



ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Pesquisa de exercícios sobre a linguagem de programação Python.

Miguel Nunes (43039)

Tiago Santos (43443)

Orientador(es)

Professor Doutor João Belezã

Setembro, 2020

Resumo

Este projeto tem como objetivo a criação de um motor de pesquisa para um sistema de gestão de exercícios de correção automática. Os exercícios em causa são exercícios sobre a linguagem de programação Python. Pretende-se que a solução adotada seja desenvolvida com a linguagem Python. Para tal é utilizada a biblioteca Whoosh, que permite criar um motor de pesquisa desenvolvido em Python puro. Os exercícios são fornecidos no formato PDF. Para extrair o texto dos documentos em formato PDF, sobre o qual são efetuadas as pesquisas, é utilizada a biblioteca Python PyMUPDF.

Abstract

The goal of this project is the creation of a search engine for a system of automatic correcting exercises. This exercises are about the programming language Python. The intended solution has to be developed using the Python language. To achieve this solution the Python library Whoosh is used, which allows the creation of a search engine using only Python. The exercises are provided in PDF format. To extract the text from PDF format documents, whereupon the search is made, the Python library PyMuPDF is used.

Agradecimentos

Gostaríamos de agradecer ao nosso Orientador e Professor, João Beleza, por nos ter acompanhado e por ter estado sempre disposto a ajudar na realização deste projeto.

Gostaria também de agradecer ao nosso Professor Paulo Trigo por nos ter acompanhado numa fase inicial do projeto assim como por nos dar os alicerces para um projeto mais coerente e elaborado.

Miguel Nunes

Queria aproveitar este segmento do projeto para agradecer a todos aqueles que me acompanharam e me motivaram ao longo de todo o curso.

Queria conceder um agradecimento especial ao nosso Orientador e Professor, João Beleza por nos ter acompanhado e ajudado ao longo deste projeto, ao Professor Paulo Trigo por nos ter ajudado na fase inicial do projeto e a ambos os professores por nos ter disponibilizado esta oportunidade para desenvolvermos um projeto que servirá de base para nosso trabalho futuro.

Por último queria também agradecer aos meus pais por me terem suportado no decorrer deste curso.

Tiago Santos

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Tabelas	ix
Lista de Figuras	xi
1 Introdução	1
2 Modelo Proposto	3
2.1 Requisitos	3
2.2 Fundamentos	7
2.3 Abordagem	9
3 Implementação do Modelo	13
3.1 Criar e remover índices	13
3.2 Adicionar e remover documentos ao índice	15
3.3 Pesquisa	17
3.4 Interface gráfica	21
4 Validação e Testes	33
4.1 Conversão de ficheiros para texto	33
4.2 Simplificação de texto	35
4.3 Criação de um novo índice	35

4.4	Eliminação de um índice	39
4.5	Pesquisa	41
4.6	Pesquisa ordenada	43
4.7	Eliminação de um exercício	45
5	Conclusões e Trabalho Futuro	47
	Bibliografia	49

Lista de Tabelas

2.1	Funções do Sistema	4
2.2	Atributos do Sistema	5

Lista de Figuras

2.1	Diagrama de casos de utilização	6
3.1	Diagrama de Pesquisa	19
3.2	Menu principal	22
3.3	Menu principal ajuda	23
3.4	Menu seleção de índice	24
3.5	Menu criação de um novo índice	24
3.6	Menu verificação da criação de um novo índice	25
3.7	Menu confirmação de criação do índice	25
3.8	Menu configuração do índice	26
3.9	Menu adição de exercícios	27
3.10	Menu adição de exercícios	28
3.11	Menu pesquisa	29
3.12	Menu resultado de pesquisa	31
4.1	Teste conversão de ficheiros para texto	34
4.2	Teste de simplificação de texto	35
4.3	Teste de adição de um novo índice	36
4.4	Teste de adição de um índice já existente	36
4.5	Teste de adição de um índice já existente segunda parte	37
4.6	Teste de eliminação de um índice	39
4.7	Teste de pesquisa	41
4.8	Teste de pesquisa ordenada	43
4.9	Teste de eliminação de um exercício	45

Capítulo 1

Introdução

Nas unidades curriculares de "Matemática Discreta e Programação" e de "Matemática para Computação Gráfica", da Licenciatura em Engenharia Informática e Multimédia, do ISEL, é usado um sistema de gestão de exercícios de correção automática. Este sistema é usado para criar trabalhos de casa com exercícios sobre a linguagem de programação *Python*. A quantidade de exercícios já existentes é de tal forma grande, que dificulta a escolha de exercícios para os novos trabalhos de casa. Acontece, por exemplo, que a propósito de um determinado tema se decide escrever um novo exercício, quando se constata à posteriori que já existia um exercício parecido sobre esse tema. É neste contexto que surge a necessidade de desenvolver um motor de pesquisa que permita uma consulta eficiente destes exercícios. Pretende-se que o motor de pesquisa seja implementado usando a linguagem de programação *Python* e que seja possível ser utilizado num browser. Para que seja possível utilizar este motor de pesquisa num browser é utilizado *Jupyter notebooks* e a biblioteca Python *Voilà*.

Os exercícios em questão são ficheiros PDF, no entanto, estes geralmente são construídos a partir de um ficheiro LaTeX e de um programa Python. Como tal é necessário efectuar o processamento dos ficheiros para que seja possível a extração e o processamento do texto que se encontra nos ficheiros PDF.

Os exercícios encontram-se em directorias com todas as suas versões, cada versão com todos os ficheiros dos tipos previamente referidos. Para tal a nossa solução necessita que os exercícios sejam todos guardados no seguinte formato "question.../versionX/..." (onde o caracter X pode ser substituído por

qualquer conjunto de algarismos, dentro da pasta "question..." pode existir mais do que uma versão). Dentro de cada uma destas directorias, pretende-se efetuar a pesquisa no ficheiro "true_or_false_question.pdf", que terá de existir sempre, e ainda nos ficheiros "true_or_false_question.tex" e "program.py", caso existam.. Estas directorias podem ser directorias locais criadas pelo utilizador ou directorias criadas através de uma hiperligação para uma DropBox.

Para implementar a nossa solução é utilizado a biblioteca Python *Whoosh*, [Chaput, 2017], que permite criar um índice de documentos onde a pesquisa é posteriormente aplicada. Este índice é um repositório de informação dos documentos individuais. Para a construção deste índice é necessário um esquema, ou como é referenciado no *Whoosh*, *Schema*, que consiste num modelo de informação a ser aplicada ao documento. Quando é criado um índice é criado uma directoria com o nome dado ao índice onde são guardadas as informações relacionadas com o índice, ao esquema usado e aos documentos individuais. Esta informação é comprimida através da biblioteca *Whoosh* e a informação relacionada com os documentos é também processada por esta solução para facilitar as pesquisas. Existe também um ficheiro, denominado de "IndexFile.txt" que serve para guardar todos os índices criados pelo utilizador. Apesar de existir a funcionalidade de criar vários índices de modo a facilitar a organização dos documentos é uma funcionalidade opcional, o utilizador pode trabalhar apenas com um índice para todos os documentos.

Capítulo 2

Modelo Proposto

2.1 Requisitos

De modo a desenvolvermos o modelo proposto necessitamos de primeiro identificar quais os requisitos funcionais e não funcionais necessários para o nosso projeto.

Requisitos funcionais:

- Definir os repositórios de exercícios onde serão realizadas as pesquisas, estes repositórios podem ser locais ou na cloud.
- Pesquisar exercícios em diferentes tipos de ficheiros, esses tipos de ficheiros podem ser documentos *PDF*, *LATEX* e *Python*.
- Permitir pesquisas personalizadas onde o utilizador pode seleccionar tipos de ordenações ou limitar a pesquisa segundo um certo e determinado atributo como por exemplo o tipo de documento.
- Aplicar mecanismos de redução de palavras no texto introduzido pelo utilizador e no conteúdo dos ficheiros nos quais a pesquisa está a ser realizada de modo a otimizar a pesquisa.

Requisitos não funcionais:

- Interface simples e intuitiva
- Ser rápido e eficiente o suficiente a realizar a pesquisa de modo a não deixar o utilizador muito tempo a espera de resultados.
- Programa desenvolvido na linguagem de programação Python.

Depois de identificados e definidos os requisitos do projeto podemos começar a definir e categorizar as funções que o nosso sistema é capaz de executar e os atributos que o compõem. A categorização divide-se em 3 tipos, visível, invisível e adorno, uma função visível é uma função obrigatória de realizar e cuja a sua execução é conhecida pelo utilizador, uma função invisível também necessita de ser realiza no entanto o utilizador não tem conhecimento da sua execução e por último uma função de adorno é uma função opcional cuja a sua ausência não afeta o resto das funções. Os atributos podem ser obrigatórios ou desejáveis se não forem fundamentais para o desenvolvimento e funcionamento do projeto.

