

Relatório de Integração de Sistemas de Informação



Unidade Curricular: Integração de Sistemas de Informação

Docente: Luis Gonzaga Martins Ferreira

Trabalho realizado por:

Joel Faria, a28001

Escola de Tecnologia

1ºSemestre 2025/26

Índice

Introdução.....	4
Problema Proposto	4
Arquitetura da Solução e Tecnologias Utilizadas	5
Geração de Dados "Sujos" em C#	6
Objetivos da Geração de Dados	6
Funcionamento da App	7
Geração de Inconsistências.....	8
Processos ETL no Knime	9
Estrutura do Workflow KNIME	9
Nós Utilizados no Workflow	10
Extração	10
Transformação	10
Carregamento	10
Expressões Regex Utilizadas	11
warehouse_id.....	11
sku	12
quantity_available.....	13
location.....	14
last_updated	15
Notificação Automática de Stock Baixo	16
Armazenamento dos Dados Limpos em SQL Server	17
Backend C# (ASP.NET Core).....	18
Frontend React	19
Vídeo de Demonstração.....	20
Conclusão	21
Bibliografia	22

Índice figuras

Figura 1 - Função Generate	7
Figura 2 - Função AddSpaces	8
Figura 3 - Adicionar dados a BD	17
Figura 4 - QR Code com Vídeo de Demonstração	20

Introdução

Este relatório apresenta o trabalho prático desenvolvido na Unidade Curricular (UC) de Integração de Sistemas de Informação (ISI), que faz parte do 1º semestre do 2º ano do curso de Engenharia de Sistemas Informáticos.

Este texto expõe o trabalho desenvolvido para a primeira atividade prática da disciplina, na qual se sugeriu a fusão de dados entre sistemas diversos, abrangendo desde a criação de dados incoerentes até a sua exibição por meio de um painel web interativo.

Neste documento, será possível examinar a estrutura da solução implementada, uma explicação minuciosa dos processos de ETL criados no KNIME Analytics Platform, a implementação de uma API REST em ASP.NET Core para apresentação de dados normalizados e a criação de um frontend em React para exibição de métricas de negócios.

Problema Proposto

Neste trabalho, pretende-se aplicar conceitos fundamentais de integração de sistemas e processos ETL (Extract, Transform, Load) para resolver um problema real de gestão de dados empresariais, relacionando geração de dados, normalização, APIs RESTful e visualização.

O objetivo é desenvolver uma solução completa capaz de processar dados "sujos" provenientes de um sistema de e-commerce (vendas e inventário de armazém), aplicar transformações de limpeza e padronização através de workflows KNIME, expor os dados normalizados através de endpoints REST documentados e apresentar KPIs de negócio num dashboard web responsivo.

Arquitetura da Solução e Tecnologias Utilizadas

O conteúdo a seguir apresenta a estrutura geral da solução desenvolvida, juntamente com as principais tecnologias utilizadas na implementação do projeto. A proposta centra-se na unificação de informações provenientes de diferentes sistemas, com destaque para a produção, padronização, apresentação e exibição dos dados, em termos de arquitetura, a solução é formada por quatro camadas essenciais:

Geração de Dados: Executada com um aplicativo C# que simula dados de vendas e estoque com falhas comuns em sistemas reais.

Normalização de Dados: Realizada na KNIME Analytics Platform, onde são criados workflows para purificação, alteração e uniformização dos dados não tratados.

API REST: Criada com ASP.NET Core, esta camada oferece endpoints para a consulta de dados normalizados, facilitando a integração com outras aplicações.

Frontend Web: Desenvolvido em React, o painel oferece uma interface dinâmica com KPIs, gráficos e tabelas para a análise dos dados de vendas e estoque.

As tecnologias escolhidas foram minuciosamente selecionadas para otimizar a interoperabilidade, a performance e a simplicidade de manutenção. A utilização do KNIME possibilita fluxos de trabalho visuais e reaproveitáveis para ETL.

