Politécnico do Cávado e Ave
Engenharia de Sistemas Informáticos
Integração de Sistemas de Informação

~

Ano Académico 2025–2026

Sistema ETL para Análise de Dados Financeiros

Professor
Prof. Óscar Ribeiro

Estudante
Gabriel Araújo
27978

19 de outubro de 2025

Conteúdo

Lista de Figuras **5**

Lista de Tabelas **6**

Lista de Códigos **7**

Glossário **7**

Lista de Nomenclatura **7**

1 Resumo **8**

2 Enquadramento **9**

2.1 Contexto 9

2.2 Motivação 9

2.3 Âmbito 9

2.4 Limitações 10

3 Problema **11**

3.1 Descrição do Problema 11

3.2 Requisitos Funcionais 11

3.3 Requisitos Não-Funcionais 12

3.4 Arquitetura Proposta 12

4 Estratégia Utilizada **13**

4.1 Escolha das Tecnologias 13

4.1.1 Pentaho Data Integration (Kettle 10.2) 13

4.1.2 PostgreSQL 16 13

4.1.3 Node-RED 13

4.1.4 Alpha Vantage API 14

4.2 Modelação de Dados 14

4.3 Operações ETL 15

5 Transformações **16**

5.1 Transformação 1: Extract from Alpha Vantage API 16

5.1.1 Objetivo 16

5.1.2 Diagrama 16

5.1.3 Steps 17

5.1.4 Configurações Críticas 17

5.1.5 Output 17

5.1.6 Desafio 17

5.2 Transformação 2: Clean and Normalize 18

5.2.1 Objetivo 18

5.2.2 Diagrama 18

5.2.3 Steps 19

5.2.4 Regex Validation 19

5.2.5 Cálculo Daily Return 19

5.2.6 Output 20

5.3 Transformação 3: Calculate Technical Indicators 21

5.3.1 Objetivo 21

5.3.2 Diagrama 21
5.3.3 Steps 21
5.3.4 Algoritmos Implementados 21
5.3.5 Output 22
5.4 Transformação 4: Load to PostgreSQL 23
5.4.1 Objetivo 23
5.4.2 Diagrama 23
5.4.3 Steps 23
5.4.4 Row Normaliser 23

6 Jobs **24**

6.1 Master ETL Pipeline Job 24
6.1.1 Objetivo 24
6.1.2 Diagrama 24
6.1.3 Entries 25
6.1.4 Controlo de Fluxo 25
6.1.5 Variáveis 25
6.1.6 Logging 25
6.1.7 Notificações por Email 26

7 Dashboard e Visualização **27**

7.1 Arquitetura 27
7.2 Componentes do Dashboard 27
7.2.1 Price History 27
7.2.2 RSI Indicators 28
7.2.3 Comparative Table 29
7.3 Fluxos Node-RED Implementados 31

7.4 Design System 31
7.5 Screenshots 31

8 Dificuldades e Lições Aprendidas 34

8.1 Desafios Técnicos 34
8.1.1 JSON Parsing da Alpha Vantage 34
8.1.2 Cálculo de Daily Return 34
8.1.3 RSI Não Aparecia no PostgreSQL 35
8.2 Desafios Conceptuais 35
8.2.1 Agregações em ETL vs BI Layer 35
8.2.2 Normalização da Tabela technical_indicators 35
8.3 Ferramentas e Ambiente 35
8.3.1 Curva de Aprendizagem Pentaho 35
8.4 Gestão de Tempo 36
8.5 Reflexão 36

9 Conclusão e Trabalhos Futuros 37

9.1 Síntese do Trabalho 37
9.2 Contribuições 37
9.3 Limitações 38
9.4 Trabalhos Futuros 38
9.4.1 Curto Prazo 38
9.4.2 Longo Prazo (projetos futuros) 38
9.5 Reflexão Final 39

Bibliografia 40

Lista de Figuras

3.1 Arquitetura em 4 Camadas 12

4.1 Diagrama Entidade-Relacionamento 14

5.1 Transformação 1 - Fluxo Completo 16

5.2 Transformação 2 - Fluxo Completo 18

5.3 Transformação 3 - Fluxo Completo 21

5.4 Transformação 4 - Fluxo Completo 23

6.1 Job Master ETL Pipeline - Fluxo Completo 24

7.1 Dashboard - Página Overview Completa 31

7.2 Componente Price History com Seletor de Período 32

7.3 RSI Indicators - Cards Responsivos 32

7.4 Comparative Table com Funcionalidades de Ordenação 32

7.5 Node-RED Flow: Componente Price History com Seletor de Período . 33

7.6 Node-RED Flow: RSI Indicators - Cards Responsivos 33

7.7 Node-RED Flow: Comparative Table com Funcionalidades de Ordenação 33

Lista de Tabelas

3.1 Requisitos Funcionais 12

4.1 Operações por Transformação 15

5.1 Steps da Transformação 1 17

5.2 Steps da Transformação 2 19

5.3 Steps da Transformação 3 21

5.4 Steps da Transformação 4 23

6.1 Entries do Job 25

7.1 Fluxos Node-RED do Dashboard 31

Listings

5.1 Cálculo Daily Return 19
7.1 Price History Query 28
7.2 RSI Indicators Query 29
7.3 Comparative Table Query 30

Capítulo 1

Resumo

A análise de dados financeiros de mercados de ações requer sistemas de extração, transformação e carregamento (ETL), capazes de processar grandes volumes de dados em tempo real.

Este trabalho apresenta o desenvolvimento de um sistema ETL que extrai dados financeiros via API da Alpha Vantage, calcula indicadores técnicos avançados e disponibiliza visualizações interativas através de dashboard web.

O pipeline ETL foi implementado em Pentaho Data Integration (Kettle 10.2), compreendendo 4 transformações principais: extração dos dados via REST API com parsing de JSON, validação e normalização dos dados com expressões regulares, cálculo de indicadores técnicos (SMA, EMA, MACD, RSI, Bollinger Bands) e carregamento numa base de dados PostgreSQL. Um job orquestrador gere o fluxo completo com tratamento de erros e logging de auditoria. Os dados processados são disponibilizados através de dashboard interativo desenvolvido em NODE-RED.

