Relatório Integração de Sistemas de Informação

Trabalho Prático I

Análise de Popularidade e Sentimento de Temas com ETL em KNIME

mongo DB

.NET

Trabalho realizado por:

Diogo Carvalho - 25245

Licenciatura em Engenharia Sistemas Informáticos

3.ºano

Barcelos, outubro de 2025

Índice

Índice de Imagens

Índice de Tabelas

1. Introdução

O presente relatório foi desenvolvido no âmbito da unidade curricular Integração de Sistemas de Informação, do 3.º ano da Licenciatura em Engenharia de Sistemas Informáticos do Instituto Politécnico do Cávado e do Ave. O trabalho surge com o propósito de aplicar, de forma prática, os conhecimentos adquiridos ao longo do semestre, através da aplicação e experimentação de ferramentas em processos de ETL (Extract, Transformation and Load), inerentes a processos de Integração de Sistemas de informação ao nível dos dados.

O aumento da quantidade de informação disponível na internet, principalmente em sites de notícias e redes sociais, permite analisar em tempo real a forma como determinados assuntos são discutidos pela sociedade. No entanto, estes dados encontram-se dispersos, com diferentes formatos, estruturas e níveis de fiabilidade, o que torna necessária a criação de processos de integração, tratamento e análise automática. É neste contexto que surge o presente projeto.

O trabalho desenvolvido consiste na criação de um sistema capaz de recolher, processar e analisar diariamente dados de diferentes fontes online, com o objetivo de avaliar a popularidade de um tema e o sentimento associado a esse tema ao longo do tempo. Para isso, foi desenvolvido um conjunto de componentes que trabalham de forma integrada, para a extração e transformação e análise dos dados, e para armazenamento histórico dos resultados e dos registos de execução.

Este sistema permite automatizar todo o processo, desde a recolha da informação, até à geração de métricas como número de notícias/publicações referentes ao tema escolhido por dia, tendências de popularidade e classificação de sentimento (positivo, neutro ou negativo).

Ao longo deste relatório são descritos os objetivos do projeto, as tecnologias utilizadas, a arquitetura do sistema, a construção dos processos ETL, os resultados obtidos, bem como possíveis melhorias futuras.

2. Enquadramento

O trabalho tem como objetivo aplicar conceitos de integração de dados e processos ETL (Extract, Transform, Load) na construção de um sistema real com fontes de dados heterogéneas.

2.1 Contextualização

O tema escolhido consiste na análise da popularidade e do sentimento associado a temas específicos, com base em notícias e publicações provenientes da internet. Para isso, foi criada uma API em **.NET** que agrega dados de *APIs* publicas como a **NewsAPI** e da **Reddit API** (com possibilidade de expansão), centralizando a recolha de informação num único ponto de acesso. Os dados recolhidos são depois processados na plataforma **KNIME**, onde são aplicadas operações de transformação, normalização, análise de sentimento com *scripts* **Python** e cálculo de popularidade por dia.

Os resultados são armazenados numa base de dados **MongoDB**, juntamente com registos de execução (*logs* de sucesso e insucesso), o que permite manter um histórico das tendências ao longo do tempo. Todo o processo é automatizado através de uma aplicação em **C#**, que executa diariamente o *workflow* do **KNIME** de forma autónoma.

3. Problema

Atualmente, a informação sobre determinados temas — como tecnologia, política, marcas, mercados financeiros é publicada de forma constante em jornais online e redes sociais. Ou seja, esta informação encontra-se **dispersa por múltiplas plataformas**, cada uma com o seu próprio formato, estrutura e qualidade de dados, o que dificulta a análise integrada e automática da **popularidade desses temas e do sentimento do público em relação a eles**.

Para além disso:

- **Os dados são heterogéneos e frequentemente inconsistentes**, uma vez que podem conter campos diferentes, textos informais, abreviações, sarcasmo ou ruído.
- **O sentimento do público pode mudar rapidamente**, especialmente em temas voláteis como as **criptomoedas**, onde o entusiasmo ou o medo dos investidores varia consoante eventos de mercado, notícias de regulação ou declarações de figuras públicas.

- Da mesma forma, **empresas e marcas** podem necessitar de avaliar como o público reage a um **lançamento de um produto**, **campanha publicitária** ou a uma **crise de reputação**, analisando publicações que expressem opiniões positivas, negativas ou neutras sobre a organização.

Perante este cenário, o problema a resolver consiste em **desenvolver um sistema capaz de:**

- Recolher automaticamente informação de diferentes fontes online (ex.: NewsAPI, Reddit etc…);
- Unificar esses dados num formato único e estruturado;
- Limpar, transformar e remover inconsistências ou duplicados;
- **Calcular indicadores de popularidade**, como o número de notícias/publicações por dia sobre um determinado tema;
- **Avaliar o sentimento predominante** dos textos;
- **Armazenar o histórico de resultados** para permitir comparações ao longo do tempo;
- Garantir que todo o processo é **automático, repetível e escalável**, gerando logs de sucesso e erro para garantir fiabilidade.

Deste modo, torna-se possível responder a questões como:

- *"A popularidade do tema 'cryptocurrency' aumentou ou diminuiu nos últimos dias?"*
- *"O sentimento do público em relação à marca X é maioritariamente positivo, neutro ou negativo?"*
- *"Houve mudanças bruscas de opinião após determinado evento ou notícia?"*

4. Estratégia Geral do Sistema

A estratégia adotada para este projeto baseia-se na implementação de um processo completo de **ETL (Extract, Transform, Load)**, capaz de recolher dados de fontes externas, transformá-los e armazená-los de forma automática e estruturada.

O sistema foi desenvolvido com uma arquitetura modular composta por quatro componentes principais:

4.1 Extração de dados (Extract)

A informação é recolhida a partir de duas fontes externas:

NewsAPI - fornecedora de notícias de fontes jornalísticas.

Reddit API - plataforma social com opiniões e discussões de utilizadores.

Para integrar ambas as fontes de forma uniforme, foi criada uma **API própria em .NET**, que recebe parâmetros como tema, número de dias, língua e ordenação, consulta ambas as APIs externas e devolve um único JSON normalizado com todas as publicações.

4.2 Transformação de dados (Transform)

Os dados recebidos são processados no **KNIME**, onde é implementado o fluxo ETL completo:

- Conversão do JSON para tabela de dados;
- Limpeza e normalização (Filtragem dos campos necessários e limpeza de ruído textual);
- Conversão e formatação de datas com expressões regulares;
- Cálculo de popularidade (número de publicações por dia);
- Cálculo de sentimento com scripts em Python aplicados ao texto do título;
- Preparação dos dados final com toda a informação necessária para armazenamento.

4.3 Carregamento de dados (Load)

Após o processamento, os dados são inseridos numa base de dados **MongoDB**, organizada em coleções como:

topic_trends - dados transformados, com data, sentimento, popularidade e tema;

execution_logs - registo de sucesso, falhas, timestamps e parâmetros de cada execução.

4.4 Automação do processo

Para garantir que o sistema funciona diariamente sem intervenção manual, foi criada uma aplicação em **C# (Job Scheduler)** que:

- Executa o workflow do KNIME em modo batch;
- Gera logs de execução (sucesso/erro);
- Permite a recolha contínua de dados históricos.

4.5 Analise dos resultados

Ainda dentro do KNIME é possível visualizar um relatório com os gráficos da informação captada durante uma determinada execução.