	Funções/ Atributos do sistema (Relação entre funções e atributos do sistema)	
--	---	--

Requisito	Função (Descrição do requisito)	Categoria (Visível, Invisível, Adorno)
R1	Permitir que o utilizador defina um espaço de pesquisa	Visível
R2	Indexar o conteúdo dos exercícios no espaço de pesquisa juntamente com os atributos do exercício num ficheiro de índices	Invisível
R3	Converter o conteúdo dos diferentes tipos de ficheiros dos exercícios(pdf, tex e py) para texto simples	Invisível
R4	Processar o conteúdo dos ficheiros	Adorno
R5	Pesquisar pelos ficheiros de exercícios definidos no ficheiro de índices com base num conjunto de palavras definidas pelo utilizador	Visível
R6	Permitir uma pesquisa personalizada pelo utilizador	Adorno
R7	Processamento do conjunto de palavras introduzidas pelo utilizador na pesquisa	Adorno
R8	Apresentar o resultado da pesquisa ao utilizador	Visível

Tabela 2.1: Funções do Sistema

Atributo (Identificação do atributo)	Detalhe / Restrição Fronteira (Identificação dos detalhes e/ou valores do atributo)	Categoria (Obrigatório, Desejável)
Plataforma de desenvolvimento	A plataforma de desenvolvimento necessita de ser Jupyter utilizando a linguagem de programação Python	Obrigatório
Apresentação gráfica	A parte gráfica do projeto necessita de ser desenvolvida em Python como uma aplicação web recorrendo à biblioteca Voilà	Obrigatório
Disponibilização	Disponibilizar o projeto numa máquina virtual a correr GNU Linux CentOS 8	Desejável
Interface gráfica	Desenvolver uma interface gráfica intuitiva e fácil de usar	Desejável
Tempo de execução	Tempo de execução dos pedidos do utilizador rápidos o suficiente de modo a não deixar o utilizador muito tempo à espera de uma resposta	Obrigatório

Tabela 2.2: Atributos do Sistema

Com a estrutura e funções do sistema bem definidos podemos proceder à criação de casos de utilização. Os casos de utilização permitem-nos criar narrativas que descrevem uma possível sequência de eventos provocados por um ator ao utilizar e interagir com o nosso sistema, através da narrativa criada é possível de forma abstrata descrever em que contexto o nosso sistema se aplica, perceber onde é que os requisitos anteriormente definidos se aplicam e gerar casos de teste futuros.

	Casos de Utilização - Actores (Definição dos actores do sistema)	
--	---	--

Ator	Processo(s) em que participa
Utilizador	-Definição do espaço de procura de exercícios -Pesquisa de exercícios

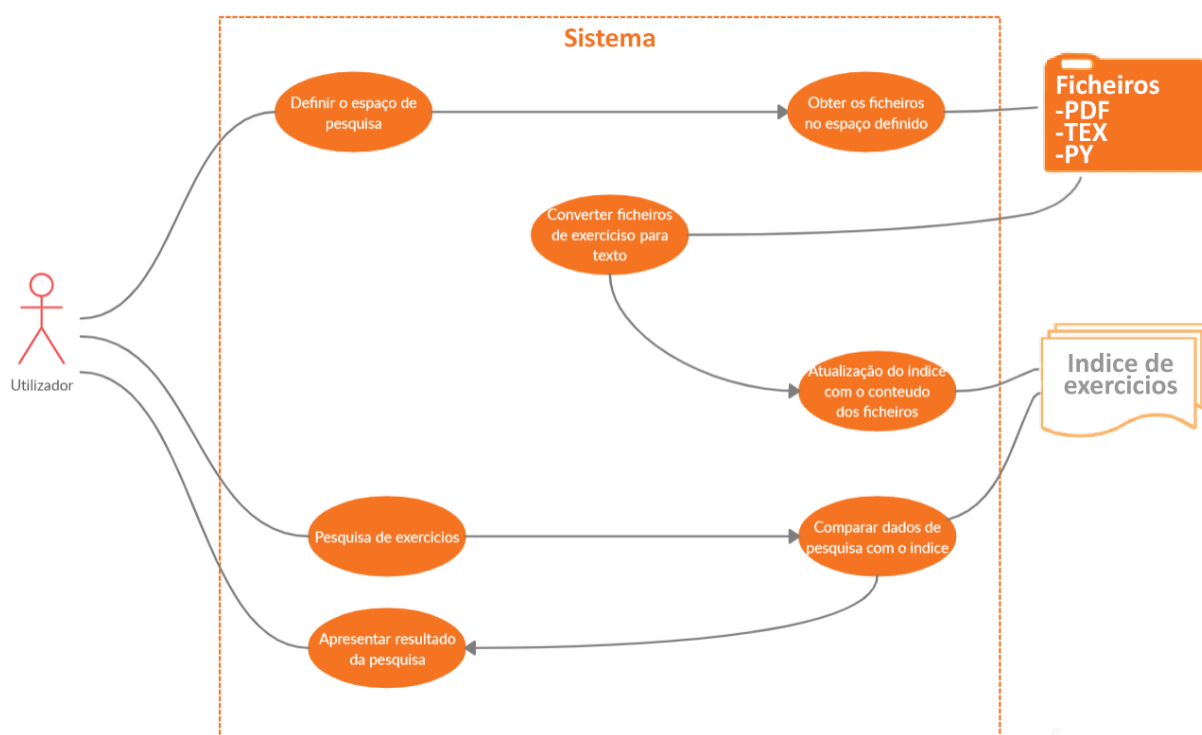


Figura 2.1: Diagrama de casos de utilização

2.2 Fundamentos

Este projeto baseia-se em 5 pontos principais:

- Definir o espaço de pesquisa;
- Converter o conteúdo dos ficheiros de exercícios dentro do espaço de pesquisa para um formato de texto simples;
- Indexar o conteúdo dos exercícios;
- Pesquisar exercícios tendo em conta as palavras introduzidas pelo utilizador no motor de pesquisa;
- Apresentar os resultados obtidos;

Uma vez que não seria eficiente que o programa fosse realizar a pesquisa em todos os ficheiros presentes no computador do utilizador, decidimos encarregar o utilizador de definir os espaços onde a pesquisa é realizada. Estes espaços de pesquisa tanto podem ser directorias que contêm um exercício, como pastas onde constam várias directorias de exercícios, o utilizador apenas necessita de especificar no programa os caminhos para as directorias de exercícios ou para as pastas que contêm essas directorias. Para que o programa consiga obter os exercícios é necessário que o utilizador os organize dentro de uma pasta com o seguinte formato `questionX/versionX/...` para que depois o programa consiga obter os seguintes ficheiros se existirem, `"true_or_false_question.pdf"`, `"true_or_false_question.tex"` e `"program.py"`.

Depois de ser definidos os espaços de pesquisa o programa tem de percorrê-los de modo a encontrar os ficheiros de exercícios `"true_or_false_question.pdf"`, `"true_or_false_question.tex"` e `"program.py"`. O conteúdo destes ficheiros é convertido para formato de texto simples e indexado, ou seja, o seu conteúdo juntamente com o caminho para a sua directoria, a data de criação e o tipo de ficheiro são guardadas num índice (pasta de ficheiros criada pelo *Whoosh*).

Com um índice de exercícios criado é possível pesquisar sobre o índice de cada exercício recorrendo a uma *query*, ou seja, é comparado os parâmetros presentes na *query* com os parâmetros de cada exercício presente no índice.

A *query* permite definir as condições que serão aplicadas numa pesquisa condições essas que definem quando é que um documento é devolvido como

resultado da pesquisa ou não. No nosso programa uma *query* é sempre composta pela condição de correspondência entre o conjunto de palavras introduzido pelo utilizador com o conteúdo dos exercícios presentes no índice. O utilizador pode depois personalizar a sua pesquisa/*query* especificando o método de ordenação (número de ocorrências ou data) e quais as restrições de pesquisa (tipo de ficheiro e data limite do exercício).

Por fim será apresentado ao utilizador os caminhos para as directorias dos vários documentos que satisfazem a sua pesquisa, se um documento for do tipo *pdf* é-lhe dada a opção de visualizar o documento na aplicação

2.3 Abordagem

No desenvolvimento do motor de pesquisa do nosso projeto tirámos partido de uma biblioteca Python denominada de *Whoosh*. Esta biblioteca permite-nos implementar um motor de pesquisa onde os vários exercícios são indexados com uma determinada estrutura num ficheiro de índices, depois deste estar criado é possível realizarmos uma pesquisa rápida sobre o seu conteúdo.

Para que seja mais fácil perceber o funcionamento desta biblioteca e como é que se aplica ao nosso projeto iremos explicar como é que a utilizamos no nosso trabalho. Primeiro criamos a estrutura de cada ficheiro a ser indexado no ficheiro de índices, essa estrutura foi a seguinte.

- Diretoria - Identificador (Caminho do ficheiro)
- Conteúdo - Texto (Conteúdo em texto simples do ficheiro)
- Data - DATETIME (Data de criação do ficheiro)
- Tag - Texto (Tipo de ficheiro, pdf, tex ou py)
- Numero de Ocorrências - Numérico (Numero de vezes que um determinado ficheiro foi adicionado ao indice)

A diretoria é necessária uma vez que funciona como identificador único do ficheiro e permite ao programa mostrar onde se encontra os exercícios que vão de encontro com a pesquisa do utilizador. O conteúdo dos ficheiros é necessário para que seja possível pesquisar por exercícios segundo um conjunto de palavras introduzidas pelo utilizador uma vez que é necessário verificar a existência dessas palavras no conteúdo dos vários exercício. A data, a *tag* e o número de ocorrências serão utilizados na personalização da pesquisa, por exemplo numa pesquisa em que o utilizador deseje que todos os exercícios que apareçam na sua pesquisa sejam ficheiros Python ordenados pela data de criação.

O próximo passo é a adição de exercícios ao índice, essa adição necessita de ser realizada com toda a informação descrita na estrutura definida acima para que toda a informação seja acessível na altura da pesquisa.

O ficheiro de índices irá permitir ao programa realizar uma pesquisa muito mais rápida e eficiente. Se não utilizássemos este ficheiro, sempre que fosse

necessário realizar uma pesquisa era necessário carregar todos os ficheiros e voltar a processar todo o seu conteúdo. Para fazer uma pesquisa com este método apenas é necessário carregar o ficheiro de índices, onde já se encontram todos os exercícios processados (convertidos para texto simples e com *Stemming* aplicado) e comprimidos. Para além disso todos os exercícios encontram-se organizados com a mesma estrutura tornando mais fácil aceder aos seus parâmetros e personalizar a pesquisa do utilizador.

Depois de um utilizador ter definido o seu índice de exercícios, pode pesquisar pelos exercícios utilizando um conjunto de palavras. A sua pesquisa pode ser ordenada de 3 formas, pela função de ordenação standard do *Whoosh*, pela data em que os ficheiros foram criados e pelo numero de vezes que um documento foi adicionado ao indice.