O sistema demonstrou capacidade de processar uma quantidade significante de registos de cotações diárias num curto espaço de tempo. Com estes registos foram calculados vários indicadores técnicos, estes armazenados de forma estruturada. O dashboard apresenta gráficos de evolução histórica, gauges de indicadores, e cards informativos.

O trabalho demonstra a aplicação prática de conceitos de integração de sistemas, resultando numa solução extensível e reutilizável para processamento e ánalise de dados financeiros.

Capítulo 2

Enquadramento

2.1 Contexto

Este trabalho foi desenvolvido no âmbito da unidade curricular de Integração de Sistemas de Informação, do 3º ano de Engenharia de Sistemas Informáticos, no ano letivo 2024/2025.

2.2 Motivação

O mercado financeiro global imensas transações diárias. Sistemas ETL são essenciais para transformar dados brutos em informação acionável para investidores e analistas.

Este projeto demonstra a aplicação prática de conceitos de integração através da implementação de um pipeline completo desde extração de dados via API até visualização em dashboard interativo.

2.3 Âmbito

O projeto inclui:

- 4 transformações Pentaho (extração, limpeza, cálculo, load)
- Base de dados PostgreSQL com 3 tabelas
- Cálculo de 5 indicadores técnicos
- Dashboard Node-RED

- Job orquestrador com logging

2.4 Limitações

API free tier: 5 calls/min, 100 dias de histórico. Não inclui: machine learning, trading automático, autenticação multi-user, como as aplicações mais populares.

Capítulo 3

Problema

3.1 Descrição do Problema

Analistas financeiros necessitam de dados atualizados de mercados de ações, processados e enriquecidos com indicadores técnicos para suportar decisões de investimento.

Desafios principais:

1. Dados brutos de APIs não são diretamente utilizáveis
2. Cálculo de indicadores técnicos requer histórico e lógica complexa
3. Validação e limpeza de dados é essencial para qualidade
4. Visualização deve ser clara

3.2 Requisitos Funcionais

Na tabela abaixo é possível ver os requisitos funcionais e respectivas prioridades.

Tabela 3.1: Requisitos Funcionais

ID	Descrição	Prioridade
RF01	Extrair cotações via API Alpha Vantage	Alta
RF02	Validar dados com expressões regulares	Alta
RF03	Calcular SMA (10, 20, 50, 200 dias)	Alta
RF04	Calcular EMA, MACD, RSI, Bollinger Bands	Média
RF05	Armazenar em PostgreSQL	Alta
RF06	Dashboard com gráficos e gauges	Alta
RF07	Logging de execuções	Média

3.3 Requisitos Não-Funcionais

Os requisitos não funcionais, para trabalho menos técnico, estão listados a seguir.

- **Performance:** Pipeline rápida com quantidade significante de registos
- **Fiabilidade:** Taxa de sucesso alta com dados verificados
- **Escalabilidade:** Suportar adição de novos stocks
- **Manutenção:** Código modular e documentado

3.4 Arquitetura Proposta

Sistema organizado em 4 camadas:

1. **Extract:** API Alpha Vantage (REST, JSON)
2. **Transform:** Pentaho (validação, cálculos, normalização)
3. **Load:** PostgreSQL (6 tabelas relacionais)
4. **Visualize:** Node-RED (dashboard interativo)

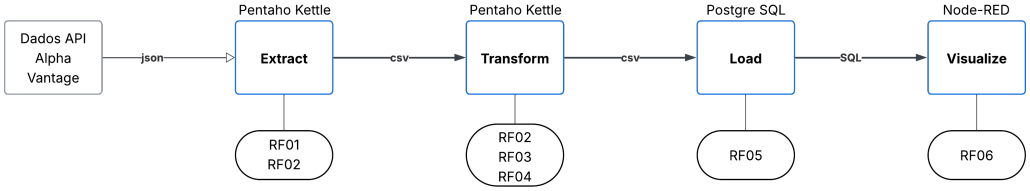

Figura 3.1: Arquitetura em 4 Camadas

Capítulo 4

Estratégia Utilizada

4.1 Escolha das Tecnologias

4.1.1 Pentaho Data Integration (Kettle 10.2)

Justificação:

- Interface visual drag-and-drop
- Open-source com comunidade ativa
- Suporte nativo REST, JSON, PostgreSQL
- Jobs com orquestração e logging

Alternativas consideradas: Apache NiFi (complexidade excessiva), Talend (menos recursos free), Python pandas (mais tempo desenvolvimento).

4.1.2 PostgreSQL 16

Escolhido por: funções analíticas (LAG, LEAD), performance comprovada, suporte de JSON.

4.1.3 Node-RED

Low-code, dashboard integrado, conexão PostgreSQL nativa, prototipagem rápida.

4.1.4 Alpha Vantage API

API gratuita, dados credíveis, documentação clara, free tier adequado para projeto.

4.2 Modelação de Dados

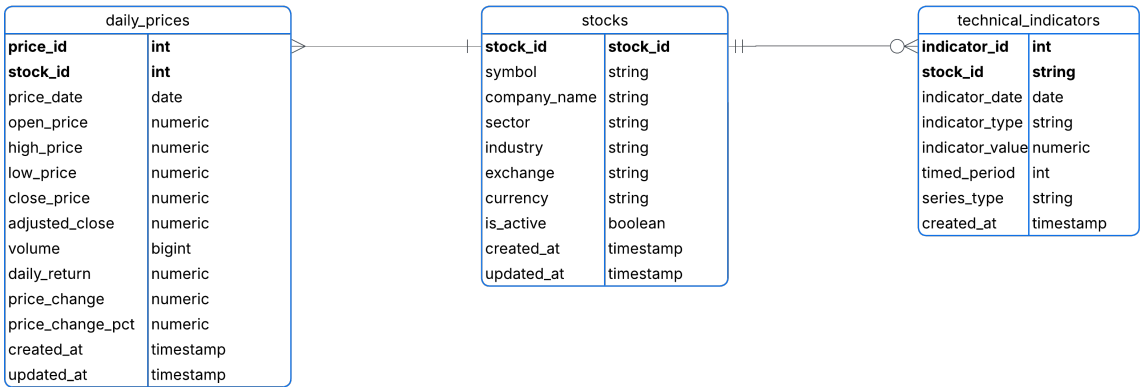

Figura 4.1: Diagrama Entidade-Relacionamento

Tabelas implementadas (6):

- `stocks` - Master de ações
- `daily_prices` - Cotações OHLC + volume
- `technical_indicators` - SMA, EMA, MACD, RSI, BB

Decisão crítica: Separação de `daily_prices` e `technical_indicators` evita explosão de colunas (13 indicadores × múltiplos períodos) e facilita extensibilidade.

