemplo

Na figura 5.21 apresenta-se uma das várias representações de um *Pipeline* usando os *bots* desenvolvidos pelo utilizador.

Rectângulo 1 *Bots* do tipo produtor

Rectângulo 2 *Bots* do tipo nodo

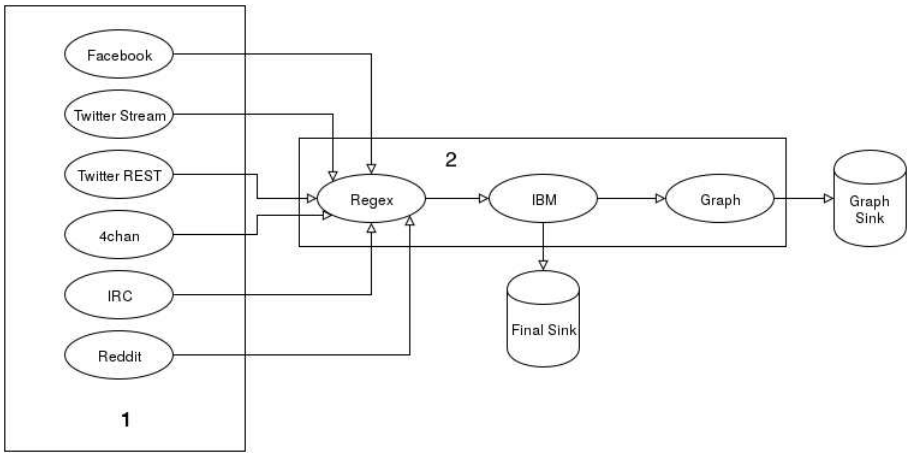

Figura 5.21: Representação exemplo do Pipeline Social

O trabalho dos *bots* do tipo nodo, neste caso, é filtrar a informação recolhida pelos produtores. Nem toda a informação apresenta conteúdo importante e isto ajuda o analista a focar-se apenas nos eventos que precisam da sua atenção. A filtragem é feita com base na deteção de expressões regulares e relevância e pontuação dos resultados devolvidos pela plataforma da IBM.
Importa ainda realçar que este é apenas um exemplo, *bots* podem ser removidos ou modados de posição na *pipeline*, mas tendo sempre em conta o tipo de cada um.

5.3 Interface Gráfica

Para este estágio foram desenvolvidas um conjunto de interfaces gráficas assentes em HTML, CSS e JavaScript e na *framework* Django. As tecnologias mencionadas são atualmente usadas no Portolan, como já foi referido.
As várias componentes desenvolvidas no âmbito da parte gráfica apresentam-se nas próximas secções.
Importa ainda referir que foi necessária a compreensão a fundo de todo o código escrito, pois a interface é a origem da maioria das *queries* de leitura à base de dados. Foi mais uma vez necessário dar suporte a *queries* que envolvessem listas.

5.3.1 Widgets

Um *widget* é um componente de uma interface gráfica de utilizador, como por exemplo botões, menus, gráficos, entre outros. Pode também ser entendido como uma pequena aplicação que flutua pela interface e fornece funcionalidades específicas ao utilizador. No Portolan estes são compostos por gráficos de barras, de linhas e sob a forma de tarte, tabelas e semáforos.
As mudanças fundamentais assentam em dois pontos:

- Passaram a suportar listas (como já tinha sido referido, o modelo de dados anterior não apresentava necessidade de lidar com listas). O MongoDB possui um operador que permite desconstruir uma lista, sendo que é criado um documento novo para cada elemento da lista. Depois de efetuada esta operação

pode-se lidar com o documento da mesma forma que na versão anterior.

- As *queries* funcionam com agregações, em que vários documentos são agrupados segundo determinado campo e depois são aplicadas operações sobre a informação agrupada. A única operação permitida era somar o valor 1 sempre que agrupavam documentos, isto é, permitia contar em quantos documentos determinado campo aparecia. A nova versão permite somar outros valores, sejam eles um simples número, ou o valor de um determinado campo. Por exemplo, se quiser saber o número total de vezes que uma determinada expressão regular casou é necessário usar o valor que ficou guardado na base de dados e não somar o valor 1, porque aí apenas ir-se-ia saber em quantos documentos diferentes é que determinada expressão aparece. Além das operações de soma foi também adicionada o cálculo de médias, que pode se usada por exemplo para calcular o valor médio de vezes que uma expressão regular casou.

Na figura 5.22 apresenta-se um exemplo de um *widget*, sob a forma de um gráfico de barras. A mesma divide-se em duas partes, em que a primeira representa os parâmetros de configuração e a segunda representa o resultado final. Este exemplo em particular apresenta as 5 expressões regulares mais correspondidas. Além do

Figura 5.22: Widget Exemplo

gráfico de barras podem ser escolhidos outros tipos de gráficos ou tabelas como já foi mencionado. A parte visual destes últimos não será aqui apresentada, pois apenas a parte de seleção da estatística desejada foi implementada pelo estagiário e que já encontra representada na figura 5.22.

5.3.2 Tabelas

As tabelas usadas no *Portolan* eram estáticas, isto é, não era possível escolher quais as colunas que se queria usar, pois este focava-se especialmente em evento com informação de rede e as tabelas apenas suportavam este domínio. Na nova versão o utilizador pode escolher quais colunas pretende visualizar. O leque de colunas possíveis é constituído por todos os campos presentes no modelo de dados (ver anexo C.4.2), exceto os campos com tipo *List*. Esta escolha foi de encontro

com um requisito não funcional mencionado, a Manutenência, pois este mecanismo dinâmico permite que o utilizador escolha os dados que quer visualizar na tabela tendo já em mente possíveis adições de novas fontes ou novos tipo de informação. A implementação foi realizada usando a biblioteca **bootstrap-select**[77]. A escolha de colunas possui ainda as seguintes funcionalidades:

- Selecionar todas
- Desselecionar todas
- Pesquisa baseada em palavras-chave associadas a cada campo

Na figura 5.23 apresenta-se um exemplo da escolha de colunas. Depois de apresen-

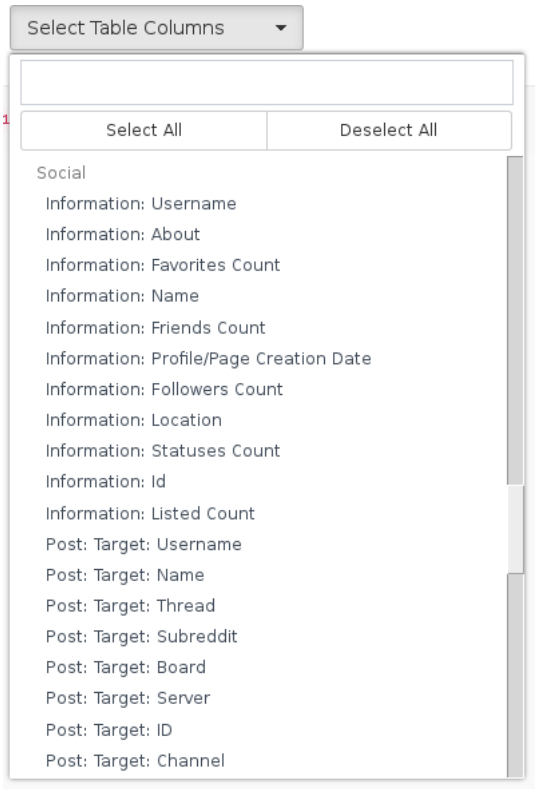

Figura 5.23: Exemplo da escolha de colunas

tada a tabela e se o utilizador escolher determinadas colunas, pode visualizar em detalhe a informação. Dependendo da coluna é possível visualizar a informação em JSON e tabela. No caso de um *post* a informação pode ainda ser apresentada de forma semelhante à usada nas fontes. Novamente, foi dado suporte a dados do tipo *Lista*. Um exemplo para o último caso (*post*) apresenta-se na figura 5.24. Mais exemplos serão apresentados no anexo C.1, incluindo os resultados da tabela e as visualizações em detalhe de certos elementos.

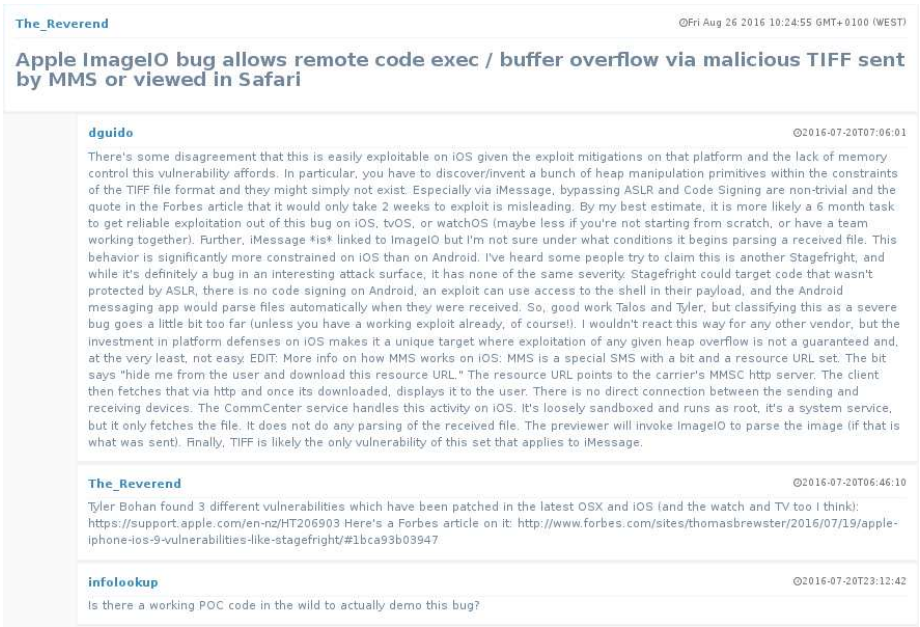

Figura 5.24: Apresentação de um post de forma estruturada

5.3.3 Matriz

A matriz é outra das formas de explorar a base de dados. Permite definir quais os campos que queremos para as linhas e para as colunas, mostrando todos os pares XY de informação para os campos definidos. É possível mostrar, por exemplo, quantas vezes é que cada expressão regular ocorreu em cada fonte ou por cada pessoa (figura 5.25).
À semelhança dos casos anteriores também foi necessário alterar o código de acesso à base de dados para passar a suportar listas.

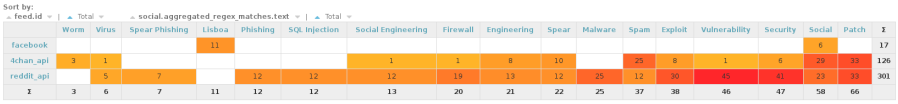

Figura 5.25: Apresentação da interface de exploração da base de dados usando uma matriz

5.3.4 Classificação Manual de Eventos

Com o intuito de se poder analisar e classificar resultados que necessitem de ser classificados ou que a sua classificação seja validada foi criada uma interface que permita a classificação manual dos vários eventos.
Esta interface permite implementar um mecanismo com características *Human-in-the-loop*[78], isto é, um mecanismo que requer interação humana e permite ao

utilizador mudar o resultado de um evento ou processo. Esta interface, juntamente com este mecanismo poderá no futuro ser integrada com um modelo de *machine learning*[79] em que primeiro o modelo é usado para etiquetar os dados fornecidos, devolvendo o resultado com um nível de confiança associado. Se esta confiança estiver abaixo de um determinado nível definido envia os dados para serem julgados por um humano. Este novo julgamento é posteriormente utilizado para melhorar o modelo de *machine learning*. Como este ponto não ia de encontro ao objetivo do presente estágio, não foi discutido e avaliado detalhadamente.
Um exemplo desta interface relativo à escolha e submissão da classificação do evento encontra-se na figura 5.26. Mais exemplos encontram-se no anexo C.2, incluindo a

Figura 5.26: Interface de escolha de classificação de um evento

escolha das bases de dados de fonte e de destino e os vários tipos de visualização dos dados.

5.3.5 Visualização do Grafo

A informação extraída das várias fontes, dada a sua natureza (mecanismos de amizade e gostos, por exemplo), potencia a sua representação sobre a forma de grafo com o objetivo de analisar as ligações que possam existir. Isto é algo que já é feito pelas próprias empresas como meio de ajuda às várias áreas de negócio.
Foi por isso decidido que seria importante a criação de um gráfico que representasse as ligações entre a informação extraída e que possibilitasse a sua análise.
Esta interface faz uso da estrutura criada pelo *bot* mencionado em 5.2.6 e usa a biblioteca **D3js**. Como pode verificar pela lista de requisitos o grafo é gerado a partir de um nodo selecionado, com uma profundidade definida e apenas com informação adicionada nas últimas horas. Isto permite limitar o número de nodos a mostrar, pois caso o grafo fosse mostrado todo de uma só vez facilmente se degradaria a experiência do utilizador. Isto permite não apresentar o grafo todo de uma só vez, caso contrário facilmente se iria degradar a experiência do utilizador e usabilidade e consequentemente dificultaria a análise do grafo.
É ainda possível pesquisar os vários nodos, sendo que é disponibilizada um lista de todos os nodos existentes, que vai sendo filtrada à medida que se escreve. Isto permite ao utilizador saber, à partida, se um nodo existe ou não.
Para ajudar à visualização, em particular quando o grafo possui muitos nós e li-

gações, é ainda possível realçar a parte do grafo que se encontra selecionada, ao mesmo tempo que é mostrada informação sobre o nodo ou a ligação.
Por fim é ainda possível expandir o grafo a partir do nodo selecionado, mantendo ou removendo os restantes nodos, mediante a opção escolhida pelo utilizador.
Um exemplo encontra-se na figura 5.27. Mais exemplos serão apresentados no

Figura 5.27: Interface de visualização do grafo

anexo C.3.

Especificação da Implementação
Como já foi referido a criação do grafo foi realizada usando a biblioteca **D3js**, em particular o módulo ***d3-force***.
A implementação começa com a definição das dimensões da área de visualização, o que, neste caso, correspondem às dimensões da *tag* sobre o qual o conteúdo iria ser inserido.
Depois é necessário definir a informação para a criação do grafo. O *Force Layout* requer a existência de duas listas com informação. Uma primeira chamada *nodes* contém o objetos que serão usado para criar os pontos focais da visualização. A segunda lista chama-se *links* identifica todas as ligações entre os *nodes*. Estas duas listas foram geradas pelo *bot* mencionado em 5.2.6. Relembra-se que a lista *links* possui dois campos, *source* e *target*, sendo que estes representam a posição do nodo na lista *nodes*. Contudo assim que começar a ser criada a estrutura do grafo estes serão substituídos por uma referência para os objetos em vez do índices.
Tendo os dados bem definidos começa-se por se criar uma *tag* SVG, que é o que vai conter toda a parte de visualização e especifica-se as dimensões definidas previamente.
Posteriormente cria-se um objeto *Force Layout* e define-se as suas propriedades: dimensões, nodos e ligações.
De seguida adicionam-se os nodos e as ligações à visualização (dentro da *tag* SVG

previamente criada). É necessário colocar as ligações primeiro e depois os nodos, para que estes apareçam "por cima". O CSS permite definir uma propriedade de um elemento que indica qual a sua posição da pilha de visualização: *z-index*. Um elemento com um índice superior fica sempre "por cima" de um com índice mais baixo. No caso do SVG isto é feito pela ordem de adição: os últimos elementos a serem adicionados ficam sempre "por cima". Uma ligação é representada pela *tag line*. Já um nodo é representado pela *tag circle*.
São ainda adicionados os eventos do rato e respetivas funções para manipulação desses eventos, com o objetivo de implementar as funcionalidades mencionadas no inicio da presente subsecção. Por fim inicia-se o *Force Layout* e o grafo tornar-se-á visível.

5.3.6 Configuração de bots

A configuração engloba adicionar, editar e eliminar *bots*. Além dos campos comuns a todos os *bots* existem outros específicos a cada *bot*, com nuances específicas que tiveram de ser implementadas pelo estagiário. A criação da interface de configuração de cada *bot* é feita através de um *template* em formato JSON que cada *bot* possui.
Depois de criado, o *bot* é adicionado ao *Pipeline*. O mesmo pode ser arrastado, reposicionado e ligado a outros *nodes* ou *sinks* já existentes. Ao fazer duplo clique no nodo é aberta a janela de edição, na qual o utilizador pode fazer as alterações que pretender e guardar a nova configuração ou eliminá-lo.
Os *templates* e respetivas interfaces encontram-se no anexo C.4.

Especificação da Implementação
Como foi mencionado anteriormente as interfaces são geradas a a partir de um *template* em formato JSON.
A implementação inicial construia as interfaces da seguinte forma:

- Para cada chave do *template* JSON era criado um campo de entrada de texto com um rótulo associado ao mesmo. Esse rótulo possuía o mesmo nome da chave.
- O valor associado à chave era usado para preencher o campo de entrada de texto.

Como podemos verificar é uma forma muito simples e conveniente de gerar novas interfaces. Contudo encontra-se limitado a este tipo de estrutura.
os *bots* a implementar necessitavam de outras formas de representação para que fosse possível apresentar listas e escolhas binárias. Para o efeito a forma como as interfaces eram geradas teve de sofrer alterações. Manteve-se a base inicial, no entanto mediante chaves específicas usou-se outros tipos de elementos, além das já usadas caixas de texto. As chaves a reter são:

target, api e login Mediante a existência destas chaves são gerados *Panels*[80]. O número de *Panels* depende do valor associado à chave. Se o valor for uma lista de elementos será criado um *Panel* por cada elemento. Adicionalmente será adicionado um botão de "Adicionar Mais". Se não for uma lista será apenas criado um *Panel*. Esses *Panels* serão depois preenchidos com a informação do elemento.

pagination Apenas é mudado o tipo de dados da caixa de entrada de dados para *Número* e é adicionado um valor mínimo permitido de 1.

fields Campos de escolha binária em que se usam *checkboxes*[81].

page, friendship e convert_username_to_id Para este caso são usadas listas de seleção[81] com Verdadeiro ou Falso.

search_paths Exclusivo do *bot* extrator de expressões regulares e é onde é feita a escolha dos campos onde vão ser pesquisados as expressões regulares. os caminhos de procura (em inglês, *search paths*) englobam todo a parte do modelo de dados sob a chave *social*. A implementação consistiu da criação de um estrutura em forma de arvre com caixas de seleção que vão expandindo ou colapsando os nós da árvore. Foi implementado usando a biblioteca **bootstrap-treeview**[82].

Capítulo 6

Validação dos Requisitos

Neste capítulo será descrita a validação do projeto desenvolvido. Fase muito importante pois permite verificar se a solução vai de encontro aos requisitos especificados.

