Mestrado em Engenharia Informática
Estágio
Relatório Final

# Projeto EDUCA

Nuno Borges
nborges@student.dei.uc.pt

Orientador (FdU):
Nuno Francisco

Orientador (DEI):
Paulo Gomes

Data: 2/07/2013

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

# Resumo

O projeto EDUCA é uma plataforma informática para suportar um repositório de conteúdos multimédia para pesquisa, agregação e proteção dos conteúdos. Nesta plataforma é pretendido que seja feita a extração automática de informação, classificação e sumarização automática de documentos de texto.

A extração automática de informação corresponde a obter a informações contidas no ficheiro, como titulo, data de criação, autores, entre outros. A informação aqui descrita corresponde aos metadados. No entanto, no caso de documentos de texto a informação extraído contempla também o próprio texto.

A classificação automática de documentos de texto assenta na classificação dos documentos em várias categorias, permitindo depois ao utilizador pesquisar os documentos por categoria.

No caso da sumarização automática de texto tem como objetivo criar um resumo do documento. A existência de um resumo sobre um qualquer documento indica ao utilizador sobre o que é que o documento retrata.

A abordagem que é pretendida seguir na classificação automática de documentos é semi-supervisionada, tendo como expectativa tirar partido de análises anteriores, para melhorar as análises posteriores. No caso da sumarização de documentos a abordagem será superficial, ou seja, uma com base estatística.

O presente relatório pretende mostrar todo o trabalho desenvolvido pelo estagiário ao longo do estágio, para resolver os problemas de classificação e sumarização automática de documentos de texto.

# Palavras-chave

"Classificação automática de documentos", "Plataforma Web", "Reconhecimento de entidades mencionadas", "Repositório de conteúdos digitais", "Sumarização automática de documentos"

# Índice

# Lista das Figuras

# Lista dos Gráficos

# Lista das Tabelas

# Capítulo 1
# Introdução

O objetivo deste relatório é apresentar o trabalho do estágio, realizado no primeiro e segundo semestres. Faz parte da proposta de estágio "Projeto EDUCA – Classificação e Sumarização de Documentos", do ano letivo 2012/2013, para a disciplina de "Estágio/Dissertação" do Mestrado em Engenharia Informática (MEI) da Faculdade de Ciências e Tecnologia (FCT) da Universidade de Coimbra (UC).

O presente estágio foi orientado pelo Professor Paulo Gomes, docente do Departamento de Engenharia Informática (DEI) e pelo Eng.º Nuno Francisco, gestor de projetos na Flor de Utopia (FdU).

O projeto EDUCA tem por objetivo desenvolver uma plataforma informática para suportar um repositório de conteúdos digitais, onde seja possível agregar, pesquisar, publicar e proteger estes conteúdos. Nesta plataforma é esperado que seja possível proceder automaticamente à extração de informação, classificação e sumarização de documentos. Até este momento, parte disto já se encontra definido, designadamente no que respeita à extração de informação e reconhecimento de entidades mencionadas. Contudo, falta definir a parte classificação e sumarização automática de documentos texto.

O foco principal do estágio é melhorar o módulo de análise semântica do projeto EDUCA. As tarefas que devem ser feitas para melhorar este módulo são:

- Desenvolver a classificação automática de documentos de texto;
- Desenvolver a sumarização automática de documentos de texto;
- Testar as funcionalidades implementadas.

A abordagem que vai ser seguida para o desenvolvimento da tarefa de classificação de documentos é a semi-supervisionada, ou seja, a utilização de documentos previamente classificados para que seja possível classificar novos documentos, com base nas classificações anteriores. No caso da sumarização a abordagem a ser seguida é uma abordagem superficial, ou seja, apenas utiliza valores estatísticos para sumarizar automaticamente os documentos de texto. No final deste projeto é esperado que estas tarefas estejam funcionais. Embora possam não ficar com uma qualidade ótima, pois o tempo não permite, mas que cumpram as tarefas de forma razoável.

## 1.1. Estrutura do relatório

Este relatório encontra-se organizado em 9 capítulos:

- Capítulo 1 (Introdução) – é feito a apresentação do projeto;
- Capítulo 2 (Estado de arte) – pretende mostrar o estudo previamente feito pelo estagiário para colmatar a falta de conhecimento das áreas que o estágio aborda.
- Capítulo 3 (Avaliação do Projeto EDUCA) – visa validar algumas ferramentas utilizadas no projeto por comparação com outras.
- Capítulo 4 (Análise de requisitos, arquitetura e design) – apresenta os requisitos e a arquitetura que assenta a plataforma Web.
- Capítulo 5 (Implementação e testes) – descreve de forma sumária como foi feita a implementação dos requisitos e como foi testado o sistema.

- Capítulo 6 (Teste e experimentação) – mostra os testes que foram realizados para mostrar a qualidade e performance da implementação da classificação e sumarização automática de documentos de texto.
- Capítulo 7 (Plano de Trabalho) – onde se encontra o planeamento feito para o projeto.
- Capítulo 8 (Conclusão) – é feito uma reflexão final sobre o estágio e o produto que foi implementado.
- Capitulo 9 (Bibliografia) – onde se encontram todas as referências utilizadas pelo estagiário no relatório.

# Capítulo 2
# Estado da Arte

Ao longo deste capítulo, são retratados diversos conceitos importantes para o projeto EDUCA. Aborda-se a web semântica (capítulo 2.1), o que é, a sua arquitetura em camadas, as diferentes camadas, a forma como guarda os dados (triple stores), processamento de linguagem natural (capitulo 2.2) onde é descrito quais os seus diferentes níveis e as diferentes abordagens utilizando o processamento de linguagem natural, reconhecimento de entidades mencionadas (capitulo 2.3) onde é descrito o que é, alguns problemas que existem no reconhecimento de entidades mencionadas e ferramentas que servem para reconhecer entidades mencionadas. No capítulo 2.4 é apresentada a classificação automática de documentos de texto, os diferentes algoritmos e ferramentas que implementam alguns algoritmos descritos. No capítulo 2.5 é descrito a sumarização automática de documentos de texto e as diferentes abordagens, que são utilizadas para implementar a sumarização automática de documentos de texto.

## 2.1. Web Semântica

A *Web,* também conhecida como WWW, tem vindo a evoluir consoante o crescente aumento de utilizadores conectados. Com isto, a Web tem-se adaptado às necessidades dos utilizadores, tentando facilitar o acesso à quantidade de dados que cada vez mais existe. Se recuarmos cinquenta anos para trás, nunca podíamos imaginar que a *Web* pudesse evoluir tanto. Com o aparecimento de novas linguagens, a *Web* passou de uma abordagem estática para dinâmica, onde os utilizadores são parte integrante, podendo adicionar novos dados, alterá-los entre outras opções. É neste ponto em que a *Web* está, a *Web* 2.0. Mas esta nova forma ainda não satisfaz todos os desejos dos utilizadores, pois existem biliões de dados e quando um utilizador necessita de procurar algo específico, depara-se com muitos dados que não estão relacionados com o que deseja encontrar. Se pensarmos no Google, que permite fazer pesquisas, devolve quase sempre muitas páginas de informação, em que maior parte não é aquilo que necessitamos. A Web semântica pretende relacionar todos os dados entre si, de maneira a poder-se extrair, de uma forma mais rápida e eficaz, a informação realmente pretendida. A *Web* semântica pretende dar significado aos dados estabelecendo-lhes conexões lógicas. Para este fim, foi necessário a criação de um formato que pudesse ligar os diferentes dados, os *Universal Resource Identifiers* (*URIs*)[1].

Os *URIs* têm como objetivo principal identificar recursos, tendo assim uma função fulcral na web semântica; têm um âmbito global e são interpretados consistentemente através de contextos. A associação de um URI com um recurso permite que qualquer pessoa se possa ligar a ele, ou seja, referir-se a ele, ou recuperar uma representação dele. Os *URIs* fornecem a informação necessária para a relação entre objetos [**1**]

Por forma a auxiliar todo este cenário, a web semântica utiliza uma panóplia de conceitos e linguagens. A arquitetura da web semântica mostra a divisão de várias linguagens em diferentes camadas como mostra a Figura 1. Nos capítulos que se seguem, são descritas algumas destas linguagens.

[1] http://www.w3.org/Addressing/URL/URI_Overview.html

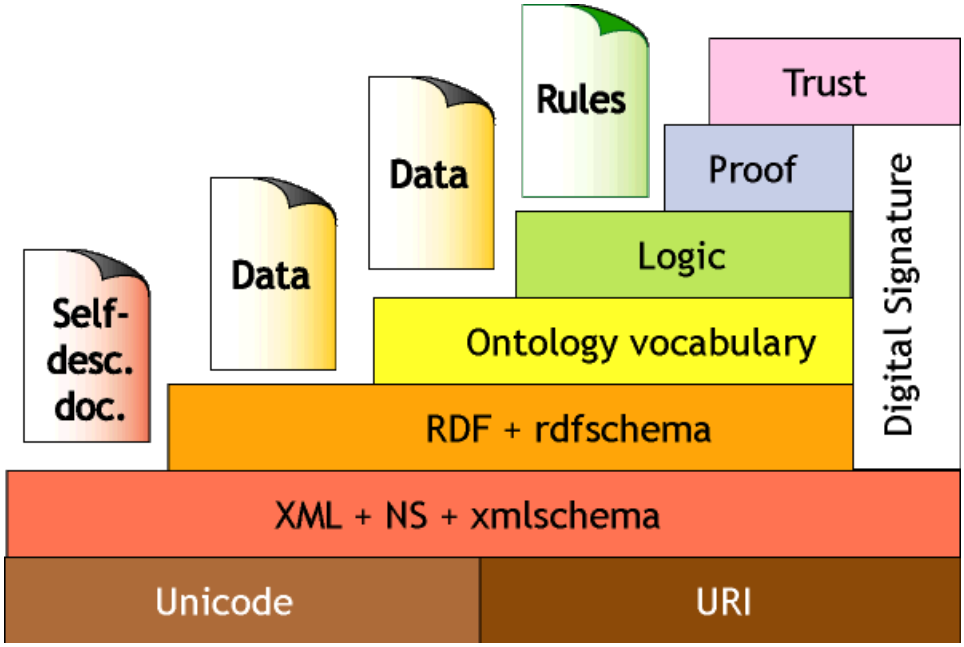



Figura 1 - Arquitetura da Web semântica[2]

### 2.1.1. Resource Description Framework[3]

Como foi referido acima, na web semântica é necessário adicionar relações entre os diversos dados por forma a ser possível interligá-los e, consequentemente, inserir uma forma de significado a esses dados. Para isso, foi criada a linguagem Resource Description Framework (RDF) em 1997, pela *World Wide Web Consortium* (W3C)[4], oferecendo uma nova estrutura de representação de dados, que fornece uma poderosa representação baseada em triplos para os *URIs*. O aparecimento desta nova linguagem foi uma mais-valia para aumentar a interoperabilidade e as funcionalidades da web.

O RDF permite ainda associar um específico *URI* para os seus campos individuais. Observando a Figura 2, podemos visualizar certas relações, onde o Eric Miller neste caso é um *individual*, sendo este identificado pelo *Resource* http://www.w3.org/People/EM/contact#me, a property que os interliga é definida por http://www.w3.org/2000/10/swap/pim/contact#fullName. [**1**]

[2] Figura apresentada por Tim Berners-Lee - http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html
[3] http://www.w3.org/RDF/
[4] http://www.w3.org/


Figura 2 - Grafo RDF que representa o Eric Miller

O RDF fornece também uma sintaxe XML chamada de RDF/XML (ver Figura 6).

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
        <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
                <contact:fullName>Eric Miller</contact:fullName>
                <contact:mailbox rdf:resource="mailto:em@w3.org"/>
                <contact:personalTitle>Dr.</contact:personalTitle>
        </contact:Person>
</rdf:RDF>
```

Figura 3 - Trecho em RDF/XML, que descreve o grafo da Figura 2

### 2.1.2. Resource Description Framework Schema[5]

O RDF Schema [**2**] é um vocabulário utilizado para descrever propriedades e classes de recursos RDF, tornando-se recomendação em Fevereiro de 2004. O RDF Schema trouxe benefícios para enriquecer a representação de modelos RDF e introduziu primitivas básicas na modelagem ontológica na Web. Em outras palavras, o RDF Schema fornece um sistema base do tipo de modelos RDF. Este tipo de sistema utiliza alguns termos predefinidos como classe, *subPropertyOf* e *subClassOf,* entre outros. A Figura 4 mostra um exemplo da implementação destes vocabulários.