Geração de Dados "Sujos" em C#

A secção seguinte descreve a primeira fase do projeto, dedicada ao desenvolvimento de uma aplicação em C# destinada à geração de dados com inconsistências intencionais. O objetivo consistiu em simular cenários reais em que sistemas legados ou fontes externas produzem informações que requerem normalização antes de serem utilizadas.

Objetivos da Geração de Dados

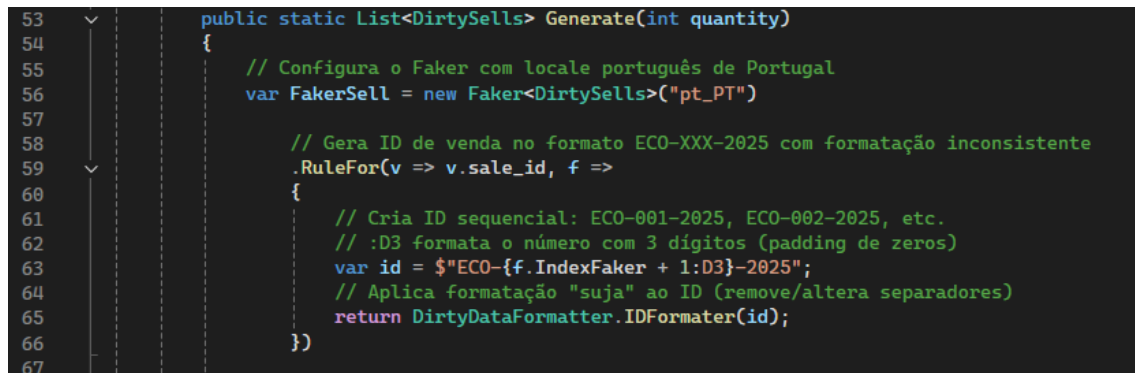
A aplicação desenvolvida teve como principal propósito criar conjuntos de dados de vendas (sales) e inventário de armazém (warehouse stock) que refletissem problemas comuns encontrados em ambientes empresariais, tais como:

- Formatos inconsistentes de identificadores (com e sem prefixos, maiúsculas/minúsculas)
- Valores numéricos com diferentes separadores decimais (vírgula vs ponto)
- Espaços em branco desnecessários
- Campos de data em formatos variados
- Dados duplicados ou com pequenas variações

Funcionamento da App

A aplicação C# utiliza a biblioteca **Bogus**, uma ferramenta poderosa para gerar dados falsos realistas de forma automática. Aqui está como funciona:

Cria-se um `Faker<T>` para cada tipo de dados. Por exemplo:



```
53 public static List<DirtySells> Generate(int quantity)
54 {
55     // Configura o Faker com locale português de Portugal
56     var FakerSell = new Faker<DirtySells>("pt_PT")
57
58     // Gera ID de venda no formato ECO-XXX-2025 com formatação inconsistente
59     .RuleFor(v => v.sale_id, f =>
60     {
61         // Cria ID sequencial: ECO-001-2025, ECO-002-2025, etc.
62         // :D3 formata o número com 3 dígitos (padding de zeros)
63         var id = $"ECO-{f.IndexFaker + 1:D3}-2025";
64         // Aplica formatação "suja" ao ID (remove/altera separadores)
65         return DirtyDataFormatter.IDFormatter(id);
66     })
67 }
```

Figura 1 - Função Generate

Como pudemos ver na linha 56, cria-se um `Faker` para gerar dados com inconsistências através de regras definidas pelo `.RuleFor()`.

Na linha 59, define-se a geração do `sale_id`. A linha 63 usa interpolação de strings com `f.IndexFaker` (fornece índices sequenciais) formatado com 3 dígitos (`1:D3`), gerando IDs como "ECO-001", "ECO-002", etc.

