Desafio Final: Assistente de IA Consultor Especializado

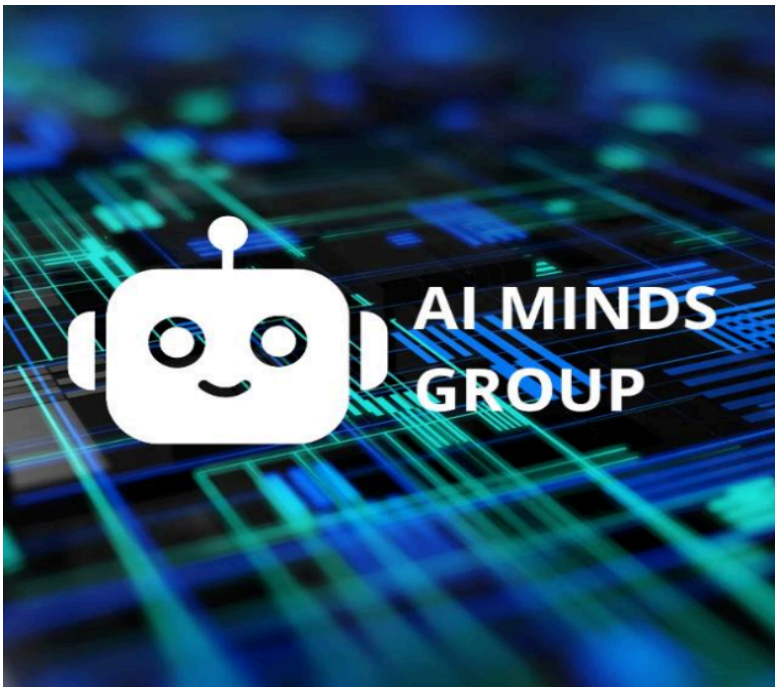
Curso Agentes Autônomos - Fase 6

Prazo de entrega: 29/10/2025

Grupo AI Minds - 2025

SUMÁRIO

Nome do Grupo



Integrantes do Grupo

Nome	E-mail
Aldenir Gil de Oliveira	aldenir.gil@gmail.com
Gustavo da Silva Câmara	gustavo.guky@gmail.com
Ieda Ferreira Alves Flock	iedabcc@gmail.com
Roberto dos Santos Rocha **(Líder)**	rsantos.rocha@gmail.com
Vinícius Alves dos Santos	vinicius.alvs@hotmail.com
Simon Flock Schiesl Ferreira	simflock@gmail.com

Descrição do Tema

Criação de um agente inteligente capaz de automatizar validações, responder perguntas em linguagem natural, gerar relatórios gerenciais e fornecer orientações estratégicas, facilitando o dia a dia dos escritórios contábeis e agregando valor aos seus clientes, que são pequenas e médias empresas (PMEs).

Público-alvo do projeto

Escritórios Contábeis de Pequeno e Médio Porte.

Justificativa do Tema

A rotina dos escritórios contábeis (BPOs) é marcada por um alto volume de documentos fiscais, equipe muitas das vezes reduzida, e necessidade constante de atualização sobre legislação e procedimentos tributários ou fiscais. As principais dores desse público são:

- Volume alto de notas fiscais vs equipe enxuta.
- Erros manuais na escrituração, que geram retrabalho e multas.
- Necessidade de orientação rápida sobre CFOP, CST, NCM e regimes tributários distintos por cliente.
- Pressão por eficiência e expansão de escala sem aumento de custos operacionais.

Esses desafios afetam diretamente a reputação, a rentabilidade e a capacidade de crescimento dos escritórios contábeis. Um assistente consultor especializado em IA pode transformar o cenário, automatizando tarefas críticas, reduzindo erros e liberando o contador para atuar de forma mais consultiva e estratégica.

Agregação de valor

Objetivos principais do assistente

Suporte para dúvidas e decisões estratégicas.

Informações sobre contabilidade e tributação.

Funcionalidades do Agente

Validação instantânea de documentos fiscais: Identificação automática de inconsistências em campos como CFOP, CST, NCM antes do lançamento.

Respostas em linguagem natural

O agente responde dúvidas comuns (“o CFOP do cliente X está correto?”) fundamentando-se na legislação vigente.

Geração automática de relatórios gerenciais e comparativos

Relatórios prontos para envio mensal ao cliente, facilitando o acompanhamento e a tomada de decisão.

Alertas proativos

Notificações sobre obrigações acessórias, prazos e mudanças legislativas relevantes.

Checklists inteligentes

Auxílio no fechamento contábil e fiscal, reduzindo a dependência de especialistas humanos.

Elementos Adicionais

Tabela: Dores e Soluções

Dores da Área Fiscal/Contábil	Solução com Agente IA Especializado
Legislação complexa e mutável	Atualização automática e alertas personalizados
Erros de classificação/apuração	Conferência automatizada e validação em tempo real
Falta de automação	Execução automática de tarefas repetitivas
Perda de oportunidade fiscais	Análise de dados e sugestões de economia tributária
Riscos com obrigações acessórias	Controle de prazos e envio automático de declarações
Falta de informação para decisões	Relatórios claros e insights estratégicos automatizados

Legenda dos Códigos Fiscais

Código Fiscal	Significado	Função
CFOP	**Código da Operação e Prestações**	É um sistema de numeração com 4 dígitos, com finalidade de identificar operações e prestações em documentos fiscais eletrônicos, escriturações de livros fiscais e obrigações acessórias.
CST	**Código da Situação Tributária**	É representado por uma combinação de 3 números com a finalidade de demonstrar a origem de um produto e determinar a forma de

		tributação que incidirá sobre ele.
NCM	**Nomenclatura Comum do Mercosul**	É utilizado para fins de controle estatístico do comércio exterior e, principalmente, para determinar quais impostos federais (como IPI, PIS, COFINS e Imposto de Importação) e estaduais (ICMS) incidem sobre o produto específico.
CEST	**Código Especificador da Substituição Tributária**	Criado em 2019 para padronizar e identificar as mercadorias que se encaixam nos regime de substituição tributária e de antecipação do ICMS.

Nota: Novas tabelas ou recursos visuais serão utilizados ao longo do relatório.

Fluxo de uso do agente

1. Usuário faz uma pergunta ou solicita validação de documento.

2. O agente processa a solicitação, consulta a base de dados e retorna a resposta ou relatório.

3. O agente envia alertas automáticos conforme regras pré-definidas (exemplo: vencimento de obrigações).

4. Todas as interações são registradas para acompanhamento e melhoria contínua.

Esquema simplificado:

[Usuário] → [Agente IA] → [Consulta Base de Dados/Legislação] → [Resposta/Relatório/Alerta] → [Usuário]

Resultado Esperado

- Menos multas e retrabalho
- Mais previsibilidade de caixa para o escritório e seus clientes
- Liberação de horas-analista para iniciativas de maior valor agregado
- Transformação do escritório contábil de operacional para consultivo

O Desenvolvimento Front End

O Frontend do projeto possui uma interface de usuário construída para interagir com o agente IA fiscal e visualizar os dados processados, focando na usabilidade para o MVP (Produto Mínimo Viável). Em suma, o sistema FrontEnd, batizado pelo grupo como FiscalMind, constitui um sistema inteligente de análise de notas fiscais e dados contábeis.

O Framework

Foi utilizado o React 18 como o principal framework. O TypeScript garantiu a tipagem e o Vite serviu como bundler e ambiente de desenvolvimento rápido, no entanto, o ambiente de execução principal para o frontend requer Node.js 18+.

As tecnologias utilizadas

Categoria	Recurso	Implementação
Interface/ Estilização	**shadcn/ui + Tailwind CSS**	Tailwind CSS é usado para estilização rápida e utilitária, e shadcn/ui fornece componentes de interface modernos. O componente de Tabela (shadcn/ui/table) foi especificamente implementado para exibir o relatório de Listagem de NF-e, incluindo paginação básica, se necessário. Componentes como o Calendar do shadcn foram usados para a

		seleção de data nos filtros.
Gerenciamento de Estado	**React Query (TanStack Query)**	Utilizado para gerenciar o estado assíncrono e o caching de dados, essencial para a integração com a API.
Roteamento	**React Router v6**	Responsável por gerenciar a navegação entre as diferentes seções da aplicação, como a rota /relatórios.
Cliente HTTP	**Axios**	Usado para fazer requisições HTTP. O Axios é configurado para funcionar com detecção automática de porta do backend (8000 ou 8001), eliminando a necessidade de configuração manual do host.

Estrutura do projeto

```
csv-chat-flow/
├── src/
│   ├── components/     # Componentes React
│   ├── hooks/          # Custom hooks
│   ├── lib/            # Utilitários e configurações
│   ├── pages/          # Páginas da aplicação
│   └── integrations/   # Integrações externas
├── docs/               # Documentação técnica
├── public/             # Arquivos estáticos
└── ...                 # Arquivos de configuração
```

```
src/
├── components/
│   ├── ui/          # Componentes shadcn (Button, Input, Table, etc.)
│   ├── ChatInterface.tsx
│   ├── FileUploader.tsx
│   ├── FilesList.tsx
│   ├── MetricsBar.tsx
│   └── CFOPValidator.tsx
├── lib/
│   └── axios.ts     # Configuração HTTP inteligente
└── services/
    └── apiService.ts # Serviços de API
```

Como a solução Front End foi implementada

Relembramos que o objetivo principal foi entregar um produto minimamente viável, abreviado como MVP. Assim, a estrutura da solução Frontend foi organizada em páginas/rotas (como a rota **/relatorios**) e componentes (como o **FileUploader.tsx**).

A implementação do MVP foi focada em atingir uma experiência do usuário (UX) Funcional que suportasse o fluxo fiscal essencial da seguinte forma:

1. Criação de Rotas e Páginas: Uma nova rota/página React (**/relatorios**) foi criada para abrigar a Seção Básica de Relatórios.