6.1 Validação dos Requisitos Funcionais

Os testes aos requisitos funcionais foram elaborados ao longo do desenvolvimento, como foi especificado em 2.4. Estes testes serviram para apurar se as funcionalidades implementadas se encontravam a funcionar corretamente. Em casos em que uma funcionalidade implicava alteração de outras já implementadas e aceites, estas últimas e todas as suas dependentes foram novamente testadas para garantir que não tinham sido comprometidas.
Para verificar quais os requisitos que foram de facto desenvolvidos e destes quais passaram com sucesso nos testes de aceitação e de sistema apresenta-se no anexo A.3 uma lista que pretende mostrar o sucesso de implementação dos mesmos. Estes foram os únicos tipos de teste realizados nos *bots* dada a dificuldade em criar casos de teste diversos e próximos dos casos reais.
A lista completa dos testes encontra-se no anexo A.4. Como se pode verificar nem todos os requisitos que foram implementados passaram nas diferentes fases de validação e foram apresentadas soluções para alguns deles:

RF-1aiG e RF-1biAVII Não foi possível extrair as partilhas de um *post* no Facebook porque a versão *mobile* não permite a sua visualização.

RF-1ai Extrair Informação pela API do Facebook O primeiro teste com vista à validação de todo o *bot* (desde extração, ao processamento e terminando na base de dados) falhou. Isto porque a informação extraída engloba listas de informação, nomeadamente ao nível dos comentários, gosto, entre outros. O método responsável pela validação dos dados extraídos tendo em conta o modelo de dados definido não se encontrava preparado para tratar e validar listas. Dada a natureza crítica da validação dos dados extraídos, pois afeta todos os *bots*, foi alterada de imediato.

RF-1bi Extrair informação do Facebook via crawling Este teste permitiu avaliar se e a extração de várias páginas de informação estava a ser feita de forma correta. O que não foi o caso porque a estrutura do HTML era diferente para

a primeira e restantes páginas. Isto levou a que o XPath usado para extrair informação fosse diferente. O problema foi solucionado de imediato.

RF-1bivB Extrair informação sobre utilizador Teste realizado ao *bot* IRC com o intuito de validar a execução do comando *whois*. Este comando quando enviado ao servidor IRC faz com que este retorne informação sobre o utilizador pedido. Este teste falhou porque para receber a resposta ao comando era necessário definir um método de retorno. O problema foi resolvido de imediato.

RF-4aiiB Editar Alvos de um Bot Quando validado pela primeira vez, apenas era possível definir um alvo para cada *bot*. Foi decidido que um *bot* deveria poder ser configurado com vários alvos. Dada a importância deste requisito funcional o mesmo foi prontamente alterado.

RF-4ci Representação do grafo Apesar das várias funcionalidades que compõem este requisito terem passado no teste de aceitação, ficou claro que, no âmbito geral, o grafo não possuía um conjunto de funcionalidades essenciais. Essas funcionalidades foram colocadas no fim da lista de prioridades e caso houvesse tempo seriam implementadas. Algumas dessas funcionalidades (ou correções) seriam:

1. Não apresentar todo o grafo de uma só vez, pois facilmente degrada o desempenho da aplicação e dificulta a análise e interação com o mesmo.
2. Permitir a definição de um intervalo temporal para a criação do grafo.
3. Permitir a definição da profundidade do grafo a criar.

RF-4cii Widgets Este teste serviu para validar se os resultados apresentados, de acordo com os parâmetros de configuração, se encontravam corretos. O teste falhou porque as *queries* não suportavam listas de informação. Problema foi prontamente solucionado. Importa ainda referir que, apesar de não documentado, este teste foi aplicado também nas interfaces de visualização em tabela e em matriz. Ambas as interfaces não se encontravam preparadas para lidar com listas e tiveram de ser prontamente alteradas.

6.2 Validação Requisitos Não Funcionais

Na secção 4.2 foram mencionados os requisitos não funcionais. Além dos requisitos funcionais também foi do interesse do estagiário validar os requisitos não funcionais. Nas secções seguintes explicar-se-á como foram validados.

6.2.1 Manutenência

No capítulo da análise de riscos foi assumido que as fontes de informação se apresentavam como um risco a ser considerado. E assumindo-se estas com a base da solução a implementar faz sentido que todo o código relativo às mesmas seja de fácil alteração. Para tal todo o código foi escrito de forma o mais genérica possível. Os vários métodos foram mantidos curtos e por vezes agrupados em classes. Além de permitir que as modificações sejam mais fáceis, facilita também a análise, compreensão e teste do código.

A juntar ao mencionado importa ainda referir que os vários *bots* apresentam várias mensagens de *log* com diferentes níveis (sendo maioritariamente utilizados os níveis de erro, informação e *debug*).
Por fim cada tipo da acesso encontra-se no seu módulo. Por exemplo, o Portolan possui três métodos de acesso diferentes: acesso a um URL, à API do Twitter e à API do Reddit. Ao separar em módulos cada um dos acessos permite-se que eventuais alterações sejam mais fáceis de realizar, pois apenas se torna necessário alterar o código no módulo respetivo e não nos *bots* que utilizam esse tipo de acesso.

6.2.2 Usabilidade

Como foi mencionado em 4.2.4 foram seguidas algumas linhas de guia para garantir a usabilidade da interface gráfica desenvolvida. Nesta secção apresentam-se os aspetos que foram de encontro às linhas de guia e porque não foi possível validar este requisito não funcional.
Para ajudar os utilizadores são bloqueados os campos que não necessitam de ser preenchidos. Sempre que algum dos campos necessários não é preenchido o utilizador é avisado de tal facto. Nesta situação é mostrada uma mensagem a vermelho ao lado do campo respetivo e este último é circundado também a vermelho. Além das mensagens de erro a vermelho também são mostradas mensagens de sucesso a verde.
A juntar ao mencionado alguns campos têm associados alguns avisos (ativar determinada funcionalidade pode afetar significativamente o desempenho). Outros apresentam informação de ajuda sobre o que aqueles campos representam e como devem ser preenchidos.
Outro ponto a ter em conta foi o facto dos dados de entrada introduzidos pelo utilizador provocarem erros no sistema, sendo necessária a validação dos mesmos.
A juntar às já enumeradas, e em particular na interface do grafo, junta-se o facto de na caixa de procura de um nodo ser apresentada uma lista com todos os nodos disponíveis que é filtrada à medida que os utilizadores escrevem, para fornecer ao utilizador uma forma de perceber quais os nodos existentes. Além do mencionado são dadas formas de interação bem conhecidas ao utilizador, como agarrar e arrastar, aproximar e afastar usando a roda do rato. É ainda apresentada uma legenda sobre os tipos de nodos do grafo e mostrada informação sobre mesmos, bem como sobre as ligações.
A validação da usabilidade de uma solução é feita através da experimentação da aplicação por um conjunto de utilizadores. Esse grupo de utilizadores deverão fazer parte do público-alvo da aplicação e desconhecer a aplicação, ao nível do *design* e da implementação. O único conjunto de utilizadores disponíveis seriam os funcionários da Dognædis. Sendo a Dognædis um empresa pequena os elementos da parte técnica estão bastante envolvidos em todos os projetos, como o caso o Portolan e dos desenvolvimentos que foram realizados pelo estagiário. Assim sendo restam apenas os elementos dos serviços de apoio, nomeadamente os departamentos financeiros e de recursos humanos, sendo que estes não são o público-alvo. Ou seja, não pôde ser realizada a validação deste requisito. Contudo, foram tomadas as medidas já mencionadas para tentar colmatar este problema.

6.2.3 Segurança

A injeção NoSQL pode ser executada nas várias camadas da aplicação. Tipicamente será onde a cadeia de caracteres inserida pelo utilizador da aplicação numa caixa de texto é concatenada com outra na chamada à API da base de dados. Para assegurar a proteção da aplicação contra este tipo de ataques, antes de ser realizada a concatenação é necessário verificar se a cadeia de caracteres possui caracteres especiais, sendo estes os seguintes:

- '
- "
- \
- ;
- {
- }

A titulo de exemplo, atente na seguinte *query* à base de dados MongoDB, em que campos entre <> correspondem aos dados de entrada colocados pelo utilizador, sem qualquer validação:

```
db.contas_utilizador.find({nome_utilizador: <nome_utilizador>,
    ↪  palavra_passe: <palavra_passe>});
```

Imagine agora que os dados de entrada colocados pelo utilizador, em formato JSON, seriam:

```
{
    "nome_utilizador": {$gt: ""},
    "palavra_passe": {$gt: ""}
}
```

Isto resultaria na seguinte *query*:

```
db.contas_utilizador.find({nome_utilizador:  {$gt: ""},
    ↪ palavra_passe:  {$gt: ""}});
```

No *MongoDB $gt* escolhe os documentos cujo campo escolhido (neste caso nome de utilizador e palavra passe) sejam maiores que o valor fornecido (neste caso uma cadeia de caracteres vazia). Ou seja, a *query* devolve sempre *Verdadeiro*, o que faz com que qualquer pessoa consiga fazer *login*, mesmo sem usar qualquer tipo de credenciais. Importa referir que os exemplos mostrados não vão de encontro à real implementação do Portolan. As contas de utilizador não são guardadas no MongoDB, mas sim numa base de dados relacional cujo acesso e validação de injeções SQL são tratados pelo ORM do Django.
Fazendo a validação mencionada permite detetar este tipo de dados de entrada e consequentemente torna uma aplicação segura contra ataques de injeção NoSQL.
Além do mencionado devem ser também validados os dados de entrada contra o conteúdo esperado, sendo este representado pelo modelo de dados da aplicação.
De salientar que o único momento em que esta validação é feita é quando se classifica um evento manualmente, pois só aqui são introduzidos dados no MongoDB provenientes de um utilizador. Toda a informação está disponível a todos os

momentos para todos os utilizadores ligados e tendo em conta que não é possível acessos não autorizados (a menos que consigam obter as credenciais de outra pessoa ilegalmente) a probabilidade de existência de injeções NoSQL é remota.

6.2.4 Desempenho

O desempenho foi outro dos requisitos não funcionais mencionados e está presente tanto na parte gráfica da aplicação como nos *bots*.

6.2.4.1 Interface Gráfica de Utilizador

Para assegurar um bom desempenho nesta componente da aplicação foram tidos em contra os seguintes pontos:

Minimizar pedidos HTTP Um das formas de melhorar o desempenho é diminuindo o número de pedidos HTTP. Para o efeito colocou-se todo o CSS e JavaScript num único ficheiro (cada um no seu).

Usar versão reduzida do CSS, HTML e JavaScript Tornar código na sua versão reduzida implica remover caracteres desnecessários, como espaços em branco, comentários e caracteres de mudança de linha. Como se se escreve-se o código numa única linha. Isto permite reduzir o tamanho dos ficheiros, o que , consequentemente diminui o tempo de descarga, logo a página carrega mais rápido.

Colocar CSS no topo da página Ajuda a que a página pareça carregar mais rápido, pois a página vai renderizando progressivamente.

Colocar JSS no fim da página Para mostrar a página ao utilizador todos os ficheiros tem de ser carregados antes. Colocar o JavaScript no final permite que a página seja mostrada antes deste ser carregado.

JavaScript e CSS como ficheiros externos Torna a aplicação mais ápida porque geralmente ficam na *cache* do *browser* ao invés se ser carregados em cada novo acesso, o que acontece se o código estiver embutido no HTML.

Evitar duplicados de JavaScript e CSS Mais ficheiros, mais tempo para carregar a página. Neste caso, o desempenho é degradado sem qualquer utilidade.

Existem muitas mais regras a ter em contra, no entanto encontram-se fora do âmbito de trabalho do estagiário, pelo que não foram mencionadas.

6.2.4.2 Bots

A implementação usada nos *bots* do Portolan permite que a solução trabalhe com diferentes configurações internas. É, assim, imperativo analisar quais as configurações que permitem as melhores condições de funcionamento. Os testes executados e apresentados neste secção servirão também para validar a escalabilidade.
Para a realização dos testes foi usado um ambiente fixo com a seguinte configuração:

Sistema Operativo Debian GNU/Linux 8 (Jessie) 64-bit

Processador Intel Core i3-2100 CPU @ 3.10GHz x 4

Memória 3.8 GiB

Os testes foram feitos para cada *bot* em separado. Como já foi mencionado os mesmos podem possuir várias configurações internas diferentes e é importante verificar quais as que permitem um melhor funcionamento dos mesmos.
Para testar cada um dos *bots* as configurações variadas foram:

- Número de processos
- Número de informação a processar

Cada teste é executado trinta vezes e no final são calculadas as médias e desvios padrão dos tempos de execução e da memória utilizada.
Importa ainda referir que os testes foram realizados excluindo a fase de extração e apenas incidiu na fase exatamente seguinte: o processamento da informação para a enviar para o nodo seguinte. Esta foi a única maneira possível de fornecer grandes quantidades de informação aos *bots* , pois os limites de pedidos que as várias APIs possuem não o permitiram.

***Bot* Facebook**
Para testar o *bot* do Facebook foi-lhe fornecido um conjunto de eventos cuja média de tamanho foi cerca de 25 kB. De seguida foram testadas as seguintes variações das configurações:

Número de Processos 1, 2, 3, 4, 5, 10, 100

Número de Eventos 200000, 500000, 1000000

Os testes começaram pelo seguinte número de processos, pela ordem apresentada: 1, 5, 10. Entre 5 e 10 processos não houve uma variação significativa no tempo de execução e por esse motivo o estagiário decidiu aumentar novamente, mas desta vez para um valor muito superior. Este aumento teve o propósito de averiguar se a mudança de 5 para 10 processos não teria sido suficiente ou se se tinha atingido um ponto onde o aumento do número de processos deixa de melhorar o desempenho e até pode ter o efeito contrário, devido a máquina onde se realizam não possuir poder de processamento suficiente. Novamente, não houve alterações significativas no tempo de execução, o que leva a concluir que a segunda opção, das duas anteriormente mencionadas, se apresenta como correta.
Entre 1 e 5 processos houve uma melhoria temporal, mudança essa que, como já foi referido, não se verificou para números de processos superiores. Ou seja, o valor (ou os valores) ótimo para o número de processos estaria entre este intervalo e foi isso que levou à realização dos restante testes.
A alteração do número de eventos fornecidos permitiu avaliar a variação do desempenho com o aumento de tarefas a realizar e consequentemente permitiu também avaliar a escalabilidade.
Na tabela 6.1 apresentam-se os resultados (a primeira linha representa a variação do número de processos, enquanto a primeira coluna representa o número de eventos), acompanhados da figura 6.1.

Pela análise da figura 6.1 podemos verificar que os tempos médios mais baixos são os obtidos nos testes com três processos. Como esperado, usando apenas um processo obtemos piores resultados comparativamente aos testes multi processo. É possível ainda verificar que a partir de quatro processos o desempenho vai ligeiramente piorando.

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	83,343	63,567	63,179	65,949	66,621	68,093	68,845	68,514	6,871
500000	204,026	158,091	155,907	164,486	164,521	170,75	171,864	169,949	16,138
1000000	412,044	314,253	313,008	326,954	330,962	345,07	341,109	340,486	33,811

Tabela 6.1: Tabela com os tempos de execução, em segundos. A primeira linha representa a variação do número de processos, enquanto que a primeira coluna representa o número de eventos.

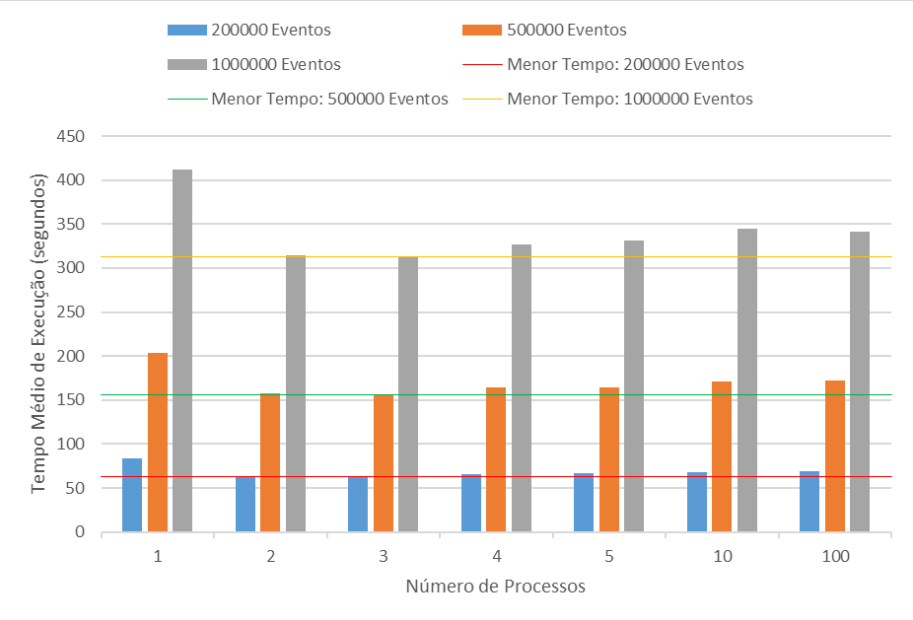

Figura 6.1: Média dos tempos de execução, em segundos, dos vários testes realizados

Usando como base de comparação o número mais baixo de eventos fornecidos (200000), apresentam-se na tabela 6.2 os valores dos tempos médios comparativamente ao valor base.

Analisando a tabela 6.2 verifica-se que os aumentos de 150% e 400% no número

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000 - 100%	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000 - 250%	245%	249%	247%	249%	247%	251%	250%	248%	2%
1000000 - 500%	494%	494%	495%	496%	497%	507%	495%	497%	4%

Tabela 6.2: Comparação percentual entre tempos de execução assumidos como base e restantes

de eventos a processar resultaram também nas mesmas proporções nas suas médias de tempo de execução. Um aumento de 150% no número de eventos levou a um aumento médio de 148% do tempo de execução. Já um aumento de 400% no número de eventos levou a um aumento médio de 397% do tempo de execução.

