RELATÓRIO FINAL DO PROJETO

Grupo: SEM NOME III

Integrantes: Bernardo Moschen, Alan Oliveira, Christian Souza, Luis Fernando, Robson Klug

Data: Outubro/2025

1. Resumo Executivo

Este relatório consolida a proposta, o pitch e o repositório do projeto 'Fiscal Document Agent', um agente inteligente apoiado por LLM para processamento, validação, classificação e arquivamento de documentos fiscais brasileiros (NFe, NFCe, CTe, MDF-e).

O objetivo é reduzir esforço operacional, minimizar erros e garantir conformidade fiscal, oferecendo uma UI simples (Streamlit), armazenamento estruturado (SQLite/SQLModel) e recursos de exportação e auditoria.

2. Identificação do Projeto

Nome do Grupo: SEM NOME III

Integrantes: Bernardo Moschen , Alan Oliveira, Christian Souza, Luis Fernando, Robson Klug

Repositório GitHub:
https://github.com/BernardoMoschen/i2a2_agent_final_project/tree/main

3. Problema e Objetivos

Problema: Processos manuais de gestão documental fiscal geram retrabalho, risco e custos.

Objetivos Específicos:

• Automatizar parsing e normalização de XMLs fiscais.

• Validar documentos com regras declarativas e report estruturado.

• Classificar por centro de custo/categoria com suporte de LLMs.

• Armazenar dados normalizados + XML bruto com trilha de auditoria.

• Disponibilizar relatórios CSV/XLSX e visualizações.

• Viabilizar operação por meio de uma UI simples e segura.

4. Público-Alvo

• Departamentos Contábil/Financeiro/Administrativo de empresas.

• Escritórios de contabilidade e BPO financeiro.

• Autônomos e PMEs que lidam com grande volume de XMLs.

5. Justificativa e Valor Agregado

A automação reduz tempo de digitação e conferência, diminui erros e fortalece a conformidade. Com trilha de auditoria e logs com anonimização de PII, a solução melhora governança e prepara o ambiente para auditorias internas/externas.

6. Detalhamento da Solução

6.1 Funcionalidades Principais

1) Parsing/Normalização: defusedxml/lxml + modelos Pydantic.

2) Validação Fiscal: motor de regras com issues estruturadas.

3) Classificação (LLM): mapeamento de centro de custo/categoria + score de confiança.

4) Persistência: SQLite + SQLModel/SQLAlchemy (dados normalizados + XML).

5) Arquivamento: diretórios organizados com metadados.

6) Relatórios: CSV/Excel; gráficos via matplotlib.

7) UI Streamlit: upload (arquivo ou ZIP), chat, validação e dashboards.

8) Auditoria: logging completo + redação de PII.

6.2 Tecnologias e Dependências

• Linguagem: Python 3.11+

• Framework LLM: LangChain (com integração Gemini)

• Banco: SQLite + SQLModel/SQLAlchemy

• UI: Streamlit

• XML: defusedxml, lxml

• Dados/Relatórios: pandas, matplotlib, openpyxl

• Qualidade/CI: black, isort, ruff, mypy, pytest

6.3 Arquitetura Lógica

Componentes:

• UI (Streamlit): upload, visualização, chat.

• Serviço de Parsing/Validação: pipeline OCR/parse → validações.

• Serviço de Classificação: prompt + few-shot + políticas; devolve classe e confiança.

• Banco de Dados: persistência de metadados e dados normalizados.

• Arquivo (Object Storage local): arquivos XML/PDF arquivados.

• Módulo de Relatórios: geração de CSV/XLSX e gráficos.

Fluxo de Dados (alto nível):

[Usuário/UI] -> Upload XML/ZIP -> [Parser] -> [Validador] -> [Classificador (LLM)] -> [Banco + Arquivo] -> [Relatórios/Visualizações] -> [Chat/Consulta]

6.4 Estrutura de Pastas (proposta/README)

src/models – modelos Pydantic (InvoiceModel, ValidationIssue etc.)

src/tools – ferramentas do agente (xml_parser, validator, classifier)

src/agent – núcleo do agente e orquestração LangChain

src/database – schemas e operações

src/ui – app Streamlit (upload, painéis, chat)

src/utils – utilidades compartilhadas

tests – testes unitários e de integração

archives – arquivos fiscais arquivados

reports – relatórios gerados

6.5 Operação (Setup e Uso)

Pré-requisitos: Python 3.11+, pip/uv.

Instalação:

python -m venv venv

source venv/bin/activate # Windows: venv\Scripts\activate

pip install -r requirements.txt

Variáveis de ambiente (.env): GEMINI_API_KEY, DATABASE_URL (sqlite:///fiscal_documents.db), ARCHIVE_DIR, LOG_LEVEL.

Execução da UI: streamlit run src/ui/app.py (porta 8501).

Testes: pytest; cobertura: pytest --cov=src --cov-report=html.

Qualidade: black, isort, ruff, mypy.