2. Visualização de Relatórios:

◦ Foi implementado um componente de Tabela (utilizando **shadcn/ui/table**) para exibir os dados do relatório de Listagem de NF-e.

◦ A interface de relatórios inclui seletores de data/filtro de período, implementados com componentes de seleção de data (como o **Calendar** do shadcn).

◦ Foi realizada a integração com a API do relatório de listagem, garantindo que os seletores de data funcionem e que as chamadas à API busquem os dados corretamente, tratando os estados de loading e erro.

3. Upload de Documentos Fiscais: O fluxo de upload foi ajustado para aceitar explicitamente arquivos .xml. A interface de upload (**FileUploader.tsx**) foi configurada para permitir a seleção/arrastar desses arquivos, fornecendo feedback visual claro.

4. Responsividade: Foi garantida a responsividade básica das novas telas para que sejam usáveis em desktop e não quebrem visualmente em telas menores.

5. Refatoração: Uma tarefa essencial (prioridade Média/Essencial) no Frontend foi a remoção de código/arquivos relacionados ao antigo Módulo de Detecção de Fraude (Cartão de Crédito) para simplificar e focar no core fiscal.

Segue abaixo um resumo sobre as principais melhorias efetuadas no frontend visando o MVP:

UX Funcional para MVP: A interface permite upload de XML de NF-e e interação básica com o chat e o relatório de listagem de forma clara.

Tarefa UI/UX - Ajustado fluxo de upload para aceitar XML explicitamente: Interface de upload (**FileUploader.tsx**) permite selecionar/arrastar arquivos .xml; feedback visual claro.

Tarefa Técnica - Adicionado filtro de período no Frontend para o relatório: Componentes de seleção de data (ex: Calendar do shadcn) adicionados à página de relatórios, valores são enviados à API.

Tarefa Técnica - Garantido responsividade básica das novas telas: Tela de relatório é usável em desktop, ou seja, não quebra visualmente em telas menores (pode ter scroll horizontal na tabela se necessário no MVP).

Funcionalidades gerais

As funções completas do sistema frontend estão discriminadas abaixo:

• Upload de arquivos (originalmente CSV de notas fiscais, mas **o MVP foca em XML de NF-e**).

• Chat interativo com IA para análise contábil e fiscal.

• Análise de impostos e apuração tributária.

• Geração de relatórios fiscais inteligentes.

• Visualização de arquivos processados e métricas em tempo real.

• Interface moderna e responsiva.

Funções do MVP (Produto Mínimo Viável):

Para o MVP, as funções essenciais da interface são:

1. Permitir o upload de XML de NF-e.

2. Permitir a interação básica com o chat.

3. Permitir que o usuário acesse e visualize o relatório de Listagem de NF-e, podendo aplicar filtros por período.

Execução do sistema:

• A aplicação Frontend é executada e na seguinte VPS: http://srv774816.hstgr.cloud:9001/

• **Pré-requisito:** a API do Backend **(agentnfe-backend**) precisa estar em execução.

Integração Avançada com Backend Python

Sistema de Detecção Automática de Backend

typescript

// lib/axios.ts - Conexão inteligente:

• Detecta automaticamente porta 8000 (Python) ou 8001

• Fallback estratégico se backend não responder

• Interceptors para retry automático em caso de falha

• Timeouts configuráveis por tipo de requisição

• Logs detalhados para debugging

Gestão de Estado e Sincronização

• Estado local para UI responsiva

• Sincronização em tempo real entre componentes

• Persistência session-based para dados temporários

• Error boundaries para falhas gracefully

Responsividade e UX Avançada

Design System Consistente

• Componentes shadcn/ui padronizados

• Cores semânticas (success, warning, error, info)

- Tipografia hierárquica para informações fiscais
- Espaçamento consistente com sistema de escala

Experiência Mobile-First

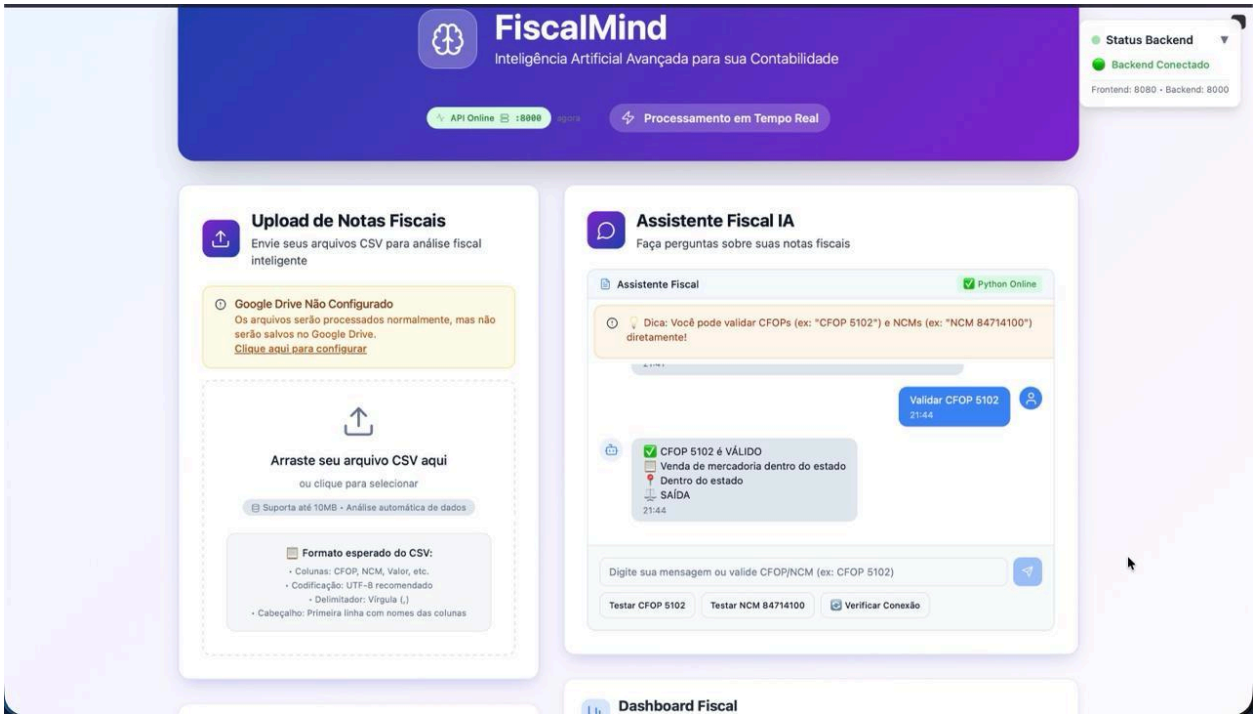
- Grids responsivos (1 coluna mobile, 2 tablet, 4 desktop)
- Touch targets ampliados para mobile
- Scroll horizontal em tabelas quando necessário
- Textos truncados com tooltips informativos