Além dos tempos de execução também foi analisada a memória utilizada. Os resul-

tados apresentam-se na tabela 6.3 e figura 6.2.
Pela análise da tabela 6.3 e da figura 6.2 verifica-se que, como esperado, os valores

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	88,111	91,41	93,217	102,61	92,825	91,399	91,597	93,024	4,535
500000	125,037	138,455	138,78	161,985	139,132	132,705	135,93	138,861	11,347
1000000	149,11	176,164	181,051	223,064	161,261	169,279	175,006	176,419	23,187

Tabela 6.3: Média da memória utilizada por teste

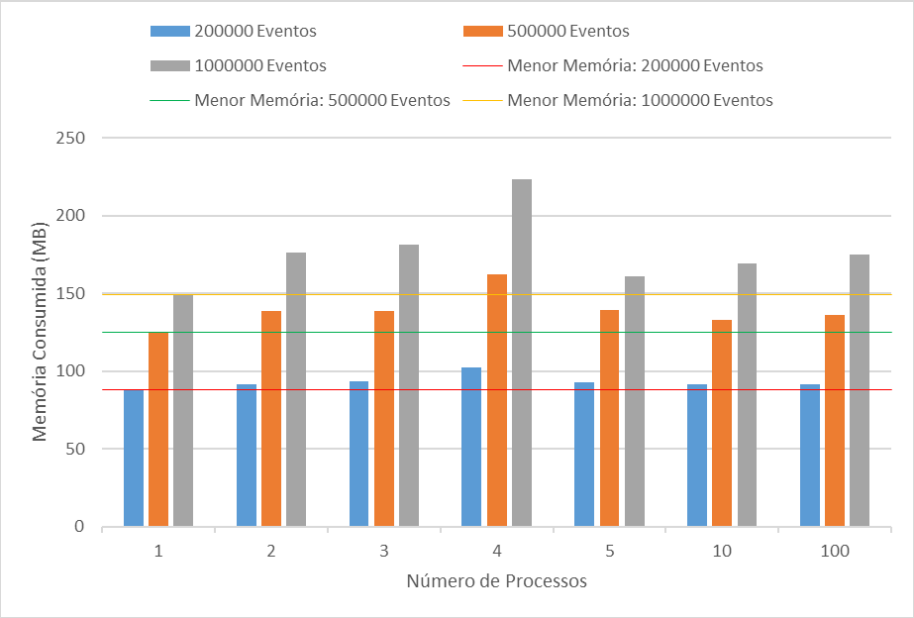

Figura 6.2: Média da memória utilizada por teste

para apenas um processo são inferiores aos casos que foram usados mais processos, pois mais processos implicaram mais objetos em memória. Podemos ainda verificar que os testes usando quatro processos são os que apresentam o maior consumo de memória, contudo não foram os que apresentaram melhor desempenho, como já se verificou.
À semelhança dos tempo médios de execução, também para as médias de memória ocupada se vai apresentar as comparações percentuais entre valores, considerando os valores base aqueles que foram obtidos com os testes usando 200000 eventos. Os resultados encontram-se na tabela 6.4.

A tabela 6.4 permite-nos verificar que os aumentos de memória não foram na mesma proporção que o número de eventos fornecidos, pois nestes testes os processos são *CPU-bound* (em português, Ligado à CPU, *Central Processing Unit*). Um processo é *CPU-bound* quando a sua velocidade de execução depende principalmente da velocidade da CPU.
Estes testes vêm, assim, validar o desempenho do *bot*. Permitem ainda que se afirme que o *bot* é escalável, pois o tempo de execução aumentou linearmente e nas mesmas proporções que o aumento de trabalho.

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000 - 100%	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000 - 250%	142%	151%	149%	158%	150%	145%	148%	149%	5%
1000000 - 500%	169%	193%	194%	217%	174%	185%	191%	189%	16%

Tabela 6.4: Comparação percentual entre valores de memória consumida assumidos como base e restantes

Bot Extrator de Expressões Regulares
Este *bot* é *bot* responsável pela procura de expressões regulares na informação extraída. Para testar este *bot* foi-lhe fornecido um conjunto de eventos cuja média de tamanho foi cerca de 250 kB. De seguida foram testadas as seguintes variações das configurações:

Número de Processos 1, 2, 3, 4, 5, 10, 100

Número de Eventos 200000, 500000, 1000000

Os testes seguiram a ordem apresentada acima. Verificou-se um aumento do desempenho entre 1 e 3 processos. O contrário aconteceu entre os 3 e os 100 processos. Ou seja, o *bot* apresentou o melhor desempenho com 3 processos, à semelhança do que havia sido apurado na validação de desempenho do *bot* do Facebook. Os resultados pode ser visualizados na tabela 6.3 e figura 6.1. Como esperado os testes com apenas um processo possuem o desempenho mais baixo. A juntar ao

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	85,011	55,097	49,529	51,430	51,917	52,491	53,797	57,039	11,535
500000	208,261	143,159	124,223	129,012	130,244	131,848	133,650	142,914	27,208
1000000	412,928	287,937	248,175	260,345	261,791	265,840	276,008	287,575	52,479

Tabela 6.5: Tabela com os tempos de execução, em segundos. A primeira linha representa a variação do número de processos, enquanto que a primeira coluna representa o número de eventos

mencionado e à semelhança dos testes anteriores apresentam-se os aumentos em termos percentuais relativamente aos valores apresentados pelos testes com 200000 eventos. Essa informação apresenta-se na tabela 6.6. Como podemos verificar os valores do tempo de execução aumentaram na mesma proporção que o número de eventos submetidos o que permite validar a escalabilidade e desempenho deste *bot*.
Além dos tempos de execução também se extraiu o consumo de memória nos

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000	245%	260%	251%	251%	251%	251%	248%	251%	4%
1000000	486%	523%	501%	506%	504%	506%	513%	506%	11%

Tabela 6.6: Comparação percentual entre tempos de execução assumidos como base e restantes

vários testes. Os resultados apresentam-se nas tabelas 6.7 e 6.8 e na figura 6.4. Como podemos verificar a memória aumenta linearmente com o número de eventos

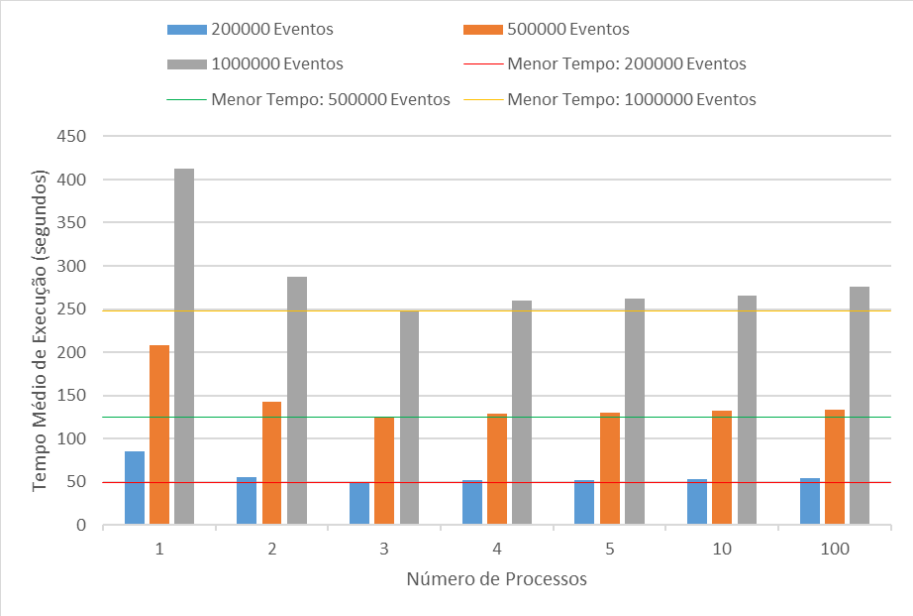

Figura 6.3: Média dos tempos de execução, em segundos

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	111,142	111,299	109,433	110,568	114,865	116,108	121,756	113,596	4,005
500000	155,506	161,925	159,748	155,850	154,725	162,386	183,637	161,968	9,307
1000000	217,494	213,685	218,294	206,490	197,335	212,181	257,064	217,506	17,471

Tabela 6.7: Média da memória utilizada por teste

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000	140%	145%	146%	141%	135%	140%	151%	143%	5%
1000000	196%	192%	199%	187%	172%	183%	211%	191%	13%

Tabela 6.8: Comparação percentual entre valores de memória consumida assumidos como base e restantes

submetidos, mas em proporções mais baixas que o aumentos destes últimos, sendo superior nos casos em que são usados mais processos.

Bot IBM AlchemyLanguage

Este *bot* é o responsável por aceder à API do *IBM AlchemyLanguage*. Para testar este *bot* foi-lhe fornecido um ficheiro com cerca de 200 kB. As configurações testadas form as mesmas que nos *bots* anteriores:

Número de Processos 1, 2, 3, 4, 5, 10, 100

Número de Eventos 200000, 500000, 1000000

Os resultados dos tempos médios de execução apresentam nas tabelas 6.9 e 6.10 e na figura 6.5. Os tempos assumidos como base foram os que resultaram dos testes

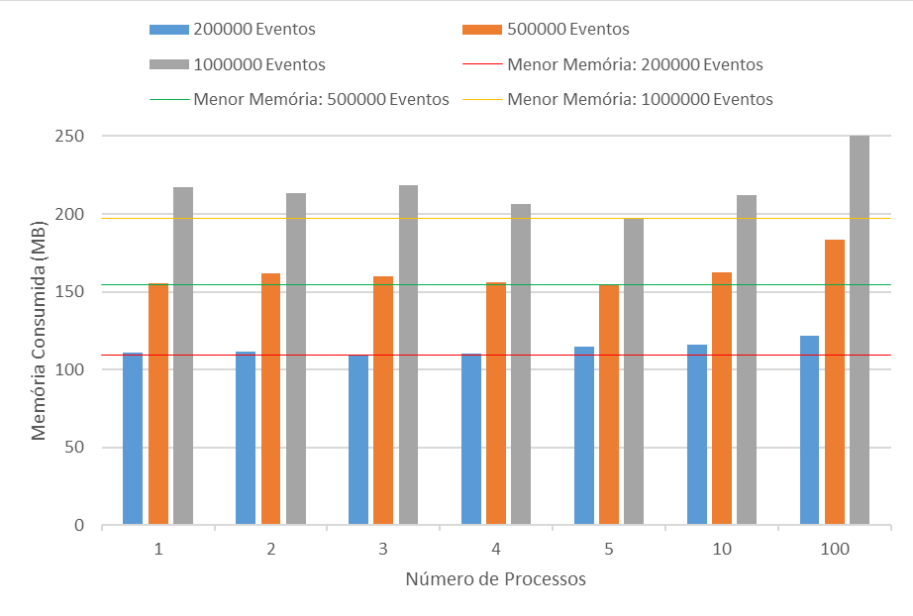

Figura 6.4: Média da memória utilizada por teste

com 200000 eventos. Verifica-se novamente que tempos de execução mais baixo quando se usam 3 processos e que os mesmos acompanham proporcionalmente os aumentos no número de eventos. Considera-se, assim, validados o desempenho e escalabilidade do *bot*.

Como nos *bots* anteriores também será mostrada a memória consumida. Os re-

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	79,948	58,162	49,796	51,793	52,540	52,978	53,964	57,026	9,653
500000	205,683	139,848	124,577	130,257	131,626	132,369	137,561	143,132	25,947
1000000	400,115	295,417	246,255	258,466	258,155	263,919	300,990	289,045	49,094

Tabela 6.9: Tabela com os tempos de execução, em segundos. A primeira linha representa a variação do número de processos, enquanto que a primeira coluna representa o número de eventos.

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000	257%	240%	250%	251%	251%	250%	255%	251%	5%
1000000	500%	508%	495%	499%	491%	498%	558%	507%	23%

Tabela 6.10: Comparação percentual entre tempos de execução assumidos como base e restantes

sultados apresentam-se nas tabelas 6.11 e 6.12 e figura 6.6. Novamente verifica-se um crescimento da memória com o número eventos a processar. Verifica-se também que esse crescimento não segue as mesmas proporções que os aumentos no número de eventos.

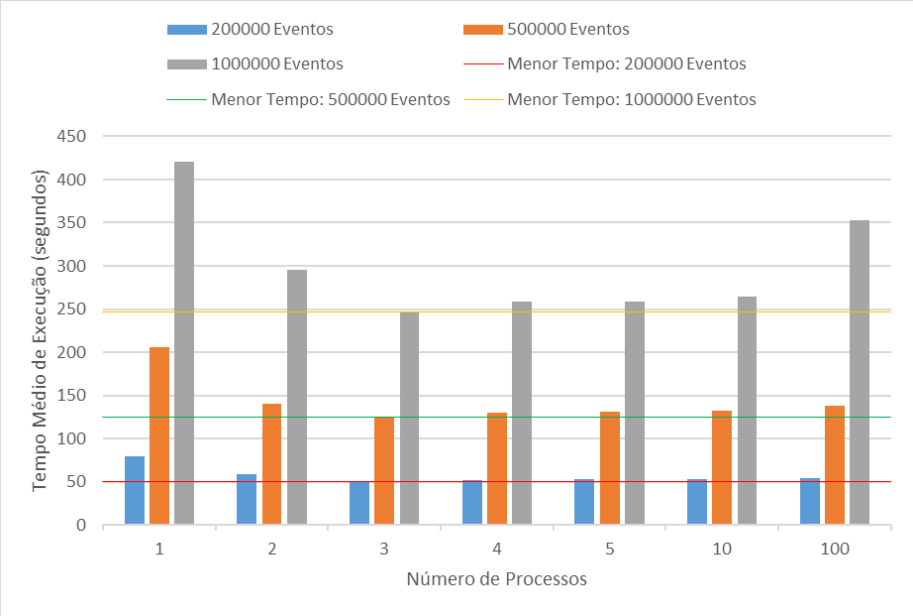

Figura 6.5: Média dos tempos de execução, em segundos, dos vários testes realizados

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	99,626	101,355	101,325	106,589	103,845	103,753	106,676	103,310	2,504
500000	140,271	150,190	149,264	158,918	146,929	147,546	159,784	150,414	6,378
1000000	183,302	194,924	199,672	214,777	179,298	190,730	216,035	196,963	13,256

Tabela 6.11: Média da memória utilizada por teste

	1	2	3	4	5	10	100	Média	Desvio Padrão
200000	100%	100%	100%	100%	100%	100%	100%	100%	0%
500000	141%	148%	147%	149%	141%	142%	150%	146%	4%
1000000	184%	192%	197%	201%	173%	184%	203%	191%	11%

Tabela 6.12: Comparação percentual entre valores de memória consumida assumidos como base e restantes

Conclusões

Como podemos verificar pelos testes apresentados todos os *bots* apresentaram melhor desempenho quando configurados com 3 processos. Isto deve-se ao facto de a máquina onde foram corridos os testes possuir 4 núcleos de processamento. Um desses núcleos encontrava-se ocupado pelo processo pai, responsável por distribuir as tarefas pelos processos trabalhadores da *pool*.

Verificámos também que os aumentos de tempo de execução seguiram as mesmas proporções que o aumento de tarefas a realizar, facto que nos permitiu validar a escalabilidade dos *bots*. À semelhança do tempo de execução também a memória aumenta, mas em proporções mais baixas. Isto deve-se ao facto de , como já foi

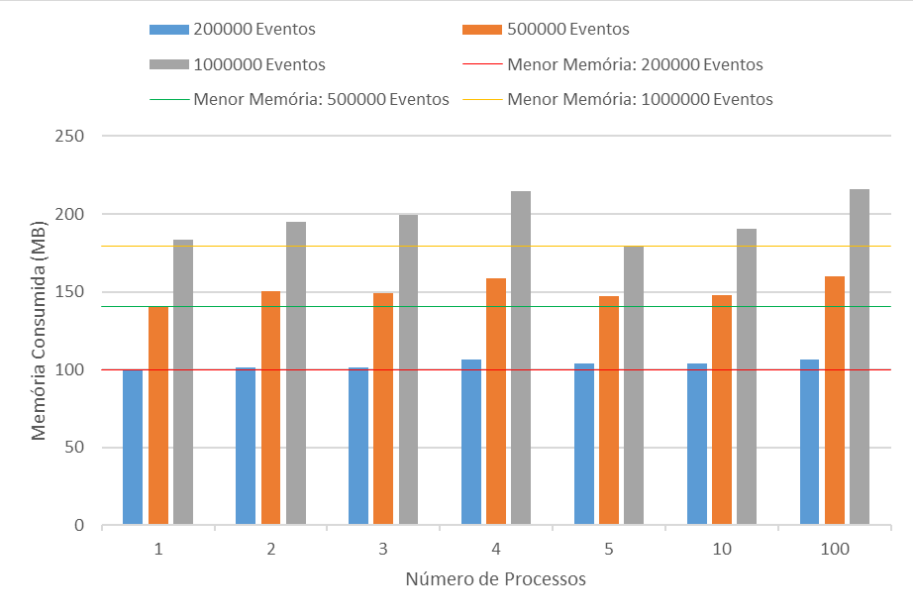

Figura 6.6: Média da memória utilizada por teste

referido, os testes terem envolvido maioritariamente processamento.
Conclui-se também que, como esperado, o testes com apenas um processo apresentam os piores desempenhos. De notar que esta comparação não é de todo justa porque uma versão sequencial dos testes não foi escrita. Uma aproximação à mesma foi feita correndo o programa usando apenas um único processo trabalhador na *pool*. Isto levou a um provável tempo de execução superior ao de uma versão sequencial devido às despesas temporais da criação da *pool* e envio da tarefa para o único processo da mesma. Assim sendo, estes resultados possivelmente exageram, ainda que de uma forma ligeira, a melhoria de desempenho da versão multi processador.
Por fim, importa referir que dada a semelhança de execução entre o *bot* do Facebook e os restantes *bots* de extração de informação e dada também a natureza repetitiva dos testes o estagiário optou por não incluir a documentação dos mesmos. Os testes realizados nestes *bots* (produtores) limita-se à parte de processamento dos dados por forma a colocá-los de acordo com o modelo de dados, não incidindo sobre a parte de recolha de informação. Caso contrário, como já foi referido, não iria ser possível realizar um número tão elevado de pedidos como os números usados nestes testes.

Capítulo 7

Conclusão

Este capítulo serve para apresentar as reflexões finais ao estágio realizado pelo autor deste documento. Apresentam-se as notas conclusivas e o futuro da aplicação.
O presente relatório de estágio documenta o esforço realizado ao longo dos últimos doze meses. O projeto de estágio assentou na expansão dos coletores de informação do Portolan, em especifico para extração de dados dos média sociais e na visualização dos mesmos. A escolha de requisitos teve o objetivo de alinhar o Portolan com os seus concorrentes de mercado, mas também diferenciá-lo dos mesmos. À semelhança do que acontece em muitos projetos houve algumas dificuldades que foram resolvidas da melhor forma com a ajuda dos membros da Dognædis. Das dificuldades encontradas apresentam-se as seguintes:

Modelo de Dados Uma das maiores dificuldades foi o a existência de dados do tipo *lista* no modelo de dados. A rotina de validação dos dados de entrada com o modelo de dados não se encontrava preparada para este tipo de dados, pois até à data não tinha sido necessário. Foi então necessária a compreensão do código existente para que se procede-se à implementação de forma a garantir o funcionamento tanto do já existente como do que se pretendia adicionar.