```
<rdfs:Class rdf:about=”Book”/>
<rdfs:Class rdf:about=”HardCover”>
        <rdfs:subClassOf rdf:resource=”#Book”/>
</rdfs:Class>
<HardCover
rdf:resource=”http://www.books.org/ISBN0062515861”/>
```

Figura 4 - Exemplo do uso de Classes e Subclasses em RDF Schema

[5] http://www.w3.org/TR/rdf-schema/

### 2.1.3. Ontology Web Language

O RDF e RDF Schema trouxeram muitos benefícios, mas era necessário uma ferramenta que pudesse facultar uma maior expressividade perante o constante aumento da complexidade de relacionamento de objetos. Antes de se avançar para o conceito de Ontology Web Language (OWL)[6] , irá ser feita uma breve descrição do que são ontologias, dado estarem na base daquela linguagem [**1**]

Ontologia, (de onto- "ser; que é", e –logia: "ciência, estudo, teoria") é o estudo filosófico do ser, da existência ou da realidade tal como ela é, bem como das categorias básicas de ser e das suas relações [**3**]. Visto o conceito lato de ontologia, transportando agora para o objetivo para que foi criado, pode dizer-se que é um modelo de dados que representa um conjunto de definições dentro de um domínio e os relacionamentos entre estes. A ontologia modela o vocabulário e significados em domínios de interesse: os objetos (coisas) em domínios; as relações entre as coisas em propriedades, funções e processos que envolvam essas coisas; e restrições em regras sobre essas coisas [**4**].

A ideia nuclear trazida pelo OWL é permitir uma representação eficiente de ontologias. Verifica se uma ontologia é logicamente consistente ou determina se um dado conceito se enquadra na ontologia [**1**]. De uma forma simples, o OWL veio colmatar algumas fragilidades existentes no RDF.

Existem várias sub-linguagens do OWL, que são: OWL Lite, OWL DL e OWL Full. A expressividade aumenta do OWL Lite para o OWL DL e do OWL DL para o OWL Full. Segue-se uma pequena descrição de cada uma destas linguagens.

O OWL Lite tem suporte para uma classificação hierárquica e restrições simples como a cardinalidade, embora esta seja apena 0 ou 1 [**5**]. O OWL DL permite representação com máxima expressividade, embora garantindo que todas as conclusões computáveis e conclusivas terminem num tempo finito. Esta sub-linguagem permite que uma classe seja subclasse de muitas classes. No entanto, esta não pode ser instância de outra classe. Foi-lhe dado este nome devido à correspondência com descrições lógicas [**5**]. Por fim o OWL Full, que permite tirar a máxima expressividade, bem como ter a liberdade sintática do RDF, sem ter nenhum compromisso de que seja computável. Por exemplo, uma classe pode ser tratada simultaneamente como um conjunto de indivíduos ou o próprio individuo [**5**].

### 2.1.4. Triple stores

Perante a enorme quantidade de dados, e tendo em conta que a utilização de RDF e RDF Schema tem aumentado, foi necessário criar repositórios para armazenar todos estes conteúdos [**1**]. No caso das triple stores, a estrutura dos dados passa a ser um grafo, sendo este armazenado, e não os dados, em diversas tabelas. Perante isto, o sql não funcionaria, pelo que foram criadas outras linguagens, como por exemplo o Semantic Web Rule Language (SWRL)[7] e o SPARQL[8], sendo esta última a mais utilizada devido à fácil compreensão e implementação.

Relativo às *triple stores,* existem muitas implementações variando com a sua capacidade. Algumas têm como foco fornecer um meio rico para relacionar sobre os triplos (JENA TDB[9] , Jena SDB[10] ), outros concentram-se em armazenar grandes quantidades de dados (3store[11]). Alguns funcionam como plug-ins para *browsers* atuais (Piggy Bank[12]) e, por fim, outras como sistemas que funcionam com uma variedade de base de dados, Sesame[13] [**1**].

---

[6] http://www.w3.org/2004/OWL/
[7] http://www.w3.org/Submission/SWRL/
[8] http://www.w3.org/TR/rdf-sparql-query/
[9] http://jena.apache.org/documentation/tdb/index.html
[10] http://jena.apache.org/documentation/sdb/index.html
[11] http://sourceforge.net/projects/threestore/
[12] http://simile.mit.edu/wiki/Piggy_Bank
[13] http://www.openrdf.org/index.jsp

## 2.2. Processamento de Linguagem Natural

O processamento de linguagem natural (NLP) estuda os problemas da compreensão automática de linguagens humanas. Estes sistemas visam transformar as formas linguísticas humanas em representações, tornando-as desta forma numa fonte de conhecimento para os programas de computadores. A melhor maneira de explicar o que realmente um sistema de processamento de linguagem natural faz, é por meio de uma abordagem por níveis de linguagem. O processamento de linguagem natural pode ser divido em sete níveis distintos: fonologia, morfologia, léxico, sintático, semântico, discurso e pragmático.

Comecemos pela fonologia. É neste primeiro nível que se trata a interpretação dos sons, nas palavras e entre as palavras. Existem três tipos de regras na análise fonológica: a primeira contempla as regras fonéticas, ou seja, para os sons dentro das palavras, a segunda representa as regras fonémicas, ou seja, para quando existem alterações na pronúncia das palavras aquando faladas todas juntas; por último, as regras prosódicas para as alterações de *stress* e entoação ao longo das frases.

O seguinte nível é o da morfologia que tem como objetivo lidar com a natureza das palavras. As palavras são formadas com morfemas – fragmentos menores capazes de expressar significado. Por exemplo, a palavra "infelizmente" pode ser dividida em três morfemas diferentes: prefixo "in-", radical "feliz" e sufixo "-mente". Desde que o significado de cada morfema permaneça o mesmo nas palavras, as palavras desconhecidas podem ser quebradas, pois assim é possível descobrir-se qual o seu significado. Perante isto, um sistema de NLP também pode reconhecer o significado da palavra através da divisão em morfemas.

O próximo nível é o léxico; aqui os sistemas de NLP, como os humanos, interpretam o significado de cada palavra individualmente. A este nível vários tipos de processamento podem contribuir para a compreensão – o primeiro pode ser a atribuição de uma única Part-of-Speech tag (POS-tag)[14] para cada palavra. Neste processamento, as palavras que possam funcionar como mais que uma POS-tag, são associados ao mais provável baseando-se no contexto em que ocorram. Aquelas palavras, que têm um único sentido ou significado, podem ser substituídas por uma representação semântica do que significam. Neste nível pode ser exigido um léxico, e a abordagem particular no sistema NLP irá determinar se um léxico vai ser utilizado, bem como a informação que nele vai ser codificada. A complexidade dos léxicos vai depender, se utilizam apenas as palavras e as suas POS-tags, ou se irão conter informações relativas à classe semântica da palavra.

No nível sintático, a análise das palavras de uma frase é a fonte, tendo como objetivo descobrir a estrutura gramatical desta. Para isto, é necessário um analisador e uma gramática. A saída do processamento neste nível é uma representação da frase, revelando as relações de dependência estrutural entre as palavras.

O nível semântico, como o próprio nome indica, é o que determina os possíveis significados da frase, focando-se nas interações entre o significado ao nível da palavra na frase. Aqui pode ser incluído a desambiguação semântica nas palavras que têm mais que um sentido. A desambiguação semântica permite que um, e apenas um significado nas palavras polissémicas, seja selecionado e, consequentemente, incluída na representação semântica da frase.

O penúltimo nível é o do discurso, onde ao contrário dos níveis anteriores (semântico e sintático) que trabalham com unidades do tamanho da frase, lida com unidades texto maiores que uma frase. O discurso centra-se nas propriedades do texto como um todo, que transmitem significado através da conexão dos componentes da frase. A este nível muitas vezes ocorre a resolução de anáfora, substituindo pronomes que semanticamente são muito vagos, por entidades a que realmente se referem.

---

[14] Etiquetas parte do discurso

Por último, o nível pragmático tem como principal preocupação utilizar propositadamente a língua em determinadas situações e utilizar o contexto para além do conteúdo do texto, para o entendimento. Isto requer muito conhecimento do mundo, incluindo o entendimento das intenções, planos e objetivos. Nalgumas aplicações de NLP, utilizam bases de conhecimento e módulos de inferência.

### 2.2.1. Abordagens para Processamento de Linguagem Natural

Existem quatro categorias de abordagens de processamento de linguagem natural: simbólica, conexionista, estatística e híbrida. Na abordagem simbólica é desempenhado uma análise profunda dos fenómenos linguísticos, baseando-se em representações explícitas de factos sobre a linguagem através do conhecimento bem compreendido da representação de esquemas e algoritmos associados. A abordagem estatística utiliza várias técnicas matemáticas, e muitas vezes, usam grandes *corpora* de textos para desenvolver modelos generalizados, aproximados a modelos baseados em fenómenos linguísticos. Um modelo usado no modelo estatístico é o Hidden Markov Model (HMM). HMM é um autómato de estados finito, que tem um conjunto de estados com probabilidades associadas nas transições entre estados.

As abordagens estatísticas tipicamente têm sido usadas em tarefas como reconhecimento de recurso, aquisição lexical, análise, POS-tag, máquinas de tradução estatísticas, etc..

A abordagem conexionista é semelhante à estatística. O que muda nesta abordagem, em relação aos outros métodos estatísticos, é o facto de este combinar a aprendizagem estatística com várias teorias de representação – pelo que as representações conexionistas permitem transformações, inferência e manipulação de lógica de fórmulas. A utilização desta metodologia pode ser feita em tarefas, como a análise sintática, tarefas de tradução limitadas ao domínio, etc..

Por último, a abordagem híbrida é usada para juntar as forças de cada abordagem, na tentativa de resolver os problemas de NLP de uma forma mais eficaz e da maneira mais flexível possível [**6**].

O NLP utiliza-se em várias tarefas, como em reconhecimento de entidades mencionadas, classificação automática de documentos, sumarização automática de documentos de texto, etc.. Nos capítulos seguintes, serão abordados de forma mais exaustiva as tarefas de NLP anteriormente enumeradas.

## 2.3. Reconhecimento de Entidades Mencionadas

O conceito de entidades mencionadas vem do inglês "*named entities*", e isto não é mais do que atribuir a certas entidades existentes no mundo real uma designação através de palavras, ex.: "Lisboa", "Coimbra", "Jumbo", "Worten", etc. O reconhecimento de entidades mencionadas (REM) tem como tarefa classificar entidades em determinadas categorias, tentando levar a um acordo geral. Porém, a extração destas entidades não é uma tarefa fácil, pois existem alguns problemas com a identificação de determinados termos, dado a vagueza existente na semântica. Os maiores problemas que podemos identificar são a metonímia e a ambiguidade.

A metonímia é o uso de uma entidade para referenciar outra entidade diferente, uma entidade que é de um tipo diferente. Por exemplo, "Edison ilumina o mundo", quer dizer que as lâmpadas que foram inventadas por Thomas Edison iluminam o mundo, sendo o nome Edison a entidade que referencia outra entidade.

A ambiguidade traz uma complexidade maior ao reconhecimento, pois as entidades podem ter vários significados diferentes e, ao fazer esta identificação, pode levar a que certas classificações, por vezes, não sejam as mais corretas. Este caso pode ser visto por uma entidade ter mais do que um significado dependendo do contexto em que está inserido. Por exemplo, a entidade banco pode representar o objeto onde nos sentamos, ou então, a entidade onde são colocados os rendimentos, entidade bancária.

Para avaliar o sucesso da identificação e, consequentemente, a classificação automática dos nomes próprios na língua portuguesa, surgiu o HAREM[15]. O HAREM é uma avaliação conjunta na área do reconhecimento de entidades mencionadas em português.

Seguidamente serão apresentadas algumas ferramentas que participarem no HAREM, com o objetivo de reconhecer entidades mencionadas. [**7**]

### 2.3.1. Reconhecimento de Entidades Mencionadas Baseado em Relações e Análise Detalhada do Texto (REMBRANDT)

A abordagem seguida pela ferramenta REMBRANDT [**8**] divide-se em três etapas, Figura 5.

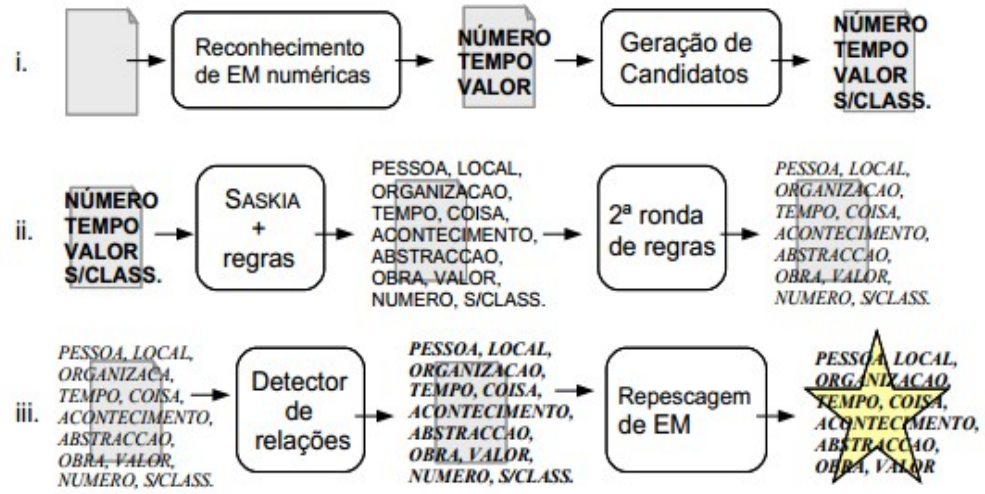



Figura 5 - Etapas do REMBRANDT [8]

A primeira etapa é o reconhecimento de expressões numéricas e geração de candidatos a Entidades Mencionadas (EM). Aqui os textos são divididos em frases e em unidades, com o auxílio do atomizador da Linguateca. Aqui, um primeiro conjunto de regras identifica expressões numéricas. Após isto, são aplicadas regras para reconhecer expressões temporais e valores, tirando proveito dos números anteriormente reconhecidos.

A forma utilizada para geração de candidatos a EM é através de sequências de unidades com, pelo menos, uma letra maiúscula e/ou um algarismo.

A segunda etapa trata da classificação de EM onde cada uma das candidatas a EM é classificada primeiro pela *SPARQL and API Service for Knowledge and Information Access* (SASKIA), e depois novamente classificada por regras gramaticais. O facto de utilizar esta "dupla classificação" permite obter vantagens: primeiro, onde a SASKIA realiza uma classificação de acordo com os vários significados que a EM pode ter, e depois são reunidas nas páginas de desambiguação da *Wikipédia*. Assim, o processo de desambiguação pode ter um papel importante na seleção de significados corretos da EM; Segundo, as regras gramaticais englobam indícios externos e internos das EM, pelo que permite supervisionar as classificações da SASKIA no contexto da EM. Para finalizar esta etapa, é executada uma segunda ronda de regras gramaticais, que usufrui das classificações existentes para detetar EM com uma morfologia mais complexa.

Por último, a etapa da repescagem de EM: aqui é feita a deteção de relações entre EM através de um conjunto de regras específicas para a tarefa. O passo anterior descrito permite repescar algumas EM sem classificação, mas que estão relacionadas com as EM classificadas corretamente.

[15] http://www.linguateca.pt/aval_conjunta/HAREM/

Depois de detetadas as relações, é feita uma última repescagem de EM com nomes de pessoas, através de uma comparação com uma lista de nomes comuns. Por fim, todas as EM que não têm classificação são eliminadas, assim como os números por extenso sem uma letra maiúscula.

### 2.3.2. RENA

O RENA [**7**]foi um protótipo de sistema desenvolvido por Edgar Alves sob a supervisão de J. J. Almeida no âmbito do projeto IKS[16], com o objetivo de extrair/reconhecer entidades mencionadas, tentando quanto possível tornar a informação obtida o mais completa possível.

De uma forma superficial, este sistema é constituído por uma biblioteca Perl, onde se encontram um conjunto de ficheiros de configuração passiveis de receber alterações, funcionalidades para extrair as listas de entidades de vários textos e, por último, ou alternativamente, permite a marcação de entidades num conjunto de texto. Para além do que foi descrito, também contém um conjunto de programas para fazer o processamento de entidades.

Após descrito o sistema, pode-se agora fazer um *zoom in* e explorar a estrutura interna, de maneira a perceber com maior detalhe todos os passos que são necessários, aquando da identificação das entidades.

A estrutura interna, do ponto de vista algorítmico, inicia-se na procura por entidades e constrói uma sequência de textos simples e entidades: (texto × entidade)*, após isto, este objeto é processado por uma série de filtros com assinatura, que vão processar os pares texto-entidades, de forma a obter uma informação mais completa. Estes filtros utilizados têm como tarefas:

- O tratamento de entidades com elementos de uma única letra;
- Tratamento de aspas ligadas às entidades;
- Remoção de entidades entre aspas;
- Tratamento de entidades com traços interiores;
- Tratamento de entidades em início de frase;
- Enriquecimento por análise de regras de contexto;
- Enriquecimento por análise de almanaque de nomes e cultura geral;
- Tratamento de acrónimos;
- Reconhecimento e unificação de entidades iguais e criação de atributos de ligação entre as várias ocorrências da mesma entidade.

Todos estes filtros, referidos anteriormente, podem ser desativados ou ativados de acordo com o que o utilizador pretenda. Por fim, é criado, de acordo com o pretendido, um texto com as entidades anotadas e um resumo das entidades presentes.

### 2.3.3. SIEMÊS

A ferramenta SIEMÊS [**7**] foi desenvolvida com o intuito de melhorar a robustez da classificação de entidades mencionadas, tendo para isto combinado um conjunto de regras de análise de contexto com a consulta de almanaques, de onde se pode obter informação importante e que facilita a análise posterior. O SIEMÊS assume que, se numa primeira fase conseguir gerar um conjunto de hipóteses para classificar um determinado candidato, através da informação existente nos almanaques, será possível numa segunda fase desambiguar semanticamente a classe e a forma de menção do referido candidato usando regras de análise de contexto simples.

No entanto, como alguns resultados não foram os melhores, foi desenvolvido uma segunda versão desta. O SIEMÊS v2 foi feito de raiz, por forma a corrigir os problemas observados na versão anterior tentando, no entanto, não alterar a filosofia da versão anterior.

[16] Projeto IKS (Information + Knowledge + Fusion)

Uma das grandes vantagens desta nova versão é a possibilidade de criar bancos de regras externos que são interpretados por um motor genérico. Desta forma, foi possível complementar a estratégia da outra versão, pois consegue elevar o número de regras que lidam com contextos bem definidos.
O funcionamento desta versão pode ser decomposto em duas camadas principais:

1. Camada de identificação de candidatos, através de pistas formais, como a presença de maiúsculas ou números. Esta camada recorre a um banco de regras para identificar candidatos alfabéticos e a um outro onde é feita em simultâneo a identificação e classificação semântica de entidades numéricas. Neste tipo de entidades, a identificação e classificação são feitas no mesmo passo, pois não existem grandes problemas de ambiguidade.
2. Camada de classificação para as entidades alfabéticas. A composição desta camada é uma cadeia de classificação com cinco componentes que permitem gerar hipóteses de classificação dos candidatos, através de estratégias diferentes. Depois desta cadeia, aplica-se o componente final de desambiguação, que procura escolher de entre as várias hipóteses geradas qual a mais correta, tendo em conta informação adicional acerca do contexto.

Como foi identificado anteriormente, a camada de classificação contém uma cadeia de geração de hipóteses com cinco componentes, pelo que são invocados sequencialmente. Os componentes, bem como as suas estratégias, são descritos a seguir:

1. Bloco de regras "simples" – este componente é composto por um conjunto de regras que foram codificadas manualmente. A composição das regras é feita de forma compacta baseando-se no conhecimento de certas classes semânticas de palavras, como cargos, tipos de povoações, tipos de organizações, entre outros grupos de palavras importantes no contexto do REM.
2. Bloco de pesquisa direta REPENTINO – este bloco permite fazer uma pesquisa sobre o almanaque REPENTINO possuindo, para isso, um módulo Perl onde armazena toda a informação deste. Para um dado candidato, é verificada a quantidade de entradas, no almanaque, que possuam a mesma representação lexical, sendo guardada a informação relativa às suas classes e subclasses, ficando a ser consideradas hipóteses de classificação.
3. Bloco de emparelhamento de prefixo sobre o REPENTINO – este bloco tenta encontrar no REPENTINO as instancias que possuam o mesmo prefixo que o candidato. A pesquisa efetua-se inicialmente dado um certo número de palavras do candidato (as duas ou quatro primeiras) e são pesquisadas as instâncias no almanaque que se iniciem pelas mesmas palavras. Os resultados obtidos da pesquisa no almanaque são agrupados por categorias. Quando uma dessas categorias inclui mais de 40% das referidas instâncias, é gerada uma hipótese de classificação, que se baseia nessa categoria e subcategorias. Caso não se alcance os 40%, então reduz-se uma palavra à pesquisa de prefixos e tenta-se um novo emparelhamento com o almanaque. Isto é repetido até a tentativa de emparelhamento incluir apenas uma palavra ou então até ser atingido o limite de cobertura de 40%. Pode não haver geração de nenhuma hipótese, levando o processo de pesquisa de hipótese para os outros blocos.
4. Bloco de semelhança sobre o REPENTINO – neste bloco são utilizadas duas funções heurísticas que tentam estabelecer a semelhança entre um determinado candidato e o conteúdo no REPENTINO. Quanto maior for a semelhança entre o candidato e as instâncias dentro de uma categoria e subcategoria, mais elevado é considerado o grau de pertença a essa categoria e subcategoria, sendo portanto gerada uma hipótese de classificação.
5. Bloco posterior de recurso – este último bloco contem as regras, a usar no fim da cadeia de classificação, e que pretendem explorar algumas pistas contextuais muito genéricas. Estas regras embora simples, podem ser suficientes para resolver mais alguns casos que ainda não foram anteriormente tratados.

### 2.3.4. CORTEX

O CORTEX [**7**] é uma ferramenta de processamento de linguagem desenvolvido para o reconhecimento de entidades mencionadas. O processamento desta divise-se em várias etapas, como mostra a Figura 6.

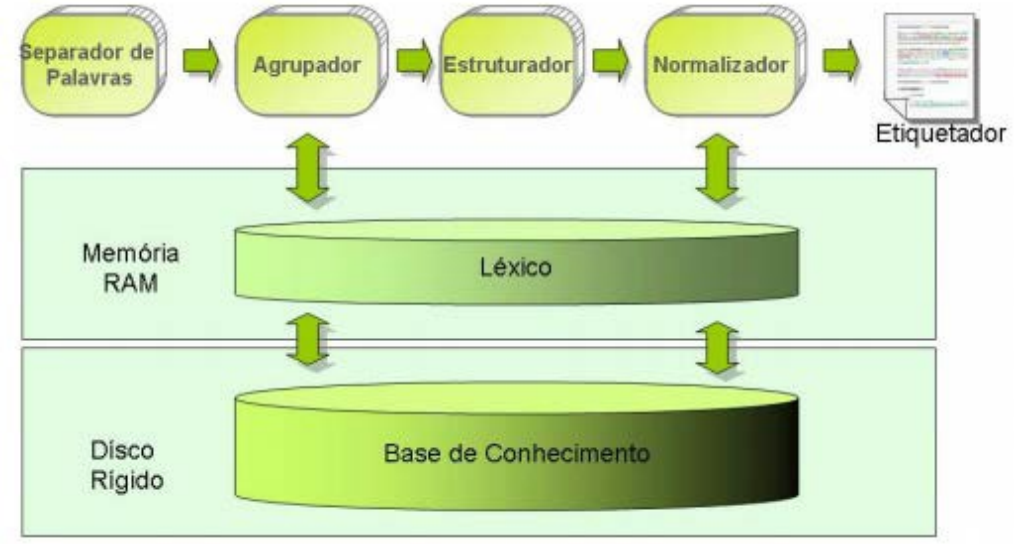



Figura 6 - Etapas de avaliação do CORTEX [7]

Etapa a etapa o CORTEX é capaz de inferir resultados futuros baseando-se em passos anteriores.

Fazendo um *zoom in* à estrutura do CORTEX, a tarefa inicial é a separação das palavras; após isto, segue-se a tarefa de reconhecer as entidades que possam ser constituídas por mais de uma palavra. Aqui podem ser encontrados substantivos compostos e locuções.

O reconhecimento dos termos é executado através da utilização de um autómato, sendo este fundamentalmente criado para identificar padrões de formação de entidades compostas, baseando-se num repertório de regras. Aqui, o resultado é armazenado no conhecimento existente no léxico, e só depois à base de dados.

Segue-se o passo do classificador de termos previamente extraídos. O banco de informação existente permite ao CORTEX obter informação sobre uma palavra (sua classe, significado, etc.), no entanto, isto não é tomado como uma definição absoluta, possibilitando estas serem questionadas e reavaliadas sempre que seja necessário.

Este facto torna o CORTEX um mecanismo dinâmico, dado a utilização da experiência previamente adquirida, por forma a poder inferir um maior número de novas informações/textos.

A obtenção das informações por parte do CORTEX é feita através de quatro fontes de dados distintas, como é possível observar na Figura 10.

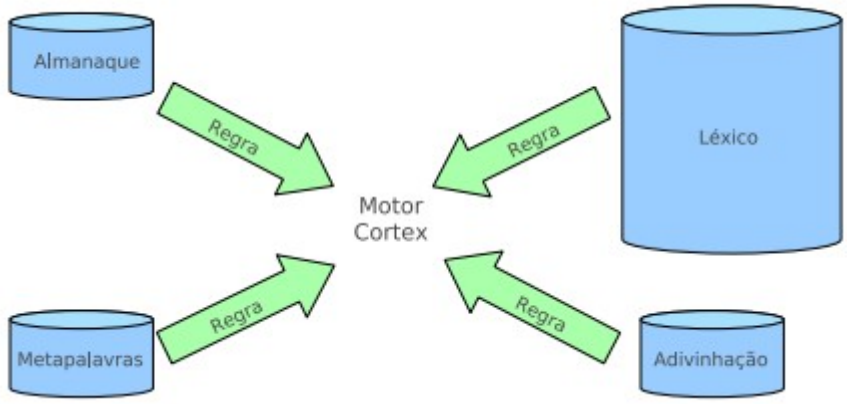



Figura 7 - Fontes de dados do CORTEX [7]

A fonte Almanaque contém uma lista de entidades de uma determinada categoria provenientes de uma fonte enciclopédica; A fonte Metapalavras é constituída por uma lista de termos que aparecem nas vizinhanças das entidades; A fonte de Adivinhação contém um conjunto de termos, que constituem entidades mencionadas, como, Dr., Sr., Prof., Eng.; Por último, a fonte Léxico que armazena todo o conhecimento apreendido nos textos anteriormente processados.

## 2.4. Classificação automática de documentos de texto

O processo de classificação envolve associar a cada documento uma e só uma categoria. Em termos de Text Mining, o problema da classificação pode estar ligada a diversos domínios. Estes são:

- Filtragem e organização de notícias – A maior parte dos serviços de notícias estão hoje em formato eletrónico. Por isso, a quantidade de notícias criadas, cada dia, por organização é grande, levando à necessidade de organizar tudo de forma automática, pois executar esta tarefa manualmente seria muito difícil.
- Organização e recuperação de documentos – A filtragem, como anteriormente referido, pode ser usada também na organização de documentos. Esta organização pode ser em vários domínios que incluem coleções web, literatura científica, grandes bibliotecas de documentos, etc..
- Classificação de emails e filtros de spam – Por vezes, é necessário classificar *email* para determinar, de forma automática se é o sujeito ou se é lixo eletrónico. Isto também é chamado de filtragem de *spam* ou *email*.
- Reconhecimento de entidades mencionadas – Neste campo é necessário organizar em diferentes classes tudo o que seja extraído do texto, pois só assim se pode agrupar e criar conhecimento proveniente das entidades reconhecidas.

Existem algumas metodologias que permitem a classificação do texto, descritas a seguir:

- Árvores de decisão – utilizam uma estratégia de divide-and-conquer, de maneira a decompor a complexidade do problema de classificação em sub-problemas de menor complexidade. A descriminação de uma árvore é obtida pela divisão dos espaços definidos pelos atributos em subespaços, sendo que, a cada subespaço, é associado uma classe. Um exemplo de um algoritmo, baseado nisto, é o ID3 ou C45.

- Padrões baseados em regras – determina os padrões de palavras que têm uma maior probabilidade de estar relacionados com diferentes classes. Depois são constituídas um conjunto de regras que, de um lado (esquerdo) ficam os padrões de palavras e, do outro lado, a etiqueta desta classe. Um algoritmo baseado em regras é, por exemplo, o One-R.
- Máquina de vetores de suporte (SVM) – utiliza delimitações lineares ou não lineares na tentativa de particionar o espaço de dados. A ideia chave é estes classificadores determinarem os limites ideais entre as diferentes classes.
- Redes neuronais – a utilização deste tipo de classificadores está presente em muitos domínios. Em relação ao reconhecimento de padrões em texto, a principal diferença trazida pelas redes neuronais é a adaptação às características das palavras. Estes classificadores estão relacionados com os de SVM, pois ambos encontram-se na categoria de discriminativos em contraste aos generativos.
- Baysianos (Naïve Bayes) – tem como ideia fundamental a criação de um classificador probabilístico baseado na modelagem das características das palavras em diferentes classes.
- Para além destes classificadores descritos anteriormente haveria outros a apontar, como o *nearest neighbor* e os baseados em algoritmos genéticos [**9**].

### 2.4.1. Ferramentas para Classificação automática de Documentos de texto

Como ferramentas para auxiliar a implementação da classificação automática de documentos de texto, encontraram-se essencialmente quatro: *MAchine Learning for LanguagE Toolkit (MALLET)*[17], *Weka*[18], *Apache Mahout*[19] e *Apache OpenNLP*[20]. O objetivo, aqui, é conhecer melhor as ferramentas que podem ser utilizadas na classificação automática de documentos de texto.

O *MALLET* é uma ferramenta baseada em Java para o processamento de linguagem natural estatística, classificação de documentos, *clustering*, modelação de tópicos, extração de informação, entre outras aplicações de *machine learning* sobre texto. Esta ferramenta possui alguns métodos que ajudam na classificação de documentos. Estes métodos são *Naïve Bayes*, Máxima Entropia e Árvores de Decisão. Para além disto, também possui código para avaliar o desempenho do classificador em termos de qualidade [**10**]. Esta ferramenta possui uma boa documentação e muitos exemplos de implementações.

O *Weka* é um projecto desenvolvido pela Universidade de Waikato, baseado em *Java*, para *machine learning* e *data mining*. A área da classificação é um dos seus pontos fortes, tendo diversas abordagens de Machine learning como *Naïve Bayes*, Árvores de decisão. Para além destas abordagens possui ainda uma técnica não supervisionada, *Clustering* [**11**].

O *Apache Mahout* é uma ferramenta escalável para grandes quantidades de *datasets*. Possui um núcleo de algoritmos para *clustering*, classificação de texto. Alguns dos algoritmos de classificação que o *Apache Mahout* contém são: *Naïve Bayes*, SVM, redes neuronais, entre outros [**12**]. Esta ferramenta tem um senão que é a dificuldade da implementação, devido à fraca documentação existente.

