# Documentação Técnica Aprofundada: Pipeline de Automação de Notas Fiscais

## 1. Visão Geral e Justificativa do Projeto

Este projeto foi concebido para automatizar a complexa tarefa de aquisição, processamento e organização de Notas Fiscais Eletrônicas (NF-e) e seus Documentos Auxiliares (DANFE em PDF). A solução proposta visa mitigar gargalos operacionais, reduzir erros manuais e otimizar o fluxo de trabalho de compliance fiscal, proporcionando um mecanismo robusto para a gestão documental.

A arquitetura da pipeline é fundamentada no padrão **E.T.L. (Extract, Transform, Load)**, uma metodologia amplamente reconhecida na engenharia de dados. Esta escolha arquitetural promove uma separação clara de responsabilidades entre os estágios, o que facilita o desenvolvimento modular, a depuração isolada, a escalabilidade de componentes e a manutenibilidade do código-base.

## 2. Arquitetura da Pipeline (Modelo E.T.L. Detalhado)

A pipeline é composta por estágios lógicos e fisicamente distintos, orquestrados por um módulo central e suportada por serviços auxiliares de configuração e logging.

### 2.1. Estágio de Extração (Extract)

- **Módulo:** `src/pipeline/extract.py`
- **Finalidade:** Este estágio atua como a **camada de ingestão de dados brutos**, sendo responsável por coletar as chaves de acesso das notas fiscais a partir de fontes definidas. A precisão na extração é vital para a integridade dos dados subsequentes.
- **Funcionalidades Detalhadas:**
    - **Leitura de Arquivos TXT:** Itera sobre todos os arquivos `.txt` localizados no diretório de entrada (`./CHAVES NOTAS/`).
    - **Extração de Chaves de Acesso:** Processa o conteúdo linha por linha, utilizando **expressões regulares** para validar que cada linha contém exatamente o formato padrão de 44 dígitos numéricos de uma chave de acesso. Linhas inválidas são ignoradas e logadas como aviso.

    - **Identificação de Filial:** Extrai o código da filial diretamente do nome do arquivo (ex: "CHAVES FILIAL 04.txt" é parseado para "FILIAL 04"). Esta informação é crucial para a organização hierárquica dos arquivos de saída.
    - **Tratamento de Exceções:** Inclui mecanismos para lidar com erros de acesso ao sistema de arquivos (ex: arquivo não encontrado), garantindo a resiliência do processo.
- **Saída:** Uma lista de tuplas `(filial_code, key)`, onde `filial_code` é a identificação da filial e `key` é a chave de acesso validada, pronta para ser consumida pelo estágio de Transformação.

## 2.2. Estágio de Transformação (Transform)

- **Módulo:** `src/pipeline/transform.py`
- **Finalidade:** Este é o **core inteligente** da pipeline, responsável por converter uma chave de acesso em documentos fiscais completos (XML e DANFE PDF). Sua implementação emprega uma **estratégia híbrida** (API direta e Web Scraping) para maximizar a resiliência e a taxa de sucesso na obtenção dos dados, adaptando-se a falhas ou limitações de uma única fonte.
- **Componentes e Funcionalidades Detalhadas:**
    - **Comunicação API Direta (`requests`):**
        - **Primeira Tentativa (Eficiência):** Inicia tentando obter o XML via uma requisição `POST` para uma API de download de XML.
        - **Formato da Requisição:** A URL completa é construída dinamicamente. Os cabeçalhos HTTP são cuidadosamente definidos para simular uma requisição de navegador (incluindo `User-Agent`, `Referer`, `Content-Type: text/plain;charset=UTF-8`, etc.).
        - **Validação da Resposta:** O `status_code` da resposta é verificado. Se não for `200 OK` (sucesso), a API direta é considerada falha para aquela chave, e o fluxo de fallback é acionado.
    - **Estratégia de Fallback (Web Scraping com `selenium`):**
        - **Ativação de Contingência:** Este caminho é ativado **apenas se a tentativa via API direta falhar** (ou seja, se a resposta não for `200 OK`). Isso garante a priorização do método mais eficiente.
        - **Inicialização do WebDriver:** Uma função dedicada inicializa o navegador Chrome (em modo `headless` por padrão), otimizado para automação. Configurações especiais são aplicadas para **forçar downloads automáticos** (desabilitando pop-ups de "Salvar Como" e as verificações de segurança do Chrome, como a Navegação Segura) e para garantir estabilidade em ambientes sem interface gráfica. A ferramenta