A função de ordenação standard utilizada pelo *Whoosh* é o *Okapi BM25F*. Esta função permite estimar a importância de cada documento numa dada pesquisa com base na quantidade de vezes que as palavras introduzidas, pelo utilizador no motor de pesquisa, aparecem em cada documento. Dada um conjunto de palavras Q que contém as palavras q_1, \dots, q_n a classificação de um documento(exercício) D é determinada pela seguinte função.

$$classificacao(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

Onde $f(q_i, D)$ é a frequência que uma determinada palavra no conjunto de palavras Q aparece no documento D . $|D|$ é o numero de palavras do documento D e $avgdl$ é a média do numero de palavras de todos os documentos presentes no índice. Os parâmetros k_1 e b são variáveis que o *Whoosh* por omissão iguala da 1.2 e 0.75 respetivamente. Por último o $IDF(q_i)$ é o peso IDF (inverse document frequency - frequência inversa do documento) da palavra q_i , calculado pela seguinte função.

$$IDF(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

Onde o valor N representa o numero total de documentos no índice e $n(q_i)$ o numero de documentos que contém a palavra q_i

Ou seja o algoritmo *Okapi BM25F*, e.g., [Garcia, 2016], irá atribuir uma maior classificação aos exercícios de maneira semelhante ao TF-IDF, ou seja

Term Frequency - Inverse Document Frequency. A parte do Term Frequency vai dar um peso ao documento diretamente proporcional à frequência de aparecimento das palavras procuradas pelo utilizador, comparativamente ao tamanho total do documento, ou seja quanto maior a frequência melhor vai ser o resultado. A parte do Inverse Document Frequency vai reduzir o peso de palavras mais comuns entre documentos enquanto aumenta o peso de palavras menos comuns. Ou seja, palavras raramente usadas entre todos os documentos guardados irão dar maior peso a documentos onde estas aparecem. [QuantStack, 2019], e a sua documentação [Voila, 2020].

A pesquisa do utilizador também pode ser limitada ao tipo de ficheiros que quer obter (pdf, tex e py) e à data limite (exercícios com uma semana de idade, um mês de idade ou um ano de idade).

Um exercício é adicionado ao resultado da pesquisa se contém pelo menos uma das palavras procurada pelo utilizador e se obedecer às limitações impostas pelo utilizador na pesquisa.

Uma vez que é necessário que o motor de pesquisa percorra diferentes tipos de ficheiros e cada tipo de ficheiro tem a sua forma distinta de processamento e leitura é necessário ter métodos para processamento de cada tipo de ficheiro. Como tal ter um método de processamento e análise para cada tipo de ficheiro tornava o programa pouco escalável e iria criar incoerência no conteúdo dos ficheiros, uma vez que dois exercícios iguais mas guardados em 2 tipos diferentes de ficheiros poderiam obter resultados diferentes na pesquisa, por isso decidimos que converter todo o conteúdo dos ficheiros para texto simples seria a melhor opção. Ao convertermos o conteúdo dos ficheiros para texto simples, apenas é necessário desenvolver um método de processamento e análise de texto, isto torna o programa mais escalável porque se quisermos adicionar mais um tipo de ficheiros aos ficheiros aceites pelo programa basta criar uma função que converta o seu conteúdo para texto simples.

Por último ainda é utilizado no projeto o *stemming* no processamento do conteúdo dos exercícios e das palavras introduzidas pelo utilizador. O *stemming* é um processo de redução de palavras flexionadas à sua raiz/base, por exemplo, reduzir a palavra "programas" a "program" visto que esta é base de várias palavras como "programador", "programa" e "programar". Ao utilizarmos *stemming* tornamos o motor de pesquisa mais flexível e não limitado

às palavras exatas que o utilizador introduz. Voltando novamente ao exemplo da palavra "programas", se a pesquisa do utilizador contém esta palavra e se um exercício contém a palavra "programa", faz sentido apresentar na lista de resultados este exercício e não excluí-lo simplesmente porque as palavras não são exatamente iguais.

Capítulo 3

Implementação do Modelo

Tal como foi previamente descrito no capítulo "Modelo Proposto" o modelo engloba:

- Criar e remover índices.
- Adicionar e remover documentos ao índice.
- Pesquisa nos documentos presentes no índice.
- Interface gráfica.

Para tal são criadas duas classes distintas. A primeira, *WhooshHandler*, tem como objectivo a manipulação de índices e documentos presentes nestes. A segunda classe é denominada de *Searcher* que serve para aplicar a procura a um dado índice.

3.1 Criar e remover índices

Para a manipulação de índices temos uma classe chamada *WhooshHandler*. Para criar índices esta classe tem o método *createSchema* que recebe um *Schema* que contém o esquema de informação que fica guardada para cada documento, no caso por definição guardamos o caminho para o documento, o conteúdo, a data de criação, o tipo de ficheiro e o número de vezes que este ficheiro foi introduzido neste índice. O que este método faz é, recebendo o *Schema* vai criar um ficheiro *IndexFiles.txt*, caso este não exista, para guardar uma lista dos índices existentes e adiciona o nome deste a lista,

de seguida criar uma directoria na directoria onde o script esta a correr onde todos os ficheiros do índice ficarão guardados.

A classe *WhooshHandler* também tem um método que serve para apagar índices, com o nome *deleteIndex* este método recebe o nome do índice a ser apagado, vai procurar este nome no ficheiro *IndexFiles.txt* para garantir que este índice esta presente na lista de índices criados pelo utilizador, caso isto se verifique o método tenta apagar a directoria com o nome do índice. No caso de ser bem sucedido o nome é também apagado do ficheiro *IndexFiles.txt*.

3.2 Adicionar e remover documentos ao índice

Tal como para a criação e eliminação de índices a classe *WhooshHandler* também tem métodos para a manipulação de documentos dentro dos índices.

Para adicionar documentos a um índice temos duas maneiras distintas, adicionar documentos individuais ou adicionar múltiplos documentos ao mesmo tempo agrupando os em uma directoria. Devido ao formato das perguntas, tem de estar numa directoria com o nome "question" com a pergunta em formato *PDF* e, opcionalmente, os ficheiros *Latex* e *python* que foram utilizados para criar a pergunta dentro desta, é possível utilizar o mesmo método para ambas as maneiras de adicionar documentos. O formato das perguntas vem do formato previamente existente utilizado pelo professor coordenador deste projecto. Este método tem o nome de *writeSchema* que vai percorrer a directoria dada pelo utilizador a procura de directorias com "question" no nome, ao encontrar uma directoria com esse parâmetro vai aceder a primeira versão da pergunta e envia o caminho para o método *writeSchema2* que vai tentar ir buscar a pergunta em formato *PDF* e os ficheiros *Latex* e *python* e adiciona os ao índice.

Antes destes documentos serem adicionados é preciso ainda haver algum processamento de texto, no caso do ficheiro *PDF* é preciso converter o documento de imagem para texto, para tal utilizamos a biblioteca python *PyMuPDF*, para *Latex* utilizamos a biblioteca python *pylatexenc* para obtermos o texto sem as tags específicas ao *Latex* e por fim para *python* não é necessário nada pois este já se encontra em texto simples. Por ultimo é ainda utilizado um analisador de texto do *Whoosh*, este analisador tem como função simplificar o texto reduzindo-o às palavras mais importantes do texto, como tal utilizamos um analisador de linguagem, com a língua Portuguesa como opção, para se criar um filtro de letras minúsculas que serve para converter todas as palavras para letras minúsculas, uma lista de Stop words que vai retirar palavras mais frequentes e que não adicionam ao conteúdo do texto, por exemplo as palavras "e", "a", "na" e "no", e por ultimo um filtro de Stemming que vai ser aplicado ao texto para reduzir as palavras a sua palavra raiz se possível facilitando assim a pesquisa ao utilizador.

Quando um documento é adicionado ao índice o conteúdo deste é comparado com o conteúdo de documentos já presentes no índice, caso se verifique que este documento já existe em vez de ser adicionado é atualizado o numero

de ocorrências do documento já presente no índice. Isto para que, em vez de haver varias entradas do mesmo exercício no índice podemos dar um maior peso a este nas pesquisas.

Para o caso de eliminar documentos do índice é dado esta escolha ao utilizado quando este procura por exercícios. Caso o utilizador queira eliminar um documento é chamado o método *deleteEntry* da classe *Seacher* com o caminho da directoria onde os ficheiros se encontram. para que quando se elimine a entrada de um ficheiro é eliminado o *PDF* assim como as entradas dos ficheiros *Latex* e *python* correspondente a este.

3.3 Pesquisa

A parte do projeto responsável pela realização da pesquisa de exercícios é a classe *Searcher*. Quando um utilizador deseja realizar uma pesquisa começa-se por criar uma instância da classe *Searcher* com o nome do índice onde a pesquisa irá ser feita. No construtor da classe *Searcher* é, em primeiro lugar, feito a verificação de que o nome do índice passado se encontra-se corretamente formado e que de facto representa o nome de um índice existente, caso a verificação falhe em algum dos pontos é lançada uma exceção com a indicação da falha que ocorreu. Esta verificação é necessária uma vez que impede que a pesquisa seja feita num índice inexistente e informa o utilizador de um possível erro ou problema com o nome do índice escolhido. Após a verificação é guardado numa variável global o nome do índice pretendido.

A classe *Searcher* contém os métodos *parser* e *deleteEntry*.