Por fim, o Apache OpenNLP é uma ferramenta baseada em *machine learning*, para o processamento de linguagem natural (NLP) em texto. Assim, tem suporte para muitas tarefas de NLP, como a tokenização, segmentação de frases, POS-tag, reconhecimento de entidades mencionadas, *chuncking*, *parsing*, classificação de documentos. A *OpenNLP* utiliza o algoritmo de *machine learning* para a classificação automática de documentos de texto: Máxima Entropia [**13**].

[17] http://mallet.cs.umass.edu/
[18] http://www.cs.waikato.ac.nz/ml/weka/
[19] http://mahout.apache.org/
[20] http://opennlp.apache.org/

## 2.5. Sumarização automática de documentos de texto

A sumarização, de uma forma geral, é uma atividade bastante comum. Por exemplo, quando se narra um determinado evento a uma pessoa, normalmente faz-se um resumo do que aconteceu, e não é narrado na íntegra tudo o que se passou nesse evento. Do mesmo modo, também na escrita são feitos resumos, quer seja sobre noticias, artigos científicos, artigos de revistas, entre muitos outros. Dada esta enormidade de utilizações de sumarização, o presente capítulo irá descrever diversas metodologias utilizadas para gerar automaticamente sumários.

A sumarização automática pode ser dividida essencialmente em duas abordagens distintas: a superficial e a profunda. Nos subcapítulos seguintes, serão apresentados cada uma destas abordagens, bem como os métodos que lhes estão associados.

### 2.5.1. Abordagem Superficial

Uma abordagem superficial [**14**] utiliza técnicas estatísticas, o que torna a aplicação desta abordagem mais simples, pois não necessita de algoritmos complexos, podendo ser aplicado a qualquer tipo de texto, sejam estruturas de texto bem formadas ou não. Por outro lado, os métodos utilizados nesta abordagem não levam em consideração todo o potencial conhecimento da língua. Alguns dos métodos que pertencem a esta abordagem são: o método das palavras-chave, método da localização e o método das palavras-chave usando o algoritmo *Term Frequency – Inverse Sentence Frequency* (TF-ISF), método este baseado em *Text Mining*. O *Text Mining* é uma área que contempla o estudo das relações existentes entre componentes de texto não estruturado. Este inter-relacionamento pode ser interno relacionando apenas os componentes de um texto, ou externo abrangendo vários textos. No caso prático em questão, sumarização automática, é importante identificar informações relevantes em um dado contexto textual, por isso, apenas será contemplado um texto de cada vez – o que será sumarizado.

Todos os métodos, que serão apresentados a seguir, passam por quatro fases de pré-processamento:

- Separação de frases – tem como função decompor o texto em unidades menores, que são as frases.
- *Case-folding* – que corresponde a converter todos os caracteres do documento para um mesmo formato, ou seja, em letras maiúsculas ou minúsculas.
- *Stemming* – que consiste em converter cada palavra à sua forma radical, ou seja, sua forma neutra sem inflexões verbais ou de número.
- *Remoção de stop words* – as *stop words* são palavras de classe fechada, ou seja, que não acrescentam qualquer tipo de significado como, por exemplo, os artigos e pronomes. Esta remoção é feita através do auxílio de uma lista previamente preenchida com estas *stop words*. [**14**]

#### *2.5.1.1. Método das palavras-chave*

O método das palavras-chave parte do pressuposto que as ideias principais de um texto podem ser expressas por algumas palavras-chave. De acordo com Black e Johnson [**14**,**15**], quando as ideias vão sendo transcritas para o texto, as palavras-chave aparecem com maior frequência. Assim, a ideia é determinar a distribuição estatística das palavras-chave do texto e, de acordo com a sua frequência, extrair as frases que as contenham, agrupando-as de forma a constituir um sumário e ordenando-as na forma como aparecem no texto original.

#### *2.5.1.2. Método da localização*

O método da localização parte do princípio que a posição em que uma frase ocorre no texto poderia estar associada à sua importância no contexto textual. Por exemplo, a primeira e a

última frase de um parágrafo podem conter suas ideias principais e, portanto, estas seriam as escolhidas para incorporar no sumário. Foi mostrado, por Baxendale [**14**,**16**] que em 85% de uma amostra de 200 parágrafos, a frase principal era a primeira e, em 7% a última. Os restantes 8%, as informações mais importantes encontravam-se entre a primeira e a última frase de um mesmo parágrafo. No entanto, o autor considerou que a última frase, embora com a baixa percentagem, era relevante para o sumário, dado constituir o elo de ligação com o parágrafo seguinte.

#### *2.5.1.3. Método das palavras-chave e text mining*

Este método varia relativamente ao primeiro na forma como identifica as palavras-chave, onde o primeiro se baseia na frequência que as palavras ocorrem no texto, neste método utiliza-se um algoritmo de sumarização que representa cada frase como um vetor de pesos TF-ISF (Term Frequency – Inverse Sentence Frequency). A relevância (peso) de uma certa palavra é calculada de acordo com a fórmula seguinte:

$$TF - ISF(w,s) = TF(w,s) \times ISF(w)$$

onde TF(w,s) é o número de vezes que a palavra w ocorre na frase s, e a frequência inversa da frase é obtida através da fórmula:

$$ISF(w) = \log \frac{|S|}{SF(w)}$$

onde a |S| corresponde ao número de frases existentes no texto e SF(w) é o número de frases nas quais a palavra w ocorre. Por fim o valor do peso da palavra na frase é dado por

$$Score(s) = \sum TF \times ISF(w,s).$$

#### *2.5.1.4. Método das palavras-chaves baseado em Text Mining e localização*

Por último, ainda existe um outro algoritmo que faz o cruzamento entre o método das palavras-chave baseada nos pesos TF-ISF e o método da localização, sendo atribuídos pesos às frases relativamente à posição onde ocorrem no texto. Assim as frases iniciais e finais têm maior relevância do que as que se encontram entre estes dois limites. Neste caso, deve ser encontrada uma heurística que dê pesos às frases, relativamente à posição que elas ocupam no texto.

### 2.5.2. Abordagem Profunda

A abordagem profunda varia relativamente à superficial, na medida em que incorpora conhecimento linguístico e/ou extralinguístico presente no texto original. A abordagem profunda tem como princípios:

1. A identificação da proposição central no texto original;
2. Identificar quais as informações que complementam esta proposição central, com a finalidade de transmitir e preservar no sumário a ideia principal do texto original;
3. Por último, permitir a estruturação do sumário, tendo como base nas partes (1) e (2) e em restrições e normas relativas à coesão e coerência, de modo a que o sumário final seja também um texto coerente e estruturado. [**14**]

A utilização da abordagem profunda traz grandes vantagens ao nível da sumarização automática, dado usar informação linguística para a criação de um sumário. No entanto, a modelação deste tipo de abordagem encontra grandes problemas em termos da definição, estruturação e manipulação da informação proveniente do texto original, fazendo com que se dê preferência à abordagem superficial. Assim sendo, este subcapítulo serviu apenas para indicar que existe outra abordagem que se pode utilizar na sumarização automática, no entanto, como o tempo existente não é muito, optou-se apenas pela abordagem superficial.

# Capítulo 3
# Avaliação do projeto EDUCA

O projeto EDUCA surge de uma parceria entre a Flor de Utopia, a Metatheke e a Faculdade de Ciências e Tecnologia da Universidade de Coimbra. O objetivo é desenvolver uma plataforma informática para suportar um repositório de conteúdos digitais distribuído e escalável, utilizando tecnologias abertas, para agregar, pesquisar, publicar e proteger os conteúdos multimédia empresariais, científicos, educativos e culturais. Pretende-se ainda que a plataforma seja também capaz de extrair automaticamente informação, classificando e sumarizando documentos de texto.

Ao longo do semestre passado, foram feitos testes para avaliar o estado de algumas ferramentas utilizadas no projeto, como o caso do motor de NER e o repositório de conteúdos. Por forma a ser possível avaliá-los, procedeu-se à comparação destes com outras ferramentas existentes. Foram feitos dois tipos de teste:

1. Performance das triple stores (Jena TDB, Sesame e Mulgara), sendo o Mulgara a triple store utilizada pelo fedora commons;
2. Performance e qualidade do motor de reconhecimento de entidades mencionadas do apache stanbol e REMBRANDT;

Todos estes testes foram feitos numa máquina com as seguintes características:

- Processador core i7 1,6GHz quad-core;
- 6GB de RAM;
- 500GB de disco;
- Sistema operativo: Windows 7 64 bits.

Para a realização destes testes foi utilizado a linguagem Java. Os argumentos usados na máquina virtual do java foram: -Xmx4096m –XX:MaxPermSize=4096m. Os subcapítulos, que se seguem, abordam cada um destes testes, explicando a maneira como foram elaborados e os resultados obtidos.

## 3.1. Avaliação da Performance das *Triple Stores*

Os testes de performance das triple stores foram feitos em termos de inserção de triplos (bulkload) e execução das queries. Na geração dos *datasets* necessários no *bulkload,* foi usado o Berlin SPARQL Benchmark (BSBM)[21]. Os datasets gerados têm os tamanhos: 100 mil triplos, 500 mil triplos, 1 milhão de triplos, 5 milhões de triplos, 10 milhões de triplos, 20 milhões de triplos e 50 milhões de triplos. As *queries* utilizadas estão escritas em SPARQL e também pertencem a este benchmark, sendo que para o efeito apenas 10 foram utilizadas. Estas *queries* foram ordenadas das menos complexas para as mais complexas. As *queiries* podem ser consultadas no Anexo A do relatório. Todos os testes foram executados 35 vezes, sendo descartados os primeiros 5 para evitar o *warmup*.

### 3.1.1. *Bulkload*

Os resultados obtidos relativos ao *bulkload* podem ser vistos nas Tabela 1, 2 e 3 e no Gráfico 1.

[21] http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

| Jena TDB | 100K | 500K | 1M | 5M | 10M |
|---|---|---|---|---|---|
| **Média** | 4887,6667 | 26716,3 | 48429,33 | 291226,07 | 604781,33 |
| **Desvio Padrão** | 2759,4737 | 1863,2366 | 1139,1894 | 18636,751 | 44501,595 |
| **Intervalo de Confiança** | 987,44684 | 666,73839 | 407,64621 | 6668,9533 | 15924,399 |

Tabela 1 – Valores em milissegundos do *Jena TDB*

| Sesame | 100K | 500K | 1M | 5M | 10M | 20M | 50M |
|---|---|---|---|---|---|---|---|
| **Média** | 4358 | 25668,27 | 53876,03 | 468409,6 | 973647,6 | 3002750,4 | 6798012,6 |
| **Desvio Padrão** | 111,67 | 671,02 | 465,69 | 87251,06 | 52529,23 | 310253,79 | 617757,37 |
| **Intervalo de Confiança** | 39,96 | 240,119 | 166,64 | 31221,82 | 18796,99 | 111020,85 | 221057,57 |

Tabela 2 – Valores em milissegundos do *Sesame*

| Mulgara | 100K | 500K | 1M | 5M | 10M |
|---|---|---|---|---|---|
| **Média** | 12399,4667 | 89841,6667 | 171782,467 | 987540,7 | 2972238,83 |
| **Desvio Padrão** | 272,271401 | 3230,26418 | 4731,18825 | 60840,6181 | 596448,627 |
| **Intervalo de Confiança** | 97,4292791 | 1155,91395 | 1693,00286 | 21771,1355 | 213432,478 |

Tabela 3 – Valores em milissegundos do Mulgara

Gráfico 1 – Performance de bulkload das diferentes triple stores

Ao observar os valores do gráfico, pode dizer-se que o *Jena TDB* em termos de *bulkload,* tem a melhor performance até aos 10 milhões de triplos; no entanto, quando se passa este valor, o *Jena TDB* necessita de mais memória, fazendo com que não consiga executar o *bulkload* para 20 e 50 milhões de triplos.
O *Mulgara,* que é o triple store utilizado pelo *Fedora-commons,* teve a pior performance, não sendo possível concluir os testes de 20 e 50 milhões devido ao tempo necessário para executar as 35 vezes.
Por último, o *Sesame,* embora não tivesse o melhor tempo de *bulkload,* foi a única triple store onde foi possível concluir todas as inserções.
Perante isto, podemos concluir que o *Fedora-commons* utilizando o *Mulgara* irá ser muito lento no acesso aos dados dentro da triple store.

### 3.1.2. Queries

Os testes de performance, que foram desenvolvidos às *queries*, foram feitos aos diversos *bulkloads* descritos anteriormente. Cada *querie* foi executada 35 vezes, sendo descartados os primeiros 5 por causa do *warmup*. Nos Gráficos 2 e 3 encontram-se a soma das médias das 10 e 8 *queries*, respetivamente.

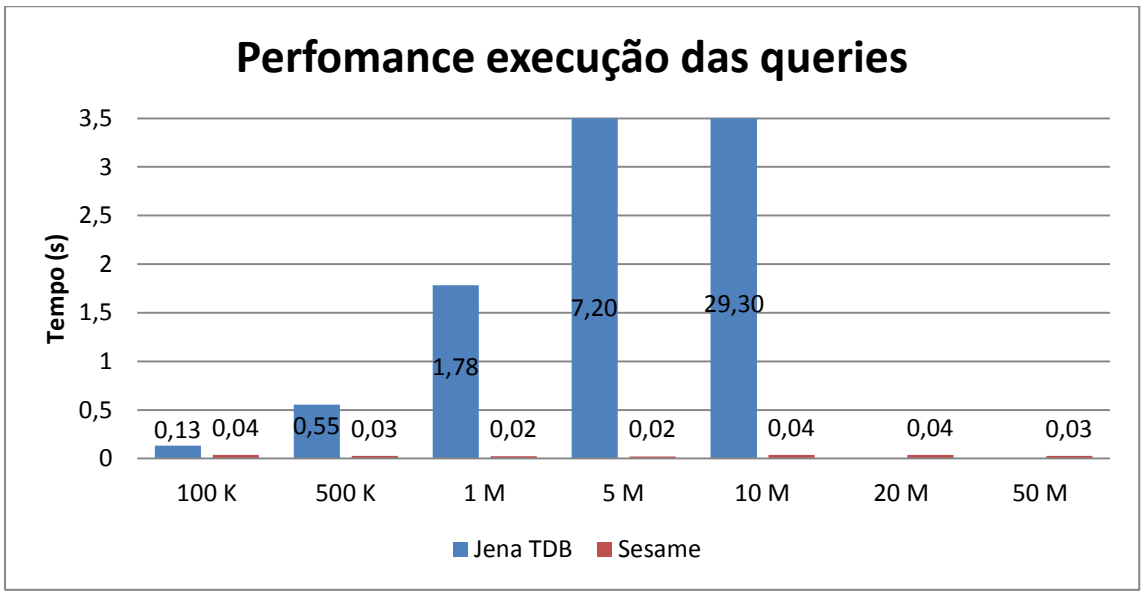



Gráfico 2 – performance da execução das queries no Jena TDB e Sesame

O Gráfico 2 mostra os resultados obtidos entre o *Jena TDB* e o *Sesame*, onde foi possível correr as 10 *queries*. No caso do Gráfico 3 mostra os resultados obtidos pelo *Mulgara*, pelo que apenas foi possível executar as primeiras 8 *queries*, dado que as últimas *queries* não eram suportadas pelo *Mulgara*.

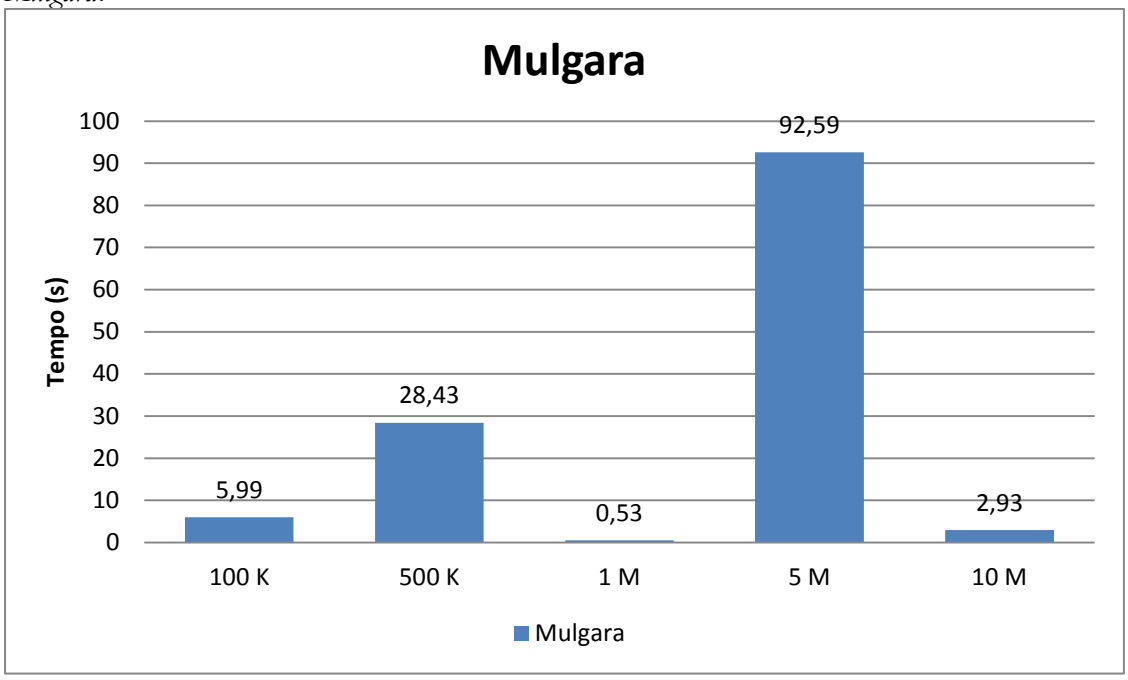



Gráfico 3 – Performance da execução das queries no *Mulgara*

Observando os dados apresentados no Gráfico 2 podemos dizer que o *Sesame* é o mais rápido em termos de execução de *queries*, mantendo a performance qualquer que seja o número de triplos existentes na triple store. O *Jena TDB* demonstra diminuição do desempenho quando aumenta o número de tri plos.

Em relação aos valores apresentados no Gráfico 3, este mostra que os valores de performance são muito inconsistentes, não sendo possível concluir qualquer coisa a partir deles. Possivelmente, devido a problemas com a utilização do *SPARQL* pelo *Mulgara*, pois a linguagem principal de *queries* utilizada pelo *Mulgara* é o *Three Query Language* (TQL)[22], havendo ainda

[22] http://xml.coverpages.org/Conforti-webdb2002.pdf

algumas questões pendentes relativas à implementação do *SPARQL* no *Mulgara*. Outro problema poderá ter a ver com a forma de acesso a esta triple store, onde o acesso é feito por *Web Service*. O facto de a ligação ser feita a um *Web Service,* pode depender da disponibilidade deste. No caso deste teste haveria de se ter retirado o tempo que demora a fazer o acesso ao serviço.

## 3.2. Performance e qualidade do reconhecimento de entidades mencionadas

Para desenvolver os testes ao reconhecimento de entidades mencionadas foram utilizadas as coleções douradas do HAREM. Estas são constituídas por 129 documentos, com as entidades mencionadas já reconhecidas de forma manual. Estes testes tiveram como objetivo comparar a performance e a qualidade do *Apache Stanbol* e do *REMBRANDT*. A metodologia utilizada para executar os testes foi:

1. Separar os diversos documentos em ficheiros;
2. Ler os ficheiros que contêm os documentos e, com o motor de reconhecimento de entidades mencionadas, anotar as entidades encontradas.

Nos subcapítulos que se seguem são apresentados os resultados relativos à performance e qualidade do *Apache Stanbol* e *REMBRANDT*.

### 3.2.1. Performance

A performance foi calculada executando 35 vezes o reconhecimento de entidades mencionadas para os 129 documentos, descartando os 5 primeiros devido ao *warmup*. A Tabela 4 e o Gráfico 4 mostram a performance do *Rembrandt* e do *Apache Stanbol*.

| | REMBRANDT | Apache Stanbol |
|---|---|---|
| **Média** | 3081825,1 | 100193,8 |
| **Desvio Padrão** | 111914,37 | 2556,757 |
| **Intervalo de Confiança** | 40047,308 | 914,907 |

**Tabela 4 – Valores da performance dos motores de reconhecimento de entidades mencionadas em milissegundos**

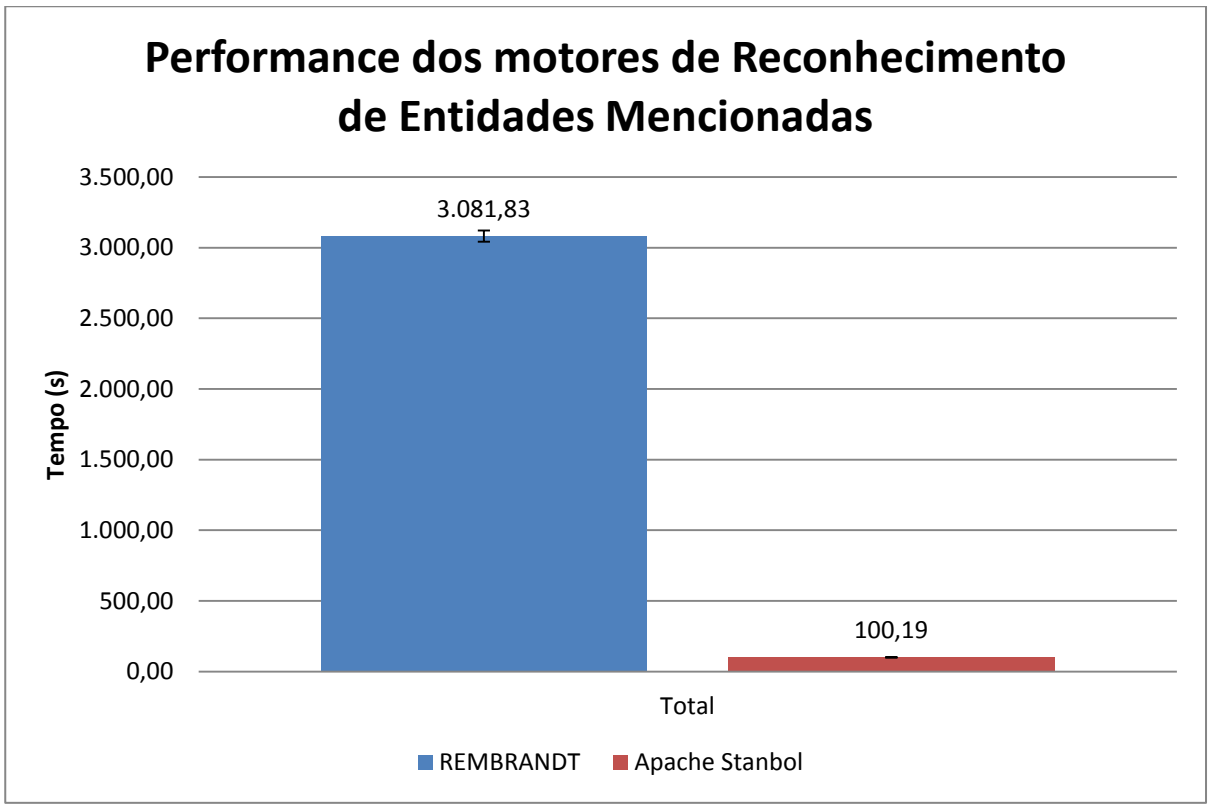



Gráfico 4 – Performance do reconhecimento de entidades mencionadas

Observando os resultados do Gráfico 4 pode dizer-se que, em termos de performance, o *Apache Stanbol* consegue ser extremamente rápido no reconhecimento de entidade mencionadas, fazendo o reconhecimento dos 129 documentos em apenas 100 segundos, aproximadamente. O *REMBRANDT,* em termos de performance, apresenta maus resultados, pois demora mais de 3000 segundos para fazer o reconhecimento de entidades mencionadas no mesmo número de documentos.

### 3.2.2. Qualidade

A análise de qualidade do reconhecimento de entidades mencionadas foi segundo as fórmulas da precisão, abrangência (*recall*) e medida-F (*F-measure*) que podem ser vistas a seguir:

- $Precisão = \frac{Verdadeiros\ Positivos}{(Verdadeiros\ Positivos + Falsos\ Psitivos)}$
- $Abrangência\ = \frac{VeVerdadeiros\ Positivos}{(Verdadeiros\ Positivos + Falsos\ Negativos)}$
- $Medida - F = 2\ \times \frac{(precisão \times abrangência)}{(precisão + abrangência)}$

Os verdadeiros positivos são as entidades reconhecidas pelo motor de reconhecimento de entidades mencionadas que correspondem às anotadas nas coleções douradas. Os falsos positivos são as entidades que são reconhecidas pelo motor de reconhecimento de entidades mencionadas, mas que não estão anotadas nas coleções douradas. Os falsos negativos são as entidades que estão anotadas nas coleções douradas, mas que não foram anotadas pelo motor de reconhecimento de entidades mencionadas.
No Gráfico 5 pode-se ver os resultados obtidos para o *REMBRANDT* e para o *Apache Stanbol.*

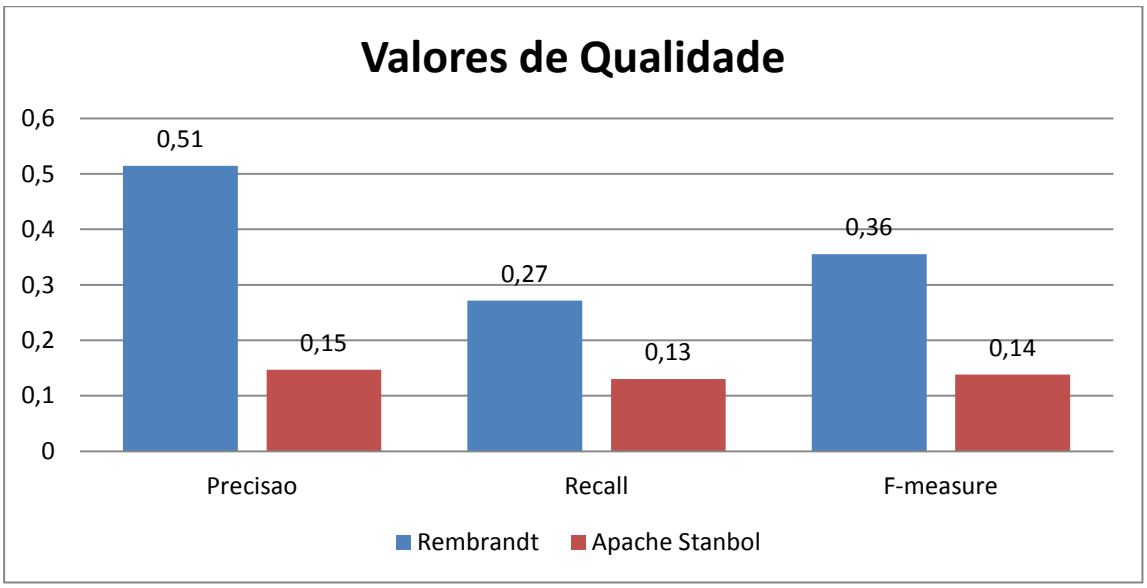



Gráfico 5 – Valores de qualidade do Rembrandt e do Apache Stanbol

Observando os resultados presentes no Gráfico 5, podemos tirar 2 conclusões. A primeira, é que comparando o *REMBRANDT* com o *Apache Stanbol*, o *REMBRANDT* tem melhor qualidade no reconhecimento de entidades mencionadas. A segunda conclusão é que, apesar do *REMBRANDT* ser o melhor, os resultados apresentados são baixos, não garantindo grande qualidade no reconhecimento de entidades mencionadas, pois apenas consegue em termos de *F-measure (medida-f)* alcançar o 36%. Isto indica que consegue reconhecer com certeza 3 entidades em 10, aproximadamente. Apesar disto, é necessário referir que a área de reconhecimento de entidades mencionadas ainda se encontra em investigação, na procura de uma solução que apresente melhor qualidade no reconhecimento de entidades mencionadas.

## 3.3. Conclusões

As conclusões que se podem tirar desta avaliação são: em termos de repositório, o *Fedora-commons* irá apresentar algumas limitações quanto à performance, pelo que numa próxima versão do projeto EDUCA, deveria-se considerar a hipótese de estudar novas ferramentas, avaliá-las e comparar o seu desempenho com o *Fedora-Commons,* a ver se compensava o investimento de alterar a ferramenta.

Relativamente ao reconhecimento de entidades mencionadas, existem dois problemas: o facto do que apresenta melhores resultados em termos de qualidade, apresentar os piores resultados em termos de performance e, o que apresenta melhores resultados em termos de performance ser pior em termos de qualidade.

Neste caso, a utilização do REMBRANDT, que apresenta melhores resultados em termos de qualidade, torna a tarefa de reconhecimento de entidades mencionadas muito morosa, pelo que optando por fazer isto em *background* e não mostrar os resultados para o utilizador, seja uma boa solução.

# Capítulo 4
# Análise de requisitos, Arquitetura e *Design*

Neste capítulo, será feita uma pequena descrição sobre o projeto EDUCA, seguida da descrição dos requisitos necessários para alcançar os objetivos propostos no projeto EDUCA, no âmbito dos módulos da responsabilidade da empresa Flor de Utopia e onde o estágio teve uma contribuição. Serão ainda descritos os casos de uso que suportam os requisitos, a arquitetura e *design* do projeto.

O projeto EDUCA, como indicado anteriormente, surge de uma parceria. O objetivo é desenvolver uma plataforma informática para suportar um repositório de conteúdos digitais, utilizando tecnologias para agregar, pesquisar, publicar e proteger os conteúdos multimédia empresariais, científicos, educativos e culturais. Outro foco deste projeto é que a plataforma seja capaz de automaticamente extrair informação, classificar e sumarizar documentos de texto. Neste momento, todas as soluções já se encontram implementadas repartidas por vários módulos.

Seguidamente vai ser descrito cada módulo, a sua função dentro do projeto e a instituição responsável pelo seu desenvolvimento. Os módulos do projeto EDUCA da responsabilidade da empresa Metatheke são:

- **educa-aggregator** – tem como função a ligação a repositórios externos e importar os metadados para o repositório EDUCA. Esta importação utiliza o protocolo *Open Archives Initiative Protocol* for *Metadata Harvesting* (OAI-PMH[23]) e o protocolo *Open Archives Initiative Object Reuse and Exchange* (OAI-ORE[24]) utilizado para complementar o protocolo *OAI-PMH*;
- **educa-catalog** – tem como função a gestão do catálogo de conteúdos onde é feita a ponte entre a plataforma e o repositório (*Fedora-Commons*). Neste módulo também se encontra implementada a gestão de licenças. Inicialmente a gestão de licenças estava prevista ser gerida por um módulo específico para esse efeito, designado **educa-drm**;
- **educa-search** – responsável pela indexação da informação no *Apache Stanbol* e pela pesquisa dos conteúdos digitais.

Os módulos da responsabilidade da Flor de Utopia são:

- **educa-analysis** – onde se encontra implementada toda a análise de conteúdos, como por exemplo, extração de metadados, extração de conteúdos de ficheiros de texto e reconhecimento de entidades mencionadas;
- **educa-content** – tem como função a gestão de conteúdos do utilizador, permitindo adicionar/editar conteúdos simples, conteúdos compostos ou colaborativos e conteúdos do tipo coleção;
- **educa-store** – módulo responsável pela gestão da loja virtual à qual estão associados, como produtos, os conteúdos digitais que se encontram no repositório;
- **educa-core** – tem como principal função a gestão de todos os módulos do projeto e consequentemente o acesso ao módulo **educa-datamodel**;
- **educa-person** – a sua principal função é fazer a gestão de utilizadores na plataforma;
- **educa-contact** – este módulo tem como finalidade a gestão dos contatos dos utilizadores.

Os módulos de responsabilidade conjunta de ambas as empresas, devido à sua transversalidade, são:

[23] http://www.openarchives.org/pmh/
[24] http://www.openarchives.org/ore/

- **educa-webapp** – módulo onde se encontra toda a parte visual da plataforma web;
- **educa-datamodel** – módulo onde se encontram todos os modelos representantes das tabelas existentes na base de dados;
- **educa-test –** módulo onde são realizados os testes unitários.

As tarefas do estágio estão enquadradas neste projeto, nomeadamente em parte dos desenvolvimentos da responsabilidade da Flor de Utopia. A implementação de funcionalidades de classificação semiautomática de texto e sumarização automática de texto estavam inicialmente previstas no âmbito do módulo **educa-analysis**. No entanto, optou-se pela criação de dois sub-módulos para executar estas tarefas:

- **educa-classification** – Tem como função o treino do classificador automático de documentos de texto e a classificação automática de documentos de texto.
- **educa-summarization –** tem como função fazer a sumarização automática de documentos de texto.

Foi decidido fazer esta divisão para ser possível utilizar estes módulos separadamente em outras aplicações. Ainda no âmbito do estágio, e de modo a interligar os módulos **educa-classification** e **educa-summarization** e dar-lhes uma utilização prática e efetiva na plataforma, foi definido, em conjunto com a Flor de Utopia, os requisitos necessários para gerir os conteúdos elaborados pelos utilizadores-produtores. A implementação do módulo **educa-content** torna possível a criação de conteúdos simples e de conteúdos compostos de forma colaborativa, que sejam depois sujeitos a uma classificação e a uma sumarização.
No diagrama da Figura 8, seguinte, podem ser vistas as dependências entre estes módulos, as instituições responsáveis e os módulos que o estagiário trabalhou.

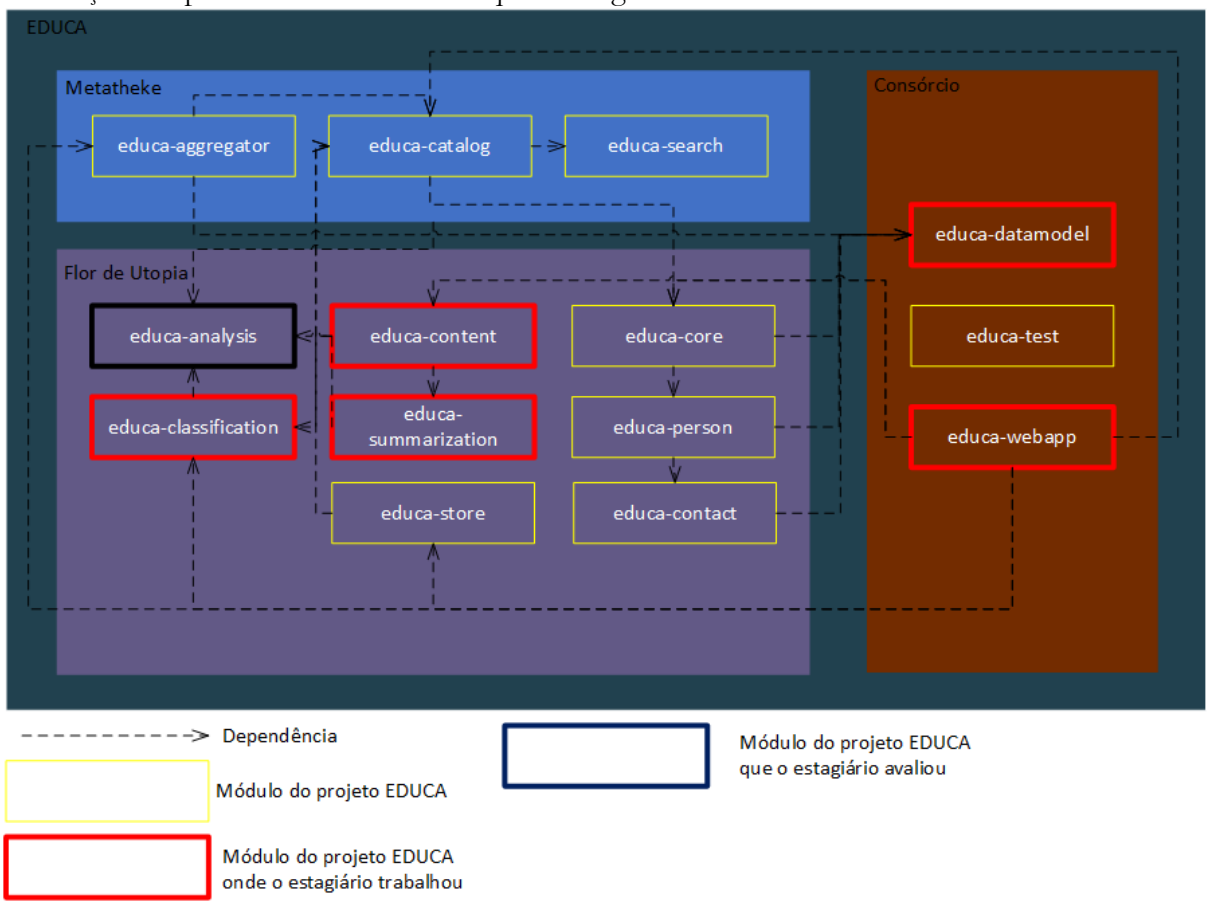



Figura 8 – Módulos do projeto EDUCA

## 4.1. Requisitos funcionais da plataforma

Nesta secção são listados os requisitos funcionais necessários nos módulos **educa-content, educa-classification** e **educa-summarization**, onde irão ser priorizados de acordo com o método de análise **MoSCoW** [**17**]. Este método divide os requisitos em 4 categorias:

- **Must**: Requisitos essenciais que devem ser implementados para que a solução apresentada possa apresentar sucesso.
- **Should:** Requisitos importantes devem ser implementados, mas, se não o forem, o sistema poderá ser implantado e usado normalmente.
- **Could:** Requisitos desejáveis que podem ser implementados se não afetarem outros requisitos e o *deadline* do projeto.
- **Won't:** Requisitos que não serão implementados durante o projeto, por questões de tempo, mas que podem ser implementados no futuro.

Este método de análise foi aplicado de acordo com duas prioridades: importância dentro do estágio e importância para o projeto EDUCA.
Na tabela seguinte encontram-se apresentados os principais grupos de requisitos:

| Código | Descrição |
|---|---|
| **RF 1** | Gestão de conteúdo digital |
| **RF 2** | Classificação automática de documentos de texto |
| **RF 3** | Sumarização automática de documentos de texto |

Tabela 5 – Requisitos Funcionais de alto nivel

Definidos os requisitos de alto nível, segue-se uma descrição mais pormenorizada de cada um, bem como uma decomposição destes em sub-requisitos funcionais, caso se verifique necessidade.

### 4.1.1. RF 1 – Gestão de conteúdo digital

Este requisito, como foi indicado acima, tem como principal função a gestão de conteúdos digitais. Assim sendo, pode dividir-se este requisito em outros, como pode ser visto na seguinte tabela:

| Código | Descrição |
|---|---|
| **RF 1.1** | Conteúdo simples |
| **RF 1.2** | Conteúdo composto ou colaborativo |
| **RF 1.3** | Coleção de conteúdos do utilizador |
| **RF 1.4** | Gestão de *Layouts* |
| **RF 1.5** | Gestão de *Templates* |

Tabela 6 – Sub-requisitos do Requisito Funcional gestão de conteúdo digital (RF 1)

#### *4.1.1.1. RF 1.1 – Conteúdo simples*

O conteúdo simples corresponde ao utilizador adicionar um ficheiro ao repositório. Este ficheiro pode ser uma imagem, vídeo, ficheiro de áudio, documento de texto, etc. Os requisitos que se encontram na tabela seguinte correspondem às operações possíveis em relação ao requisito de nível imediatamente acima "Conteúdo Simples".

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.1.1** | Criar conteúdo simples | Should | Must |

| | | | |
|---|---|---|---|
| **RF 1.1.2** | Editar conteúdo simples | Could | Should |
| **RF 1.1.3** | Remover conteúdo simples | Could | Should |
| **RF 1.1.4** | Listar conteúdos simples | Should | Must |

Tabela 7 – Sub-requisitos do Requisito Funcional conteúdo simples (RF 1.1)

#### *4.1.1.2. RF 1.2 – Conteúdo composto ou colaborativo*

Um conteúdo composto ou colaborativo representa um ficheiro que é pensado para ser feito por um ou mais utilizadores, ou seja, contém diversos "espaços" para adicionar conteúdo. Estes conteúdos têm que ser definidos de maneira a ter uma ordem ou uma estrutura.

Os sub-requisitos que lhe estão associados podem ser vistos na tabela seguinte:

| **Código** | **Descrição** | **Prioridade Estágio** | **Prioridade projeto EDUCA** |
|---|---|---|---|
| **RF 1.2.1** | Criar conteúdo composto ou colaborativo | Could | Must |
| **RF 1.2.2** | Editar conteúdo composto ou colaborativo | Could | Should |
| **RF 1.2.3** | Remover conteúdos compostos ou colaborativos | Could | Should |
| **RF 1.2.4** | Listar conteúdo composto ou colaborativo | Could | Must |
| **RF 1.2.5** | Pré-visualizar conteúdo composto ou colaborativo | Could | Must |
| **RF 1.2.6** | Adicionar permissão a utilizadores para a colaboração do preenchimento do conteúdo | Could | Must |
| **RF 1.2.7** | Permitir a escolha de um ou mais tipos de conteúdos a inserir | Could | Must |
| **RF 1.2.8** | Preencher conteúdo composto ou colaborativo | - | - |
| **RF 1.2.9** | Editar preenchimento conteúdo composto ou colaborativo | - | - |
| **RF 1.2.10** | Remover conteúdo preenchido | Could | should |
| **RF 1.2.11** | Validação do conteúdo preenchido em cada painel | Could | Must |
| **RF 1.2.12** | Finalização do conteúdo composto ou colaborativos | Could | Must |

Tabela 8 – Sub-requisitos do Requisito Funcional conteúdo composto ou colaborativo (RF 1.2)