Interface de Configuração de bots Outra dificuldade encontrada foi a compreensão de como o processo de geração das interfaces era feito, bem como a melhor forma de alterar a mesma. Isto porque se pretendia respeitar as linhas de guia apresentadas pela interface existente. A primeira assenta na criação de uma interface simples e intuitiva criada a partir de um *template* em formato JSON. A segunda assenta nos tons de cores usados no Portolan, tudo muito à base dos cinzentos. Foi também necessário analisar qual seria a melhor maneira de agrupar o conteúdo relativo a determinado elemento para que quando fosse feita a leitura dos dados de entrada se soubesse facilmente a que elemento pertencia.

Debug Outra dificuldade foi o tempo de *debug*, em particular na validação dos *crawlers*. Dada a dificuldade de criação de casos de teste reais, para cada teste era necessário executar sempre o programa desde o inicio até à parte que se queria testar. E como foi mencionado, os *crawlers* demoram mais tempos a extrair a informação, pois são feitos *sleeps* de no mínimo dois segundos entre pedidos. Isto leva a que o processo de *debug* se torne demorado.

O facto de se estar a trabalhar em ambiente empresarial, sobre um produto já existente e em produção, além da utilização de tecnologias até então desconhecidas pelo estagiário contribuiu para o crescimento profissional e pessoal. Além da aquisição de novos conhecimentos, este percurso permitiu também consolidar conhecimentos já possuídos, em particular as tecnologias *web*.
Face aos objetivos propostos pode-se afirmar que o resultado do trabalho desenvolvido é positivo. Apenas dois requisitos funcionais dos propostos não foram implementados, nomeadamente a extração das partilhas de um *post*.
Durante o desenvolvimento foram aparecendo ideias para novas funcionalidades, expansão das já existentes ou até mesmo a integração das funcionalidades implementadas com dados provenientes de outras fontes:

Adição de mais ações ao grafo Um exemplo de uma ação seria efetuar uma pesquisa na base de dados para obter todo o conteúdo relativo a determinado nodo. Outro exemplo seria analisar a existência de uma ligação entre determinados dois nodos.

Aplicação do grafo desenvolvido a outro tipo de dados O grafo desenvolvido poderá facilmente permitir a visualização das ligações entre outro tipo de dados. Um exemplo seriam as ligações entre endereços IP.

Realização de testes de usabilidade Como foi mencionado não foi possível realizar testes de usabilidade devido à inexistência de sujeitos adequados para a realização dos mesmos. Posto isto, os testes de usabilidade deverão ser realizados com os primeiros utilizadores da aplicação.

Classificador baseado em machine learning Quando foi descrita a interface de configuração manual mencionou-se que esta funcionava como um mecanismo *Human-in-the-loop*. Seria interessante do ponto de vista do autor, e possivelmente vantajoso para o futuro do Portolan, a validação da possibilidade de utilização desta interface como meio de aprendizagem de um modelo de *machine learning* capaz de classificar automaticamente a informação que lhe é fornecida.

Bibliografia

[1] "Intellingence: Open source intelligence." https://www.cia.gov/news-information/featured-story-archive/2010-featured-story-archive/open-source-intelligence.html, 2010. Acedido: 2015-12-16.

[2] P. D. M. Vieira, "Module 3: Planning & Tracking – Planning & Monitoring," 2015.

[3] P. D. M. Vieira, "Module 3: Planning & Tracking - Bottom-up Estimation: A Real Process," 2015.

[4] "The scrum guide." http://www.scrumguides.org/scrum-guide.html#artifact-transparency-done, 2014. Acedido: 2016-08-24.

[5] "The oauth 2.0 authorization framework." https://tools.ietf.org/html/rfc6749. Acedido: 2016-8-23.

[6] "Oauth core 1.0a." https://oauth.net/core/1.0a/. Acedido: 2016-8-25.

[7] "Facebook." https://www.facebook.com. Acedido: 2016-8-23.

[8] "Number of monthly active facebook users worldwide as of 3rd quarter 2015 (in millions)." http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/. Acedido: 2015-12-16.

[9] "Graph api." https://developers.facebook.com/docs/graph-api. Acedido: 2016-8-23.

[10] "Facebook social graph example." http://www.programmableweb.com/wp-content/open.graph-600x403.png. Acedido: 2016-8-23.

[11] "Twitter." https://twitter.com/. Acedido: 2016-8-23.

[12] "Twitter | empresa | about." https://about.twitter.com/pt/company. Acedido: 2015-12-16.

[13] "Twitter rest api." https://dev.twitter.com/rest/public. Acedido: 2016-8-23.

[14] "Json." http://www.w3schools.com/json/. Acedido: 2016-8-23.

[15] "Twitter streaming api." https://dev.twitter.com/streaming/overview. Acedido: 2016-8-23.

[16] "Hypertext transfer protocol – http/1.1." https://www.ietf.org/rfc/rfc2616.txt. Acedido: 2016-8-23.

[17] "Imagem twitter operations." http://cdn2.hubspot.net/hub/722270/file-3846299689-jpg/blog-files/screen-shot-2010-12-08-at-7.12.jpg?t=1472024556015. Acedido: 2016-8-25.

[18] "Reddit." https://www.reddit.com/. Acedido: 2016-8-23.

[19] "60 amazing reddit statistics (january 2016)." http://expandedramblings.com/index.php/reddit-stats/. Acedido: 2015-12-16.

[20] "Snapchat phone number database leaked - 4.6 million users affected: netsec." https://www.reddit.com/r/netsec/comments/1u4xss/snapchat_phone_number_database_leaked_46_million/recorded. Acedido: 2016-3-2.

[21] "Advertise - 4chan." http://www.4chan.org/advertise. Acedido: 2015-12-16.

[22] "4chan api." https://github.com/4chan/4chan-API. Acedido: 2015-12-16.

[23] "Taking the rick | music | the guardian." https://www.theguardian.com/music/2008/mar/19/news. Acedido: 2016-3-2.

[24] "Rfc 2810 - internet relay chat: Architecture." https://tools.ietf.org/html/rfc2810. Acedido: 2016-3-2.

[25] "Github - neweracracker/loic: Low orbit ion cannon - an open source network stress tool, written in c-sharp. based on praetox's loic project. use on your own risk. without any express or implied warranties.." https://github.com/NewEraCracker/LOIC. Acedido: 2016-3-2.

[26] "Cyveillance." https://www.cyveillance.com/. Acedido: 2015-12-16.

[27] "Digitalstakeout." http://www.digitalstakeout.com/. Acedido: 2015-12-16.

[28] "Recorded future." https://www.recordedfuture.com. Acedido: 2015-12-16.

[29] "Cyber intelligence | sensecy." https://www.sensecy.com/. Acedido: 2015-12-16.

[30] "Social media security & threat intelligence | zerofox." https://www.zerofox.com/. Acedido: 2015-12-16.

[31] "Figura 3.3." http://www.socialmediatoday.com/sites/default/files/MarketingXLerator/files/Cyveillance.jpg. Acedido: 2016-8-23.

[32] "Threat intelligence platform." http://www.digitalstakeout.com/threat-intelligence-platform. Acedido: 2015-12-16.

[33] "Recorded future's web intelligence engine." https://www.recordedfuture.com/web-intelligence-engine/. Acedido: 2015-12-16.

[34] "Cyber intelligence feeds | sensecy." https://www.sensecy.com/products/cyber-intelligence-feeds. Acedido: 2015-12-16.

[35] "Platform | zerofox." https://www.zerofox.com/platform/. Acedido: 2015-12-16.

[36] "Platform | zerofox | image." https://media.licdn.com/media/AAEAAQAAAAAAAAOpAAAAJDE0MTc4YWVlLTA2ZmQtNDQ1ZC05ZGE2LThiYmQ0MGQxNGM5Yw.png. Acedido: 2015-12-16.

[37] "Scrapy | a fast and powerful scraping and web crawling framework." http://scrapy.org/. Acedido: 2016-8-24.

[38] "20.5. urllib — open arbitrary resources by url - python 2.7.11 documentation." https://docs.python.org/2/library/urllib.html. Acedido: 2016-8-24.

[39] "20.6. urllib2 — extensible library for opening urls - python 2.7.11 documentation." https://docs.python.org/2/library/urllib2.html. Acedido: 2016-8-24.

[40] "Requests: Http for humans - requests 2.9.1 documentation." http://docs.python-requests.org/en/latest/. Acedido: 2016-8-24.

[41] "mechanize." http://wwwsearch.sourceforge.net/mechanize/. Acedido: 2016-8-24.

[42] "19.1. htmlparser — simple html and xhtml parser - python 2.7.11 documentation." https://docs.python.org/2/library/htmlparser.html. Acedido: 2016-8-24.

[43] "lxml - processing xml and html with python." http://lxml.de/. Acedido: 2016-8-24.

[44] "html5lib/html5lib-python: Standards-compliant library for parsing and serializing html documents and fragments in python." https://github.com/html5lib/html5lib-python. Acedido: 2016-8-24.

[45] "Beautiful soup: We called him tortoise because he taught us.." http://www.crummy.com/software/BeautifulSoup/. Acedido: 2016-8-24.

[46] "oauthlib 1.1.2 : Python package index." https://pypi.python.org/pypi/oauthlib. Acedido: 2016-8-24.

[47] "requests-oauthlib 0.6.2 : Python package index." https://pypi.python.org/pypi/requests-oauthlib. Acedido: 2016-8-24.

[48] "Welcome to python.org." https://www.python.org/. Acedido: 2016-8-23.

[49] "W3c html." https://www.w3.org/html/. Acedido: 2016-8-23.

[50] "Xml tutorial." http://www.w3schools.com/xml/. Acedido: 2016-8-24.

[51] "Xml path language (xpath)." https://www.w3.org/TR/xpath/. Acedido: 2016-8-23.

[52] "Comparison of html parsers - wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/Comparison_of_HTML_parsers. Acedido: 2016-3-2.

[53] "Xhtml 1.0: The extensible hypertext markup language (second edition)." https://www.w3.org/TR/xhtml1/. Acedido: 2016-8-23.

[54] "Html standard." https://html.spec.whatwg.org/multipage/. Acedido: 2016-8-23.

[55] "Html tree." http://www.w3schools.com/js/pic_htmltree.gif. Acedido: 2016-3-2.

[56] "Sopel irc bot - github." https://github.com/sopel-irc. Acedido: 2016-8-24.

[57] "brosner/bosnobot - python - github." https://github.com/brosner/bosnobot. Acedido: 2016-8-24.

[58] "minchat." http://twistedmatrix.com/documents/current/_downloads/minchat.py. Acedido: 2016-8-24.

[59] "irclogbot." http://twistedmatrix.com/documents/current/_downloads/ircLogBot.py. Acedido: 2016-8-24.

[60] "Response time limits: Article by jakob nielsen." https://www.nngroup.com/articles/response-times-3-important-limits/. Acedido: 2016-8-24.

[61] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, ch. 2, pp. 74, 75. Pearson, 2009.

[62] "Redis." http://redis.io/. Acedido: 2016-8-25.

[63] "Mongodb for giant ideas | mongodb." https://www.mongodb.com/. Acedido: 2016-8-25.

[64] "Arquitetura de uma aplicação django que serviu como base para a figura 5.3." http://medcosts.web.unc.edu/files/2014/04/Architecture_Diagram.png. Acedido: 2016-8-24.

[65] "Django documentation | django documentation | django." https://docs.djangoproject.com/en/1.10/#the-model-layer. Acedido: 2016-8-2.

[66] "Django documentation | django documentation | django." https://docs.djangoproject.com/en/1.10/#the-view-layer. Acedido: 2016-8-2.

[67] "Django documentation | django documentation | django." https://docs.djangoproject.com/en/1.10/#the-template-layer. Acedido: 2016-8-2.

[68] "Selenium - web browser automation." http://www.seleniumhq.org/. Acedido: 2016-8-25.

[69] "Imagem reactor." https://cygnusclouducm.files.wordpress.com/2013/04/bucle-reactor1.png. Acedido: 2016-8-25.

[70] "Imagem twitter timeline." https://g.twimg.com/dev/sites/default/files/images_documentation/paging-04.png. Acedido: 2016-8-25.

[71] "Streaming api request parameters | twitter developers." https://dev.twitter.com/streaming/overview/request-parameters#track. Acedido: 2016-8-2.

[72] "Streaming api request parameters | twitter developers." https://dev.twitter.com/streaming/overview/request-parameters#follow. Acedido: 2016-8-2.

[73] "D3.js - data-driven documents." https://d3js.org/. Acedido: 2016-8-25.

[74] "Alchemylanguage | ibm watson developer cloud." https://www.ibm.com/watson/developercloud/alchemy-language.html. Acedido: 2016-8-25.

[75] "Ibm - portugal." https://www.ibm.com/pt-pt/. Acedido: 2016-8-25.

[76] "Ibm watson." http://www.ibm.com/watson/. Acedido: 2016-8-25.

[77] "bootstrap-select." https://silviomoreto.github.io/bootstrap-select/. Acedido: 2016-8-25.

[78] "Why human-in-the-loop computing is the future of machine learning | computerworld." http://www.computerworld.com/article/3004013/robotics/why-human-in-the-loop-computing-is-the-future-of-machine-learning.html. Acedido: 2016-8-25.

[79] "Machine learning: What it is and why it matters | sas." http://www.sas.com/en_id/insights/analytics/machine-learning.html. Acedido: 2016-8-25.

[80] "Bootstrap panels." http://www.w3schools.com/bootstrap/bootstrap_panels.asp. Acedido: 2016-8-25.

[81] "Bootstrap form inputs." http://www.w3schools.com/bootstrap/bootstrap_forms_inputs.asp. Acedido: 2016-8-25.

[82] "Bootstrap tree view." https://github.com/jonmiles/bootstrap-treeview. Acedido: 2016-8-25.

[83] M. A. Russel, *Mining the Social Web*, ch. 1, p. 9. O'Reilly Media, 2011.

[84] "The cyveillance cyber threat center." https://www.cyveillance.com/home/security-solutions/security-platform-tools/cyveillance-cyber-threat-center/. Acedido: 2015-12-16.

[85] "The graph api." https://developers.facebook.com/docs/graph-api. Acedido: 2015-12-16.

[86] "Rest apis." https://dev.twitter.com/rest/public. Acedido: 2015-12-16.

[87] "The streaming apis." https://dev.twitter.com/streaming/overview. Acedido: 2015-12-16.

[88] "reddit.com: about reddit." https://www.reddit.com/about/. Acedido: 2015-12-16.

[89] J. Oikarinen and D. Reed, "Internet Relay Chat Protocol." https://tools.ietf.org/html/rfc1459, 1993. Acedido: 2015-12-16.

[90] E. Hammer and E. Lahav, "Rfc 5849 - the oauth 1.0 protocol." https://tools.ietf.org/html/rfc5849, 2010. Acedido: 2016-3-3.

[91] "Clearing up confusion - deep web vs. dark web." http://www.brightplanet.com/2014/03/clearing-confusion-deep-web-vs-dark-web/. Acedido: 2016-8-24.

[92] "The internet, the deep web, and the dark web." https://danielmiessler.com/study/internet-deep-dark-web/. Acedido: 2016-8-24.

[93] "Owasp secure coding practices quick reference guide." https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf, 2010. Acedido: 2016-8-24.

[94] "Top 10 secure coding practices - secure coding - cert secure coding standards." https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Coding+Practices. Acedido: 2016-8-24.

[95] *Joint Intelligence*, pp. GL–08. 2013. Acedido: 2016-1-2.

[96] "Twitter: Working with timelines." https://g.twimg.com/dev/sites/default/files/images_documentation/paging-04.png. Acedido: 2016-8-24.

[97] "Anonymous (group) - wikipedia, the free encyclopedia." https://en.wikipedia.org/wiki/Anonymous_(group). Acedido: 2016-8-24.

[98] "Rfc 2617 - http authentication: Basic and digest access authentication." https://tools.ietf.org/html/rfc2617. Acedido: 2016-8-24.

[99] "O que é botnet?." https://www.microsoft.com/pt-br/security/resources/botnet-whatis.aspx. Acedido: 2016-8-24.

[100] "Cascading style sheets." https://www.w3.org/Style/CSS/. Acedido: 2016-8-24.

[101] "What is a ddos attack?." http://www.digitalattackmap.com/understanding-ddos/. Acedido: 2016-8-24.

[102] "What is drilldown?." http://searchsqlserver.techtarget.com/definition/drilldown. Acedido: 2016-8-24.

[103] "Regular-expressions.info - regex tutorial, examples and reference - regexp patterns." http://www.regular-expressions.info/. Acedido: 2016-8-24.

[104] "What is framework?." http://whatis.techtarget.com/definition/framework. Acedido: 2016-8-24.

[105] "What is event handler?." http://searchsoa.techtarget.com/definition/event-handler. Acedido: 2016-8-24.

[106] "What is hacktivism?." http://searchsecurity.techtarget.com/definition/hacktivism. Acedido: 2016-8-24.

[107] "Http/1.1: Header field definitions." https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html. Acedido: 2016-8-24.

[108] "Imageboard." https://en.wikipedia.org/wiki/Imageboard. Acedido: 2016-8-24.

[109] "What is message broker?." http://whatis.techtarget.com/definition/message-broker. Acedido: 2016-8-24.

[110] "What is middleware?." http://searchsoa.techtarget.com/definition/middleware. Acedido: 2016-8-24.

[111] "Bootstrap modals." http://www.w3schools.com/bootstrap/bootstrap_modal.asp. Acedido: 2016-8-24.

[112] "What is polling?." http://whatis.techtarget.com/definition/polling. Acedido: 2016-8-24.

[113] "What is proxy server?." http://whatis.techtarget.com/definition/proxy-server. Acedido: 2016-8-24.

[114] "Rfc 6749 - the oauth 2.0 authorization framework." https://tools.ietf.org/html/rfc6749#page-10. Acedido: 2016-8-24.

[115] "Database triggers." https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch15.htm. Acedido: 2016-8-24.

[116] "The web framework for perfectionists with deadlines | django." https://www.djangoproject.com/. Acedido: 2016-8-25.

Apêndice A

Requisitos Funcionais

Neste capítulo serão apresentados a lista, a tabela de priorização e descrição, a tabela com a implementação e os testes aos requisitos funcionais.

A.1 Lista de Requisitos Funcionais

Nesta secção apresenta-se a lista de requisitos funcionais final do projeto desenvolvido.