Normalização: 3NF para consistência e integridade.

4.3 Operações ETL

Tabela 4.1: Operações por Transformação

Transformação	**Operações**	**Steps**
T1: Extract	REST, JSON parse, validação HTTP	
T2: Clean	Regex, deduplicação, cálculos	
T3: Calculate	SMA, EMA, MACD, RSI, BB	
T4: Load	Insert/Update, normalização	

Capítulo 5

Transformações

5.1 Transformação 1: Extract from Alpha Vantage API

5.1.1 Objetivo

Extrair cotações diárias (OHLC + volume) de 3 stocks via API, validar respostas HTTP, persistir dados brutos em CSV.

5.1.2 Diagrama

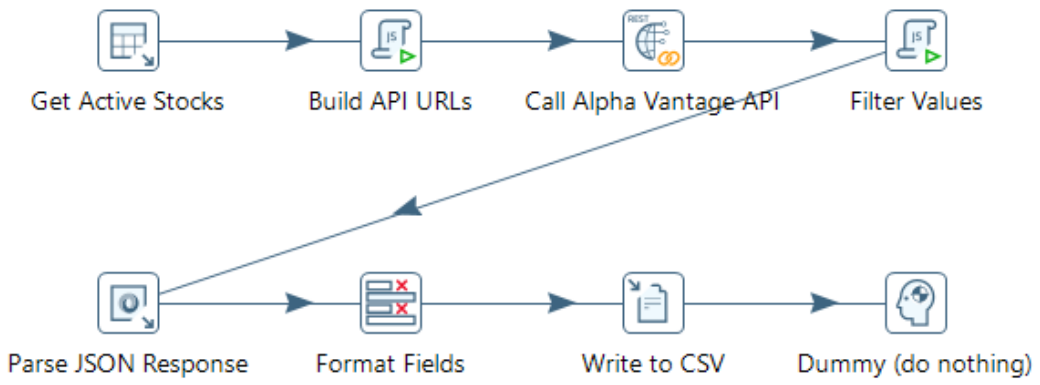

Figura 5.1: Transformação 1 - Fluxo Completo

5.1.3 Steps

Tabela 5.1: Steps da Transformação 1

#	Nome	Descrição	Tipo
1	Get Active Stocks	Query PostgreSQL	Table Input
2	Build API URLs	Constrói URL + API key (JS)	Modified JS Value
3	Call Alpha Vantage API	HTTP Get	REST Client
4	Filter Values	Filtra os dados	Modified JS Value
5	Parse JSON Response	Extrai campos	JSON Input
6	Format Fields	Conversão de tipos	Select Values
7	Write to CSV	Output ficheiro	Text File Output

5.1.4 Configurações Críticas

JSON Input: Path `$['Time Series (Daily)'][*]~` obtém chaves (datas) de objeto aninhado.

5.1.5 Output

Ficheiro: `raw_prices.csv`

5.1.6 Desafio

Rate limit API limitada. Solução: LIMIT 10 stocks, os símbolo provêm da BD.

5.2 Transformação 2: Clean and Normalize

5.2.1 Objetivo

Validar com regex, remover duplicados, calcular `daily_return` e `price_change`.

5.2.2 Diagrama

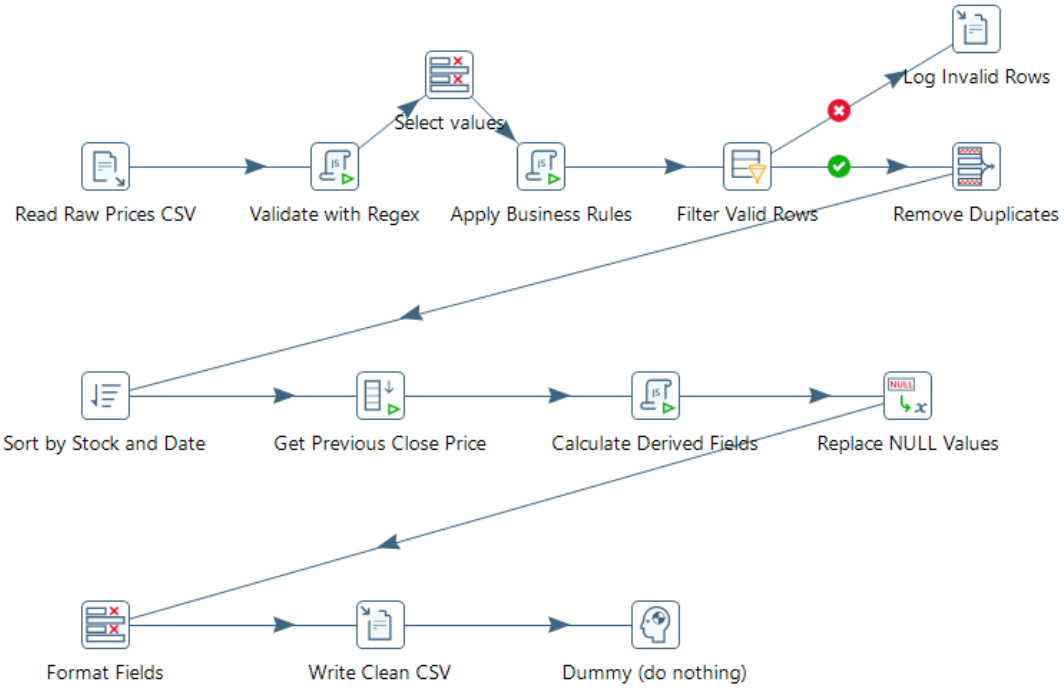

Figura 5.2: Transformação 2 - Fluxo Completo

5.2.3 Steps

Tabela 5.2: Steps da Transformação 2

#	Nome	Descrição	Tipo
1	Read Raw Prices CSV	Ler dados do CSV	CSV File Input
2	Validate with Regex	Símbolos, datas, preços	Modified JS Value
3	Select values	Selecionar campos necessários	Select Values
4	Apply Business Rules	Validação dos dados	Modified JS Value
5	Filter Valid Rows	Separa válidos/inválidos	Filter Rows
6	Remove Duplicates	Por stock_id + date	Unique Rows
7	Sort by Stock and Date	stock_id, price_date ASC	Sort Rows
8	Get Previous Close	LAG(close_price)	Analytic Query
9	Calculate Derivate Fields	Cálculos valores técnicos	Modified JS Value
10	Replace NULL Values	Defaults para NULLs	If NULL
11	Format Fields	Definir campos finais	Select Values
12	Write Clean CSV	Output	Text File Output