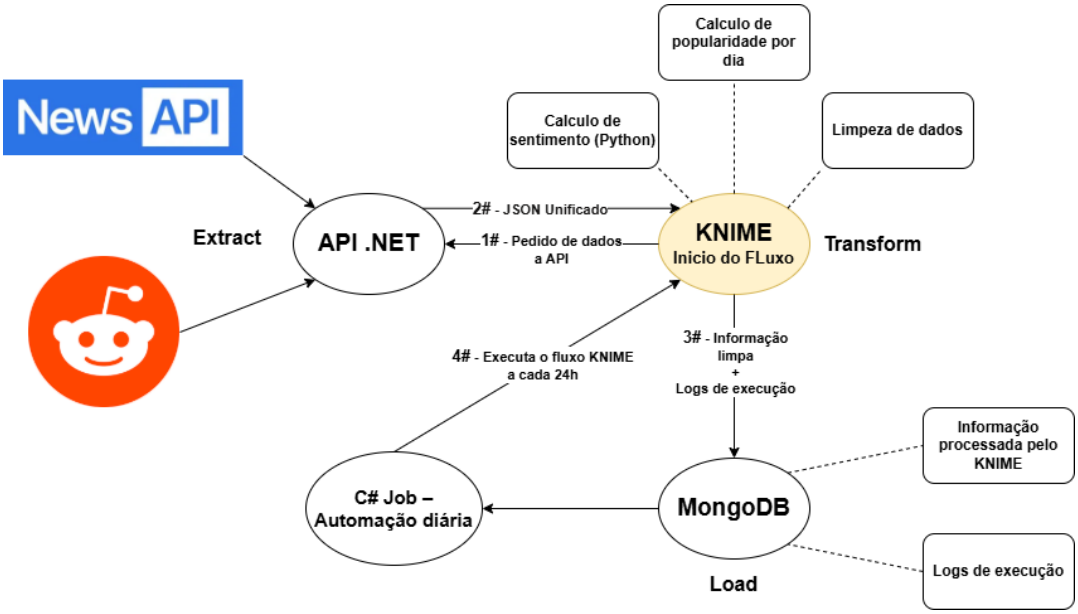


Figura 1 - Diagrama Fluxo Geral do Sistema (Resumo ETL)

5. Extração de Dados — API em .NET (Extract)

5.1 Objetivo da API

A API em .NET funciona como uma **camada de agregação** entre as fontes externas (NewsAPI e Reddit) e o KNIME.

Os seus objetivos são:

- **Unificar formatos** (cada fonte retorna um JSON diferente) em um **modelo único (ArticleDto)**;

```
public class ArticleDto
{
    public string Source { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public string Content { get; set; }
    public string Url { get; set; }
    public DateTime? PublishedAt { get; set; }

}
```

Figura 2 - Modelo de dados ArticleDto

- **Aplicar filtros comuns** (tema, idioma, número de dias, ordenação, paginação) no ponto de entrada;

topic	crypto
days	7
pageSize	100
pageNumber	1
language	pt
sortBy	popularity

Figura 3 - Exemplo de filtros comuns aplicados na API

- **Reduzir o acoplamento** do KNIME às APIs externas (qualquer mudança nas APIs é tratada na API de Agregação);
- **Melhorar desempenho** lançando chamadas às duas fontes **em paralelo** e removendo **duplicados** antes de devolver ao consumidor.

5.2 Arquitetura do Projeto

A solução segue uma organização por camadas:

- **Controllers/**
  - **NewsController.cs** - apresenta o endpoint /api/news, recebe parâmetros, validações e devolve a resposta unificada.
- **Services/**
  - **AggregationService.cs** - **orquestra** as duas fontes em paralelo, combina resultados, **remove duplicados por URL**, ordena por data e faz a **paginação final**.
  - **NewsService.cs** - consome a **NewsAPI**, mapeia para ArticleDto.
  - **RedditService.cs** - consome a **Reddit API** (endpoint de pesquisa), aplica seleção de subreddits por idioma, filtra por intervalo temporal e mapeia para ArticleDto.
- **Models/**
  - **ArticleDto.cs** - modelo **único** devolvido ao cliente.
  - **NewsApiResponse.cs** - modelos para desserializar a resposta da NewsAPI.
  - **RedditModels.cs** - modelos para desserializar a resposta do Reddit.
- **Infraestrutura**
  - **Program.cs** - registo de serviços.

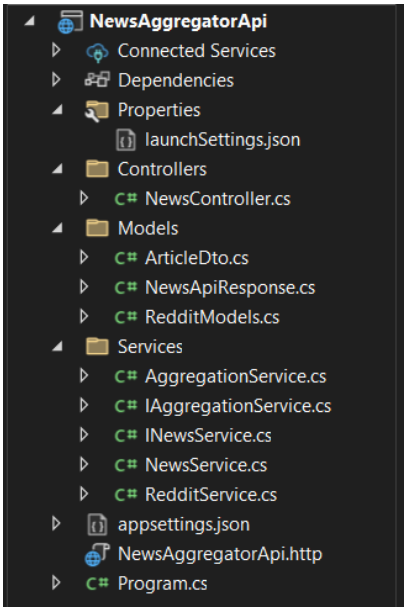


Figura 4 - Demonstração da arquitetura da API

5.3 Funcionamento do endpoint /api/news

O endpoint principal está no **NewsController** e recebe os parâmetros de consulta: topic (obrigatório), days, pageNumber, pageSize, language (opcional), sortBy (opcional). Faz logging, chama o serviço de agregação e devolve um **JSON com metadados + lista de artigos**.

Pipeline interno:

1. Controller valida topic e regista o pedido (logging).
2. Chama **AggregationService**.*GetAggregatedArticlesAsync(...)*.
3. O AggregationService lança **em paralelo**:
    - **NewsService**.*GetArticlesAsync*(...) — NewsAPI;
    - **RedditService**.*GetRedditArticlesAsync*(...) — Reddit;
4. Junta resultados, **remove duplicados por URL**, ordena por **PublishedAt** (das mais antigas para as mais recentes) e aplica **paginação**.
5. Controller devolve objeto com topic, daysUsed, totalResults e articles (lista ArticleDto).


```
{
  "topic": "crypto",
   "daysUsed": 7,
   "totalResults": 7,
   "articles": [
       {
           "source": "Reddit - r/criptomoedas",
           "title": "USDc e USDt: Qual o pulo do gato?",
           "description": "Ola, eu entendo que essas stable coins estão/vão mudar a forma como o dinheiro transita pelo mundo",
           "content": null,
           "url": "https://www.reddit.com/r/criptomoedas",
           "publishedAt": "2025-10-12T23:42:41Z"
       }
  ]
}
```


Figura 5 - Exemplo da resposta da API com um registo ilustrativo

Filtros e ordenação suportados:

- topic - termo de pesquisa em ambas as fontes.
- days - janela temporal (NewsAPI limitada a 30 dias na versão gratuita).
- language - passado à NewsAPI; no Reddit influencia seleção de subreddits (pt/en).
- sortBy — NewsAPI (ex.: publishedAt); Reddit mapeado para new/top consoante popularidade e janela temporal.
- pageNumber, pageSize — paginação final após agregação e limpeza de duplicados.

5.4 Serviço de Agregação – AggregationService

Este serviço é oda API, responsável por **combinar os dados das duas fontes externas - NewsAPI e Reddit - num único resultado estruturado**.

Funções principais:

- Recebe os parâmetros do controller (topic, days, language, sortBy, pageNumber, pageSize).
- Chama simultaneamente os serviços NewsService e RedditService.
- Aguarda ambas as respostas com Task.WhenAll(...).
- Junta todos os artigos num único conjunto de dados.
- **Remove duplicados com base no campo URL** (útil porque uma notícia viral pode aparecer nas duas plataformas).
- Ordena os artigos por data (PublishedAt) do mais antigo para o mais recente.
- No final aplica paginação.