A linha 65 é crucial: `DirtyDataFormatter.IDFormat(id)` recebe o ID limpo e aplica transformações aleatórias (remove hífen, altera capitalização, adiciona underscores), simulando diferentes sistemas com formatos distintos. Este padrão repete-se para todos os campos, permitindo gerar milhares de registos com inconsistências controladas.

Geração de Inconsistências

Após a geração base dos dados com Bogus, a aplicação aplica **transformações controladas** através da classe `DirtyDataFormatter` para introduzir inconsistências realistas que simulam cenários de integração de sistemas heterogéneos, como por exemplo:

Função `AddSpaces()`

Esta função adiciona espaços em branco indesejados aos campos de texto. O processo funciona assim:

Gera um número aleatório (0 a 3) para decidir o tipo de inconsistência

Aplica transformações usando um switch:

Adiciona espaços no início: " Laptop HP"

Adiciona espaços no fim: "Laptop HP "

Adiciona em ambos os lados: " Laptop HP "

Mantém sem alteração: "Laptop HP"

```
62 // Referências
63 public static string AddSpaces(string text)
64 {
65     // Gera um número aleatório entre 0 e 3 para decidir onde adicionar espaços
66     var op = random.Next(0, 4);
67
68     return op switch
69     {
70         0 => $" {text}",           // Adiciona 2 espaços no início
71         1 => $"{text} ",          // Adiciona 2 espaços no fim
72         2 => $" {text} ",         // Adiciona 2 espaços em ambos os lados
73         _ => text                 // Retorna o texto sem modificação
74     };
75 }
```

Figura 2 - Função `AddSpaces`

Processos ETL no Knime

O desenvolvimento de workflows ETL (Extract, Transform, Load) foi realizado na plataforma **KNIME Analytics Platform** com o objetivo de normalizar os dados “sujos” gerados pela aplicação em C#. A escolha do KNIME deveu-se à sua interface visual intuitiva, à vasta biblioteca de nós de processamento disponíveis e à capacidade de criar pipelines de dados reutilizáveis, eficientes e devidamente documentados.

Estrutura do Workflow KNIME

O processo ETL desenvolvido segue uma estrutura lógica dividida em três fases principais:

Extract (Extração): Leitura dos ficheiros CSV, JSON e XML gerados pela aplicação C# através de nós específicos como CSV Reader, JSON Reader e XML Reader. Esta fase assegura que os dados são corretamente importados para o ambiente KNIME

Transform (Transformação): Aplicação de múltiplos nós de manipulação de dados para normalizar as inconsistências. Esta é a fase mais complexa, envolvendo operações como remoção de espaços em branco (String Manipulation), padronização de IDs com expressões regulares (Regex), conversão de separadores decimais (String Replacer)

Load (Carregamento): Exportação dos dados normalizados para formatos limpos (CSV, JSON, XML) prontos para integração com a API REST ou análise posterior.

Nós Utilizados no Workflow

O workflow implementa um processo ETL completo dividido em três fases:

Extração

- **CSV Reader / JSON Reader / XML Reader:** Importam os dados sujeitos nos três formatos gerados pela aplicação C#

Transformação

- **String Manipulation:** Normaliza texto (remove espaços, padroniza capitalização, aplica regex)
- **String to Number:** Converte valores de texto para numéricos, tratando separadores decimais
- **Concatenate:** Combina dados de múltiplas fontes num único fluxo
- **JSON to Table / Table to JSON:** Converte entre formatos hierárquicos e tabulares
- **XML to JSON / JSON to XML:** Realiza conversões entre formatos XML e JSON
- **XPath:** Extrai valores específicos de documentos XML

Carregamento

- **CSV Writer:** Exporta dados normalizados para CSV limpo
- **JSON Row Combiner and Writer:** Agrupa e grava dados em formato JSON estruturado
- **XML Row Combiner and Write:** Exporta dados para XML quando necessário

O workflow demonstra modularidade: cada formato passa por pipelines específicos de normalização antes de serem combinados e exportados nos formatos finais limpos.