O método *parser* tem como objetivo analisar as palavras e os parâmetros de personalização de pesquisa introduzidos pelo utilizador no motor de pesquisa, percorrer o ficheiro de índice(definido no construtor da classe) em busca dos exercícios que vão de encontro com a pesquisa do utilizador e por ultimo devolver o resultado da pesquisa. O método *parser* recebe como argumento uma *string* composta pelas palavras que se pretende procurar nos exercícios, uma *string* que representa o tipo de documentos a ser procurado (pdf, tex, py ou qualquer tipo), uma *string* que indica o tipo de classificação (classificação predefinida, pelo numero de vezes que o exercício ocorre ou pela data de criação) e uma *string* que indica a idade limite dos documentos (um ano, um mês ou uma semana). Quando invocado, o método *parser* começa por chamar o método *analyzeText* passando como argumento as palavras utilizadas na pesquisa de modo a aplicar-lhes o processo de redução de palavras flexionadas *stemming*, de seguida é iniciada a construção da *query* de pesquisa, esta *query* irá conter todos os parâmetros definidos pelo utilizador na pesquisa e será usada na correspondência desses parâmetros com os exercícios presentes no espaço de pesquisa (Ver figure 3.1). A *query* terá o seguinte formato:

- -Conteúdo: palavras utilizadas na pesquisa (Utilizado na comparação com o conteúdo(texto) dos exercícios)
- -Data: data mínima até data máxima (Utilizado na comparação com

a data de criação dos exercícios)

- -Etiqueta: tipo de documento (Utilizado na comparação com a extensão dos exercícios, utilizado apenas se especificado)

De modo a obter a data limite de criação dos exercícios é necessário primeiro calcula-la com base na data atual e a idade máxima que um documento pode ter segundo o que foi estipulado pelo utilizador. Após a data limite ter sido calculada, é adicionada a *query*, apenas se necessário, o tipo de documento que o utilizador deseja obter. Com a *query* criada procede-se a realização da pesquisa. A pesquisa começa pela criação de um objeto *Searcher* associado ao índice definido, este objeto é criado recorrendo a uma declaração *with* para que seja fechado automaticamente assim que a sua utilização for concluída, é bastante importante fechar este objeto uma vez que objetos do tipo *Searcher* representam uma série de ficheiros abertos que se não forem devidamente fechados poderá afetar o desempenho do programa ou o programa pode acabar por ficar sem *file handles*. O objeto *Searcher* será responsável pela leitura do ficheiro de índice ao qual está associado. Antes de se realizar a pesquisa ainda é preciso converter a *query* para *query objects* (objetos do módulo *whoosh.query*), este objeto irá permitir realizar a correspondência entre os campos de pesquisa introduzidos pelo utilizador com os respetivos campos nos documentos indexados (com a estrutura definida pelo esquema no ficheiro de índices). A seguir realiza-se a pesquisa recorrendo ao método *search* do objeto *Searcher* anteriormente apresentado. Este método recebe como argumento o objeto *query* anteriormente criado, o tipo de ordenação e o numero limite de exercícios que devolve, através da *query* recebida este método verifica em que documentos a *query* é obedecida e retorna-os ordenados pelo tipo de ordenação definido.

O seguinte diagrama apresenta de forma simplificada como é que a pesquisa é realizada.

O método *deleteEntry* tem como objetivo remover um determinado exercício do ficheiro de índices especificado na inicialização da classe *Searcher*. Este método recebe uma string onde se encontra especificado o caminho para a diretoria onde se encontra o exercício que se deseja remover e começa por verificar se o ficheiro de índices existe, para que caso não existe retornar uma mensagem de erro e cancelar a operação de eliminação de exercício.

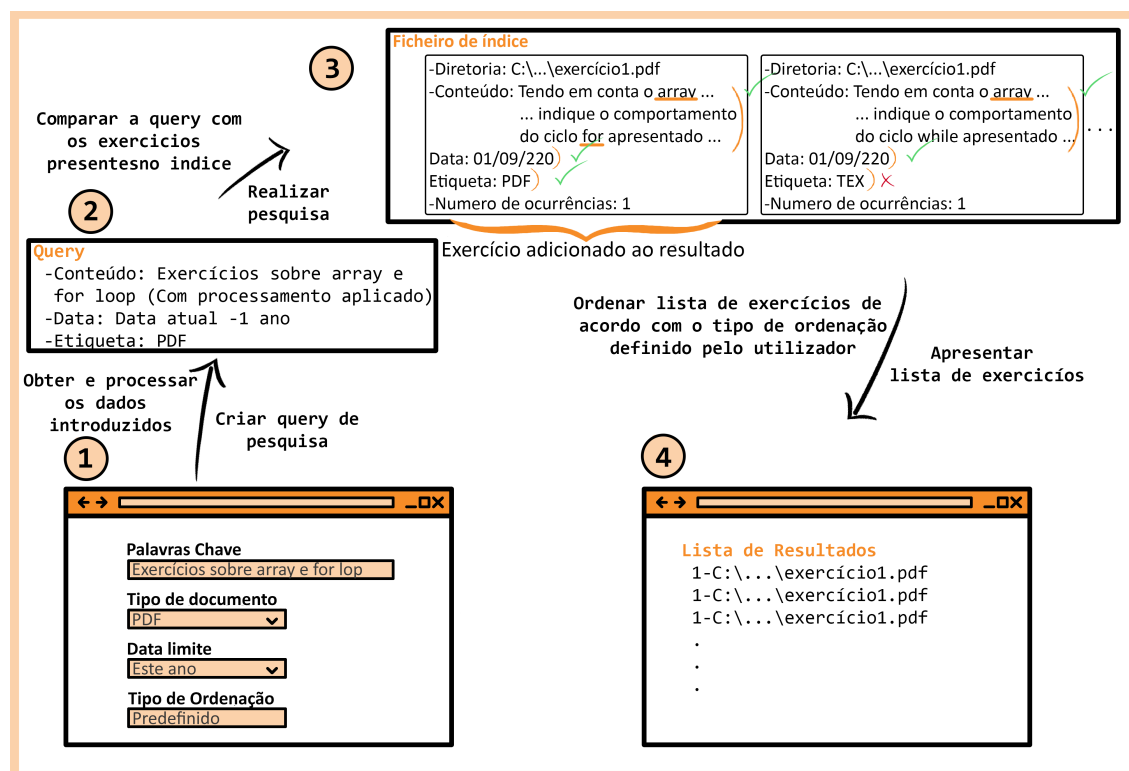


Figura 3.1: Diagrama de Pesquisa

De seguida o método elimina todos os exercícios presentes no índice cujo parâmetro *“path”* seja igual ao caminho recebido juntamente com as extensões, *“true_or_false_question.pdf”*, *“program.py”*, *“true_or_false_question.tex”*, para que sejam removidos, se existirem os diferentes formatos do exercício. Por último é retornado uma mensagem que indica se a operação foi um sucesso ou se falhou, devido a poder ocorrer exceções na remoção de exercícios.

3.4 Interface gráfica

A interface gráfica tem como objetivo apresentar ao utilizador um espaço visual intuitivo e com secções de ajuda onde este possa interagir com o programa de forma mais facilitada.

No nosso projeto a interface gráfica foi desenvolvida recorrendo à biblioteca python *ipywidgets* e apresentada no *browser* recorrendo ao *Voilà*. Foi escolhido o *Voilà* para a apresentação da interface de utilizador no *browser* devido às vantagens que este acrescenta sobre o *Jupyter Notebook*. Se utilizássemos apenas o *Jupyter Notebook* seria apresentado ao utilizador a interface gráfica no formato *html* juntamente com os blocos de código o que iria permitir que o utilizador corresse código na máquina onde o programa estava a ser executado. Ao acrescentarmos o *Voilà* ao *Jupyter Notebook*, o *Jupyter Notebook* é convertido numa aplicação web onde apenas é apresentado a interface de utilizador impedindo assim o utilizador de inserir código arbitrário. Outra vantagem do *Voilà* é a de ser executado um processo por utilizador [QuantStack, 2019], [Voila, 2020].

O *ipywidgets* permite criar objetos python do tipo *Widgets* estes objetos podem depois ser utilizados pelo browser de modo a apresentarem texto, botões, *sliders*, entre outros elementos gráficos. Ao juntar vários dos elementos gráficos mencionados anteriormente é possível construir uma aplicação com uma interface do utilizador que recebe entradas, através de botões ou caixas de texto, processa-as e devolve o resultado associado às entradas sobre forma de texto [Jupyter, 2019].

O primeiro passo no desenvolvimento da interface gráfica é a definição da janela de saída que irá conter os vários elementos gráficos, esta janela é criada recorrendo ao *widget output*. Existem 2 janelas na aplicação a primeira permite ao utilizador escolher o menu a que pretende aceder e a segunda janela apresenta o menu escolhido pelo utilizador.

A primeira janela, definida na função *firstWidget*, é estática e é composta por um título e 4 botões. O título é construído através do *widget HTML* que nos permite escrever exatamente qual a tag HTML que pretendemos que apareça na aplicação web. A razão pela qual escolhemos utilizar o elemento HTML e não um dos outros elementos de texto que o *ipywidgets* disponibiliza é devido a já conhecermos HTML e por ser mais flexível na customização do texto, por esses dois motivos todos os elementos que não necessitam de

entrada por parte do utilizador serão representados pelo *widget* HTML. Os botões apresentados são realizados através do *widget* *BUTTON* o que permite associar a cada botão um evento *on click* que irá realizar uma determinada ação, esta ação geralmente é chamar uma nova função. Os botões encontram-se agrupados dois a dois, onde os botões à esquerda abrem o seu respetivo menu na segunda janela e os botões na direita abrem uma pequena janela (por debaixo do grupo de botões) com texto explicativo referente à função de cada menu. Para agrupar os botões é utilizado o *widgets* *HBox*, este *widget* recebe vários *widget* e agrupa-os horizontalmente. No final da função com o objeto *output* da primeira janela é chamada a função *display* em que os seus argumentos serão o título, os dois grupos de botões e as duas janelas onde é apresentado o texto de ajuda.