5.5 Serviço de Notícias – NewsService (NewsAPI)

Este serviço trata exclusivamente da comunicação com a **NewsAPI**.

Principais etapas:

- Recebe os mesmos parâmetros (topic, days, language, sortBy, pageNumber, pageSize).
- Gera a URL da NewsAPI com a chave em appsettings.json.

  Exemplo:

  https://newsapi.org/v2/everything?q=crypto&from=2025-10-12&sortBy=publishedAt&page=1&pageSize=20&language=en&apiKey=XXXX
- Efetua a requisição HTTP e valida o código de resposta.
- Converte o JSON recebido para o modelo NewsApiResponse.
- Faz o mapeamento para o formato unificado ArticleDto (título, descrição, conteúdo, data, etc.).


```
{
"status": "ok",
"totalResults": 6593,
"articles": [
    {
        "source": {
            "id": null,
            "name": "Popular Science"
    },
    "author": "Mack DeGeurin",
    "title": "Robot hands are becoming more human",
    "description": "From Boston Dynamics' three-fingered Atlas bot to Figure's five-digit model.\nThe post Robot hands
    "url": "http://www.popsci.com/technology/robot-hands-are-becoming-more-human/",
    "urlToImage": "https://www.popsci.com/wp-content/uploads/2025/10/Atlas_hands_front_facing.jpg?quality=85&w=1200",
    "publishedAt": "2025-10-19T18:00:00Z",
    "content": "If you want to guess the purpose of any given futuristic humanoid robot, look at its hands. Last week,
    }
]
}
```

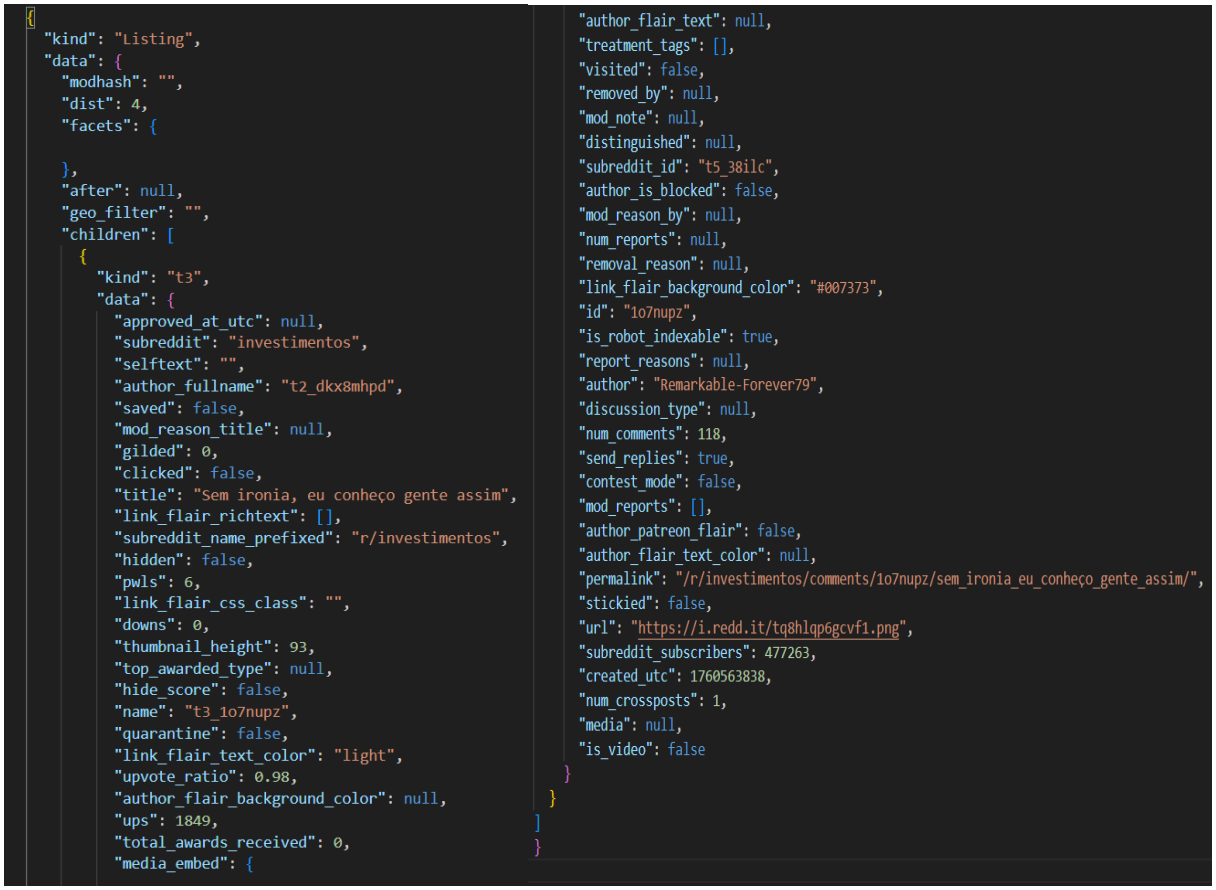

Figura 6 - Exemplo real da resposta da news api antes da transformação para ArticleDto

5.6 Serviço de Reddit – RedditService

Este serviço é responsável pela recolha de publicações da **Reddit API** (endpoint de pesquisa pública https://www.reddit.com/search.json).

Principais funcionalidades:

- Define user-agent para evitar bloqueios da API pública.
- Seleciona automaticamente subreddits diferentes conforme a língua (pt ou en).
- Constrói a query combinando o *tema* com a lista de subreddits (ex: crypto (subreddit: Bitcoin ou subreddit: CryptoCurrency)).
- Aplica filtros por ordenação (top, new) e por intervalo de tempo (week, month, etc.).
- Filtra resultados por data usando timestamp UNIX created ou created_utc (campos da resposta real da Reddit API).
- Converte o resultado para ArticleDto com:
    - Source = Reddit - r/Subreddit
    - Title = título do post
    - Description = selftext
    - Url = Permalink completo
    - PublishedAt = conversão do timestamp UNIX



```
{
  "kind": "Listing",
  "data": {
    "modhash": "",
    "dist": 4,
    "facets": {

    },
    "after": null,
    "geo_filter": "",
    "children": [
      {
        "kind": "t3",
        "data": {
          "approved_at_utc": null,
          "subreddit": "investimentos",
          "selftext": "",
          "author_fullname": "t2_dkx8mhpd",
          "saved": false,
          "mod_reason_title": null,
          "gilded": 0,
          "clicked": false,
          "title": "Sem ironia, eu conheço gente assim",
          "link_flair_richtext": [],
          "subreddit_name_prefixed": "r/investimentos",
          "hidden": false,
          "pwls": 6,
          "link_flair_css_class": "",
          "downs": 0,
          "thumbnail_height": 93,
          "top_awarded_type": null,
          "hide_score": false,
          "name": "t3_1o7nupz",
          "quarantine": false,
          "link_flair_text_color": "light",
          "upvote_ratio": 0.98,
          "author_flair_background_color": null,
          "ups": 1849,
          "total_awards_received": 0,
          "media_embed": {
          "author_flair_text": null,
          "treatment_tags": [],
          "visited": false,
          "removed_by": null,
          "mod_note": null,
          "distinguished": null,
          "subreddit_id": "t5_38ilc",
          "author_is_blocked": false,
          "mod_reason_by": null,
          "num_reports": null,
          "removal_reason": null,
          "link_flair_background_color": "#007373",
          "id": "1o7nupz",
          "is_robot_indexable": true,
          "report_reasons": null,
          "author": "Remarkable-Forever79",
          "discussion_type": null,
          "num_comments": 118,
          "send_replies": true,
          "contest_mode": false,
          "mod_reports": [],
          "author_patreon_flair": false,
          "author_flair_text_color": null,
          "permalink": "/r/investimentos/comments/1o7nupz/sem_ironia_eu_conheço_gente_assim/",
          "stickied": false,
          "url": "https://i.redd.it/tq8hlqp6gcvf1.png",
          "subreddit_subscribers": 477263,
          "created_utc": 1760563838,
          "num_crossposts": 1,
          "media": null,
          "is_video": false
        }
      }
    ]
  }
}
```


