Projeto Final em Engenharia Informática

Ferramenta de Pesquisa e Análise de Logs JSON e XML LOGrasper

FERRAMENTA DE PESQUISA E ANÁLISE DE MENSAGENS GERADAS POR COMUNICAÇÕES DE SISTEMAS DISTRIBUÍDOS

Jorge Miguel Cabrita Matias – Aluno 1901807

Orientador

RICARDO JOSÉ VIEIRA BAPTISTA

2023-06-28

Índice

Lista de Figuras

Lista de Tabelas

Glossário de Siglas

API,	Application Programming Interface
C#,	C – Sharp Programming Language
IDE,	Integrated Development Environment
JSON,	Java Script Object Notation
MB,	Megabyte
MVVM,	Model-View-ViewModel
REST,	Representational State Transfer
SOAP,	Simple Object Access Protocol
TPL,	Task Parallel Library
UC,	Unidade Curricular
UI,	User Interface
UID,	Unique ID
VS,	Versus
WPF,	Windows Presentation Foundation
XAML,	Extensible Application Markup Language
XML,	Extensible Markup Language

Introdução

Este projeto insere-se no universo da análise de ficheiros de LOG. A necessidade de pesquisar e analisar Logs tornou-se importante para a compreensão de comportamentos de sistemas distribuídos. É com foco nesta necessidade que surge a ideia de desenvolver uma aplicação que facilite o trabalho do analista de forma simples, rápida e eficaz.

Este relatório irá incidir na forma como se pretende desenvolver uma *ferramenta "stand alone"* que garante segurança dos dados e preserve a arquitetura dos sistemas em análise, assim como permite a rápida localização, baseada em palavras-chave inseridas, e apresentação de linhas de LOG.

Organização do Relatório

Este relatório apresenta as primeiras fases do projeto procurando aprofundar a temática proposta, com uma análise e investigação dos temas de fundo, que irão permitir a arquitetura e desenvolvimento deste projeto.

Temos no primeiro capítulo a descrição dos objetivos, resultados esperados e cronograma atualizado.

No segundo capítulo, a investigação e análise de mercado e requisitos.

No terceiro capítulo, a arquitetura.

No quarto capítulo, as conclusões e trabalho futuro e por último, há os anexos do projeto que contextualizam com maior detalhe algumas partes do desenvolvimento do projeto.

Capítulo 1

1. Descrição e objetivos do trabalho

O trabalho proposto é o desenvolvimento de uma aplicação desktop Windows para suporte na análise de Logs de mensagens geradas por processos de comunicação de sistemas distribuídos (web services, REST, SOAP) no formato JSON e XML.

1.1. MOTIVAÇÃO PARA A PROPOSTA DO TRABALHO

Recentemente, ingressei numa nova aventura profissional na área de analista de suporte. A empresa para a qual trabalho, a Omnibees S.A., intitula-se de Channel Manager e tem como foco a comunicação entre Hotéis, Operadoras turísticas e os seus canais de vendas. Neste âmbito, pensei o quanto a implementação desta ferramenta irá ter um impacto significativo no trabalho que executo na empresa. Após dar conhecimento do meu projeto à empresa e ter efetuado o pedido para poder utilizar dados reais (Logs), a resposta foi positiva e houve interesse na construção da ferramenta. No entanto existem algumas restrições ao nível da informação divulgada, assim, alguns dos Logs utilizados nos testes tiveram o seu conteúdo modificado. Foram também facultadas algumas das versões da aplicação durante o seu desenvolvimento a colaboradores da empresa, estes tiveram a oportunidade de utilizar e testar, facultado feedback e sugestões preciosas que motivaram alterações e mudanças no paradigma da aplicação ao nível do principal objetivo.

Como analista, o meu quotidiano incide maioritariamente na análise de Logs, onde estes, contêm mensagens trocadas entre as diferentes API de todo o Universo Omnibees.

Neste âmbito, eu sinto necessidade de ter uma ferramenta que permita pesquisar por keywords, em pastas com imensos ficheiros de Log, alguns com muitas centenas de MB, logo muito grandes, quer ao nível da quantidade de informação, quer ao nível do armazenamento.

As ferramentas já existentes de pesquisa de ficheiros são genéricas, por exemplo, a Fileseek ou a AstroGrep, que apesar de contarem com inúmeras opções de pesquisa, não permitem o relacionamento entre keywords no universo de pesquisa, apensas permitem pesquisa em ficheiros e seus conteúdos, nem têm a possibilidade de devolver apenas linhas (log) que contenham duas ou mais keywords, ou se têm não são de fácil utilização. Por exemplo o Fileseek, permite efetuar procuras com expressões regulares que, com a expressão certa, permite a procura de várias keywords, no entanto nem todos os utilizadores têm conhecimentos para o fazer, e mesmo os que têm não estão dispostos a perder tanto tempo para gerar a expressão. Já com o AstroGrep, a pesquisa de várias keywords também pode ser efetuada com expressões regulares, e efetuada com

pesquisas em resultados previamente obtidos, dependendo assim de várias pesquisas, aumentando a dificuldade e perda de tempo. Nenhuma delas permite a procura por várias keywords de forma simples e rápida.

Devido á minha necessidade como colaborador/utilizador julgo ter identificado uma oportunidade importante para o desenvolvimento de uma solução de projeto de engenharia.

Como exemplo concreto, temos a pesquisa por uma mensagem com a origem de uma reserva, que tem uma determinada tarifa e quarto, que tenha sido criado num determinado dia, etc., localizada numa pasta com dezenas ou até centenas de ficheiros, mas, que também, permita relacionar essa mensagem com outras no mesmo universo, desde request Ids a trace Ids, etc. através de elementos da mensagem, quer sejam pares key value num JSON ou tags XML, através de templates criados na ferramenta.

A ferramenta irá chamar-se LOGrasper e terá o seguinte tema:

Figura 1 - Logótipo da Aplicação

1.2. OBJETIVOS

O objetivo principal do projeto final é o desenvolvimento de uma ferramenta (aplicação Windows) de pesquisa em Logs. A razão para o desenvolvimento de uma aplicação "*stand-alone*", prende-se essencialmente na segurança dos dados analisados, ou seja, sem qualquer acesso de rede os dados particulares quer sejam de clientes, ou de propriedade empresarial, estarão seguros e serão da responsabilidade do utilizador. A ferramenta será 100% segura a esse nível. A opção de ser uma aplicação Windows, é devido ao facto de a empresa utilizar tecnologias Microsoft.

A possibilidade de se poderem criar *templates* de linhas de *Logs* conhecidas, que possam ser relacionados entre si, era também uma das funcionalidades que considerava importante, pois neste universo, raramente as comunicações entre as API para um objetivo específico, utilizam apenas um par *request-response*, principalmente na área de negócio onde laboro, há sempre operações que têm de ser efetuadas quer a nível interno quer a nível externo, a possibilidade de se obter todo um fluxo de comunicação é uma ajuda enorme para quem analisa. No entanto, no decorrer do desenvolvimento, e após recolher opiniões junto dos utilizadores de testes, estes não a consideraram como uma funcionalidade chave, sendo que a performance, neste caso a velocidade de pesquisa foi considerada a funcionalidade chave. Devido a falta de tempo, optei por focar todos os recursos nesta parte, ficando a funcionalidade de templates adiada para ser implementada no futuro.

Assim, um utilizador ao pesquisar por uma ou mais *keywords* num conjunto de ficheiros, obteria um output apenas com as linhas onde essas *keywords* constassem, podendo posteriormente selecionar os elementos (baseado no template) que deseja que sejam relacionados. Após esta seleção, a Ferramenta procederia a nova pesquisa, no mesmo universo de ficheiros, que iria devolver todas as relações pedidas, e assim, mostrar todo um fluxo de comunicações relacionadas.

Ao nível da linguagem de programação para o desenvolvimento, pretendo utilizar C#, juntamente com o XAML, criando um projeto no Visual Studio 2019, com o template .NET Core WPF Application.

1.3. RESULTADOS ESPERADOS

A apresentação de uma aplicação funcional, que permita, pesquisar em ficheiros de *Logs* extensos, por palavras-chave (*keywords*), as possa facilmente relacionar, e assim obter informações importantes, que serão cruciais na identificação de diversos problemas, dependendo do contexto em que se inserem. Como exemplo, temos a resolução de tickets de suporte ou a identificação de melhorias nas API, entre muitas outras.

A *user interface (UI)*, foi trabalhada de forma a garantir que a utilização da ferramenta, seja simples e intuitiva. O suficiente para, sem grandes conhecimentos técnicos, qualquer utilizador a possa utilizar e compreender. Neste âmbito, na fase final do projeto dediquei bastante tempo a tornar a *UI* robusta de forma que independentemente da ação do utilizador, a aplicação nunca falhe.

Os resultados esperados foram validados por utilizadores reais, que trabalham no meio ao qual a ferramenta se foca, e exteriores ao desenvolvimento, ou seja, analistas de suporte. Numa primeira fase de testes, a versão alfa foi facultada a 6 utilizadores experientes, com o objetivo de identificarem e retornarem erros e possíveis melhorias na ferramenta. Estes utilizadores já utilizam as ferramentas acima descritas podendo assim ser obtido um feedback de comparação entre as ferramentas existentes e a que está a ser desenvolvida. Esta 1ª fase de testes incidiu na funcionalidade de pesquisa de *keywords* e apresentação de resultados. Foi também posteriormente, pedido a estes utilizadores que respondessem a um pequeno questionário, os seus resultados serão mostrados neste relatório.

Nos Objetivos referi que a performance foi o foco chave desta aplicação, e que devido ao curto tempo restante, a opção foi focar na performance, deixando a fase Beta (funcionalidade de templates) para ser desenvolvida futuramente. Para que o utilizador possa ter algum tipo de "memória" sobre as keywords utilizadas, ao gravar o *"output"* a aplicação ao criar o ficheiro, refere as keywords utilizadas, assim como o tempo que demorou a pesquisa.

1.4. CRONOGRAMA

Como foi já referido o Projeto sofreu um atraso (não previsto) de cerca de duas semanas, este atraso deveu-se a circunstâncias académicas (Unidade Curricular paralela que necessitou de mais tempo que o previsto) e profissionais, no entanto, mantenho a confiança que conseguirei cumprir, no final, com os objetivos propostos. No seguinte cronograma estão as atualizações das novas datas de previsão de implementação. Sendo que os retângulos a representam o cronograma do Relatório intermédio, e os vermelhos as tarefas não realizadas. Cronograma completo em[1].