Figura 3.2: Menu principal

Quando o utilizador clica num dos vários botões de ajuda presentes na aplicação é chamada a função *showHelp*. Esta função tem como objetivo mostrar/esconder determinado texto numa determinada janela de *output*, para tal recebe nos seus argumentos a janela de *output* (*outHelp*), onde é suposto apresentar o texto de ajuda, e o texto de ajuda(*text*) a ser escrito. A função começa por verificar qual o estado atual da visibilidade da janela, no caso de esta se encontrar visível, é necessário alterar o seu estado para escondido e eliminar todo o seu conteúdo. Caso o estado da visibilidade se encontre igual a escondido é necessário muda-lo para visível e exibir no seu conteúdo o texto.

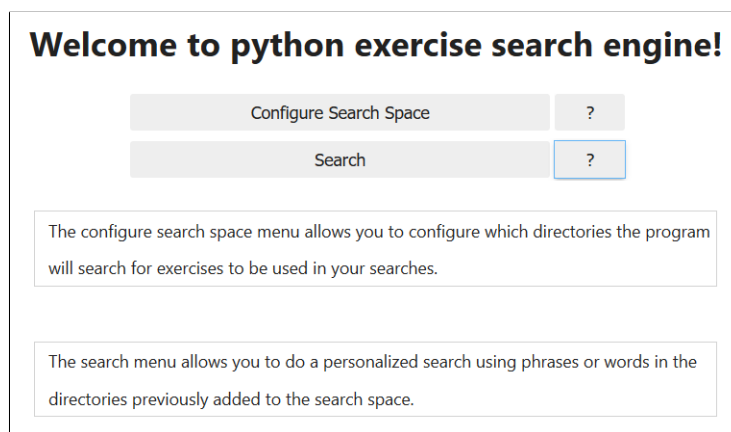


Figura 3.3: Menu principal ajuda

Quando o utilizador clica num dos botões de apresentação de menu presentes na primeira janela é chama a função *defineIndex*. Esta função constrói uma janela com os elementos necessários para que o utilizador escolha qual o índice que deseja escolher para as suas operações futuras. Esta função apenas recebe um parâmetro denominado de *operation* que indica qual o menu a ser apresentado ao utilizador. Se o utilizador tiver clicado no botão de configuração de espaço de pesquisa então esta função é chamada com o parâmetro *operation* igual a "configure", caso o utilizador tiver clicado no botão de pesquisa a função é chamada com o parâmetro *operation* igual a "search". Antes da página referente a operação escolhida ser construída é verificada a existência de ficheiros de índices. Se já existirem um ou mais ficheiros de índices então é construída a página referente ao menu selecionado onde consta uma caixa de texto do tipo *DropDown* que permite ao utilizador

selecionar um dos índices já criados anteriormente, no caso de se encontrar no menu de configuração de pesquisa também lhe será apresentada a opção de criar um novo índice. Se não existir nenhum índice previamente criado então o utilizador será redirecionado para a página de criação de índice.

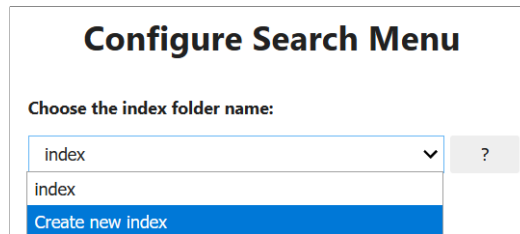


Figura 3.4: Menu seleção de índice

Na página de criação de índice é apresentado ao utilizador uma caixa de texto para que este insira o nome do índice que pretende criar, esta caixa é acompanhada de um botão de ajuda que explica ao utilizador para que serve o índice. Nesta página o utilizador pode cancelar a operação premindo o botão de *cancel* ou submeter o novo índice.

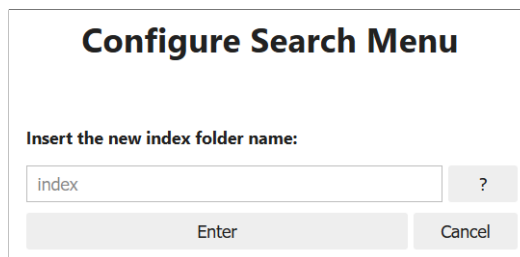


Figura 3.5: Menu criação de um novo índice

Quando o utilizador submete o nome de um novo índice é chamada a função *verifyNewIndex* passando como argumento o nome do novo índice. Esta função tem como objetivo analisar o nome introduzido, de modo a impedir que seja introduzido um nome inválido e caso seja introduzido um nome repetido avisar o utilizador. Um nome de índice é considerado inválido se for vazio, caso o utilizador introduza um nome vazio a função *verifyNewIndex* constrói uma página que indica o problema e disponibiliza um botão para que o utilizador volte à seleção de índices. Se o utilizador introduzir um nome válido, compara-se o nome com os nomes de índices já presentes no ficheiro

IndexFiles.txt. Se o nome do novo índice já constar no ficheiro *IndexFiles.txt*, então é construída uma página que avisa o utilizador desta ocorrência e pergunta ao utilizador se este deseja continuar com a operação e escrever por cima do ficheiro já existente, juntamente com a pergunta é apresentado um aviso e dois botões, um para que a operação continue e outro para cancelar a operação de criação de um novo índice.

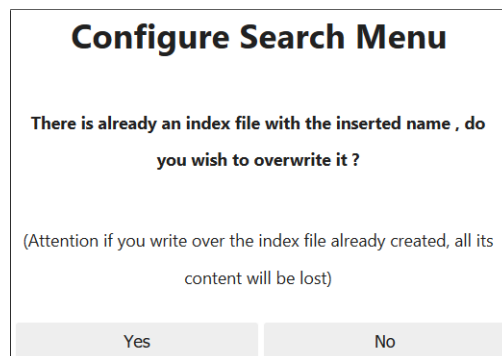


Figura 3.6: Menu verificação da criação de um novo índice

Se o nome for válido e ainda não existir nenhum índice com esse nome então é chamada uma nova função *confirmNewIndex* passando como argumento o nome do novo índice.

Na função *confirmNewIndex*, é criada uma instância da classe *WhooshHandler*, com o nome do novo índice, e chamada a função que cria um novo índice(*createSchema()*) igualando o resultado da função a uma variável. Por fim a função cria uma página onde descreve o resultado da criação do novo índice (se foi bem sucedido ou não) e um botão para que o utilizador volte à seleção de índices.

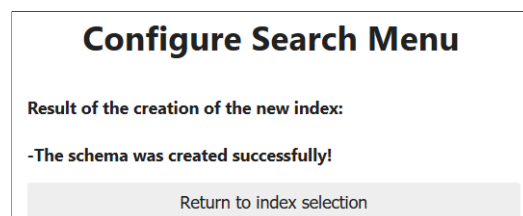
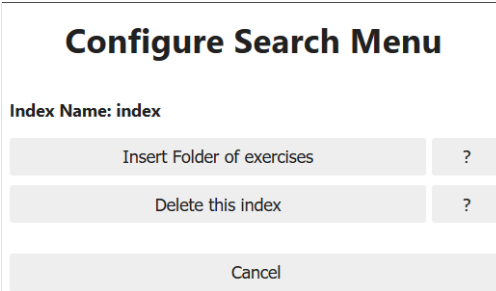


Figura 3.7: Menu confirmação de criação do índice

Quando um índice é selecionado na página de seleção de índices com o valor do parâmetro *operation* igual a *configure* é chamada a função *create-*

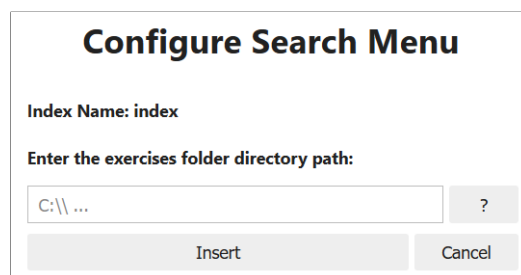
ConfigureSearch() passando como argumento o nome do índice selecionado. Se o nome do índice for a opção de criar um novo índice então é chamada a função *createNewIndex()*, caso contrário é verificada a existência do ficheiro selecionado pelo utilizador no ficheiro *IndexFiles.txt*. Caso o índice escolhido não conste no ficheiro de índices criados então é apresentada uma página de erro e um botão para que o utilizador volte à página de seleção de índices, caso contrário é construído o menu de configuração de pesquisa para o índice selecionado e inicializada uma nova instância de *WhooshHandler()* com o índice selecionado. Este menu é composto por 2 botões principais seguidos dos seus respetivos botões de ajuda, que indicam a função dos botões, e um quarto botão para que o utilizador volte à seleção de índices. Os 2 botões principais são um botão para inserir novos exercícios ao índice e um botão para eliminar o índice escolhido.



Configure Search Menu	
Index Name: index	
Insert Folder of exercises	?
Delete this index	?
Cancel	

Figura 3.8: Menu configuração do índice

Quando o utilizador seleciona o botão de inserir novos exercícios é chamada a função *insertExercises* passando como argumento o nome do índice e a instância de *WhooshHandler()* previamente criada, esta função apresenta ao utilizador uma caixa de texto para que este introduza o caminho para a diretoria onde será realizada a procura de exercícios, adicionalmente é apresentado um botão de ajuda para que o utilizador perceba o que tem de introduzir na caixa de texto e 2 botões, o botão de submeter e o botão de cancelar operação que redireciona o utilizador para a página anterior.



Configure Search Menu

Index Name: index

Enter the exercises folder directory path:

C:\\ ... ?

Insert Cancel

Figura 3.9: Menu adição de exercícios

Se o utilizador decidir submeter uma nova diretoria de exercícios então é chamada a função *insertExercisesResult()* com os seguintes argumentos, a instância de *WhooshHandler()*, o nome do índice e o caminho para a diretoria escolhida. Nesta função é chamado o método *writeSchema()* do *WhooshHandler()* passando como argumento o caminho definido, de seguida a função constrói uma página onde indica o resultado da inserção dos novos exercícios, para que o utilizador consiga perceber se os exercícios foram todos adicionados corretamente. Por fim, se o utilizador decidir selecionar a opção de eliminar o índice é chamada a função *deleteIndex* passando como argumento a instância de *WhooshHandler()* e o nome do índice, de seguida é chamada a função *deleteIndex()* do *WhooshHandler()*. Por fim a função constrói uma página com o resultado da eliminação do índice devolvido pela função *deleteIndex()*.