Figura 7 - Exemplo da resposta real da Reddit API antes da transformação para ArticleDto

6. Transformação de Dados no KNIME (Processo ETL)

6.1 Objetivo da fase de Transformação

Depois de os dados serem recolhidos pela API .NET, o KNIME é responsável por:

- Ler os dados em formato JSON;
- Transformá-los numa tabela estruturada;
- Limpar textos, converter tipos, extrair datas;
- Calcular popularidade (nº de publicações por dia);
- Calcular sentimento com Python;
- Separar caminho de sucesso e insucesso;
- Inserir os resultados no MongoDB, bem como os logs de execução.

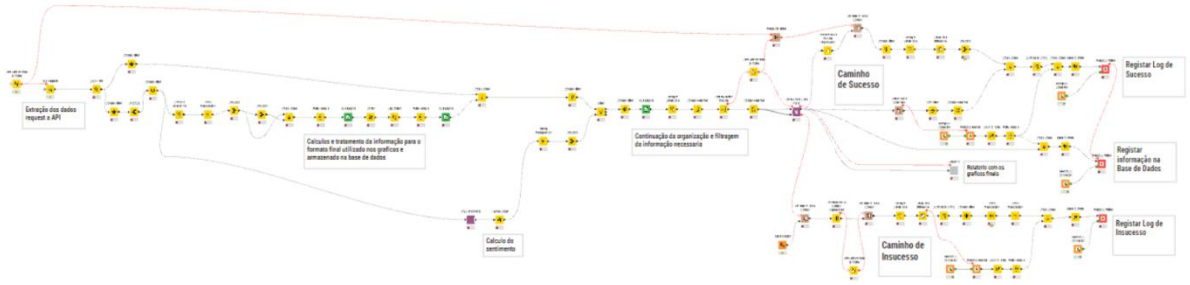


Figura 8 - Visão geral do Workflow no KNIME

6.2 Visão Geral do Workflow

O processo de transformação dos dados foi desenvolvido integralmente na plataforma **KNIME Analytics Platform**, utilizando um workflow visual que representa todas as etapas do ETL. O fluxo está dividido em várias zonas funcionais claramente identificadas:

- **Extração dos dados (request à API)**: Utilização de um node Java Edit Variable para receber o *topic*, seguido de um **GET Request** para consumir a API .NET.
- **Transformação principal dos dados**: Conversão do JSON recebido em tabela, limpeza das colunas, conversão de datas, contagem de artigos por dia e cálculo de métricas.
- **Cálculo de Sentimento**: Através de um **Python Script Node**, utilizando bibliotecas de análise de texto para atribuir um valor de sentimento a cada publicação.
- **Caminho de Sucesso e Caminho de Insucesso**: Caso o processo decorra corretamente, os dados transformados são inseridos no MongoDB e é criado um log de sucesso.
  Se ocorrer erro, é aberto o fluxo alternativo onde os dados de erro são guardados na base de dados e registados como insucesso.
- **Finalização**: Inserção dos dados na coleção topic_trends, e registo de execução em execution_logs

6.3 Registo do início da execução

- O processo começa com um **Java Edit Variable**, que cria uma variável chamada exec_start.
- Esta variável guarda o timestamp do momento em que a execução começou (formato yyyy-MM-dd'T'HH:mm:ss.SSS).
- Esta informação é posteriormente usada para registar logs na base de dados, permitindo identificar quando a execução iniciou e terminou.

6.4 Extração dos dados da API

- Utiliza-se o nó **GET Request** para fazer a chamada ao endpoint da API .NET.
- O resultado vem em formato JSON.
- O nó **JSON Path** extrai o array de artigos da resposta e transforma-o numa tabela.
- Cada linha da tabela representa uma notícia/publicação com campos como título, descrição, URL, fonte e data.

6.5 Transformação dos dados

Após a extração do JSON da API, os dados são preparados, limpos e transformados para permitir análises de popularidade e sentimento. Esta fase do workflow inclui: normalização de datas, contagem de publicações por dia, cálculo de métricas de popularidade e atribuição de rótulos (labels) como “Alta”, “Normal”, “Baixa”, “Crescimento”, “Declínio” ou “Estável”.

6.5.1 Conversão e normalização de datas

- A data original vem no formato ISO (2025-10-14T12:30:00Z).
- Utiliza-se o nó **String Manipulation** com regexReplace($PublishedAt$, "T.*", "") para remover a parte da hora e ficar apenas com a data (2025-10-14).
- O nó **String to Date&Time** converte essa string para um tipo Local Date, permitindo operações temporais.

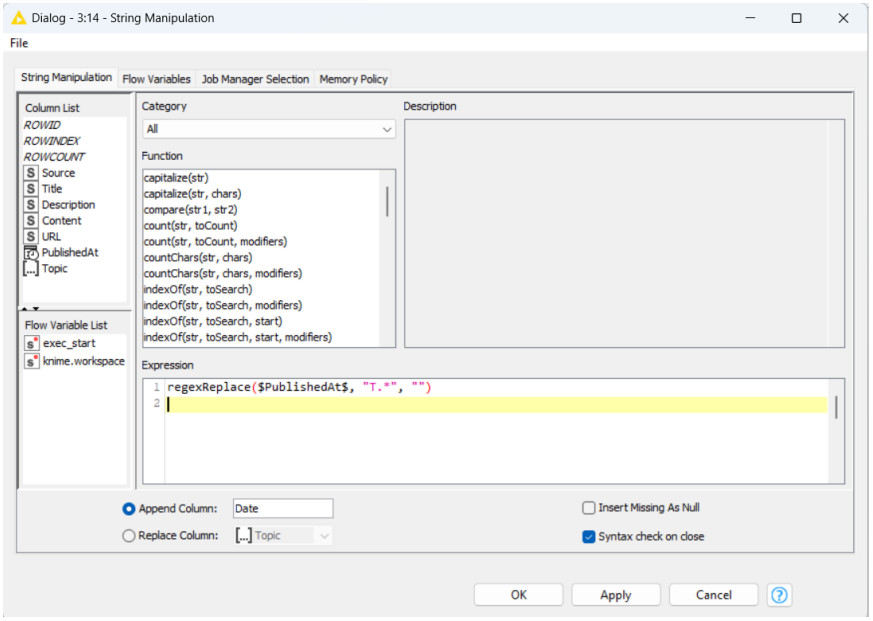


Figura 9 - Expressão aplicada a data no KNIME

6.5.2 Cálculo de Popularidade (número de publicações por dia)

- Com o nó **GroupBy**, conta-se o número de artigos por data (Count(URL)), de modo a medir a popularidade diária do tema.
- Para garantir que dias sem publicações também aparecem, é criado um calendário de datas e usado um **Cross Joiner**, preenchendo dias em falta com 0 (popularidade nula).
- Depois disso, os dados são ordenados por data usando **Sorter**.