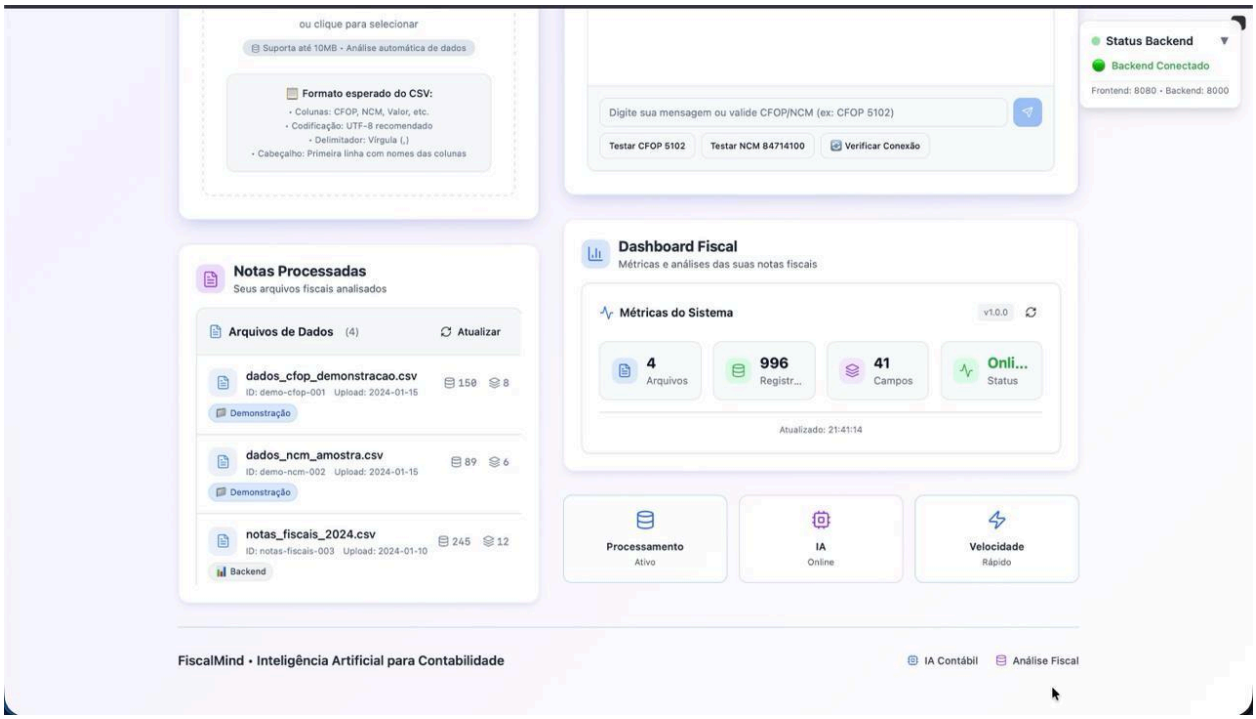
Integração Opcional

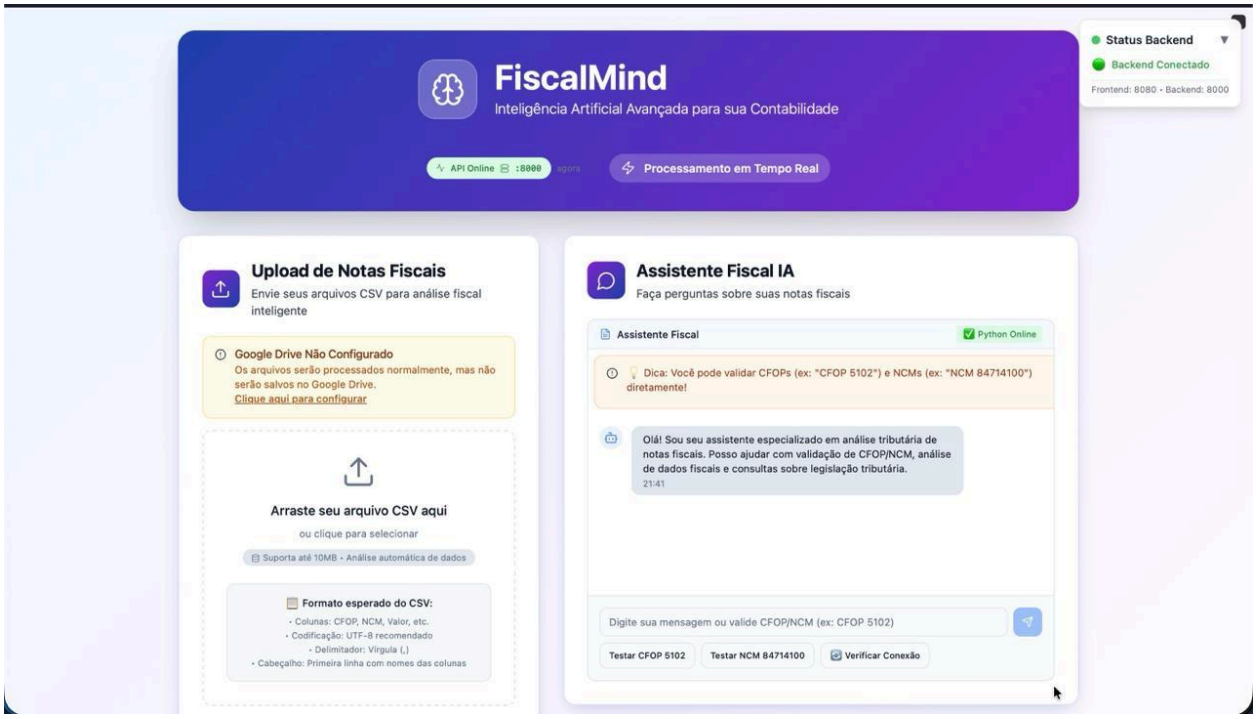
O sistema permite upload automático para o Google Drive, porém, essa integração não se encontra configurada pois requer credenciais no Google Cloud Console e variáveis de ambiente no arquivo **.env** para que os arquivos sejam salvos no Drive. De qualquer modo, essa integração é completamente dispensável visto que a aplicação funciona perfeitamente sem essa alternativa.

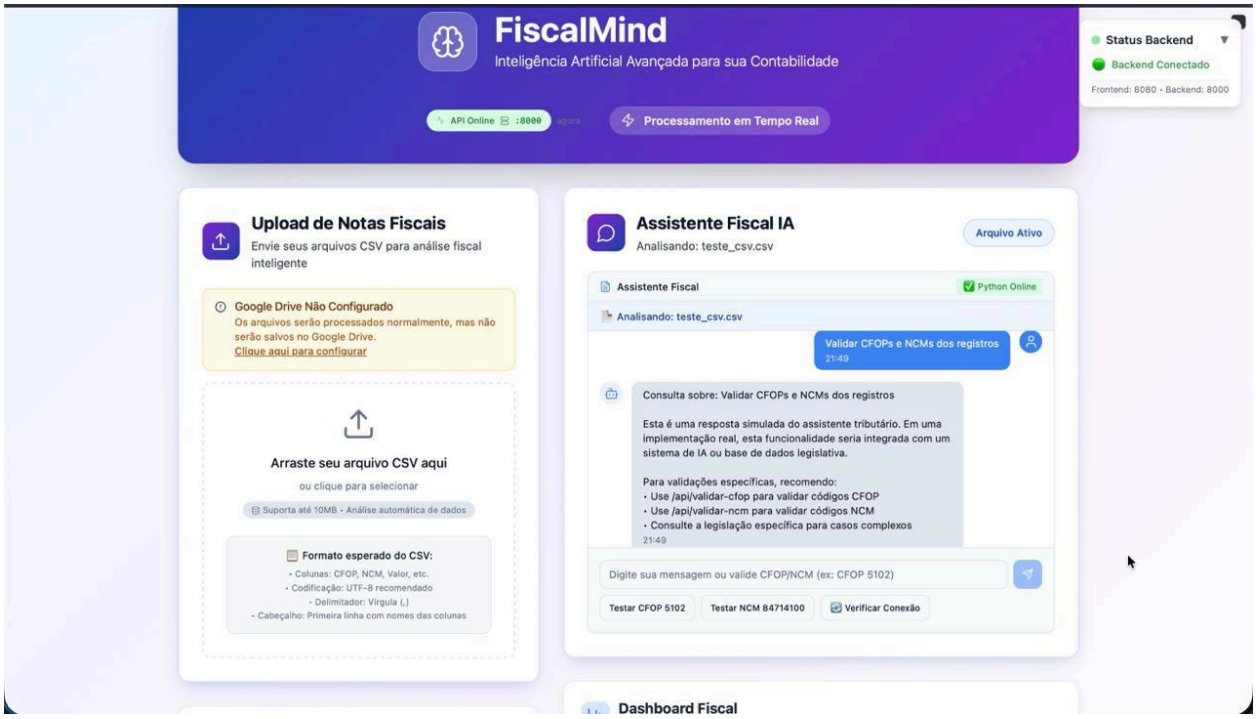
Segue abaixo demonstrações da interface do sistema:

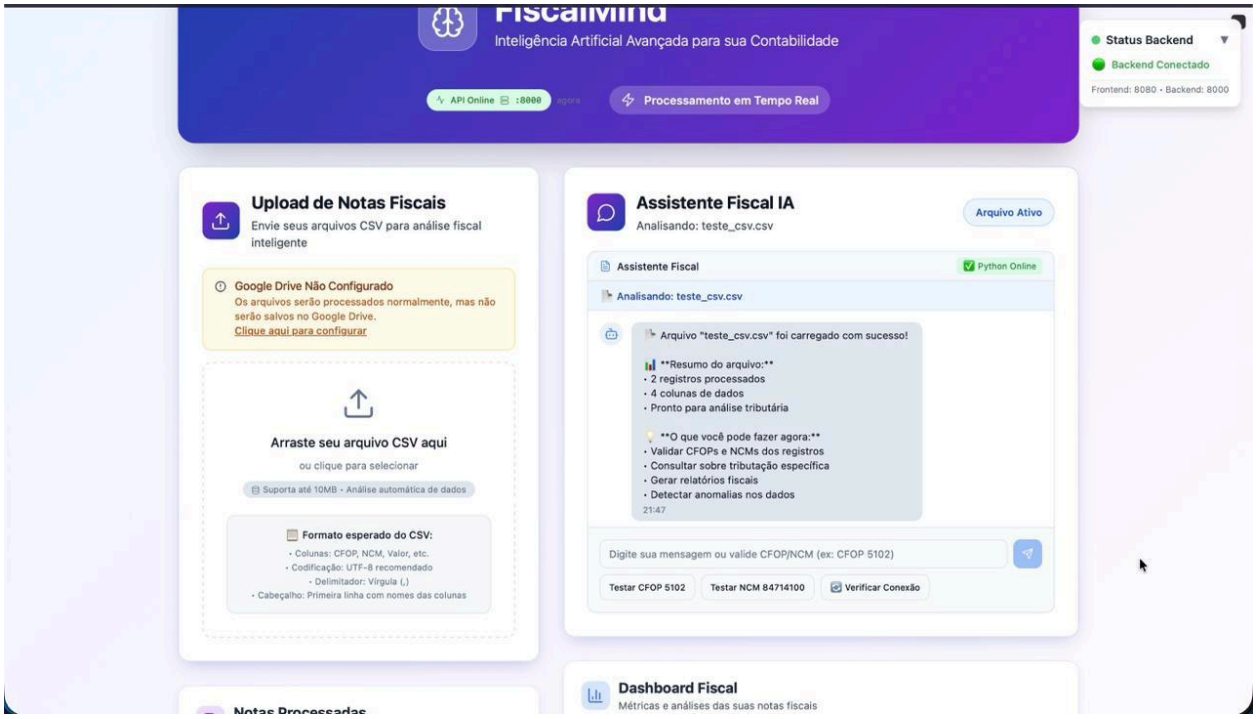
Link de acesso: http://srv774816.hstgr.cloud:9001