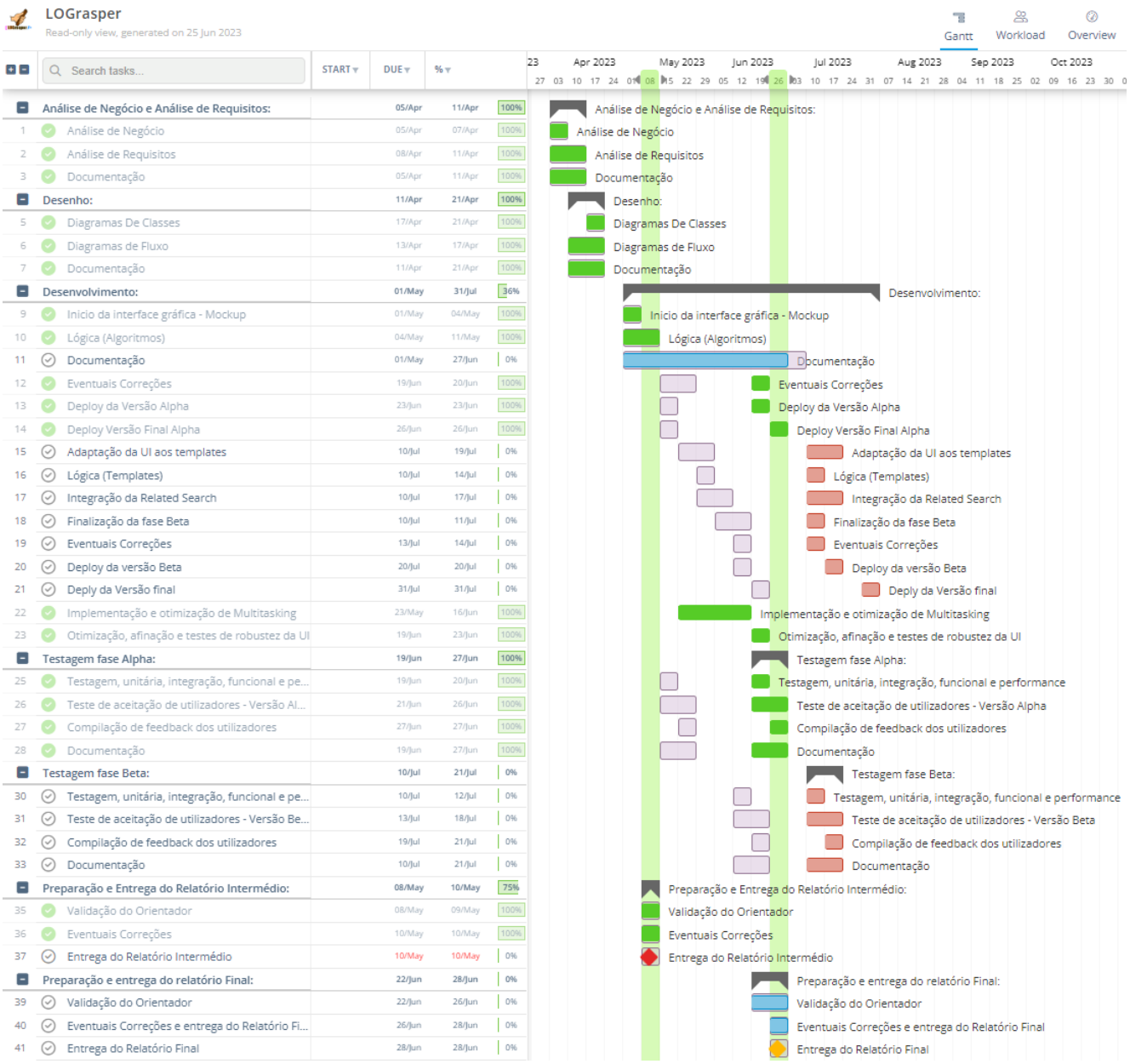


Figura 2 - Mapa de Gantt Final

[1] https://app.instagantt.com/shared/64983050371a3e6bf3266ec6

Capítulo 2

2. Fundamentos/Estado-da-arte

Este capítulo irá incidir sobre a análise de negócio que ajudou a gerar análise de requisitos, a arquitetura da aplicação

2.1 ALGORITMOS DE PESQUISA

Na proposta de projeto referi que, ao nível de algoritmos de pesquisa, após alguma investigação deparei-me com vários algoritmos, o Aho-Corasick (Trivedi, 2020) (Moshiri, Advanced Data Structures: Aho-Corasick Automaton, 2020) (Moshiri, Advanced Data Structures: Aho-Corasick Dictionary Links, 2020) (Vashchegin, s.d.), que permite pesquisar e localizar um conjunto finito de strings (dicionário de keywords) simultaneamente e apenas numa leitura do input. Sendo este um dos principais candidatos a ser implementado. O algoritmo trie-based é similar ao Aho-Corasick, mas pode não ser tão eficiente, serão ambos verificados. No entanto existem outros algoritmos de pesquisa em ficheiros grandes, mas procuram apenas por um único padrão, o que pode ser vantajoso se o utilizador apenas quiser procurar por uma keyword. Assim procederei também ao estudo do algoritmo Boyer-Moore (comparação direita-esquerda saltando caracteres incompatíveis com o padrão de procura), e ao algoritmo Rabin-Karp (Comparação por valores hash).

Após vários testes de implementação básica, concluí que para cumprir os objetivos propostos, o algoritmo Aho-Corasick (Referido como AC) é o mais indicado para pesquisas com várias keywords. Neste momento e tendo em conta o atraso verificado no projeto, irei, no âmbito deste relatório, referir-me apenas ao algoritmo AC, no entanto, com plena consciência que se a lista de keywords for constituída apenas por uma keyword, o algoritmo AC não será o mais eficiente. Esta será uma característica a melhorar no desenvolvimento futuro da aplicação, pois penso melhorá-la a todos os níveis.

2.1.1. Algoritmo Aho-Corasick (AC)

Utilizei uma aproximação a este algoritmo que passo a explicar. Na sua essência, o algoritmo faz uso de uma máquina de estados ou autómato para pesquisar padrões em strings.

2.1.1.1 Autómato AC

A máquina de estados ou autómato, funciona como uma árvore binária que é criada em função das palavras-chave (keywords) fornecidas. A estrutura de cada nó da árvore, no código a classe *TrieNode,* é constituído por um parent do tipo *TrieNode* por um *Children* do tipo Dictionary que irá conter uma única instância de um caracter, pelo caracter em si do tipo char, por um *Failure* do tipo *TrieNode*, e finalmente pelo *KeywordEnd* do tipo *FinalNode*. O tipo *FinalNode* conta com dois boolenos o *End* e o *AllKeywordsMatched* e uma string *Keyword* que contém a própria keyword.

A inicialização (construtor) é inicializado com o nó root sem filhos. A partir daqui são inicializadas as keywords inseridas pelo utilizador através do método ***AddKeyword()*** que iniciando na root, para cada caracter da keyword, vai criando nós *Children* (desde que estes não contenham o caracter chave do dicionario), até que, ao atingir o último caracter, coloca o último nó como *End* e atribui a string da keyword. Este passo é repetido para cada keyword, ficando uma arvore com todas as keywords inseridas.

Este autómato será construído no *Model SearchEngine* que será explicado em maior detalhe no ponto 3.4.4.3.1 . Neste, primeiramente serão adicionadas as keywords e depois será construído o Autómato. Na construção do autómato o método ***BuildAutomaton(),*** tem como objetivo encontrar sequências de caracteres idênticas na arvore e apontar os Failure de forma que quando falhe uma sequência se possa aproveitar o caracter do nó atual para inicializar outra sequência (Keyword), por exemplo se uma *keyword* for uma *substring* de outra keyword. Esta fase do algoritmo está muito bem explicada em (Moshiri, Advanced Data Structures: Aho-Corasick Automaton, 2020) [2] e (Moshiri, Advanced Data Structures: Aho-Corasick Dictionary Links, 2020)[3].

2.1.1.2 AC Search

Para o método de pesquisa, optei por definir como uma Lista de *Tuplos* com o objetivo de guardar os resultados da pesquisa, ou seja, obter as keywords e a posição em que estas se encontram, inicialmente a posição seria para rapidamente identificar a keyword na linha de output, mas com a mudança de paradigma da UI acabei por não utilizar, no entanto está implementada no algoritmo para futuramente ser utilizada. O método *Search()* em si, obtém, como já foi dito os resultados da procura, ao percorrer todos os carateres da string enviada como argumento (no caso cada linha de um ficheiro), e, para cada caracter carregado, se o nó atual não for o nó root e se os seus filhos não conterem a chave de dicionario igual ao caracter carregado, percorre a arvore

[2] https://www.youtube.com/watch?v=O7_w001f58c
[3] https://www.youtube.com/watch?v=OFKxWFew_Lo

até encontrar um nó válido ou a root, seguidamente, se o caracter existir como filho do nó atual, move-se para este, verifica se é *KeywordEnd*, se sim, adiciona aos resultados encontrados, se os resultados encontrados já corresponderem a todas as keywords, foi adicionada uma condição que retorna imediatamente os resultados, ignorando o resto da linha, senão percorre todos os caracteres até ao final. Esta condição que foi adicionada, poderá ter um impacto grande na performance, pois no universo dos Logs existem ficheiros que contêm linhas com milhões de caracteres, ou seja, na melhor das hipóteses as keywords encontram-se no início das linhas e poupam-se muitos bytes de análise.

2.1.1.3 AC Conclusão

Ao estudar o algoritmo Aho-Corasick apercebi-me do potencial deste, pensei na forma como me poderia ser útil na forma como com uma passagem apenas consegue encontrar todas a keywords inseridas no seu Autómato. Adaptei o algoritmo de forma a servir o interesse da minha aplicação, mas ao mesmo tempo penso que tenho um algoritmo que pode ser reutilizado e útil em muitas situações futuras.

2.2. ANÁLISE DE NEGÓCIO E ANÁLISE DE REQUISITOS

2.2.1. Análise de negócio

Para a análise de negócio, o foco foi direcionado essencialmente pela experiência profissional adquirida. Isto é, na experiência adquirida no dia a dia laboral, com a utilização de diversas aplicações de pesquisa em ficheiros. A necessidade de efetuar pesquisas regularmente, em pastas com centenas de ficheiros com muita informação, impeliu-me a procurar as ferramentas que melhores resultados apresentassem, e com o melhor desempenho. Nesta procura, destacaram-se três aplicações. O Fileseek, o Astrogrep e o Agent Ransack ou FileLocater (estes últimos idênticos e da mesma empresa), todas elas têm os seus pros e contras, foi da experiência adquirida a utilizar todos eles que nasceu a ideia de desenvolver uma aplicação focada para a pesquisa de Logs.

Estas ferramentas (aplicações) serão referidas no decorrer desta análise como "existentes".

2.2.1.2 Comparativo de Mercado

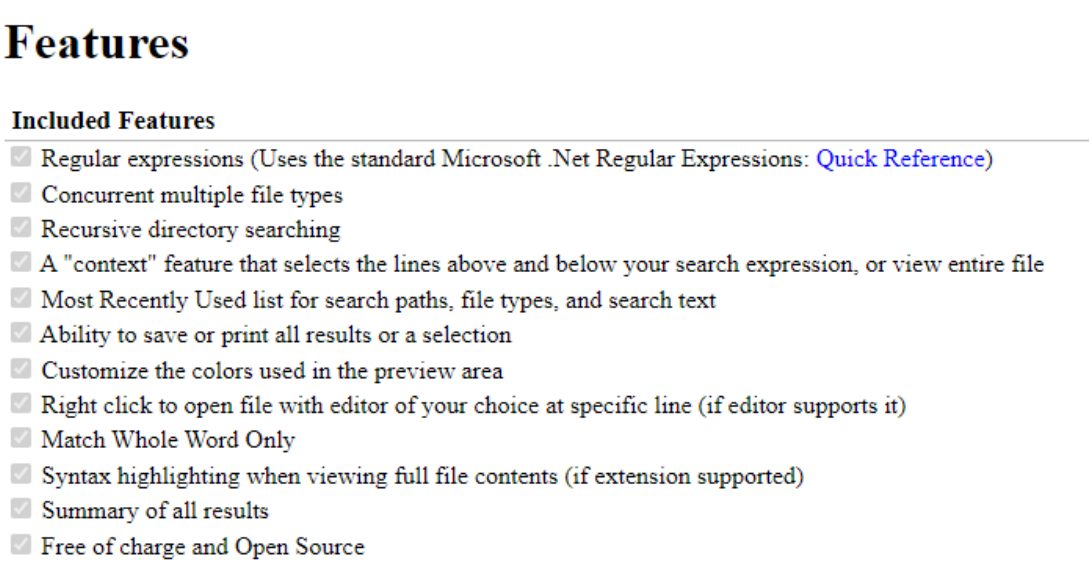
Assim, proponho-me a desenvolver uma nova ferramenta, que tenha em conta os aspetos referidos acima. No decorrer da minha ação profissional, procurei bastante por uma ferramenta deste género, pelo que não encontrei nenhuma que oferecesse estas funcionalidades. As Ferramentas mais utilizadas no meio que laboro têm as seguintes funcionalidades.

FileSeek (fileseek.ca, s.d.)

Feature	Free	Pro Standard
Search multiple paths at once	✓	✓
Exclude one or more paths from search results	✓	✓
Filter by filename (ex: '*.txt' or '*bill*.doc')	✓	✓
Filter filenames using regular expressions, advanced text queries, or patterns	✓	✓
Exclude by filename (ex: '*.zip' or 'bill*.zip')	✓	✓
Exclude filenames using regular expressions, advanced text queries, or patterns	✓	✓
Search sub-folders automatically	✓	✓
Search inside files using **simple text queries** (ex: 'dog')	✓	✓
Search inside files using **advanced text queries** (ex: '+dog +cat +horse')	✓	✓
Search inside files using **regular expressions**	✓	✓
Show a text preview for each search result	✓	✓
Ability to customize search columns	✓	✓
Full support for long filenames and folders (longer than 260 characters)	✓	✓
Open searches in multiple tabs	⊖	✓

Figura 3 - FileSeek - Features

AstroGrep (astrogrep.sourceforge.net, s.d.)