6.5.3 Normalização da Popularidade (Popularity Score)

Para comparar dias com mais ou menos publicações de forma relativa, é feita uma normalização da popularidade usando a fórmula:

(Publicações do dia – Média de publicações) / Desvio Padrão

- Esta fórmula é implementada no nó Math Formula, criando a coluna PopularityScore.
- Se o desvio padrão for zero (todos os dias com o mesmo número de publicações), o valor é definido como 0 para evitar divisões inválidas.

6.5.3 Classificação do nível de popularidade

Com base no valor de PopularityScore, o nó **Rule Engine** cria a coluna PopularityLevel, com as seguintes regras:

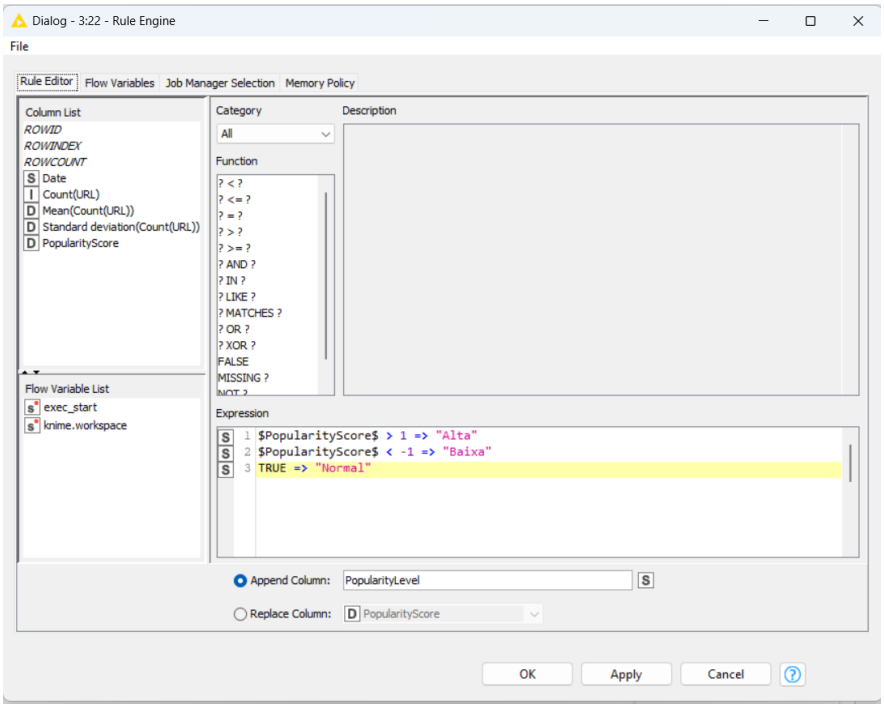


Figura 10 - Nó rule engine onde é criada a coluna PopularityLevel

6.5.4 Análise de tendência (Trend)

- Utilizado o nó **Lag Column** para comparar a popularidade de hoje com a do dia anterior.
- Criada a coluna Diff com a fórmula no **Math Formula**:

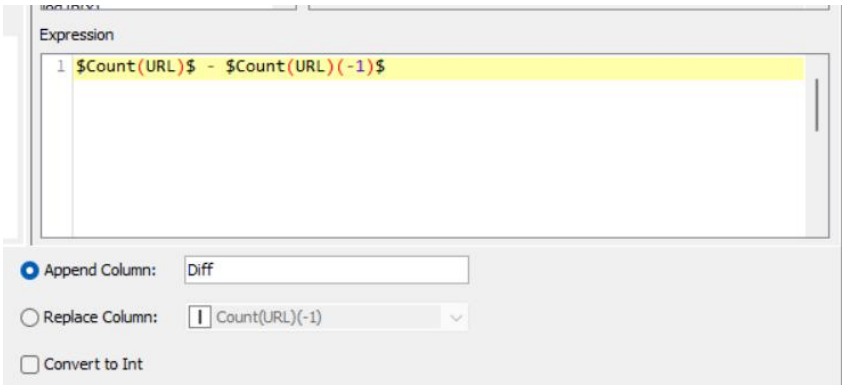


Figura 11 - Nó math formula onde é criada a coluna Diff

Com base nessa diferença, o nó **Rule Engine** atribui o campo Trend:

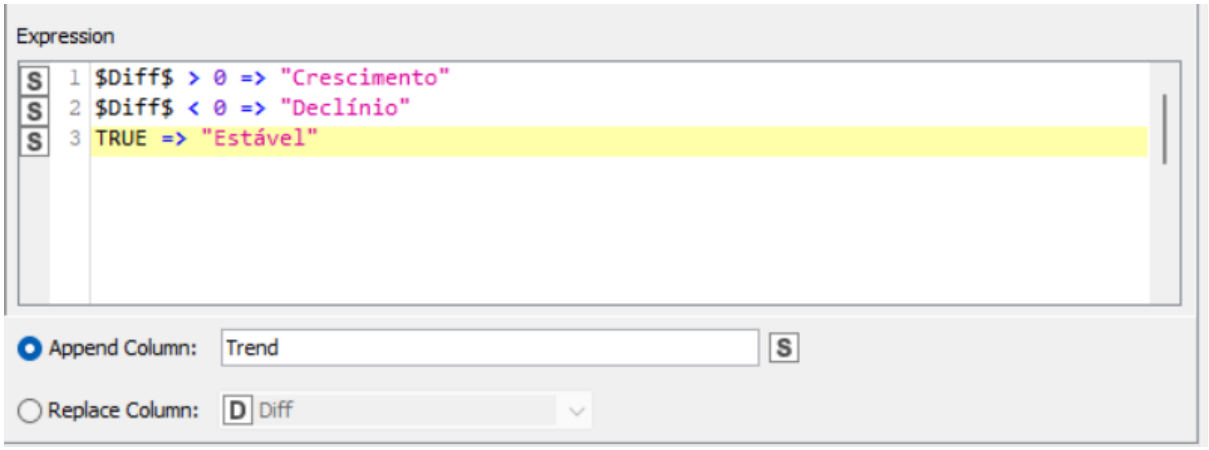


Figura 12 - Nó rule engine onde é definido o valor da coluna Trend

6.5.5 Cross Joiners e Column Filters

- Vários **Cross Joiner** são utilizados para combinar tabelas auxiliares (como datas ou valores agregados) com a tabela principal.
- Os **Column Filter** são aplicados várias vezes para remover colunas desnecessárias e manter apenas a informação relevante (data, contagem, popularidade, sentimento, URL, etc.). Isto reduz o tamanho da tabela e evita redundâncias na base de dados.

