**Documentação do Workflow: Importação de Fatura de Cartão (My\_workflow\_V6)**

Este documento descreve o fluxo de trabalho automatizado no n8n para importar dados de faturas de cartão de crédito (formato PDF) recebidas por e-mail para uma base de dados PostgreSQL.

**Objetivo**

Automatizar a coleta, o parsing e a preparação de dados de transações de faturas de cartão de crédito para inserção no banco de dados financeiro pessoal.

**Arquitetura**

O workflow utiliza uma arquitetura modular baseada em containers Docker:

* **Container n8n:** Orquestra o fluxo de trabalho, monitora e-mails, salva arquivos temporariamente e chama o serviço de parsing.
* **Container de Serviço de Parsing (Python/Flask):** Um microserviço dedicado que recebe requisições HTTP, lê arquivos PDF de um volume compartilhado, parseia o conteúdo e retorna dados estruturados em JSON.
* **Base de Dados PostgreSQL:** Armazena as transações financeiras importadas.

Um volume Docker compartilhado é usado para que o container n8n possa salvar os arquivos PDF e o container de serviço de parsing possa lê-los.

**Fluxo de Trabalho (Nós e Lógica)**

O workflow My\_workflow\_V6 consiste nos seguintes nós e etapas:

1. **Gmail Trigger:**
   * **Função:** Atua como o ponto de partida. Monitora e-mails de fatura e baixa anexos PDF.
   * **Detalhes:** Configurado para monitoramento agendado (a cada hora) e download de anexos de remetente específico.
   * **Saída:** Itens representando e-mails com anexos.
2. **Code (Extract Attachments):**
   * **Função:** Processa a saída do "Gmail Trigger" para extrair os anexos individuais e criar um item separado no workflow para cada anexo. Também extrai informações relevantes do e-mail original (ID, nome do arquivo, assunto) e as adiciona ao JSON de cada item de anexo.
   * **Lógica:** Script Python que itera sobre os anexos de cada e-mail de entrada e cria um novo item de saída para cada anexo, copiando os dados binários e adicionando metadados do e-mail.
   * **Saída:** Itens representando anexos individuais com metadados.
3. **Save Attachment to Staging:**
   * **Função:** Salva o anexo binário de cada item de entrada em um volume compartilhado (/shared\_staging).
   * **Detalhes:** Nó Read/Write Files configurado para escrita no caminho /shared\_staging/{{ $json.fileName }}.
   * **Saída:** Itens com o caminho onde o arquivo foi salvo.
4. **HTTP Request (Call Parser Service):**
   * **Função:** Chama o microserviço de parsing Python via HTTP para processar o arquivo PDF salvo.
   * **Configuração:** Configurado para fazer uma requisição POST para o endpoint /parse\_pdf do serviço de parsing (por exemplo, http://parser\_service\_rfa:5000/parse\_pdf). Envia o caminho do arquivo salvo e metadados no corpo JSON.
   * **Saída:** Item contendo a resposta JSON do serviço de parsing (dados da fatura e lista de transações).
5. **Telegram Notification:**
   * **Função:** Notifica via Telegram sobre o resultado do parsing.
   * **Detalhes:** Conectado após o "HTTP Request". Usa dados da saída do "HTTP Request" (contagem de transações, total, vencimento, nome do arquivo) para compor a mensagem.
6. **Gmail:**
   * **Função:** Marca o e-mail original como lido.
   * **Configuração:** Conectado após o "HTTP Request". Usa a operação markAsRead e o messageId original do e-mail (obtido da saída do "HTTP Request", que o serviço de parsing incluiu na resposta).
   * **Saída:** Retorna um item indicando o sucesso da operação no Gmail.
7. **Split Out Transactions:**
   * **Função:** Pega a lista de transações (do campo transactions da saída do HTTP Request) e cria um item separado no workflow para *cada* transação individual na lista.
   * **Configuração:** Nó Split Out configurado para dividir pelo campo transactions.
   * **Saída:** Itens, cada um representando uma transação individual.
8. **Set Transaction Fields:**
   * **Função:** Adiciona campos derivados (reference\_month, reference\_year, notes), campos fixos (status, origin, account\_id, user\_id, created\_at, updated\_at) e item\_index a cada item de transação.
   * **Detalhes:** Nó Set com expressões para derivar valores da data de vencimento e descrição, definir valores fixos e usar $itemIndex para o item\_index.
   * **Saída:** Itens de transação com campos adicionais.
9. **Merge (Transactions and Fields):**