Features

Included Features

- Regular expressions (Uses the standard Microsoft .Net Regular Expressions: Quick Reference)
- Concurrent multiple file types
- Recursive directory searching
- A "context" feature that selects the lines above and below your search expression, or view entire file
- Most Recently Used list for search paths, file types, and search text
- Ability to save or print all results or a selection
- Customize the colors used in the preview area
- Right click to open file with editor of your choice at specific line (if editor supports it)
- Match Whole Word Only
- Syntax highlighting when viewing full file contents (if extension supported)
- Summary of all results
- Free of charge and Open Source

Figura 4 - AstroGrep – Features

Agent Ransack (Mythicsoft, s.d.)

Recentemente também se começou a utilizar o Agent Ransack, apresento um resumo das features, para mais detalhes verificar na página da internet do programa[4] .

Feature Comparison

	Lite (free)	Professional
+ SEARCH ENGINE	●●○○	●●●●
+ INDEXING	○○○○	●●●●
+ OFFICE/PDF SUPPORT	●○○○	●●●●
+ COMPRESSED ARCHIVES	○○○○	●●●●
+ DATA DISCOVERY	●●○○	●●●●
+ REPORTS &EXPORTING	●○○○	●●●●
+ INTERNATIONALIZATION	●○○○	●●●●
+ PROGRAMMABILITY	●○○○	●●●●
+ ADVANCED FEATURES	○○○○	●●●●

Figura 5 - Agent Ransack - Features

[4] https://www.mythicsoft.com/agentransack/information/#searchfeatures

2.2.1.3 Comparativo de Mercado VS proposta de projeto

As ferramentas acima referidas apresentam bastantes opções de pesquisa, sendo que passo a detalhar as opções mais relevantes para o objetivo em questão (procura em ficheiros de Logs), e a forma como projeto as opções que a nova ferramenta irá ter, ou seja, a análise de alguns requisitos por comparação, sendo:

E – existentes e **L** - LOGrasper :

- **E** - A procura recursiva em diretorias, em múltiplos caminhos, e a possibilidade de excluir caminhos específicos (As 3 referidas suportam).
  L - Procura recursiva em diretorias.
- **E** - Critério de pesquisa por texto, sejam palavras ou expressões regulares num formulário, os resultados da pesquisa têm como foco todas as linhas dos ficheiros onde se encontram um dos critérios de pesquisa (as 3 suportam).
  L - Procura simplificada, através da inserção de palavras-chave (keywords) ou da utilização de elementos de templates como critérios, no resultado da pesquisa serão apenas as linhas dos ficheiros onde todos os critérios se verifiquem.
- **E** - Suporte para múltiplos tipos de ficheiros (as 3 suportam).
  L - Suporte apenas garantido para ficheiros .txt, sendo que fica identificada uma possibilidade de melhoria, o suporte para múltiplos tipos de ficheiros.
- **E** - Histórico de pesquisa (as 3 suportam).
  L - Identificada possibilidade de melhoria, inclusão de histórico de pesquisa (na sessão).
- **E** - Descompressão dos ficheiros de contexto antes da pesquisa (Agent Ransack)
  L - Identificada possibilidade de melhoria, descompressão de ficheiros que estejam comprimidos nas pastas de procura.
- **E** - Possibilidade de abrir um ficheiro onde seja identificada uma linha como resultado num programa externo (Fileseek e Astrogrep), ou salvar o resultado da pesquisa num ficheiro de texto (as 3 suportam).
  L – A Ferramenta a desenvolver deverá ter ambas as funcionalidades, no que diz respeito à aplicação exterior, a aplicação utilizada deve ser a pré-definida atualmente no sistema operativo. Sendo aqui também identificada uma oportunidade de melhoria, a seleção da aplicação exterior na própria ferramenta, com a possibilidade de o editor de destino dar enfase á linha clicada.
- **E** – Pesquisa Multi-Thread (Agent Ransack)
  L – As pesquisas deverão suportar threads múltiplos.

2.2.2 Análise de Requisitos

Na comparação que anteriormente vimos, foram então identificados vários requisitos a desenvolver para a Ferramenta, a lista seguinte engloba alguns deles e outros também identificados. A análise de requisitos tal como o projeto, apresenta-se em duas fases a Alfa e a Beta, sendo que nesta fase do projeto estão referenciados os requisitos da fase Alfa. Como já referido anteriormente, devido a falta de tempo o foco foi na fase Alfa e na sua Otimização ao nível da performance. O feedback gerado pelos utilizadores reais, na fase de testes com a *release* da primeira versão, provocou uma ligeira alteração em alguns dos requisitos, que tornaram a aplicação mais rápida na pesquisa e mais robusta.

FASE ALFA

- **A procura recursiva em diretorias:** A Ferramenta deverá ter um campo de formulário, onde o utilizador define a pasta raiz a pesquisar. Este campo será preenchido através de um botão "*Browse*", que após o utilizador clicar irá aparecer uma janela de explorador de ficheiros, onde se poderá selecionar a pasta raiz e de seguida clicar no "*OK*", ou no "*Cancel*" para cancelar a seleção. Quando for iniciada a pesquisa, a ferramenta deverá pesquisar em todos os ficheiros da pasta raiz e subpastas desta.

- **Procura simplificada, através da inserção de palavras-chave (keywords):** A Ferramenta deverá ter um campo de formulário, onde o utilizador irá inserir as keywords, após inserir uma keywords, o utilizador clicará no botão "*Add*" e a keyword correspondente irá ser adicionada numa caixa de texto visível no ecrã. Caso o utilizador queira alterar ou apagar a keywords, deverá clicar na mesma, o botão "*Edit*" para que possa modificar a keyword e o botão "*Delete*" para que a possa apagar, ficarão disponíveis para o utilizador.
  Após estas operações e quando todas as keywords estejam adicionadas, o utilizador poderá clicar no botão "*Search*" e dar início á pesquisa.

- **Procura Multi-Thread:** A Ferramenta deverá possibilitar a utilização de vários processos na mesma pesquisa. Esta opção ainda requer estudo sobre a forma de implementação, se será uma opção fixa ou se o utilizador poderá ou não definir opções de *multi-threading*. Neste requisito inicialmente a intenção seria usar *threads*, mas ao estudar a melhor forma de realizar este requisito, por questões que serão explicadas no capítulo 3, optei por utilizar *task*s.

- **Suporte apenas garantido para ficheiros .txt:** O tipo de ficheiro que garantidamente é suportado é o ficheiro de texto, sendo que no decorrer do desenvolvimento deverá ser estudada a forma como se irá comportar a aplicação no caso de uma extensão diferente, embora o ficheiro contenha texto.

- **Apresentação dos resultados da pesquisa:** Os resultados da pesquisa, serão apresentados numa caixa de texto, onde cada linha de ficheiro, onde estejam contemplados os critérios de pesquisa, irá conter a seguinte informação:
  - "Caminho e nome do Ficheiro" (o nome do ficheiro deverá ser destacado)
  - Para cada linha encontrada, a disposição será a seguinte.
    - "Número de Linha": "Linha completa" (keywords deverão ser destacadas opcionalmente).

      Para este requisito, inicialmente estava previsto que as keywords fossem destacadas na *UI.* Mas, na fase de testes reais, um dos utilizadores verificou que quando existiam linhas muito grandes para apresentar, a aplicação bloqueava. De forma a contornar esta situação, optei por limitar o tamanho máximo das linhas a mostrar, a 9600 caracteres, sendo mostrada uma mensagem por baixo da linha a avisar que a linha foi cortada por motivos de performance. Nos testes surgiram linhas de output com mais de 4 Milhões de caracteres. O utilizador ao gravar o ficheiro de output obtém as linhas completas.

- **Visualização em aplicações exteriores:** Após a obtenção dos resultados, o utilizador poderá dar "*duplo clique*" na linha que pretende, após esta ação, o editor pré-definido no Sistema Operativo irá abrir o ficheiro do contexto.

- **Gravar resultados da pesquisa:** Após a obtenção dos resultados, o utilizador poderá clicar em "*Save Results*", aparecerá uma janela de explorador de ficheiros. Aqui o utilizador poderá escolher a pasta de destino, introduzir o nome pretendido para o ficheiro e seguidamente clicar em "*Save*". Será criado um ficheiro de texto com a informação da caixa de resultados. Neste requisito, foi adicionada a funcionalidade de abrir automaticamente o ficheiro criado. Devido à já referida alteração na apresentação do output na *UI,* o ficheiro criado não será uma cópia exata do apresentado, por sua vez irá conter as linhas completas encontradas na pesquisa, assim como as palavras-chave utilizadas na pesquisa e o tempo que demorou a pesquisa.

Capítulo 3

3. Protótipo

Este capítulo irá abordar a arquitetura e implementação do projeto, e serão focadas as opções de arquitetura, opções técnicas, opções de UI, e apresentados os vários diagramas, tais como de casos de uso, diagrama de sequências e o digrama de classes.

3.1. ARQUITETURA DA APLICAÇÃO

Neste contexto, uma vez que a decisão foi desenvolver a aplicação como uma WPF (*Windows Presentation Foundation*), com XAML (adegeo & v-trisshores, 2023), e C#, investiguei qual a melhor opção de arquitetura para o desenvolvimento. Após vários artigos consultados a opção recaiu no padrão MVVM (Model-View-ViewModel) (Syromiatnikov & Weyns, s.d.).

Na Componente UI, a construção da *"interface"* será efetuada por partes (componentes) que compõem a View principal. A View e seus componentes, irão atuar nas ViewModels quando um evento for despoletado, assim como as ViewModels irão atuar nos componentes da View quando necessário alterar alguns dos seus estados. A ViewModel por sua vez sempre que for necessária lógica de contexto, "pedirá" a funcionalidade ao Model que gere todo o contexto lógico da aplicação.

Os diagramas apresentados seguidamente foram contruídos com a ajuda da aplicação VisualParadigm[5], utilizado o manual aconselhado na UC Modelação de sistemas de informação (Silva & Videira, 2001) e sua teoria.

[5] https://www.visual-paradigm.com/
https://www.visual-paradigm.com/download/community.jsp

3.1.1. Diagrama de Classes

Como já referido apenas foi desenvolvida a fase Alfa. O diagrama de classes contém os componentes da View, que embora não sejam propriamente classes, implementam todos os controlos e atualizações da View. O SearchViewViewModel, contem a agregação de todos os ViewModels que atuam nos componentes da View, associadas a cada ViewModel, estão as classes de comandos, que gerem os diferentes eventos, maioritariamente eventos gerados por cliques em botões. A classe Model é uma Abstração, pois existem diferentes classes para cada Model na realidade, sendo cada atributo representado um Model diferente, para uma questão de representação escolhi mostrar como uma só classe.