5.2.4 Regex Validation

Campo	Pattern
`symbol`	`^[A-Z]{1,5}$`
`price_date`	`^\d{4}-\d{2}-\d{2}$`
`close_price`	`^\d+\.?\d*$`

5.2.5 Cálculo Daily Return

Listing 5.1: Cálculo Daily Return

```
var daily_return = null;
if (previous_close != null && previous_close > 0) {
    daily_return = ((close_price - previous_close) /
       previous_close) * 100;
}
...

if (daily_return != null) {
    daily_return = Math.round(daily_return * 10000) / 10000;
}
```

Nota crítica: Sort por `price_date ASC` essencial para LAG funcionar corretamente.

5.2.6 Output

`clean_prices.csv`

5.3 Transformação 3: Calculate Technical Indicators

5.3.1 Objetivo

Calcular SMA (10,20,50,200), EMA (12,26), MACD, RSI, Bollinger Bands.

5.3.2 Diagrama

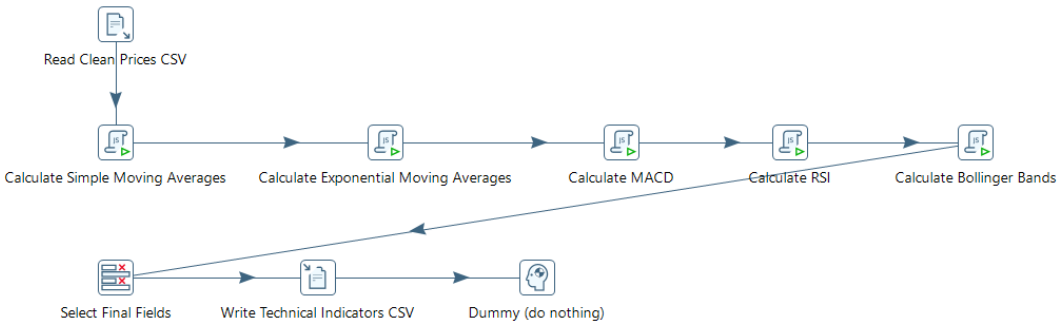

Figura 5.3: Transformação 3 - Fluxo Completo

5.3.3 Steps

Tabela 5.3: Steps da Transformação 3

#	Nome	Descrição	Tipo
1	Read Clean Prices CSV	Input	CSV File Input
3	Calculate SMA	10, 20, 50, 200 períodos	Modified JS Value
4	Calculate EMA	12, 26 períodos	Modified JS Value
5	Calculate MACD	Line, Signal, Histogram	Modified JS Value
6	Calculate RSI	14 períodos	Modified JS Value
7	Calculate BB	20 dias, 2σ	Modified JS Value
8	Select Final Fields	Campos finais	Select Values
9	Write CSV	Output	Text File Output

5.3.4 Algoritmos Implementados

SMA: Média aritmética simples de N períodos.

EMA: Média exponencial, multiplier = $2/(N+1)$.

MACD: EMA(12) - EMA(26), Signal = EMA(9) do MACD.

RSI: $(100 - 100/(1 + RS))$, onde $RS = avg\_gain/avg\_loss$.

Bollinger: $Middle = SMA(20)$, $Upper/Lower = Middle \pm 2\sigma$.

5.3.5 Output

`technical_indicators.csv`,

5.4 Transformação 4: Load to PostgreSQL

5.4.1 Objetivo

Carregar `daily_prices` e normalizar indicadores para `technical_indicators`.

5.4.2 Diagrama

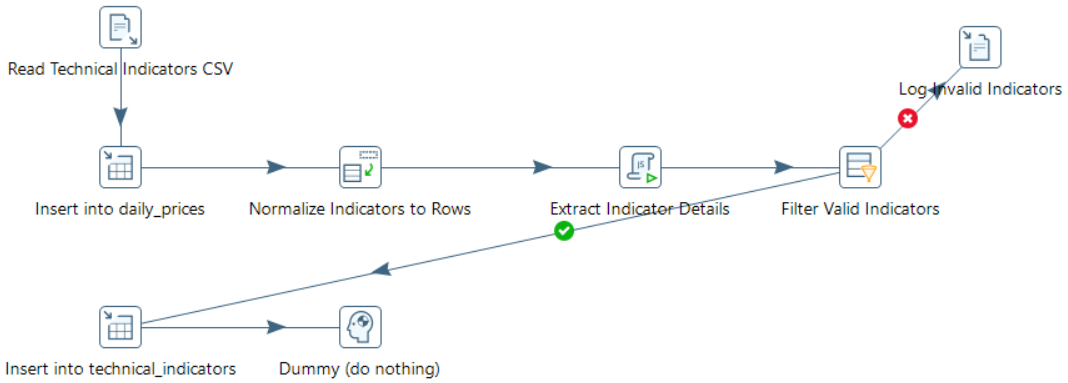

Figura 5.4: Transformação 4 - Fluxo Completo

5.4.3 Steps

Tabela 5.4: Steps da Transformação 4

#	Nome	Descrição	Tipo
1	Read Technical Indicators CSV	Input	CSV File Input
2	Insert into daily_prices	Truncate + Insert	Table Output
4	Normalize Rows	13 indicadores → linhas	Row Normaliser
5	Extract Details	Parse indicator_type	Modified JS Value
6	Filter NULLs	Remove valores NULL	Filter Rows
7	Insert into technical indicators	Truncate + Insert	Table Output

5.4.4 Row Normaliser

Transforma 1 linha com 13 colunas (sma_10, sma_20...) em 13 linhas com 1 coluna (indicator_value).