Expressões Regex Utilizadas

warehouse_id

Usou-se a seguinte expressão para padronizar os separadores nos IDs de armazém, convertendo todas as variações para um formato único com hífen.

```
regexReplace($warehouse_id$, "[_\\s]", "-")
```

Esta expressão regex é utilizada no nó **String Manipulation** para normalizar identificadores de armazéns (warehouse_id) que apresentam formatos inconsistentes.

Decomposição da Expressão:

- **regexReplace()**: Função do KNIME que substitui padrões encontrados através de expressões regulares
- **\$warehouse_id\$**: Referência à coluna que contém os IDs de armazém
- **"[_\\s]"**: Padrão regex a procurar:
 - **_**: Caracter underscore
 - **\\s**: Qualquer espaço em branco (espaço, tab, etc.)
 - **[]**: Classe de caracteres - encontra **qualquer** um dos caracteres dentro dos parênteses retos
- **"-"**: String de substituição - todos os matches são substituídos por hífen

sku

Este regex garante que SKUs com diferentes formatos de entrada (usando espaços, underscores, slashes ou hífen) sejam todos convertidos para um **formato único e consistente** com hífen como separadores.

```
regexReplace($sku$, "([A-Z]+)[\\s/_-]*([A-Z]+)[\\s/_-]*(\\d{3})[\\s/_-]*(\\d{4})", "$1-$2-$3-$4")
```

Decomposição do Padrão de Captura

`([A-Z]+)` - Captura uma ou mais letras maiúsculas (prefixo do SKU)

`[\\s/_-]*`: Encontra zero ou mais separadores inconsistentes (espaços, underscores, slashes ou hífen)

`([A-Z]+)` - Captura um segundo grupo de letras maiúsculas (categoria ou tipo de produto)

`[\\s/_-]*`: Novamente, separadores inconsistentes

`(\\d{3})` - Captura exatamente 3 dígitos (código numérico)

`[\\s/_-]*`: Separadores inconsistentes

`(\\d{4})` - Captura exatamente 4 dígitos (número sequencial)

String de Substituição: "\$1-\$2-\$3-\$4"

Os grupos capturados são reorganizados com **hífen como separador único**, criando um formato padronizado.

Exemplos de Transformação

- **Input:** "SKU/LAPTOP/123/4567" → **Output:** "SKU-LAPTOP-123-4567"

quantity_available

Remove caracteres indesejados de campos numéricos que foram contaminados com texto, espaços, símbolos ou formatação inconsistente, mantendo **apenas os dígitos**.

```
regexReplace($quantity_available$, "[^0-9]", "")
```

Decomposição da Expressão

- \$quantity_available\$: Coluna que contém valores de quantidade disponível
 - "[^0-9]": Padrão de busca usando classe de caracteres negada:
 - []: Define uma classe de caracteres
 - ^: Dentro de parênteses retos, significa NEGAÇÃO
 - 0-9: Intervalo de todos os dígitos de 0 a 9
 - Significado completo: Encontra qualquer carácter que NÃO seja um dígito
- "": String vazia - remove os caracteres encontrados (substitui por nada)

Exemplos de Transformação

- **Input:** "150 units" → **Output:** "150"
- **Input:** "Qty: 250" → **Output:** "250"

location

lowerCase(\$location\$)

Esta é uma **função simples de manipulação de strings** do KNIME utilizada para padronizar a capitalização de campos de texto

Componentes da Função

lowerCase(): Função do KNIME String Manipulation que converte todos os caracteres alfabéticos para minúsculas

\$location\$: Referência à coluna que contém dados de localização (cidade, país, região, etc.)

Exemplos de Transformação

- **Input:** "Lisboa" → **Output:** "lisboa"
- **Input:** "PORTO" → **Output:** "porto"

last_updated

A expressão utiliza **três níveis de regexReplace() encadeados** que processam o campo \$last_updated\$ de dentro para fora.