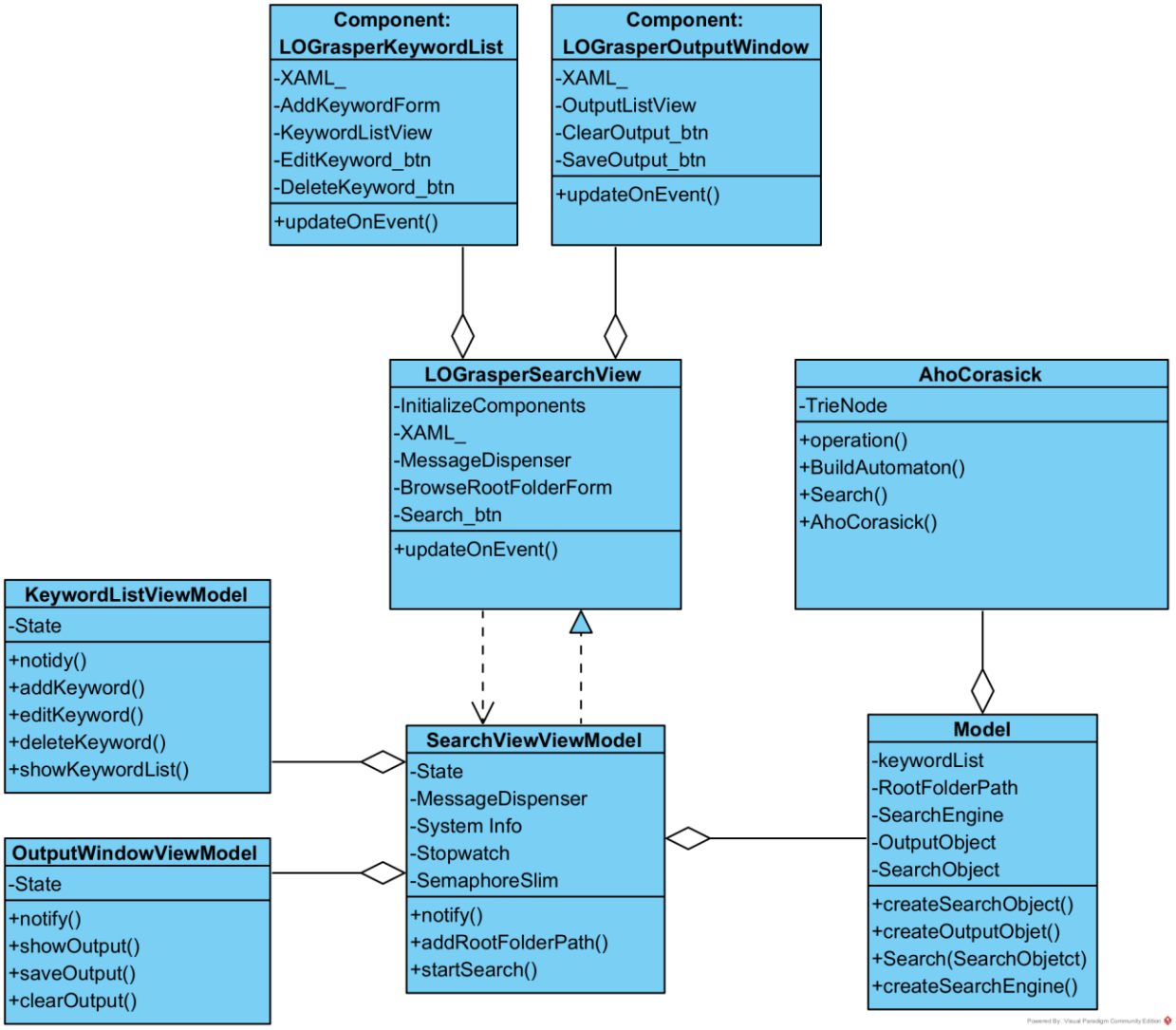


Figura 6 - Diagrama de Classes Fase Alfa

3.1.2. Diagrama de Sequência preliminar completo da fase Alfa

Seguidamente apresento o diagrama de sequência preliminar completo da fase alfa, onde se podem identificar os diversos componentes que irão compor a aplicação, assim como o fluxo de dados. De notar que as classes representadas são uma abstração, poderão existir várias Views, ViewModels, Models e Search Algorithm.

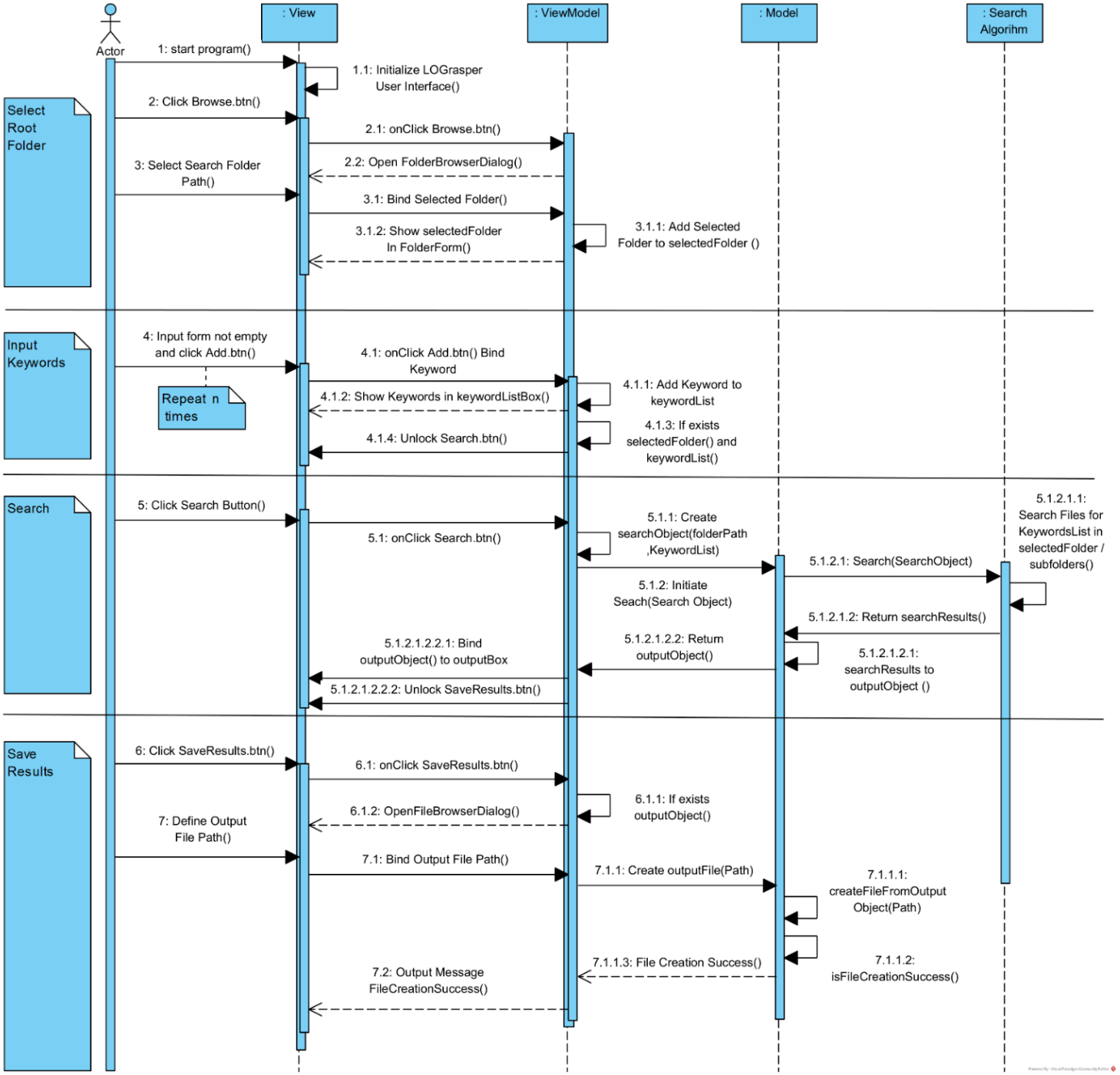


Figura 7 - Diagrama de Sequência fase Alfa

3.2 CASOS DE USO

Apresentam-se neste tópico, casos de uso da aplicação, com a descrição dos passos necessários para a obtenção de resultados. Os subtópicos seguintes são referentes às duas fases do projeto, a Alfa e a Beta.

3.2.1 Caso de uso

Exemplo de fluxo (Sem templates):

1. Utilizador efetua pesquisa:
    - Adiciona como keyword um número inteiro que corresponde por exemplo, a um dado UID;
    - Clica em add new keyword, insere uma dada string;
    - Repete o processo até que todas as keywords que pretende estejam adicionadas;
    - Seleciona a pasta raiz para a pesquisa;
    - Clica em search;
    - Obtém resultados.

2. Utilizador verifica resultados:
    - Analisa as linhas devolvidas nos resultados;
    - Selecionada a linha pretendida;
    - Tem a possibilidade de ver a linha num editor de texto ou IDE que pretenda;
    - Na Ferramenta, tem a possibilidade de gerar um template em função da linha selecionada, seja um Objeto JSON ou um documento XML digamos uma mensagem SOAP; (Ainda Não implementado)
    - Ao criar o Template, pode selecioná-lo, a Ferramenta nesta fase irá permitir que o utilizador após a escolha do template, selecione elementos desse template baseados na linha de output escolhida, ou seja um par key value ou uma determinada tag XML com o seu conteúdo; (Ainda Não Implementado)
    - Nesta fase o utilizador pode clicar em Relate Search, e a Ferramenta irá devolver todas as linhas que existirem no universo de ficheiros, que contêm os elementos selecionados do template (relação). (Ainda Não Implementado)
    - Os templates poderão ser gravados em ficheiros JSON por forma a poderem ser importados e exportados na Ferramenta, logo serão reutilizáveis e partilháveis. Num segundo momento será ainda verificada a possibilidade de existir um repositório de partilha, mas este terá sempre de ser dentro do contexto da empresa de forma que não haja fugas de informação tecnológica (estrutura das mensagens). (Ainda Não Implementado)

3.2.2 Diagrama de casos de utilização Completo

O seguinte diagrama representa os possíveis casos de utilização da aplicação, inclui a fase Alfa e Beta. Neste incluem-se casos como por exemplo a pesquisa através de um template gerado a partir de um output.

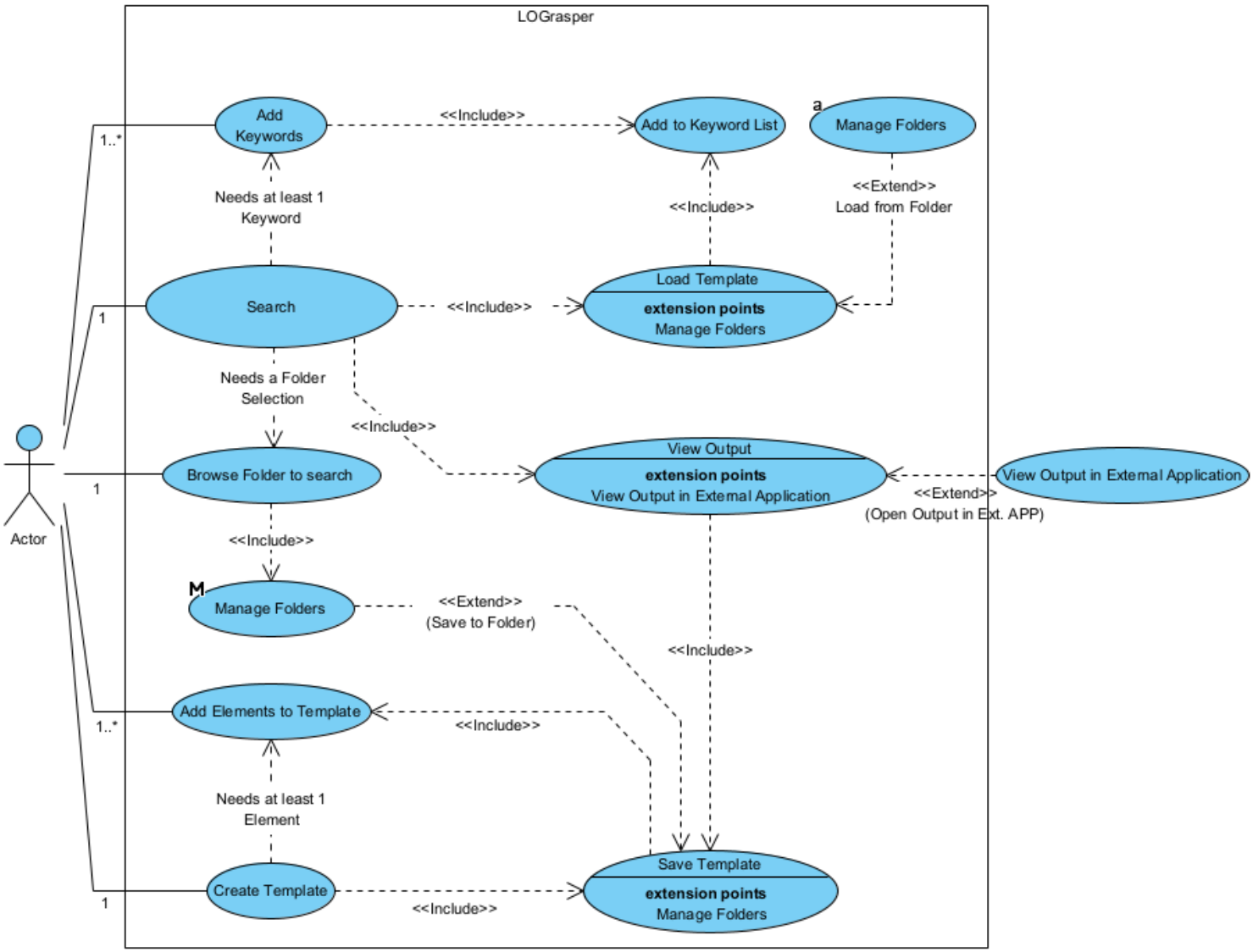


Figura 8 - Diagrama de Caso de Utilização Genérico Completo

3.3. MOCKUP – USER INTERFACE

A *"user interface" foi* desenvolvida para apresentar as *"Views"(XAML)* necessárias à fase Alfa, e apesar de ainda poderem existir algumas alterações no UI (estilos), a base dos componentes está estabelecida e pronta receber o desenvolvimento do código *C#*.