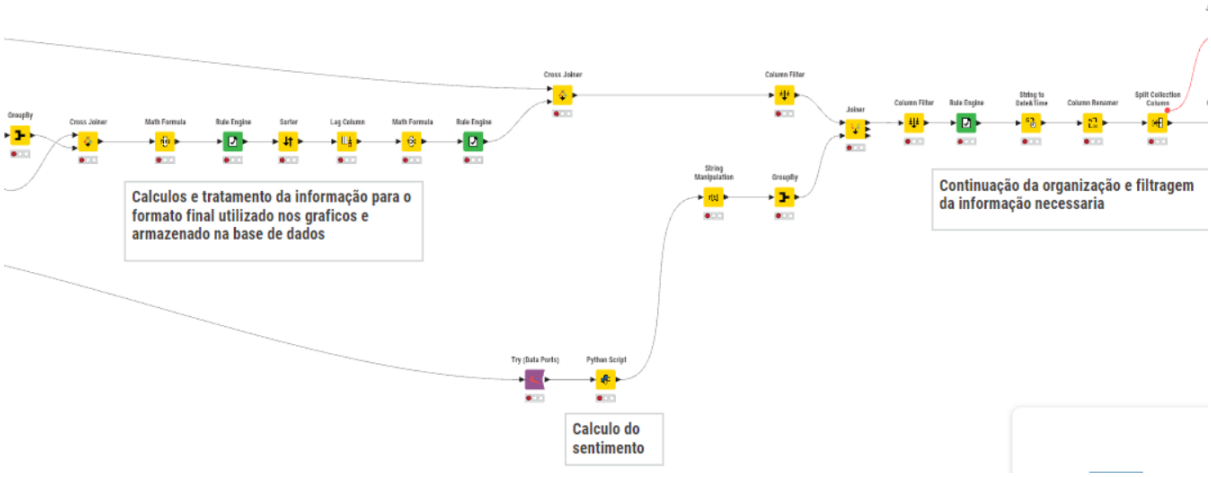


Figura 13 - Amostra da fase de limpeza, transformação e organização da informação no KNIME

6.6 Cálculo do Sentimento com Python (KNIME)

A análise de sentimento é a etapa responsável por identificar se o conteúdo das publicações recolhidas expressa uma perceção **positiva, negativa ou neutra** sobre o tema estudado. Esta análise é feita através de um **nó Python Script no KNIME**, que aplica técnicas de processamento de linguagem natural (NLP - *Natural Language Processing*).

6.6.1 Lógica de funcionamento do script

O script recebe uma tabela com as publicações (título e, quando existe, descrição), e executa os seguintes passos:

1. **Pré-processamento do texto**
    - Combina título e descrição na coluna TextoAnalise, pois usar apenas o título pode não ser suficientemente representativo.
    - Normaliza o texto para string limpa, removendo valores nulos ou não textuais.
2. **Tradução opcional do texto (PT - EN)**
    - A biblioteca TextBlob, utilizada para análise de sentimento, funciona melhor com texto em inglês.
    - Para garantir uma análise consistente, o texto é traduzido automaticamente para inglês usando GoogleTranslator (deep_translator).
    - Se a tradução falhar, o texto original é usado.

3. **Cálculo do sentimento (Polarity Score)**
    - A função TextBlob(...).sentiment.polarity devolve um **valor entre -1 e 1**:
        - Próximo de **1** → linguagem claramente positiva
        - Próximo de **-1** → linguagem negativa
        - Próximo de **0** → neutro / sem emoção relevante
    - Esta métrica é baseada em modelos de linguagem treinados com milhões de frases reais em inglês, analisando **palavras chave, contextos e composição linguística (adjetivos, advérbios, negações)**.
4. **Classificação do texto (rótulo final):**

Com base no resultado numérico (polaridade), é atribuído um rótulo no nó Rule Engine seguinte:

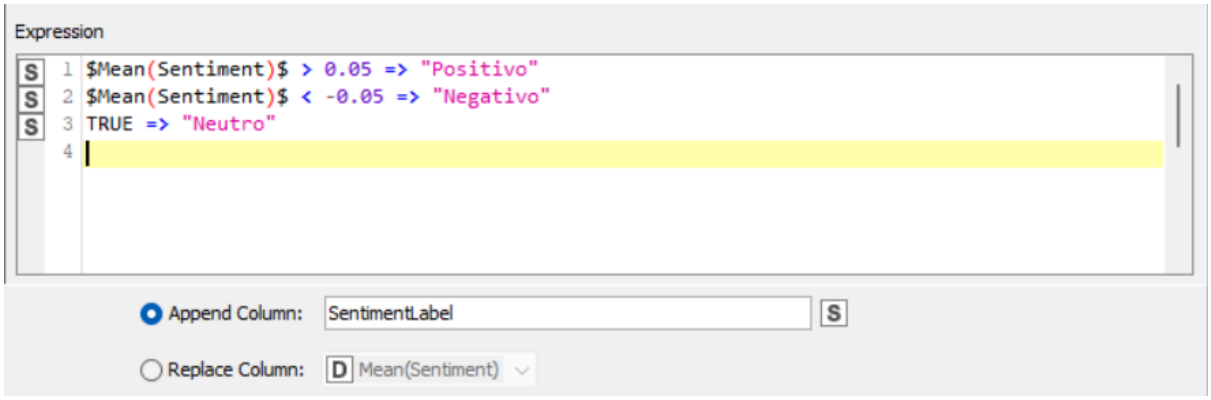


Figura 14 - Nó rule engine onde é atribuído o valor da coluna sentimentLabel

Estes limiares evitam que frases neutras ou ambíguas sejam classificadas incorretamente.

7. Inserção dos Dados na Base de Dados (Load – MongoDB)

Após a transformação completa dos dados (popularidade, tendências, sentimento médio e etiquetas), o workflow procede ao carregamento dos resultados para a base de dados MongoDB.

7.1 Estrutura dos dados finais a guardar

Cada registo inserido na coleção topic_trends contém, para cada dia analisado:

Tabela 1 - Campos a inserir na base de dados após todos os tratamentos e filtros

Campo	Descrição
execution_id	Número incremental da execução diária
topic	Tema pesquisado (ex: “crypto”)
date	Data do registo (YYYY-MM-DD)
RecordsByDay	Número total de artigos/publicações nesse dia
popularity_score	Score normalizado de popularidade
popularity_level	“Alta”, “Normal” ou “Baixa”
trend	“Crescimento”, “Declínio” ou “Estável”
avg_sentiment	Valor médio do sentimento (float)
sentiment_label	“Positivo”, “Neutro” ou “Negativo”

7.2 Inserção e registo de logs

Existem dois destinos no MongoDB:

topic_trends: Dados finais por dia, conforme tabela acima.

execution_logs: Informação da execução.

- O caminho **de sucesso** do workflow envia os dados corretamente processados para topic_trends e guarda um log com status = Success.


```
_id: ObjectId('68f4f0446b761c2b8827f1be')
topic : "crypto"
status : "Success"
exec_start : "2025-10-19T15:05:39.628"
exec_end : "2025-10-19T15:05:55.399"
duration : "15.771 seconds"
records_processed : 5
execution_id : 8
```


Figura 15 - Exemplo de log de sucesso

- Se ocorrer falha, o fluxo passa para o **Caminho de Insucesso**, onde é registado status = Failed com a mensagem de erro.


```
_id: ObjectId('68f4deee5c3e9362833fef1d')
_error_details : "Execute failed: Exception: Simulacao de erro no processamento"
_error_reason : "Execute failed: Exception: Simulacao de erro no processamento"
_error_node : "Python Script"
_error_caught : 1
exec_start : "2025-10-19T13:51:47.938"
status : "Failed"
exec_end : "2025-10-19T13:51:57.988"
duration : "10.05 seconds"
execution_id : 6
```


Figura 16 - Exemplo de log de insucesso (com simulação de erro no python script)

7.3 Conversão para JSON antes de inserir

O nó MongoDB Writer exige que os dados sejam enviados no formato JSON.

Por isso, imediatamente antes da escrita, é utilizado o nó Table to JSON, que converte cada linha da tabela final num documento JSON válido.