Remoção de Timestamps

```
regexReplace($last_updated$, "([\\d/-]+)[\\sT].*", "$1")
```

Pattern: ([\\d/-]+) captura dígitos, slashes e hífen (a data), seguido de [\\sT].* (espaço ou "T" e qualquer coisa depois)

Conversão ISO para DD/MM/YYYY

```
regexReplace(..., "^(\\d{4})[-/](\\d{2})[-/](\\d{2})$", "$3/$2/$1")
```

Pattern: Captura 4 dígitos (ano), 2 dígitos (mês), 2 dígitos (dia) separados por hífen ou slash

Substituição: "\$3/\$2/\$1" reorganiza para dia/mês/ano com slashes

Normalização de Separadores

```
regexReplace(..., "^(\\d{2})-(\\d{2})-(\\d{4})$", "$1/$2/$3")
```

Objetivo: Converte hífen para slashes em datas DD-MM-YYYY

Pattern: Captura formato DD-MM-YYYY com hífen

Substituição: Mantém a ordem mas troca hífen por slashes

Notificação Automática de Stock Baixo

Após a conclusão do processo ETL e a normalização dos dados de warehouse stock, o workflow KNIME implementa uma **funcionalidade de alerta automático** para monitorização de níveis de inventário críticos.

Implementação do Sistema de Alertas

Filtragem de Dados: Utiliza-se o nó **Rule engine** para identificar produtos cujo `quantity_available` está abaixo do `minimum_level` definido no sistema.

Preparação da Mensagem: Os registos filtrados (produtos com stock baixo) são formatados numa tabela que inclui informações críticas como SKU do produto, nome, quantidade disponível, nível mínimo e localização do armazém.

Envio de Email: O nó **Send Email** do KNIME é configurado com:

- Servidor SMTP (ex: Gmail, Outlook corporativo)
- Credenciais de autenticação
- Destinatários (equipa de gestão de inventário, responsáveis de armazém)
- Corpo do email com a lista de produtos que necessitam reposição

Armazenamento dos Dados Limpos em SQL Server

O workflow utiliza as seguintes etapas principais:

- **CSV Reader:** Lê os ficheiros de vendas e stock já normalizados.
- **Duplicate Row Filter:** Remove linhas duplicadas para garantir integridade dos dados antes de guardar.
- **Microsoft SQL Server Connector:** Estabelece ligação à base de dados SQL Server.
- **DB Writer:** Insere os registos limpos diretamente nas respetivas tabelas do SQL Server (vendas e stock).

Este passo é fundamental porque:

- Garante que os dados utilizados pelo backend/API estão **padronizados** e livres de duplicados
- Permite consultas rápidas, integrações e persistência fiável para aplicações futuras (ex: dashboards React, sistemas de gestão)

O diagrama ilustra dois workflows independentes: um para vendas e outro para stock, ambos seguindo o ciclo de leitura dos dados, filtragem de duplicados e escrita direta na base de dados SQL Server. Desta forma, todos os dados normalizados já estão disponíveis para serem consumidos pela API desenvolvida em C#.

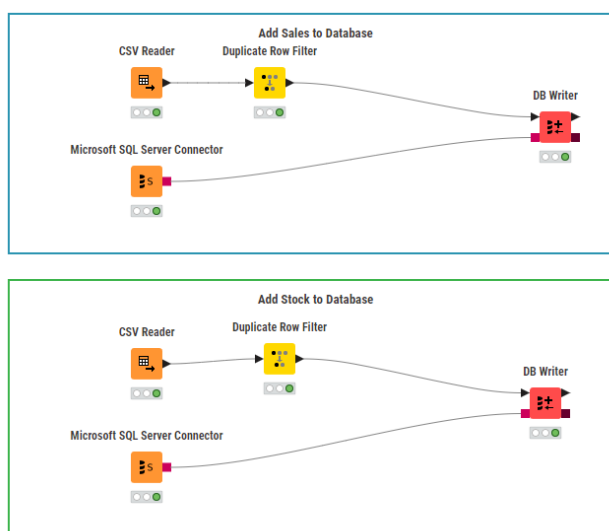


Figura 3 - Adicionar dados a BD

Backend C# (ASP.NET Core)

Após a carga dos dados limpos na base de dados SQL Server, o backend em C# entra em ação para disponibilizar estes dados de forma controlada e segura para o frontend e para outros sistemas.