A arquitetura da UI é constituído essencialmente por uma *"View"* principal, "`LOGrasper.Views.LOGrasperSearchView`" e dois outros componentes que compõem a *"View"* Principal:

- `LOGrasper.Components.LOGrasperKeywordList`
- `LOGrasper.Components.LOGrasperOutputWindow`

Seguidamente apresenta-se uma imagem da *"View"* Principal.

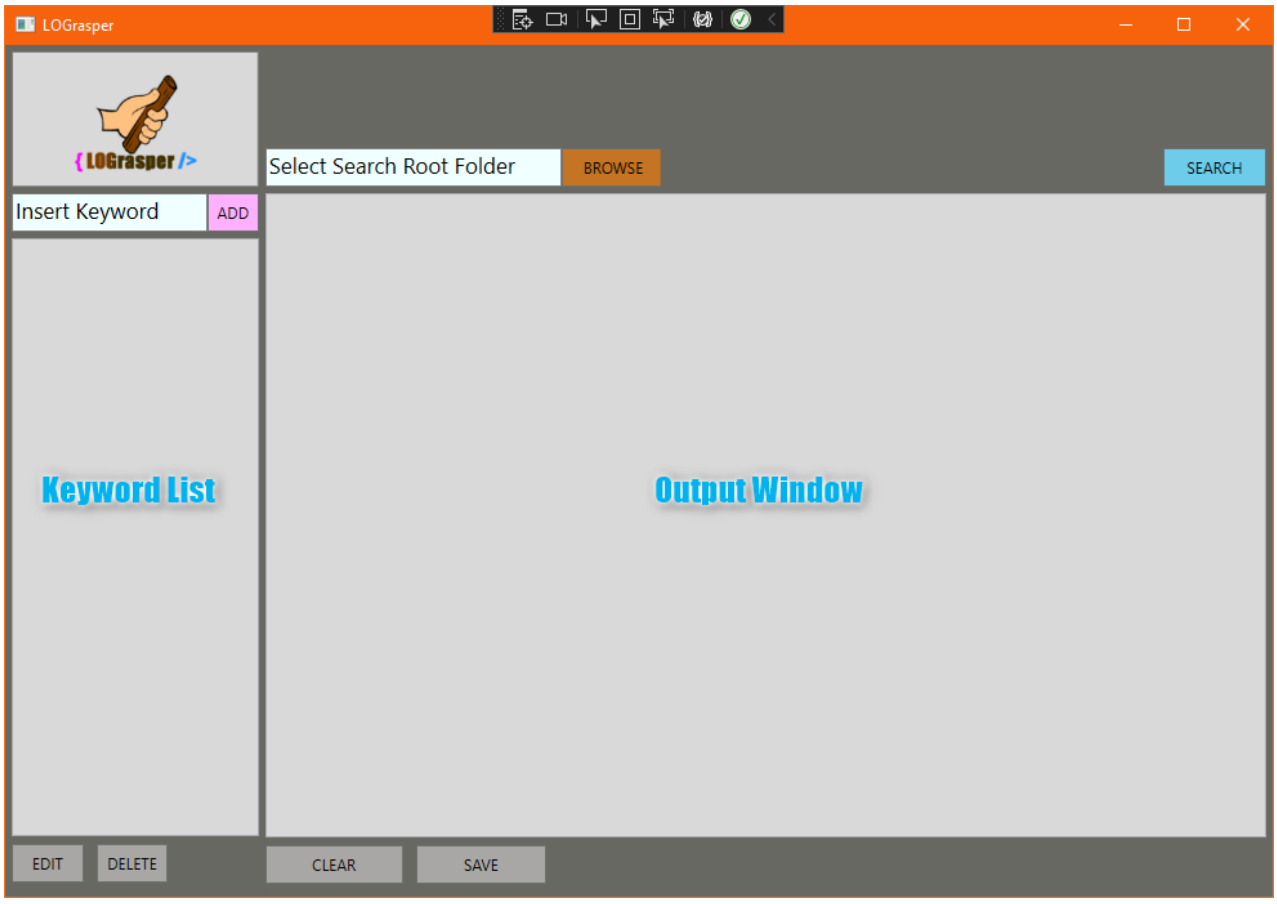


Figura 9 - Mockup - User Interface

A opção por utilizar uma View à parte da *"MainWindow"* prevê a criação de outras *"Views"*, ou seja, o UI é escalável e totalmente independente. Basta colocar lógica adequada para "carregar" outra *"View"* com componentes diferentes, permitindo até uma sensação de *"Navegação"*.

3.4 EVOLUÇÃO DO PROJETO

Como referido anteriormente, o projeto sofreu um atraso de cerca de duas semanas até ao relatório Intermédio, ainda assim, foi possível desenvolver uma boa aproximação ao pretendido.

3.4.1 Foco no Algoritmo Aho-Corasick

Na lógica do algoritmo de pesquisa foi desenvolvida inicialmente uma pequena aplicação de consola como teste. Esta, permitiu verificar a funcionalidade e fiabilidade do algoritmo AC num contexto real (pesquisa em *Logs*). Ao efetuar estes testes, verifiquei que existem diversos fatores a ter em conta no restante projeto, tais como:

- Lógica de leitura e escrita na drive de armazenamento, na consola de teste, foi implementada uma lógica de scan por linha, gerando a noção que em termos de performance não será a melhor opção devido ao número elevado de linhas a analisar. Retira-se a conclusão de que uma aproximação de scan por blocos de texto será mais eficaz, facilitando até uma perspetiva de *multi-threading*. Assim este será um aspeto a ter em conta no restante desenvolvimento do projeto.
- Na implementação em consola, verifiquei também um *"bug"* relativamente á pesquisa recursiva de pastas, em que quando uma subpasta não tem ficheiros, mas tem outra subpasta o programa não pesquisa nesta subpasta, será também um problema a resolver.

Ao nível do algoritmo, as pesquisas funcionaram bem, e foi atingido o objetivo, que era encontrar as linhas dos ficheiros que contêm as *keywords* introduzidas.

3.4.2 Foco na User Interface

A UI desenvolvida correspondeu ao esperado, e as sensações da primeira impressão e opiniões correspondem ao objetivo proposto de obter uma aplicação de simples utilização. A utilização de cores diferentes nos botões principais, chama a atenção para os campos obrigatórios, sendo que as cores utilizadas correspondem às cores do logotipo, como se fosse uma mnemónica. Durante o desenvolvimento, foram sempre tomados em atenção todos os detalhes quer ao nível gráfico, quer ao nível de bloqueios que pudessem colocar em causa a integridade da aplicação, e ou, provocar confusão no utilizador.

Ao nível da programação ao utilizar MVVM tentei tirar o máximo partido dos *ViewModels*, fazendo a "ponte" entre as *Views* e o *Models*. Utilizei também *Commands* que gerem os *Event Handlers* despoletados pelas *Views* e são parte integrante dos ViewModels. Existem, no entanto, algumas exceções , como por exemplo nos casos em que o controlo é possível de ser gerido apenas na View (Por exemplo o evento de bloqueio do botão *TasksButton,* ou o *eventHandler IntegerUpDown_PreviewTextInput* que apenas permite que se insiram valores numéricos maiores que 1, neste caso para não permitir que o número de tasks seja zero ou menor).

De forma a tornar o código mais legível e reutilizável foram criadas as classes *ViewModelBase* e *CommandBase*, em ambos os casos, o objetivo é que ao criar um *ViewModel* ou *Command* o mesmo herde as propriedades da base, implementando os eventHandlers definidos na base. Sendo que as próprias bases implementam *INotifyPropertyChanged* no caso dos *ViewModels*, e *ICommand* no caso dos *Commands*. Os detalhes de cada ViewModel e Command encontram-se documentados no código como comentário.

Na UI foi utilizado o pacote nuget Extended.Wpf.Toolkit by Xceed, de forma a usufruir da classe *IntegerUpDown* que permite utilizar os botões de setas para alterar valores numéricos, foi necessário no, entanto criar um override do método *DecrementValue()* de forma a que o valor mínimo seja 1. Optei por colocar esta classe *CustomIntegerUpDown* na pasta dos componentes.

Quanto às Views, como já foi referido, foi utilizada a linguagem markup declarativa *XAML,* neste campo, inicialmente senti bastantes dificuldades, principalmente até conseguir entender bem a forma como funcionam as *Bindings* sendo que perdi algum tempo nesta fase do projeto. As Bindings são o núcleo e o principal elo entre os ViewModels e as Views no que a ligações dinâmicas de dados diz respeito. Estas, funcionam por meio de um mecanismo de notificações de alterações chamado de *DependencyProperty*, que permite a associação bidirecional entre uma propriedade de um objeto e um elemento da UI, facultando suporte para atualizações automáticas sempre que o valor da propriedade é alterado. A dificuldade neste caso foi perceber quando tinha de nos *ViewModels* notificar com um *OnPropertyChanged* uma determinada propriedade. Uma vez percebida esta questão o código fluiu.

Assumi neste foco na *UI* e nos *ViewModels*, o risco de gerir o valor máximo de slots de Tasks de processamento que o *SemaphoreSlim* irá utilizar no ViewModel Principal, o *SearchViewViewModel*, mas quis dar ao utilizador a possibilidade de poder alterar o número máximo de Tasks de Pesquisa utilizados. Pode ser uma ferramenta útil em função do universo de pesquisa. Sobre este tema falarei no Foco no Processamento Paralelo.

Após este foco, a junção entre o algoritmo de pesquisa e a UI foi relativamente fácil de implementar, permitindo a publicação da primeira versão com UI do LOGrasper.

3.4.3 Primeira versão executável com UI

A primeira versão da aplicação era meramente o algoritmo referido no ponto 3.4.1 com a UI, funcionava síncrona e sequencialmente, isto é, era instanciado e chamado o *Model SearchEngine* que "carregava" as keywords e construía o autómato AC, seguidamente era chamado o método ainda síncrono *SearchFilesInFolder()* que percorria as pastas recursivamente, e, para cada pasta abria cada ficheiro alimentando cada uma das linhas do ficheiro ao método *Search()* do AC, que por sua vez devolvia apenas as linhas em que se encontravam todas as keywords. Esta versão tinha um problema grave que já era conhecido e foi reportado pelos utilizadores que a testaram, como vamos ver seguidamente.

Após a primeira publicação, a aplicação foi facultada a 6 utilizadores, colegas que laboram no meio de contexto da aplicação, e que a utilizaram durante algum tempo. Posteriormente foi pedido, por meio de um questionário no google Forms, que classificassem a aplicação de 1 a 7 sendo 1 o mínimo e 7 o máximo, relativamente a:

1-A sua utilidade a nível profissional.

2-A sua facilidade de utilização.

3-Aos resultados obtidos (Output).

4-Ao seu desempenho (velocidade de procura), comparativamente com outras ferramentas do mesmo gênero.

5-A sua interface gráfica.

6-Dê a sua opinião crítica pessoal sobre a LOGrasper. Sugestões e críticas construtivas serão bem-vindas.