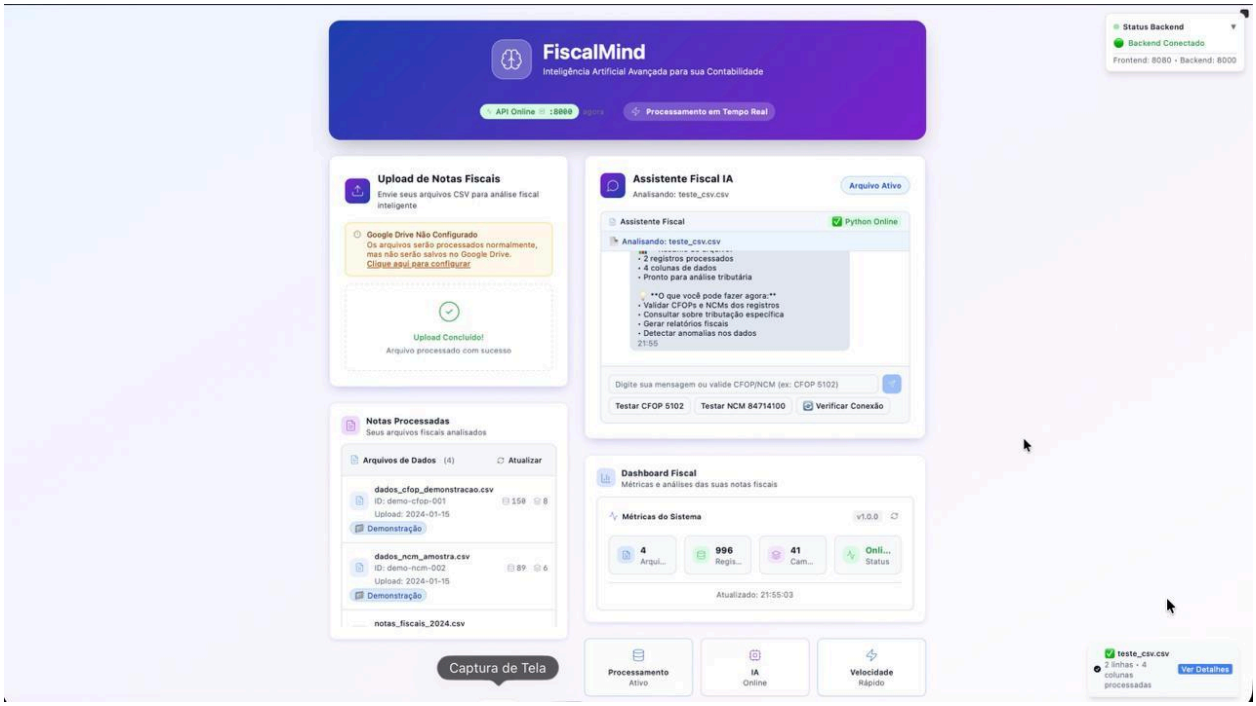

FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 - Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Arraste seu arquivo CSV aqui
ou clique para selecionar
Suporta até 10MB - Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Assistente Fiscal IA
Faça perguntas sobre suas notas fiscais
Assistente Fiscal
Python Online
Dica: Você pode validar CFOPs (ex: "CFOP 5102") e NCMs (ex: "NCM 84714100") diretamente!
Validar CFOP 5102
21:44
CFOP 5102 é VÁLIDO
Venda de mercadoria dentro do estado
Dentro do estado
SAÍDA
21:44
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Dashboard Fiscal

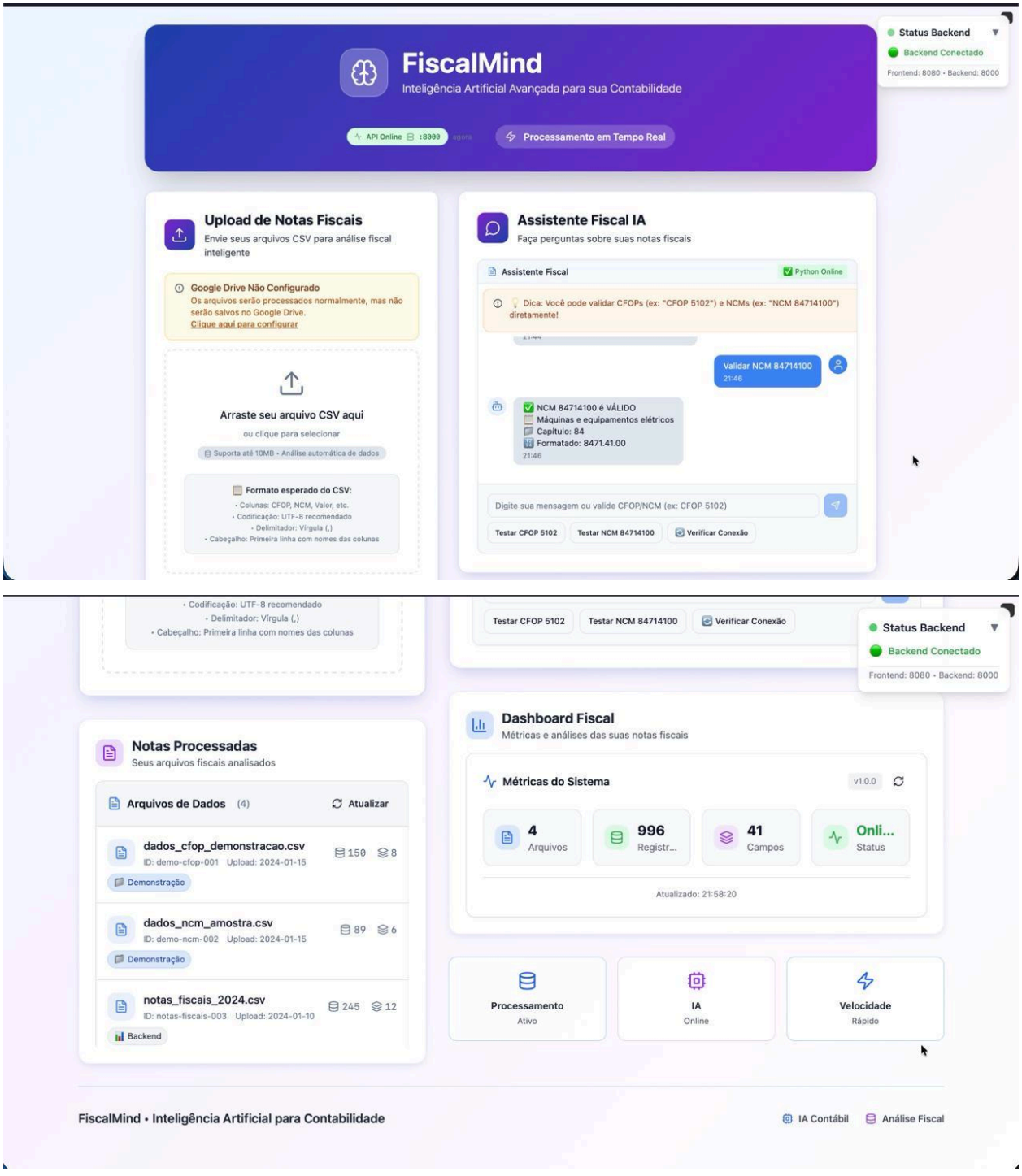

ou clique para selecionar
Suporta até 10MB - Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Status Backend
Backend Conectado
Frontend: 8080 - Backend: 8000
Notas Processadas
Seus arquivos fiscais analisados
Arquivos de Dados (4)
Atualizar
dados_cfop_demonstracao.csv
ID: demo-cfop-001 Upload: 2024-01-15
150 8
Demonstração
dados_ncm_amostra.csv
ID: demo-ncm-002 Upload: 2024-01-15
89 6
Demonstração
notas_fiscais_2024.csv
ID: notas-fiscais-003 Upload: 2024-01-10
245 12
Backend
Dashboard Fiscal
Métricas e análises das suas notas fiscais
Métricas do Sistema
v1.0.0
4
Arquivos
996
Registr...
41
Campos
Onli...
Status
Atualizado: 21:41:14
Processamento
Ativo
IA
Online
Velocidade
Rápido
FiscalMind • Inteligência Artificial para Contabilidade
IA Contábil
Análise Fiscal


FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 • Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Arraste seu arquivo CSV aqui
ou clique para selecionar
Suporta até 10MB • Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Assistente Fiscal IA
Faça perguntas sobre suas notas fiscais
Assistente Fiscal
Python Online
Dica: Você pode validar CFOPs (ex: "CFOP 5102") e NCMs (ex: "NCM 84714100") diretamente!
Olá! Sou seu assistente especializado em análise tributária de notas fiscais. Posso ajudar com validação de CFOP/NCM, análise de dados fiscais e consultas sobre legislação tributária.
21:41
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão


FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 • Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Arraste seu arquivo CSV aqui
ou clique para selecionar
Suporta até 10MB • Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Assistente Fiscal IA
Analisando: teste_csv.csv
Arquivo Ativo
Assistente Fiscal
Python Online
Analisando: teste_csv.csv
Validar CFOPs e NCMs dos registros
21:49
Consulta sobre: Validar CFOPs e NCMs dos registros
Esta é uma resposta simulada do assistente tributário. Em uma implementação real, esta funcionalidade seria integrada com um sistema de IA ou base de dados legislativa.
Para validações específicas, recomendo:
• Use /api/validar-cfop para validar códigos CFOP
• Use /api/validar-ncm para validar códigos NCM
• Consulte a legislação específica para casos complexos
21:49
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Dashboard Fiscal


FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 • Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Arraste seu arquivo CSV aqui
ou clique para selecionar
Suporta até 10MB • Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Assistente Fiscal IA
Analisando: teste_csv.csv
Arquivo Ativo
Assistente Fiscal
Python Online
Analisando: teste_csv.csv
Arquivo "teste_csv.csv" foi carregado com sucesso!
Resumo do arquivo:
• 2 registros processados
• 4 colunas de dados
• Pronto para análise tributária
O que você pode fazer agora:
• Validar CFOPs e NCMs dos registros
• Consultar sobre tributação específica
• Gerar relatórios fiscais
• Detectar anomalias nos dados
21:47
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Dashboard Fiscal
Métricas e análises das suas notas fiscais


FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 • Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Upload Concluído!
Arquivo processado com sucesso
Assistente Fiscal IA
Analisando: teste_csv.csv
Arquivo Ativo
Assistente Fiscal
Python Online
Analisando: teste_csv.csv
• 2 registros processados
• 4 colunas de dados
• Pronto para análise tributária
O que você pode fazer agora:
• Validar CFOPs e NCMs dos registros
• Consultar sobre tributação específica
• Gerar relatórios fiscais
• Detectar anomalias nos dados
21:55
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Notas Processadas
Seus arquivos fiscais analisados
Arquivos de Dados (4)
Atualizar
dados_cfop_demonstracao.csv
ID: demo-cfop-001
Upload: 2024-01-15
Demonstração
dados_ncm_amostra.csv
ID: demo-ncm-002
Upload: 2024-01-15
Demonstração
notas_fiscais_2024.csv
Dashboard Fiscal
Métricas e análises das suas notas fiscais
Métricas do Sistema
v1.0.0
4
996
41
Onli...
Status
Atualizado: 21:55:03
Processamento
Ativo
IA
Online
Velocidade
Rápido
Captura de Tela
teste_csv.csv
2 linhas • 4 colunas processadas
Ver Detalhes


FiscalMind
Inteligência Artificial Avançada para sua Contabilidade
API Online :8000
Processamento em Tempo Real
Status Backend
Backend Conectado
Frontend: 8080 • Backend: 8000
Upload de Notas Fiscais
Envie seus arquivos CSV para análise fiscal inteligente
Google Drive Não Configurado
Os arquivos serão processados normalmente, mas não serão salvos no Google Drive.
Clique aqui para configurar
Arraste seu arquivo CSV aqui
ou clique para selecionar
Suporta até 10MB • Análise automática de dados
Formato esperado do CSV:
• Colunas: CFOP, NCM, Valor, etc.
• Codificação: UTF-8 recomendado
• Delimitador: Vírgula (,)
• Cabeçalho: Primeira linha com nomes das colunas
Assistente Fiscal IA
Faça perguntas sobre suas notas fiscais
Assistente Fiscal
Python Online
Dica: Você pode validar CFOPs (ex: "CFOP 5102") e NCMs (ex: "NCM 84714100") diretamente!
Validar NCM 84714100
21:46
NCM 84714100 é VÁLIDO
Máquinas e equipamentos elétricos
Capítulo: 84
Formatado: 8471.41.00
21:46
Digite sua mensagem ou valide CFOP/NCM (ex: CFOP 5102)
Testar CFOP 5102
Testar NCM 84714100
Verificar Conexão
Notas Processadas
Seus arquivos fiscais analisados
Arquivos de Dados (4)
Atualizar
dados_cfop_demonstracao.csv
ID: demo-cfop-001 Upload: 2024-01-15
150 8
Demonstração
dados_ncm_amostra.csv
ID: demo-ncm-002 Upload: 2024-01-15
89 6
Demonstração
notas_fiscais_2024.csv
ID: notas-fiscais-003 Upload: 2024-01-10
245 12
Backend
Dashboard Fiscal
Métricas e análises das suas notas fiscais
Métricas do Sistema
v1.0.0
4
Arquivos
996
Registr...
41
Campos
Onli...
Status
Atualizado: 21:58:20
Processamento
Ativo
IA
Online
Velocidade
Rápido
FiscalMind • Inteligência Artificial para Contabilidade
IA Contábil
Análise Fiscal

O Desenvolvimento Back End

O AgentNFe Backend é um sistema multiagente robusto e escalável destinado à análise inteligente de datasets estruturados em formato CSV. A arquitetura central é baseada em Retrieval Augmented Generation (RAG), utilizando embeddings vetoriais armazenados no Supabase para responder a consultas complexas em linguagem natural.

O sistema atingiu a Versão 3.0, que se destaca pela eliminação de hard-coding massivo (cerca de 240 linhas de lógica condicional if/elif e listas fixas de palavras-chave). Essa refatoração permitiu que o sistema migrasse para uma orquestração puramente semântica e conduzida por LLM (LLM-driven).

Como resultado, o sistema oferece flexibilidade cognitiva (reconhecendo sinônimos e consultas mistas) e escalabilidade, adaptando-se a qualquer CSV sem modificação de código.

Objetivos do projeto backend

Objetivos Principais

1. Criar um sistema backend inteligente e escalável para permitir a análise automatizada de dados exploratórios (EDA) em Notas Fiscais Eletrônicas.

2. Permitir que usuários façam consultas complexas em linguagem natural sobre grandes volumes de dados fiscais em formato CSV, delegando a execução de análise estatística real a módulos especializados (PythonDataAnalyzer).

3. Garantir a conformidade arquitetural utilizando LangChain para orquestração e Supabase/pgvector como banco de dados vetorial.

Objetivos Específicos

1. Ingestão Eficiente de Dados:

- Processar arquivos CSV de NF-e (NotaFiscal e NotaFiscalItem)
- Chunking inteligente para otimizar contexto

- Geração de embeddings vetoriais para busca semântica

2. Análise Multiagente:

- Agente CSV Analysis: Análise estrutural e estatística.
- Agente Embeddings: Geração e gerenciamento de vetores.
- Agente RAG: Recuperação contextual de informações
- Orchestrator: Coordenação e roteamento inteligente

3. Interface Conversacional:

- API REST com endpoints especializados
- Sistema de memória para contexto de conversação
- Respostas contextualizadas baseadas em dados reais

4. Performance e Economia:

- Sistema de roteamento de LLM (economia de 60-70% em API)
- Cache inteligente para requisições subsequentes
- Lazy loading de agentes pesados

5. Refinamento de Queries:

- Implementação de uma rotina (**QueryRefiner**) para refinar consultas que falham na busca vetorial inicial.

Framework escolhido

O projeto utiliza duas frameworks principais para a construção de sua arquitetura de backend e inteligência artificial, além de depender de uma infraestrutura de banco de dados vetorial específica, são elas: FastAPI e LangChain.

LangChain

A LangChain foi escolhida como a framework central de orquestração de Inteligência Artificial e abstração de Modelos de Linguagem de Grande Escala (LLMs). O uso da LangChain é um requisito do projeto.

A LangChain foi utilizada para as seguintes funcionalidades:

Uso da Framework	Detalhes da Implementação
Orquestração Multiagente	A LangChain é usada para estruturar e coordenar os fluxos de trabalho, especialmente na Versão 3.0, que se baseia em um design LLM-driven (conduzido por LLM), eliminando lógica condicional rígida e hard-coding.
Abstração de LLMs	A framework fornece a camada de abstração que permite ao sistema suportar múltiplos provedores (OpenAI, Google Gemini, Groq), com fallback automático em caso de falha de um provedor. O agente principal **(RAGDataAgent)** está 100% integrado ao LangChain.
Memória Persistente	O sistema utiliza componentes da LangChain para gerenciar a memória conversacional e o contexto dinâmico. Isso é feito através do **LangChainSupabaseMemory** (que persiste dados nas tabelas SQL do Supabase) para que o contexto seja mantido entre requisições.
Segurança e Execução de Código	Está planejado o uso de LangChain Tools (como o **PythonREPLTool** com sandbox) para executar código Python de forma segura, evitando vulnerabilidades críticas de segurança que existiam em versões anteriores **(exec())**.
Recuperação Aumentada (RAG)	Embora a busca vetorial utilize principalmente as funções do Supabase (pgvector), a LangChain provê os templates de mensagens, a estrutura para os agentes de retrieval (recuperação) e a base para a síntese da resposta.

FastAPI

A FastAPI é a framework escolhida para a camada de API REST do projeto, garantindo que o backend multiagente possa receber consultas e retornar resultados de forma assíncrona e eficiente. A FastAPI foi utilizada para:

Uso da Framework	Detalhes da Implementação
API REST	Implementa a arquitetura de microserviços, expondo endpoints como **/chat** (para análise conversacional), **/csv/upload** (para ingestão de dados) e **/health** (para monitoramento).
Desenvolvimento e Produção	É o motor para executar o sistema, tanto em ambiente de desenvolvimento (com auto-reload na porta 8000) quanto em produção (com workers otimizados e portas configuráveis, como a 8011).
Validação de Entrada	A framework gerencia a validação de entrada de dados, o middleware CORS e o tratamento robusto de erros na camada de API.

Em resumo, a FastAPI provê a porta de entrada (o corpo) para o sistema, enquanto a LangChain atua como sistema central (o orquestrador de IA).

Tecnologias utilizadas

1. Linguagem de Programação

• **Python (3.10+):** A linguagem base para todo o sistema.

◦ **Implementação:** É o ambiente onde a arquitetura multiagente, os agentes especializados, as ferramentas de análise (**Pandas, scikit-learn**) e o servidor (via **Uvicorn**) são executados.

2. Análise e Processamento de Dados

• **Pandas:** Biblioteca crítica para manipulação e análise de dados.

◦ **Implementação:** Utilizado por módulos internos, como o **PythonDataAnalyzer**, para realizar análise estatística descritiva, detecção de outliers e correlações. O Pandas permite que o sistema seja genérico para qualquer CSV, pois é capaz de detectar dinamicamente o header e os tipos de colunas (numéricas, categóricas) do arquivo carregado, eliminando hard-coding.

◦ Em cenários específicos (como visualização de histogramas), o Pandas é usado para reconstruir o DataFrame completo a partir dos dados brutos do CSV, pois os chunks de metadados não são suficientes para a reconstituição total.

• **NumPy:** Usado em conjunto com Pandas para cálculos numéricos e manipulação eficiente de vetores de dados.

• **scikit-learn (sklearn):** Biblioteca de aprendizado de máquina.

◦ **Implementação:** Utilizada para executar algoritmos de machine learning, como o KMeans, para realizar a análise real de clustering (agrupamento). O sistema usa o **StandardScaler** do **sklearn** para normalizar os dados antes de aplicar o KMeans, garantindo a precisão da análise.

3. Banco de Dados e Armazenamento Vetorial

• **Supabase (PostgreSQL):** O banco de dados principal que provê persistência de dados e serviços.