Capítulo 6

Jobs

6.1 Master ETL Pipeline Job

6.1.1 Objetivo

Orquestrar as 4 transformações em sequência, com tratamento de erros, validação de conexões, e logging centralizado.

6.1.2 Diagrama

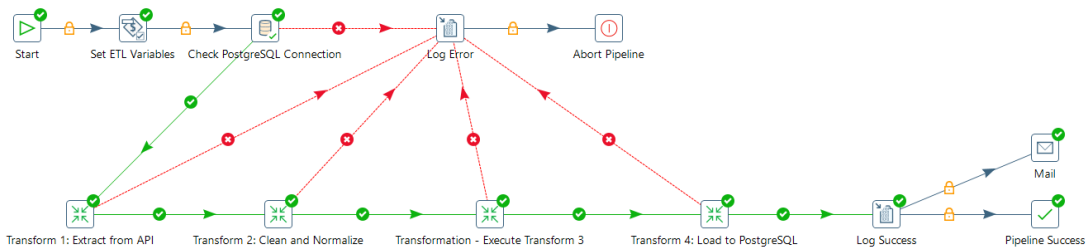

Figura 6.1: Job Master ETL Pipeline - Fluxo Completo

6.1.3 Entries

Tabela 6.1: Entries do Job

#	Nome	Ação	Condição
1	START	Início	-
2	Set Variables	Define PROJECT_HOME, API_KEY	TRUE
3	Check PostgreSQL	Valida conexão DB	TRUE
4	Transform 1	Extract API	TRUE
5	Transform 2	Clean	TRUE
6	Transform 3	Calculate	TRUE
7	Transform 4	Load DB	TRUE
8	Log Success	Mensagem sucesso	TRUE
9	Pipeline Sucess	Fim	-
10	Mail	Notificação email	TRUE
11	Log Error	Captura erros (FALSE)	FALSE
12	Abort	Termina job	FALSE

6.1.4 Controlo de Fluxo

Hops TRUE: Execução sequencial se step anterior teve sucesso.

Hops FALSE: Todos steps ligam a "Log Error"em caso de falha, seguido de Abort.

6.1.5 Variáveis

- `${PROJECT_HOME}`: C:/projetos/etl_financeiro
- `${ETL_DATE}`: YYYY-MM-DD (data execução)
- `${API_KEY}`: Chave Alpha Vantage

6.1.6 Logging

Cada transformação gera log individual:

- `logs/transform1.log`
- `logs/transform2.log`
- `logs/transform3.log`

- `logs/transform4.log`

Nível: Basic (rows processed, duration, status).

6.1.7 Notificações por Email

O job implementa notificações automáticas por email via SMTP Office 365:

Email de Sucesso:

- Trigger: Após "Log Success"
- Destinatário: Email institucional
- Conteúdo: Resumo da execução

Configuração SMTP: smtp.office365.com, porta 587, TLS.

Teste de falha: Simulação de DB offline resultou em abort correto com log de erro.

Capítulo 7

Dashboard e Visualização

7.1 Arquitetura

O dashboard foi implementado em Node-RED, estabelecendo conexão com a base de dados PostgreSQL (`etl_stocks`) para visualização em tempo real dos dados processados. A interface web está acessível em `http://localhost:1880/ui`.

A arquitetura utiliza o módulo `node-red-contrib-postgresql` (versão 0.15.4) para queries SQL e o `node-red-dashboard` (versão 3.6.6) para componentes visuais. Todos os fluxos estão organizados na tab "Overview".

7.2 Componentes do Dashboard

7.2.1 Price History

Este componente oferece visualização temporal dos preços de fecho das ações ativas no sistema. O fluxo inicia-se automaticamente após deploy e permite seleção dinâmica do período temporal através de um dropdown.

Funcionalidades principais:

- Seletor de período: 1 dia, 1 semana, 2 semanas, 1 mês, 3 meses, 6 meses, 1 ano
- Seletor de ações a visualizar
- Gráfico de linhas multi-série com legendas por símbolo
- Atualização dinâmica mediante seleção do período

- Formato de data DD/MM no eixo horizontal

Query SQL utilizada:

Listing 7.1: Price History Query

```
SELECT
    s.symbol,
    dp.price_date,
    dp.close_price
FROM daily_prices dp
INNER JOIN stocks s ON dp.stock_id = s.stock_id
WHERE s.is_active = TRUE
  AND dp.price_date >= CURRENT_DATE - INTERVAL '{{msg.
     interval}}'
ORDER BY s.symbol, dp.price_date;
```

O nó `Calculate Interval` converte o período selecionado em intervalos SQL apropriados, enquanto o `Format Chart Data` transforma os resultados numa estrutura compatível com o componente `ui_chart`, enviando cada ponto de dados com timestamp, tópico (símbolo) e valor (preço).

7.2.2 RSI Indicators

Apresentação visual profissional dos indicadores RSI (Relative Strength Index) de 14 períodos para todas as ações ativas. O layout adapta-se automaticamente ao número de ações disponíveis.

Características do design:

- Cards responsivos em grid adaptativo
- Valor numérico destacado
- Barra de progresso horizontal refletindo o valor RSI
- Indicadores de zona: 0, 30, 70, 100
- Badge de status: Oversold (¡ 30), Neutral (30-70), Overbought (¿ 70)
- Atualização automática a cada hora

Esquema de cores:

- Verde (#10b981): Oversold - potencial compra
- Amarelo (#f59e0b): Neutral - sem sinal claro

- Vermelho (#ef4444): Overbought - potencial venda

Query SQL implementada:

Listing 7.2: RSI Indicators Query

```
SELECT DISTINCT ON (s.symbol)
    s.symbol,
    ti.indicator_value as rsi,
    ti.indicator_date
FROM technical_indicators ti
INNER JOIN stocks s ON ti.stock_id = s.stock_id
WHERE LOWER(ti.indicator_type) = 'rsi'
    AND s.is_active = TRUE
ORDER BY s.symbol, ti.indicator_date DESC;
```

7.2.3 Comparative Table

Tabela comparativa completa com dados consolidados de todas as ações, oferecendo funcionalidades avançadas de análise e exportação.