- Inquérito de Feedback LOGrasper Beta1

Obtive 6 respostas:

	Perguntas					
Cargo	1	2	3	4	5	Sugestões
Analista de TI	7	7	7	5	7	
Coordenador da equipa de suporte N3	5	4	5	4	4	a aplicação não deveria travar a interface gráfica enquanto realiza a pesquisa, poderia ser útil ver o progresso da pesquisa, alertas e condicionar a ação do utilizador até o critério de utilização dos botões estar cumprido (exemplo, não permitir a pesquisa sem ter uma pasta escolhida e pelo menos uma keyword indicada)
Estagiária	7	7	7	6	5	poderia ter uma interface mais apelativa
Consultora Informática	5	6	7	6	5	Prático e cumpre com os requisitos
Analista informático	7	6	6	6	5	
Analista de Integração	7	7	5	5	4	acho que falta só algo que deixe mais claro que o LOGrasper está efetuando a busca, algo mais visual para sinalizar que ele está buscando as informações

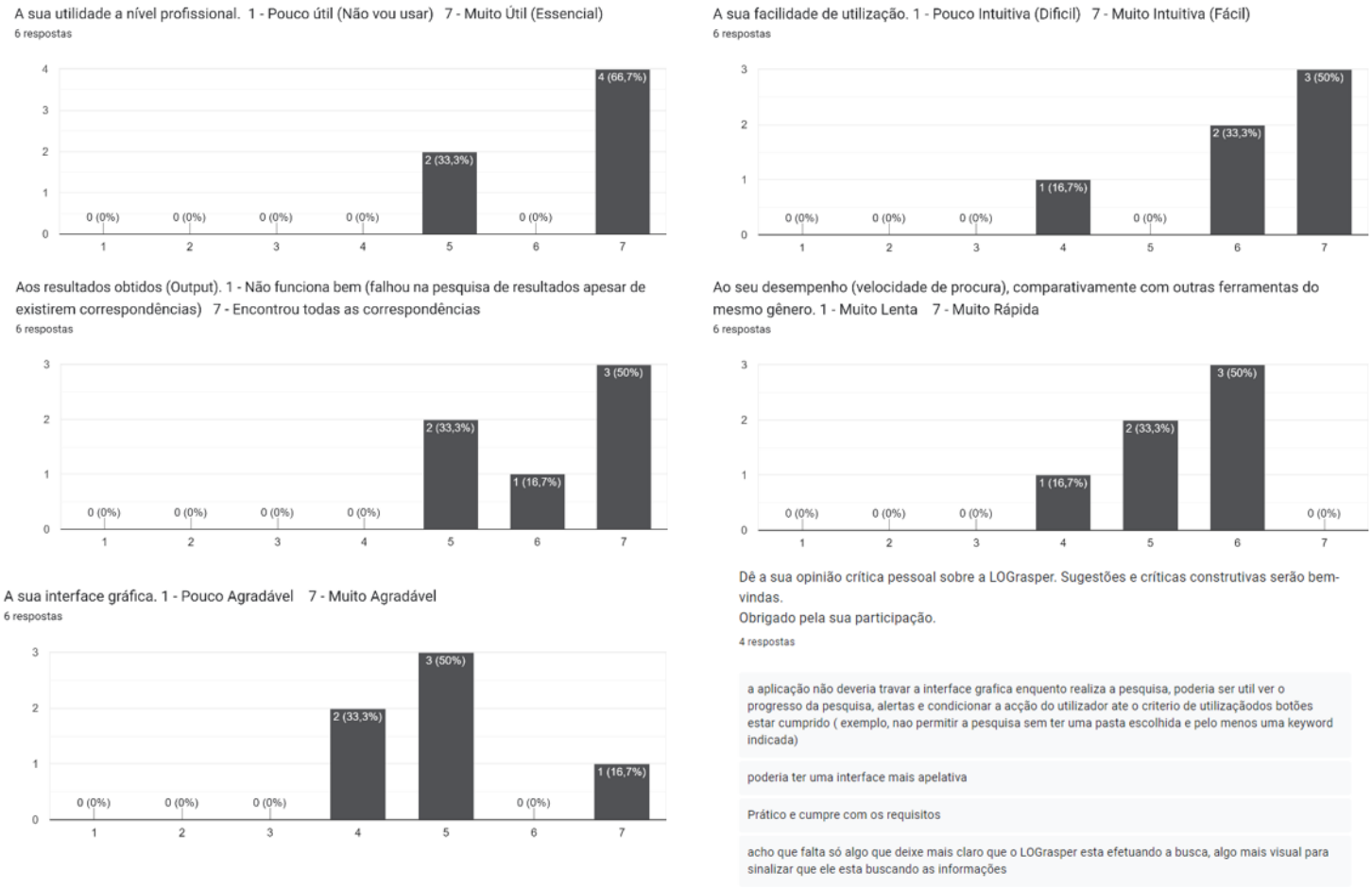


Figura 10 - Respostas ao Questionário online

A partir destas respostas, e de diálogos que mantive com os colegas inquiridos, o feedback obtido apontou sempre na direção da melhoria na performance, e da clareza e robustez da *UI. O* problema grave reportado foi o facto de a aplicação "bloquear" enquanto efetuava a pesquisa, pelo facto de o método de pesquisa estar a ser chamado sincronamente. Assim, decidi focar estes aspetos ao máximo.

O objetivo deste projeto foi desde início desenvolver uma aplicação que pudesse ser útil no dia a dia do analista de suporte, e nada melhor que ter o utilizador alvo a opinar para que se percebam as necessidades, e assim focar o desenvolvimento no que realmente é importante e prioritário. Penso que esta experiência foi decisiva na decisão de adiar o desenvolvimento da fase Beta (templates), para depois da apresentação deste projeto. Será seguramente desenvolvida fora do âmbito desta unidade curricular.

Tenho orgulho em dizer que foi uma aposta ganha na minha ótica, pois neste momento (Release Final), a equipa de trabalho em que me insiro já utiliza o LOGrasper com alguma regularidade, tirando partido da pesquisa por linhas que muita utilidade tem no nosso meio.

3.4.4 Foco na Performance/MultiTasking

A performance passou então a ser o foco principal, nesta fase resolvi algumas questões reportadas pelos *testers*. Um deles, o problema do bloqueio da aplicação enquanto efetuava a pesquisa foi resolvido assim que a pesquisa foi "transformada" numa Task Assíncrona.

3.4.4.1 Tasks Assíncronas

No contexto desta aplicação (linguagem c#), uma task assíncrona permite que um programa execute operações, neste caso métodos, de forma a não bloquear a execução das restantes operações, ou seja, o programa não necessita de esperar que a tarefa termine para que as restantes possam executar. Estas Tasks Assíncronas podem ser identificadas no código com a palavra-chave *async Task,* que acaba por definir o tipo do método, retornando por isso um tipo *Task*, podem também ser do tipo *Task<T>*, onde T é o tipo de retorno desejado. Na chamada aos métodos do tipo *Task* é normalmente utilizada a palavra-chave *await*. Quando o *await* é utilizado na chamada de um método assíncrono, permite que o fluxo de execução continue devolvendo o controlo à chamada do método. O *thread* (processo) atual não é bloqueado ficando livre para realizar outras tarefas enquanto a operação assíncrona está a decorrer. Assim, podemos dizer que o *await* tem uma natureza não bloqueante. Após a conclusão da tarefa assíncrona, o fluxo de execução retorna ao ponto após o *await*, continuando assim a execução a partir desse ponto. (BillWagner, 2023)

A execução assíncrona é particularmente útil em situações em que se pretende evitar bloqueios em operações de Input/Output, como leitura ou escrita em ficheiros acessos a bases de dados, chamadas a Web Services entre outras. Mais importante, permite iniciar várias operações simultaneamente, sem que existam bloqueios (se assim for desejado), mantendo o controlo do fluxo. É, portanto, uma peça chave nesta aplicação, e na forma como é responsiva.

3.4.4.2 Tasks Versus Threads para processamento paralelo

No C# quando se fala em processamento paralelo, existem algumas opções, as classes e bibliotecas, Thread, Task Parallel Library (TPL), Parallel.ForEach, Parallel LINQ, e Async/Await que utiliza TPL. De entre todas as opções optei por estudar as Threads e a TPL com Async/Await.

Uma *Thread* representa uma unidade básica de execução de um programa, é responsável por executar uma sequência de instruções de forma independente, cada uma com o seu contexto de execução, sejam *stacks* de chamadas, registos, etc, ou seja, para que várias threads possam compartilhar recursos, sejam variáveis ou até objetos, têm de ser sincronizadas para evitar problemas de concorrência. É, portanto, uma opção de mais baixo nível e traria dificuldades acrescidas no contexto desta aplicação, já que existem muitos objetos partilhados no decorrer das tarefas de pesquisa, sejam do algoritmo AC, sejam do próprio *OutputObject.* (Microsoft v. a., 2023)

Por outro lado, uma *Task* é uma abstração de alto nível que é fornecida pela biblioteca TPL do .NET. Encapsula uma unidade de trabalho assíncrona que é executada numa Thread disponível, mas de forma mais simples e eficiente comparando com o uso direto de Threads. A TPL é construída em cima do modelo de threads implícito do sistema operativo, mas, gere as threads automaticamente, otimizando a alocação e utilização para um processamento paralelo mais eficiente. Como características importantes saliento a já referida abstração da unidade de trabalho; o agendamento automático; a possibilidade de compor tarefas, isto é, de permitir a criação de hierarquias de tarefas, definir dependências entre elas e até de realizar operações em lote (batch); o importante suporte a operações assíncronas; e o tratamento integrado de exceções. (Microsoft, 2023)

Resumindo, as *Tasks*, simplificam a aproximação ao processamento paralelo assíncrono, simplificando a escrita de código com operações concorrentes, lidando com a complexidade da gestão de threads ao mesmo tempo que fornecem uma forma eficiente de aproveitar os recursos de processamento disponíveis. Dão, portanto, ao programador um nível de abstração muito grande, facilitando em muitos níveis a arquitetura do código.

Estas foram as razões por que escolhi trabalhar com o paradigma Async/Await suportado pela biblioteca TPL.

3.4.4.3 Fluxo e Opções Tomadas nas Tarefas de Pesquisa

Como referido anteriormente, o LOGrasper utiliza o *Model SearchEngine* para efetuar as pesquisas em ficheiros. Este processo inicia quando o utilizador clica no botão *Search* após a respetiva seleção das *keywords* e da pasta raiz dos ficheiros a pesquisar. O ViewModel principal *SearchViewViewModel*, recebe o evento do clique no botão e chama o comando *SearchCommand*, que após algumas operações na UI, inicializa o método *async Task InitiateAsyncSearch()*. Este começa por criar um *SearchObject* que é construído com o caminho da pasta raiz dos ficheiros a pesquisar, e com lista de keywords. Seguidamente é instanciado e construído o *SearchEngine* com o *SearchObject* e o próprio SearchViewViewModel (aqui para que seja possível enviar mensagens para o message dispenser). No passo seguinte é chamada a execução da Task search atribuída à execução da tarefa assíncrona *SearchAC()* do *SearchEngine*, sendo a instrução seguinte um await da tarefa search, que permite que as restantes operações da aplicação possam executar.

3.4.4.3.1 Search Engine

No Model *SearchEngine* é onde decorre toda a operação nuclear da aplicação, a pesquisa nos ficheiros. Como foi referido no ponto 2.1.1.1, o processo inicia-se com a criação do objeto do algoritmo de pesquisa, *AhoCorasick,* neste objeto são carregadas todas as keywords e é construído o autómato AC, é então chamada a tarefa recursiva assíncrona *SearchFilesInFolder()* com todos os argumentos necessários à pesquisa. O algoritmo desta tarefa consiste em percorrer todos os ficheiros contidos em cada uma das pastas e sub-pastas (chamada recursivamente).