◦ **Implementação:** Utilizado para armazenar:

1. **Embeddings Vetoriais.**
2. **Metadados** de chunks analíticos (como estruturas de colunas, estatísticas).
3. **Memória Persistente** (histórico de conversação e contexto dinâmico) em quatro tabelas SQL específicas (agent_sessions, agent_conversations, etc).

• **pgvector:** Extensão do PostgreSQL.

◦ **Implementação:** Essencial para o funcionamento do Retrieval Augmented Generation (RAG), pois permite o armazenamento e a busca por similaridade vetorial (match_embeddings) diretamente no Supabase.

4. Modelos de Inteligência Artificial

• **Google Gemini, OpenAI e Groq:** São os Large Language Models (LLMs) utilizados como a inteligência central do sistema.

◦ **Implementação:** Os modelos são usados para tarefas cognitivas de alto nível: classificação semântica de intenções (eliminando lógica condicional rígida), síntese de respostas a partir de chunks recuperados, e refinamento de consultas (paraphrase via LLM) quando a busca inicial falha. A arquitetura inclui um gerenciador que suporta múltiplos provedores com fallback automático.

• **Sentence Transformers:** Biblioteca de processamento de linguagem natural (NLP).

◦ **Implementação:** Usada pelo **EmbeddingGenerator** para vetorizar os chunks de texto e metadados gerados durante a ingestão, transformando-os em vetores numéricos de 384 ou 768 dimensões. Esses vetores são então armazenados via **pgvector**.

5. Visualização e Testes

• **Matplotlib/Seaborn:** Bibliotecas de visualização de dados.

◦ **Implementação:** Utilizadas pelo **GraphGenerator** (módulo interno) para gerar gráficos como histogramas, scatter plots e heatmaps de correlação.

O sistema integra essas tecnologias de forma modular: o Pandas e o scikit-learn fazem a computação real dos dados, o **Supabase/pgvector** gerencia o contexto e a busca e os LLMs fornecem a interpretação e a orquestração semântica.

Abaixo segue uma tabela resumo do stack principal das tecnologias e ferramentas:

Camada	Tecnologia	Versão	Finalidade
Backend	Python	3.10+	Linguagem principal
Framework	FastAPI	0.104+	API REST
LLMs	Google Gemini	1.5/2.0	Modelos de linguagem
Orquestração	LangChain	0.2.1	Framework multiagente

Banco Dados	Supabase	Cloud	PostgreSQL + Auth
Vetorização	pgvector	0.5+	Busca semântica
Embeddings	Sentence Transformers	5.1.1	Geração de vetores
DataOps	Pandas	2.2.2	Manipulação de dados
Logging	Python logging	Built-in	Monitoramento

Dependências críticas

```
# requirements.txt
langchain==0.2.1
langchain-google-genai==1.0.1
langchain-community==0.2.1
fastapi==0.104.1
uvicorn[standard]==0.24.0
supabase==2.0.0
pandas==2.2.2
sentence-transformers==5.1.1
psycopg[binary]==3.1.12
python-dotenv==1.0.0
pydantic==2.5.0
```

Quais as principais funções do sistema

- Processar grandes volumes de dados de NF-e (arquivos CSV com mais de 150 mil linhas).
- Análise contextual inteligente via múltiplos agentes especializados.
- Busca semântica em banco vetorial com embeddings de 768 dimensões.
- Interface conversacional para consultas em linguagem natural.

- Detecção de padrões e anomalias em dados fiscais.
- Roteamento inteligente de LLM baseado em complexidade da consulta.

Como o sistema opera

I. Ingestão e Preparação de Dados

1. Carga de Dados e Conformidade: O **RAGAgent** é o único agente autorizado a realizar a ingestão. O sistema implementa um pipeline completo que lê o arquivo CSV, divide-o em chunks, gera embeddings vetoriais e os armazena na tabela embeddings do **Supabase/pgvector**.

2. Suporte a CSV Genérico: O sistema é agnóstico ao dataset (não é dependente de colunas fixas como V1-V28, Time, Amount, ou Class), funcionando com qualquer CSV sem depender de nomes de colunas (hard-coding). Utiliza chunking inteligente (como a estratégia **CSV_ROW**) e detecta dinamicamente as colunas e tipos de dados de qualquer CSV carregado.

3. Geração de Chunks Analíticos: Durante a ingestão, o sistema gera 6 chunks analíticos de metadados, que contêm análises estatísticas completas, tipologia, correlações e padrões temporais do dataset. Esses chunks permitem que o sistema RAG responda a perguntas sobre as características gerais dos dados.

II. Orquestração e Classificação Inteligente

1. Orquestração LLM-driven (V3.0): A arquitetura V3.0 (Versão 3.0) migrou para uma orquestração puramente conduzida por LLM (LLM-driven), eliminando cerca de 240 linhas de lógica condicional (if/elif) e hard-coding de keywords fixas.

2. Classificação de Intenção: O **IntentClassifier** usa LLMs para classificar semanticamente a intenção do usuário, suportando o reconhecimento de sinônimos ilimitados e múltiplas intenções simultâneas (ex: "média, desvio padrão e um histograma").

3. Roteamento: O **AnalysisOrchestrator** coordena e delega a execução para módulos especializados com base na classificação da LLM.

III. Capacidades de Análise e Visualização

1. Análise Exclusiva via Embeddings: Os agentes de análise acessam exclusivamente a tabela embeddings, nunca o arquivo CSV diretamente, exceto em cenários específicos de visualização ou fallback, com auditoria e documentação da exceção, para responder às perguntas, mantendo a conformidade arquitetural.

2. Análises Especializadas (Módulos V3.0): O sistema suporta diversas análises exploratórias:

- **Análise Estatística:** Cálculo de métricas de tendência central (média, mediana), dispersão (desvio padrão, variância) e posição (quartis, min/max).
- **Análise Temporal:** Detecção e análise de tendências, sazonalidade e anomalias temporais.
- **Análise de Clustering:** Execução REAL de algoritmos de agrupamento (como **KMeans**) nos dados, em vez de apenas interpretar chunks estáticos.
- **Análise de Frequência:** Distribuição de valores e contagem de ocorrências.

3. Geração de Gráficos: O sistema pode gerar gráficos (Histogramas, Scatter Plots, Boxplots, Heatmaps de Correlação). Para visualizações que exigem dados brutos, é implementada uma exceção controlada e documentada à política embeddings-only, lendo o CSV original de forma auditada (read-only).

IV. Inteligência, Contexto e Qualidade

1. Refinamento de Queries: O **QueryRefiner** utiliza embeddings de queries históricas bem-sucedidas e também pode gerar variações semânticas da query original via LLM (paraphrase) para aumentar o recall, especialmente se a busca vetorial inicial falhar, resultando em um ganho estimado de 20-35% de recall em consultas não-padrão.

2. Memória Persistente: O sistema utiliza **LangChainSupabaseMemory** (integrando LangChain com as tabelas SQL do Supabase) para manter o histórico conversacional e o contexto dinâmico entre as interações.

3. Segurança e Qualidade: O projeto estabelece altos padrões de qualidade (V3.0), visando a implementação de sandbox seguro (via LangChain **PythonREPLTool**) para execução de código Python. Além disso, utiliza logging estruturado para rastreabilidade e auditoria.

Estruturação do projeto

A estrutura é dividida em camadas lógicas (Agentes, Roteamento, Armazenamento) e uma organização física de diretórios padronizada.

1. Estrutura Física de Diretórios

O código fonte é segregado em pastas com responsabilidades únicas:

Diretório	Responsabilidade Principal	Detalhes da Implementação
src/	Código de produção do sistema multiagente.	Contém subpastas como **agent/, data/, memory/, e vectorstore/**.
src/agent/	Implementação dos agentes especializados (**RAGDataAgent, OrchestratorAgent**, etc.)	
src/memory/	Sistema de memória persistente	
src/embeddings/ ou src/vectorstore/	Integração com Supabase e pgvector	
tests/	Testes unitários e de integração.	Estrutura espelha src/.
docs/	Documentação técnica, arquitetura e histórico.	Inclui **architecture/, guides/, changelog/, archive/**.

configs/	Variáveis de ambiente e arquivos de configuração (**.env** e credenciais)	
data/	Arquivos CSV de teste e demonstração.	Datasets grandes devem ser ignorados pelo **.gitignore**
examples/	Scripts de demonstração e tutoriais.	Contém scripts executáveis que demonstram funcionalidades
Raiz	Arquivos essenciais.	Contém **README.md, CHANGELOG.md, api_completa.py** (API principal na porta 8001) e **api_simple.py** (API de testes na porta 8000).

2. Arquitetura Lógica e Camadas (V3.0)

A arquitetura do projeto é baseada em microserviços e uma cadeia de agentes (Multiagente) orquestrada. A Arquitetura V3.0 é centrada em inteligência assistida por LLM e zero hard-coding.

A. Camada de API e Entrypoints

A camada de API usa FastAPI. O entrypoint principal é **api_completa.py**, responsável por:

1. Receber requisições REST, como /chat e /csv/upload.

2. Gerenciar o lazy loading de agentes pesados (60-90s na primeira requisição, 2-10s nas subsequentes).

3. Aplicar roteamento de LLM baseado na complexidade da consulta.

B. Camada de Agentes (O Core Multiagente)

Os agentes herdam do **BaseAgent**, que fornece a infraestrutura para memória persistente.