Funcionalidades disponíveis:

- Ordenação por qualquer coluna (clique no cabeçalho)
- Pesquisa em tempo real (campo de busca)
- Exportação para CSV com timestamp
- Scroll vertical e horizontal independentes
- Hover states para melhor usabilidade

Colunas apresentadas:

1. **Symbol**: Ticker da ação (bold)
2. **Company**: Nome completo da empresa
3. **Price**: Preço de fecho atual ($)
4. **Open**: Preço de abertura ($)
5. **High**: Máximo do dia ($)
6. **Low**: Mínimo do dia ($)
7. **Change %**: Variação percentual com badge colorido (▲/▼)

8. **RSI**: Indicador com código de cores
9. **Volume**: Volume transacionado (formatado)

Query SQL complexa:

Listing 7.3: Comparative Table Query

```
SELECT
    s.symbol,
    s.company_name,
    dp.close_price,
    dp.open_price,
    dp.high_price,
    dp.low_price,
    dp.volume,
    dp.price_change_pct,
    dp.daily_return,
    ti.rsi_value,
    dp.price_date
FROM stocks s
INNER JOIN daily_prices dp ON s.stock_id = dp.stock_id
LEFT JOIN (
    SELECT stock_id, indicator_value as rsi_value,
           ROW_NUMBER() OVER (PARTITION BY stock_id
                              ORDER BY created_at DESC) as rn
    FROM technical_indicators
    WHERE LOWER(indicator_type) = 'rsi'
) ti ON s.stock_id = ti.stock_id AND ti.rn = 1
INNER JOIN (
    SELECT stock_id, MAX(price_date) as max_date
    FROM daily_prices
    GROUP BY stock_id
) latest ON dp.stock_id = latest.stock_id
        AND dp.price_date = latest.max_date
WHERE s.is_active = TRUE
ORDER BY s.symbol;
```

O nó `Format Table Data` processa os resultados, formatando valores numéricos, convertendo volumes para notação com separadores de milhares, e determinando a direção da variação para aplicação correta dos badges visuais.

A funcionalidade de exportação CSV é implementada client-side através de JavaScript, gerando um blob com todos os dados visíveis e trigger de download automático com nomenclatura temporal.

7.3 Fluxos Node-RED Implementados

Tabela 7.1: Fluxos Node-RED do Dashboard

Fluxo	Trigger	Refresh	Output
Price History	Manual + Startup	Dropdown	ui_chart
RSI Indicators	Inject	3600s (1h)	ui_template
Comparative Table	Inject	3600s (1h)	ui_template

Todos os fluxos partilham a mesma configuração PostgreSQL (`22527f1d94f6a320`) e estão organizados na tab "Overview" (`152c379456ac1dae`) para facilitar manutenção e navegação.

7.4 Design System

O dashboard segue princípios modernos de UI/UX:

- **Palette de cores**: Tailwind-inspired (grays, blues, greens, reds)
- **Typography**: Sans-serif system fonts, hierarquia clara
- **Spacing**: Grid de 8px, padding/margins consistentes
- **Interatividade**: Hover states, transitions suaves (0.2s-0.3s)
- **Responsividade**: Grid auto-fit, min-width constraints
- **Acessibilidade**: Contraste WCAG AA, estados de foco visíveis