Aqui ainda é possível decompor o RF 1.2.8 e RF 1.2.9 em sub-requisitos que passam a ser descritos na tabela seguinte.

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.2.8.1, RF 1.2.9.1** | Adicionar texto ao conteúdo composto | Could, Could | Must, Should |
| **RF 1.2.8.2, RF 1.2.9.2** | Adicionar conteúdo simples de áudio | Could, Could | Must, Should |
| **RF 1.2.8.3, RF 1.2.9.3** | Adicionar conteúdo simples de video | Could, Could | Must, Should |
| **RF 1.2.8.4, RF 1.2.9.4** | Adicionar conteúdo simples documento texto/imagem | Could, Could | Must, Should |

Tabela 9 – Sub-requisitos dos Requisitos Funcionais Preencher conteúdo composto ou colaborativo (RF 1.2.8) e Editar preenchimento do conteúdo composto ou colaborativo (RF 1.2.9)

#### *4.1.1.3. RF 1.3 – Coleção de conteúdos do utilizador*

A coleção de conteúdos do utilizador, em termos de funcionalidade, é o utilizador de uma lista de conteúdos previamente adicionados no catálogo, escolher vários, criando uma coleção de diversos ficheiros, escolhidos de forma não estruturada. Os sub-requisitos aqui presentes podem ser consultados na Tabela seguinte:

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.3.1** | Criar coleção de conteúdo | Could | Must |
| **RF 1.3.2** | Editar coleção de conteúdo | Could | Should |
| **RF 1.3.3** | Remover coleção de conteúdo | Could | Should |
| **RF 1.3.4** | Listar as coleções de conteúdo | Could | Must |

Tabela 10 – Sub-requisitos do Requisito Funcional coleção de conteúdos do utilizador (RF 1.3)

### 4.1.2. RF 1.4 – Gestão de layouts

A gestão de *layouts* surge com o intuito de possibilitar ao utilizador a forma como irá definir o conteúdo composto ou colaborativo, ou seja, como o vai agrupar/ordenar. Um *layout* é constituído por vários painéis (containers) que são as unidades mais simples de cada *layout*, e que representam a estrutura final do *layout*.

Os sub-requisitos que se podem retirar daqui podem ser vistos na tabela seguinte:

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.4.1** | Criar *layout* | - | - |
| **RF 1.4.2** | Editar *layout* | - | - |
| **RF 1.4.3** | Remover *layout* | Could | should |
| **RF 1.4.4** | Listar *layouts* | Could | Must |
| **RF 1.4.5** | Pré-visualizar *layout* | Could | Must |

Tabela 11 – Sub-requisitos do Requisito Funcional gestão de *layouts* (RF 1.4)

A partir destes requisitos ainda se podem dividir em sub-requisitos, pelo que na tabela seguinte pode-se ver esta divisão:

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.4.1.1, RF 1.4.2.1** | Adicionar painel (*container*) | Could, Could | Must, should |
| **RF 1.4.1.2, RF 1.4.2.2** | Redimensionar painel (*container*) | Could, Could | Must, should |
| **RF 1.4.1.3, RF 1.4.2.3** | Remover painel (*container*) | Could, Could | Must, should |

Tabela 12 – Sub-requisitos dos Requisitos funcionais criar *layout* (RF 1.4.1) e editar *layout* (RF 1.4.2)

### 4.1.3. RF 1.5 – Gestão de templates

Os *templates* são outra parte fundamental da criação de conteúdos compostos ou colaborativos. Aqui é escolhido um *layout* anteriormente criado. O principal objetivo do *template* é definir as cores de fundo, quer dos painéis quer do fundo onde se encontra o painel, e o alinhamento do conteúdo dentro do painel. Os sub-requisitos estão declarados na tabela seguinte:

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.5.1** | Criar *template* | - | - |
| **RF 1.5.2** | Editar *template* | - | - |
| **RF 1.5.3** | Remover *template* | Could | should |
| **RF 1.5.4** | Listar *templates* | Could | Must |
| **RF 1.5.5** | Pré-visualizar *template* | Could | Must |

Tabela 13 – Sub-requisitos do Requisito Funcional gestão de *templates* (RF 1.5)

Nesta tabela ainda existem alguns requisitos que podem ser subdivididos, pelo que, na tabela seguinte, pode ser vista esta subdivisão.

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 1.5.1.1, RF 1.5.2.1** | Adicionar cor de fundo ao template | Could, Could | Must, should |
| **RF 1.5.1.2, RF 1.5.2.2** | Adicionar cor de fundo a cada painel (container) | Could, Could | Must, should |
| **RF 1.5.1.3, RF 1.5.2.3** | Escolher alinhamento do conteúdo dentro do painel | Could, Could | Must, should |

Tabela 14 – Sub-requisitos do Requisito Funcional criar *template* (RF 1.5.1) e editar *template* (RF 1.5.2)

### 4.1.4. RF 2 – Classificação automática de documentos de texto

A classificação automática de documentos de texto funciona de duas formas distintas: a primeira, na inserção de conteúdo simples, onde após o utilizador adicionar um ficheiro de texto, os campos das categorias são pré-preenchidos, no entanto, o utilizador poderá antes de terminar, alterar esta classificação gerada, acrescentando ou removendo categorias; a segunda é na inserção de conteúdo composto ou colaborativo, são geradas várias classificações (uma para cada painel) e depois o utilizador, com essa informação gerada, escolhe a classificação final do conteúdo que será criado.

De acordo com isto, e visto que a classificação utiliza uma abordagem de aprendizagem, as principais operações deste requisito são:

- Treino do classificador automático
- Classificação do conteúdo

Na tabela seguinte podem ser vistos estes requisitos:

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 2.1** | Iniciar treino do classificador | Must | Must |
| **RF 2.2** | Classificar automaticamente o conteúdo | Must | Must |

Tabela 15 – Sub-requisitos do Requisito Funcional classificação automática de documentos de texto (RF 2)

### 4.1.5. RF 3 – Sumarização automática de documentos de texto

A sumarização automática tem como principal função resumir de forma automática documentos de texto. Esta tarefa irá permitir ao utilizador aquando da inserção de um novo conteúdo simples do tipo de documento de texto, a sumarização, substituindo o utilizador na escrita manual de um resumo acerca do conteúdo que inseriu. Este requisito pode ser visto na tabela seguinte.

| Código | Descrição | Prioridade Estágio | Prioridade projeto EDUCA |
|---|---|---|---|
| **RF 3** | Sumarização automática de documentos de texto | Must | Must |

Tabela 16 – Requisito Funcional sumarização automática de documentos de texto (RF 3)

## 4.2. Arquitetura e *design* do projeto EDUCA

Em termos gerais, a arquitetura da plataforma assenta numa infraestrutura eficiente para programação colaborativa e uma arquitetura modular que permite uma boa escalabilidade.

A infraestrutura utilizada permite à equipa da Flor de Utopia e Metatheke um desenvolvimento colaborativo em interações curtas, automatizando o processo de compilação, execução de testes e integração entre módulos, geração de documentação e gestão de qualidade de código. O servidor que foi preparado para este efeito (*Komodo*) tem instalado o *Jenkins CI* que permite a compilação de projetos de *software* ou agendar estas compilações para uma data específica. O *Jenkins* também se encontra integrado com o repositório de código e controlo de versões CVS e com a ferramenta que gere os módulos e dependências do projeto – Apache Maven versão 3. Outro beneficio que é trazido pelo *Jenkins* é a execução automática de testes, bem como o *deploy* automático da aplicação web num *Web Container*. No caso do projeto EDUCA é utilizado o *Apache Tomcat* 7 como Web Container.

A plataforma segue um modelo de multicamadas, o qual hoje em dia se encontra implementado em muitas aplicações. As camadas na qual o projeto se encontra dividido são: *Presentation*, *Business, Integration, Data*. Cada uma destas camadas desempenha um papel específico dentro da aplicação e utilizam diferentes tecnologias para desempenhar certas tarefas. Na Figura 9, seguinte, pode ser visto o diagrama de arquitetura de alto nível.

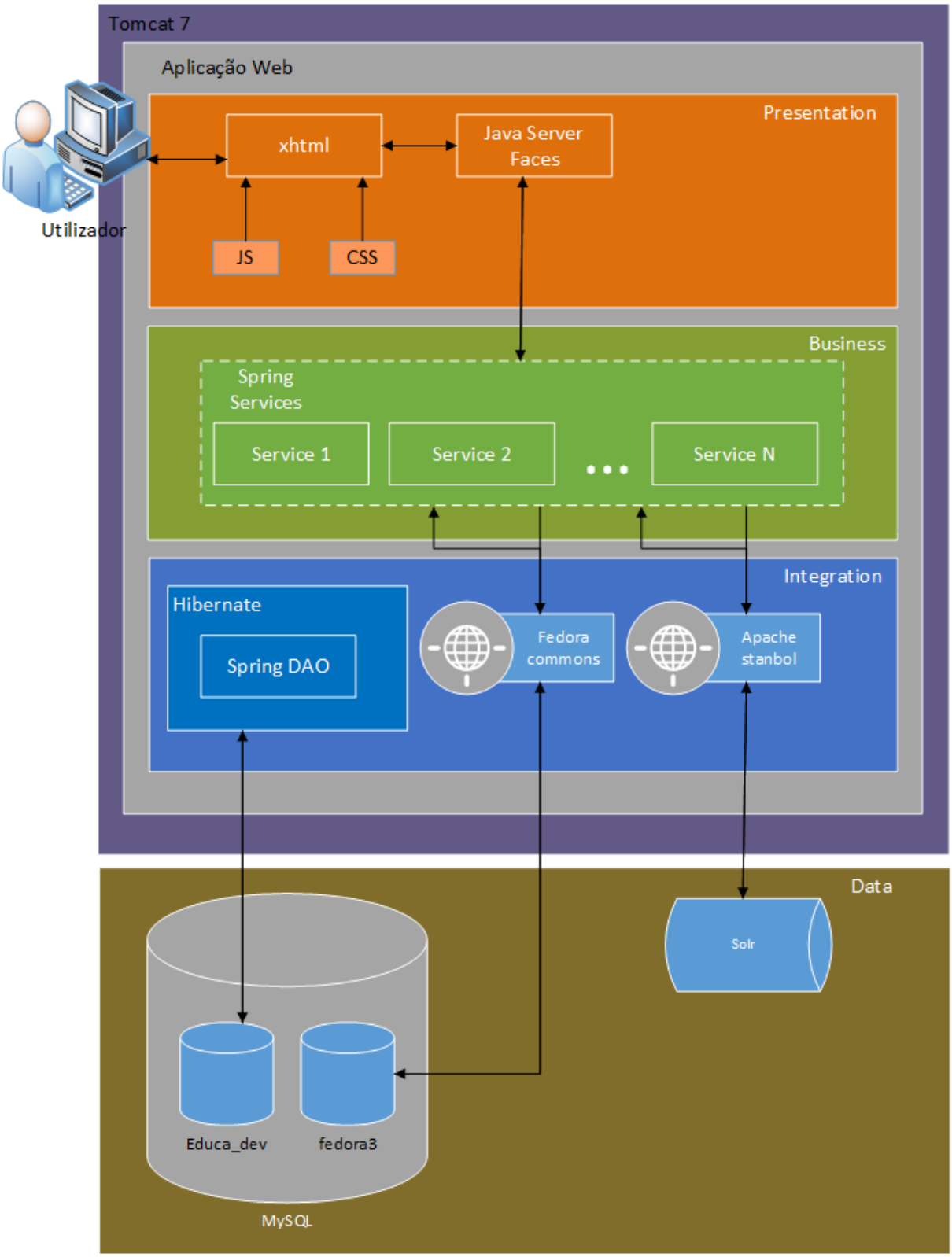



Figura 9 – Arquitetura do projecto EDUCA

Através da imagem é possível ter a perceção das diferentes camadas e das diferentes ferramentas usadas nelas. Seguidamente, é apresentado de uma forma mais geral cada camada e as ferramentas usadas em cada uma, bem como as relações existentes entre elas.

Na camada *Presentation* encontra-se toda a parte visual da aplicação. No caso do projeto EDUCA são utilizados ficheiros xhtml para visualizar as diversas páginas do *website*, sendo que o *Java Server Faces* (*JSF*), é responsável por todas as ligações à camada *Business*, dado fazer a ponte

entre estas duas camadas. Nesta camada também são usados os *RichFaces,* componentes específicos do *JSF*, que abstraem certos aspetos de código como, por exemplo, chamadas *AJAX*.

A camada *Business* é responsável por toda a lógica da aplicação, nomeadamente a navegação, validação de erros e ligação entre as duas outras camadas da aplicação *Presentation* e *Integration*. Nesta camada as ferramentas utilizadas são os *Spring Services*. Os *Spring Services* comunicam com a camada de *Integration*, quer seja para guardar algum dado referente a utilizadores, contactos, grupos entre outros, quer seja utilizar algum *Web Service*. Assim, estes serviços também fazem o acesso e o registo de conteúdos no catálogo, utilizando para isso, uma chamada ao *Web Service Fedora-commons* existente na camada *Integration*. Ao nível da pesquisa, estes serviços fazem uma chamada ao *Web Service Apache Stanbol*, que contém um motor de índexes o *Solr* [25], onde faz o armazenamento e a pesquisa dos metadados no *Solr*.

A camada *Integration* é responsável pelo armazenamento e acesso aos dados em base de dados. No caso especifico do projeto EDUCA são utilizados para aceder à base de dados *educa_dev* os *Spring DAO* (Data-Access-Object), que são responsáveis pelo acesso aos objetos da base de dados. Os Spring DAO permitem a utilização de diversas abordagens de gestão de dados como o caso de *JDBC* (*Java database Connectivity*)[26], *JPA* e *Hibernate*. No caso do projeto foi utilizado o *Hibernate* para fazer o mapeamento dos dados das tabelas diretamente para objetos e vice-versa.

A camada Data é uma camada física que contém o motor de base de dados que é utilizado, *MySQL*. Aqui, e como pode ser visto na figura com o diagrama de arquitetura, são utilizadas duas bases de dados diferentes, o *educa_dev* e o *fedora3*. O *educa_dev* é onde se encontram todas as tabelas usadas na implementação, para guardar informação relativa a utilizadores, grupos, menus, contactos entre outros. No caso do *fedora3* é uma base de dados que apenas é utilizada pelo *Fedora-commons*, onde guarda todos os dados relativos aos conteúdos que serão guardados no catálogo. Nesta camada também pode ser visto o motor de índex, *Solr,* com o qual o *Web Service Apache Stanbol* comunica, quer para armazenar os índexes quer para executar a pesquisa.

Como referido anteriormente, são utilizados no projeto dois *Web Services*, um que serve como repositório (*Fedora-commons*) e outro como motor de pesquisa (*Apache Stanbol*). O *Apache Stanbol* é um projeto que tem como principal objetivo a gestão de conteúdo semântico, sendo o acesso disponibilizado via *Web Service*. Dado o projeto EDUCA estar focado para a web semântica, necessita de serviços para guardar informações existentes nos conteúdos que serão inseridos pelos utilizadores e torna-los em "conhecimento". No caso especifico do projeto, a utilização é exclusivamente a pesquisa dos dados extraídos/indexados no motor *Solr* onde armazena a informação numa triple store (*Jena TDB*). Esta ferramenta encontra-se implementada no módulo **educa-search.**

O *Fedora-commons* é um repositório de conteúdos que possibilita a gestão destes, mantendo-os a longo prazo. Permite também gerir relações entre conteúdos, tendo como triple store o *Mulgara*; no entanto esta o triple store *Mulgara* não é utilizada, devido ao projeto criar metadados específicos diferentes dos que são utilizados internamente pelo *Fedora-commons*. A utilização desta ferramenta é feita dentro do módulo **educa-catalog.**

### 4.2.1. Casos de Uso

Na Figura 10 podem ser vistos os diversos casos de uso e as relações entre eles, bem como a relação entre estes casos de uso e os atores, administrador e utilizador.

[25] http://lucene.apache.org/solr/

[26] http://www.oracle.com/technetwork/java/javase/jdbc/index.html - API responsável pela execução de comandos SQL através da linguagem de programação java

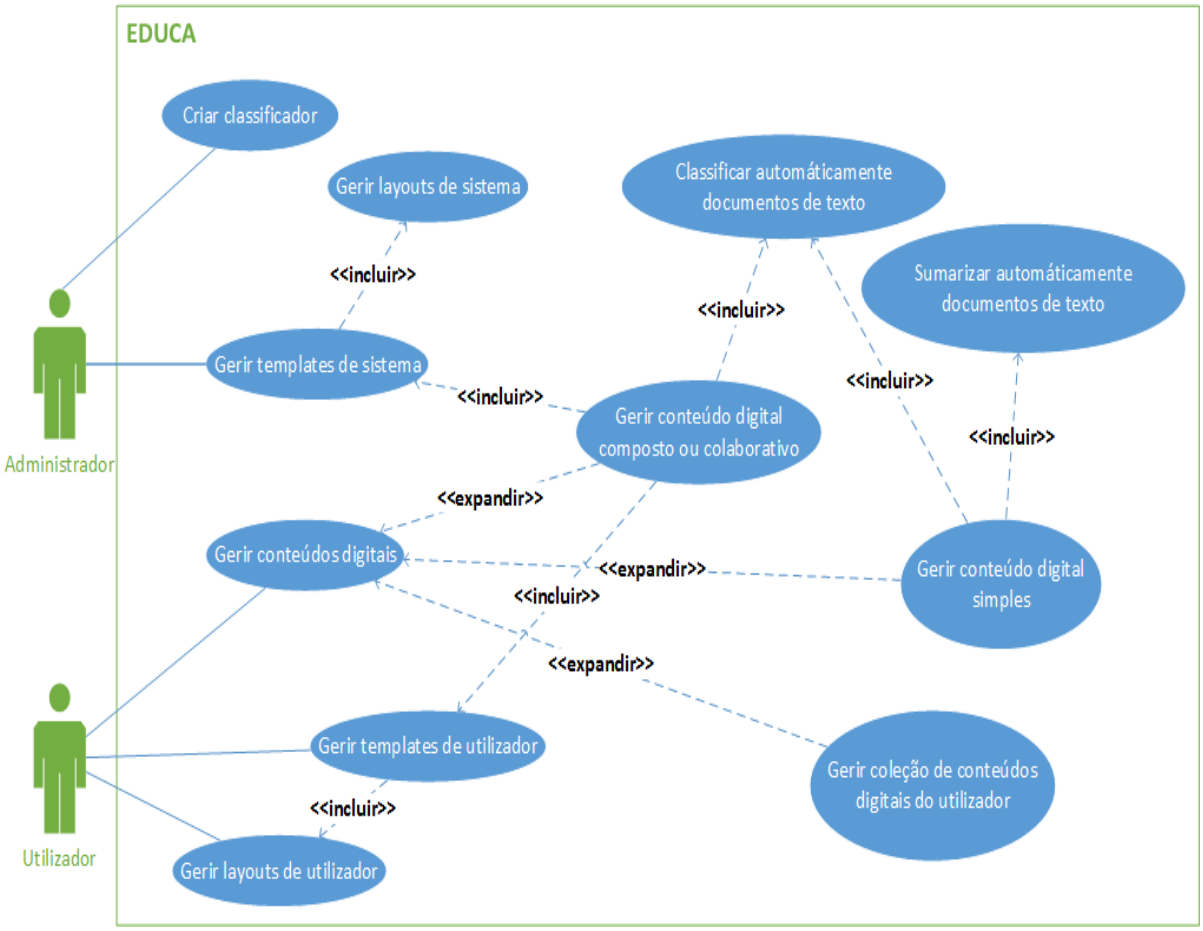



Figura 10 – Diagrama de casos de uso

### 4.2.2. Diagrama ER

Na Figura 11 pode ser visto o diagrama *ER* que dá suporte à implementação dos requisitos descritos no capítulo 4.1. Este diagrama apresenta as entidades necessárias para criação de *templates*, *layouts* e conteúdos compostos ou colaborativos.

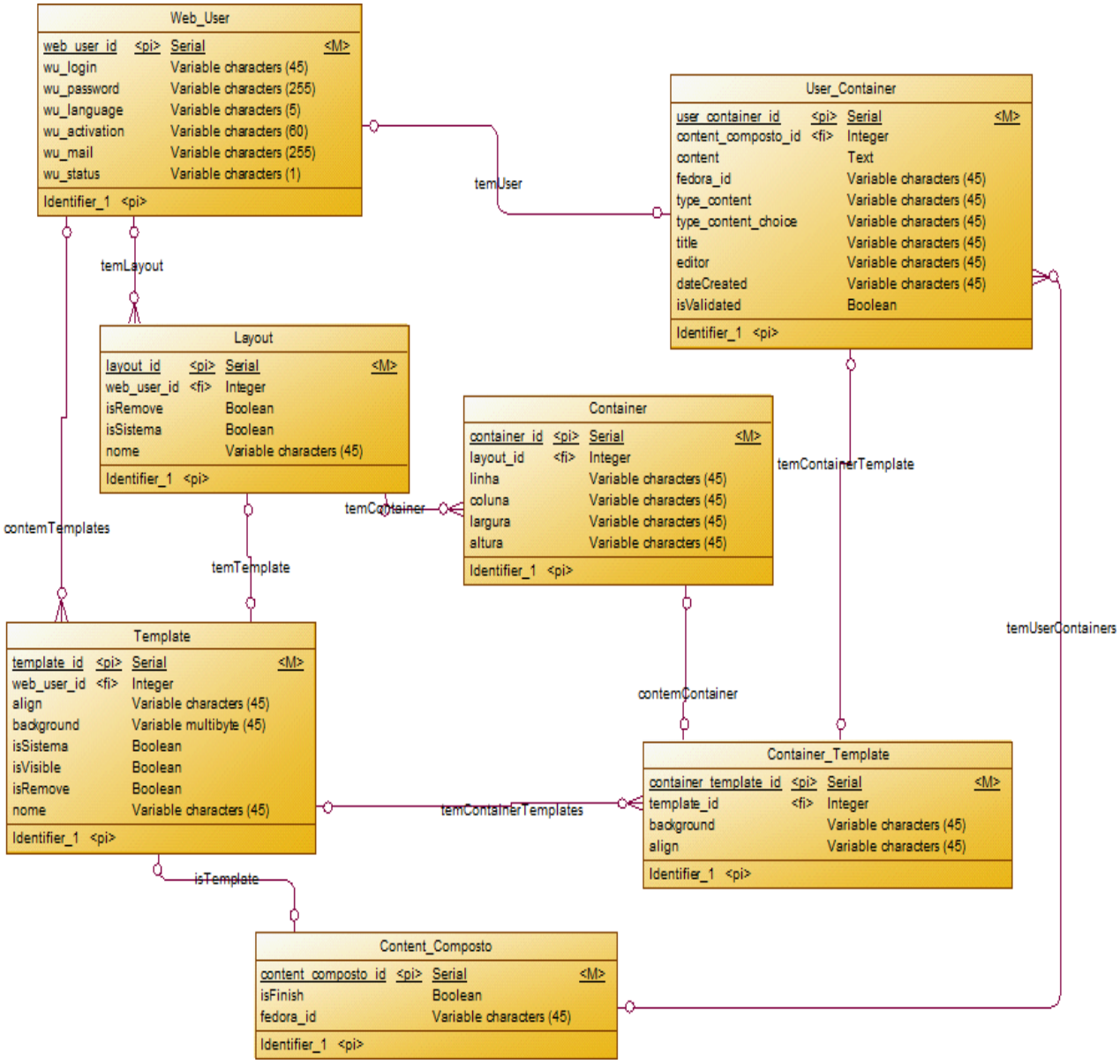



Figura 11 – Diagrama ER da parte relativa ao módulo a desenvolver

O diagrama apresentado na Figura 11 apenas contempla a parte implementada pelo estagiário, uma vez que o diagrama ER global contém 53 entidades e a sua inclusão na Figura tornava a sua leitura impercetível.

# Capítulo 5
# Implementação e testes

Ao longo deste capítulo será apresentada de forma sumária, a implementação feita pelo estagiário, começando pela descrição da implementação do módulo **educa-content**, depois o **educa-classification** e, por fim, o módulo **educa-summarization.** No final será apresentado, em termos de requisitos, o que se encontra e o que não se encontra feito. O Anexo B do presente relatório contém um manual do utilizador com a explicação da aplicação na perspetiva do utilizador, da parte que foi implementada.

## 5.1. Módulo educa-content

O módulo **educa-content** é o responsável por toda a gestão de *layouts*, *templates* e conteúdos digitais. Para ser mais percetível para o utilizador como foi elaborado o desenvolvimento deste módulo, foi decidido dividir em três subcapítulos onde será retratado primeiro a gestão de *layouts*, depois a gestão de *templates,* acabando na gestão de conteúdos digitais. A gestão de conteúdos digitais será dividida em 3 tipos: conteúdo simples, conteúdo composto ou colaborativo e coleção de conteúdos do utilizador.

### 5.1.1. Gestão de *layouts*

O desenvolvimento da gestão de *layouts* foi feito através da criação de diferentes páginas xhtml. Nestas páginas é possível criar, visualizar e listar *layouts*. No caso da criação do *layout* foi utilizado um *plugin* de *jquery*, *gridster*.js[27]. Este *plugin* permite o *drag'n'drop* e redimensionamento dos painéis (*containers*). No desenvolvimento da gestão de layouts ainda é utilizado um *Spring Service* e *Spring DAO* para fazer a navegação e o acesso aos dados, respetivamente. Para utilizar o *Spring DAO* foi necessário incluir o módulo **educa-datamodel,** pois é neste módulo que estão contidos os objetos que são mapeados da base de dados.

De forma a comtemplar as mesmas tarefas para o perfil de administração sem recorrer a repetição de código, os métodos utilizados são os mesmos, existindo uma *flag* booleana a indicar se são ou não *layouts* de sistema, onde *true* corresponde a ser do sistema e *false* do utilizador.

### 5.1.2. Gestão de *templates*

O desenvolvimento da gestão de *templates* foi feito através da criação de algumas páginas *xhtml*. A página utilizada para listar *layouts* é a mesma utilizada para listar os *templates*, onde é usado um componente de *richFaces*, com a função de mostrar ou ocultar a lista que é pretendida visualizar. Neste caso foram ainda criadas páginas para criar e ver *templates*. Para criar um *template,* existiram certos aspetos a ter em conta: primeiro é a escolha de um *layout*, onde foi criado um *select box* com os *layouts* que o utilizador pode escolher para formar um *template*; o segundo diz respeito à possibilidade de alterar a cor do fundo quer do *layout* em si, quer de cada painel (*container*). Para realizar a tarefa de mudança de cor dos fundos, *layout* e painel, foi utilizado um *plugin* de *jquery*, *spectrum*.js[28].

Por último, um aspeto importante, é o permitir ao utilizador acompanhar todas as mudanças que estão a ocorrer, pois qualquer alteração que o utilizador faça é logo vista, como se estivesse já finalizado o *template*.

Na implementação da gestão de *templates,* foi necessário a utilização de um *Spring Service* e *Spring DAO* para fazer a navegação e gestão de dados dentro da base de dados, respetivamente.

[27] http://gridster.net/
[28] http://bgrins.github.io/spectrum/

Para utilizar o *Spring DAO* foi necessário incluir o módulo **educa-datamodel**, pois é neste módulo que ficam mapeados os objetos vindos da base dados.

Como no caso da gestão de *layouts*, a gestão de *templates* tem também uma parte para o perfil de administração, fazendo com que toda a implementação seguisse a mesma abordagem feita na gestão dos *layouts*, replicando as páginas para o *backoffice* (administrador) e, nos métodos, é introduzida uma *flag* booleana a indicar o perfil de utilização.

### 5.1.3. Gestão de conteúdos digitais

O desenvolvimento da gestão de conteúdos digitais englobou a gestão de conteúdo digital simples, conteúdo digital composto ou colaborativo e coleção de conteúdos digitais do utilizador. A listagem destes três tipos de conteúdos é feita numa única página xhtml.

#### *5.1.3.1. Gestão de conteúdos digitais simples*

A implementação da gestão de conteúdos digitais simples teve grande importância no projeto, pois foi com esta implementação que foi possível avaliar a implementação dos módulos **educa-classification** e **educa-summarization.** Esta implementação cobriu os requisitos de criação e listagem de conteúdos simples. Assim, para se conseguir criar e listar estes conteúdos digitais simples, foi necessário fazer uma página xhtml, para a criação de um conteúdo digital simples.

Na página da criação do conteúdo digital simples existe um campo para o utilizador fazer o *upload* de um ficheiro (imagem, documento de texto, áudio, vídeo, etc.). Quando o utilizador faz o *upload* do ficheiro, é iniciada a análise da informação, ou seja, extração de metadados. Caso o ficheiro seja um documento de texto, é feita também a extração deste texto e depois a categorização e sumarização automática. A análise da informação é auxiliada pelos módulos **educa-analysis** para fazer a análise do texto; **educa-classification** para fazer a classificação do texto; e **educa-summarization** para sumarizar o texto**.** A criação do conteúdo simples ainda necessita da utilização do módulo **educa-catalog** para fazer a publicação do conteúdo digital simples no repositório. A navegação entre páginas é assegurada por um *Spring Service*.

#### *5.1.3.2. Gestão de conteúdos digitais compostos ou colaborativos*

A implementação da gestão de conteúdos digitais compostos ou colaborativos engloba dois perfis de utilizadores:

- Proprietário – Utilizador que cria o conteúdo composto ou colaborativo e que valida o que os colaboradores introduzem neste mesmo conteúdo.
- Colaborador – Utilizador que preenche o conteúdo composto ou colaborativo.

O proprietário do conteúdo composto ou colaborativo pode também ser colaborador, contribuindo também no preenchimento do mesmo.
A implementação da gestão de conteúdos digitais compostos ou colaborativos teve a necessidade de implementar algumas páginas xhtml:

- **Criação de conteúdo composto ou colaborativo** – onde o proprietário escolhe o *template* que será utilizado, os utilizadores que irão colaborar no preenchimento deste conteúdo e o tipo de dados que poderão ser inseridos pelos colaboradores (escrever texto, conteúdo simples inseridos no repositório);
- **Preenchimento do conteúdo composto ou colaborativo** – onde é feito o preenchimento do conteúdo por parte dos utilizadores escolhidos para colaborarem;
- **Pré-visualização do conteúdo preenchido** – onde o colaborador visualiza como ficou o conteúdo que introduziu;

- **Validação dos conteúdos inseridos pelos colaborados** – onde o proprietário aprova ou rejeita os conteúdos inseridos pelos colaboradores. No caso do proprietário rejeitar o conteúdo, este não irá fazer parte do conteúdo composto ou colaborativo final que irá ser publicado no repositório;
- **Finalização do conteúdo composto ou colaborativo –** onde o proprietário finaliza e publica o conteúdo composto ou colaborativo.

A navegação destas páginas é assegurada por um *Spring Service*. Já o acesso aos dados necessários nestas páginas é feito por um *Spring DAO*. Na implementação deste tipo de conteúdo, foi necessário utilizar outros módulos: **educa-datamodel** onde se encontram mapeados os objetos que o *Spring DAO* necessita; **educa-classification** para fazer a classificação dos conteúdos que sejam de texto; e o **educa-catalog** para aceder ou publicar os conteúdos no repositório.

#### *5.1.3.3. Gestão de coleção de conteúdos digitais do utilizador*

Na implementação da gestão de coleção de conteúdos digitais do utilizador, foi necessário criar duas páginas xhtml: uma, onde o utilizador escolhe os conteúdos que pretende englobar na coleção; e outra onde o utilizador diz quais os utilizadores ou grupos que têm permissões para aceder à coleção a ser criada. Além destas páginas xhtml, foi necessário também criar um *Spring Service* que faz a navegação entre as páginas e foi utilizado o módulo **educa-catalog** para aceder aos conteúdos existentes no repositório e, listá-los, para o utilizador escolher quais é que irão fazer parte da coleção.

## 5.2. Módulo educa-classification

A implementação do módulo **educa-classification**, responsável pela classificação automática de documentos de texto, seguiu uma abordagem de *machine learning*. A ferramenta escolhida foi a *MAchine Learning for LanguagE Toolkit* (*MALLET*), devido a ter sido a primeira a ser testada de entre as descritas no estado de arte, pelo que satisfez o valor definido de sucesso.
Dado a classificação ser automática, mas existir uma decisão final por parte do utilizador, definiu-se um parâmetro de qualidade aceitável. O valor aceitável para o classificador foi de 70% da precisão global. Este valor foi escolhido de uma forma empírica pela empresa, dado a classificação não ser final, tendo como última palavra o utilizador. Para se alcançar este valor teve que ser definido um corpus de documentos de texto previamente classificados dentro das seguintes categorias:

- Arte e desporto;
- Ciências aplicadas e medicina;
- Ciências exatas e naturais;
- Ciências sociais;
- Ciências da tecnologia e conhecimento;
- Filosofia e psicologia;
- Geografia e história;
- Linguística e literatura;
- Religião e teologia.

Seguindo para a estrutura da implementação, podemos separá-la em duas fases: a fase de treino com o corpus de documentos de texto, e a fase que promove a classificação. Para se executar o treino do texto extraído dos diferentes documentos, é necessário fazer um pré-processamento deste texto. Este pré-processamento passa pela tokenização, normalização dos *tokens* para letra minúscula, remoção das *stop words*. Seguidamente estes *tokens* são convertidos para inteiros. Estes inteiros correspondem aos índices onde ocorrem estes *tokens* dentro de uma

sequência. Por fim são contabilizadas o número de vezes que um determinado *token* ocorre dentro da sequência anterior, criando um novo vetor com os índices e o contador de ocorrência de *tokens*. Após ser feita o pré-processamento é aplicado o algoritmo – o escolhido foi o da Máxima Entropia – sendo este o algoritmo responsável pela "aprendizagem", para poder decidir, mais tarde, em qual categoria irá classificar os documentos de texto. Desta forma, obtém-se o classificador utilizado na próxima fase.

A próxima fase, a da classificação, utiliza o classificador gerado pelo treino e o texto que se quer classificar. Este classificador vai pré-processar o texto do documento que é pretendido classificar e irá dar-lhe uma probabilidade de ser de uma determinada categoria. Neste caso foi definido um valor limiar, pois os textos podem ter mais que uma categoria e este limiar define o número de categorias em que o documento se insere.

Os desenvolvimentos destes algoritmos foram integrados no módulo **educa-classification,** onde foi criado um *Spring Service,* para ser possível iniciar o treino do classificador. No caso do treino do classificador, foi criado uma *thread* que executa o treino em *background* assim que o administrador o inicia dentro do *backoffice (*zona de administração do sistema*)*. Logo que é iniciado o treino, não é permitido que sejam executados outros treinos sem que acabe o que está em execução, evitando a concorrência de *threads*. A existência de concorrência de *threads* irá reduzir a performance da aplicação, podendo também levar à ocorrência de erros de memória.

## 5.3. Módulo educa-summarization

A implementação do módulo **educa-summarization**, responsável pela sumarização automática de documentos de texto, para conseguir executar a tarefa pretendia começa por fazer um pré-processamento do texto e, só depois, é que é aplicado um método de sumarização automática. Este pré-processamento do texto pode ser dividido em várias fases. Numa primeira fase o texto é separado em frases, com o auxílio do openNLP. Após as frases estarem separadas, passa-se para uma nova fase onde vão ser inseridas na ferramenta Lucene, pelo que aqui irão ocorrer quatro subfases:

- Separação de frases em palavras;
- Remoção das stop words;
- Lematização;
- Contagem do número de vezes que uma palavra ocorre numa frase, e no texto.

As fases, anteriormente descritas, utilizam um analisador de língua portuguesa, pois todo o texto a ser sumarizado está escrito em língua portuguesa.

Na implementação da sumarização automática definiram-se três métodos:

- Palavras-chave;
- TF-ISF[29];
- TF-ISF + Localização.

O primeiro método parte do princípio de que as palavras que mais vezes ocorrem no texto são as palavras-chave, pelo que as frases onde elas ocorrem são consideradas importantes. Assim sendo, vai juntar as frases onde ocorrem estas palavras e formar o sumário.

O segundo método parte do conceito de que quanto mais representativas forem as palavras numa frase, mais importante ela será no texto. Assim, utilizou-se o algoritmo do TF-ISF (Term Frequency – Inverse Sentence Frequency). Portanto, a importância de uma palavra w em uma frase s, TF-ISF(w,s), é calculada através da fórmula:

$$TF - ISF(w,s) = TF(w,s) \times ISF(w),$$

[29] Term Frequency – Inverse Sentence Frequency

onde TF(w,s) é o número de vezes que a palavra w ocorre na frase s, e a frequência inversa da sentença é obtida através da fórmula:

$$ISF(w) = \log \frac{|S|}{SF(w)},$$

onde |S| corresponde ao número de frases que existem no texto e SF(w) é o número de vezes que a palavra w ocorre no texto.
O valor utilizado nesta abordagem é obtido através da fórmula:

$$Score(s) = \sum TF \times ISF(w, s),$$

obtendo-se assim o peso da frase no texto.

No caso do último método, para além da utilização da fórmula descrita anteriormente, ainda se considerou que nas frases iniciais e finais existe uma maior probabilidade de se encontrar as frases mais importantes do texto. Aqui, para aumentar a probabilidade de ser escolhida uma frase inicial, utilizou-se uma multiplicação com valores entre 0.5 e 1, pelo que nas frases iniciais se multiplicava por 1 e, conforme se ia avançando nas frases, ia decrescendo o valor até 0.5. Este valor volta a ser incrementado assim que se chega ao número médio de frases do texto, fazendo com que as frases finais também viessem a ter um valor a multiplicar por 1.

## 5.4. Testes

Os testes representam uma parte fulcral no desenvolvimento de qualquer aplicação. Esta fase permite não só detetar erros que não são percetíveis aquando da implementação, mas também fornecer alguma garantia de que a aplicação, quando instalada num novo ambiente, possa funcionar de forma correta. Partindo deste pressuposto, o estagiário considerou essenciais dois tipos de testes:

- Testes de aceitação – são testes em que o cliente valida se a funcionalidade se encontra implementada de acordo com o que era expectável;
- Testes de sistema – são testes que são realizados pelo programador, do ponto de vista de utilizador final, utilizando a aplicação nas mesmas condições que o utilizador final.

No caso dos testes de sistema, foram feitos em dois ambientes distintos. O primeiro ambiente tinha as seguintes características:

- Processador intel core i7 1,6GHz quad-core;
- 6GB de RAM;
- 500GB de disco;
- Sistema operativo: Windows 7 64 bits;
- JDK 6 de 64 bits;
- Tomcat 7;

O segundo ambiente tinha as características seguintes:

- Processador Intel(R) Xeon(R) CPU E5504 @ 2.00GHz quad-core;
- 4GB de RAM;
- 250GB de disco;
- Sistema operativo: CentOS 6.3 de 64 bits;
- JDK 6 de 64 bits;
- Tomcat 7;

Estes testes consistiram em testar o requisito quando implementado, validando assim se estava de acordo ou não com o que era expectável, sendo que, como são dois ambientes diferentes, o

que estava a funcionar localmente (Windows 7), poderia não se encontrar a funcionar no ambiente do servidor (CentOS 6.3).
O estagiário criou um documento com os testes de aceitação que se encontram no Anexo C. Estes testes encontram-se a ser executados por *testers* das empresas Flor de Utopia e Metatheke.

## 5.5. Requisitos implementados e não implementados

Na Tabela seguinte é apresentada uma lista dos requisitos que foram recolhidos pelo estagiário, indicando para cada um o estado (se implementado ou não). O ordenamento foi baseado primeiro na Prioridade do Estágio e depois Prioridade do Projeto EDUCA, aparecendo a seguir estas prioridades separadas por vírgula.

- Must, Must;
- Should, Must;
- Could, Must;
- Should, Should;
- Could, Should;
- Could, Could;

| Código | Descrição | Prioridade Estágio | Prioridade Projeto EDUCA | Estado |
|---|---|---|---|---|
| **RF 1.1.1** | Criar conteúdo simples | Should | Must | Implementado |
| **RF 1.1.2** | Editar conteúdo simples | Could | Should | Não implementado |
| **RF 1.1.3** | Remover conteúdo simples | Could | Should | Não implementado |
| **RF 1.1.4** | Listar conteúdos simples | Should | Must | Implementado |
| **RF 1.2.1** | Criar conteúdo composto ou colaborativo | Could | Must | Implementado |
| **RF 1.2.2** | Editar conteúdo composto ou colaborativo | Could | Should | Não implementado |
| **RF 1.2.3** | Remover conteúdos compostos ou colaborativos | Could | Should | Não implementado |
| **RF 1.2.4** | Listar conteúdo composto ou colaborativo | Could | Must | Implementado |
| **RF 1.2.5** | Pré-visualizar conteúdo composto ou colaborativo | Could | Must | Implementado |
| **RF 1.2.6** | Adicionar permissão a utilizadores para | Could | Must | Implementado |

| | | | | |
|---|---|---|---|---|
| | a colaboração do preenchimento do conteúdo | | | |
| **RF 1.2.7** | Permitir a escolha de um ou mais tipos de conteúdos a inserir | Could | Must | Implementado |
| **RF 1.2.10** | Remover conteúdo preenchido | Could | Must | Implementado |
| **RF 1.2.11** | Validação do conteúdo preenchido em cada painel | Could | Must | Implementado |
| **RF 1.2.12** | Finalização do conteúdo composto ou colaborativos | Could | Must | Implementado |
| **RF 1.2.8.1, RF 1.2.9.1** | Adicionar texto ao conteúdo composto | Could, Could | Must, Should | Implementado |
| **RF 1.2.8.2, RF 1.2.9.2** | Adicionar conteúdo simples de áudio | Could, Could | Must, Should | Não implementado |
| **RF 1.2.8.3** | Adicionar conteúdo simples de video | Could, Could | Must, Should | Não implementado |
| **RF 1.2.8.4** | Adicionar conteúdo simples documento texto/imagem | Could, Could | Must, Should | Não implementado |
| **RF 1.3.1** | Criar coleção de conteúdo | Could | Must | Implementado |
| **RF 1.3.2** | Editar coleção de conteúdo | Could | Should | Não implementado |
| **RF 1.3.3** | Remover coleção de conteúdo | Could | Should | Não implementado |
| **RF 1.3.4** | Listar as coleções de conteúdo | Could | Must | Implementado |
| **RF 1.4.3** | Remover layout | Could | should | Implementado |
| **RF 1.4.4** | Listar layouts | Should | Must | Implementado |
| **RF 1.4.5** | Pré-visualizar layout | Should | Must | Implementado |
| **RF 1.4.1.1, RF 1.4.2.1** | Adicionar painel (*container*) | Could, Could | Must, should | Implementado, Não implementado |

| | | | | |
|---|---|---|---|---|
| **RF 1.4.1.2, RF 1.4.2.2** | Redimensionar painel (*container*) | Could, Could | Must, should | Implementado, não implementado |
| **RF 1.4.1.3, RF 1.4.2.3** | Remover painel (*container*) | Could, Could | Must, should | Implementado, não implementado |
| **RF 1.5.3** | Remover template | Could | should | Não implementado |
| **RF 1.5.4** | Listar templates | Could | Must | Implementado |
| **RF 1.5.5** | Pré-visualizar template | Could | Must | Implementado |
| **RF 1.5.1.1, RF 1.5.2.1** | Adicionar cor de fundo ao template | Could, Could | Must, should | Implementado, não implementado |
| **RF 1.5.1.2, RF 1.5.2.2** | Adicionar cor de fundo a cada painel (container) | Could, Could | Must, should | Implementado, não implementado |
| **RF 1.5.1.3, RF 1.5.2.3** | Escolher alinhamento do conteúdo dentro do painel | Could, Could | Must, should | Implementado, não implementado |
| **RF 1.5.1.1, RF 1.5.2.1** | Adicionar cor de fundo ao template | Could, Could | Must, should | Implementado, não implementado |
| **RF 2.1** | Iniciar treino do classificador | Must | Must | Implementado |
| **RF 2.2** | Classificar automaticamente o conteúdo | Must | Must | Implementado |
| **RF 3** | Sumarização automática de documentos de texto | Must | Must | Implementado |

**Tabela 17 – Lista de requisitos recolhidos pelo estagiário**

# Capítulo 6
# Experimentação

Ao longo deste capítulo serão descritos a experimentação relativa à classificação e sumarização de documentos de texto. Esta experimentação englobou alguns testes, que foram realizados no mesmo ambiente, numa máquina com as seguintes características:

- Processador core i7 1,6GHz quad-core;
- 6GB de RAM;
- 500GB de disco;
- Sistema operativo: Windows 7 de 64 bits.

Nos subcapítulos seguintes serão descritos os diferentes testes feitos para avaliar as abordagens implementadas na classificação e sumarização automática de documentos de texto.

## 6.1. Classificação

A classificação automática de documentos de texto utiliza uma abordagem de *machine learning*, pelo que para desenvolver esta abordagem é necessário escolher primeiro qual o algoritmo que devolve melhores valores em termos de qualidade e performance, bem como o *corpus*. Assim sendo, e visto que já existem ferramentas que utilizam esta abordagem, testou-se a *MAchine Learning for LanguagE Toolkit (MALLET)* que já contém alguns algoritmos capazes de classificar textos automaticamente. Desta forma apenas era necessário avaliar estes algoritmos e verificar qual apresentava os melhores resultados.
Os testes que aqui foram realizados com o intuito de avaliar esta ferramenta foram:

- Treino
  - **Qualidade do classificador** – onde foi avaliado a precisão (precision), abrangência (*recall)* e medida-f (*f-measure)*.
  - **Performance do classificador** – onde é avaliado o tempo que o classificador demora no treino.
- Classificação
  - **Performance do classificador** – tempo que o classificador demora a classificar documentos de texto.

Os algoritmos que foram testados foram:

- Naïve Bayes;
- Árvores de Decisão (ID3 simplificado);
- Máxima Entropia.

Foi ainda testado o C45 que também pertence às árvores de decisão, no entanto, não foi possível obter resultados devido ao tempo necessário para treinar o classificador.

### 6.1.1. Testes de treino

Os testes de treino foram realizados com um *dataset* de 900 documentos de texto das categorias:

- Arte e desporto;
- Ciências aplicadas e medicina;

- Ciências exatas e naturais;
- Ciências sociais;
- Ciências da tecnologia e conhecimento;
- Filosofia e psicologia;
- Geografia e história;
- Linguística e literatura;
- Religião e teologia.

Ficando em cada categoria 100 documentos de texto, de modo a se ter um *dataset* o mais semelhante possível em termos de quantidade de documentos. Todos os testes realizados foram executados 35 vezes, sendo descartados os primeiros 5 de modo a evitar o *warmup*.

#### *6.1.1.1. Qualidade*

Os testes de qualidade foram executados utilizando *cross-folding,* ou seja, é escolhida do *dataset* de forma aleatória uma percentagem para treinar e outra para validar a qualidade de treino, sendo este processo repetido o número de *folds* que se quiser; no caso presente o *cross-folding* foi executado 10 vezes. Os resultados obtidos, que são apresentados no Gráfico 6, correspondem à média das 10 *folds*. Além ser feito o *cross-folding, os testes* foram ainda executados 30 vezes, retirada a média da precisão, abrangência e medida-f de cada algoritmo.

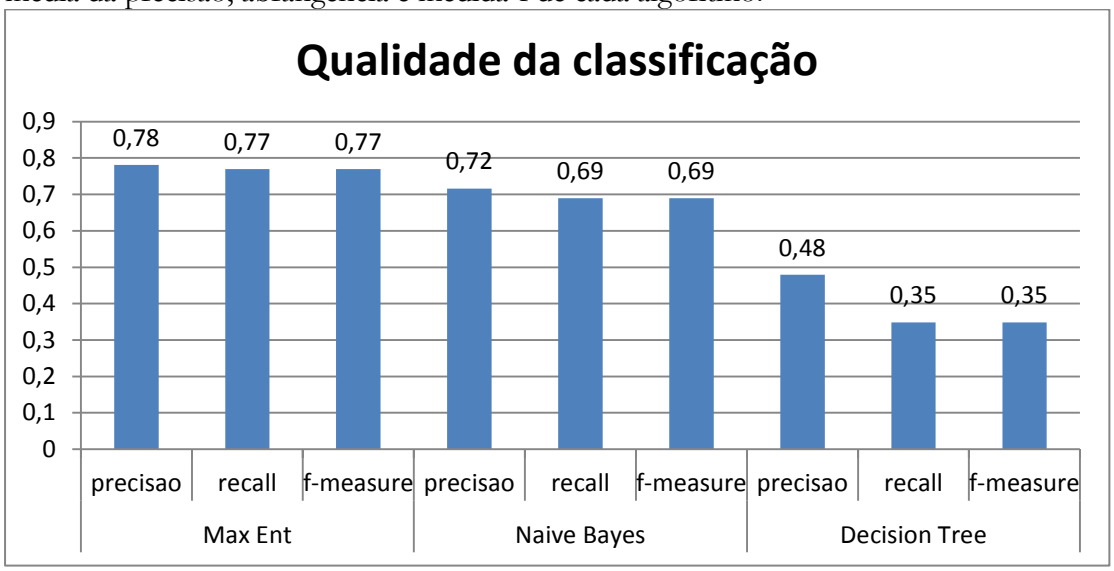



Gráfico 6 - Qualidade dos diferentes algoritmos no treino

Avaliando os resultados expressos no gráfico pode ser visto que o algoritmo Árvores de decisão obtém os piores resultados não conseguindo alcançar os 0.5. O algoritmo Máxima Entropia, obtém os melhores resultados atingindo na precisão quase os 0.8, enquanto o algoritmo Naïve Bayes chegou aos 0.72 aproximadamente. Assim, a partir destes resultados, pode-se concluir que o algoritmo de Máxima entropia é o melhor em termos de qualidade.

#### *6.1.1.2. Performance*

Nos testes de performance, que serão apresentados, foi descartado o tempo de extração de dados de cada ficheiro, apenas é contabilizado o tempo que os algoritmos demoram a treinar o classificador. Os resultados da performance podem ser vistos na Tabela 18 e no Gráfico 7, pelo que os valores presentes no gráfico dizem respeito à média das 30 vezes que os algoritmos foram executados. No total os algoritmos foram executados 35 vezes, mas os 5 primeiros não foram incluídos para evitar o *warmup*.

| | Máxima Entropia | Naïve Bayes | Árvores de Decisão |
|---|---|---|---|
| Média | 134523 | 218,4333333 | 4250,166667 |
| Desvio Padrão | 2917,47 | 17,04441909 | 414,2591044 |
| Intervalo de Confiança | 1043,98 | 6,099154965 | 148,2379927 |

Tabela 18 – Valores da performance de treino do classificador em milissegundos

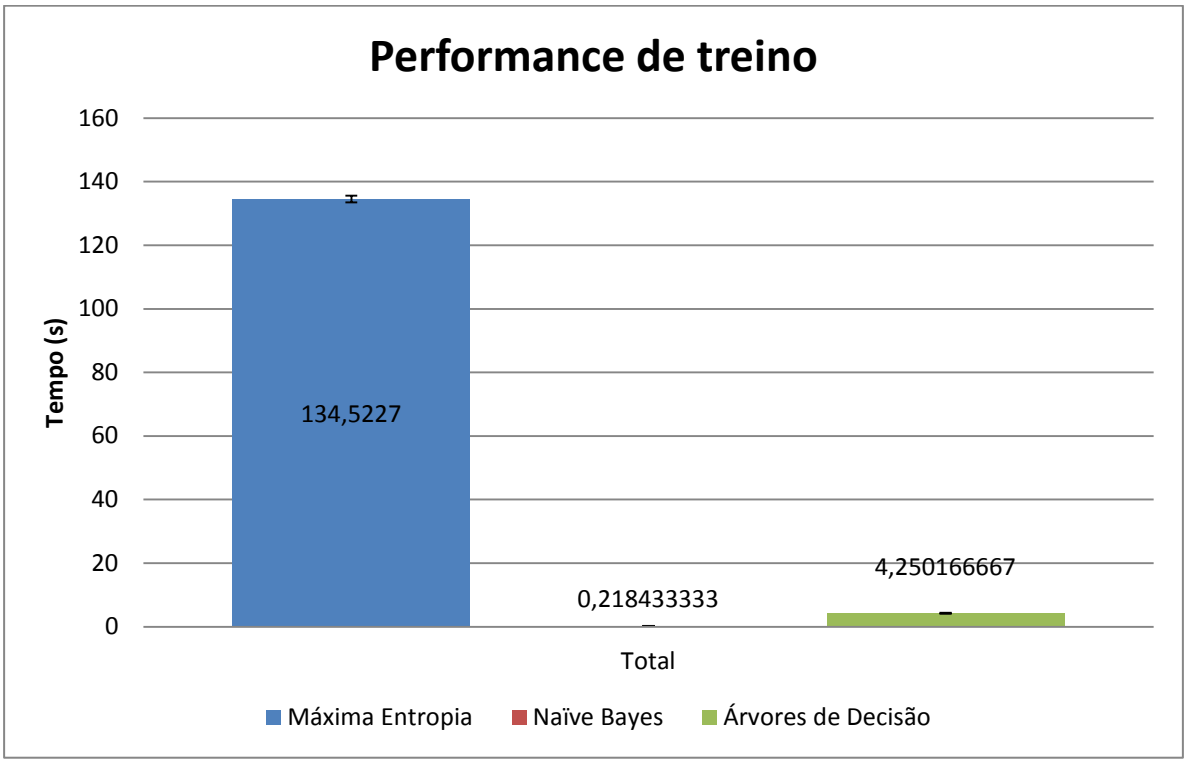



Gráfico 7 – Performance de treino dos diferentes algoritmos

Observando os valores presentes neste gráfico, pode dizer-se que o algoritmo que demorou menos tempo no treino foi o *Naïve Bayes*, não chegando a demorar um segundo. O algoritmo de Árvores de Decisão também apresentou um desempenho bom, embora mais lento que o *Naïve Bayes*. Por fim, o algoritmo Máxima Entropia, que anteriormente apresentou os melhores resultados, demorou mais de 2 minutos para executar o treino do classificador.

Perante estes resultados, pode-se concluir que, em termos de performance de treino, o melhor algoritmo foi o *Naïve Bayes*.

### 6.1.2. Testes de classificação

Relativamente aos testes de classificação, apenas foi considerado o tempo que o classificador demorou a classificar um documento. Neste caso, validar apenas um documento de cada vez, era pouco, então decidiu-se começar por classificar 100 de uma vez, incrementando 100 até perfazer no 900 documentos. A Tabela 19 e o Gráfico 8 mostram os resultados obtidos neste teste.

| | | **Máxima Entropia** | **Naïve Bayes** | **Árvores de Decisão** |
|---|---|---|---|---|
| **100 documentos** | Média | 1000,5 | 1010,333333 | 985,4 |
| | Desvio Padrão | 39,2893 | 33,38995192 | 31,75489464 |
| | Intervalo de Confiança | 14,0592 | 11,94822129 | 11,3631343 |
| **200** | Média | 1791,866667 | 1788,733333 | 1760,833333 |

| | | | | |
|---|---|---|---|---|
| **documentos** | | | | |
| | Desvio Padrão | 117,6256019 | 61,30466721 | 102,435698 |
| | Intervalo de Confiança | 42,09100761 | 21,93719031 | 36,65547019 |
| **300 documentos** | Média | 3438,8 | 3468,4 | 3337,333333 |
| | Desvio Padrão | 191,827 | 162,6359124 | 136,2970612 |
| | Intervalo de Confiança | 68,6432 | 58,1974444 | 48,77238074 |
| **400 documentos** | Média | 4218,13 | 4155,833333 | 4090,466667 |
| | Desvio Padrão | 219,603 | 156,8634403 | 196,7807127 |
| | Intervalo de Confiança | 78,5826 | 56,13182973 | 70,41577975 |
| **500 documentos** | Média | 5207,8 | 5222,366667 | 5101,766667 |
| | Desvio Padrão | 128,143 | 109,0915467 | 107,4711382 |
| | Intervalo de Confiança | 45,8545 | 39,03719129 | 38,45734623 |
| **600 documentos** | Média | 6447,13 | 6468,7 | 6291,466667 |
| | Desvio Padrão | 330,221 | 375,7001597 | 326,3827746 |
| | Intervalo de Confiança | 118,166 | 134,4401051 | 116,7924298 |
| **700 documentos** | Média | 7268,133333 | 7300,166667 | 7057,5 |
| | Desvio Padrão | 313,66678 | 260,2514276 | 289,8045031 |
| | Intervalo de Confiança | 112,2421532 | 93,12806602 | 103,7033039 |
| **800 documentos** | Média | 8307,6 | 8446,8 | 8200,433333 |
| | Desvio Padrão | 262,248 | 324,4898971 | 274,9886887 |
| | Intervalo de Confiança | 93,8424 | 116,1150847 | 98,40163027 |
| **900 documentos** | Média | 10520 | 10589,76667 | 10186,5 |
| | Desvio Padrão | 682,802 | 693,4581306 | 301,7515258 |
| | Intervalo de Confiança | 244,333 | 248,1462452 | 107,9784126 |

**Tabela 19 – Valores da performance da classificação em milissegundos**

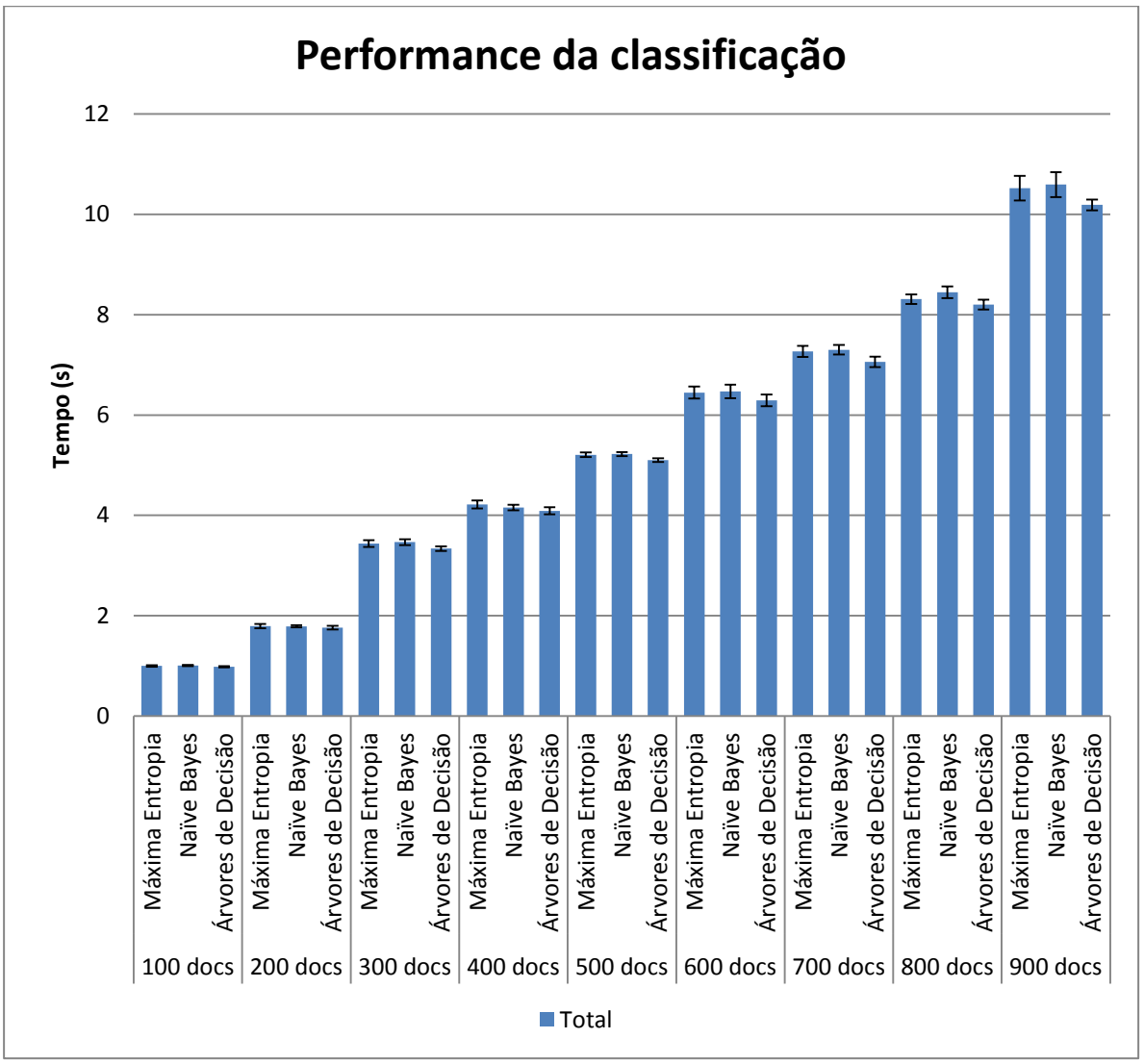



Gráfico 8 – Performance da classificação de múltiplos documentos

Observando os valores obtidos, pode-se constatar que ambos têm uma boa performance na classificação de múltiplos documentos, pois a variação existente entre eles é mínima. No entanto, dos três, o que apresenta melhores tempos é o algoritmo de Árvores de Decisão.
Após a observação de todos os resultados, pode-se constatar que o melhor algoritmo a utilizar é o Máxima Entropia, pois embora em termos de performance de treino não demonstre ser o melhor, pois demora muito tempo a treinar o classificador, em termos de qualidade apresenta os melhores resultados. Como o treino é para correr em *background*, não existe grande impedimento de usar este algoritmo.
Como o Algoritmo de Máxima Entropia conseguiu alcançar os valores definidos aceitáveis de qualidade, 70% da precisão global, decidiu-se não fazer testes a mais nenhuma ferramenta e utilizar a *MAchine Learning for LanguagE Toolkit (MALLET)*.

## 6.2. Sumarização automática de documentos de texto

Os testes de sumarização de documentos de texto tiveram como objetivo avaliar os três métodos implementados, com o intuito de se poder escolher um deles para utilizar no projeto. Para avaliar estes três métodos foram feitos três tipos de testes:

- **Avaliação de performance** – onde foram executadas 35 vezes a sumarização automática de 100 documentos de texto, sendo que os primeiros 5 foram descartados para evitar o *warmup*.
- **Avaliação de trigramas**[30] – para executar este teste utilizou-se o *Jaccard Similarity* (Semelhança de *Jaccard*)que utiliza a seguinte formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Onde A é o conjunto de trigramas do resumo original e B é o conjunto de trigramas do sumário gerado a comparar.

- **Avaliação de utilizadores** – nesta avaliação foi criada uma plataforma web para os utilizadores compararem o sumário escrito por uma pessoa com o sumário gerado de forma automática. Esta avaliação foi dividida em duas métricas:
  - ***Rating*** – onde o utilizador escolhia um valor entre 1 e 3:
    1. Corresponde a Mau, em que o sumário gerado é caracterizado pela falta de coerências na estrutura frásica e pela não transmissão das ideias chave transmitidas pelo resumo original;
    2. Corresponde a Suficiente, em que o sumário gerado contém uma estrutura frásica coerente e contém algumas ideias chave do resumo original, embora sejam poucas;
    3. Corresponde a Bom, em que o sumário gerado contém uma estrutura frásica coerente e, praticamente, todas as ideias chave apresentadas no resumo original.
  - **Ordenamento** – onde o utilizador ordenava os sumários gerados automaticamente, por ordem de preferência, ou seja, escolhia do que mais preferia para o que menos preferia, para substituir o sumário original.

### 6.2.1. Resultados da avaliação de performance

Os resultados obtidos nesta avaliação podem ser vistos no Gráfico 9.

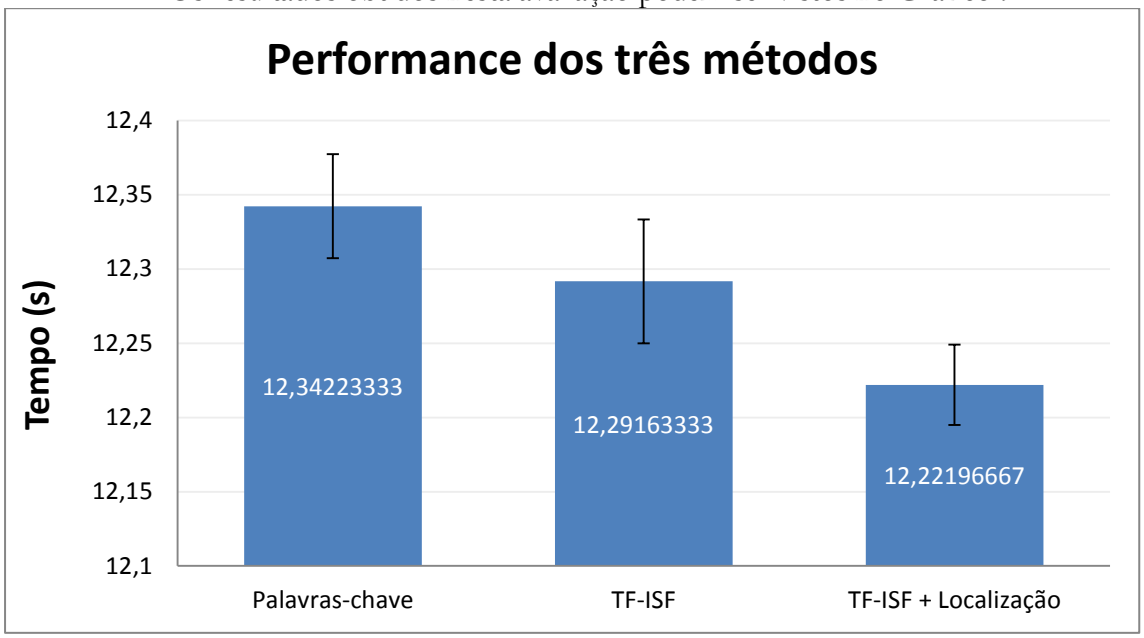



Gráfico 9 – Performance dos três métodos de sumarização automática

[30] Trigramas – são representações de parte de uma palavra com 3 caracteres. Por exemplo a palavra DATA pode ser representada pelos trigramas DAT, ATA.

Observando os resultados do Gráfico 9 pode-se constatar que o método TF-ISF + Localização é o que tem melhor performance, neste caso prático, na sumarização de 100 documentos de texto. No entanto, a discrepância de tempos entre os diferentes métodos é baixa.

### 6.2.2. Resultados da avaliação de trigramas

Os resultados dos trigramas podem ser vistos no Gráfico 10.

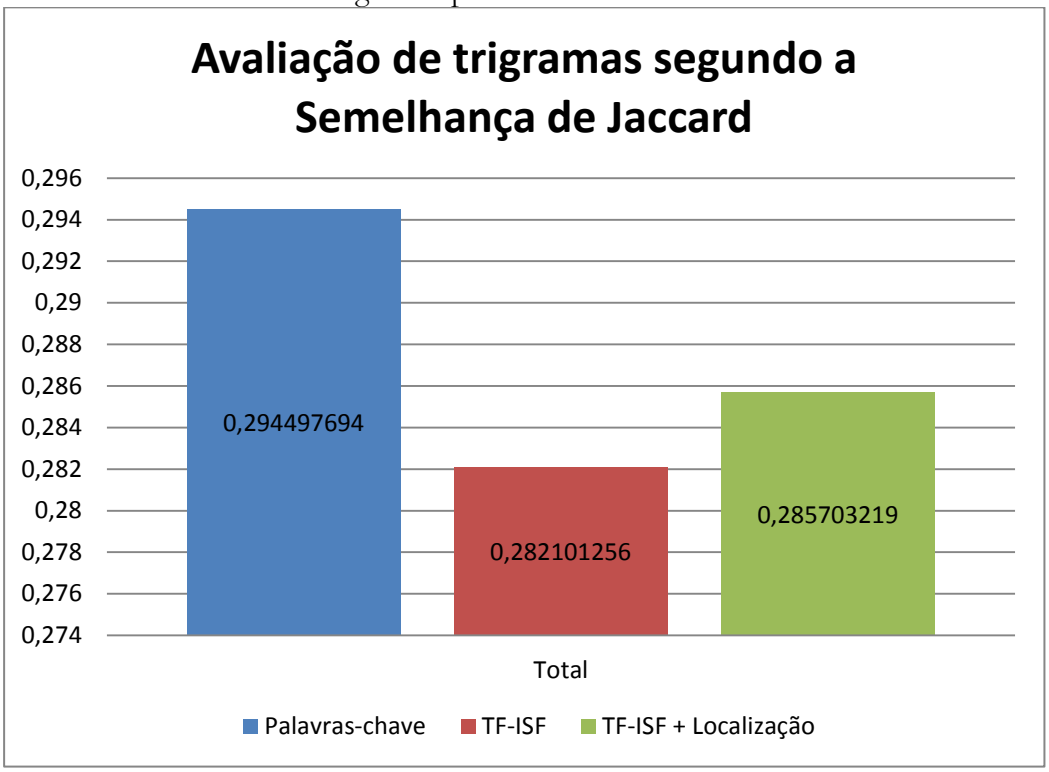



Gráfico 10 – Avaliação de trigramas segundo a semelhança de *Jaccard*

Observando os resultados obtidos, pode-se dizer que o sumário que tem maior semelhança com o sumário criado por uma pessoa, é o que foi gerado pelo método das palavras-chave.

### 6.2.3. Resultados da avaliação de utilizadores

Os resultados obtidos nesta avaliação podem ser vistos nos Gráficos 11 e 12, sendo que participaram nesta avaliação 10 pessoas.

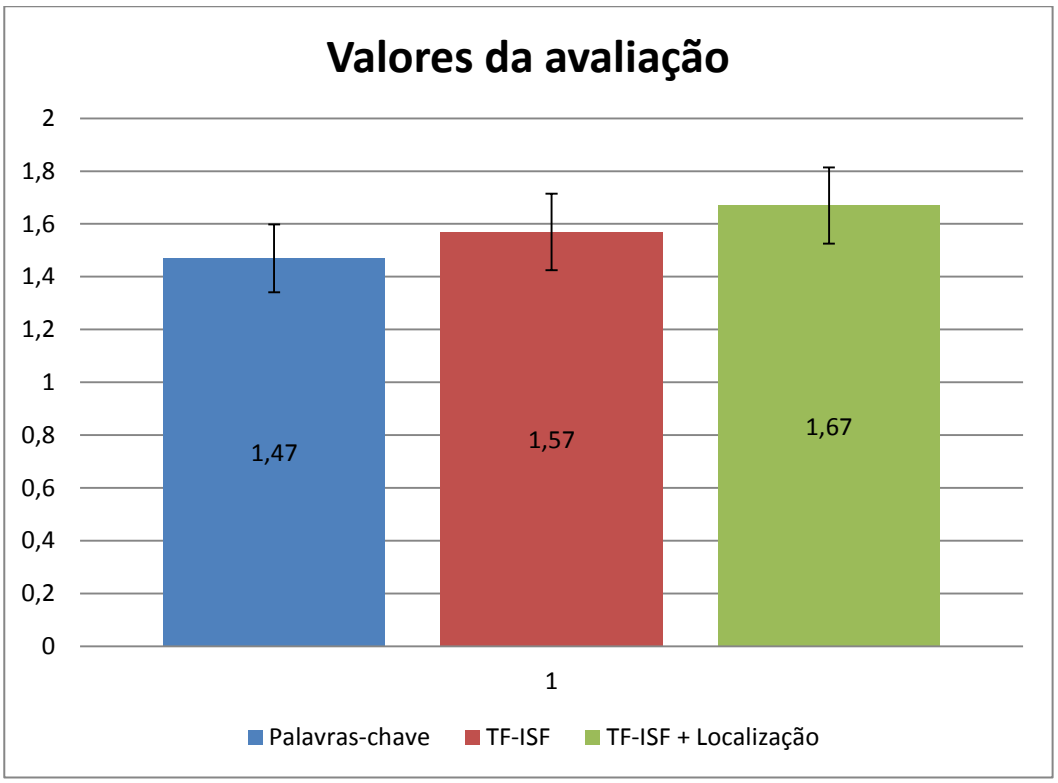



Gráfico 11 – Valores da avaliação dos utilizadores

Observando o Gráfico 11, o método TF-ISF + Localização tem maior *rating*, no entanto estatisticamente a diferença é ténue, pelo que se acrescentasse mais utilizadores ou mais sumários para analisar os resultados podiam ser obtidos valores diferentes. Além disto, utilizou-se ainda o algoritmo *Fleiss-Kappa* para avaliar o grau de concordância dos utilizadores em cada categoria. Os resultados obtidos foram 0,151 no método palavras-chave, 0,116 para o método TF-ISF e 0,023 para o método TF-ISF + localização. A interpretação destes valores pode ser vista na Tabela seguinte.

| **Kappa** | **Interpretação** |
|---|---|
| **< 0** | Concordância baixa |
| **0,01 – 0,20** | Concordância ligeira |
| **0,21 – 0,40** | Concordância razoável |
| **0,41 – 0,60** | Concordância moderada |
| **0,61 – 0,80** | Concordância substancial |
| **0,81 – 1** | Concordância quase perfeita |

Tabela 20 – Interpretação dos valores de *inter-raters agreement*

De acordo com os valores da tabela, a concordância em todas as categorias é ligeira, o que demonstra a subjetividade que o utilizador usou para avaliar os resumos.

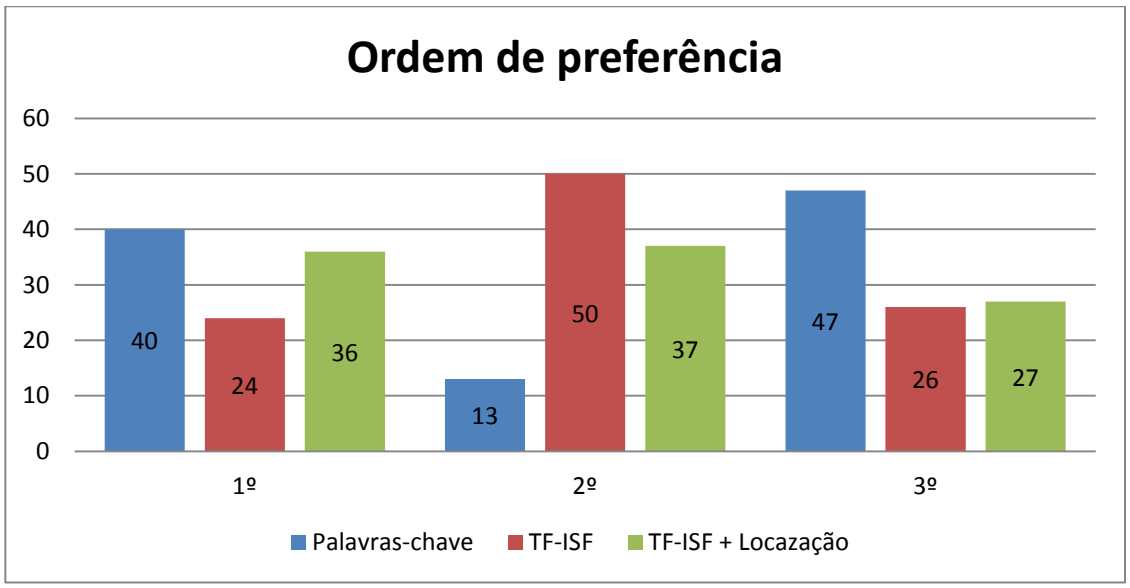



Gráfico 12 – Ordem de preferência dos utilizadores para cada método

Observando os valores do Gráfico 12, pode-se dizer que a primeira escolha dos utilizadores é o método de palavras-chave. No entanto, a diferença não é grande dado que o método TF-ISF + Localização apenas foi escolhido menos 4 vezes em primeiro que o método palavras-chave. Se compararmos os valores do ordenamento com os valores obtidos pelo *rating*, nota-se que a primeira escolha não está de acordo com o método com maior valor de *rating*. Assim fez-se mais uma vez o *inter-raters agreement*, para avaliar o grau de concordância entre os utilizadores. Os resultados para o método palavras-chave foi 0,007, para o TF-ISF 0,011 e para o método TF-ISF -0,023. Perante estes valores e fazendo a interpretação através da tabela 13, o método palavras-chave e TF-ISF tiveram uma concordância ligeira, enquanto o método TF-ISF + Localização teve uma concordância baixa.
Mais uma vez a métrica do *inter-raters agreement* veio demonstrar a subjetividade da escolha do utilizador, fazendo com que, através destes valores, não seja possível retirar grandes conclusões sobre qual o melhor método de sumarização a utilizar.

# Capítulo 7
# Plano de Trabalho

O desenvolvimento do projeto foi feito utilizando uma aproximação à metodologia do **Model em Cascata modificada**. A escolha desta metodologia, deveu-se ao facto de o desenvolvimento do projeto se centrar na especificação de requisitos, implementação e *testing* dos mesmos, em tudo semelhante à fase de "*Requirement*", "*Implementation*" e "*Verification*" da metodologia em cascata e, de ser a metodologia usada na empresa.

A metodologia **Modelo em Cascata modificado** permite uma maior flexibilização no planeamento das tarefas. Esta flexibilização trouxe a possibilidade de se sobrepor diferentes fases do ciclo de vida do *software*. Já no **Modelo em Cascata Tradicional,** só era possível avançar para outra fase no ciclo de vida quando esta estivesse completamente desenvolvida, nunca podendo voltar a regredir quando se avançasse para outra fase. Por exemplo, estamos na fase da implementação e surgiu a necessidade de alterar um requisito; no **modelo em Cascata tradicional** isto não é possível, enquanto no **modelo em Cascata modificado** é possível voltar à fase anterior e refazer se for necessário.

## 7.1. Planeamento

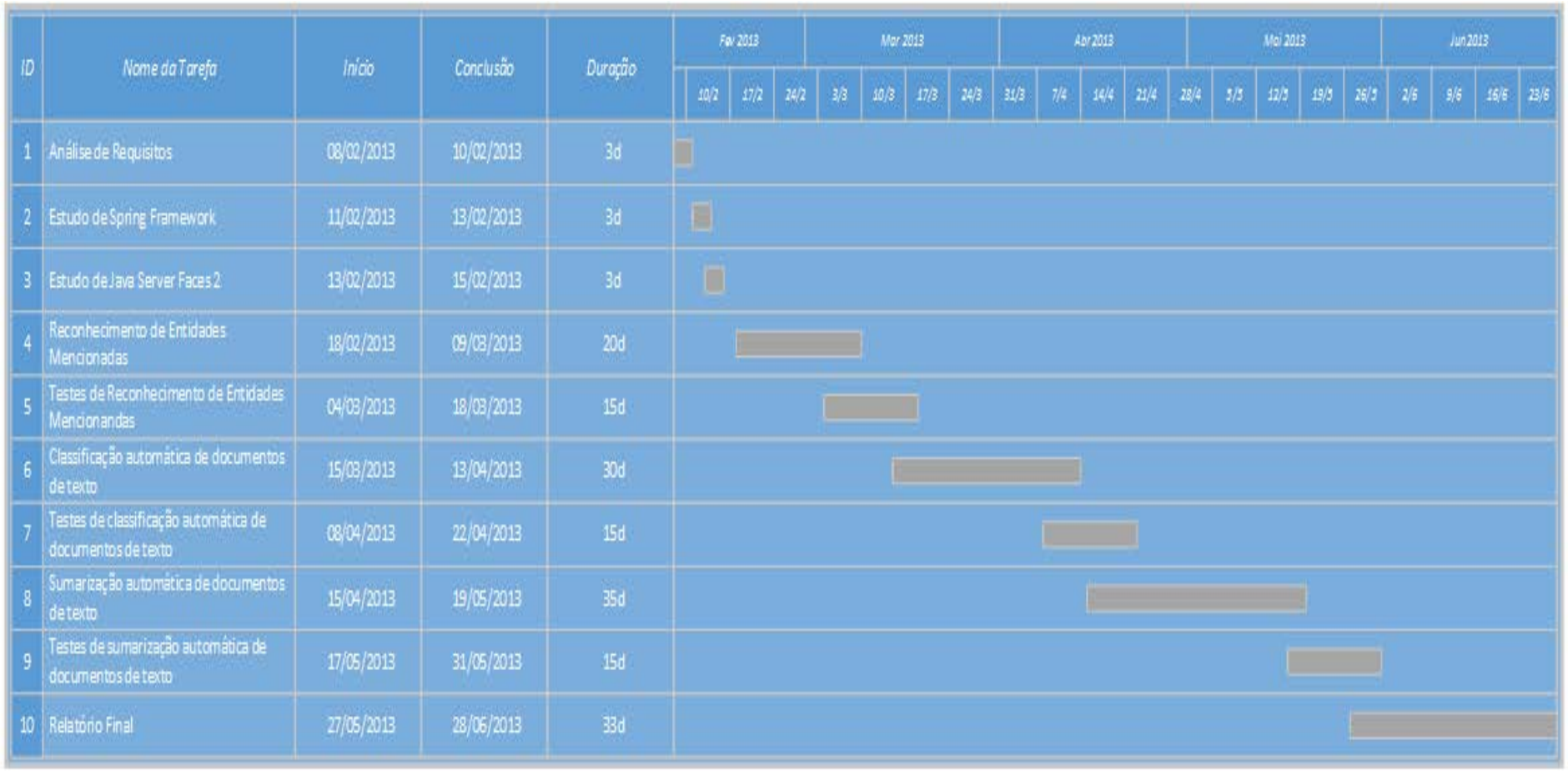
No início do segundo semestre de estágio definiu-se, como tarefas principais a análise de requisitos, o reconhecimento de entidades mencionadas, a classificação e sumarização automática de documentos de texto. O planeamento que foi feito para a execução destas tarefas pode ser visto no diagrama de GANTT da Figura 12.
Neste planeamento o tempo para cada tarefa foi definido de forma empírica, tendo como mote o estudo feito através do estado da arte. O tempo de execução para cada tarefa foi de acordo com a complexidade dessa tarefa e dos testes que serão necessários fazer para avaliá-las.

## 1.1. Resultado da execução do projeto

O projeto seguiu o planeamento inicial, no entanto, a execução das tarefas nem sempre corresponde fielmente ao que foi planeado, pelo que foi necessário replanear o projeto para melhorar todos os aspetos relativos ao mesmo.

A finalização da análise de requisitos encontrou novas tarefas necessárias na implementação levando a reduzir o tempo de algumas tarefas e a acrescentar outras novas ao planeamento. O resultado da execução do projeto pode ser visto no diagrama apresentado na Figura 13.

No diagrama da Figura 13, nota-se que na tarefa reconhecimento de entidades mencionadas foi reduzido o tempo de execução e retirados os testes a esta tarefa que foi reduzida porque foram encontradas outras tarefas, como a implementação de certas partes do módulo **educa-content,** e definidas como mais importantes para o seguimento do projeto, do que a tarefa do reconhecimento de entidades mencionadas. Outro motivo foi a existência de uma implementação de Reconhecimento de Entidades Mencionadas no projeto e as partes dentro do módulo **educa-content,** essenciais na validação da classificação e sumarização de documentos, estarem por desenvolver.



| ID | Nome da Tarefa | Início | Conclusão | Duração |
|---|---|---|---|---|
| 1 | Análise de Requisitos | 08/02/2013 | 10/02/2013 | 3d |
| 2 | Estudo de Spring Framework | 11/02/2013 | 13/02/2013 | 3d |
| 3 | Estudo de Java Server Faces 2 | 13/02/2013 | 15/02/2013 | 3d |
| 4 | Reconhecimento de Entidades Mencionadas | 18/02/2013 | 09/03/2013 | 20d |
| 5 | Testes de Reconhecimento de Entidades Mencionandas | 04/03/2013 | 18/03/2013 | 15d |
| 6 | Classificação automática de documentos de texto | 15/03/2013 | 13/04/2013 | 30d |
| 7 | Testes de classificação automática de documentos de texto | 08/04/2013 | 22/04/2013 | 15d |
| 8 | Sumarização automática de documentos de texto | 15/04/2013 | 19/05/2013 | 35d |
| 9 | Testes de sumarização automática de documentos de texto | 17/05/2013 | 31/05/2013 | 15d |
| 10 | Relatório Final | 27/05/2013 | 28/06/2013 | 33d |



**Figura 12 – Planeamento inicial do projeto EDUCA**

| ID | Nome da Tarefa | Início | Conclusão | Duração |
|---|---|---|---|---|
| 1 | Análise de Requisitos | 08/02/2013 | 10/02/2013 | 3d |
| 2 | Estudo de Spring Framework | 11/02/2013 | 13/02/2013 | 3d |
| 3 | Estudo de Java Server Faces 2 | 13/02/2013 | 15/02/2013 | 3d |
| 4 | Reconhecimento de Entidades Mencionadas | 18/02/2013 | 27/02/2013 | 10d |
| 5 | Classificação automática de documentos de texto | 28/02/2013 | 29/03/2013 | 30d |
| 6 | Testes de classificação automática de documentos de texto | 18/03/2013 | 01/04/2013 | 15d |
| 7 | Sumarização automática de documentos de texto | 29/04/2013 | 02/06/2013 | 35d |
| 8 | Testes sumarização automática de documentos de texto | 27/05/2013 | 10/06/2013 | 15d |
| 9 | Implementação módulo content | 01/04/2013 | 14/06/2013 | 75d |
| 10 | Testes de Sistema | 01/04/2013 | 14/06/2013 | 75d |
| 11 | Relatório Final | 01/05/2013 | 28/06/2013 | 59d |

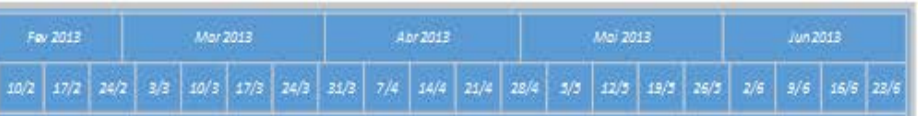



**Figura 13 – Execução final das tarefas do estágio**

# Capitulo 8
# Conclusão

Ao longo deste relatório foi pretendido abordar vários temas importantes na execução das tarefas indicadas nos objetivos. Assim, inicialmente foi necessário exercer um breve estudo sobre tecnologias/conceitos relevantes para o projeto EDUCA. Seguidamente, o foco do trabalho foi analisar o estado do projeto, tendo para isso sido realizados diversos testes por comparação com outras ferramentas existentes, de forma a conseguir-se compreender os pontos fortes e as limitações da implementação existente.

De acordo com a análise feita do estado do projeto, foram extraídos os requisitos necessários para cumprir as tarefas definidas. Seguiu-se para a implementação, onde foi feita a descrição das tarefas principais do estágio, classificação e sumarização automática de documentos de texto, bem como da implementação sobre o módulo **educa-content**, preciso para validar a implementação das tarefas referidas. Por último, a fase de experimentação da classificação e sumarização automática de documentos de texto, onde foi avaliado em termos de qualidade e precisão com o intuito de indicar o grau de sucesso da implementação de cada uma das tarefas.

Em termos de trabalho desenvolvido ao longo do estágio, existem algumas conclusões a reter. A primeira diz respeito à classificação de documentos de texto, pois através dos testes feitos pode-se constatar que apresenta uma boa performance e qualidade pois consegue alcançar os 70% de precisão global; a outra diz respeito à sumarização automática de documentos de texto, onde os resultados não foram tão animadores como os obtidos na classificação automática de documentos de texto, uma vez que pelos testes não foi possível retirar grandes ilações, além de que existe a possibilidade de ocorrerem problemas em termos de coerência/coesão dos textos. No entanto, a sumarização automática ainda é uma área em estudo, pelo que é possível que venham a surgir novas formas de fazer a sumarização automática de documentos de texto. Outra coisa a ressalvar prende-se com o reconhecimento de entidades mencionadas, o que se tivesse sido explorado no estágio, iria comprometer as outras duas tarefas propostas inicialmente no mesmo. Perante isto, pode-se dizer que o trabalho realizado ao longo do estágio teve sucesso.

As contribuições feitas pelo estagiário recaíram sobre a implementação da classificação e sumarização automática de documentos de texto e sobre o módulo **educa-content**. Relativo a este último, foi importante desenvolver a parte de gestão de conteúdos simples, dado ser aqui que a sumarização e classificação automática de documentos de texto são utilizadas. No entanto, e como houve tempo, ainda foram implementados outros requisitos, não tão importantes no decorrer do estágio, mas sim para o projeto EDUCA.

## 8.1. Propostas de melhoria do sistema

Durante o decurso do estágio foram encontradas algumas tarefas que podem ser melhoradas. Estes melhoramentos prendem-se essencialmente em retirar o *Fedora-commons* de repositório e encontrar uma outra ferramenta que sirva também como repositório. No reconhecimento de entidades mencionadas, é necessário encontrar uma ferramenta que consiga obter melhores resultados em termos de performance e qualidade. Relativamente à sumarização, procurar talvez outras metodologias, como por exemplo em vez de considerar a separação do texto apenas em frases, considerar a separação do texto em parágrafos e depois em frases.

Além disto ficaram ainda requisitos por implementar, que não sendo importantes no estágio, são para o projeto. Por último uma tarefa que foi implementada mas que, no caso de reformulação deveria seguir outra implementação; o requisito do conteúdo composto ou colaborativos. Este requisito poderia alargar o número de utilizadores dentro do mesmo painel, pelo que, agora, apenas é permitido um utilizador por painel, de forma a evitar problemas relativos à concorrência.

# Anexo A – *Queries* utilizadas na avaliação das triple stores

## Query 1

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

SELECT DISTINCT ?product ?label
WHERE {
   ?product rdfs:label ?label .
        ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
   ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
   ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
   ?product bsbm:productPropertyNumeric1 ?value1 .
        FILTER (?value1 > 10)
        }
ORDER BY ?label
LIMIT 10
```