1. Extrair dados
    a. API
        i. Facebook
            A. Extrair conteúdo
            B. Extrair autor
            C. Extrair URL
            D. Extrair data
            E. Extrair localização (coordenadas, país, cidade, ...)
            F. Extrair *hashtags*
            G. Extrair partilhas
            H. Extrair reações
            I. Extrair *tags*
            J. Comentários
                I. Extrair conteúdo
                II. Extrair autor
                III. Extrair URL
                IV. Extrair data
                V. Extrair *hashtags*
                VI. Extrair "gostos"
                VII. Extrair *tags*
                VIII. Extrair respostas
                    1. Extrair conteúdo
                    2. Extrair autor

3. Extrair URL
4. Extrair data
5. Extrair *hashtags*
6. Extrair "gostos"
7. Extrair *tags*

ii. Twitter
A. Tweets
I. Extrair conteúdo
II. Extrair autor
III. Extrair URL
IV. Extrair data
V. Extrair localização
VI. Extrair *tags*
VII. Extrair *hashtags*
VIII. Extrair *retweets*
IX. Extrair *Replies*
1. Extrair conteúdo
2. Extrair autor
3. Extrair URL
4. Extrair data
5. Extrair localização
6. Extrair *tags*
7. Extrair *hashtags*

iii. Reddit
A. Extrair informação de uma *thread*
I. Extrair conteúdo
II. Extrair autor
III. Extrair URL
IV. Extrair data
V. Extrair número de votos para cima
VI. Extrair numero de votos para baixo
VII. Extrair respostas
1. Extrair conteúdo
2. Extrair autor
3. Extrair URL
4. Extrair data
5. Extrair número de votos para cima
6. Extrair numero de votos para baixo

iv. 4chan
A. Extrair informação de uma *thread*
I. Extrair conteúdo
II. Extrair URL
III. Extrair data

IV. Extrair respostas
1. Extrair conteúdo
2. Extrair URL
3. Extrair data

b. Crawler
i. Facebook
A. *Posts* no mural de um utilizador
I. Extrair conteúdo
II. Extrair autor
III. Extrair URL
IV. Extrair data
V. Extrair localização (coordenadas, país, cidade, ...)
VI. Extrair *hashtags*
VII. Extrair partilhas
VIII. Extrair reações
IX. Extrair *tags*
X. Comentários
1. Extrair conteúdo
2. Extrair autor
3. Extrair URL
4. Extrair data
5. Extrair *hashtags*
6. Extrair "gostos"
7. Extrair *tags*
8. Extrair respostas
a. Extrair conteúdo
b. Extrair autor
c. Extrair URL
d. Extrair data
e. Extrair *hashtags*
f. Extrair "gostos"
g. Extrair *tags*
B. Extrair informação de *friendship* (*friendship* mostra tudo o que existe em comum entre dois utilizadores.)
I. Extrair data de amizade no Facebook
II. Extrair locais de residência
III. Extrair locais de educação
IV. Extrair *posts*
C. Extrair informação de perfil
I. Extrair locais de trabalho
II. Extrair locais de educação
III. Extrair locais de residência
IV. Extrair data de nascimento

V. Extrair religião
VI. Extrair campo "About"
VII. Extrair relacionamentos
VIII. Extrair família

ii. Twitter
A. Extrair localização de um perfil
B. Extrair data de nascimento de um perfil
C. Extrair campo "About"
D. Extrair comentários de um *post*
E. Extrair gostos d um *post*

iii. Servidores/Canais IRC
A. Extrair informação das mensagens trocadas
I. Extrair conteúdo
II. Extrair autor
III. Extrair destinatário
IV. Extrair data
B. Extrair informação de um autor
C. Extrair eventos
I. Entrada no canal
II. Saída no canal
III. Expulsão do canal
IV. Mudança de nome de utilizador

2. Utilizar *proxy*

3. Processar dados

a. Analisar Padrões
i. Procurar IPs
A. Procurar IPv4
B. Procurar IPv6
ii. Procurar Domínios
iii. Procurar endereços MAC
iv. Procurar *Keywords*
v. Procurar pares *tag:list*

b. Análise de Texto

4. Desenvolver Interfaces Gráficas

a. *Back-office* para configuração
i. Inserir novo *bot*
ii. Configurar *bot*
A. Editar nome
B. Editar alvo
C. Editar pesquisa de endereços IPs
I. Editar IPv4

II. Editar IPv6
D. Editar pesquisa de Domínios
E. Editar pesquisa de endereços MAC
F. Editar pesquisa de *keywords*
G. Editar pares *tag*:*list*
H. Editar parâmetros de Análise Textual
I. Editar número de páginas a extrair
J. Editar credenciais de acesso à API
K. Editar credenciais de login na fonte
L. Alterar campos para procura de padrões
M. Alterar campos a extrair

iii. Eliminar bot
iv. Adicionar *proxy*
v. Remover *proxy*

b. Interface para classificação de dados
i. Apresentar dados
A. JSON
B. Apresentação ao estilo Facebook/Twitter
C. Tabela
ii. Rotular/categorizar/classificar manualmente evento
iii. Eliminar evento

c. Interface de visualização de dados
i. Grafo
A. Centrar Nodo
B. Realçar Nodo
C. Realçar Ligações
D. Fazer zoom (in e out)
E. Arrastar Grafo e Nodos
F. Pesquisar Nodo
G. Definir Profundidade do Grafo
H. Definir Intervalo Temporal
I. Usar Menu de Contexto para expandir grafo mantendo os nodos existentes
J. Usar Menu de Contexto para expandir grafo removend os nodos não conectados
ii. Widgets
A. Definir tipo de estatística (Soma ou Média)
B. Definir campo a ser usado no cálculo da estatística
iii. Tabela
A. Escolher Colunas
B. Pesquisar colunas
C. Visualizar detalhadamente evento

5. Definir Modelo de Dados

6. Permitir Suporte a Listas
    a. Suportar listas no modelo de dados
    b. Suportar listas nos *widgets*
    c. Suportar listas na tabela
    d. Suportar listas na matriz

A.2 Tabela de Prioridade dos Requisitos Funcionais

Esta secção apresenta os requisitos funcionais, a descrição e prioridade de implementação dos mesmos.

ID	**Descrição**	**Prioridade**
RF-1aiA	O sistema consegue extrair o conteúdo de um *post* no Facebook	A
RF-1aiB	O sistema consegue extrair o autor de um *post* no Facebook	A
RF-1aiC	O sistema consegue extrair o URL de um *post* no Facebook	A
RF-1aiD	O sistema consegue extrair a data de criação de um *post* no Facebook.	A
RF-1aiE	O sistema consegue extrair a localização de um *post* no Facebook.	A
RF-1aiF	O sistema consegue extrair as *hashtags* de um *post* no Facebook.	A
RF-1aiG	O sistema consegue extrair as partilhas de um *post* no Facebook.	A
RF-1aiH	O sistema consegue extrair as reações de um *post* no Facebook.	A
RF-1aiI	O sistema consegue extrair as *tags* de um *post* no Facebook.	A
RF-1aiJI	O sistema consegue extrair o conteúdo dos comentários de um *post* no Facebook.	A
RF-1aiJII	O sistema consegue extrair o autor dos comentários de um *post* no Facebook.	A
RF-1aiJIII	O sistema consegue extrair o URL dos comentários de um *post* no Facebook.	A
RF-1aiJIV	O sistema consegue extrair a data de criação dos comentários de um *post* no Facebook.	A
RF-1aiJV	O sistema consegue extrair as *hashtags* dos comentários de um *post* no Facebook.	A
RF-1aiJVI	O sistema consegue extrair os "gostos"dos comentários de um *post* no Facebook.	A
RF-1aiJVII	O sistema consegue extrair as *tags* dos comentários de um *post* no Facebook.	A
RF-1aiJVIII1	O sistema consegue extrair o conteúdo das respostas aos comentários de um *post* no Facebook.	A

ID	Descrição	Prioridade
RF-1aiJVIII2	O sistema consegue extrair o autor das respostas aos comentários de um *post* no Facebook.	A
RF-1aiJVIII3	O sistema consegue extrair o URL das respostas aos comentários de um *post* no Facebook.	A
RF-1aiJVIII4	O sistema consegue extrair a data de criação das respostas aos comentários de um *post* no Facebook.	A
RF-1aiJVIII5	O sistema consegue extrair as *hashtags* das respostas aos comentários de um *post* no Facebook.	A
RF-1aiJVIII6	O sistema consegue extrair os "gostos"das respostas aos comentários de um *post* no Facebook.	A
RF-1aiJVIII7	O sistema consegue extrair as *tags* das respostas aos comentários de um *post* no Facebook.	A
RF-1aiiAI	O sistema consegue extrair o conteúdo de um *tweet*.	A
RF-1aiiAII	O sistema consegue extrair o autor de um *tweet*.	A
RF-1aiiAIII	O sistema consegue extrair o URL de um *tweet*.	A
RF-1aiiAIV	O sistema consegue extrair a data de um *tweet*.	A
RF-1aiiAV	O sistema consegue extrair a localização de um *tweet*.	A
RF-1aiiAVI	O sistema consegue extrair as *tags* de um *tweet*.	A
RF-1aiiAVII	O sistema consegue extrair as *hashtags* de um *tweet*.	A
RF-1aiiAVIII	O sistema consegue extrair os *retweets* de um *tweet*.	A
RF-1aiiAIX1	O sistema consegue extrair o conteúdo das respostas de um *tweet*.	A
RF-1aiiAIX2	O sistema consegue extrair o autor das respostas de um *tweet*.	A
RF-1aiiAIX3	O sistema consegue extrair o URL das respostas de um *tweet*.	A
RF-1aiiAIX4	O sistema consegue extrair a data das respostas de um *tweet*.	A
RF-1aiiAIX5	O sistema consegue extrair a localização das respostas de um *tweet*.	A
RF-1aiiAIX6	O sistema consegue extrair as *tags* das respostas de um *tweet*.	A
RF-1aiiAIX7	O sistema consegue extrair as *hashtags* das respostas de um *tweet*.	A
RF-1aiiiAI	O sistema consegue extrair o conteúdo de uma *thread*.	A
RF-1aiiiAII	O sistema consegue extrair o autor de uma *thread*.	A
RF-1aiiiAIII	O sistema consegue extrair o URL de uma *thread*.	A
RF-1aiiiAIV	O sistema consegue extrair a data de uma *thread*.	A
RF-1aiiiAV	O sistema consegue extrair os votos para cima de uma *thread*.	A
RF-1aiiiAVI	O sistema consegue extrair os votos para baixo de uma *thread*.	A
RF-1aiiiAVII1	O sistema consegue extrair o conteúdo dos comentários de uma *thread*.	A
RF-1aiiiAVII2	O sistema consegue extrair o autor dos comentários de uma *thread*.	A

ID	Descrição	Prioridade
RF-1aiiiAVII3	O sistema consegue extrair o URL dos comentários de uma *thread*.	A
RF-1aiiiAVII4	O sistema consegue extrair a data dos comentários de uma *thread*.	A
RF-1aiiiAVII5	O sistema consegue extrair os votos para cima dos comentários de uma *thread*.	A
RF-1aiiiAVII6	O sistema consegue extrair os votos para baixo dos comentários de uma *thread*.	A
RF-1aivAI	O sistema consegue extrair o conteúdo de uma *thread* do 4chan.	A
RF-1aivAII	O sistema consegue extrair o URL de uma *thread* do 4chan.	A
RF-1aivAIII	O sistema consegue extrair a data de uma *thread* do 4chan.	A
RF-1aivAIV1	O sistema consegue extrair o conteúdo dos comentários de uma *thread* do 4chan.	A
RF-1aivAIV2	O sistema consegue extrair o URL dos comentários de uma *thread* do 4chan.	A
RF-1aivAIV3	O sistema consegue extrair a data dos comentários de uma *thread* do 4chan.	A
RF-1biAI	O sistema consegue extrair o conteúdo de um *post* no mural de um utilizador.	A
RF-1biAII	O sistema consegue extrair o autor de um *post* no mural de um utilizador.	A
RF-1biAIII	O sistema consegue extrair o URL de um *post* no mural de um utilizador.	A
RF-1biAIV	O sistema consegue extrair a data de um *post* no mural de um utilizador.	A
RF-1biAV	O sistema consegue extrair a localização de um *post* no mural de um utilizador.	A
RF-1biAVI	O sistema consegue extrair as *hashtags* de um *post* no mural de um utilizador.	A
RF-1biAVII	O sistema consegue extrair as partilhas de um *post* no mural de um utilizador.	A
RF-1biAVIII	O sistema consegue extrair as reações de um *post* no mural de um utilizador.	A
RF-1biAIX	O sistema consegue extrair as *tags* de um *post* no mural de um utilizador.	A
RF-1biAX1	O sistema consegue extrair o conteúdo dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX2	O sistema consegue extrair o autor dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX3	O sistema consegue extrair o URL dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX4	O sistema consegue extrair a data dos comentários de um *post* no mural de um utilizador.	A

ID	Descrição	Prioridade
RF-1biAX5	O sistema consegue extrair as *hashtags* dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX6	O sistema consegue extrair os gostos dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX7	O sistema consegue extrair as *tags* dos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8a	O sistema consegue extrair o conteúdo das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8b	O sistema consegue extrair o autor das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8c	O sistema consegue extrair o URL das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8d	O sistema consegue extrair a data das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8e	O sistema consegue extrair as *hashtags* das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8f	O sistema consegue extrair os gostos das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biAX8g	O sistema consegue extrair as *tags* das respostas aos comentários de um *post* no mural de um utilizador.	A
RF-1biB	O sistema consegue extrair a informação de amizade. A informação de amizade mostra tudo o que existe em comum entre dois utilizadores.	A
RF-1biBI	O sistema consegue extrair a data de amizade no Facebook entre dois utilizadores.	A
RF-1biBII	O sistema consegue extrair os locais de residência comuns entre dois utilizadores.	A
RF-1biBIII	O sistema consegue extrair os locais de educação comuns entre dois utilizadores.	A
RF-1biBIV	O sistema consegue extrair a informação de *posts* comuns entre dois utilizadores.	A
RF-1biC	O sistema consegue extrair a informação de perfil de um utilizador.	A
RF-1biCI	O sistema consegue extrair os locais de trabalho de um utilizador.	A
RF-1biCII	O sistema consegue extrair os locais de educação de um utilizador.	A
RF-1biCIII	O sistema consegue extrair os locais de residência de um utilizador.	A
RF-1biCIV	O sistema consegue extrair a data de nascimento de um utilizador de um utilizador.	A
RF-1biCV	O sistema consegue extrair a religião de um utilizador.	A
RF-1biCVI	O sistema consegue extrair o campo "About"de trabalho de um utilizador.	A
RF-1biCVII	O sistema consegue extrair os relacionamentos de um utilizador.	A

ID	Descrição	Prioridade
RF-1biCVIII	O sistema consegue extrair as relações familiares de um utilizador.	A
RF-1biiA	O sistema consegue extrair a localização de um perfil do Twitter.	A
RF-1biiB	O sistema consegue extrair a data de nascimento de um perfil do Twitter.	A
RF-1biiC	O sistema consegue extrair o campo "About"de um perfil do Twitter.	A
RF-1biiD	O sistema consegue extrair os comentários de um Tweet.	A
RF-1biiE	O sistema consegue extrair os gostos de um Tweet.	A
RF-1biiiA	O sistema consegue extrair as *threads* nas quais o utilizador foi ativo (comentou, fez *upvote* ou *downvote*). Posteriormente o processo de extração de informação será o mesmo que o realizado pela API.	A
RF-1bivA	O sistema consegue extrair informação de mensagens trocadas em servidores IRC.	A
RF-1bivAI	O sistema consegue extrair o conteúdo das mensagens trocadas em servidores IRC.	A
RF-1bivAII	O sistema consegue extrair o autor das mensagens trocadas em servidores IRC.	A
RF-1bivAIII	O sistema consegue extrair o destinatário das mensagens trocadas em servidores IRC.	A
RF-1bivAIV	O sistema consegue extrair a data das mensagens trocadas em servidores IRC.	A
RF-1bivB	O sistema consegue extrair informação de um utilizador que se encontre no canal.	A
RF-1bivC	O sistema consegue extrair informação sobre os eventos que acontecem em determinado canal IRC .	A
RF-1bivCI	O sistema consegue extrair informação sobre os utilizadores que entram no canal e guardar informação sobre essa entrada.	A
RF-1bivCII	O sistema consegue guardar informação sobre essa saída.	A
RF-1bivCIII	O sistema consegue guardar informação sobre a expulsão de um utilizador do canal.	A
RF-1bivCIV	O sistema consegue guardar informação sobre a mudança de alcunha de um utilizador e mantém um registo de todas as alcunhas utilizadas pelo mesmo.	A
RF-2	O sistema utiliza *proxies* para fazer a distribuição da carga dos pedidos. A escolha do próximo *proxy* a utilizar será feita com base no algoritmo *Round Robin*, em que os *proxies* vão sendo escolhidos de forma circular.	A
RF-3aiA	O sistema consegue procurar endereços IPv4 nos dados.	A
RF-3aiB	O sistema consegue procurar endereços IPv6 nos dados.	A
RF-3aii	O sistema consegue procurar domínios nos dados.	A
RF-3aiii	O sistema consegue procurar endereços MAC nos dados.	A

ID	Descrição	Prioridade
RF-3aiv	O sistema consegue procurar *keywords* nos dados.	A
RF-3av	O sistema consegue procurar um conjunto de palavras (*list*) associadas a uma determinada palavra (*tag*).	A
RF-3b	O sistema consegue utilizar uma ferramenta externa para fazer análise textual dos dados e filtrar os mesmos de acordo co o configurado.	A
RF-4ai	O utilizador consegue criar um novo *bot*.	A
RF-4aii	O utilizador consegue configurar um bot.	A
RF-4aiiA	O utilizador consegue editar o nome de um *bot*.	A
RF-4aiiB	O utilizador consegue editar os alvos dentro de um *bot*.	A
RF-4aiiCI	O utilizador consegue editar os endereços IPv4 a procurar.	A
RF-4aiiCII	O utilizador consegue editar os endereços IPv6 a procurar.	A
RF-4aiiD	O utilizador consegue editar os domínios a procurar.	A
RF-4aiiE	O utilizador consegue editar os endereços MAC a procurar.	A
RF-4aiiF	O utilizador consegue editar *keywords* a procurar.	A
RF-4aiiG	O utilizador consegue editar os pares *tag*:*list* a procurar.	A
RF-4aiiH	O utilizador consegue editar os parâmetros de análise textual.	A
RF-4aiiI	O utilizador consegue editar o número de páginas a extrair de uma determinada fonte.	
RF-4aiiJ	O utilizador consegue editar as credenciais de acesso à API.	A
RF-4aiiK	O utilizador consegue editar as credenciais de login numa fonte.	A
RF-4aiiL	O utilizador consegue editar os campos para a procura de expressões regulares.	A
RF-4aiiM	O utilizador consegue editar quais os campos que serão extraídos de uma determinada fonte.	A
RF-4aiii	O utilizador consegue eliminar um *bot*.	A
RF-4aiv	O utilizador consegue adicionar um novo *proxy*.	A
RF-4av	O utilizador consegue remover um *proxy*.	A
RF-4biA	O utilizador consegue visualizar os dados de um evento apresentados em formato JSON.	B
RF-4biB	O utilizador consegue visualizar os dados de um evento apresentados de forma semelhante a um*post* no Facebook.	B
RF-4biC	O utilizador consegue visualizar os dados de um evento apresentados numa tabela (exclusivamente para expressões regulares e resultados da análise de texto).	B
RF-4bii	O utilizador consegue classificadas manualmente um evento.	B
RF-4biii	O utilizador consegue eliminar um evento.	B