Exemplo do formato final:


```
_id: ObjectId('68f4f0446b761c2b8827f1c3')
execution_id : 8
topic : "crypto"
date : "2025-10-17"
RecordsByDay : 3
popularity_score : 1.7888543819998322
popularity_level : "Alta"
trend : "Crescimento"
avg_sentiment : 0.12061342592592594
sentiment_label : "Positivo"
```


Figura 17 - Exemplo de dados inseridos na base de dados depois de uma execução

8. Automação da Execução (Job em C#)

Para garantir que o processo ETL (extração de dados da API, transformação no KNIME e armazenamento na base de dados) é executado automaticamente todos os dias, foi desenvolvido um pequeno programa em **C#**, responsável por iniciar o workflow do KNIME de forma autónoma e registar logs da execução.

8.1 Objetivo do Job

- Executar o workflow do KNIME **uma vez por dia**, sem necessidade de intervenção manual.
- Garantir que, mesmo em caso de falha, o erro é registado para posterior análise.
- Manter um ficheiro de logs com o histórico de todas as execuções (data/hora, sucesso ou erro).

8.2 Funcionamento Geral

O programa realiza o seguinte ciclo:

1. Regista a hora de início da execução no ficheiro de log.
2. Inicia o KNIME em modo **batch** (linha de comandos), executando o workflow indicado.
3. Aguarda até o workflow terminar.
4. Regista se a execução terminou com sucesso ou se ocorreu algum erro.
5. Aguarda **24 horas** (Thread.Sleep(86400000)) e repete o processo.

8.3 Execução do KNIME por Linha de Comandos

O programa utiliza o processo:

knime.exe -nosplash -reset -nosave -application org.knime.product.KNIME_BATCH_APPLICATION -workflowDir="(...path...)"

Onde:

- -nosplash — inicia sem interface gráfica;
- -reset — reinicia os estados dos nós antes da execução;
- -nosave — evita alterações ao workflow;
- -application org.knime.product.KNIME_BATCH_APPLICATION — garante que o KNIME corre em modo batch;
- -workflowDir — caminho do workflow KNIME a executar automaticamente.

```
static void ExecutarKnime()
{
    string dataHora = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
    EscreverLog($"[{dataHora}] Iniciar execução do KNIME...");

    Process processo = new Process();
    processo.StartInfo.FileName = KNIME_PATH;

    // -nosave é adicionado para garantir saída limpa após execução
    processo.StartInfo.Arguments =
        $"-nosplash -reset -nosave -application org.knime.product.KNIME_BATCH_APPLICATION -workflowDir=\"{WORKFLOW_PATH}\"";

    processo.StartInfo.UseShellExecute = false;
    processo.StartInfo.RedirectStandardOutput = true;
    processo.StartInfo.RedirectStandardError = true;

    processo.Start();

    // Aguarda a conclusão sem limite de tempo
    processo.WaitForExit();

    if (processo.ExitCode == 0)
    {
        EscreverLog($"[{DateTime.Now}] KNIME terminou com sucesso.");
    }
    else
    {
        EscreverLog($"[{DateTime.Now}] Erro ao executar KNIME. Código: {processo.ExitCode}");
        string error = processo.StandardError.ReadToEnd();
        if (!string.IsNullOrWhiteSpace(error))
            EscreverLog($"Detalhes do erro: {error}");
    }
}
```

Figura 18 - Excerto do código utilizado

9. Gráficos gerados (KNIME)

Ainda dentro do KNIME é gerado um relatório com um conjunto de gráficos agregados em um componente KNIME e alimentados com os mesmos dados que são inseridos na base de dados.

Exemplos dos gráficos gerados:

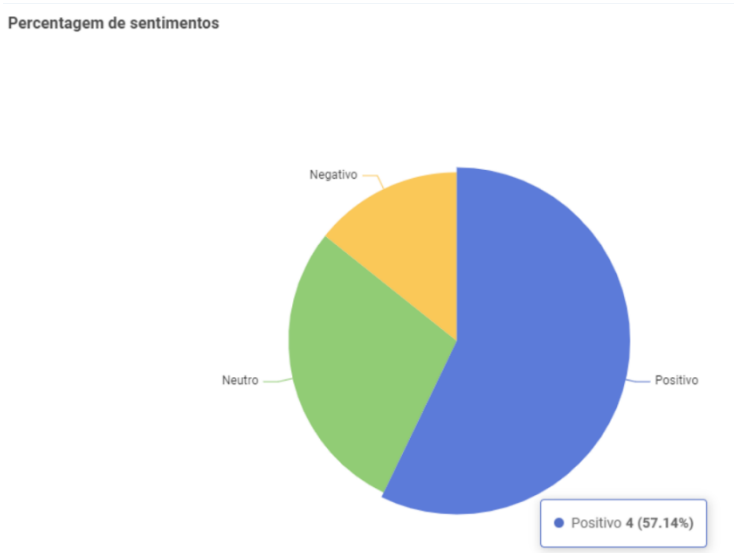


Figura 19 - Exemplo do gráfico de percentagem de sentimentos

Este gráfico permite perceber a distribuição da percentagem de sentimentos.

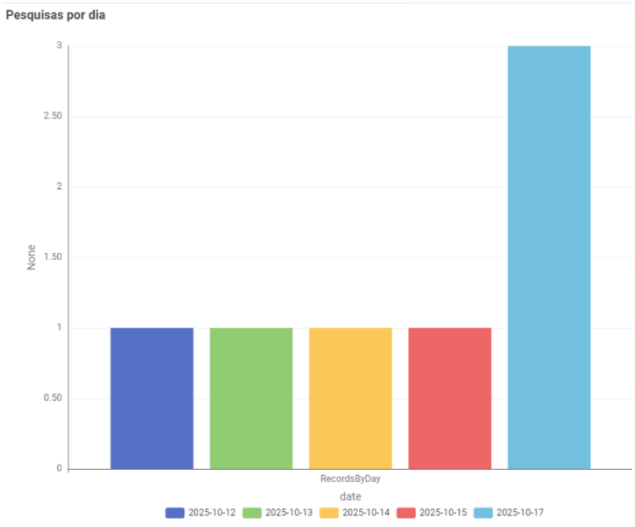


Figura 20 - Exemplo do gráfico de pesquisas por dia

O gráfico de barras permite perceber o quão mencionado é o tema nas pesquisas.

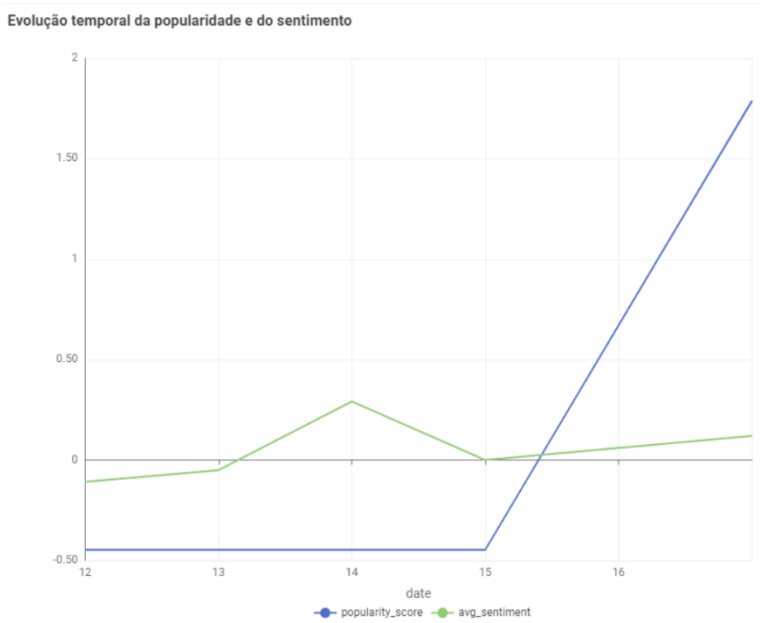


Figura 21 - Exemplo do gráfico da evolução temporal da popularidade e do sentimento

Também temos a possibilidade de perceber a evolução temporal do sentimento e da popularidade do tema e permite fazer relações entre os dois para determinadas conclusões.