Nesta fase entra o processamento paralelo dos ficheiros, sendo a chave as seguintes linhas de código:

```
// Wait for available task slot
maxTasks.Wait();

// Start a new task to search the file asynchronously
Task searchFile = Task.Factory.StartNew(() => SearchFileAsync(file), TaskCreationOptions.LongRunning).ContinueWith((task) => maxTasks.Release());

// Add the search task to the concurrent bag
searchTasks.Add(searchFile);
```

maxTasks é um semáforo do tipo *SemaphoreSlim* que controla o número máximo de tarefas concorrentes, ou seja, enquanto existirem "vagas", o semáforo permite que sejam iniciadas tarefas de pesquisa, assim que for atingido o número máximo (definido por defeito pelo número de processadores lógicos existentes na máquina) de tarefas definidas, cada tarefa representa a pesquisa num ficheiro. A linha seguinte, inicia a procura no ficheiro atual da iteração, em que Task.Factory.StartNew é utilizado para criar e iniciar uma nova *Task,* como argumentos recebe a função que define qual a task a executar, no caso o método *SearchFileAsync()* , as opções de criação da tarefa, neste caso LongRunning indica à TPL que deve tentar alocar uma nova Thread para a tarefa em vez de usar uma do pool de threads existentes. Também fornece indicação ao *TaskScheduler* de que podem vir a ser criadas mais threads para alem das threads de hardware existentes. (MicrosoftTask1, 2023), (MicrosoftTask2, 2023) O método *ContinueWith* permite que se defina uma ação a ser tomada quando uma tarefa é concluída, neste caso a ação a tomar é a libertação da vaga no semáforo.

A tarefa assíncrona SearchFileAsync, é onde decorre a pesquisa no ficheiro, este será percorrido iterativamente linha a linha, onde cada linha será argumento do método *Search()* do algoritmo AC, os resultados serão então introduzidos no Objeto de saída *OutputObject*, e no ViewModel que gere o Output *OuputWindowViewModel*, que por sua vez atualizará a janela de resultados a cada linha encontrada.

Após a criação da tarefa, esta é colocada numa lista do tipo *ConcurrentBag,* esta consiste numa estrutura de dados que fornece uma coleção segura que armazena objetos tipo task, num ambiente de concorrência entre tarefas. Várias threads podem aceder e modificar a coleção de forma simultânea sem necessidade de sincronização manual por parte do programador. Um *await Task.WhenAll(searchTasks);* permite ao controlo do

método principal, *SearchFilesInFolder* saber quando todas as tarefas de pesquisa iniciadas foram concluídas.

A opção de incidir o processamento paralelo em tarefa/ficheiro, foi devido a pretender que a aplicação esteja preparada para todo o tipo de universo de pesquisa, quer seja de poucos ficheiros, mas com linhas com milhões de caracteres, ou por muitos ficheiros com linhas com poucos caracteres, ou ambos. Poderia ter optado por processar linhas de ficheiro paralelamente, mas no caso de muitos ficheiros pequenos perderia performance.

Na leitura dos ficheiros, o bloco,

```
using (StreamReader reader = new StreamReader(file))
```

garante que os recursos utilizados sejam libertados corretamente mesmo que ocorra alguma exceção, esta funcionalidade vem da classe *StreamReader* que implementa a interface *IDisposable*, esta contém o método *Dispose()* que é chamado automaticamente quando utilizamos a palavra *using* no bloco. Esta combinação, dá à aplicação uma gestão muito eficiente da memória, pois a cada ficheiro processado os recursos são libertados, permitindo que o utilizador continue a trabalhar quase normalmente enquanto o LOGrasper pesquisa.

3.4.5 Testes de Performance

Durante o desenvolvimento foram sendo efetuados testes de performance com as várias versões da aplicação. Conseguem tirar-se conclusões que reforçam o foco na Performance, a primeira versão, com os métodos síncronos e sequenciais apresentou os seguintes resultados.

Para um universo de teste com 5 pastas, 27 ficheiros totalizando 945 Mb, efetuei testes com 1,2,3,4,5 e 6 keywords, também de forma a comprovar a eficiência do Algoritmo AC.

Tabela 1 - Keywords de teste 1

	1	2	3	4	5	6
Keywords	jorge	jorge	jorge	jorge	jorge	jorge
		2023-01-10	2023-01-10	2023-01-10	2023-01-10	2023-01-10
			20:37:09	20:37:09	20:37:09	20:37:09
				Services.Controllers	Services.Controllers	Services.Controllers
					Environment	Environment
						TRACE
Total de linhas encontradas	17	4	2	2	2	2

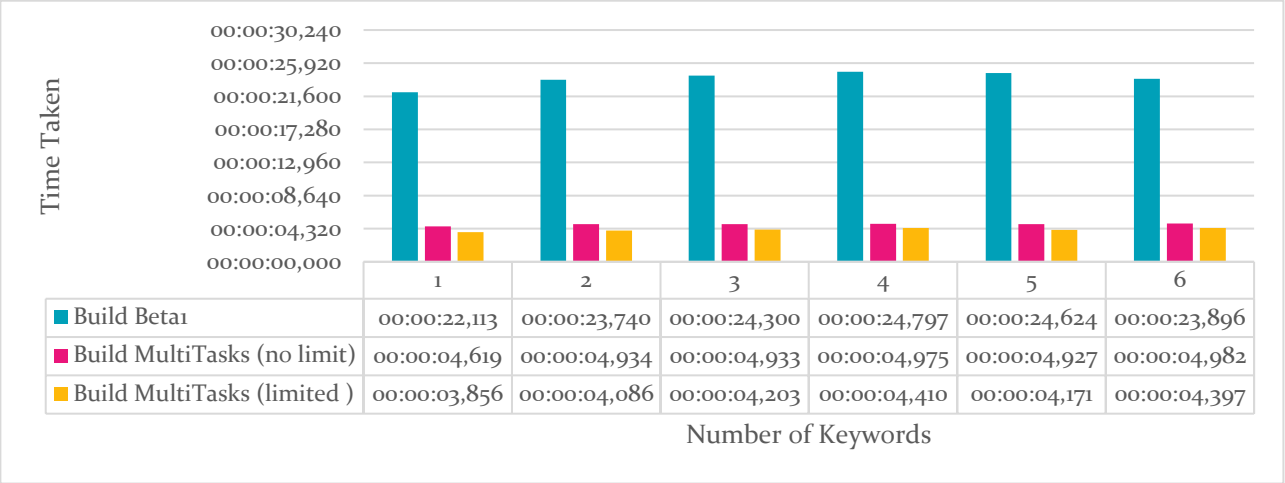


Figura 11 - Gráfico de teste de Performance de Versões

Deste gráfico, podemos concluir que a versão mais eficiente é a versão com limitação de tarefas de pesquisa, podemos também comprovar que o tempo de pesquisa varia muito pouco com o aumento do número de keywords. Tive o cuidado de utilizar o caso mais adverso do algoritmo AC que é quando as keywords não contêm substrings umas das outras, ou seja, não existem *Failure nodes*, o caso adverso em que cada falha na sequência leva o autómato de volta à raiz.

3.4.6 Testes de Memória

Teste à memória, utilizando a aplicação da *JetBrains dotMemory*[6], com um universo em que a pasta raiz contém no total, aproximadamente 53 Gb de um total de 43 pastas e 1152 ficheiros, com 3 keywords, o tempo total da pesquisa foi de cerca de 6 min.

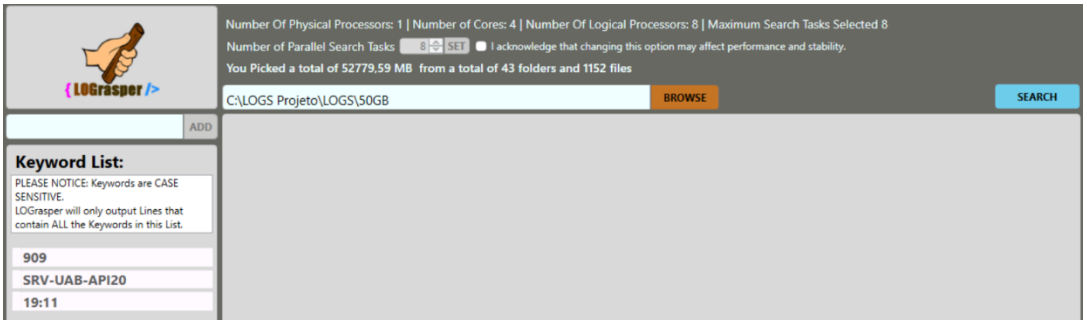


Figura 12 - Pesquisa para teste de memória

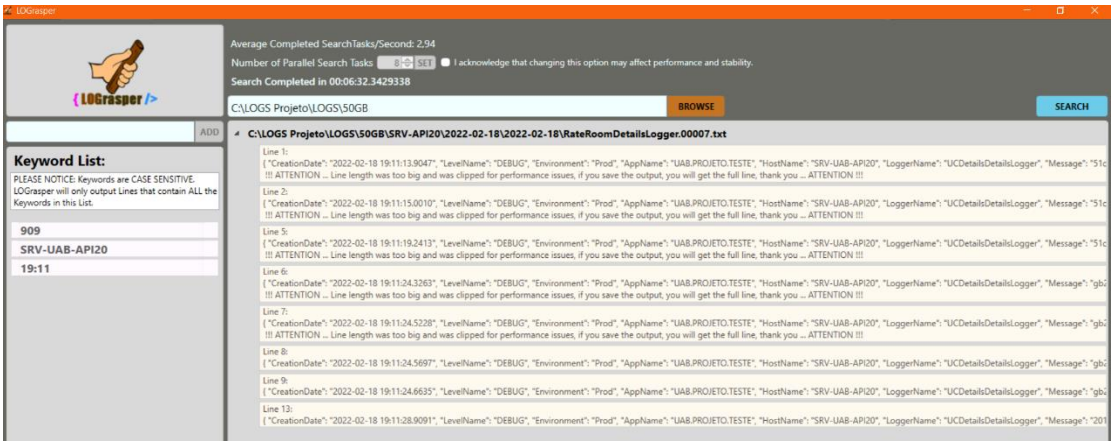


Figura 13 - Resultados da pesquisa para teste de memória

[6] https://www.jetbrains.com/dotmemory/

Nos resultados do teste efetuado pelo dotMemory, conseguem ver-se claramente os *Dispose()* chamados automaticamente no bloco *using (StreamReader reader = new StreamReader(file))*

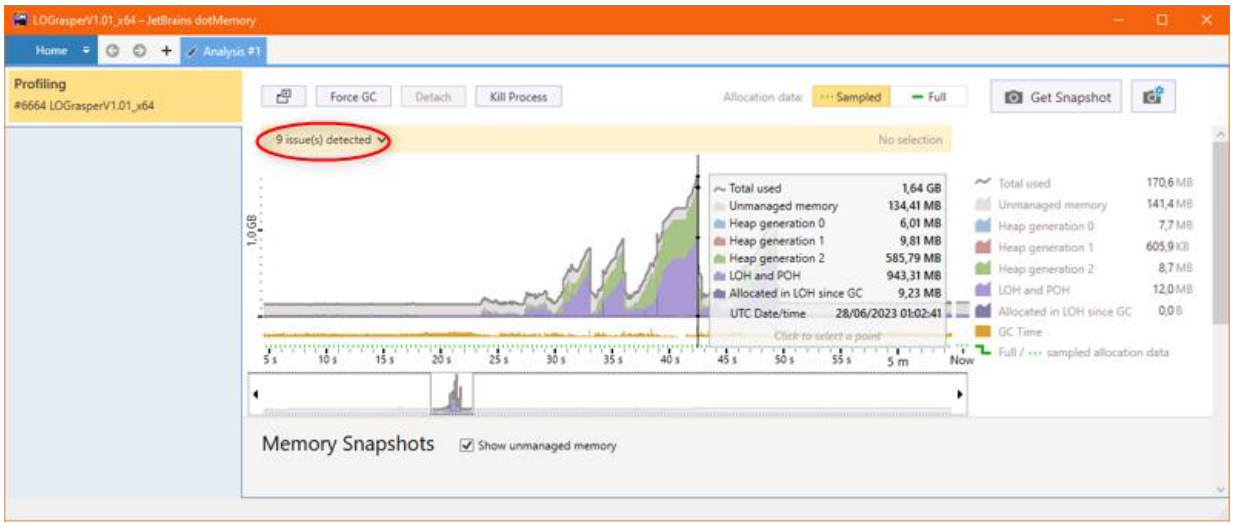


Figura 14 - Resultados do dotMemory

As 9 issues detetadas no decorrer do teste, dizem respeito a momentos de alta pressão, e a taxas de crescimento muito elevadas na memória, no caso quando são pesquisados ficheiros com linhas muito grandes, com cerca de 1 Milhão de caracteres.