Agente/Componente	Função Principal	Status/Localização
OrchestratorAgent	Coordenador central. Recebe a query do usuário, analisa a complexidade e roteia para os agentes especializados. Gerencia o contexto da conversação.	**src/agent/orchestrator_agent.py**
IntentClassifier	Classificação semântica da intenção do usuário via LLM (Versão V3.0). Substitui a lógica condicional (if/elif) e keywords fixas (presentes na V2.0).	**src/analysis/intent_classifier.py**
RAGDataAgent	Agente principal que lida com a busca vetorial (RAG), análise de dados, e síntese da resposta. Também é o único agente autorizado para a ingestão de CSV (Requisito 1).	**src/agent/rag_data_agent.py**
Analisadores Especializados	Módulos que realizam computação real (KMeans, estatísticas, etc.) nos dados. Incluem **StatisticalAnalyzer, TemporalAnalyzer, ClusteringAnalyzer**, etc..	**src/analysis/**
RAGSynthesisAgent	Módulo especializado em consolidar respostas fragmentadas do sistema RAG e formatar o resultado em linguagem natural humanizada via LLM.	**src/agent/rag_synthesis_agent.py**
LLM Manager	Camada de abstração para LLMs, suportando múltiplos provedores (OpenAI, Google Gemini, Groq) com fallback automático.	**src/llm/manager.py**

C. Camada de Dados e Armazenamento

Esta camada fornece a infraestrutura para a persistência e a inteligência de busca:

1. Armazenamento Vetorial (VectorStore): Utiliza Supabase (PostgreSQL) com a extensão pgvector. Armazena os embeddings e metadados analíticos.

2. Memória Persistente: O sistema utiliza **LangChainSupabaseMemory** para persistir o contexto conversacional. A infraestrutura de memória é composta por quatro tabelas SQL no Supabase: **agent_sessions, agent_conversations, agent_context, e agent_memory_embeddings**.

3. Geração de Embeddings: O sistema usa tecnologias como **Sentence Transformers** para a vetorização de chunks de texto e metadados, com dimensões configuráveis.

4. Chunks Analíticos: Durante a ingestão de qualquer CSV (Requisito 3 - CSV Genérico), são gerados **6 chunks analíticos** (sobre tipologia, distribuições, variabilidade, correlações, padrões temporais, e agrupamentos) que permitem ao RAG responder perguntas sobre as características gerais do dataset.

3. Fluxo de Execução Simplificado (V3.0)

O fluxo de execução principal na V3.0 é:

1. O **OrchestratorAgent** recebe a query do usuário.

2. O **IntentClassifier** usa o LLM para classificar a intenção e determinar quais analisadores são necessários (suportando múltiplas intenções).

3. O **Orchestrator** delega a execução aos módulos especializados (e.g., **StatisticalAnalyzer, ClusteringAnalyzer**), que acessam o contexto e os dados via Embeddings.

4. Se a busca inicial via RAG falhar, o **QueryRefiner** pode ser acionado para gerar variações semânticas da consulta original via LLM (paraphrase), aumentando o recall.

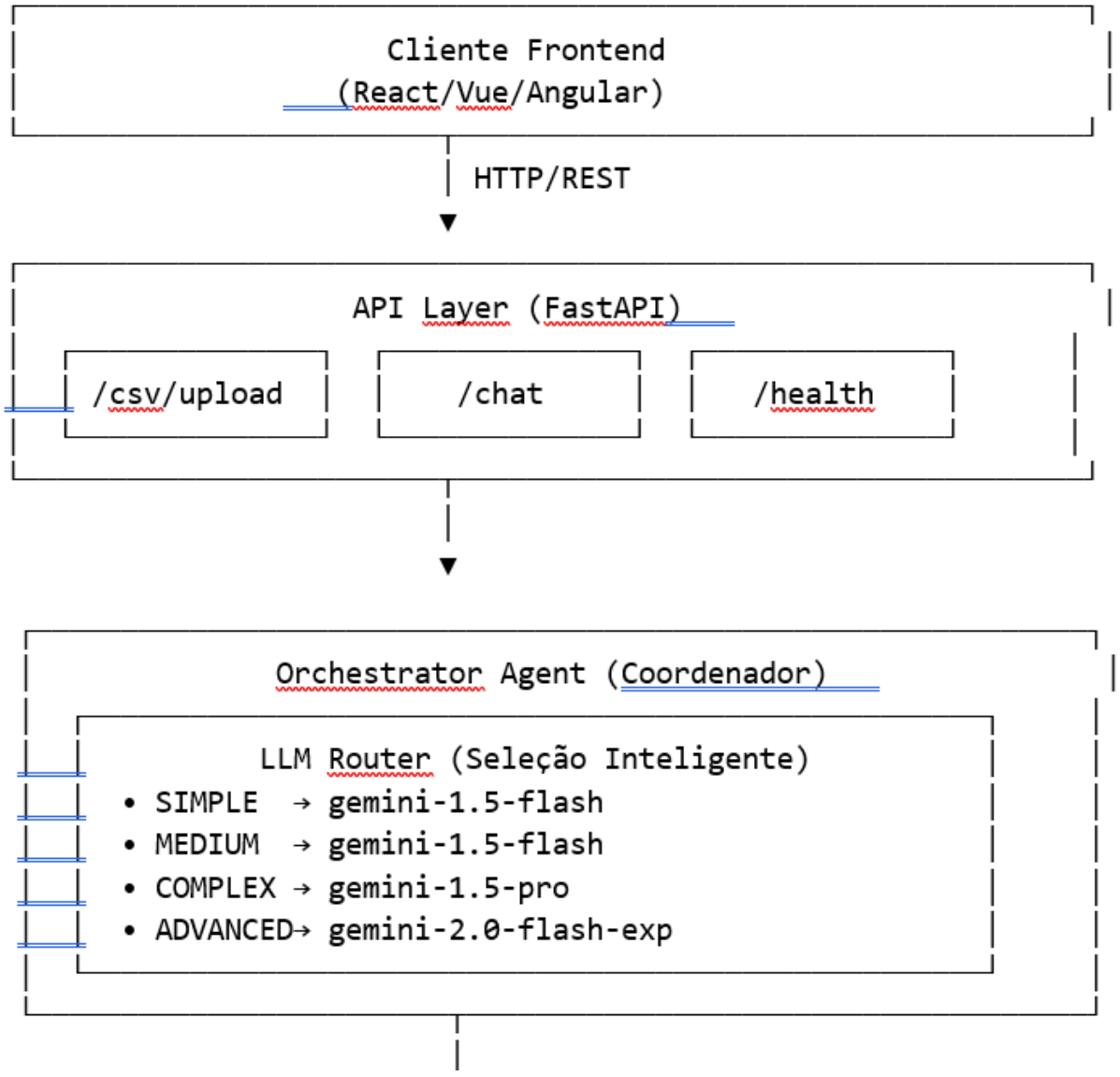
5. Os resultados técnicos são reunidos e o **RAGSynthesisAgent** usa o LLM para formatar a resposta final humanizada.

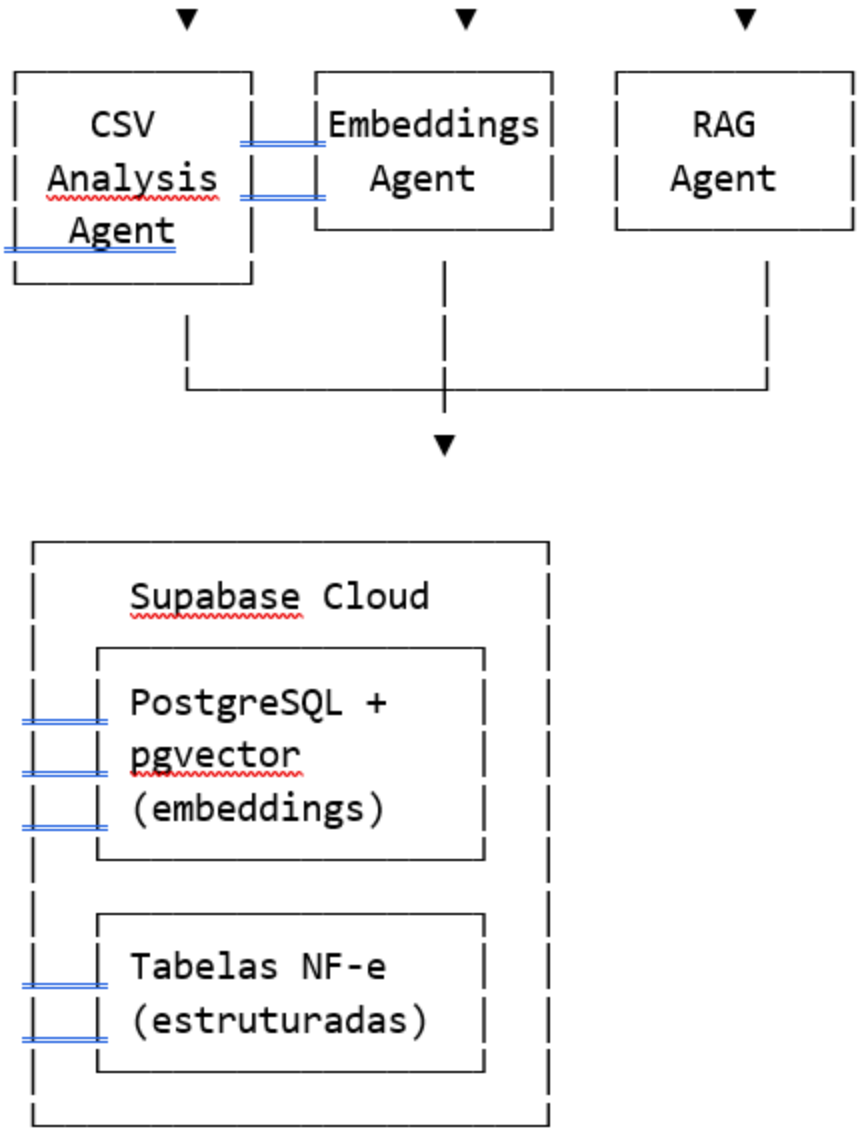
Em resumo, a estrutura combina:

- Organização de diretórios modular (src/, tests/, docs/), uma arquitetura LLM-driven de agentes especializados.
- Infraestrutura robusta de banco de dados vetorial (**Supabase/pgvector**) e memória persistente (**LangChainSupabaseMemory**).