7.5 Screenshots

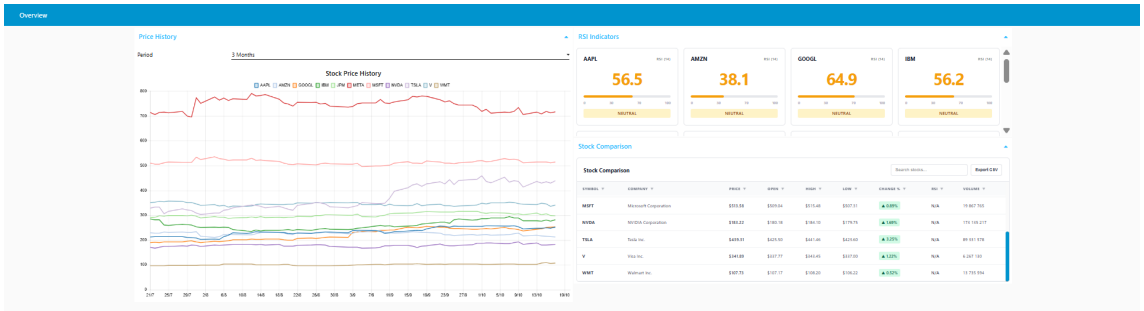

Figura 7.1: Dashboard - Página Overview Completa

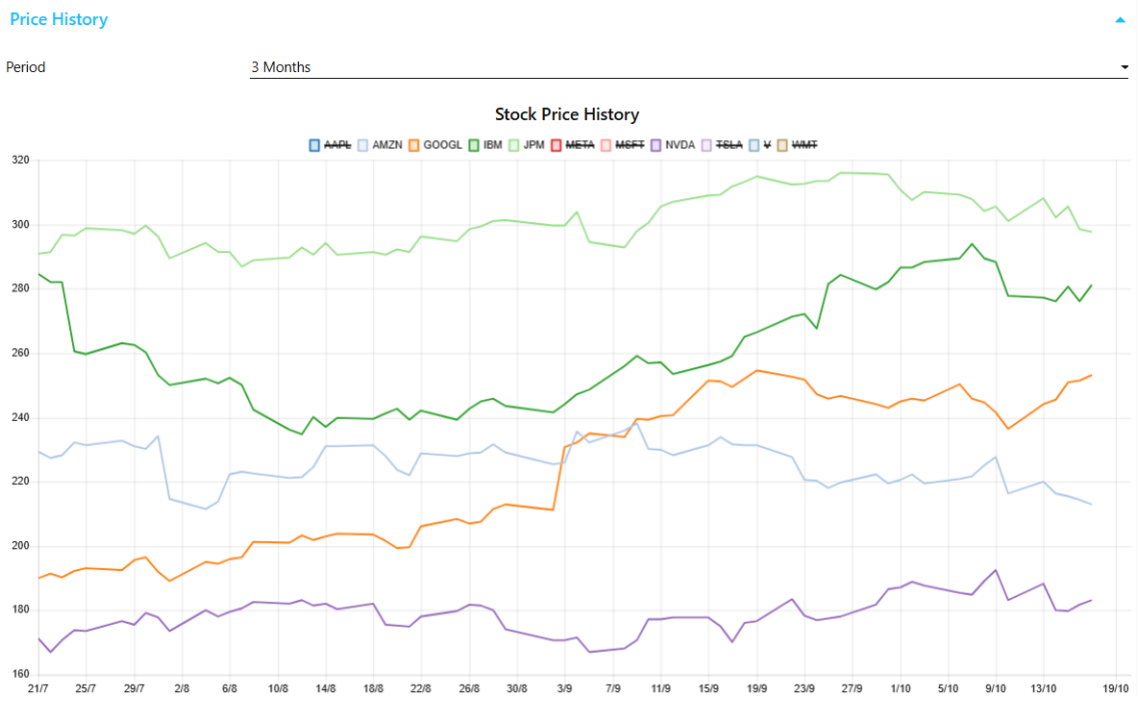

Figura 7.2: Componente Price History com Seletor de Período

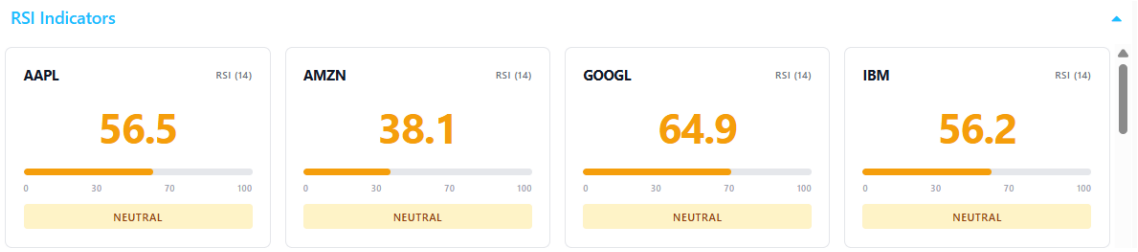

Figura 7.3: RSI Indicators - Cards Responsivos

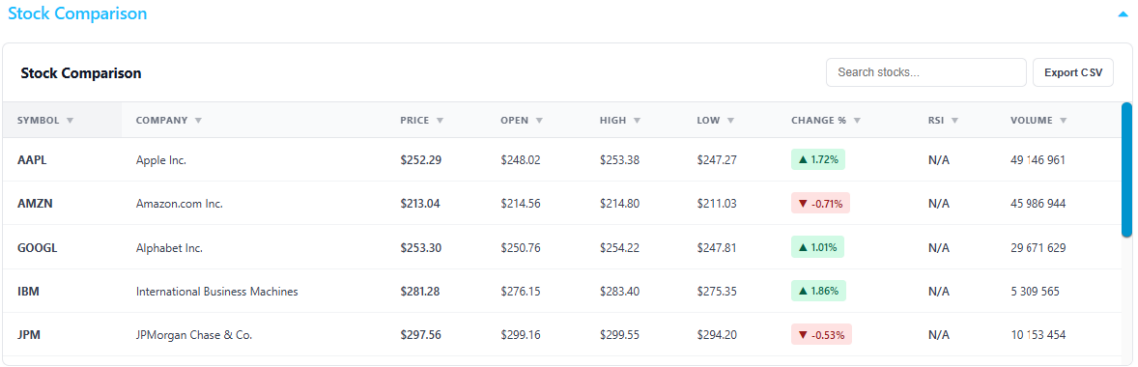

Figura 7.4: Comparative Table com Funcionalidades de Ordenação

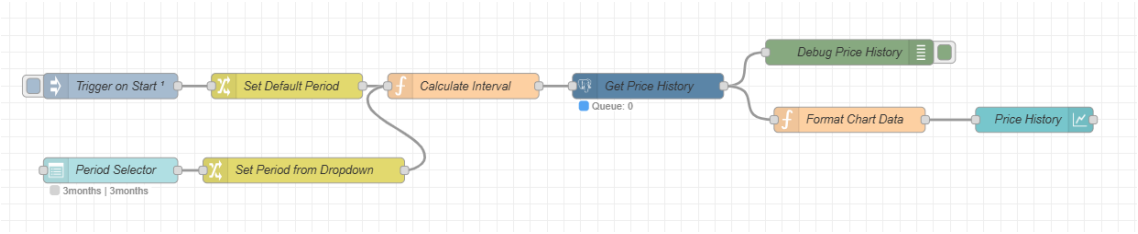

Figura 7.5: Node-RED Flow: Componente Price History com Seletor de Período

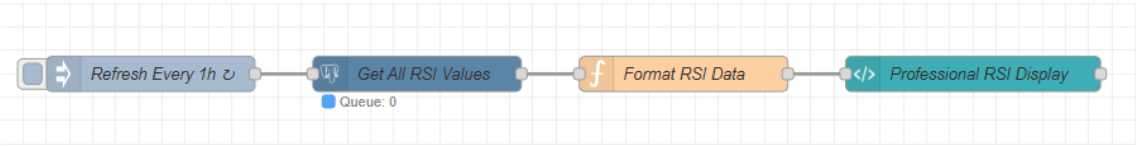

Figura 7.6: Node-RED Flow: RSI Indicators - Cards Responsivos

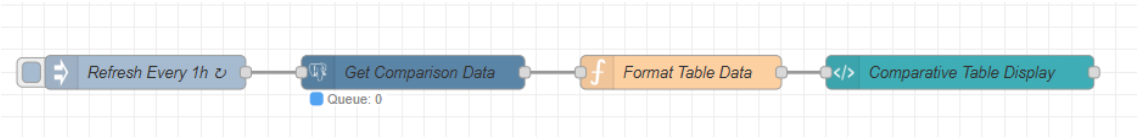

Figura 7.7: Node-RED Flow: Comparative Table com Funcionalidades de Ordenação

Capítulo 8

Dificuldades e Lições Aprendidas

8.1 Desafios Técnicos

8.1.1 JSON Parsing da Alpha Vantage

Problema: Estrutura JSON aninhada não-padrão com datas como chaves de objeto.