## Query 2

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dc: http://purl.org/dc/elements/1.1/

SELECT ?label ?comment ?producer ?productFeature ?propertyTextual1
?propertyTextual2 ?propertyTextual3
 ?propertyNumeric1 ?propertyNumeric2 ?propertyTextual4 ?propertyTextual5
?propertyNumeric4
WHERE {
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> rdfs:label ?label .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>  rdfs:comment
?comment .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>  bsbm:producer
?p .
   ?p rdfs:label ?producer .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> dc:publisher ?p .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productFeature ?f .
   ?f rdfs:label ?productFeature .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyTextual1 ?propertyTextual1 .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyTextual2 ?propertyTextual2 .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyTextual3 ?propertyTextual3 .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyNumeric1 ?propertyNumeric1 .
   <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyNumeric2 ?propertyNumeric2 .
   OPTIONAL                { <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyTextual4 ?propertyTextual4 }
   OPTIONAL                { <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyTextual5 ?propertyTextual5 }
   OPTIONAL                { <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>
bsbm:productPropertyNumeric4 ?propertyNumeric4 }}
```

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?product ?label
WHERE {
   ?product rdfs:label ?label .
   ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
         ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
         ?product bsbm:productPropertyNumeric1 ?p1 .
         FILTER ( ?p1 > 50 )
         ?product bsbm:productPropertyNumeric3 ?p3 .
         FILTER (?p3 < 135 )
   OPTIONAL {
      ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
      ?product rdfs:label ?testVar }
   FILTER (!bound(?testVar))
}
ORDER BY ?label
LIMIT 10
```

## Query 4

```
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?product ?label ?propertyTextual
WHERE {
   {
     ?product rdfs:label ?label .
     ?product                    rdf:type                    <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
     ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
          ?product bsbm:productFeature bsbm-inst:ProductFeature2 .
     ?product bsbm:productPropertyTextual1 ?propertyTextual .
          ?product bsbm:productPropertyNumeric1 ?p1 .
          FILTER ( ?p1 > 50 )
   } UNION {
     ?product rdfs:label ?label .
     ?product                    rdf:type                    <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
     ?product bsbm:productFeature bsbm-inst:ProductFeature1 .
          ?product bsbm:productFeature bsbm-inst:ProductFeature3 .
     ?product bsbm:productPropertyTextual1 ?propertyTextual .
          ?product bsbm:productPropertyNumeric2 ?p2 .
          FILTER ( ?p2> 325 )
   }
}
ORDER BY ?label
OFFSET 5
LIMIT 10
```

## Query 5

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT DISTINCT ?product ?productLabel
WHERE {
        ?product rdfs:label ?productLabel .
   FILTER (<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer339/Product16505> != ?product)
        <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> bsbm:productFeature ?prodFeature .
        ?product bsbm:productFeature ?prodFeature .
        <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> bsbm:productPropertyNumeric1 ?origProperty1 .
        ?product bsbm:productPropertyNumeric1 ?simProperty1 .
        FILTER (?simProperty1 < (?origProperty1 + 120) && ?simProperty1 > (?origProperty1 - 120))
        <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> bsbm:productPropertyNumeric2 ?origProperty2 .
        ?product bsbm:productPropertyNumeric2 ?simProperty2 .
        FILTER (?simProperty2 < (?origProperty2 + 170) && ?simProperty2 > (?origProperty2 - 170))
}
ORDER BY ?productLabel
LIMIT 5
```

## Query 6

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>

SELECT ?product ?label
WHERE {
        ?product rdfs:label ?label .
   ?product rdf:type bsbm:Product .
        FILTER regex(str(?label), "untruer","i")
}
```

## Query 7

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>

SELECT ?productLabel ?offer ?price ?vendor ?vendorTitle ?review ?revTitle
      ?reviewer ?revName ?rating1 ?rating2
WHERE {
          <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88>          rdfs:label
?productLabel .
   OPTIONAL {
      ?offer                    bsbm:product                    <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> .
                  ?offer bsbm:price ?price .
                  ?offer bsbm:vendor ?vendor .
                  ?vendor rdfs:label ?vendorTitle .
      ?vendor bsbm:country <http://downlode.org/rdf/iso-3166/countries#DE> .
      ?offer dc:publisher ?vendor .
      ?offer bsbm:validTo ?date .
      FILTER (xsd:dateTime(?date) < xsd:dateTime("2008-06-29T00:00:00") )
   }
   OPTIONAL {
          ?review                bsbm:reviewFor                <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> .
          ?review rev:reviewer ?reviewer .
          ?reviewer foaf:name ?revName .
          ?review dc:title ?revTitle .
   OPTIONAL { ?review bsbm:rating1 ?rating1 . }
   OPTIONAL { ?review bsbm:rating2 ?rating2 . }
   }
}
```

## Query 8

```
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rev: <http://purl.org/stuff/rev#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?title ?text ?reviewDate ?reviewer ?reviewerName ?rating1 ?rating2 ?rating3
?rating4
WHERE {
        ?review bsbm:reviewFor <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product88> .
        ?review dc:title ?title .
        ?review rev:text ?text .
        FILTER langMatches( lang(?text), "EN" )
        ?review bsbm:reviewDate ?reviewDate .
        ?review rev:reviewer ?reviewer .
        ?reviewer foaf:name ?reviewerName .
        OPTIONAL { ?review bsbm:rating1 ?rating1 . }
        OPTIONAL { ?review bsbm:rating2 ?rating2 . }
        OPTIONAL { ?review bsbm:rating3 ?rating3 . }
        OPTIONAL { ?review bsbm:rating4 ?rating4 . }
}
ORDER BY DESC(?reviewDate)
LIMIT 20
```

## Query 9

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

Select ?product
{
  { Select ?product
    {
      { Select ?product (count(?offer) As ?offerCount)
        {
          ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
          ?offer bsbm:product ?product .
        }
        Group By ?product
      }
    }
    Order By desc(?offerCount)
    Limit 1000
  }
  FILTER NOT EXISTS
  {
    ?offer bsbm:product ?product .
    ?offer bsbm:vendor ?vendor .
    ?vendor bsbm:country ?country .
    FILTER(?country=<http://downlode.org/rdf/iso-3166/countries#US>)
  }
}
```

## Query 10

```
prefix bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
 prefix bsbm-inst: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/>
 prefix rev: <http://purl.org/stuff/rev#>
 prefix xsd: <http://www.w3.org/2001/XMLSchema#>

 Select ?country ?product ?nrOfReviews ?avgPrice
 {
  { Select ?country (max(?nrOfReviews) As ?maxReviews)
   {
    { Select ?country ?product (count(?review) As ?nrOfReviews)
     {
      ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
      ?review bsbm:reviewFor ?product ;
              rev:reviewer ?reviewer .
      ?reviewer bsbm:country ?country .
     }
     Group By ?country ?product
    }
   }
   Group By ?country
  }
  { Select ?country ?product (avg(xsd:float(xsd:string(?price))) As ?avgPrice)
   {
    ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
    ?offer bsbm:product ?product .
    ?offer bsbm:price ?price .
   }
   Group By ?country ?product
  }
  { Select ?country ?product (count(?review) As ?nrOfReviews)
   {
    ?product a <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/instances/ProductType1> .
    ?review bsbm:reviewFor ?product .
    ?review rev:reviewer ?reviewer .
    ?reviewer bsbm:country ?country .
   }
   Group By ?country ?product
  }
  FILTER(?nrOfReviews=?maxReviews)
 }
 Order By desc(?nrOfReviews) ?country ?product
```

# Anexo B – Manual de Utilizador da Implementação do Estagiário

## Gestão de *layouts*

A gestão de *layouts* permite ao utilizador criar, remover e ver novos layouts. O acesso aos *layouts* por parte de um utilizador faz-se através do menu, clicando em *templates* como mostra a Figura 1.


Figura 14 – Aceder à gestão de layouts do utilizador

Ao clicar no menu "templates" o utilizador será redirecionado para uma nova página com a listagem de *layouts* criados por ele. Esta página pode ser vista na Figura 2.

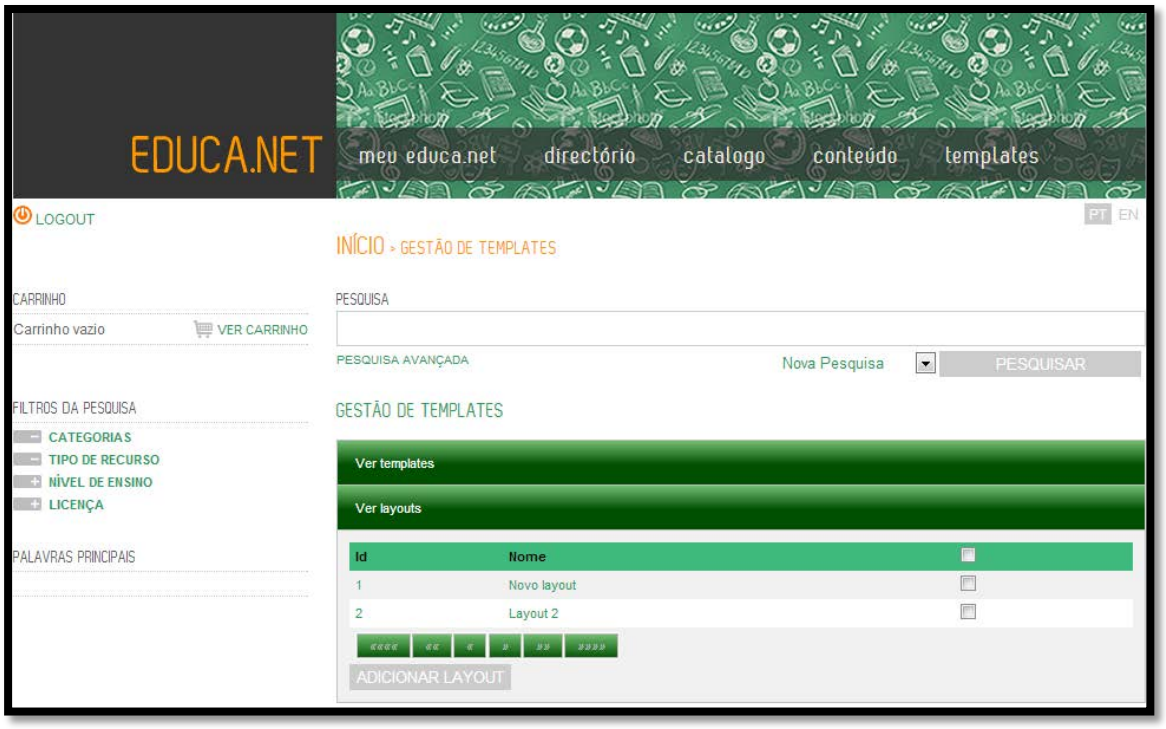



Figura 15 – Listagem de layouts do utilizador

Nesta página o utilizador pode realizar três tarefas:

- Criar novo layout;
- Ver layout;
- Remover layouts.

### Criar *layout*

Para o utilizador criar um novo *layout* deve clicar no botão adicionar layout. Após clicar no botão é enviado para a página que se encontra na Figura 3.


Figura 16 – Página de criação de layouts de utilizador

Nesta página o utilizador para criar um novo *layout* tem que ir adicionando painéis (containers). O botão adicionar adiciona sempre painéis do mesmo tamanho, sendo depois o utilizador a redimensionar o tamanho destes. A Figura 4 mostra um exemplo de um novo *layout*.

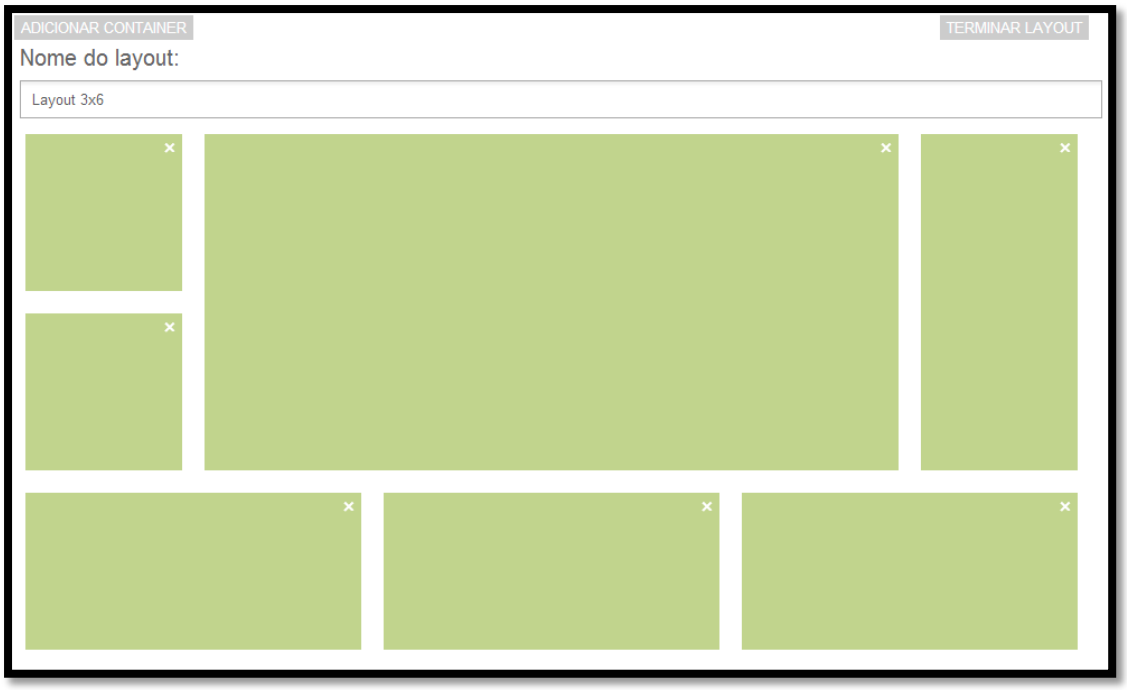



Figura 17 – Exemplo de um novo *layout*

Para além de ser possível adicionar painéis (containers) o utilizador também pode removê-los, clicando na cruz no canto superior direito.