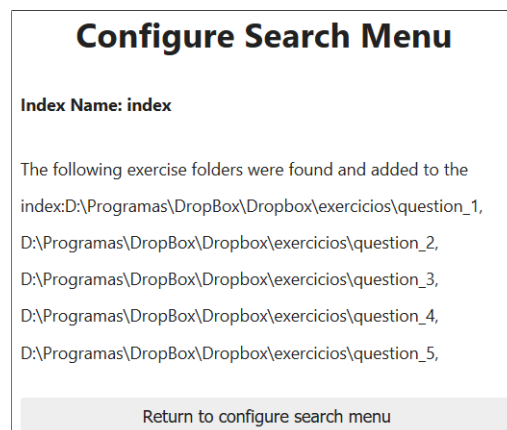
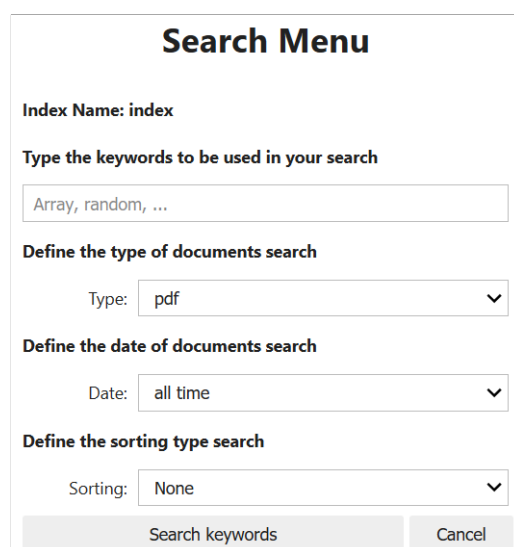


Figura 3.10: Menu adição de exercícios

Quando um índice é selecionado na página de seleção de índices com o valor do parâmetro *operation* igual a *search* é chamada a função *createSearch()* passando como argumento o nome do índice selecionado. Antes de ser criada a página de pesquisa é necessário primeiro verificar a existência do índice escolhido pelo utilizador no ficheiro *IndexFiles.txt* semelhante ao que acontece na função *createConfigureSearch()* de modo a que caso o índice seja inválido seja apresentado ao utilizador uma página com essa informação. Se o índice for válido então é construída uma página com todas as opções de pesquisa para que o utilizador personalize a pesquisa conforme as suas necessidades, essas opções de pesquisa são, uma caixa de texto onde são introduzidas as

keywords(palavras chaves introduzidas no motor de pesquisa) de pesquisa, uma caixa do tipo *drop down* onde é selecionado o tipo de ficheiros a aparecer na pesquisa, uma caixa do tipo *drop down* onde é selecionado a idade máxima de um documento e por ultimo o tipo de ordenação de resultados. No final da página são apresentados 2 botões um para progredir com a pesquisa e outro para cancela-la e voltar à seleção de índices. Esta função para além do parâmetro mencionado anteriormente também pode receber uma instância da classe *Searcher* para que não seja necessário criar sucessivas instâncias da classe sempre que o utilizador fizer sucessivas pesquisas, para tal, antes de ser criada uma instância da classe *Searcher* é verificado se foi recebido uma nos seus argumentos.



The image shows a web-based search interface titled "Search Menu". It contains several input fields and dropdown menus for configuring a search. The "Index Name" is set to "index". The "Keywords" field contains the text "Array, random, ...". The "Type" dropdown is set to "pdf". The "Date" dropdown is set to "all time". The "Sorting" dropdown is set to "None". At the bottom, there are two buttons: "Search keywords" and "Cancel".

Search Menu	
Index Name: index	
Type the keywords to be used in your search	
<input type="text" value="Array, random, ..."/>	
Define the type of documents search	
Type:	<input type="text" value="pdf"/>
Define the date of documents search	
Date:	<input type="text" value="all time"/>
Define the sorting type search	
Sorting:	<input type="text" value="None"/>
<input type="button" value="Search keywords"/> <input type="button" value="Cancel"/>	

Figura 3.11: Menu pesquisa

Se o utilizador prosseguir com a pesquisa então é necessário criar uma página com os resultados, essa página é feita através da função *search-Keyword()* passando como argumento o nome do índice, a instância da classe *Searcher* e todos os parâmetros de pesquisa definidos na página anterior. A primeira operação desta função é a realização da pesquisa recorrendo ao método *parser* da classe *Searcher* passando como argumento todos os valores dos parâmetros de pesquisa escolhidos pelo utilizador. O método *parser* retorna o resultado da pesquisa e uma lista com todos os exercícios que obedecem a pesquisa personalizada do utilizador. Se a pesquisa retornar uma

lista vazia então é apresentado ao utilizador uma mensagem com essa informação caso contrário é construída uma página com a lista de exercícios. Para construir a lista de exercícios é necessário percorrer o resultado da pesquisa recorrendo a um ciclo `for`, para cada elemento do resultado é obtido o caminho de modo a apresentá-lo na lista de resultados e juntamente com o caminho é apresentado um botão de eliminar que permite ao utilizador eliminar o exercício do índice onde foi feita a pesquisa. Se o exercício for um documento do tipo *PDF* então é possível apresentá-lo na lista de resultados como uma imagem, no entanto não seria prático nem graficamente apelativo ter texto e imagens intercalados na pesquisa, por isso, aos exercícios *PDF*, para além de ser apresentado o caminho e o botão de eliminar também é apresentado um botão de mostrar *PDF* que apresenta/esconde uma imagem do exercício. Para que isto seja possível, ao percorrer a lista de resultados é verificado se o documento é do tipo *PDF*, se for então é criada a imagem recorrendo à função *showPdf* passando como argumento o caminho para o ficheiro *PDF*, criada uma caixa de *output* vazia por de baixo do caminho do exercício para mostrar a imagem e criado um botão de mostrar/esconder o *PDF* com a ação de *on click* para chamar a função *showImage* com os parâmetros, caixa de *output* e a imagem a ser colocada na caixa de *output*.

A função *showPdf* tem como objetivo criar uma imagem com base num documento *PDF*, para tal recebe como parâmetro o caminho do documento pdf, obtém a primeira página desse documento e cria uma imagem como um *pixMap*. Como o *Image* do *ipywidgets* não aceita imagens *pixMap* para serem apresentadas, necessitamos de converter a imagem para outro formato como *PNG* esta conversão é obtida recorrendo à função *getPNGData()* da biblioteca *fitz*. Por fim é criada uma *ipywidgets* do tipo *Image* com a imagem em formato *png* obtida e retornado como resultado da função.

A função *showImage* tem como objetivo mostrar/esconder um *ipywidgets* do tipo *output* e apresentar uma imagem no seu conteúdo caso o seu estado seja visível. Para tal a função necessita de receber o *ipywidgets* do tipo *output* e uma imagem. A função começa por verificar a visibilidade do *output*, se este se encontra atualmente visível, então altera o estado da sua visibilidade para escondido e elimina o seu conteúdo, caso contrário o estado da sua visibilidade é alterado para visível e colocado no seu conteúdo a imagem recebida.

Search Result

Index Name: index

[Return to search menu](#)

0 - D:\Programas\DropBox\Dropbox\exercicios\question_1\version_1\true_or_false_questio...	Open PDF	Delete
<div>Considere o programa, Python 3, que se segue. Ignore a variável <code>seed</code> e a função <code>pseudo_random_integer</code>, que se destinam exclusivamente à geração de números pseudo-aleatórios.</div> <div><pre>seed = 1116111 def pseudo_random_integer(min_int , max_int): global seed seed = (16807*seed) % 2147483647 return int(min_int + (max_int - min_int) * seed / 2147483646)</pre></div> <div><pre>h = [] for p in range(19624): h.append(pseudo_random_integer(750, 1750))</pre></div> <div>Acrescente a este programa o código que lhe permita indicar se as afirmações seguintes são verdadeiras ou falsas. Indique se é verdadeiro ou falso.</div>		
1 - D:\Programas\DropBox\Dropbox\exercicios\question_3\version_1\true_or_false_questio...	Open PDF	Delete
2 - D:\Programas\DropBox\Dropbox\exercicios\question_4\version_1\true_or_false_questio...	Open PDF	Delete

Figura 3.12: Menu resultado de pesquisa

Capítulo 4

Validação e Testes

Os seguintes testes foram realizados à mão de modo a simular o comportamento natural do utilizador e testar as várias funcionalidades do projeto. Estes testes servem para:

1. Testar a conversão dos diferentes tipos de ficheiros para texto simples pronto a ser processado.
2. Testar a simplificação do texto usando o analisador que foi criado.
3. Testar o processo de criação de um novo índice.
4. Testar o processo de eliminação de um índice.
5. Testar o processo de pesquisa sobre um índice já criado.
6. Testar os diferentes modos de ordenação de uma pesquisa.
7. Testar o processo de eliminação de um exercício.

4.1 Conversão de ficheiros para texto

No teste de conversão de ficheiros para texto decidimos chamar o método *decodePdf()* passando como argumento um ficheiro *PDF* e chamar o método *decodeTex* passando como argumento um ficheiro *TEX*. A seguinte figura apresenta o ficheiro *PDF/Latex* original e o texto que resultou da sua conversão.

