# Sugestões de Refatoração - Processadores de Dados RF

## Contexto

Os arquivos `estabelecimento.py`, `socio.py` e `simples.py` compartilham muita lógica similar e têm oportunidades significativas de refatoração para melhorar manutenibilidade e reduzir duplicação de código.

## Problemas Identificados

1. **Duplicação de Código**
    - Funções de processamento quase idênticas nos três arquivos
    - Sistema de logging repetido
    - Funções de extração de ZIP duplicadas
    - Tratamento de erros inconsistente
2. **Inconsistências**
    - Sistema de fila presente em `socio.py` e `simples.py`, mas não em `estabelecimento.py`
    - Diferentes abordagens para otimização de memória
    - Variação no tratamento de erros entre arquivos

## Sugestões de Melhorias

### 1. Criar Classe Base Abstrata

```python
from abc import ABC, abstractmethod

class BaseProcessor(ABC):
    def __init__(self, path_zip: str, path_unzip: str, path_parquet: str):
        self.path_zip = path_zip
        self.path_unzip = path_unzip
        self.path_parquet = path_parquet
        self.logger = logging.getLogger(self.__class__.__name__)

    @abstractmethod
    def apply_transformations(self, df: pl.DataFrame) -> pl.DataFrame:
        """Aplica transformações específicas do processador"""
        pass

    def process_file(self) -> bool:
        """Implementação comum do processamento"""
        pass
```

### 2. Módulo de Utilidades Comum

```python
# utils/processing.py
class ProcessingUtils:
    @staticmethod
    def extract_file_parallel(zip_path: str, extract_dir: str) -> bool:
        """Extração paralela de arquivos ZIP"""
        pass

    @staticmethod
    def create_parquet(df: pl.DataFrame, config: ParquetConfig) -> bool:
        """Criação padronizada de arquivos Parquet"""
        pass
```

### 3. Sistema de Fila Unificado

```python
# queue_manager.py
class ProcessingQueueManager:
    def __init__(self):
        self._processing_lock = Lock()
        self._active_processes = Value('i', 0)
        self._process_queue = PriorityQueue()

    def add_to_queue(self, item: ProcessingItem):
        pass

    def process_queue(self):
        pass
```

### 4. Processadores Específicos

```python
class EstabelecimentoProcessor(BaseProcessor):
    def apply_transformations(self, df: pl.DataFrame) -> pl.DataFrame:
        # Transformações específicas para estabelecimentos
        pass

class SocioProcessor(BaseProcessor):
    def apply_transformations(self, df: pl.DataFrame) -> pl.DataFrame:
        # Transformações específicas para sócios
        pass

class SimplesProcessor(BaseProcessor):
    def apply_transformations(self, df: pl.DataFrame) -> pl.DataFrame:
        # Transformações específicas para Simples Nacional
        pass
```

## Benefícios Esperados

1. **Redução de Código**

   - Eliminação de código duplicado
   - Centralização de lógica comum
   - Melhor organização do código

2. **Manutenibilidade**
   - Mudanças podem ser feitas em um único lugar
   - Mais fácil adicionar novos processadores
   - Testes mais simples de implementar

3. **Consistência**
   - Tratamento de erros padronizado
   - Logging uniforme
   - Comportamento previsível

4. **Performance**
   - Otimizações podem ser aplicadas globalmente
   - Melhor gerenciamento de recursos
   - Facilidade para implementar melhorias

## Próximos Passos

1. **Fase 1: Preparação**
   - [ ] Criar estrutura base de classes
   - [ ] Implementar módulo de utilidades
   - [ ] Definir interfaces comuns

2. **Fase 2: Migração**
   - [ ] Migrar estabelecimento.py
   - [ ] Migrar socio.py
   - [ ] Migrar simples.py

3. **Fase 3: Otimização**
   - [ ] Implementar sistema de fila unificado
   - [ ] Adicionar testes automatizados
   - [ ] Otimizar uso de memória

4. **Fase 4: Documentação**
   - [ ] Documentar classes e métodos
   - [ ] Criar exemplos de uso
   - [ ] Atualizar README

## Notas Adicionais

- Manter compatibilidade com código existente durante migração
- Implementar gradualmente para minimizar riscos

- Adicionar testes antes de grandes mudanças
- Considerar feedback dos usuários do sistema

## Impacto na Base de Código

- **Antes**: ~2000 linhas por arquivo
- **Depois**: ~500 linhas por arquivo + código compartilhado
- **Redução**: ~60% de código duplicado

## Riscos e Mitigações

1. **Risco**: Quebra de funcionalidade existente
    - **Mitigação**: Testes extensivos antes/depois
2. **Risco**: Complexidade aumentada
    - **Mitigação**: Documentação clara e exemplos
3. **Risco**: Tempo de migração
    - **Mitigação**: Implementação gradual