Documento de Análise e Estratégia de Refatoração do Projeto

1. Code Smells e Problemas Detectados

A análise do código-fonte revelou pontos críticos ("code smells") que afetam a segurança, performance, robustez e manutenibilidade da aplicação.

1.1. Segurança (Risco Crítico)

- **Credenciais Expostas (Hardcoded Secrets):** As credenciais de banco de dados (server, username, password) estão fixadas no código. Isso é uma falha grave de segurança, expondo o banco de dados a qualquer um com acesso ao código.

1.2. Performance (Risco Alto)

- **Iteração Lenta (Inefficient Operation):** O uso de pedagios_df.apply(..., axis=1) para as funções verificar_mdfe_SP e verificar_mdfe_SP_Atua é um gargalo de performance. O Pandas itera linha por linha em Python, executando duas consultas SQL (ao DuckDB) para *cada linha* da planilha, o que é extremamente lento para arquivos grandes.
- **Consulta SQL Desnecessária a Cada Upload:** A função busca_eixos consulta o banco de dados SQL Server *ao vivo* toda vez que um arquivo é enviado. O cadastro de eixos de veículos é um dado que raramente muda, tornando essa consulta repetitiva e desnecessária.

1.3. Robustez e Boas Práticas (Risco Médio)

- **Obsessão Primitiva (Primitive Obsession) e Risco de SQL Injection:** As funções verificar_mdfe_SP e verificar_mdfe_SP_Atua constroem consultas SQL usando f-strings (ex: ...WHERE mdfe.placa = '{placa}'). Embora o alvo seja um DuckDB em memória, esta é uma prática perigosa que pode falhar com caracteres especiais ou abrir brechas de segurança.
- **Falta de Tratamento de Erros:** A função processar_planilha_sem_parar assume que a planilha do Excel terá *exatamente* as colunas esperadas (PLACA, DATA, HORA, etc.). Se uma coluna estiver ausente, a aplicação falhará com um KeyError não tratado.
- **Lógica de UI Misturada (UI Logic in Business Logic):** A função read_excel_auto chama st.error. Funções de lógica de dados não devem ter conhecimento do framework de UI.

Elas deveriam lançar uma exceção, e a camada de UI deveria capturá-la.

1.4. Manutenibilidade e Acoplamento (Risco Médio/Baixo)

- **Função Longa (Long Method):** A função processar_planilha_sem_parar é um "método deus" que viola o Princípio da Responsabilidade Única (SRP). Ela lê o arquivo, limpa dados, busca dados externos, aplica duas lógicas de negócio complexas e calcula finanças.
- **Variáveis Globais (Global Variables):** O uso de engine e duckdb_conn como variáveis de escopo global cria um acoplamento implícito, dificultando o teste e a modularização do código.
- **"Números Mágicos":** O código usa valores literais para lógica de negócio (ex: InSitSefaz = 100, CATEG == 61). Isso torna o código difícil de entender para um novo desenvolvedor.

---

2. Estratégia de Refatoração (Ferramentas e Métodos)

Para solucionar os problemas acima, propomos a seguinte estratégia:

2.1. Ferramentas Propostas

- **Streamlit Secrets (st.secrets):** Para externalizar as credenciais do banco de dados de forma segura, movendo-as para um arquivo .streamlit/secrets.toml.
- **Streamlit Cache (@st.cache_data):** Para aplicar cache nas funções de acesso a dados que não mudam frequentemente (ex: busca_eixos e as cargas iniciais de MDF-e).
- **DuckDB (Otimizado):** A ferramenta será usada de forma mais eficiente, carregando a planilha do Excel *para dentro* do DuckDB e executando uma única consulta vetorial.
- **Pytest:** Para a implementação de testes unitários e de integração.
- **Poetry (ou requirements.txt):** Para gerenciamento explícito e travamento de versões das dependências.
- **Linters (Ruff ou Flake8) e Formatadores (Black):** Para garantir a qualidade e a consistência do código, automatizando a detecção de erros de estilo e bugs comuns.

2.2. Métodos e Técnicas de Refatoração

1. **Segurança (Externalizar Credenciais):**
    - Remover as credenciais do código-fonte.
    - Acessá-las via st.secrets["db_credentials"]["username"].
2. **Performance (Operação Vetorial):**
    - As funções verificar_mdfe_SP e verificar_mdfe_SP_Atua (e o apply) serão **removidas**.
    - Após carregar pedagios_df, ele será registrado no DuckDB como uma tabela temporária (ex: pedagios_temp).
    - Toda a lógica de verificação será reescrita como uma **única consulta SQL** no DuckDB, usando LEFT JOIN / NOT EXISTS para cruzar pedagios_temp com as tabelas GTCMFESF e manifestos de forma massiva.
3. **Performance (Cache):**
    - Aplicar o decorador @st.cache_data(ttl=3600) (cache de 1 hora) nas funções que carregam os dados iniciais de MDF-e e na função busca_eixos.
4. **Robustez (Remoção de SQL Injection e Tratamento de Erros):**
    - Ao mover a lógica para uma única consulta SQL (item 2), o risco de injeção desaparece, pois os dados do usuário (planilha) já estarão dentro do banco como uma tabela.
    - Envolver o processamento principal em um bloco try...except KeyError as e: para capturar erros de colunas ausentes e informar o usuário via st.error.
5. **Manutenibilidade (Desacoplamento e Legibilidade):**
    - **Extrair Método:** Quebrar a função processar_planilha_sem_parar em funções menores (veja a Interface Fluente, item 4).
    - **Substituir "Número Mágico" por Constante:** Definir constantes no topo do script (ex: SIT_SEFAZ_AUTORIZADO = 100, MAPA_CATEGORIAS = {61: 7, ...}).
    - **Injeção de Dependência:** Passar as conexões (engine, duckdb_conn) como parâmetros para as funções que as utilizam, em vez de depender de variáveis globais.

---

3. Arquitetura do Projeto

Arquitetura Atual: Monolito Scriptado
O projeto é um único script (.py) que mistura todas as responsabilidades: Configuração, Acesso a Dados, Lógica de Negócio e Apresentação (UI).
Arquitetura Refatorada Proposta: 3 Camadas (SoC)
Propomos a separação do projeto em arquivos com responsabilidades claras (Separation of Concerns):

1. **app.py (Camada de Apresentação):**
    - Responsável *apenas* pela UI do Streamlit (st.*).
    - Importa funções das outras camadas.Trata exceções e exibe erros para o usuário.

- Contém a lógica de processamento, regras de negócio e cálculos (ex: a classe PedagioPipeline).
- Não tem conhecimento da existência do Streamlit.
- Contém todas as interações com bancos de dados (SQL Server e DuckDB).
- Funções como get_db_engine, load_mdfe_data, get_vehicle_eixos.
- Aplica cache (@st.cache_data) nessas funções.