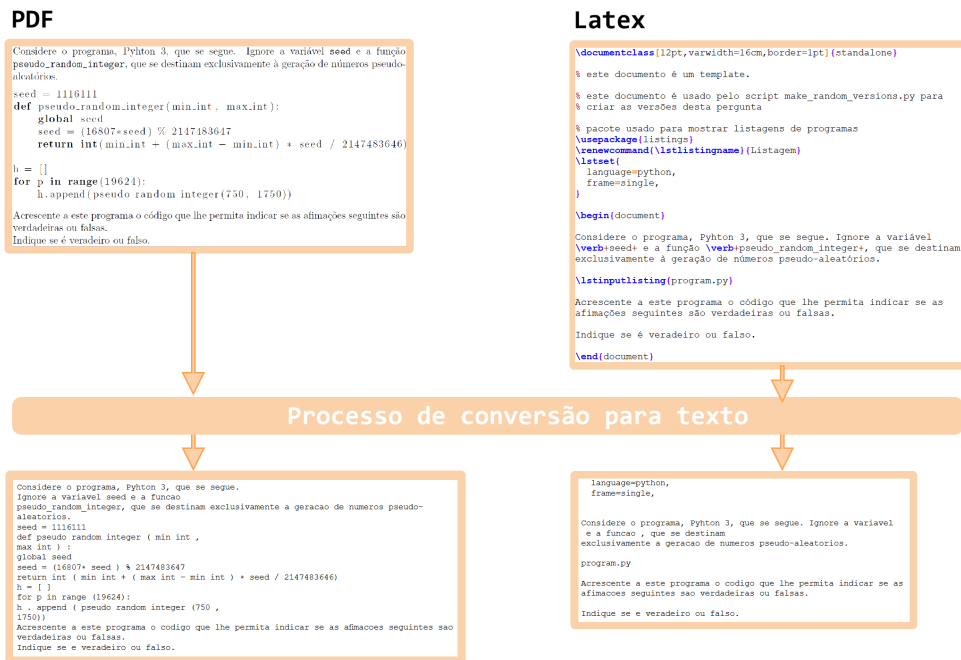


Figura 4.1: Teste conversão de ficheiros para texto

É possível observar que ambos os ficheiros *PDF* e *Latex* que após passarem pelo processo de conversão para texto foram transformados num ficheiro de texto simples onde é possível observar todo o seu conteúdo relevante. Infelizmente foi perdida informação com acentos, este perda deve-se ao tipo de codificação de caracteres que separa os acentos das letras a que estão associadas, por esse motivo decidimos ao converter os ficheiros para texto remover todos os acentos.

4.2 Simplificação de texto

No teste de processamento e simplificação de texto necessitamos de verificar se as palavras vazias (palavras sem utilidade para o processamento e pesquisa) são removidas e se as palavras são reduzidas de forma aproximada à sua palavra raiz. Para tal passamos os textos obtidos no teste anterior pelo método *analyzeText()*. A seguinte figura apresenta os resultados obtidos.

PDF após conversão para texto e simplificação de texto

```
consider program pyhton seg ignor variavel seed funca pseudo_random_integ destin exclus geraca numer pseud aleatori s
eed 1116111 def pseud random integ min int max int global seed seed 16807 seed 2147483647 return int min int max int
min int seed 2147483646 rang 19624 append pseud random integ 750 1750 acrescent program codig permit indic afimaco seg
uint sao verdadeir fals indigu veradeir fals
```

Latex após conversão para texto e simplificação de texto

```
languag python fram singl consider program pyhton seg ignor variavel funca destin exclus geraca numer pseud aleatori
program.py acrescent program codig permit indic afimaco sequint sao verdadeir fals indigu veradeir fals
```

Figura 4.2: Teste de simplificação de texto

Na figura 4.2 é possível observar a falta de palavras vazias como "o", "que", "e" e "a" e a redução das palavras para a sua raiz, como por exemplo a palavra programa passou a program.

4.3 Criação de um novo índice

No teste de criação de um novo índice deve-se verificar 2 cenários, no primeiro cenário é adicionado um índice novo e verificado se este foi criado com sucesso e adicionado ao ficheiro de texto *IndexFiles.txt* com nome do novo índice, no segundo cenário é adicionado um índice já existente e verificado se o utilizador é alertado da existência de um índice repetido, ao proceder com a adição de um índice que já exista é depois necessário confirmar que o conteúdo do índice encontra-se vazio.

Na figura 4.3 é possível observar o processo de adição de um novo índice, onde no ponto 1 a lista de índices apenas contem "index" e no ponto 4 já contem o novo índice criado "indiceTeste". Visto que o menu *drop down* apresentado na figura é uma listagem do conteúdo do ficheiro *IndexFiles.txt* é possível confirmar que este foi atualizado com sucesso.

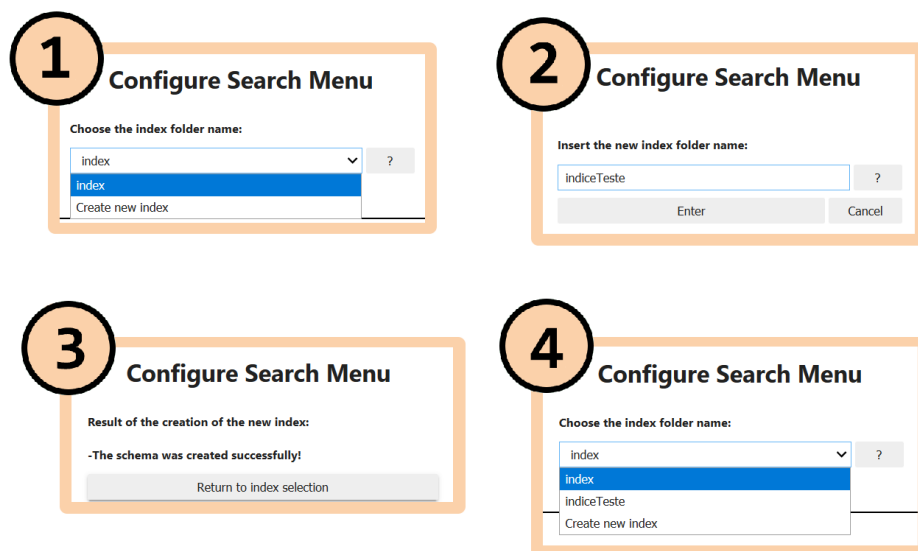


Figura 4.3: Teste de adição de um novo índice

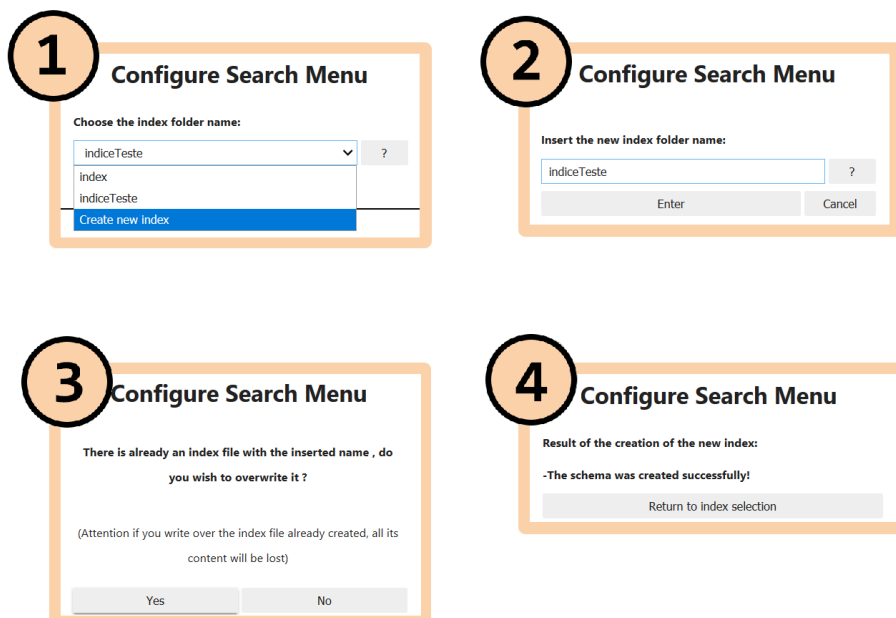


Figura 4.4: Teste de adição de um índice já existente

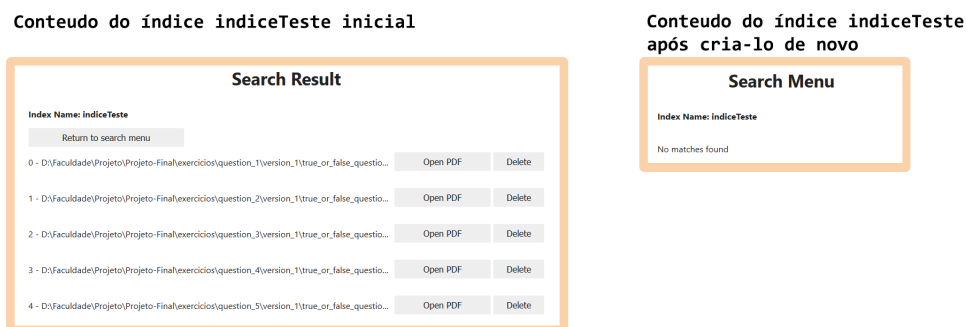


Figura 4.5: Teste de adição de um índice já existente segunda parte

Depois de adicionado o novo índice *"indeceTeste"*, foi-lhe adicionado uma série de exercícios. Na figura 4.4 é possível observar o processo de adição de um índice já existente. Na figura 4.5 esta presente os exercícios que constam no índice antes de ser adicionado um índice com o mesmo nome e depois de ser adicionado um índice com o mesmo novo, podendo assim confirmar que todo o conteúdo anterior do índice foi eliminado.

4.4 Eliminação de um índice

No teste de eliminação de índice é verificado se o índice foi eliminado com sucesso e o seu nome já não consta no ficheiro de texto *IndexFiles.txt*.