O utilizador finaliza o *layout* quando clicar no botão terminar layout, voltando para a página com a listagem de *layouts*.

### Ver layout

O utilizador para ver um *layout* criado deve clicar no nome ou no id apresentados na listagem de *layouts*. Após clicar no *layout* que pretende ver o utilizador é redirecionado para a página onde é mostrado o *layout*. A Figura 5 mostra um exemplo do layout que o utilizador escolheu.


Figura 18 – Visualização do *layout*

### Remover layouts

A remoção de *layouts* é feita selecionando um ou mais *checkbox*. A Figura 6 mostra como o utilizador deve fazer.

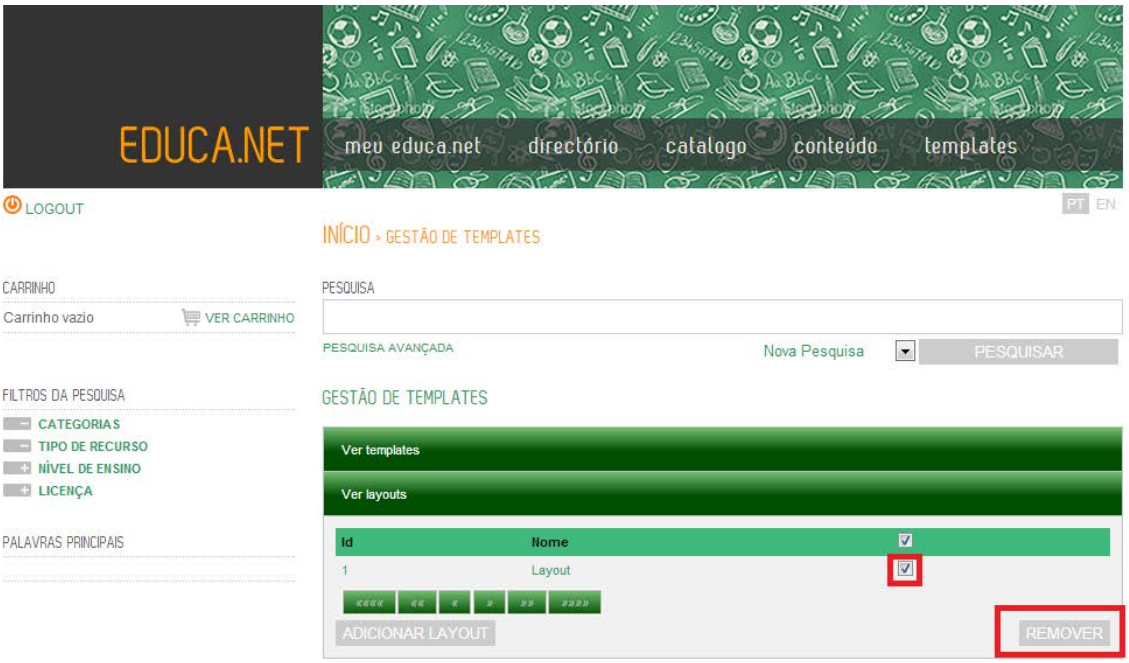



Figura 19 – Remover layouts

## Gestão de layouts de sistema

A gestão de *layouts* de sistema é em tudo semelhante à gestão de *layouts* de utilizadores. A única diferença é a forma como é acedida, sendo neste caso o acesso feito pelo painel mostrado na Figura 7.

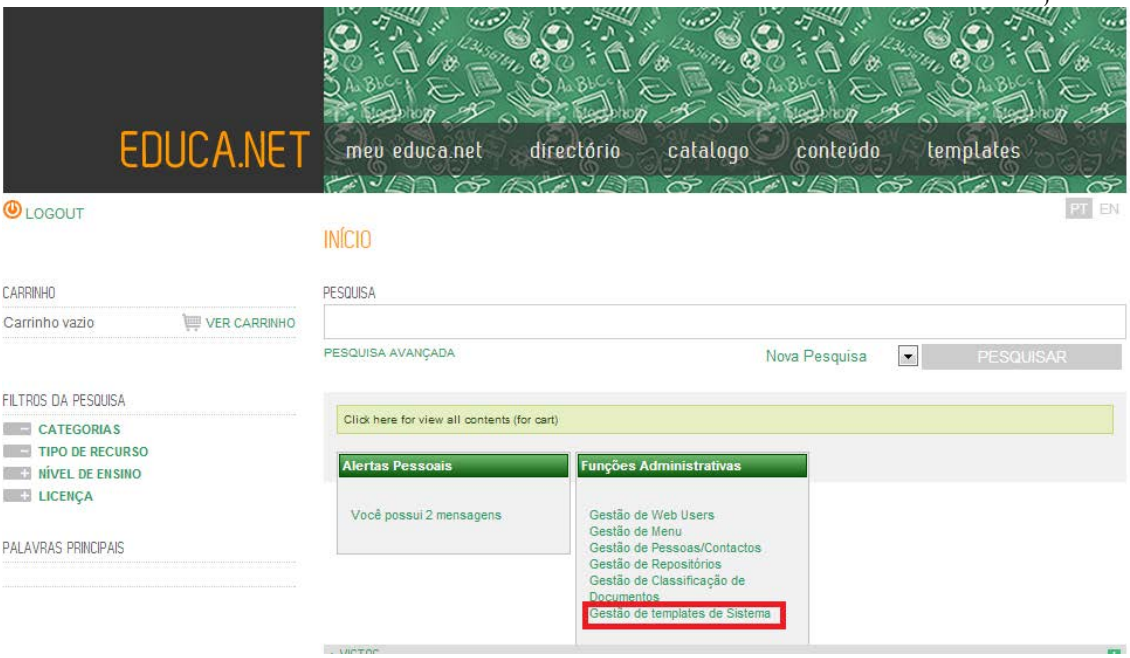



**Figura 20 – Painel de acesso de gestão de *templates* de sistema**

## Gestão de *templates*

A gestão de *templates* permite ao utilizador criar e ver novos templates. O acesso aos *layouts* por parte de um utilizador é feito através do menu, clicando em *templates* como mostra a Figura 1. Ao clicar no menu templates o utilizador será redirecionado para uma nova página com a listagem de *templates* criados por ele. Esta página pode ser vista na Figura 8.

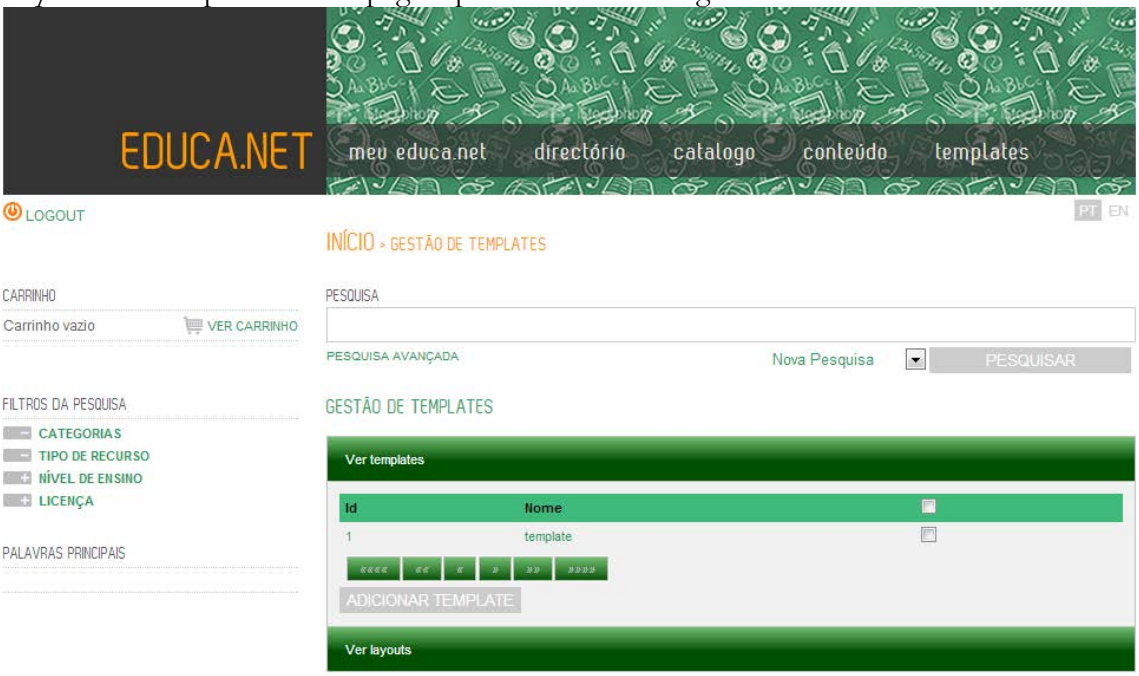



**Figura 21 – Listagem de *templates* do utilizador**

Nesta página o utilizador pode realizar duas tarefas:

- Criar *template*;
- Ver *template*;

### Criar template

Para o utilizador criar um novo *template* deve clicar no botão adicionar *template*. Após clicar no botão é enviado para a uma página onde o utilizador irá escolher o *layout* que pretende utilizar. Após escolher o *layout,* poderá alterar as cores de fundo deste, ou seja, de cada painel e do layout, e o alinhamento que pretende que o conteúdo adote. Um exemplo da criação de um template pode ser visto na Figura 9.

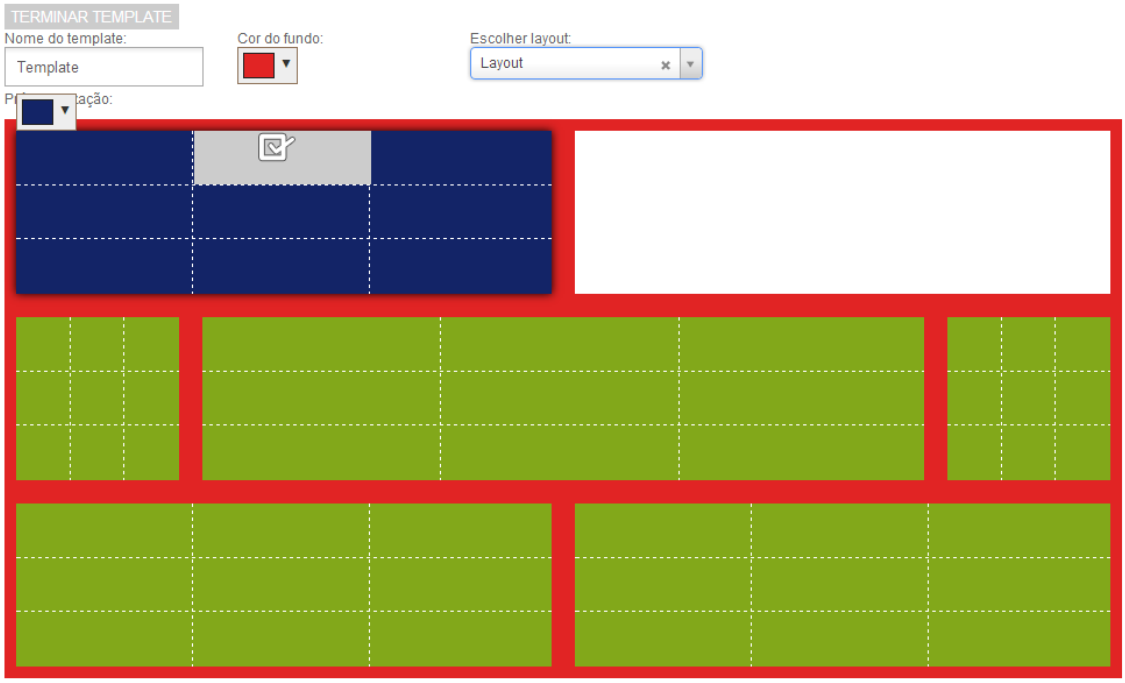



Figura 22 – Criação de *template* de utilizador

O utilizador finaliza o *layout* quando clicar no botão terminar *template*, voltando para a página com a listagem de *templates*.

### Ver *template*

O utilizador para ver um *template* criado deve clicar no nome ou no id apresentados na listagem de *templates*. Após clicar no *template* que pretende ver o utilizador é redirecionado para a página onde é mostrado o *template*. A Figura 10 mostra um exemplo do *template* que o utilizador escolheu.

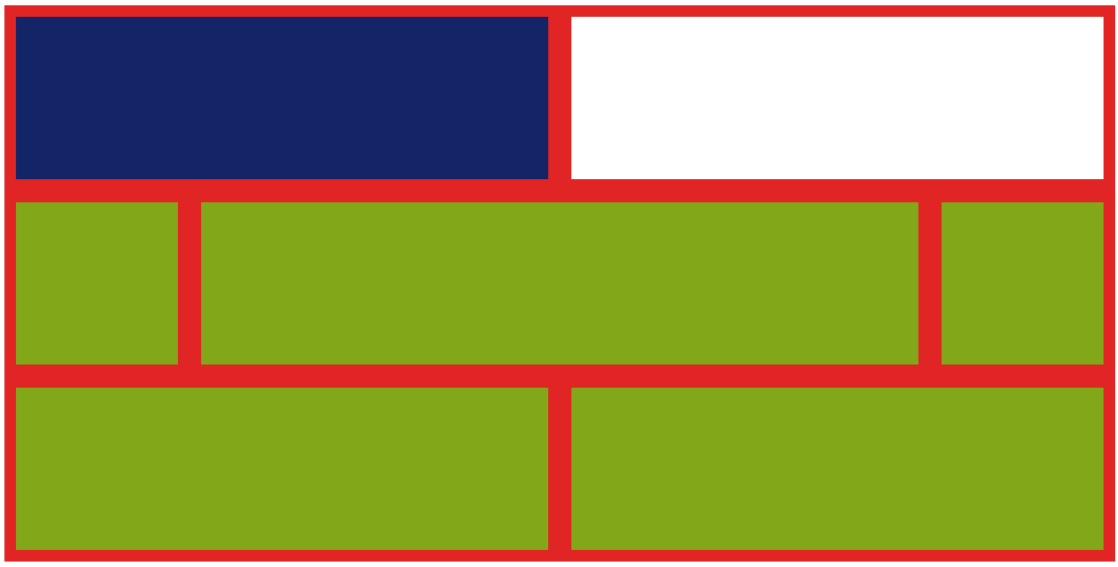


Figura 23 – Visualização de um template

## Gestão de *templates* de sistema

A gestão de *templates* de sistema é em tudo semelhante à gestão de *templates* de utilizadores. A única diferença é a forma como é acedida, sendo neste caso o acesso feito pelo painel mostrado na Figura 7.

## Gestão de conteúdos digitais

Para o utilizador aceder à gestão de conteúdos digitais deve clicar no menu conteúdo. A Figura 11 mostra o menu de acesso.


Figura 24 – Menu de acesso à gestão de conteúdos digitais

Após clicar no menu o utilizador irá ser redirecionado para a página apresentada na Figura 12.

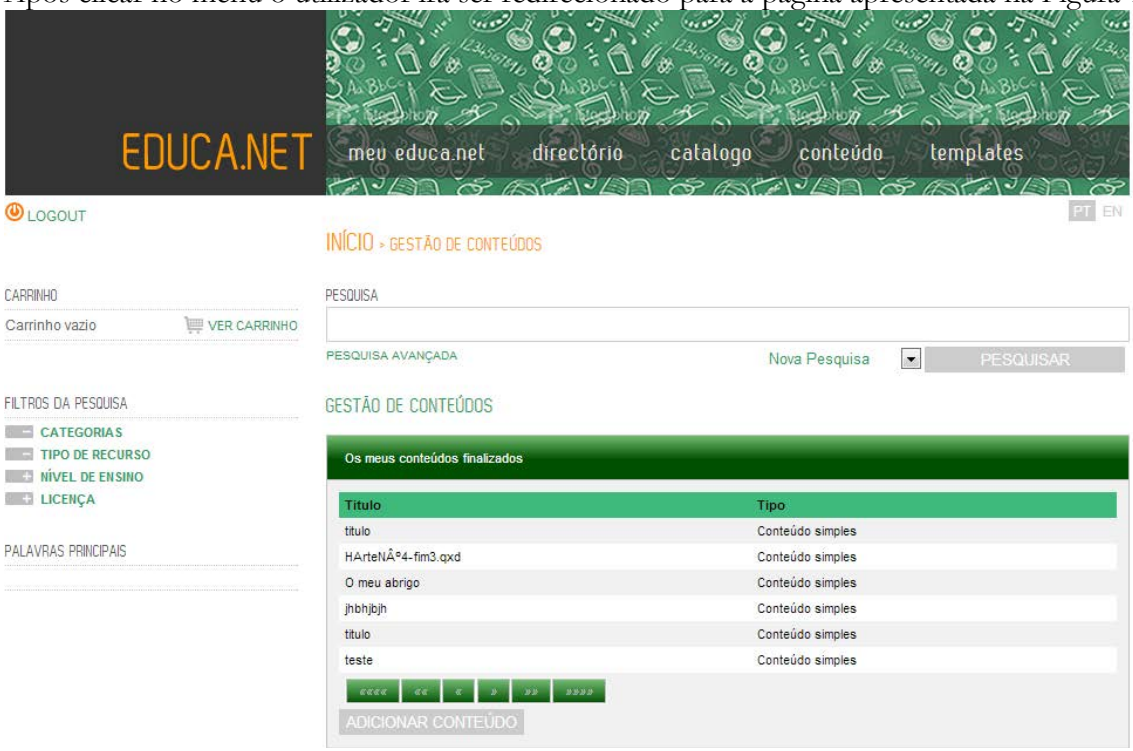



Figura 25 – Listagem de conteúdos digitais do utilizador

Nesta página o utilizador tem várias tarefas que pode fazer:

- Adicionar um novo conteúdo digital
- Ver a listagem de conteúdos compostos por finalizar (proprietário)
- Ver a listagem de conteúdos compostos para preencher (colaboradores)

## Adicionar conteúdo digital

O utilizador para adicionar um novo conteúdo digital deve clicar no botão "Adicionar Conteúdo". Após clicar nesse botão irá surgir um popup como mostra a Figura 13.

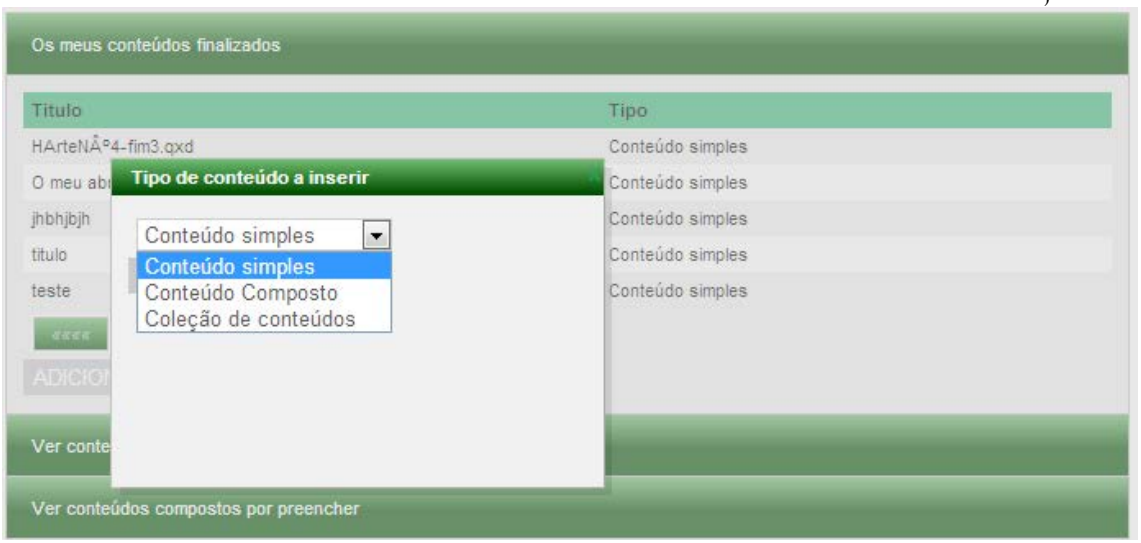



Figura 26 – Popup adicionar conteúdo digital

Neste popup o utilizador deve escolher uma das três opções:

- Conteúdo simples;
- Conteúdo Composto ou colaborativo;
- Coleção de conteúdos.

### Conteúdo simples

Caso o utilizador escolha conteúdo simples na *popup* da Figura 13, este será reencaminhado para outra página. Nesta página encontrará um formulário com um campo para fazer o *upload* de um ficheiro. Pode-se observar na Figura 14.

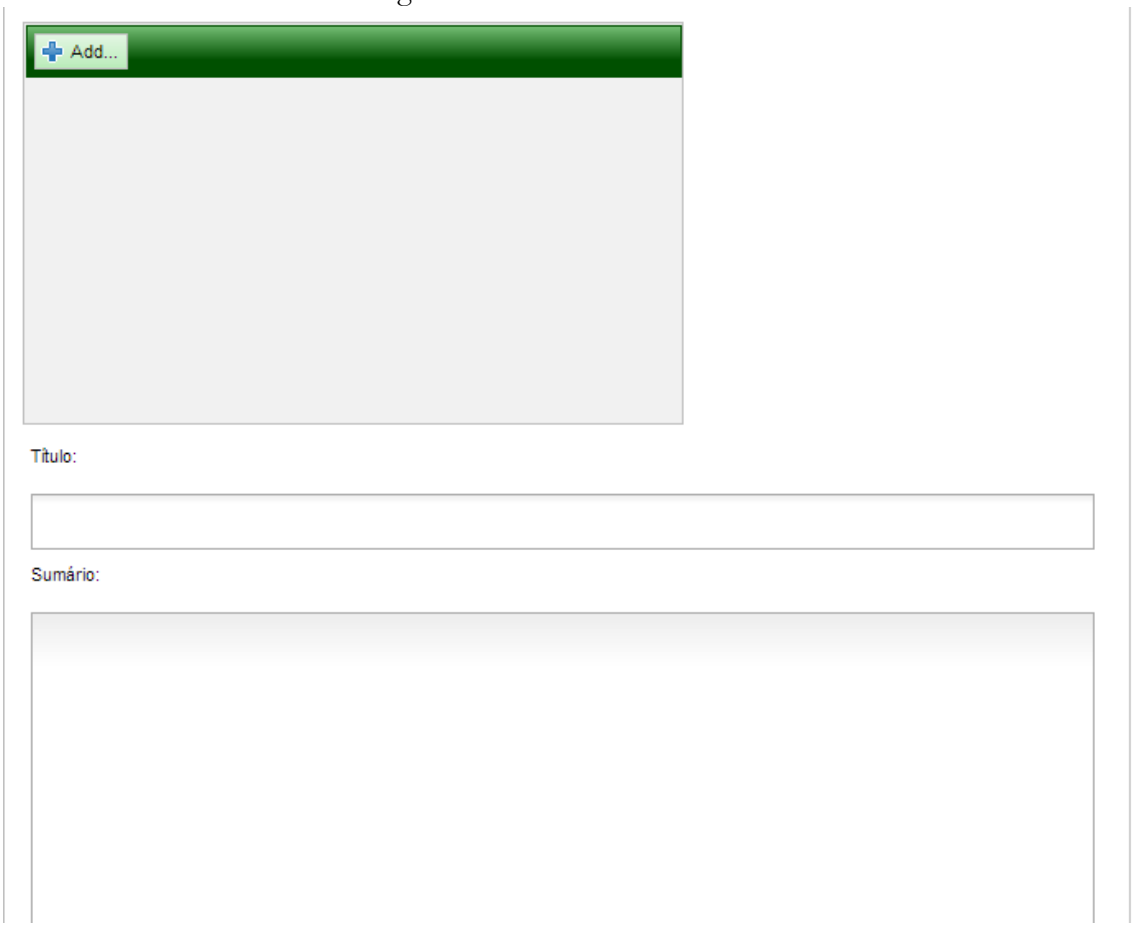



Figura 27 – Parte do formulário do conteúdo simples

Após o utilizador preencher este formulário e clicar no botão "Guardar" será criado um novo conteúdo simples e o utilizador será redirecionado para a listagem de conteúdos apresentada na Figura 12.

### Conteúdo composto ou colaborativo

Caso o utilizador escolha a opção "Conteúdo composto" no *popup* da Figura 13 será redirecionado para uma outra página para criar um novo conteúdo composto. Na Figura 15 pode ser visto como é criado um novo conteúdo composto.

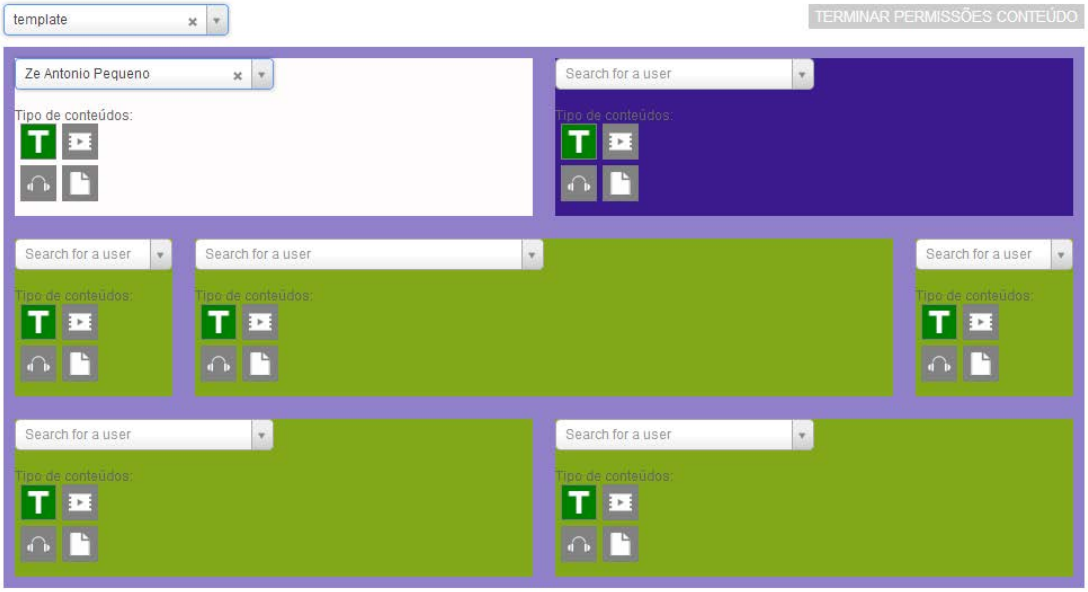



Figura 28 – Criação de conteúdo composto ou colaborativo

Nesta página o utilizador (proprietário) pode escolher que utilizadores podem colaborar no preenchimento do conteúdo composto ou colaborativo. Para além de escolher quais os utilizadores irão colaborar, também são colocados os tipos de conteúdos a colocar no painel. Para terminar a criação do conteúdo composto ou colaborativo o utilizador clica no botão "Terminar permissões de conteúdo", sendo redirecionado para a página da Figura 12.

Quando o utilizador (proprietário) cria este conteúdo, este irá aparecer nos utilizadores colaboradores na lista conteúdos compostos ou colaborativos por preencher, Figura 16.

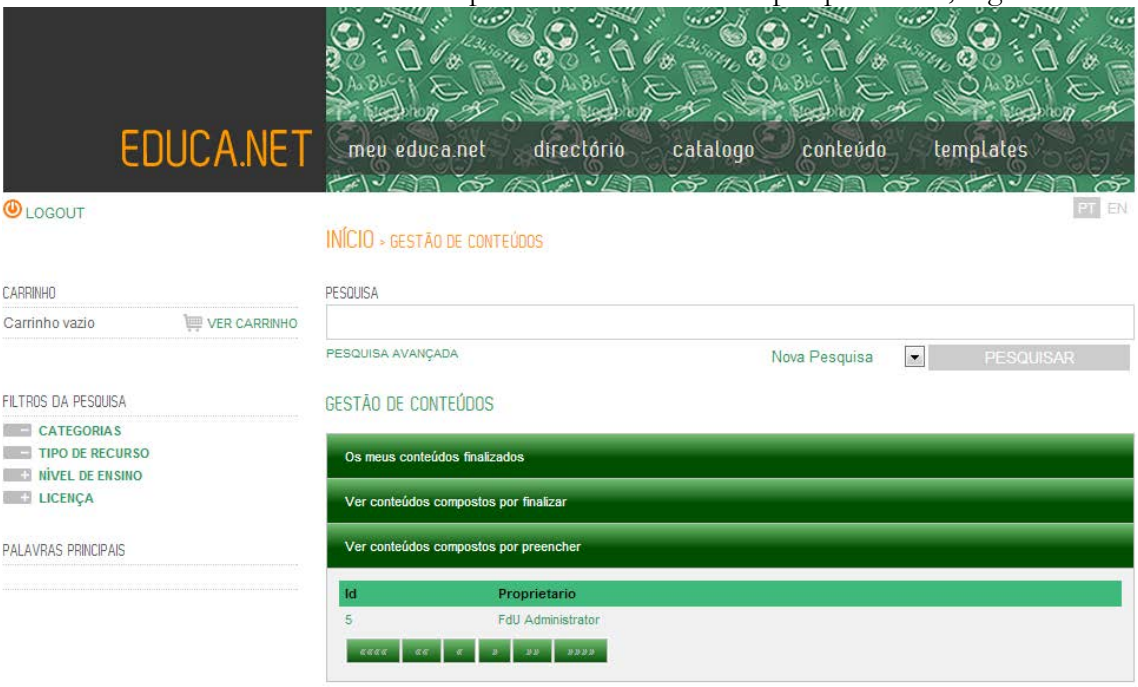



Figura 29 – Listagem de conteúdos por preencher

Para o utilizador (colaborador) proceder ao preenchimento do conteúdo composto ou colaborativo deve clicar no *link* com o nome do proprietário ou o id do conteúdo. Após clicar o colaborador irá para uma nova página com os painéis que pode preencher, Figura 17.