`webdriver_manager` assegura a compatibilidade automática do driver.
        - **Navegação e Busca:** Uma função específica navega até a URL do serviço web, insere a chave de acesso no campo de busca identificado (usando `By.XPATH` com base em `placeholder` e texto de botão) e clica para iniciar a consulta. Esperas inteligentes garantem que a página de resultados (`/ver-danfe`) seja totalmente carregada antes da próxima interação.
        - **Extração de Documentos (Scraping):** Uma função especializada interage com a página de resultados: localiza e clica nos botões/links de download para XML e DANFE. Implementa lógica para lidar com eventuais pop-ups de segurança de download. Monitora o diretório de downloads temporário para ler o conteúdo dos arquivos baixados. Em caso de falha no download via scraping, um conteúdo simulado é utilizado com aviso no log.
        - **Gerenciamento de Recursos:** O WebDriver é encerrado (`driver.quit()`) ao final da operação para liberar recursos do sistema.
    - **Extração de Número da Nota:** Uma função auxiliar analisa o conteúdo XML (obtido via API ou Selenium) utilizando `xml.etree.ElementTree` para localizar e extrair o número da Nota Fiscal (`nNF`). Esta etapa é crucial para a nomeação padronizada dos arquivos.
    - **Geração da DANFE PDF (via API):**
        - **Continuidade da Eficiência:** A geração da DANFE PDF **ainda prioriza o uso de API** (`requests.post`), mesmo que o XML tenha sido obtido via web scraping. O `xml_content` é enviado como payload (`data=xml_content`) com `Content-Type: text/plain` para a API de geração de DANFE.
    - **Tratamento de Erros Abrangente:** O módulo `process_single_key` é robusto, com múltiplos blocos `try-except` para capturar e logar detalhadamente uma vasta gama de exceções: erros HTTP (com detalhes de `status_code` e resposta), erros de conexão de rede, timeouts, exceções específicas do Selenium (falhas de interação com o navegador), erros de parsing XML e erros genéricos inesperados.
- **Saída:** Uma tupla contendo o conteúdo binário do XML, o conteúdo binário do PDF da DANFE e o número da nota fiscal correspondente, ou `(None, None, None)` em caso de falha completa no estágio.

## 2.3. Estágio de Carregamento (Load)

- **Módulo:** `src/pipeline/load.py`

- **Finalidade:** Este estágio é responsável por persistir os documentos processados no sistema de arquivos local, seguindo uma estrutura de organização bem definida e intuitiva.
- **Funcionalidades Detalhadas:**
    - **Criação de Estrutura de Pastas:** Funções dedicadas garantem a criação da hierarquia de diretórios: `./NOTAS E XML/<filial>/<AAAA-MM-DD>/XML/` e `./DANFE/`. Utiliza mecanismos para criar pastas apenas se não existirem, prevenindo erros em execuções repetidas.
    - **Salvamento de Arquivos:** Escreve o conteúdo binário do XML e do PDF nos respectivos subdiretórios. Os arquivos são nomeados de forma padronizada (`<numero_da_nota>.xml` e `<numero_da_nota>.pdf`), utilizando o número da nota extraído no estágio de Transformação.
    - **Auditoria:** Registra logs de sucesso ou falha das operações de salvamento de arquivos.
- **Saída:** Arquivos XML e PDF armazenados fisicamente no sistema de arquivos.

# 3. Módulos de Suporte

## 3.1. Orquestrador Principal (`src/main.py`)

- **Função:** `run_data_pipeline()`
- **Papel Detalhado:**
    - Inicia a execução da pipeline e configura o sistema de logging global.
    - Gerencia o loop principal que itera sobre cada chave de acesso fornecida pelo estágio de Extração.
    - Coordena a passagem de dados e o controle de fluxo entre os estágios de Extração, Transformação e Carregamento.
    - **Tratamento de Exceções Global:** Captura quaisquer exceções não tratadas nos estágios individuais, garantindo que a pipeline seja resiliente a falhas inesperadas e que todos os incidentes sejam devidamente registrados.

## 3.2. Configurações Globais (`src/config.py`)

- **Propósito:** Atua como o repositório centralizado de todas as variáveis de configuração do projeto. Esta abordagem promove a manutenibilidade e a adaptabilidade do sistema, permitindo que alterações de ambiente ou parâmetros de API sejam feitas em um único local, sem modificar o código-fonte da lógica de negócio.

- **Conteúdo:** Inclui caminhos de diretório (entrada, saída, logs), URLs completas das APIs de `meudanfe.com` (para XML e DANFE), a `MEUDANFE_WEB_URL` para o web scraping, a `MEUDANFE_API_KEY` (chave de autenticação vital para as APIs externas), e tempos limite para requisições (`REQUEST_TIMEOUT_SECONDS`).