Como funciona o backend em C# (ASP.NET Core):

- **Conexão à base de dados:** O backend (API) está configurado para ligar-se ao mesmo SQL Server onde os dados limpos foram guardados pelo KNIME. Utiliza bibliotecas populares como Entity Framework ou Dapper para mapear tabelas SQL para objetos C# e facilitar consultas e operações de escrita.
- **API RESTful:** O backend expõe endpoints HTTP (por exemplo, /api/sales, /api/warehouse-stock, /api/low-stock) que permitem a outras aplicações obterem dados, pesquisarem por produtos, consultarem o stock disponível, ou até mesmo receberem alertas de stock baixo. Todos esses endpoints seguem práticas RESTful para facilitar a integração.

Deste modo, o backend serve de ponte entre a base de dados SQL Server e as aplicações consumidoras (frontend, apps móveis, integrações externas), centralizando todas as regras de negócio e exposições seguras dos dados normalizados.

Frontend React

Após a exposição dos dados limpos pelo backend em C#, o frontend da aplicação é desenvolvido em **React**, utilizando Javascript para criar uma interface de utilizador dinâmica e altamente responsiva.

Objetivo do Frontend em React

O frontend consome a API criada em ASP.NET Core utilizando bibliotecas como axios ou o próprio fetch, e apresenta as informações de vendas, stocks, produtos e alertas numa interface web acessível aos utilizadores — como gestores ou operadores de armazém.

Componentes e Funcionalidades Principais

- **Dashboard:** Exibe KPIs, resumos de vendas, níveis de stock e notificações de stock baixo em tempo real.
- **Tabelas Dinâmicas:** Listagem de vendas, stocks e produtos com filtros, ordenação e paginação.
- **Pesquisa e Filtros:** Permitem ao utilizador consultar rapidamente informação por nome, código ou intervalo de datas.
- **Alertas Visuais:** Itens com stock abaixo do mínimo são destacados, facilitando ações rápidas.
- **Integração com API:** Utiliza Axios/Fetch para obter dados do backend, atualizar registos e, quando permitido, adicionar ou editar produtos.

Vídeo de Demonstração



Figura 4 - QR Code com Vídeo de Demonstração

Conclusão

A solução aplicada abrange todo o ciclo de vida dos dados de vendas e estoque de armazém, utilizando diversas tecnologias para garantir eficiência operacional e confiabilidade na administração de inventário. O procedimento começa com a purificação e padronização dos dados no KNIME, assegurando a qualidade e a integridade das informações por meio de filtros e transformações precisas.

Esses dados normalizados são guardados em uma base de dados SQL Server, oferecendo uma infraestrutura robusta e escalável para consulta e armazenamento. O backend criado em C# (ASP.NET Core) oferece dados e funcionalidades essenciais através de uma API REST segura e estruturada, tornando a integração com sistemas externos e outras aplicações mais simples. Por fim, o frontend React proporciona uma experiência de usuário ágil e intuitiva, permitindo que a equipe de gestão acompanhe métricas de vendas, status de estoque e receba notificações automáticas em tempo real.

Essa integração de processos oferece visibilidade, automação e rapidez, diminuindo erros manuais, apoiando a decisão e aprimorando toda a cadeia de suprimentos do armazém

Bibliografia

- PowerPoints de Luis Ferreira
- Stack Overflow
- KNIME Documentation: <https://docs.knime.com>
- Microsoft SQL Server Documentation: <https://learn.microsoft.com/pt-br/sql/>
- ASP.NET Core Documentation: <https://learn.microsoft.com/pt-br/aspnet/core/>
- React Documentation: <https://react.dev>