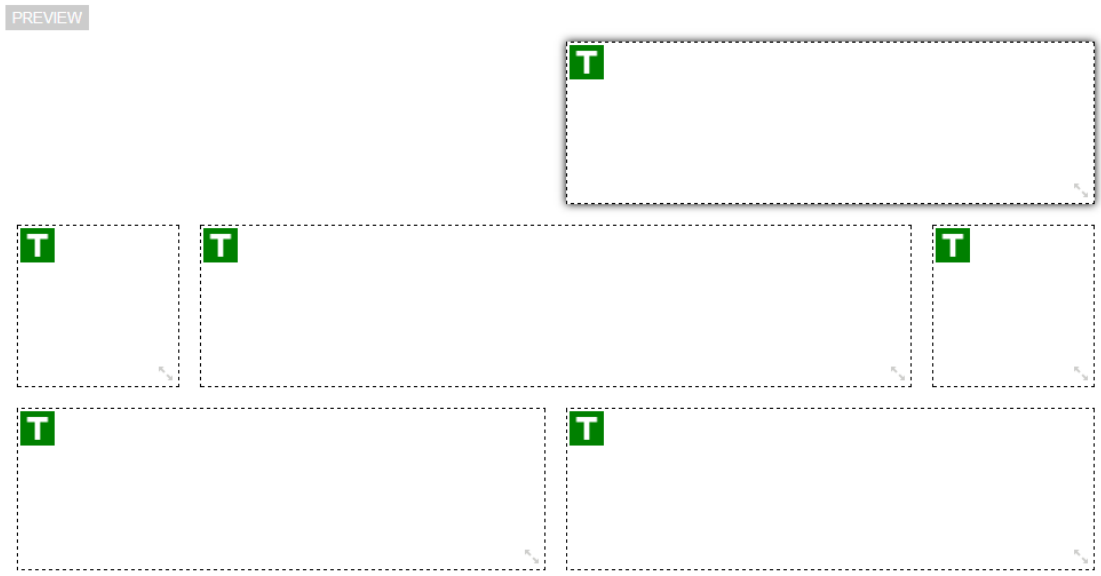



Figura 30 – Conteúdos que o colaborador pode preencher

Entrando nesta página o colaborador executa duas tarefas:

- Preencher o painel;
- Pré-visualizar conteúdos preenchidos

### Preencher o painel

Para o utilizador preencher o conteúdo deve clicar no *icon* que se encontra no canto inferior direito. Depois de clicar o painel vai ser redimensionado e irá aparecer um campo para o utilizador inserir texto, Figura 18.

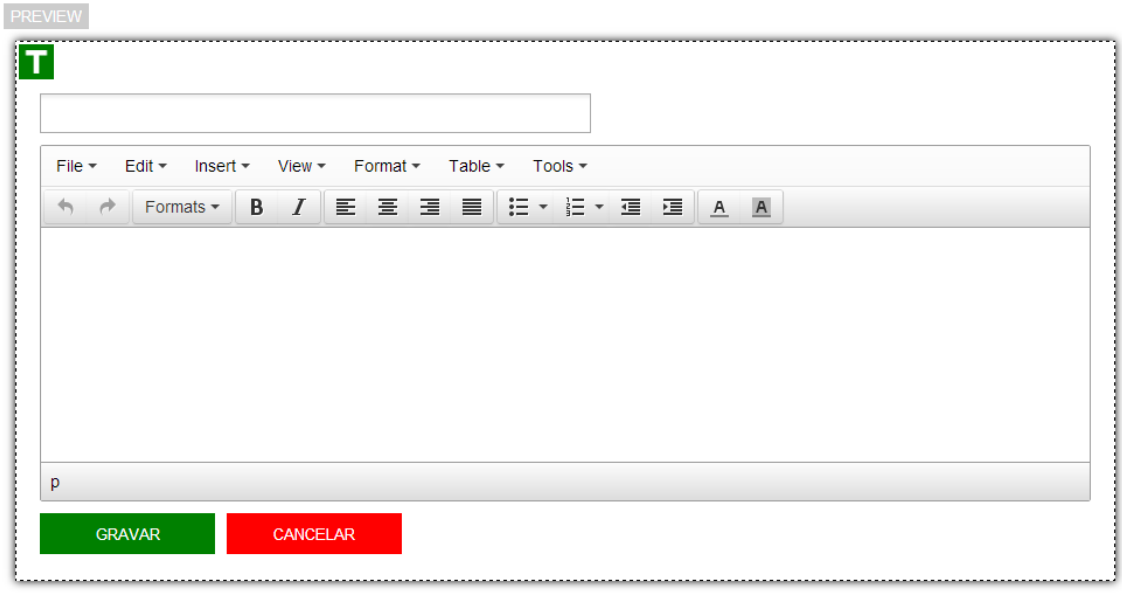



Figura 31 – Painel redimensionado para o utilizador preencher

Após o utilizador gravar ou cancelar o painel volta ao tamanho normal. Se o utilizador gravar o painel irá ficar com o mesmo semelhante ao que se encontra na Figura 19.

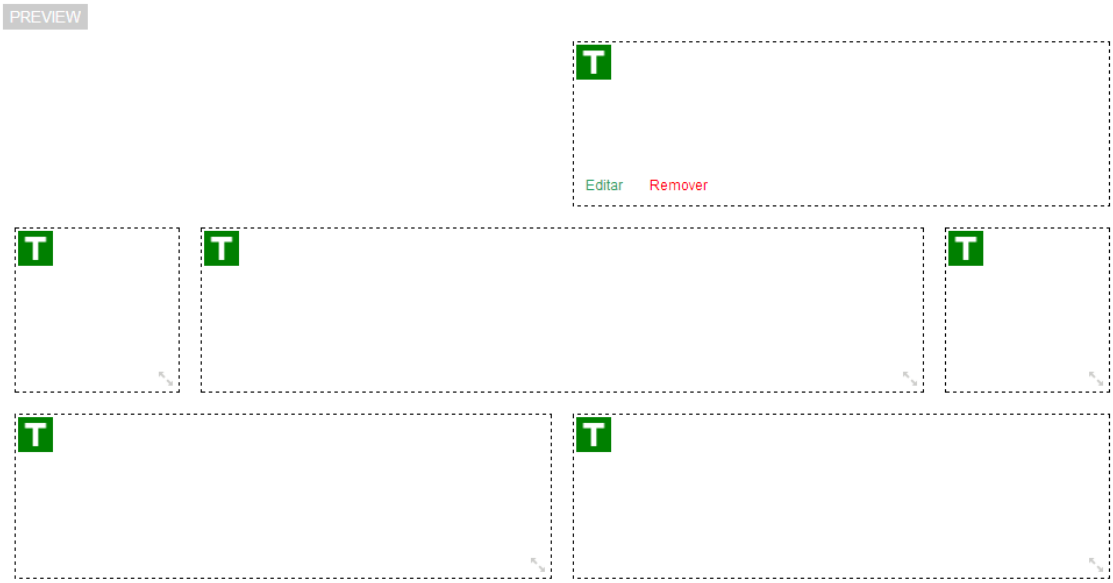



Figura 32 – Conteúdo gravado

O colaborador ainda pode editar ou remover o conteúdo que inseriu. Como mostra a Figura 19.

### Pré-visualizar conteúdo

Para o utilizador visualizar o conteúdo que inseriu o utilizador deve clicar no botão "preview", sendo redirecionado para uma página com a pré-visualização. Um exemplo dessa pré-visualização pode ser visto na Figura 20.


Figura 33 – Pré-visualização do conteúdo inserido

O utilizador ainda pode redimensionar o painel clicando no icon do canto inferior direito, ficando como o que está na Figura 21.

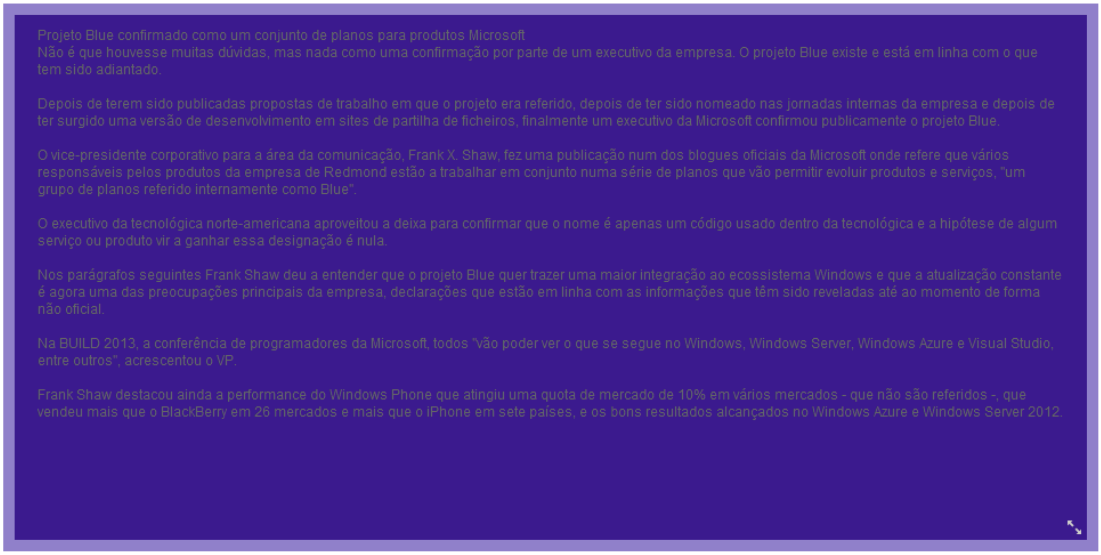



Figura 34 – Painel expandido para ser visualizado

## Validação e finalização do conteúdo composto ou colaborativo

Para o proprietário finalizar o conteúdo composto ou colaborativo deve ir à listagem de conteúdos compostos ou colaborativos por finalizar, Figura 22.

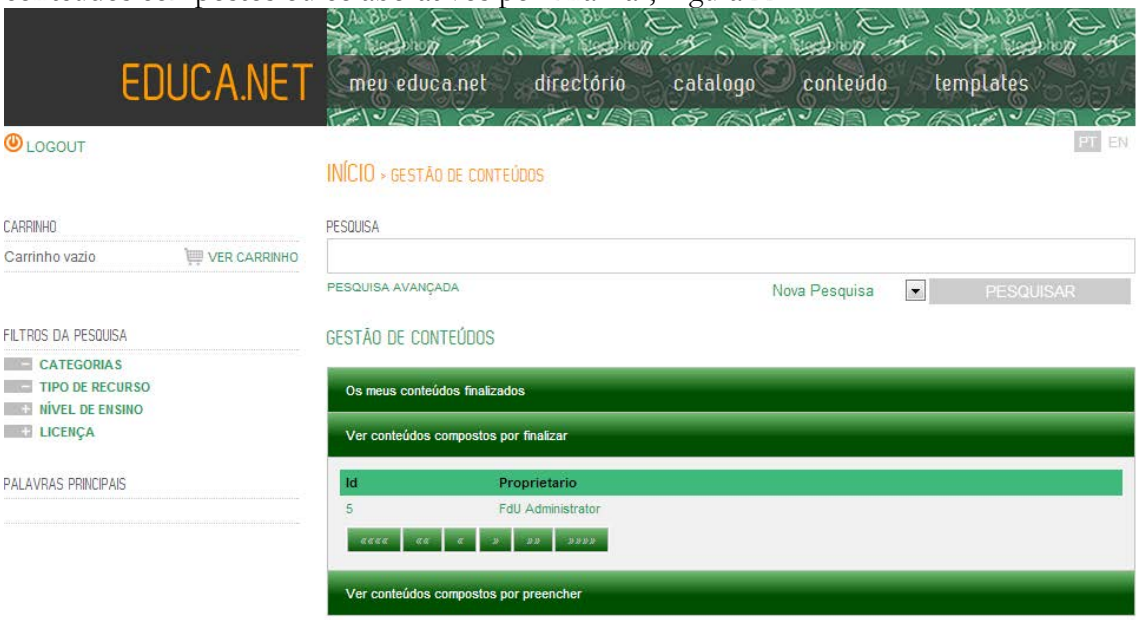



Figura 35 – Lista de conteúdos compostos por finalizar

Para o proprietário validar o conteúdo composto ou colaborativo, deve clicar no *link* com o nome do proprietário ou id do conteúdo composto ou colaborativo. Após clicar no *link* irá ser redirecionado para uma página onde o utilizador poderá aprovar ou rejeitar o conteúdo existente em cada painel. Caso o utilizador ache que todos os conteúdos inseridos estão corretos pode clicar logo no botão "Finalizar" e aprovar todos os painéis que se encontram preenchidos. Após clicar neste botão é redirecionado para outra página com um formulário para preencher, Figura 23.

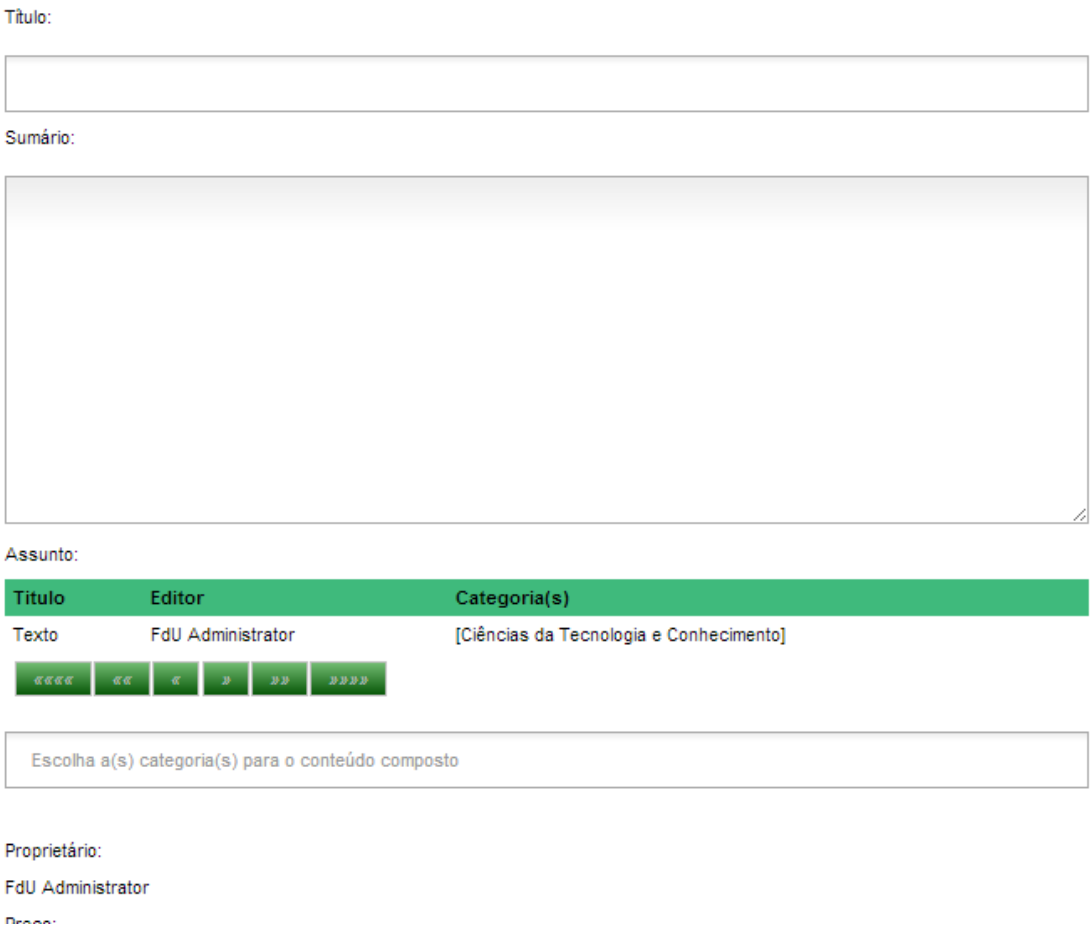



Figura 36 – formulário para finalizar o conteúdo composto ou colaborativo

Para finalizar o conteúdo composto ou colaborativo o utilizador deve preencher o formulário e depois clicar no botão "Gravar". Após concluir o utilizador é redirecionado para a página da Figura 12.

### Coleção de conteúdos de utilizador

Caso o utilizador, no popup presente na Figura 13, escolha a opção "coleção de conteúdos", este irá ser redirecionado para uma nova página. Nesta nova página o utilizador irá encontrar uma listagem com os conteúdos simples que o utilizador pode escolher para criar uma coleção. Um exemplo desta listagem pode ser visto na Figura 24.


Figura 37 – Lista de conteúdos simples que o utilizador pode escolher

Nesta listagem o utilizador terá que escolher os conteúdos que deseja para criar uma nova coleção e clicar no botão "Criar coleção" para seguir neste processo. Após o utilizador clicar será redirecionado para uma nova página para indicar quais os utilizadores ou grupos que têm permissões de acesso à coleção.

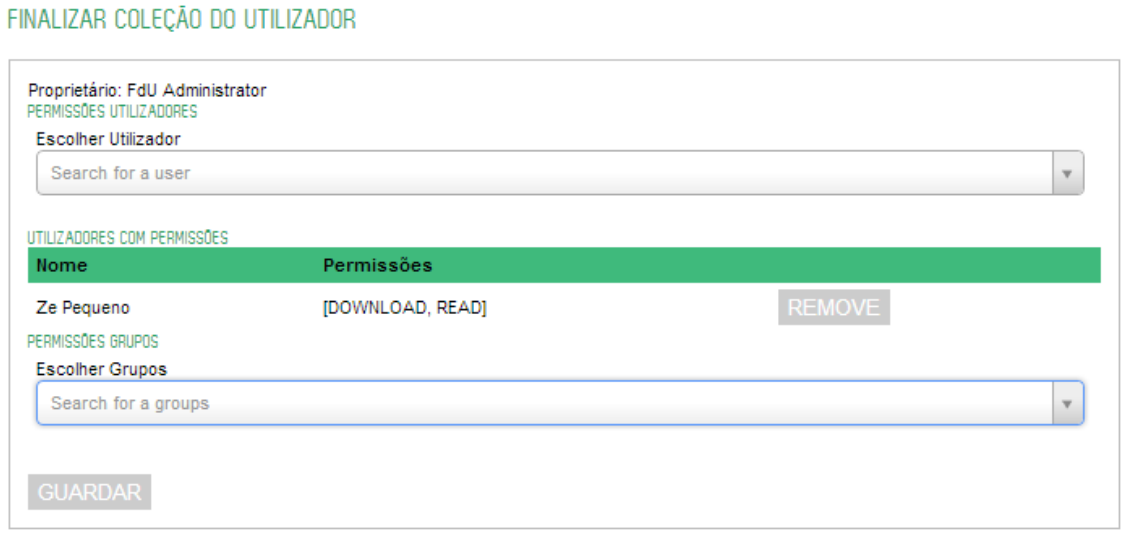



Figura 38 – Dar permissões a utilizadores e grupos

A coleção é dada como finalizada quando o utilizador clica no botão "Guardar", sendo redirecionado para a página da Figura 12.

## Classificação automática de documentos de texto

A classificação automática de documento de texto ocorre quando o utilizador faz o *upload* de um ficheiro de texto. Após o utilizador fazer o *upload* irá ser preenchido o campo assunto dentro do conteúdo simples como mostra a Figura 13


Figura 39 – Categorias sugeridas pela classificação automática

A classificação automática apenas é possível se o administrador dentro do *backoffice* treinar o classificador. Para o administrador aceder ao treino do classificador deve clicar no link identificado na Figura 27.

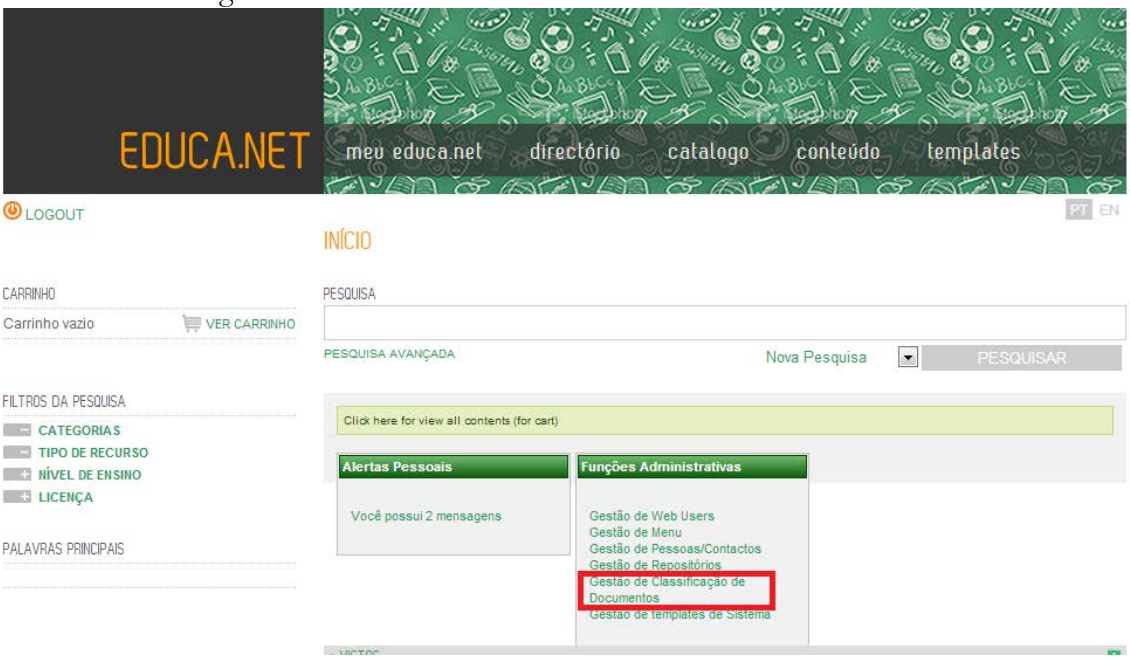



**Figura 40 – Treinar classificador**

Após clicar no *link* Gestão de Classificação de Documentos, o administrador é redirecionado para a página mostrada na Figura 28.


**Figura 41 – Página de treino do classificador**

Nesta página o administrador pode alterar o limite entre categorias, ou seja, adicionar a possibilidade de o classificador detetar mais que uma categoria.

Para iniciar o treino o administrador apenas necessita clicar no botão iniciar treino.

## Sumarização automática de documentos de texto

A sumarização automática de documentos de texto para o utilizador ocorre quando este faz o upload de um documento de texto. Após o upload é preenchido o campo sumário como mostra a Figura 29.

Sumário:

O texto terá duas partes: a censura propriamente dita - com o PS a apontar ao Governo a violação das promessas eleitorais, o falhanço das políticas, a perda de autoridade e credibilidade e a ausência de voz na Europa.
E outra de apresentação do caminho alternativo proposto pelos socialistas - com uma primeira alínea a reafirmar que o PS continua determinado no cumprimento dos seus compromissos internacionais e uma segunda a sublinhar que está a trabalhar numa estratégia credível de consolidação por via do crescimento económico.
Conscientes de que a apresentação da moção não produz efeitos constitucionais - foi, de resto, uma das razões que o levaram a anunciá-la agora -, os socialistas assumem que o seu objetivo principal é "clarificar a rutura" com um Governo que "chegou ao fim".

Figura 42 – Exemplo de um sumário gerado pelo sistema

# Anexo C – Manual de testes de aceitação

## Módulo educa-classification

| | |
|---|---|
| Módulo: | **educa-classification** |
| Versão educa: | |
| Data: | |
| Realizado por: | |
| Browser: | |
| Servidor: | |

Figura 43 – Informação relativa aos testes no módulo educa-classification

| Funcionalidade | Resultado | Descrição |
|---|---|---|
| No backoffice encontra o link para o treino do classificador | | |
| Quando o treino foi executado não consegue executar novamente | | |
| Escolher o valor do limite entre categorias | | |
| Depois do utilizador fazer o upload de um documento de texto são devolvidas as categorias a que pertencem | | |
| Ao introduzir um documento que não seja de texto não são apresentadas categorias | | |

Figura 44 – Testes de classificação automática de documentos de texto

| | |
|---|---|
| Módulo: | **educa-sumarization** |
| Versão educa: | |
| Data: | |
| Realizado por: | |
| Browser: | |
| Servidor: | |

Figura 45 - Informação relativa aos testes no módulo educa-summarization

| Funcionalidade | Resultado (*) | Descrição |
|---|---|---|
| Ao fazer o upload de um documento de texto é devolvido um sumário | | |
| Ao fazer o upload de um documento sem ser de texto não é deolvido nenhum sumário | | |

Figura 46 – Testes de sumarização automática de documentos de texto

| | |
|---|---|
| Módulo: | **educa-content** |
| Versão educa: | |
| Data: | |
| Realizado por: | |
| Browser: | |
| Servidor: | |

Figura 47 - Informação relativa aos testes no módulo educa-content

| Funcionalidade | Resultado (*) | Descrição |
|---|---|---|
| Ao escolher no menu "templates" vai para a listagem de templates e layouts | | |
| Ao escolher no menu do *backoffice* "Gestão de de templates de sistem" vai para a listagem de layouts de sistema | | |
| Na criação de um layout consegue adicionar paineis (containers) | | |
| Na criação de um layout consegue redimensionar os paineis (containers) | | |
| Na criação de um layout consegue remover os paineis (containers) | | |
| Ao terminar a criação do layout é redirecionado para a listagem de layouts/templates | | |
| Ao selecionar um ou mais checkbox aparece o botão "remover" | | |
| Clicando no botão "remover" os layouts são removidos | | |
| Clicando no link do layout é redirecionado para a visualização do layout | | |

Figura 48 – Módulo educa-content gestão de *layouts*

| Funcionalidade | Resultado (*) | Descrição |
|---|---|---|
| Ao escolher no menu "templates" vai para a listagem de templates e layouts | | |
| Ao escolher no menu do *backoffice* "Gestão de de templates de sistem" vai para a listagem de layouts de sistema | | |
| Na criação de um template consegue escolher um layout existente | | |
| Na criação de um template consegue mudar a cor de fundo do layout | | |
| Na criação de um template consegue mudar a cor de fundo dos paineis do layout | | |
| Na criação de um template consegue mudar definir o alinhamento de cada painel | | |
| Ao terminar a criação do template é redirecionado para a listagem de layouts/templates | | |
| Clicando no link do template é redirecionado para a visualização do template | | |

Figura 49 – Módulo educa-content gestão de *templates*

| Funcionalidade | Resultado (*) | Descrição |
|---|---|---|
| Ao escolher no menu "conteúdos" vai para a listagem de conteúdos | | |
| Ao clicar no botão "adicionar conteúdo" mostra um popup para escolher conteúdo | | |
| Se escolher conteúdo simples é redirecionado para a página de criação de conteúdo simples | | |
| Na página de criação de conteúdo simples ao clicar no botão "guardar" fica criado o conteúdo simples | | |

Figura 50 – Módulo educa-content gestão de conteúdos simples

| Funcionalidade | Resultado(*) | Descrição |
|---|---|---|
| Ao escolher no menu "conteúdos" vai para a listagem | | |

| | | |
|---|---|---|
| de conteúdos | | |
| Ao clicar no botão "adicionar conteúdo" mostra um popup para escolher conteúdo | | |
| Se escolher conteúdo composto ou colaborativo é redirecionado para a página de criação de conteúdo composto ou colaborativo | | |
| Na página de criação de conteúdo simples ao clicar no botão "guardar" fica criado o conteúdo simples | | |
| Na página de criação de conteúdo composto ou colaborativo consegue escolher o template | | |
| Na página de criação de conteúdo composto ou colaborativo consegue escolher um utilizador em cada painel | | |
| Na página de criação de conteúdo composto ou colaborativo consegue escolher um ou mais tipos de conteúdos a inserir | | |
| Na página de criação de conteúdo composto ou colaborativo ao finalizar o conteúdo este aparece na listagem conteúdos por preencher | | |
| Na página de criação de conteúdo composto ou colaborativo ao finalizar o conteúdo este aparece na listagem conteúdos por finalizar | | |
| Ao clicar no link do conteúdo para preencher é redirecionado para a página com os paineis para preencher | | |
| Ao clicar no icon do canto inferior direito o painel é expandido | | |
| Ao clicar no botão "cancelar" o painel volta ao tamanho normal | | |
| Ao clicar no botão "gravar" o conteúdo é guardado no painel | | |
| Ao preencher o conteúdo num painel consegue visualizá-lo na página de pre-visualização | | |

| | | |
|---|---|---|
| Ao clicar no link do conteúdo para finalizar é redirecionado para a página com os paineis para avaliar | | |
| Na página para finalizar o conteúdo consegue aprovar/rejeitar individualmente cada painel preenchido | | |
| Ao clicar em finalizar o conteúdo caso não estejam todos os paineis avaliados surge um popup para aprovar todos os preenchidos | | |
| Após avaliar todos os paineis é redirecionado para a página com um formulário | | |
| Na página com o formulário final do conteúdo composto ou colaborativo consegue ao clicar em guardar criar um novo conteúdo e ser redirecionado para a página com a lista de conteúdos | | |

Figura 51 – Módulo educa-content gestão de conteúdos compostos ou colaborativos

| **Funcionalidade** | **Resultado (*)** | **Descrição** |
|---|---|---|
| Ao escolher no menu "conteúdos" vai para a listagem de conteúdos | | |
| Ao clicar no botão "adicionar conteúdo" mostra um popup para escolher conteúdo | | |
| Ao escolher a opção coleção de conteúdos é redirecionado para a página de criação de coleção de conteúdos | | |
| Na página de coleção de conteúdos é apresentada uma lista de conteúdos para o utilizador escolher | | |
| Depois de escolher os conteúdos para a coleção e clicar no botão "criar coleção" é redirecionado para um página para dar permissões | | |
| Na página para dar permissões de utilizador ao clicar no botão "guardar" é criada a coleção e redirecionado para a página com a listagem de conteúdos | | |

Figura 52 – Módulo educa-content gestão de coleção do utilizador

# Capitulo 9
# Referências

[1] Steffen Staab, *The Sematic Web Revited.*

[2] John Davies, Dieter Fensel, and Frank van Harmelen, *Towards the semantic web.*

[3] Alexandre Pinto, *Slides - Introdução à Web semântica.*, 2011.

[4] Michael C. Daconta, Leo J. Obrst, and Kevin T. Smith, *Semantic Web - A Guide to the Future of XML, Web Services, and Knowledge Management.*

[5] Deborah L. McGuinness and Frank van Harmelen. (2004) OWL Web Ontology Language. [Online]. http://www.w3.org/TR/owl-features/

[6] E.D. Liddy, *Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.*, 2001.

[7] Diana Santos and Nuno Cardoso, *Reconhecimento de entidades mencionadas em português.*: Linguateca.

[8] Cristina Mota and Diana Santos, *Desafios na avaliação conjunta do reconhecimento de entidades mencionadas: O segundo HAREM.*: Linguateca, 2008.

[9] Charu C. Aggarwal and ChengXiang Zhai, *Mining Text Data.*

[10] Andrew Kachites McCallum. (2002) MALLET: A Machine Learning for Language Toolkit. [Online]. http://mallet.cs.umass.edu

[11] Mark Hall et al., *The WEKA Data Mining Software.*, 2009.

[12] Apache Mahout: Scalable machine learning and data mining. [Online]. http://mahout.apache.org/

[13] Welcome to Apache OpenNLP. [Online]. http://opennlp.apache.org/

[14] Camilla Martins, Thiago Pardo, Alice Espina, and Lucia Rino, *Introdução à sumarização automática.*

[15] W. J. Black and F. C. Johnson, *EXPERT SYSTEMS FOR INFORMATION MANAGEMENT. A Pratical Evaluation of Two Rule-Based Automatic Abstraction Techniques.*, 1988.

[16] P. B. Baxendale, *Machine-made index for tecnical literature - an experiment.*, 1958.

[17] *A Guide to the Business Analysis Body of Knowledge.*, 2009.