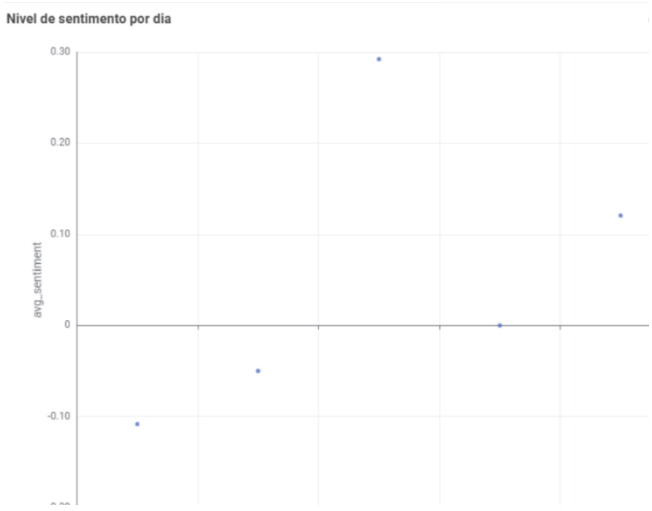


Figura 22 - Exemplo do gráfico de nível de sentimento por dia

Com o gráfico de nível de sentimento por dia podemos perceber mais ao detalhe o quão positivo ou negativo foi sentimento em cada dia oque pode ser útil no caso de eventos em dias específicos.

10. Conclusão

A realização deste projeto, desenvolvido no âmbito da unidade curricular de Integração de Sistemas de Informação da Licenciatura em Engenharia de Sistemas Informáticos do IPCA, permitiu aplicar de forma prática os conhecimentos adquiridos sobre processos de ETL, integração de fontes de dados heterogéneas, automação e armazenamento de informação. O principal objetivo consistiu no desenvolvimento de um sistema capaz de recolher automaticamente dados provenientes da NewsAPI e da Reddit API, processá-los num workflow de KNIME, calcular métricas de popularidade e sentimento e armazenar os resultados de forma estruturada numa base de dados MongoDB.

Este trabalho permitiu compreender, de forma concreta, os desafios inerentes à integração de sistemas distribuídos e à transformação de dados provenientes de fontes distintas, com formatos e estruturas diferentes. A agregação de dados de ambas as APIs revelou-se exigente, sobretudo pela necessidade de uniformizar a informação e evitar inconsistências entre os conteúdos recolhidos. Inicialmente, foi utilizada uma base de dados SQL Server; contudo, devido às limitações do KNIME e à natureza semiestruturada dos dados provenientes das APIs, tornou-se necessário migrar o sistema de armazenamento para MongoDB, o que se revelou uma solução mais adequada e flexível.

A fase de transformação e tratamento dos dados no KNIME constituiu uma das principais dificuldades do projeto, especialmente por se tratar de uma ferramenta que nunca tinha utilizado anteriormente. A construção do workflow exigiu investigação, experimentação e um entendimento progressivo das lógicas de manipulação, agrupamento, cálculos e utilização de scripts Python para o processamento do sentimento. Ainda assim, foi possível implementar um processo funcional e automatizado, capaz de processar dados diariamente e registar logs de sucesso ou insucesso através de um job desenvolvido em C#.

No final, os objetivos iniciais foram alcançados: foi construído um sistema completo de ETL, desde a extração até ao armazenamento final, com cálculo diário de tendências, popularidade e sentimento para um determinado tema. Este projeto permitiu consolidar competências técnicas essenciais na área da integração de sistemas, bem como reforçar capacidades de resolução de problemas, autonomia e adaptação tecnológica.

11. Trabalhos Futuros

Embora o sistema desenvolvido se encontre funcional e cumpra os requisitos definidos, existem várias melhorias que poderão ser implementadas para ampliar a sua utilidade e robustez. Uma das principais evoluções será a integração com ferramentas de visualização como o *Power BI* ou *Grafana*, permitindo gerar *dashboards* dinâmicos com base no histórico da base de dados MongoDB. Esta integração possibilitaria a criação de relatórios visuais, facilitando a análise longitudinal da popularidade e do sentimento de diferentes temas ao longo do tempo.

Outra melhoria relevante consiste no aperfeiçoamento do mecanismo de automação. Atualmente, a execução do workflow KNIME é realizada através de um script em C# que corre num ciclo contínuo. Futuramente, este processo poderá ser substituído por tarefas agendadas no sistema operativo, serviços em *cloud*, Docker ou mesmo uma abordagem serverless, garantindo maior estabilidade, escalabilidade e monitorização.

A estrutura da API encontra-se preparada para ser expandida com novas fontes de dados, como *Twitter, Google Trends, YouTube* ou outras plataformas de opinião pública, o que permitirá um aumento significativo da representatividade e da qualidade dos resultados. Poderá também ser incorporada uma camada de *machine learning* para prever tendências futuras ou detetar alterações súbitas de sentimento.

Apesar das dificuldades iniciais na utilização do KNIME e na adaptação de dados provenientes de fontes distintas, o desenvolvimento deste projeto constituiu uma experiência enriquecedora, permitindo consolidar conhecimentos teóricos da unidade curricular e aplicar, de forma concreta, conceitos de integração, transformação e automatização de dados, essenciais para a engenharia de sistemas de informação.

12. QR Code para vídeo de demonstração

Link original do vídeo:

https://youtu.be/NIS6dQz5FXQ

13. Referências Bibliográficas

NewsAPI. Documentação oficial. Disponível em: https://newsapi.org/docs. Acedido em: **18 de outubro de 2025**.

Reddit Inc. Reddit API Documentation. Disponível em: https://www.reddit.com/dev/api. Acedido em: **17 de outubro de 2025**.

TextBlob. TextBlob: Simplified Text Processing. Disponível em: https://textblob.readthedocs.io/en/latest/. Acedido em: **15 de outubro de 2025**.

Deep Translator. Biblioteca Python. Disponível em: https://pypi.org/project/deep-translator/. Acedido em: **14 de outubro de 2025**.

KNIME. KNIME Analytics Platform Documentation. Disponível em: https://docs.knime.com/. Acedido em: **19 de outubro de 2025**.

KNIME. Python Integration Installation Guide. Disponível em: https://docs.knime.com/latest/python_installation_guide/. Acedido em: **16 de outubro de 2025**.

MongoDB Inc. MongoDB Manual. Disponível em: https://www.mongodb.com/docs/. Acedido em: **18 de outubro de 2025**.

MongoDB Inc. MongoDB Compass. Disponível em: https://www.mongodb.com/products/compass. Acedido em: **19 de outubro de 2025**.

Microsoft. ASP.NET Core Web API Documentation. Disponível em: https://learn.microsoft.com/en-us/aspnet/core/web-api. Acedido em: **17 de outubro de 2025**.

Microsoft. System.Diagnostics.Process Class. Disponível em: https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process. Acedido em: **14 de outubro de 2025**.

KNIME. JSON Path / JSON to Table. Disponível em: https://docs.knime.com/latest/analytics_platform_user_guide/index.html#json. Acedido em: **15 de outubro de 2025**.