ID	Descrição	Prioridade
RF-4ci	O utilizador deverá ser capaz de visualizar as ligações entre as várias entidades presentes na base de dados. Como já foi referido, por entidade entenda-se tudo o que possuir um identificador único na fonte do qual foi extraído (geralmente será o nome de utilizador ou id, podendo possuir ambos).	B
RF-4ciA	O utilizador deverá ser capaz centrar um nodo fazendo clique duplo sobre o mesmo.	B
RF-4ciB	O utilizador deverá ser capaz de realçar um nodo passando com o ponteiro do rato sobre o mesmo. Um *tooltip* com informação sobre o mesmo. Ao mesmo tempo que um nodo é realçado, os nodos vizinhos também o são, bem como as ligações entre estes e o nodo sobre o qual se encontra o ponteiro.	B
RF-4ciC	O utilizador deverá ser capaz de realçar as ligações sobre as quais coloca o ponteiro rato. Uma *tooltip* com informação sobre as mesma irá aparecer.	B
RF-4ciD	O utilizador deverá ser capaz de fazer zoom do grafo utilizando a roda do rato.	B
RF-4ciE	O utilizador deverá ser capaz de arrastar todo o grafo ou apenas só um nodo.	B
RF-4ciF	O utilizador deverá ser capaz de pesquisar um nodo.	B
RF-4ciG	O utilizador deverá ser capaz de definir a profundidade do grafo.	B
RF-4ciH	O utilizador deverá ser capaz de definir o intervalo temporal a ser usado na criação do grafo.	B
RF-4ciI	O utilizador deverá ser capaz de usar um menu de contexto para expandir o grafo a a partir de um nodo, mantendo os existentes.	B
RF-4ciF	O utilizador deverá ser capaz de usar um menu de contexto para expandir o grafo a a partir de um nodo, removendo os não conectados.	B
RF-4cii	O utilizador deverá ser capaz de adicionar um conjunto de *widgets* à sua *dashboard*.	B
RF-4ciiA	O utilizador deverá ser capaz de definir o tipo de estatística que pretende para o seu *widget*.	B
RF-4ciiB	O utilizador deverá ser capaz de definir o campo do documento a ser usado para o cálculo da estatística.	B
RF-4ciii	O utilizador deverá ser capaz de visualizar a informação por meio de uma tabela.	B
RF-4ciiiA	O utilizador deverá ser capaz de escolher as colunas que pretende para a sua tabela.	B
RF-4ciiiB	O utilizador deverá ser capaz de pesquisar as colunas no menu de escolha.	B

ID	Descrição	Prioridade
RF-4ciiiC	O utilizador deverá ser capaz de visualizar em detalhe cada evento. Dentro dos tipos de visualização encontram-se: JSON, tabelas (exclusivo para as expressões regulares e resultados da análise de texto) e apresentação dos dados de forma semelhante a um *post* no Facebook	B
RF-5	O modelo de dados deverá ser expandido para ir de encontro aos novos tipos de dados extraídos.	A
RF-6a	O estagiário deverá dar suporte a dados do tipo *lista* no modelo de dados.	A
RF-6b	O estagiário deverá dar suporte a dados do tipo *lista* aos *widgets* utilizados no Portolan.	B
RF-6c	O estagiário deverá dar suporte a dados do tipo *lista* à interface de exploração da base de dados através de uma tabela.	B
RF-6d	O estagiário deverá dar suporte a dados do tipo *lista* à interface de exploração da base de dados através de uma matriz.	B

A.3 Tabela Implementação Requisitos Funcionais

Esta secção serve para apresentar os requisitos funcionais implementados e não implementados e os resultados dos testes de aceitação. De referir que dada a extensão da lista de requisitos o estagiário optou por, em alguns casos, agrupar os mesmos.

ID Requisito	Implementado	Aceitação
RF-1ai	✓	✓
RF-1aiG	✗	✓
RF-1aii	✓	✓
RF-1aiii	✓	✓
RF-1aiv	✓	✓
RF-1bi	✓	✓
RF-1biAVII	✗	✓
RF-1bii	✓	✓
RF-1biii	✓	✓
RF-3a	✓	✓
RF-3b	✓	✓
RF-4a	✓	✓
RF-4b	✓	✓
RF-4ci	✓	✓
RF-4cii	✓	✓
RF-4ciii	✓	✓
RF-5	✓	✓
RF-6a	✓	✓
RF-6b	✓	✓
RF-6c	✓	✓
RF-6d	✓	✓

Tabela A.2: Tabela Com requisitos Implementado e Aceites

A.4 Testes Requisitos Funcionais

Dada o elavado número de funcionalidades a implementar os testes foram agrupados por macro-tarefa, podendo nalguns casos serem apresentadas as micro-tarefas em separado.

A.4.1 RF-1ai Extrair Informação pela API do Facebook

ID Teste: 01
ID Requisito: RF-1ai Extrair Informação pela API do Facebook
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: O ficheiro de configuração tem de ser válido (bem estruturado em formato JSON) e são necessárias credenciais e alvos válidos
Esperado: O *bot* extrai a informação e processa-a de acordo com o modelo de dados, sem perda de informação
Observado: A informação final está de acordo com o modelo de dados
Resultado: ✓
Observações: Nenhuma

ID Teste: 02
ID Requisito: RF-1ai Extrair Informação pela API do Facebook
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: O ficheiro de configuração tem de ser válido (bem estruturado em formato JSON) e possui um alvo inválido
Esperado: O *bot* extrai a informação e analisa a resposta, verificando que ocorreu um erro. Faz log da mensagem de erro e continua a execução.
Observado: O *bot* extraiu a informação e fez log da mensagem de erro, continuando a execução no alvo seguinte.
Resultado: ✓
Observações: Nenhuma

ID Teste: 03
ID Requisito: RF-1ai Extrair Informação pela API do Facebook
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: O ficheiro de configuração tem de ser válido (bem estruturado em formato JSON) mas as credenciais são inválidas
Esperado: O *bot* extrai a informação e analisa a resposta, verificando que ocorreu um erro. Faz log da mensagem de erro e continua a execução para o próximo alvo.
Observado: O *bot* extraiu a informação e fez log da mensagem de erro, continuando a execução no alvo seguinte.
Resultado: ✓
Observações: Nenhuma

Nota: Dada a natureza repetitiva da documentação dos testes aos requisitos funcionais, o estagiário optou por não incluir os restantes. Contudo, foram realizados pelo menos estes quatro testes para cada fonte.

ID Teste: 04
ID Requisito: RF-1ai Extrair Informação pela API do Facebook
Tipo Teste: Teste de Integração
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: O ficheiro de configuração tem de ser válido (bem estruturado em formato JSON) e são necessárias credenciais e alvos válidos. MongoDB, Redis ou Logger configurados no *bot*. A informação já foi extraída e processada de acordo com o modelo de dados.
Esperado: Cria Evento e envia para os vários *outputs*.
Observado: O Evento foi criado e enviado para os vários *outputs* sem perda de informação.
Resultado: ✓
Observações: No primeiro teste o Evento foi criado, mas não foi possível popular o mesmo porque a função de validação do modelo de dados não estava preparada para lidar com listas. Dada a natureza crítica da validação do conteúdo procedeu-se de imediato à resolução do problema.

A.4.2 RF-1bi Extrair informação do Facebook via crawling

ID Teste: 05
ID Requisito: RF-1bi Extrair informação do Facebook via crawling
Tipo Teste: Teste de Unidade
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: *Bot* fez login com sucesso. As configurações definem a extração de 3 páginas.
Esperado: *Bot* extrai todas as páginas com sucesso.
Observado: *Bot* extrai todas as páginas com sucesso.
Resultado: ✓
Observações: o primeiro teste falhou. A causa deste erro prende-se com o facto do XPath necessário para a primeira página ser diferente do para as restantes. Procedeu-se à alteração de imediato.

A.4.3 RF-1bivA Extrair informação sobre mensagens trocadas

ID Teste: 06
ID Requisito: RF-1bivA Extrair informação sobre mensagens trocadas
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: O ficheiro de configuração tem de ser válido (bem estruturado em formato JSON) e alvos válidos.
Esperado: Liga-se com sucesso.
Observado: O *bot* ligou-se com sucesso.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 07
ID Requisito: RF-1bivA Extrair informação sobre mensagens trocadas
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: *Bot* encontra-se no canal e um utilizador escreve uma mensagem.
Esperado: Deteta mensagem. Processa a mesma retirando conteúdo, autor, data, destinatário (caso seja uma mensagem direccionada).
Observado: Detetou mensagem e extraiu o conteúdo corretamente.
Resultado: ✓
Observações: Nenhuma.

A.4.4 RF-1bivB Extrair informação sobre utilizador

ID Teste: 08
ID Requisito: RF-1bivB Extrair informação sobre utilizador
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: textitBot encontra-se no canal.
Esperado: Sempre que um utilizador é ativo (liga-se, sai, escreve mensagem, muda de nome, entre outros) o *bot* envia o comando *whois* e guarda essa informação.
Observado: O *bot* envia o comando *whois*, recebe informação de volta e guarda com sucesso.
Resultado: ✓
Observações: O primeiro teste falhou. O problema aqui é que quando se envia um comando é necessário definir uma função de retorno para quando o servidor responder, o que na primeira implementação não existia. Procedeu-se de imediato à resolução.

A.4.5 RF-1bivC Extrair informação sobre eventos

ID Teste: 09
ID Requisito: RF-1bivB Extrair informação sobre utilizador
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: textitBot encontra-se no canal.
Esperado: Sempre que ocorre um evento no canal (utilizador liga-se, sai, é expulso, muda de alcunha) o *bot* guarda essa informação.
Observado: O *bot* recebe os eventos e guarda a informação sobre os mesmos.
Resultado: ✓
Observações: Nenhuma.

A.4.6 RF-3a Analisar Padrões

ID Teste: 10
ID Requisito: RF-3a Analisar Padrões
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: *Bot* encontra-se definido com um conjunto de expressões regulares especificamente escolhidas para os dados de entrada em utilização.
Esperado: *Bot* deteta as expressões regulares esperadas e guarda os campos onde as mesma foram detetadas.
Observado: *Bot* detetou com sucesso as expressões e guardou os campos onde foram detetadas corretamente.
Resultado: ✓
Observações: Nenhuma.

A.4.7 RF-3b Analisar Texto

ID Teste: 11
ID Requisito: RF-3b Analisar Texto
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: *Bot* encontra-se definido com credenciais de acesso inválidas.
Esperado: *Bot* recebe a mensagem de erro da API, faz log e não procede com o normal processamento da informação.
Observado: *Bot* apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 12
ID Requisito: RF-3b Analisar Texto
Tipo Teste: Teste de Sistema
Ação: O utilizador coloca o *bot* em execução
Pré-requisitos: *Bot* encontra-se definido com credenciais de acesso válidas.
Esperado: *Bot* procede com o normal processamento da informação.
Observado: *Bot* apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.8 RF-4aii Configurar Bot

ID Teste: 13
ID Requisito: RF-4aii Configurar Bot
Tipo Teste: Teste de Sistema
Ação: O utilizador criou novo *bot*.
Pré-requisitos: *Template* para o *bot* válido.
Esperado: Interface gerada corretamente.
Observado: *Bot* apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 14
ID Requisito: RF-4aii Configurar Bot
Tipo Teste: Teste de Sistema
Ação: O utilizador carregou em "guardar".
Pré-requisitos: Campo necessário não preenchido.
Esperado: Utilizador recebe mensagem de aviso. Campo não preenchido fica com contornos a vermelho.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 15
ID Requisito: RF-4aii Configurar Bot
Tipo Teste: Teste de Sistema
Ação: O utilizador carregou em "guardar".
Pré-requisitos: Campos necessários preenchidos.
Esperado: Utilizador guarda *bot* com sucesso.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 16
ID Requisito: RF-4aiiB Editar Alvos
Tipo Teste: Teste de Aceitação
Ação: O utilizador criou novo *bot*.
Pré-requisitos: *Template* para o *bot* válido.
Esperado: Orientador valida requisito implementado.
Observado: Orientador valida requisito implementado.
Resultado: ✓
Observações: O primeiro teste falhou. Inicialmente não ficou claro que um *bot* deveria estar preparado para extrair informação de vários alvos. O requisito foi alterado para ir de encontro ao pretendido.

A.4.9 RF-4bi Apresentar Dados

ID Teste: 17
ID Requisito: RF-4bi Apresentar Dados
Tipo Teste: Teste de Sistema
Ação: O utilizador procurou um novo evento.
Pré-requisitos: Base de dados devolveu evento com sucesso.
Esperado: As várias opções de visualização são mostradas e funcionam corretamente.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.10 RF-4bii Classificar Manualmente Evento

ID Teste: 18
ID Requisito: RF-4bii Classificar Manualmente Evento
Tipo Teste: Teste de Sistema
Ação: O utilizador clicou no botão *Submit*.
Pré-requisitos: Utilizador procurou um novo evento com sucesso e clicou no botão *Classify* abrindo um *modal* para colocar a classificação ou escolher uma já existente.
Esperado: Classificado com sucesso. Interface mostra mensagem de sucesso.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

ID Teste: 19
ID Requisito: RF-4bii Classificar Manualmente Evento
Tipo Teste: Teste de Sistema
Ação: O utilizador clicou no botão *Submit*.
Pré-requisitos: Utilizador clicou no botão *Classify* abrindo um modal para colocar a classificação ou escolher uma já existente.
Esperado: Interface mostra mensagem de erro.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.11 RF-4biii Eliminar evento

ID Teste: 20
ID Requisito: RF-4bii Classificar Manualmente Evento
Tipo Teste: Teste de Sistema
Ação: O utilizador clicou no botão *Delete*.
Pré-requisitos: Utilizador procurou um novo evento com sucesso.
Esperado: Eliminado com sucesso. Interface mostra mensagem de sucesso.
Observado: A aplicação apresentou o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.12 RF-4b Interface para classificação de dados

ID Teste: 21
ID Requisito: RF-4b Interface para classificação de dados
Tipo Teste: Teste de Aceitação
Ação:
Pré-requisitos: Base de dados não se encontra vazia.
Esperado: Orientador valida requisito implementado.
Observado: Orientador valida requisito implementado.
Resultado: ✓
Observações: Nenhuma.

A.4.13 RF-4ci Grafo

ID Teste: 22
ID Requisito: RF-4ci Grafo
Tipo Teste: Teste de Aceitação
Ação:
Pré-requisitos: Grafo Existe.
Esperado: Orientador valida requisito implementado.
Observado: Orientador valida requisito implementado.
Resultado: ✓
Observações: O *design* e funcionalidades iniciais foram aceites mas, no âmbito geral, o grafo carecia de mais algumas funcionalidades. As novas funcionalidades adicionadas foram: definir profundidade do grafo e definir intervalo temporal para gerar o gráfico. A funcionalidade de pesquisa de um nodo também sofreu alterações: passou a ser usada para definir o nodo a ser usado como ponto de partida na criação do grafo e não para pesquisar um nodo no grafo existente. As mesmas foram prontamente implementadas.

A.4.14 RF-4cii Widgets

ID Teste: 23
ID Requisito: RF-4cii Widgets
Tipo Teste: Teste de Sistema
Ação: Utilizador clica no botão *Apply*
Pré-requisitos: Existe informação na Base de Dados e os parâmetros encontram-se bem configurados.
Esperado: *Widget* apresenta estatísticas corretas.
Observado: *Widget* apresenta estatísticas corretas.
Resultado: ✓
Observações: O primeiro teste falhou. O widget não apresentava as estatísticas corretas pois as *queries* não suportavam listas.

A.4.15 RF-4ciiiA Escolher Colunas

ID Teste: 24
ID Requisito: RF-4ciii Escolher Colunas
Tipo Teste: Teste de Sistema
Ação: Utilizador clica no botão *Search*
Pré-requisitos: A *query* à base de dados retorna resultados.
Esperado: Apenas são apresentadas as colunas selecionadas.
Observado: A aplicação apresenta o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.16 RF-4ciiiB Pesquisar Colunas

ID Teste: 25
ID Requisito: RF-4ciiiB Pesquisar Colunas
Tipo Teste: Teste de Sistema
Ação: Utilizador escreve na caixa de procura
Pré-requisitos: A palavra escrita pelo utilizador está presente nas colunas.
Esperado: As colunas são filtradas corretamente.
Observado: A aplicação apresenta o comportamento esperado.
Resultado: ✓
Observações: Nenhuma.

A.4.17 RF-4ciiiC Visualizar detalhadamente evento

ID Teste: 26
ID Requisito: RF-4ciiiC Visualizar detalhadamente evento.
Tipo Teste: Teste de Sistema.
Ação: Utilizador clica para visualizar dados do evento.
Pré-requisitos: A *query* à base de dados devolveu eventos.
Esperado: Todas os métodos de visualização funcionaram corretamente.
Observado: A aplicação apresenta o comportamento esperado.
Resultado: ✓
Observações: O estagiário optou por não se alongar pois este requisito foi implementado usando código desenvolvido em requisitos anteriores.

A.4.18 RF-4ciii Tabela

ID Teste: 27
ID Requisito: RF-4ciii Tabela.
Tipo Teste: Teste de Aceitação.
Ação: Orientador analisa todas as funcionalidades.
Pré-requisitos: A *query* à base de dados devolveu eventos.
Esperado: O orientador valida a implementação.
Observado: O orientador validou a implementação.
Resultado: ✓
Observações: Nenhuma.

