## Problema:

As operadoras enviam os datasets e precisamos validar se esses dados estão condizentes com o nosso layout. Este processo é manual e consome várias horas-homem

## Solução:

Validador de datasets genérico capaz de validar um grande volume de dados de forma assíncrona e paralela baseados num schema previamente criado.

### Pontos importantes:

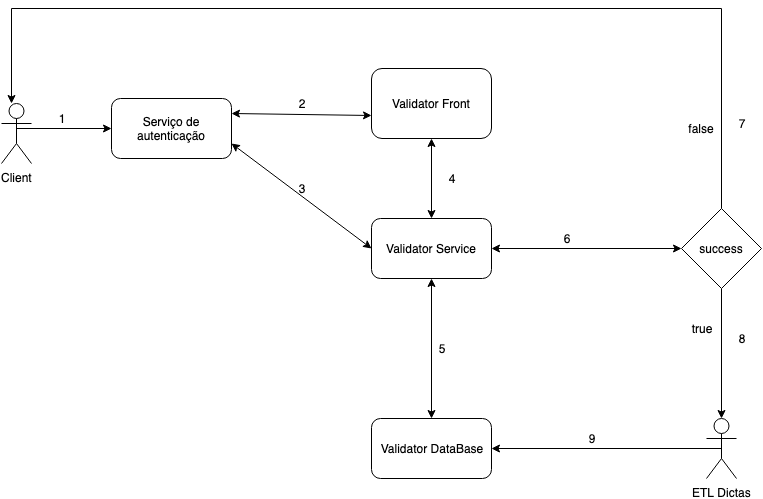
* Poder ter validadores customizados (regex ou callback)
* Poder receber arquivos compactados
* Paralelo e async
* Validar linhas duplicadas (para isso será preciso cadastrar uma chave para validar, com um ou mais campos)
* Guardar logs da transação e receber logs pós transação
* Comunicar os usuários com as informações das transações
* Receber os arquivos por upload ou um link para download do dataset

## Descrição do Serviço

Serviço responsável por receber os dados da operadora e validar se está dentro do formato do layout, levando em consideração, colunas, tipagem, formato, valores default. E retornar de forma automatizada um feedback rápido e claro para a operadora se está tudo ok, e caso não estiver identificar o que está errado, armazenar informações no banco e controlar histórico por usuário.

## 

## Diagrama



### Steps

1. **Input :** Registro e login de usuário (desacoplado).  
   **output :** Redirect para o front.
2. **Input :** Upload de csv’s via front para validação dos dados.  
   **output :** Protocolo da validação e confirmação de início da validação.
3. **Input :** Requisição passando csv’s via api para validação dos dados.  
   **output :** Protocolo da validação e confirmação de início da validação.
4. **Input :** Comunicação do frontend com o backend enviando arquivos para validação.  
   **output :** Comunicação do backend com o frontend enviando histórico das validações e disponibilizando logs legíveis com os erros.
5. **Input :** Validação dados dados utilizando o layout.
6. **Input :** Armazena no banco validação e resultado.  
   **output :** Retorna histórico de validações.
7. **output :** Caso a validação não seja aprovada notifica o usuário por email passando o protocolo e retorna para o histórico no frontend o erro.
8. **output :** Caso a validação seja aprovada notifica o etl do dictas com protocolo.
9. **input :** Com base no protocol vai até o banco para pegar extrair os dados.

## Possíveis Problemas

* Tamanho dos arquivos (muito grandes), uma solução seria quebrar em chunks, e limitar o tamanho do arquivo quando feito o upload.
* Layout inadequado, a validação refletiria os mesmos problemas.

## Observações

* Operadoras com equipes de tecnologia consomem a API, operadoras menores usam o serviço pelo frontend.
* Log após análise seria muito bem detalhado tornando claro para o cliente o erro nos dados.

## Análise de Solução

Depois de algumas pesquisas na internet, não existem muitas soluções prontas.

Encontrei uma lib que se chama **MessyTables**, porém já é um pouco antiga e tentei brevemente utilizar seguindo sua documentação, porém não obtive sucesso.

<https://messytables.readthedocs.io/en/latest/>

Pandas - Schema (verificar)

**Leitura de Arquivos: Pandas / Dask**

Boa parte dos exemplos encontrados na internet para manipular e validar valores de um CSV vem da utilização da biblioteca Pandas. No entanto, após análise de benchmarking, utilizar a biblioteca Dask pode ser uma melhor opção, já que é baseada em Pandas, porém mais performática, segundo benchmarkings de alguns leitores CSV do Python:

<https://medium.com/casual-inference/the-most-time-efficient-ways-to-import-csv-data-in-python-cc159b44063d>

**Validação e limpeza dos dados: funções Python**

Alguns exemplos de limpeza e validação de dados oriundos de arquivos neste artigo:

<https://towardsdatascience.com/data-cleaning-with-python-and-pandas-detecting-missing-values-3e9c6ebcf78b>

**API: Engine de Validação**

- **get\_dtype\_csv**: Retorna Data Types (Adivinhar tipo de campos)

- **get\_duplicated**: Retorna Linhas duplicadas, pode ser com todas as colunas ou baseadas em chaves pré-defindas;

- **get\_nulls**: Retornar linhas com campos que não podem ser null

**- valid\_col\_regex:** Retornar linhas inválidas com base em regex (pré-definida)

- Gerar arquivo corrigido, eliminando as linhas inválidas?

- Arquivo corrigido, assumindo valores defaults para erros encontrados:

* data em branco = 01/01/2019
* valor em branco = usar valor médio

Celery vs (Python assync stack line libs)

**Interface**

* API: Flask (http)
* Front-End: Flask + Front (Bootstrap ou outro)

Softplan

* **Leitura arquivo JSON (nome dos campos, tipos, tamanho, nullable, chave, expressão)**
* Salvar parametrização (schema / format) em banco de dados;

Operadora:

* Tela Upload Arquivo (local para servidor)
* Campo URL de Arquivo: (download arquivo em servidor)
* Combo-Box para selecionar schemas pré-definidos (custo\_v01, custo\_v02, receitas\_01);

2) Retorna arquivos de log (mostrar na tela)

3) Performance arquivos grandes (quebrar arquivos e processar em paralelo)

4) Notificação quando a validação estiver concluída ???

**Persistência**

Postgresql (psycopg2)

- Armazenar histórico de análises de arquivos (principais informações, nome e tamanho arquivo, qtd linhas, qtd de erros por tipo, log de referência)

https://stackabuse.com/working-with-postgresql-in-python/

https://www.dataquest.io/blog/loading-data-into-postgres/

Status: 2 check-points por semana?

Escopo: incremental - atacar primeiro a engine?

Prazo:

* Primeiro leitura do JSON (SCHEMA) e validações iniciais, gerar log. Formato pré-definido em arquivo (Receita). Validações básicas iniciais.