### 3.3. Sistema de Logging (`src/logger_config.py`)

- **Propósito:** Fornecer um mecanismo robusto para o registro detalhado de eventos, diagnósticos e erros em toda a pipeline. Essencial para depuração, auditoria e monitoramento do processo em tempo real e histórico.
- **Funcionalidades:**
    - **Configuração de Logger:** Utiliza o módulo `logging` padrão do Python para configurar um logger raiz que centraliza as saídas.
    - **Saída Dual:** As mensagens de log são direcionadas simultaneamente para um arquivo (`logging.FileHandler`) e para o console (`logging.StreamHandler`), proporcionando tanto o monitoramento em tempo real da execução quanto a capacidade de análise histórica.
    - **Rotação de Logs:** Gera arquivos de log diários (nomeados como `processamento_YYYY-MM-DD.txt`), facilitando a organização, arquivamento e revisão de registros de execuções passadas.
    - **Níveis de Log:** Suporta diferentes níveis de severidade (DEBUG, INFO, WARNING, ERROR, CRITICAL) para categorizar as mensagens, o que permite filtragem e foco em informações relevantes durante a análise.

## 4. Fluxo de Execução e Interação dos Componentes

1. A execução da pipeline é iniciada através do script `main.py`, que realiza a configuração inicial do ambiente de logging.
2. O `main.py` delega ao estágio de Extração (`src/pipeline/extract.py`) a responsabilidade de ler os arquivos de chaves de acesso do diretório de entrada, resultando em uma lista de tuplas `(filial, chave)`.
3. Para cada tupla `(filial, chave)` extraída, o `main.py` inicia um ciclo de processamento:
   a. O estágio de Transformação (`src/pipeline/transform.py`) é invocado. Ele primeiramente tenta obter o XML da nota via uma requisição `POST` para a API direta.

   b. **Caso a requisição API falhe** (indicado por um `status_code` diferente de `200`), o estágio de Transformação ativa sua lógica de fallback. Isso envolve a inicialização de um navegador Chrome (via `initialize_webdriver`), a

navegação até o website da `meudanfe.com` com a chave de acesso, a simulação de um clique no botão de busca (`perform_meudanfe_search`), e a extração subsequente do XML e DANFE (PDF) diretamente da página de resultados (`extract_xml_results_page`).

c. Independentemente da fonte de onde o XML foi obtido (API ou web scraping), o número da nota é extraído do seu conteúdo.

d. A DANFE PDF é gerada. Este processo prioriza a utilização de uma API de conversão (`requests.post`) que recebe o XML como payload.

e. Os documentos XML e PDF processados, juntamente com o número da nota, são então passados para o estágio de Carregamento.

f. O estágio de Carregamento (`src/pipeline/load.py`) é responsável por criar a estrutura de pastas apropriada e salvar o XML e o PDF nos diretórios designados.

g. Um atraso configurável (`time.sleep()`) é introduzido entre o processamento de cada chave, uma medida preventiva para gerenciar a frequência de requisições às APIs externas e garantir a estabilidade do sistema.

4. Ao longo de todo o processo, o Sistema de Logging registra eventos, avisos e erros detalhadamente, fornecendo um rastro completo para fins de depuração e auditoria.
5. Ao finalizar o ciclo de processamento de todas as chaves, o `main.py` registra o término da execução da pipeline.

## 5. Resiliência, Manutenibilidade e Escalabilidade

A concepção desta pipeline incorpora princípios fundamentais de engenharia de software para garantir sua robustez e longevidade operacional:

- **Resiliência a Falhas:** A estratégia híbrida no estágio de Transformação (API primária com fallback para Selenium) é um pilar de resiliência. Ela assegura que a pipeline possa se adaptar a instabilidades de serviço, erros temporários de API (como `502 Server Error`) ou até mesmo mudanças nas APIs, mantendo uma alta taxa de sucesso na obtenção dos documentos.
- **Manutenibilidade:** A modularização em estágios E.T.L. bem definidos e a centralização de todas as configurações em `src/config.py` resultam em uma arquitetura clara e de fácil manutenção. Alterações em URLs de API, caminhos de diretório ou chaves de autenticação podem ser realizadas de forma eficiente e isolada.

- **Observabilidade:** A implementação de um sistema de logging detalhado, com logs em arquivos diários e saída para console, proporciona uma visibilidade completa do comportamento da pipeline, facilitando a identificação proativa e a rápida resolução de problemas.
- **Potencial de Escalabilidade:** A clara separação de estágios permite que, em um ambiente de nuvem (como Google Cloud Functions ou AWS Lambda), componentes específicos possam ser escalados independentemente para lidar com volumes de processamento crescentes. A priorização do uso de `requests` (que possui uma pegada de recursos leve) contribui para a eficiência e o potencial de custo-benefício em infraestruturas de nuvem.