Apêndice B

Modelo de Dados

neste capítulo será apresentada um porção do modelos de dados do Portolan. Porção essa que representa a extensão total dos dados retirados pelos *bots* implementados. Dado que o MondoDB não força nenhum esquema e a informação é guardada em formato JSON, também o modelo de dados se apresenta neste formato.

```
{
  ...
  "social": {
    "info": {
      "id": {
        "type": "String",
        "required": false,
        "label": "Information: Id"
      },
      "name": {
        "type": "String",
        "required": false,
        "label": "Information: Name"
      },
      "username": {
        "type": "String",
        "required": false,
        "label": "Information: Username"
      },
      "location": {
        "type": "String",
        "required": false,
        "label": "Information: Location"
      },
      "about": {
        "type": "String",
        "required": false,
        "label": "Information: About"
      },
      "followers_count": {
        "type": "Int",
        "required": false,
        "label": "Information: Followers Count"
```

```
    },
    "friends_count": {
      "type": "Int",
      "required": false,
      "label": "Information: Friends Count"
    },
    "listed_count": {
      "type": "Int",
      "required": false,
      "label": "Information: Listed Count"
    },
    "creation_date": {
      "type": "Datetime",
      "required": false,
      "label": "Information: Profile/Page Creation Date"
    },
    "favourites_count": {
      "type": "Int",
      "required": false,
      "label": "Information: Favorites Count"
    },
    "statuses_count": {
      "type": "Int",
      "required": false,
      "label": "Information: Statuses Count"
    }
  },
  "post": {
    "author": {
      "name": {
        "type": "String",
        "required": false,
        "label": "Post: Author: Name"
      },
      "id": {
        "type": "String",
        "required": false,
        "label": "Post: Author: Id"
      },
      "username": {
        "type": "String",
        "required": false,
        "label": "Post: Author: Username"
      }
    },
    "tags": {
      "tag_type": {
        "type": "String",
        "required": false,
      },
      "id": {
        "type": "String",
        "required": false,
      },
```

```
      "name": {
        "type": "String",
        "required": false,
      },
      "username": {
        "type": "String",
        "required": false,
      },
      "type": "list"
    },
    "content_url": {
      "type": "list",
      "required": false,
    },
    "hashtags": {
      "type": "list",
      "required": false,
    },
    "comments": {
      "author": {
        "name": {
          "type": "String",
          "required": false,
        },
        "id": {
          "type": "String",
          "required": false,
        },
        "username": {
          "type": "String",
          "required": false,
        }
      },
      "tags": {
        "tag_type": {
          "type": "String",
          "required": false,
        },
        "id": {
          "type": "String",
          "required": false,
        },
        "name": {
          "type": "String",
          "required": false,
        },
        "username": {
          "type": "String",
          "required": false,
        },
        "type": "list"
      },
      "content_url": {
        "type": "String",
```

```
      "required": false,
    },
    "hashtags": {
      "type": "String",
      "required": false,
    },
    "comments": {
      "author": {
        "name": {
          "type": "String",
          "required": false,
        },
        "id": {
          "type": "String",
          "required": false,
        },
        "username": {
          "type": "String",
          "required": false,
        }
      },
      "tags": {
        "tag_type": {
          "type": "String",
          "required": false,
        },
        "id": {
          "type": "String",
          "required": false,
        },
        "name": {
          "type": "String",
          "required": false,
        },
        "username": {
          "type": "String",
          "required": false,
        },
        "type": "list"
      },
      "content_url": {
        "type": "String",
        "required": false,
      },
      "hashtags": {
        "type": "String",
        "required": false,
      },
      "content": {
        "type": "String",
        "required": false,
      },
      "message": {
        "type": "String",
```

```
      "required": false,
    },
    "likes": {
      "name": {
        "type": "String",
        "required": false,
      },
      "id": {
        "type": "String",
        "required": false,
      },
      "username": {
        "type": "String",
        "required": false,
      },
      "type": "list"
    },
    "shares": {
      "type": "Int",
      "required": false,
    },
    "id": {
      "type": "String",
      "required": false,
    },
    "story": {
      "type": "String",
      "required": false,
    },
    "story_tags": {
      "tag_type": {
        "type": "String",
        "required": false,
      },
      "id": {
        "type": "String",
        "required": false,
      },
      "name": {
        "type": "String",
        "required": false,
      },
      "username": {
        "type": "String",
        "required": false,
      },
      "type": "list"
    },
    "time": {
      "source": {
        "type": "DateTime",
      },
      "updated": {
        "type": "DateTime",
```

```
      }
    },
    "type": "list"
  },
  "content": {
    "type": "String",
    "required": false,
  },
  "message": {
    "type": "String",
    "required": false,
  },
  "likes": {
    "name": {
      "type": "String",
      "required": false,
    },
    "id": {
      "type": "String",
      "required": false,
    },
    "username": {
      "type": "String",
      "required": false,
    },
    "type": "list"
  },
  "type": "list",
  "shares": {
    "type": "Int",
    "required": false,
  },
  "id": {
    "type": "String",
    "required": false,
  },
  "story": {
    "type": "String",
    "required": false,
  },
  "story_tags": {
    "tag_type": {
      "type": "String",
      "required": false,
    },
    "id": {
      "type": "String",
      "required": false,
    },
    "name": {
      "type": "String",
      "required": false,
    },
    "username": {
```

```
            "type": "String",
            "required": false,
          },
          "type": "list"
        },
        "time": {
          "source": {
            "type": "DateTime",
          },
          "updated": {
            "type": "DateTime",
          }
        },
        "dislikes_count": {
          "type": "Int",
        },
        "likes_count": {
          "type": "Int",
        }
      },
      "content": {
        "type": "String",
        "required": false,
        "label": "Post: Content"
      },
      "message": {
        "type": "String",
        "required": false,
        "label": "Post: Message"
      },
      "likes": {
        "name": {
          "type": "String",
          "required": false,
        },
        "id": {
          "type": "String",
          "required": false,
        },
        "username": {
          "type": "String",
          "required": false,
        },
        "type": "list"
      },
      "shares_count": {
        "type": "String",
        "required": false,
        "label": "Post: Shares Count"
      },
      "shares": {
        "id": {
          "type": "String",
        },
```

```
    "author": {
      "name": {
        "type": "String",
        "required": false,
      },
      "id": {
        "type": "String",
        "required": false,
      },
      "username": {
        "type": "String",
        "required": false,
      }
    },
    "type": "list"
  },
  "id": {
    "type": "String",
    "required": false,
    "label": "Post: ID"
  },
  "story": {
    "type": "String",
    "required": false,
    "label": "Post: Story"
  },
  "story_tags": {
    "tag_type": {
      "type": "String",
      "required": false,
    },
    "id": {
      "type": "String",
      "required": false,
    },
    "name": {
      "type": "String",
      "required": false,
    },
    "username": {
      "type": "String",
      "required": false,
    },
    "type": "list"
  },
  "likes_count": {
    "type": "String",
    "label": "Post: Likes Count"
  },
  "dislikes_count": {
    "type": "Dislikes Count",
    "label": "Post: Dislikes Count"
  },
  "in_reply_to_status_id": {
```

```
        "type": "String",
        "label": "Post: In Reply To Status ID"
      },
      "in_reply_to_user_id": {
        "type": "String",
        "label": "Post: In Reply To User ID"
      },
      "quoted_status_id": {
        "type": "String",
        "label": "Post: Quoted Status ID"
      },
      "domain": {
        "type": "String",
        "label": "Post: Domain"
      },
      "url": {
        "type": "String",
        "label": "Post: URL"
      },
      "title": {
        "type": "String",
        "label": "Post: Title"
      },
      "private_message": {
        "type": "Boolean",
        "label": "Post: Private Message"
      },
      "directed_message": {
        "type": "Boolean",
        "label": "Post: Directed Message"
      }
    },
    "likes": {
      "type": "list",
      "name": {
        "type": "String",
      },
      "id": {
        "type": "String",
      },
      "username": {
        "type": "String",
      }
    },
    "friends": {
      "type": "list",
      "name": {
        "type": "String",
      },
      "id": {
        "type": "String",
      },
      "username": {
        "type": "String",
```

```
    }
  },
  "followers": {
    "type": "list",
    "id": {
      "type": "String",
    },
    "name": {
      "type": "String",
    },
    "username": {
      "type": "String",
    }
  },
  "following": {
    "type": "list",
    "id": {
      "type": "String",
    },
    "name": {
      "type": "String",
    },
    "username": {
      "type": "String",
    }
  },
  "target": {
    "id": {
      "type": "String",
      "label": "Post: Target: ID"
    },
    "name": {
      "type": "String",
      "label": "Post: Target: Name"
    },
    "username": {
      "type": "String",
      "label": "Post: Target: Username"
    },
    "subreddit": {
      "type": "String",
      "label": "Post: Target: Subreddit"
    },
    "board": {
      "type": "String",
      "label": "Post: Target: Board"
    },
    "thread": {
      "type": "String",
      "label": "Post: Target: Thread"
    },
    "server": {
      "type": "String",
      "label": "Post: Target: Server"
```

```
      },
      "channel": {
        "type": "String",
        "label": "Post: Target: Channel"
      }
    },
    "regex_matches": {
      "type": "list",
    },
    "aggregated_regex_matches": {
      "type": "list",
      "text": {
        "type": "String",
      },
      "count": {
        "type": "Int",
      }
    },
    "irc": {
      "nickname_change": {
        "old": {
          "type": "String",
          "label": "IRC: Nickname Change: Old Nickname"
        },
        "new": {
          "type": "String",
          "label": "IRC: Nickname Change: New Nickname"
        }
      },
      "user_joined": {
        "type": "String",
        "label": "IRC: User Joined"
      },
      "user_left": {
        "type": "String",
        "label": "IRC: User Left"
      },
      "user_quit": {
        "type": "String",
        "label": "IRC: User Quit"
      },
      "user_kicked": {
        "kicker": {
          "type": "String",
          "label": "IRC: User Kicked: Kicker"
        },
        "kickee": {
          "type": "String",
          "label": "IRC: User Kicked: Kickee"
        },
        "message": {
          "type": "String",
          "label": "IRC: User Kicked: Message"
        }
```

```
    },
    "user_info": {
      "something": {
        "type": "String",
        "label": "IRC: User Info: Something"
      },
      "host": {
        "type": "String",
        "label": "IRC: User Info: Host"
      },
      "user": {
        "type": "String",
        "label": "IRC: User Info: User"
      },
      "realname": {
        "type": "String",
        "label": "IRC: User Info: Real Name"
      },
      "known_nicknames": {
        "type": "list",
      },
      "last_nickname_in_use": {
        "type": "String",
        "label": "IRC: User Info: Last Nickname In Use"
      }
    },
    "action": {
      "type": "String",
    }
  },
  "ibm_alchemy_language": {
    "entities": {
      "type": "list",
      "relevance": {
        "type": "Float",
      },
      "count": {
        "type": "Float",
      },
      "_type": {
        "type": "String",
      },
      "disambiguated": {
        "website": {
          "type": "String",
        },
        "yago": {
          "type": "String",
        },
        "name": {
          "type": "String",
        },
        "freebase": {
          "type": "String",
```

```
        },
        "opencyc": {
          "type": "String",
        },
        "subType": {
          "type": "List",
        },
        "dbpedia": {
          "type": "String",
        },
        "ciaFactbook": {
          "type": "String",
        },
        "crunchbase": {
          "type": "String",
        },
        "geo": {
          "type": "String",
        },
        "geonames": {
          "type": "String",
        }
      },
      "text": {
        "type": "String",
      }
    },
    "taxonomy": {
      "type": "list",
      "score": {
        "type": "Float",
      },
      "label": {
        "type": "String",
      },
      "confident": {
        "type": "String",
      }
    },
    "keywords": {
      "type": "list",
      "relevance": {
        "type": "Float",
      },
      "text": {
        "type": "String",
      },
      "sentiment": {
        "type": "String",
      }
    },
    "concepts": {
      "type": "list",
      "relevance": {
```

```
      "type": "Float",
    },
    "text": {
      "type": "String",
    },
    "opencyc": {
      "type": "String",
    },
    "dgpedia": {
      "type": "String",
    },
    "freebase": {
      "type": "String",
    },
    "yago": {
      "type": "String",
    }
  },
  "docSentiment": {
    "mixed": {
      "type": "String",
      "label": "IBM Alchemy Language: Document Sentiment:
          ↪ Mixed"
    },
    "score": {
      "type": "Float",
      "label": "IBM Alchemy Language: Document Sentiment:
          ↪ Score"
    },
    "_type": {
      "type": "String",
      "label": "IBM Alchemy Language: Document Sentiment:
          ↪ Type"
    }
  },
  "docEmotions": {
    "anger": {
      "type": "Float",
      "label": "IBM Alchemy Language: Document Emotions:
          ↪ Anger"
    },
    "joy": {
      "type": "Float",
      "label": "IBM Alchemy Language: Document Emotions:
          ↪ Joy"
    },
    "fear": {
      "type": "Float",
      "label": "IBM Alchemy Language: Document Emotions:
          ↪ Fear"
    },
    "sadness": {
      "type": "Float",
      "label": "IBM Alchemy Language: Document Emotions:
```

```
                ↪ Sadness"
            },
            "disgust": {
              "type": "Float",
              "label": "IBM Alchemy Language: Document Emotions:
                ↪ Disgust"
            }
          },
          "count": {
            "type": "Int",
            "label": "IBM Alchemy Language: Number of Entries"
          }
        }
      }
      ...
}
```

Apêndice C

Interface Gráfica de Utilizador

Neste capítulo serão apresentadas as várias interfaces (ou alterações a interfaces já existentes) da solução implementada, desde a tabela à classificação manual e visualização do grafo e terminando na configuração de *bots*.

C.1 Tabelas

Nesta secção começa-se por se mostrar o resultado de uma *query* à base de dados disposto sob a forma de uma tabela com colunas previamente selecionadas (figura C.1). Importa referir que não foi o estagiário o autor deste elemento, no entanto definiu como apresentar o conteúdo nas células das colunas apresentadas (apenas para a coluna da IBM e das Expressões Regulares). Na tabela C.2 apresenta-se a forma de visualização do evento em JSON. Nas tabelas C.3, C.4, C.5 e C.6 apresenta-se informação devolvida pelo IBM *AlchemyLanguage*. Já nas tabelas C.7 e C.8 apresenta-se informação sobre as expressões regulares.

feed.id	time.observation	social.ibm_alchemy_language.count	social.regex_matches	Action
reddit_api	July 21, 2016 16:52:46	33	1089	
reddit_api	July 21, 2016 16:52:46	9	8	
reddit_api	July 21, 2016 16:52:48	25	20	
facebook	July 21, 2016 16:52:48	34	3	
facebook	July 21, 2016 16:52:48	15	28	
facebook	July 21, 2016 16:52:48	14	11	
facebook	July 21, 2016 16:52:48	19	18	
facebook	July 21, 2016 16:52:48	36	15	
reddit_api	July 21, 2016 16:52:49	2	4	
reddit_api	July 21, 2016 16:52:50	2	2	
4chan_api	July 21, 2016 16:52:50	8	1	
4chan_api	July 21, 2016 16:52:50	8	15	
reddit_api	July 21, 2016 16:52:51	18	22	
reddit_api	July 21, 2016 16:52:51	20	12	
4chan_api	July 21, 2016 16:52:51	20	2	
reddit_api	July 21, 2016 16:52:52	19	6	
reddit_api	July 21, 2016 16:52:53	25	4	
reddit_api	July 21, 2016 16:52:53	6	2	
reddit_api	July 21, 2016 16:52:53	2	2	
4chan_api	July 21, 2016 16:52:53	8	2	

Figura C.1: Interface de Exploração da Base de Dados: Tabela

Figura C.2: Interface de Exploração da Base de Dados: Tabela: Inspecionar Evento em Formato JSON

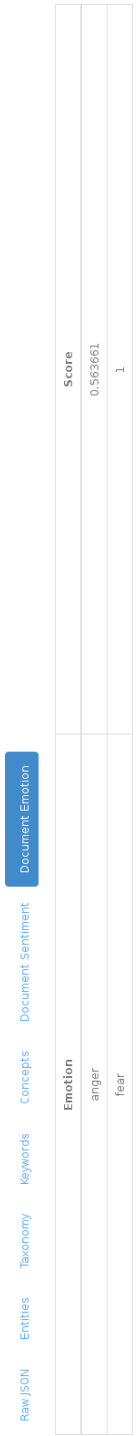

Figura C.3: Interface de Exploração da Base de Dados: Tabela: Inspecionar Resultados Do IBM Alchemy: Emoção do Documento

Figura C.4: Interface de Exploração da Base de Dados: Tabela: Inspecionar Resultados Do IBM Alchemy: Entidades 175

Raw JSON | Entities | Taxonomy | Keywords | Concepts | Document Sentiment | Document Emotion


```
{
  "count": 33,
  "taxonomy": [ ],
  "entities": [
    {
      "relevance": 0.848803,
      "count": 13,
      "type": "JobTitle",
      "text": "Security Engineer"
    },
    {
      "relevance": 0.555441,
      "count": 7,
      "type": "Country",
      "disambiguated": {
        "website": "http://www.usa.gov/",
        "yago": "http://yago-knowledge.org/resource/United_States",
        "name": "United States",
        "freebase": "http://rdf.freebase.com/ns/m.09c7w0",
        "opencyc": "http://sw.opencyc.org/concept/Mx4rvVikKpwpEbGdrcN5Y29ycA",
        "subType": [
          "Location",
          "Region",
          "AdministrativeDivision",
          "GovernmentalJurisdiction",
          "FilmEditor"
        ],
        "dbpedia": "http://dbpedia.org/resource/United_States",
        "ciaFactbook": "http://www4.wiwiss.fu-berlin.de/factbook/resource/United_States"
      },
      "text": "US"
    }
  ],
  "docEmotions": { },
  "concepts": [ ],
  "keywords": [ ]
}
```


Figura C.5: Interface de Exploração da Base de Dados: Tabela: Inspecionar Resultados Do IBM Alchemy: Resultados em JSON

Keyword	Relevance	Sentiment
security	0.915125	Unknown
security engineer	0.737764	Unknown
application security	0.66417	Unknown
network security	0.622063	Unknown
penetration testing	0.608346	Unknown
Information Security Engineer	0.58125	Unknown
Security Software Engineer	0.580304	Unknown
security testing	0.579114	Unknown
security best practices	0.573942	Unknown
security testing experience	0.573887	Unknown
common security vulnerabilities	0.571987	Unknown
security team	0.568287	Unknown
security posture	0.567672	Unknown
security testing efforts	0.559864	Unknown
information technology security	0.559333	Unknown
security consulting experience	0.558445	Unknown
security skills	0.556504	Unknown
verbal communication skills	0.555017	Unknown
software security	0.553627	Unknown
web security fundamentals	0.553028	Unknown
security advisory services	0.548714	Unknown
Head Information Security	0.548478	Unknown
Greenhouse security program	0.547235	Unknown
security devices	0.547095	Unknown
Network Security Analysis	0.5466	Unknown
information security leaders	0.54656	Unknown
Network security concepts	0.546188	Unknown
Network Security Engineer	0.545827	Unknown
Security Development Lifecycle	0.545813	Unknown
principal security consultants	0.545228	Unknown