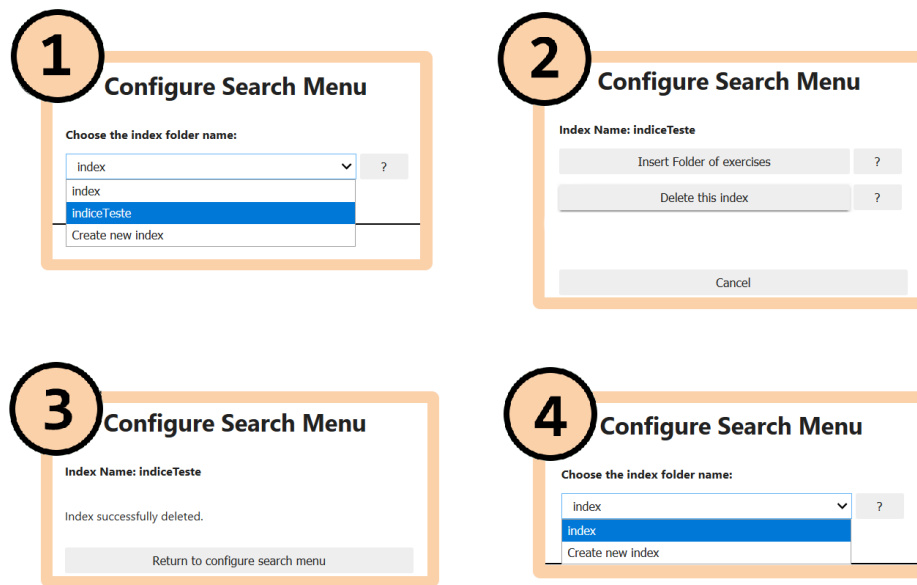


Figura 4.6: Teste de eliminação de um índice

Na figura 4.6 é possível observar o processo de eliminação de um índice, onde no ponto 1 a lista de índices contém "index" e "indiceTeste" e no ponto 4 apenas contém "index".

4.5 Pesquisa

No teste de pesquisa pretende-se obter um determinado exercício do tipo *PDF* com a seguinte frase "Sistema de equações lineares". Após ser realizada a pesquisa foram obtidos os seguintes resultados.

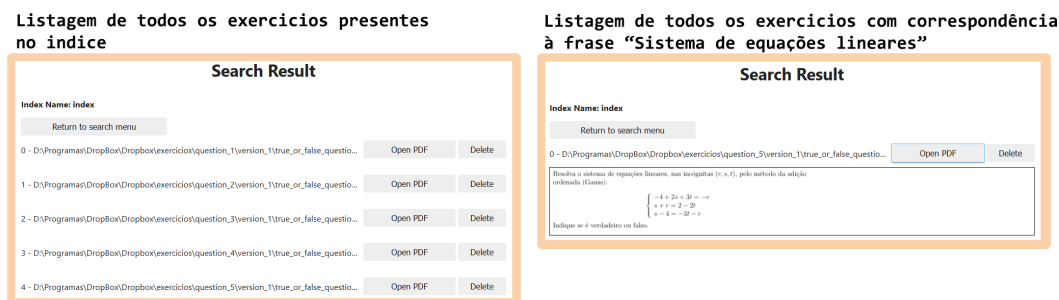


Figura 4.7: Teste de pesquisa

Na figura 4.7 é possível observar uma lista de todos os exercícios *PDF* e uma lista de todos os exercícios *PDF* obtidos nos resultados contém a frase, "Sistema de equações lineares", introduzidas no motor de pesquisa. é possível observar no resultado da pesquisa ao abrir o documento *PDF* que de facto o exercício se trata de um exercício sobre equações lineares.

4.6 Pesquisa ordenada

Neste teste pretende se ver a ordenação de resultados a ser feita consoante os parâmetros escolhidos, neste caso esta será pelo numero de ocorrências que um exercício tem na base de dados.

The figure consists of two screenshots of a web application interface titled "Search Result". Both screenshots show the same data but in different orders. Each screenshot includes a header "Index Name: index", a "Return to search menu" button, and a list of three search results. Each result is a file path followed by "Open PDF" and "Delete" buttons.

Top Screenshot (Order by Definition):

Index Name: index		
Return to search menu		
0 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_2\question\version_1\true_or_false_questi...	Open PDF	Delete
1 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_3\question\version_1\true_or_false_questi...	Open PDF	Delete
2 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_1\question\version_1\true_or_false_questi...	Open PDF	Delete

Bottom Screenshot (Order by Frequency):

Index Name: index		
Return to search menu		
0 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_3\question\version_1\true_or_false_questi...	Open PDF	Delete
1 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_1\question\version_1\true_or_false_questi...	Open PDF	Delete
2 - C:\Users\ASUS\Desktop\Semestre6\Projeto\perguntas_2\question_2\question\version_1\true_or_false_questi...	Open PDF	Delete

Figura 4.8: Teste de pesquisa ordenada

Na figura 4.8 pode se ver, primeiro uma lista de exercícios guardados pela sua ordem por definição, ou seja sem parâmetros de ordenação, e em segundo a lista de exercícios ordenados pelo numero de ocorrências de cada exercício. Os exercícios encontram se por esta ordem pois o exercício 3 foi adicionado 5 vezes, o exercício 1 foi adicionado 4 vezes e o exercício 2 foi adicionado 3 vezes.

4.7 Eliminação de um exercício

Neste teste de eliminação de exercícios o pretendido é, ao pressionar no botão *Delete* o exercício pretendido é apagado da base de dados e já não aparecerá em pesquisas futuras.

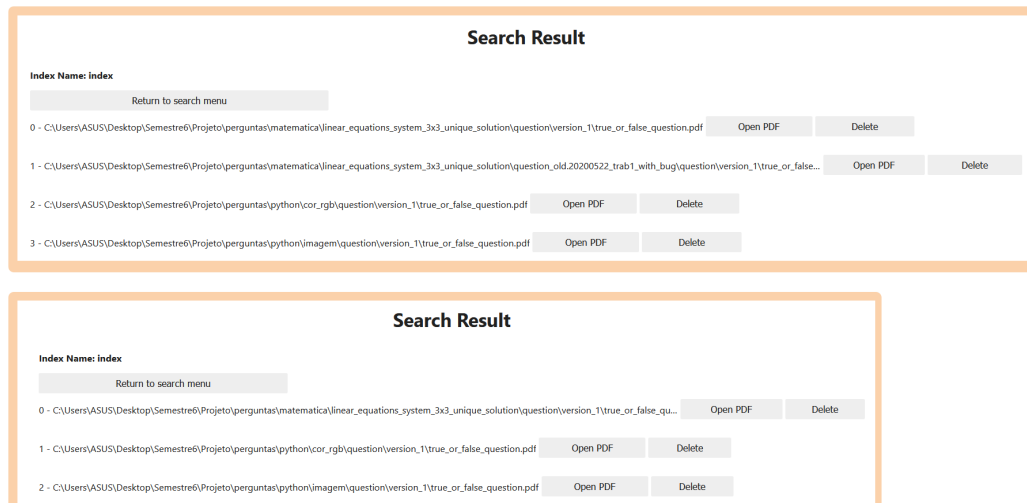


Figura 4.9: Teste de eliminação de um exercício

Na figura 4.9 é possível ver uma lista com quatro exercícios, ao pressionar no botão *Delete* o exercício pretendido é eliminado e, na seguinte figura, na próxima procura este não aparece.

Capítulo 5

Conclusões e Trabalho Futuro

Temos como objetivo a criação de um motor de pesquisa para um sistema de correção automática de exercícios na linguagem de programação Python. Para tal foi desenvolvida uma solução inteiramente na linguagem Python utilizando a biblioteca Whoosh para a criação do motor de pesquisa e a biblioteca PyMuPDF para o processamento de documentos no formato PDF. Esta solução contém vários mecanismos para o processamento de ficheiros, processamento de texto, armazenamento de informação relativa aos documentos e pesquisa sobre esta informação. A aplicação desenvolvida permite que um utilizador defina um ou mais espaços de pesquisa, administre os seus espaços de pesquisa (adicionar/remover exercícios) e realize pesquisas personalizadas nos seus espaços de pesquisa.

Com este projeto foi-nos apresentada uma oportunidade de melhorar o nosso conhecimento sobre Jupyter notebooks e a sua arquitectura. Melhoramos também o nosso conhecimento relativo a desenvolvimento de software assim como desenvolvimento de aplicações Web utilizando apenas a linguagem Python.

Apesar de termos cumprido todos os requisitos definidos no projeto e propostos pelo nosso orientador achamos que o nosso trabalho poderia ser melhorado em certos pontos. Esses pontos são processamento de texto, eficiência de pesquisa e interface de utilizador.

Relativamente ao processamento de texto, pensamos que este tema poderia ser melhor explorado de modo a encontrar melhores métodos que pudessem ser adicionados ao nosso projeto de modo a que o conteúdo guardado no índice esteja mais comprimido. Isto iria resultar numa menor ocupação

no disco do motor de pesquisa.

Ainda que o nosso motor de pesquisa seja bastante rápido a devolver o resultado de uma pesquisa ao utilizador, acreditamos que este processo poderia ser acelerado se estudássemos e pesquisássemos novas alternativas, como por exemplo, o desenvolvimento de um sistema que já tivesse pesquisas previamente guardadas associadas a um determinado conjunto de palavras chaves.

Por último a interface do utilizador poderia estar mais refinada de modo a ter um aspeto mais apelativo de modo a agradar a uma audiência de utilizadores mais abrangente.

Bibliografia

[Chaput, 2017] Chaput, M. (2017). Whoosh 2.7.4 documentation. <https://whoosh.readthedocs.io/en/latest/index.html>.

[Garcia, 2016] Garcia, E. (2016). A tutorial on the bm25f model.

[Jupyter, 2019] Jupyter (2019). ipywidgets 7.5.1 documentation. <https://voila.readthedocs.io/en/stable/>.

[QuantStack, 2019] QuantStack (2019). And voilà! <https://blog.jupyter.org/and-voil%C3%A0-f6a2c08a4a93>.

[Voila, 2020] Voila (2020). voila 0.1.21 documentation. <https://voila.readthedocs.io/en/stable/>.