7. Tabelas e Quadros

Módulo	Responsabilidades	Entradas	Saídas
Parser	Ler e normalizar XML	XML NFe/NFCe/CTe/MDF-e	Modelos Pydantic + metadados
Validador	Regras fiscais declarativas	Modelos normalizados	Lista de issues com severidade
Classificador (LLM)	Centro de custo / categoria	Metadados + texto	Classe + score confiança
Persistência	Salvar dados + XML	Modelos + arquivos	SQLite + diretórios/arquivos
Relatórios	Exportar/visualizar	Dados normalizados	CSV/XLSX + gráficos
UI/Chat	Interação humana	Uploads/consultas	Resultados, logs e respostas

Indicador	Definição	Meta (MVP)
Tempo médio por doc	Processamento (parse→classificar)	< 1.5s/doc em lote small
Acurácia de classe	% documentos com classe correta	>= 92% (com revisão humana)
Taxa de erro parse	% XML com erro de leitura	< 2%
Cobertura de testes	% linhas cobertas	>= 70%
Lead time lote	Tempo para 1k XMLs	< 25 min em notebook padrão

8. Diagramas (Descrição + PlantUML)

Componentes (PlantUML):

```
@startuml
title Arquitetura de Componentes - Fiscal Document Agent
actor Usuario
node "UI (Streamlit)" as UI
node "Parser/Validator" as PV
node "Classificador LLM" as LLM
database "SQLite (SQLModel)" as DB
folder "Arquivos (archives/)" as FS
node "Relatórios" as REP

Usuario --> UI : upload/consulta
UI --> PV : XML/ZIP
PV --> LLM : texto/metadados
```

LLM --> PV : classe + confiança
PV --> DB : save modelos
PV --> FS : store XML
UI --> REP : gerar CSV/XLSX/gráficos
REP --> DB : consulta agregações
@enduml

9. Segurança e Compliance

• Sem segredos no código; uso de variáveis de ambiente.

• Parsing seguro (defusedxml) para evitar XXE.

• Redação de PII em logs por padrão.

• Validações de entrada com Pydantic.

• Logging estruturado e trilha de auditoria.

10. Estratégia de Testes

Testes unitários: parser, validador, classificador e operações de DB (LLM mock).

Testes de integração: fluxo E2E com XML de amostra.

CI local: format/lint/typecheck + pytest; cobertura exportada em HTML.

11. Resultados Esperados e ROI

Economia de tempo administrativo e redução de erros. ROI estimado em até 5 meses, dependendo do volume.

Mês	Economia Mensal (R$)	Economia Acumulada (R$)	Custo Acumulado (R$)
0	0	0	10.000
1	2.000	2.000	10.000
2	2.000	4.000	10.000
3	2.000	6.000	10.000
4	2.000	8.000	10.000
5	2.000	10.000	10.000 (Break-even)
12	2.000	24.000	10.000

12. Operação, Monitoramento e Manutenção

• Logs estruturados com níveis (LOG_LEVEL).

• Métricas: tempo de processamento, acurácia de classificação, taxa de erros de parsing.

• Backups: diretório archives/ e base SQLite.

• Versionamento: Git e convenção de branches.

• Roadmap: melhorias de modelo, integrações ERP, autenticação e perfis de acesso.

13. Conclusão

O projeto consolida boas práticas de engenharia de dados e IA aplicada, com foco em valor prático ao negócio. A arquitetura proposta favorece evolução incremental (MVP → integrações/escala) mantendo segurança e governança.

14. Anexos

A. Slides do Pitch (síntese anexada no dossiê original).

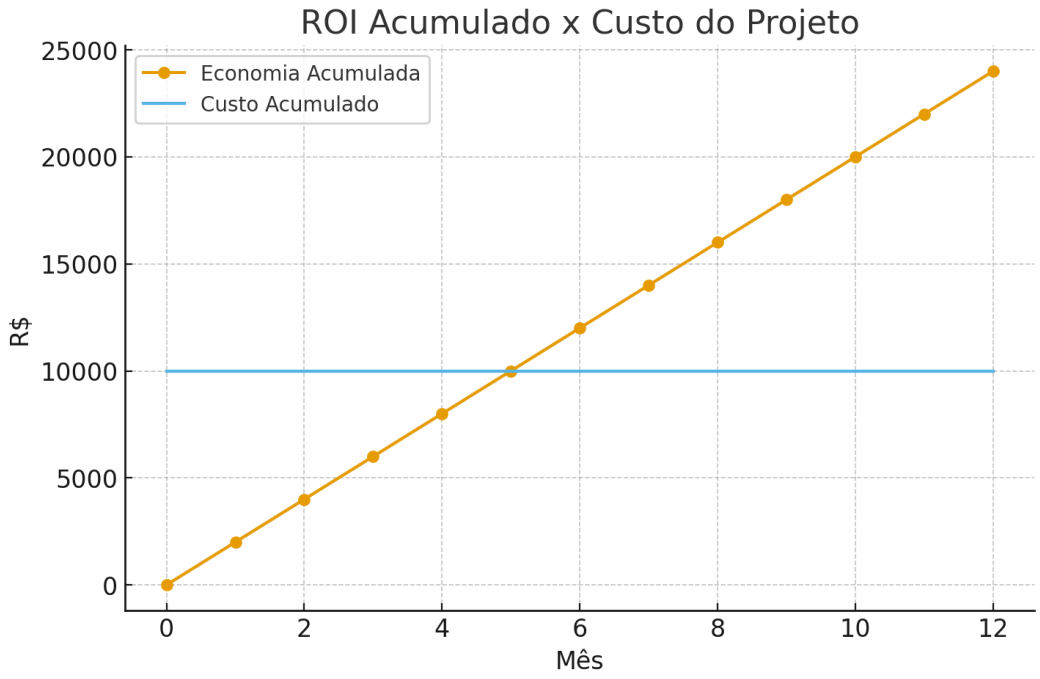
B. Proposta de Projeto (PDF) – fontes de ROI, público-alvo e escopo.

C. Estrutura completa do repositório e dependências (requirements.txt).

15. Gráficos de ROI e Economia

A seguir, os gráficos gerados a partir da análise de ROI e economia mensal estimada:

15.1 ROI Acumulado x Custo do Projeto



15.2 Economia Mensal Estimada