Figura C.6: Interface de Exploração da Base de Dados: Tabela: Inspecionar Resultados Do IBM Alchemy: Palavras-Chave

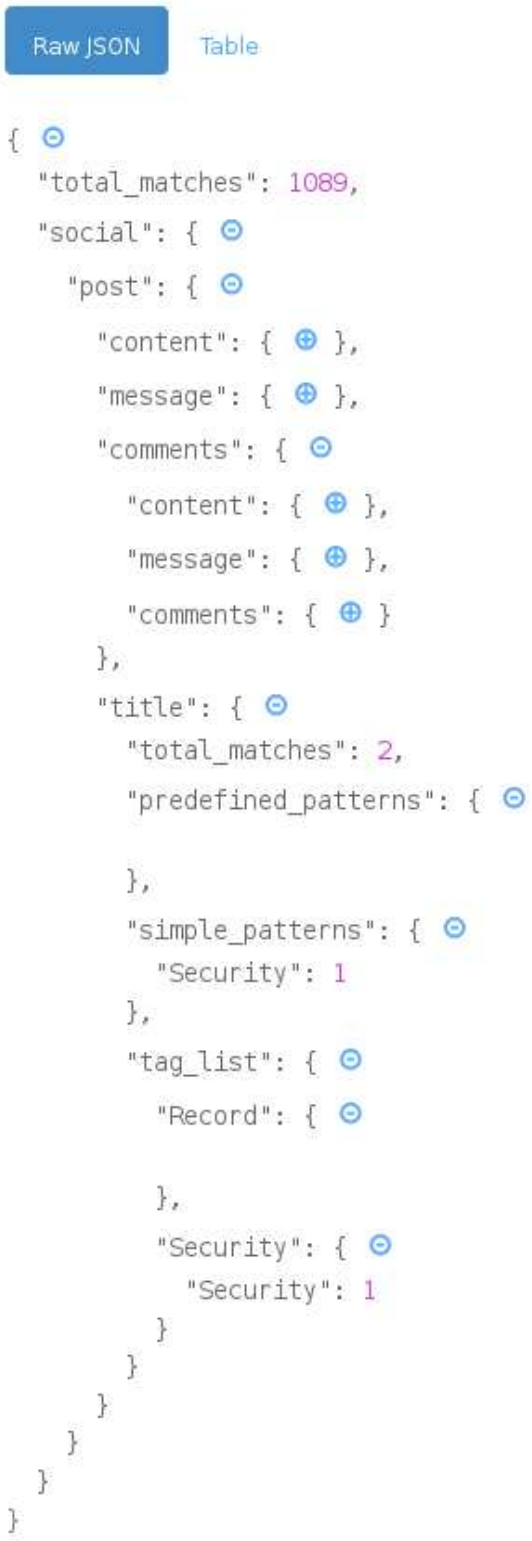

Figura C.7: Interface de Exploração da Base de Dados: Tabela: Inspecionar Expressões Regulares em JSON

Figura C.8: Interface de Exploração da Base de Dados: Tabela: Inspecionar Expressões Regulares em Tabela

C.2 Classificação Manual

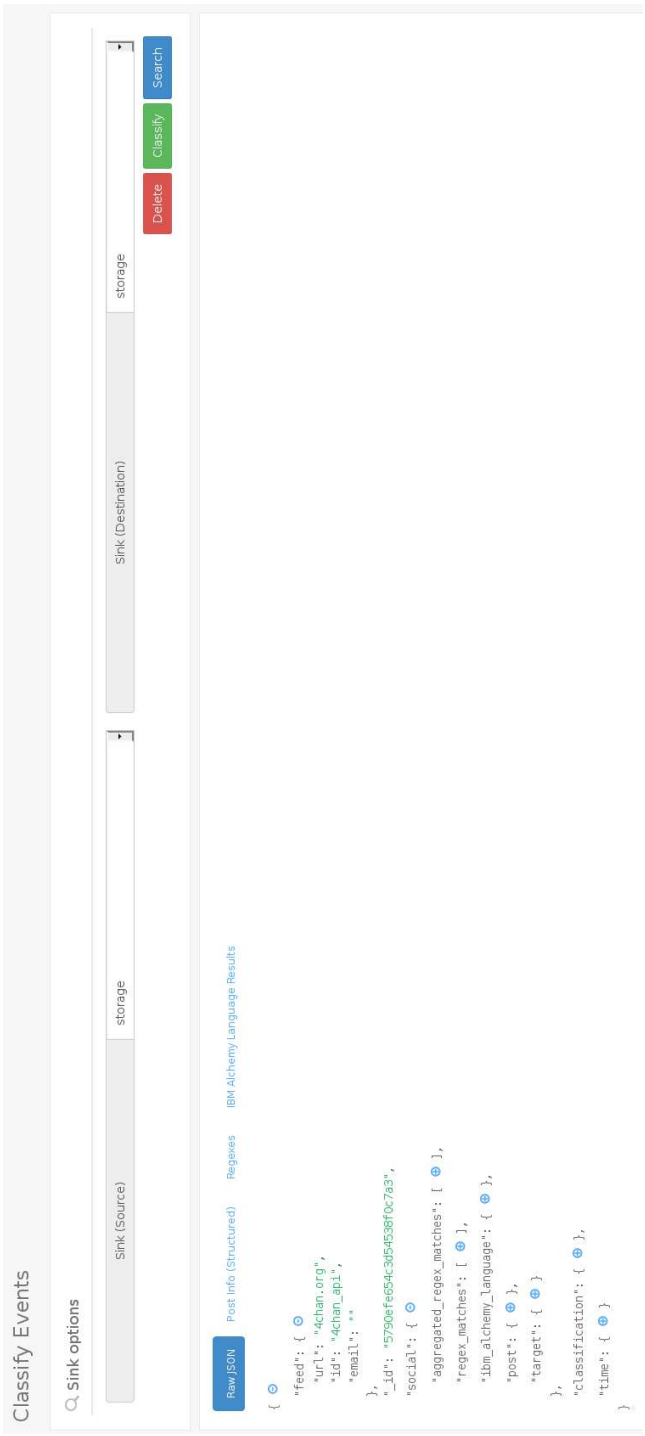

Figura C.9: Interface Classificação Manual

C.3 Visualização do Grafo

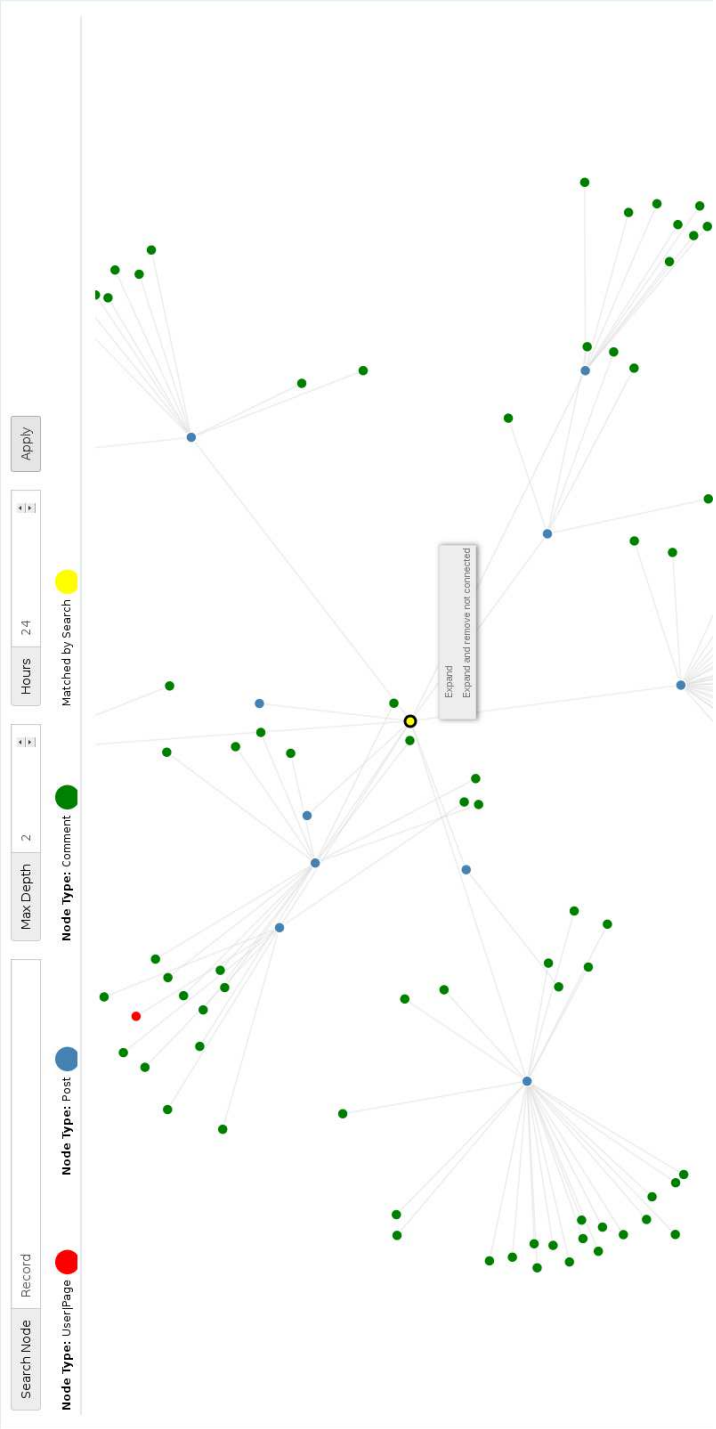

Figura C.10: Interface De Visualização do Grafo

C.4 Configuração dos Bots

Nesta secção irão ser apresentadas as interfaces gráficas dos *bots* e respetivos *templates*. Como foi mencionado na secção 5.3.6 as interfaces são geradas tendo como base um *template* no formato JSON.
Dada a natureza repetitiva dos *templates* e respetivas interfaces geradas não serão colocados exemplos para todos os *bots*, no entanto os que são apresentados contém todo o tipo de elementos usados nas interfaces dos restantes *bots*.

C.4.1 Facebook

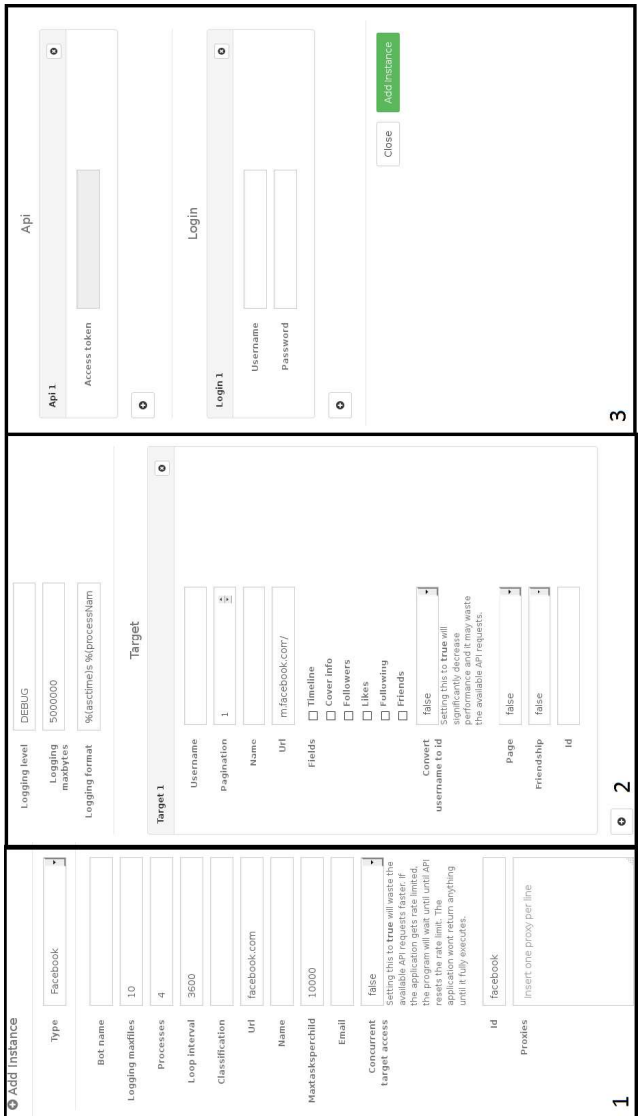

Figura C.11: Interface de Configuração do Bot do Facebook

```
{
  "logging_maxfiles":10,
  "processes":4,
  "loop_interval":3600,
  "target":[
    {
      "username":null,
      "pagination":1,
      "name":null,
      "url":"m.facebook.com/",
      "fields":{
        "cover_info":false,
        "timeline":false,
        "friends":false,
        "likes":false,
        "followers":false,
        "following":false
      },
      "id":null,
      "friendship":[
        false,
        true
      ],
      "page":[
        false,
        true
      ],
      "convert_username_to_id":[
        false,
        true
      ]
    }
  ],
  "classification":"",
  "url":"facebook.com",
  "logging_maxbytes":5000000,
  "maxtasksperchild":10000,
  "email":"",
  "api":[
    {
      "access_token":null
    }
  ],
  "login":[
    {
      "username":null,
      "password":null
    }
  ],
  "input":{

  },
  "id":"facebook",
```

```
  "output":[

  ],
  "logging_format":"%(asctime)s␣%(processName)-10s␣%(name)s␣%(
      ↪ levelname)-8s␣%(message)s",
  "proxies":{
    "hosts":[

    ],
    "pos":-1
  },
  "logging_level":"DEBUG",
  "name":"",
  "concurrent_target_access":[
    false,
    true
  ]
}
```

C.4.2 Bot Extrator de Expressões Regulares

```
{
  "logging_maxfiles":"10",
  "processes":"4",
  "loop_interval":"3600",
  "keep_content_in_pipeline_only_if_matched_something":[
    true,
    false
  ],
  "target":[
    {
      "simple_patterns":[
        ""
      ],
      "tag_list":{
        "Click to Edit":[
          "Click to Edit"
        ]
      },
      "selected_search_paths":[

      ],
      "node_name":"facebook",
      "mac_address":"^([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})$
          ↪ ",
      "as":"",
      "ipv4":"^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\\.)
          ↪ {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$",
      "ipv6":"(?<![:.\\w])(?:[A-Fa-f0-9]{1,4}:){7}[A-Fa-f0
          ↪ -9]{1,4}(?![:.\\w])",
      "domains":"^([a-zA-Z0-9]+(-[a-zA-Z0-9]+)*\\.)+[a-zA-Z
          ↪ ]{2,}$"
    }
  ],
  "classification":"",
  "maxtasksperchild":"10000",
  "logging_level":"DEBUG",
  "email":"",
  "logging_maxbytes":"5000000",
  "input":{

  },
  "output":[

  ],
  "logging_format":"%(asctime)s %(processName)-10s %(name)s %(
      ↪ levelname)-8s %(message)s",
  "id":"regex",
  "name":""
}
```

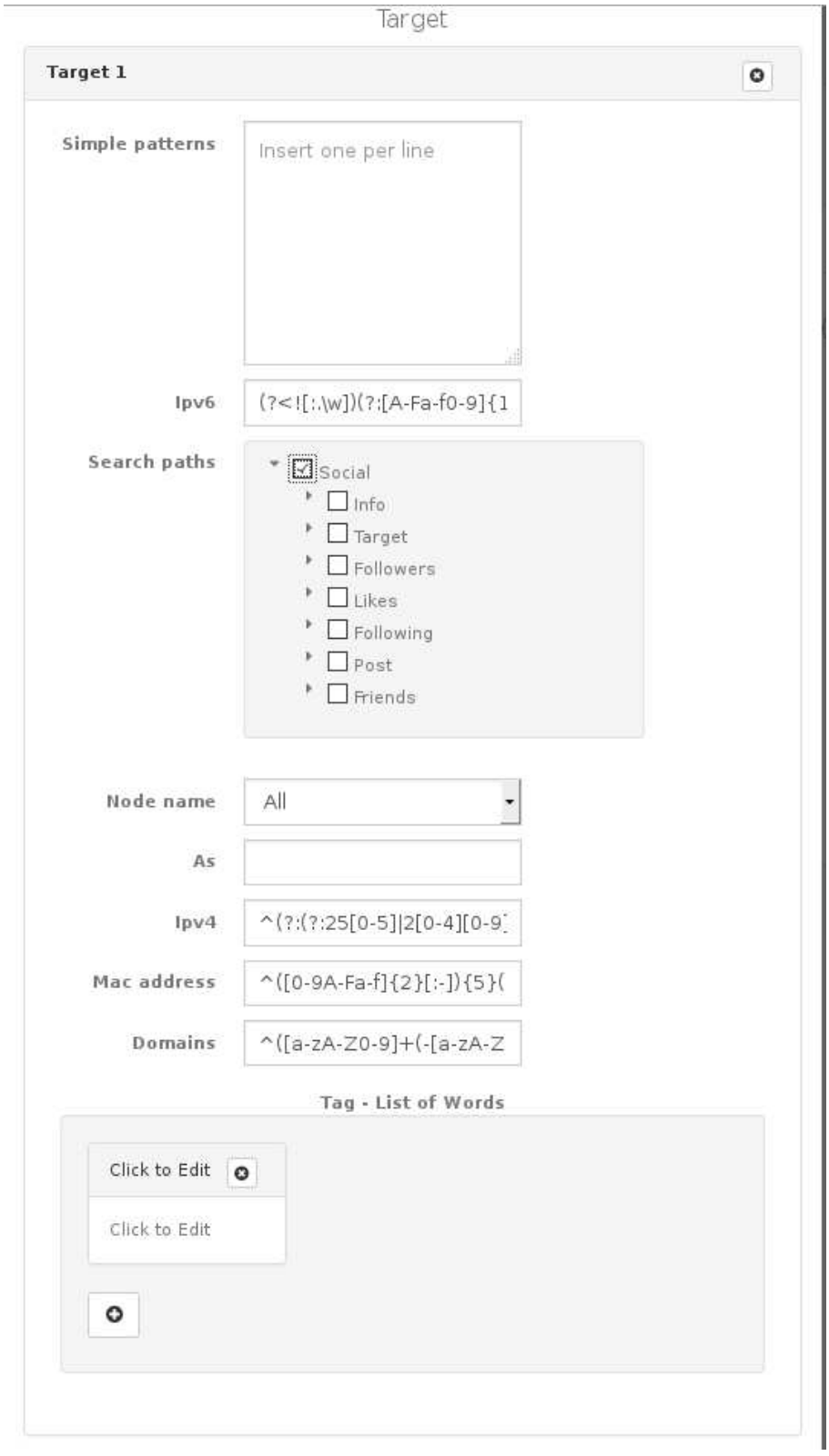

Figura C.12: Interface de Configuração do Bot Extrator de Expressões Regulares