🚀 JIRA Data Engineering Pipeline – Medallion Architecture

This project implements a robust Data Engineering pipeline to process JIRA data, utilizing the Medallion Architecture to transform raw data into business intelligence regarding SLA (Service Level Agreement).

us English Version

📋 Objective

This project automates the ingestion and processing of JIRA tickets to calculate resolution time in **business hours**, excluding weekends and national holidays.

🏗️ Pipeline Architecture

The project follows the **Medallion Architecture**:

- **Bronze Layer**: Raw ingestion from Azure Blob Storage using Service Principal authentication.
- **Silver Layer**: Data cleaning, normalization, and conversion to **Parquet** format for performance.
- **Gold Layer**: Business rules application and SLA calculation based on priority, integrated with **BrasilAPI**.
- **Data Quality**: Automated auditing of integrity, volume, and null validation.

💻 Dashboard & Visualization (Streamlit)

Interactive interface for performance management:

- **Dynamic Filters**: Date Range, Issue Type, and Analyst.
- **Trend Analysis**: Monthly SLA compliance tracking.
- **Performance Ranking**: Automated analyst classification (Top Performer, Standard, Needs Support).

🛠️ Technologies & Best Practices

- **Python 3.x** and **Pandas** for data manipulation.
- **Streamlit & Plotly** for visualization and interactive dashboards.
- **PyArrow**: Stable Parquet write engine.
- **Security**: Use of environment variables (`.env`) and sensitive data protection via `.gitignore`.

📈 Execution Evidence & Quality

The pipeline features a central orchestrator that validates each stage. In test runs:

- **Data Funnel**: 1000 raw records ➡️ 990 valid records ➡️ 804 finalized tickets.
- **Auditing**: 100% validation of priority rules and chronological integrity.

🛠️ Environment Setup Guide

Follow these steps to automatically configure the environment via CMD/Terminal using **Python 3.12**:

🚀 How to Run

1. **Open CMD (Command Prompt):**
   Use the cd command in CMD:
   `cd "C:\Users\Alex Lourenço\Desktop\project-root"` **<----- UPDATE TO YOUR ACTUAL PATH**

2. **Run Automatic Setup:**`python init_project.py`
   Run the master script. It will create the venv folder, install dependencies from requirements.txt, and ask for your credentials to set up the .env file.

3. **Execution Workflow**
   Whenever you use the project, follow this order in the terminal:
   1. Activate Environment: `venv\Scripts\activate`
   2. Run Pipeline: `python main.py`
   3.Open Dashboard: `streamlit run app.py`

BR Versão em Português

📋 Objetivo

O objetivo deste projeto é automatizar a ingestão e o processamento de chamados do JIRA para calcular o tempo de resolução em **horas úteis**, desconsiderando finais de semana e feriados nacionais.

🏗️ Arquitetura do Pipeline

O projeto segue a **Arquitetura Medallion**, garantindo organização e rastreabilidade:

- **Camada Bronze**: Ingestão de arquivos JSON brutos do Azure Blob Storage utilizando autenticação via Service Principal.
- **Camada Silver**: Limpeza e normalização dos dados. Extração de campos aninhados (`assignee`, `timestamps`) e conversão para o formato **Parquet** visando performance e preservação de metadados.
- **Camada Gold**: Aplicação de regras de negócio. Cálculo de SLA baseado na prioridade (High: 24h, Medium: 72h, Low: 120h), integrando com a **BrasilAPI** para identificação de feriados.
- **Data Quality**: Auditoria automática de integridade, volumetria e validação de nulos ao final do processo.

💻 Dashboard e Visualização (Streamlit)

O projeto inclui uma interface interativa para gestão de performance, permitindo:

- **Filtros Dinâmicos**: Seleção por Período, Tipo de Chamado e Analista.
- **Análise de Tendência**: Gráfico mensal de conformidade de SLA.

- **Ranking de Performance**: Classificação automática de analistas com status de suporte (Top Performer, Standard, Needs Support).

🛠️ Tecnologias e Boas Práticas

- **Python 3.x** e **Pandas** para manipulação de dados.
- **Streamlit & Plotly** para visualização e dashboards interativos.
- **PyArrow**: Engine de escrita Parquet estável.
- **Segurança**: Uso de variáveis de ambiente (`.env`) e proteção de dados sensíveis via `.gitignore`.

📊 Evidências de Execução e Qualidade

O pipeline conta com um orquestrador central que valida cada etapa. Em execuções de teste:

- **Funil de Dados**: Ingestão de 1000 registros ➡️ 990 registros válidos ➡️ 804 chamados finalizados.
- **Auditoria**: Validação de 100% das regras de prioridade e integridade cronológica.

🛠️ Guia de Configuração do Ambiente (Setup)

Siga estas etapas para configurar o ambiente automaticamente via CMD/Terminal utilizando o **Python 3.12**.:

🚀 Como Executar

1. **Abra o CMD (Prompt de Comando):**
   Use o comando cd no CMD (Prompt de Comando):
   `cd "C:\Users\Alex Lourenço\Desktop\project-root"` **<----- ATUALIZE O SEU ENDEREÇO**

2. **Executar o Setup Automático**`python init_project.py`
   Rode o script mestre. Ele criará a pasta venv, instalará as dependências do requirements.txt e solicitará suas credenciais para o arquivo .env:

3. **Fluxo de Execução**
   Sempre que for utilizar o projeto, siga esta ordem no terminal:
   1. Ativar Ambiente: `venv\Scripts\activate`
   2. Rodar Pipeline: `python main.py`
   3. Abrir Dashboard: `streamlit run app.py`

🗂️ Estrutura de Pastas / Project Structure

```
PROJECT-ROOT/
├── .pycache/                # Arquivos de cache do Python (Ignorado)
├── data/                    # Armazenamento Local / Local Storage (Ignorado)
├── src/                     # Código-fonte principal / Source Code
│   ├── __pycache__/         # Cache interno da src
│   ├── bronze/              # Camada de Ingestão (JSON Raw)
│   │   ├── __pycache__/
```

```
│   │   └── ingest_bronze.py
│   ├── silver/             # Camada de Limpeza e Transformação (Parquet)
│   ├── gold/               # Camada de Regras de Negócio / Analytics
│   ├── sla_calculation.py  # Motor de Cálculo de SLA
│   └── validate_pipeline.py # Auditoria e Validação de Dados
├── venv/                   # Ambiente Virtual Python (Local)
├── .env                    # Variáveis de ambiente e Credenciais
├── .gitignore              # Configuração de arquivos ignorados pelo Git
├── .python-version         # Definição da versão do Python (Ex: 3.12)
├── app.py                  # Interface do Dashboard (Streamlit)
├── init_project.py         # Script de inicialização/setup do projeto
├── main.py                 # Orquestrador Central / Ponto de Entrada
├── README.md               # Documentação principal (Markdown)
├── README.pdf              # Exportação da documentação em PDF
└── requirements.txt        # Lista de dependências do projeto
```