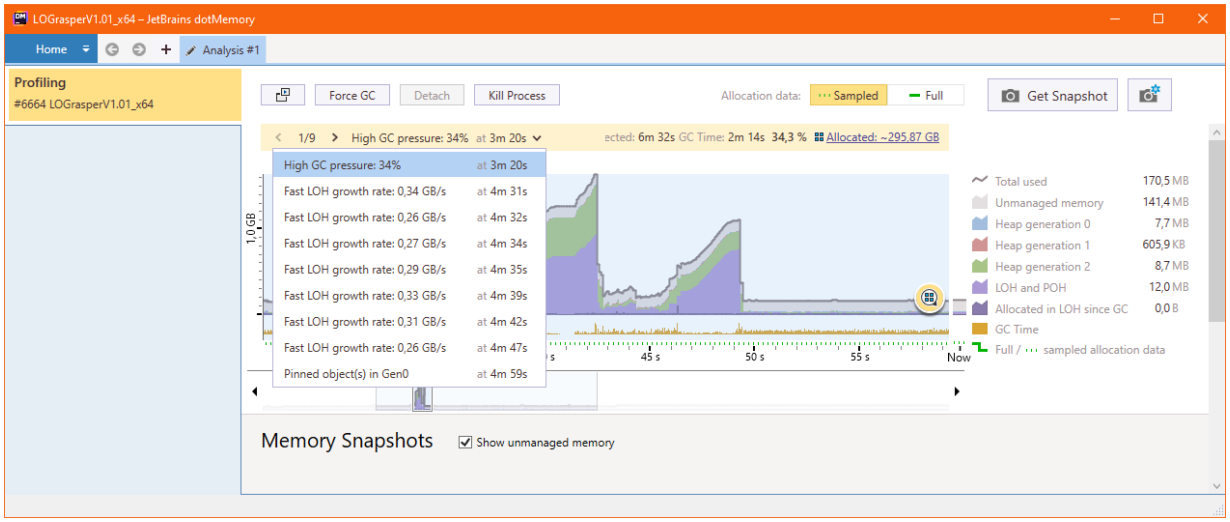


Figura 15 - Issues relatadas no dotMemory

3.4.7 Pequenas adições que não estavam nos requisitos

Durante os períodos de desenvolvimento, senti por vezes necessidade de ter informações e métricas, algumas utilizei para debug, mas acabaram por ficar na versão final, como as mensagens geradas para o *MessageDispenser*, e para o *SystemInfo*.

A funcionalidade de o sistema operativo abrir o ficheiro guardado com o output da pesquisa, assim que é guardado, com a aplicação definida por defeito também foi bastante bem recebida pelos utilizadores.

Capítulo 4

4 Conclusões e Trabalho Futuro

A experiência que ganhei com este projeto, irá ser-me muito útil no futuro profissional, sinto que o desenvolvimento desta aplicação, como que centralizou muitos dos conhecimentos que adquiri no decorrer do curso, cimentando e demonstrando a forma como tudo se une na informática. Neste projeto utilizei conceitos de diferentes unidades curriculares e disciplinas da informática, desde Gestão de Projetos Informáticos, Desenvolvimento de Software, Sistemas Operativos, Estruturas de Dados e Algoritmos Complexos.

4.1. RESULTADOS PRINCIPAIS

Os resultados deste projeto, foram no sentido prático e objetivo da aplicação, cumpridos, desenvolvi uma aplicação funcional e útil, que, apesar de não ter cumprido todos os objetivos a que me propus, me deixou bastante orgulhoso. Está a ser utilizada a nível profissional e com bons resultados.

A escolha do Algoritmo AC foi fundamental sendo um dos pontos chave desta aplicação, como vimos nos testes de performance, o aumento do número de keywords tem um impacto quase desprezável, que, com um universo com muitas keywords tem muito impacto ao nível da performance relativamente a outros algoritmos, já que com uma leitura da string consegue verificar todas as keywords. A adaptação do algoritmo ao contexto da aplicação, ou seja, terminar a pesquisa assim que valida todas as keywords numa linha, também tem impacto na performance. O conhecimento obtido a partir deste algoritmo irá com certeza ser útil no futuro, pois pode ter imensas aplicações.

Ainda na ótica da performance, o sucesso obtido com a utilização das *Tasks* assíncronas paralelas foi, a todos os níveis, outro dos fatores chave, para alem de permitir que a aplicação continue a funcionar sem bloqueios, permitiu passar de aproximadamente 20 segundos a pesquisar aproximadamente 1Gb para 4 segundos.

A escolha e utilização da *StreamReader* juntamente com o termo *using* permite uma boa gestão de recursos principalmente ao nível da memória RAM.

A escolha do *stack* de programação foi por sua vez importante, o WPF permite construir ambientes de UI muito responsivos e dinâmicos, quase como as afamadas aplicações web de com base no *javascript Single Page Aplication (SPA)*. As Bindings dão à aplicação uma comunicação entre componentes muito eficiente.

Os utilizadores que já estão a utilizar a ferramenta, encontraram também uma funcionalidade alternativa. Por vezes mesmo conhecendo as localizações dos Logs pretendidos, os utilizadores utilizam na mesma a ferramenta, mas como gerador de outputs. Colocando as keywords pretendidas de forma a obter um ficheiro apenas com as linhas que lhes interessam, funcionando como uma espécie de filtro de linhas.

4.2. LIMITAÇÕES

Gostaria de ter tido mais tempo para desenvolver mais testes de forma a poder aconselhar os utilizadores na forma mais eficiente de utilizar a aplicação. Quando o objeto de saída começa a ficar muito grande, nota-se algumas quebras na aplicação, ficando por vezes com a mensagem do Windows Não Responde, mas acaba por voltar a responder se aguardarmos. Esta situação pode ocorrer, no caso de se inserir uma keyword com apenas um carácter, e nas linhas existir sempre esse caracter, caso existam muitos ficheiros, a UI poderá ficar um tempo sem responder e lenta.

4.3. DESENVOLVIMENTO FUTURO E MELHORIAS

Para desenvolvimento futuro, espero implementar a fase Beta (Templates), objetivo que não foi desenvolvido em perterimento da obtenção do melhor desempenho, mas penso que foi uma decisão tomada com consciência e que desenvolvi uma aplicação mais eficiente pelo facto de ter abdicado desde objetivo. No entanto será desenvolvido.

Outras melhorias que espero implementar são as seguintes: ao nível da UI, tenho consciência que a escolha das cores poderá não ter sido a melhor, e que existem formas mais apelativas de mostrar a informação no ecrã, mas na verdade não tenho muita experiência em UI/UX. Ao nível das métricas de pesquisa, gostaria de implementar barras de progresso e outras formas de mostrar as pesquisas a decorrer.

Também no âmbito da UI, seria interessante implementar uma forma de abrir apenas uma linha de output a partir da janela, ou seja, uma forma de o utilizador abrir num editor externo apenas a linha que pretende.

Anexo I – Projeto Visual Studio 2022

Todo o desenvolvimento do código foi efetuado com a ferramenta GitHub, no repositório pode ver-se toda evolução do código.

Ficheiro em anexo -> "LOGrasper_VisualStudioSolution.zip"

Repositório GitHub -> https://github.com/jmcmatias/LOGrasper

One Drive -> https://uabpt-my.sharepoint.com/:f:/g/personal/1901807_estudante_uab_pt/EgYfsDu17EhMtVm20DbmMrsByaovu4esnVVr4N4ikPbFaA?e=0EY6x4

Anexo II – Executável da Aplicação

Ficheiro em anexo -> "LOGrasperV1.01_x64.exe.zip"

Release GitHub -> https://github.com/jmcmatias/LOGrasper/releases/tag/v1.0.1

One Drive -> https://uabpt-my.sharepoint.com/:f:/g/personal/1901807_estudante_uab_pt/EgYfsDu17EhMtVm20DbmMrsByaovu4esnVVr4N4ikPbFaA?e=0EY6x4

Anexo III – Manual de Instruções da Aplicação

Ficheiro em anexo -> "LOGrasper - Manual de Instruções.pdf"

Repositório GitHub -> https://github.com/jmcmatias/LOGrasper/blob/master/LOGrasper%20-%20Manual%20de%20Instru%C3%A7%C3%B5es.pdf

One Drive -> https://uabpt-my.sharepoint.com/:f:/g/personal/1901807_estudante_uab_pt/EgYfsDu17EhMtVm20DbmMrsByaovu4esnVVr4N4ikPbFaA?e=0EY6x4

Bibliografia

adegeo, & v-trisshores. (16 de 03 de 2023). *https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml*. Obtido de XAML overview (WPF .NET): https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-7.0

astrogrep.sourceforge.net. (s.d.). Obtido de astrogrep.sourceforge.net/features/: https://astrogrep.sourceforge.net/features/

BillWagner. (10 de 05 de 2023). *https://learn.microsoft.com/pt-br/dotnet/csharp/asynchronous-programming/async-scenarios*. Obtido de https://learn.microsoft.com: https://learn.microsoft.com/pt-br/dotnet/csharp/asynchronous-programming/async-scenarios

fileseek.ca. (s.d.). Obtido de fileseek.ca/Compare: https://www.fileseek.ca/Compare/

Microsoft. (s.d.). *https://learn.microsoft.com/pt-br/dotnet/api/system.threading.tasks.task-1?view=net-7.0*. Obtido de https://learn.microsoft.com: https://learn.microsoft.com/pt-br/dotnet/api/system.threading.tasks.task-1?view=net-7.0

Microsoft, v. a. (10 de 05 de 2023). *https://learn.microsoft.com/pt-br/dotnet/standard/threading/threads-and-threading*. Obtido de https://learn.microsoft.com: https://learn.microsoft.com/pt-br/dotnet/standard/threading/threads-and-threading

MicrosoftTask1. (s.d.). *https://learn.microsoft.com/pt-br/dotnet/api/system.threading.tasks.taskcontinuationoptions?view=net-7.0#system-threading-tasks-taskcontinuationoptions-longrunning*. Obtido de https://learn.microsoft.com: https://learn.microsoft.com/pt-br/dotnet/api/system.threading.tasks.taskcontinuationoptions?view=net-7.0#system-threading-tasks-taskcontinuationoptions-longrunning

MicrosoftTask2. (s.d.). *https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcreationoptions?view=net-7.0*. Obtido de https://learn.microsoft.com: https://learn.microsoft.com/en-us/dotnet/api/system.threading.tasks.taskcreationoptions?view=net-7.0

Moshiri, N. (26 de 04 de 2020). *Advanced Data Structures: Aho-Corasick Automaton*. Obtido de https://www.youtube.com/: https://www.youtube.com/watch?v=O7_w001f58c

Moshiri, N. (26 de 04 de 2020). *Advanced Data Structures: Aho-Corasick Dictionary Links*. Obtido de https://www.youtube.com/: https://www.youtube.com/watch?v=OFKxWFew_Lo

Mythicsoft. (s.d.). *https://www.mythicsoft.com*. Obtido de https://www.mythicsoft.com/agentransack/: https://www.mythicsoft.com/agentransack/

Silva, A. M., & Videira, C. A. (2001). UML, Metodologias e Ferramentas CASE. Em A. M. Silva, & C. A. Videira, *Linguagens de Modelação UML, Metodologias e Ferramentas CASE na concepção e Desenvolvimento de Software.* CentroAtlântico.pt.

Syromiatnikov, A., & Weyns, D. (s.d.). *https://ieeexplore.ieee.org/*. Obtido de A Journey through the Land of Model-View-Design Patterns: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6827095

Trivedi, U. (2020). *An Optimized Aho-Corasick Multi-Pattern Matching Algorithm for Fast Pattern Matching*. Retrieved from https://ieeexplore.ieee.org/: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9342041

Vashchegin, R. (s.d.). *Conquer String Search with the Aho-Corasick Algorithm*. Obtido de https://www.toptal.com/: https://www.toptal.com/algorithms/aho-corasick-algorithm