   * **Função:** Mescla os itens de transação originais (da saída do "Split Out Transactions") com os campos adicionados pelo "Set Transaction Fields". **(Observação:** Este nó e sua conexão da saída do Split Out parecem redundantes se o "Set Transaction Fields" já está adicionando os campos aos itens que recebe diretamente do Split Out. A saída do "Set Transaction Fields" já é a lista de transações com os campos adicionados. Este nó pode ser removido se o "Set Transaction Fields" estiver conectado diretamente ao "Postgres (Search Payee)" e "Merge (Payee Results)").
   * **Configuração:** Configurado como combine e mergeByFields.
   * **Saída:** Retorna os itens mesclados.
10. **Postgres (Search Payee):**
    * **Função:** Busca o id do Terceiro (core\_payeepayer) no banco de dados com base na descrição da transação.
    * **Configuração:** Operação select na tabela core\_payeepayer, filtrando pelo nome extraído da descrição (usando filtro LIKE). Limita a 1 resultado.
    * **Saída:** Itens contendo os dados do Terceiro encontrado (se houver).
11. **Code (Add item\_index to Payee Results):**
    * **Função:** Pega a saída do "Postgres (Search Payee)" (dicionários dos Terceiros encontrados) e adiciona o item\_index original a cada um, acessando-o do contexto do item de entrada que gerou a busca. Prepara os dados para a mesclagem.
    * **Lógica:** Script Python que itera sobre os resultados da busca do Postgres, acessa o item\_index original do contexto do n8n e o adiciona ao dicionário do resultado da busca, retornando itens n8n com .json.
    * **Saída:** Itens n8n com .json contendo os dados do Terceiro + original\_item\_index.
12. **Merge (Payee Results):**
    * **Função:** Mescla os itens de transação originais (da saída do "Set Transaction Fields") com os resultados da busca do Terceiro (da saída do "Code (Add item\_index to Payee Results)").
    * **Configuração:** Configurado como mergeByProperty usando o item\_index. Inclui todos os itens da Source 0 (transações) e adiciona os dados da Source 1 (Terceiros) em um campo (por exemplo, found\_payee\_data).
    * **Saída:** Itens de transação com campo found\_payee\_data.
13. **Set (Assign Payee ID):**
    * **Função:** Atribui o payee\_payer\_id ao item de transação. Se um Terceiro foi encontrado na etapa anterior, usa o ID encontrado; caso contrário, usa o ID padrão (1975).
    * **Configuração:** Usa uma expressão condicional baseada no campo found\_payee\_data.
    * **Saída:** Itens de transação com payee\_payer\_id finalizado.
14. **Code (Clear Transaction):**
    * **Função:** Realiza limpeza final nos itens de transação antes da inserção, removendo campos temporários ou desnecessários.
    * **Lógica:** Script JavaScript que itera sobre os itens e remove campos como json.json, binary, pairedItem, etc. Também pode conter lógica para definir o ID padrão da categoria se ainda não foi definido (embora a busca de categoria seja o próximo passo).
    * **Saída:** Itens de transação limpos.
15. **Postgres (Search Last Category by Payee):**
    * **Função:** Busca a última transação na tabela core\_transaction para o payee\_payer\_id atual para obter a categoria usada anteriormente.
    * **Configuração:** Nó Postgres Select na tabela core\_transaction.
    * **Saída:** Itens contendo o category\_id da última transação encontrada (se houver).
16. **Set (Assign Category ID):**
    * **Função:** Atribui o category\_id à transação atual. Se uma categoria foi encontrada, usa esse ID; caso contrário, usa um ID padrão (358).
    * **Configuração:** Usa uma expressão condicional baseada na busca anterior.
    * **Saída:** Itens de transação com category\_id finalizado.
17. **Merge (Category Results):**
    * **Função:** Mescla os itens de transação (que passaram pela busca de categoria) com os resultados da busca de Categoria.
    * **Configuração:** Configurado como combine e mergeByFields.
18. **Postgres (Insert Transaction):**
    * **Função:** Insere os itens de transação finalizados na tabela core\_transaction.
    * **Configuração:** Operação insert, mapeando os campos dos itens de entrada para as colunas da tabela.

**Melhorias Futuras**

* Refinar a lógica de identificação de Terceiros e Categorias (por exemplo, usando busca por similaridade ou aprendizado de máquina).
* Implementar lógica mais robusta para identificar e lidar com transações duplicadas durante a inserção.
* Criar um campo para categoria padrão na tabela core\_payeepayer.
* Explorar o uso de um agente de IA para inferir categorias (conforme discutido).
* Adicionar tratamento de erros mais granular para cada nó.