Arquitetura do Sistema

Visão Geral



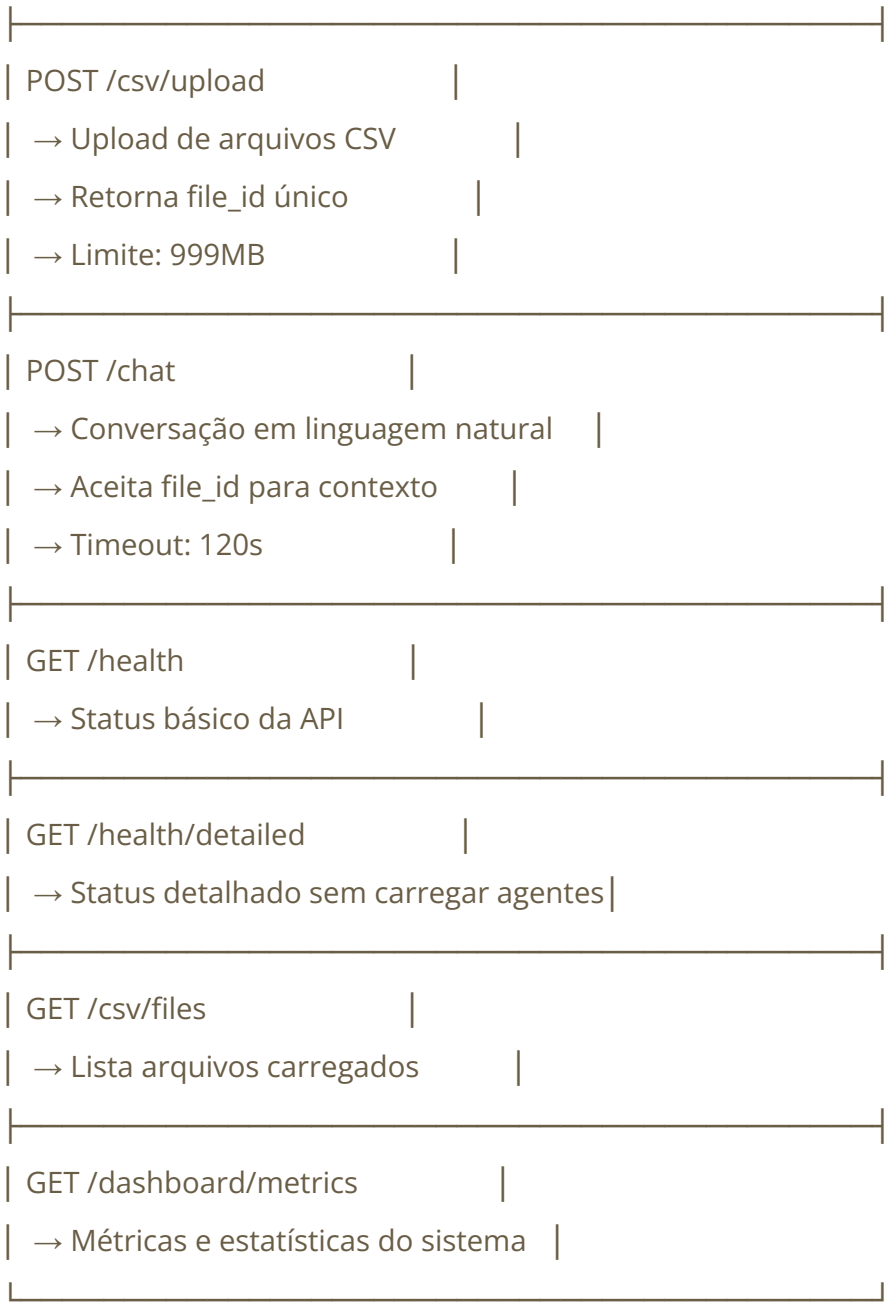


Camadas da Arquitetura

1. API Layer (FastAPI)

Python: api_completa.py - Porta 8001

| Endpoints Principais |



Características camada API Layer:

- Middleware CORS configurado
- Tratamento robusto de erros
- Logging estruturado
- Validação de entrada

- Cache em memória para **file_ids**

2. Camada de Agentes

Orchestrator Agent (`**src/agent/orchestrator_agent.py**`)

Coordenador central responsável por:
- Receber consultas do usuário
- Analisar complexidade da query
- Rotear para agentes especializados
- Integrar respostas
- Gerenciar contexto da conversação

CSV Analysis Agent (`**src/agent/csv_analysis_agent.py**`)
Especializado em análise estrutural de dados:
- Análise estatística descritiva
- Detecção de padrões
- Identificação de anomalias
- Análise de correlações
- Geração de insights

Funcionalidades:
```python
- analyze_structure()  # Tipos, valores nulos, distribuições
- detect_fraud_patterns()  # Identificação de fraudes
- generate_statistics()  # Estatísticas descritivas
- analyze_correlations()  # Análise de correlações
```

Embeddings Agent (`**src/embeddings/generator.py**`)

Gerenciamento de vetores:

- Geração de embeddings (768 dimensões)
- Chunking estratégico
- Armazenamento no Supabase
- Indexação para busca rápida

Processo de Chunking:

```python
1. Carregar CSV
2. Converter linhas em texto estruturado
3. Dividir em chunks de 1024 caracteres
4. Overlap de 100 caracteres
5. Gerar embeddings via Sentence Transformers
6. Expandir de 384 → 768 dimensões
7. Armazenar no banco vetorial
```

RAG Agent (`**src/rag/retriever.py**`)

Recuperação de contexto:

- Busca semântica no banco vetorial
- Ranking por similaridade
- Filtros por metadados
- Construção de contexto relevante

Fluxo de Busca

```python
Query → Embedding → Busca Vetorial → Top-K → Ranking → Contexto
```

3. Sistema de LLM Router

Componente crítico para economia e performance:

```python
# src/llm/llm_router.py

class QueryComplexity(Enum):
    SIMPLE = "simple"      # Consultas básicas
    MEDIUM = "medium"      # Análises simples
    COMPLEX = "complex"    # Análises complexas
    ADVANCED = "advanced"  # Análises avançadas

class LLMRouter:
    COMPLEXITY_MODELS = {
        QueryComplexity.SIMPLE: "gemini-1.5-flash",
        QueryComplexity.MEDIUM: "gemini-1.5-flash",
        QueryComplexity.COMPLEX: "gemini-1.5-pro",
        QueryComplexity.ADVANCED: "gemini-2.0-flash-exp"
```

Economia Estimada:

- Flash vs Pro: ~70% mais barato.
- Roteamento inteligente: 60-70% economia total.
- Fallback automático em caso de falha.

4. Banco de Dados (Supabase)

Schema Vetorial:

```sql
-- Tabela de Embeddings
CREATE TABLE embeddings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    chunk_text TEXT NOT NULL,
    embedding VECTOR(768),  -- pgvector
    metadata JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Índice HNSW para busca rápida
CREATE INDEX embeddings_vector_idx
ON embeddings
USING hnsw (embedding vector_cosine_ops);
```

Tabelas Estruturadas:

```sql
-- Nota Fiscal (Header)
CREATE TABLE nota_fiscal (
    id UUID PRIMARY KEY,
    chave_acesso VARCHAR(44) UNIQUE,
    numero_nota VARCHAR(20),
    serie INTEGER,
    data_emissao TIMESTAMPTZ,
    cnpj_emitente VARCHAR(14),
    valor_total NUMERIC(15,2),
    metadata JSONB,
```

```
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Itens da Nota Fiscal:
CREATE TABLE nota_fiscal_item (
    id UUID PRIMARY KEY,
    nota_fiscal_id UUID REFERENCES nota_fiscal(id),
    numero_item INTEGER,
    codigo_produto VARCHAR(50),
    descricao TEXT,
    quantidade_comercial NUMERIC(15,4),
    valor_total NUMERIC(15,2),
    metadata JSONB,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Melhorias implementadas em relação a versões anteriores:

Melhorias efetuadas na Refatoração e Limpeza: Isolamento da Funcionalidade de Fraude (Cartão de Crédito).

Nesta etapa, o objetivo principal foi isolar o Módulo de Detecção de Fraude a fim de remover código não relacionado ao core fiscal para simplificar a manutenção e o desenvolvimento do MVP.

Código, APIs, dados de exemplo e dados no DB relacionados à fraude são completamente removidos do projeto principal. Funcionalidades restantes (upload, chat básico) continuam operando. Segue abaixo os procedimentos realizados:

Tarefa Técnica - Análise: Mapear código/dados de fraude. Critério: Listar completamente os artefatos a serem removidos.

Tarefa Técnica - Execução: Remover código/arquivos no backend e frontend. Estratégia: Remoção completa.

Resultado: Código removido; Projeto compila/inicia sem erros relacionados à fraude.

Tarefa Técnica - Remover endpoints da API FastAPI de fraude.

Resultado: Rotas removidas; Swagger atualizado.

Tarefa Técnica - Remover arquivos de dados/scripts de exemplo/teste.

Resultado: Arquivos deletados do repositório.

Tarefa Técnica - Banco Vetorial: Limpar dados de fraude do Supabase/pgvector.

Resultado: Script SQL (DELETE FROM embeddings WHERE metadata->>'type' = 'creditcard'; ou similar) executado; Verificação confirma remoção.

Tarefa Técnica: Testar funcionalidade básica (upload genérico, chat) após remoção.

Resultado: Testes manuais confirmam que as operações básicas não foram quebradas.

Link para execução do sistema multiagente:

http://srv774816.hstgr.cloud:9001

Links para o repositório do GitHub

BackEnd:https://github.com/ai-mindsgroup/agentnfe-backend.git

FrontEnd: https://github.com/ai-mindsgroup/agentnfe-frontend.git