Tentativas: Calculator (não suporta), JSON Input básico (paths incorretos).

Solução: JSON Input com path `$['Time Series (Daily)'][*]~` para obter chaves como valores.

Aprendizagem: Testar APIs e estruturas JSON antes de implementar transformações.

8.1.2 Cálculo de Daily Return

Problema: Último dia sempre com daily_return = 0.

Causa: Sort não incluía `price_date`, apenas `stock_id` e `symbol`. LAG não funcionava corretamente.

Debugging: Preview step-by-step, queries SQL de validação.

Solução: Adicionar `price_date ASC` ao Sort. Remover campo `symbol` (redundante).

Aprendizagem: Funções analíticas (LAG/LEAD) dependem criticamente da ordenação correta.

8.1.3 RSI Não Aparecia no PostgreSQL

Problema: Row Normaliser tinha `rsi` (lowercase) mas script JavaScript esperava `RSI` (uppercase).

Solução: Adicionar `toUpperCase()` no JavaScript ou normalizar manualmente.

Aprendizagem: Case sensitivity é fonte comum de bugs em integrações. Sempre normalizar strings.

8.2 Desafios Conceptuais

8.2.1 Agregações em ETL vs BI Layer

Decisão: Não incluir Group By na Transformação 4.

Razão: Agregações pertencem à camada de visualização.

Trade-off: Performance de queries vs flexibilidade de análise. Optou-se por flexibilidade.

8.2.2 Normalização da Tabela technical_indicators

Decisão: Row Normaliser transforma 13 colunas em 13 linhas.

Vantagem: Extensibilidade - adicionar novos indicadores sem schema changes.

Desvantagem: Queries mais complexas.

8.3 Ferramentas e Ambiente

8.3.1 Curva de Aprendizagem Pentaho

Interface visual facilita, mas requer experiência.

Solução: Comunidade online (forums, Stack Overflow).

8.4 Gestão de Tempo

8.5 Reflexão

O que correu melhor: Modelação da base de dados, Node-RED dashboard (rápido).

O que faria diferente: Testar API e JSON parsing antes de começar transformações. Criar suite de testes desde início.

Competências adquiridas:

- Pentaho Data Integration avançado
- Debugging sistemático de pipelines ETL
- SQL analítico (LAG, LEAD)
- Integração multi-ferramenta

Capítulo 9

Conclusão e Trabalhos Futuros

9.1 Síntese do Trabalho

Este trabalho desenvolveu um sistema ETL completo para análise de dados financeiros, cumprindo os objetivos definidos:

Objetivos alcançados:

- Pipeline ETL funcional com 4 transformações + 1 job
- Extração de dados via API REST com parsing JSON
- Validação com expressões regulares
- Cálculo de 5 tipos de indicadores técnicos
- Base de dados PostgreSQL normalizada (6 tabelas)
- Dashboard interativo em Node-RED
- Sistema de logging e auditoria
- Sistema de mail

9.2 Contribuições

Académica: Aplicação prática de conceitos de Integração de Sistemas, demonstrando domínio de processos ETL, orquestração de workflows, e integração multi-ferramenta.

Técnica: Pipeline reutilizável e extensível. Código modular permite adicionar novos stocks, indicadores, ou fontes de dados com mínimas alterações.

Pessoal: Competências adquiridas em Pentaho, PostgreSQL, Node-RED, debugging complexo, e gestão de projetos de integração.

9.3 Limitações

Do trabalho atual:

- Dashboard básico
- Sem autenticação ou segurança

9.4 Trabalhos Futuros

9.4.1 Curto Prazo

1. Adicionar mais stocks
2. Adicionar notificações por email em caso de erro
3. Expandir dashboard (data tables, ETL monitoring)

9.4.2 Longo Prazo (projetos futuros)

1. Integrar outras fontes (Twitter sentiment, news APIs)
2. Implementar alertas baseados em indicadores (ex: RSI ¿ 70 envia email)
3. Criar relatórios PDF automáticos
4. Dockerizar solução para deployment facilitado
5. Adicionar mais indicadores (Stochastic, ADX, Fibonacci)
6. Machine learning para previsão de preços
7. Backtesting de estratégias de trading
8. Dashboard multi-user com autenticação
9. Cloud deployment (AWS, Azure)
10. API REST própria para consumo externo

9.5 Reflexão Final

Este projeto demonstrou a complexidade e importância de sistemas ETL em contextos reais. A integração de múltiplas ferramentas, tratamento de dados heterogéneos, e garantia de qualidade requerem planeamento cuidadoso e debugging sistemático.

A experiência adquirida em Pentaho Data Integration, PostgreSQL, e Node-RED é diretamente aplicável em ambientes empresariais. O projeto consolidou conhecimentos de Integração de Sistemas e preparou o terreno para trabalhos futuros mais complexos envolvendo big data, machine learning, e sistemas distribuídos.

Impacto: Sistema funcional que transforma dados brutos de APIs em informação acionável, demonstrando o valor de pipelines ETL bem desenhados na era do big data e analytics.

Bibliografia

[1] Alpha Vantage Inc. *Alpha Vantage API Documentation*. 2024. URL: https://www.alphavantage.co/documentation/ (acedido em 05/10/2025).

[2] Stefan Goessner. *JSONPath – XPath for JSON*. 2007. URL: https://goessner.net/articles/JsonPath/ (acedido em 10/10/2025).

[3] Hitachi Vantara. *Pentaho Data Integration Documentation*. 2024. URL: https://help.hitachivantara.com/Documentation/Pentaho/ (acedido em 05/10/2025).

[4] John J. Murphy. *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999. ISBN: 978-0735200661.

[5] OpenJS Foundation. *Node-RED Documentation*. 2024. URL: https://nodered.org/docs/ (acedido em 05/10/2025).

[6] PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*. 2024. URL: https://www.postgresql.org/docs/16/ (acedido em 05/10/2025).

[7] Joe Reis e Matt Housley. *Fundamentals of Data Engineering*. O'Reilly Media, 2022. ISBN: 978-1098108304.