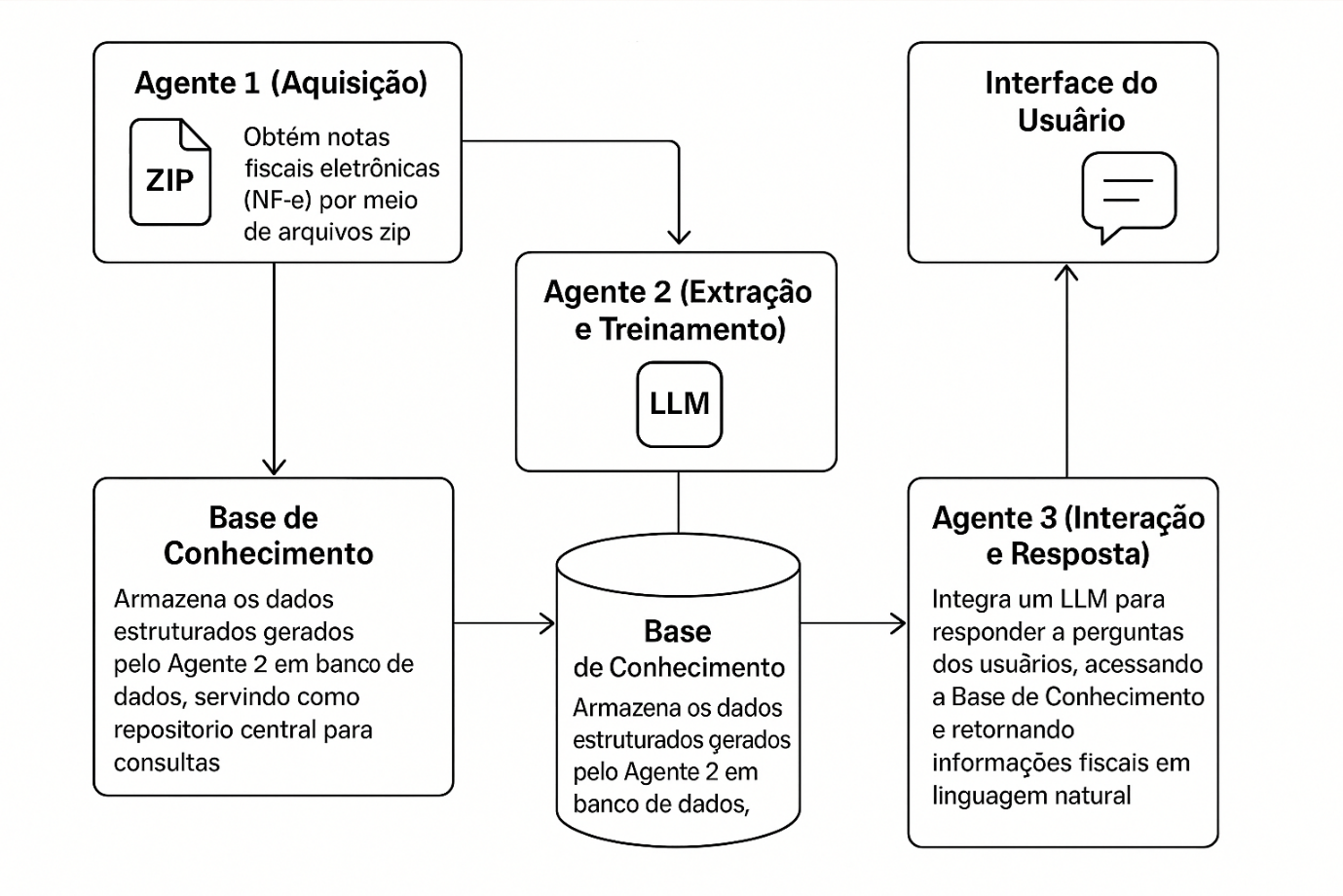
**Relatório do Agente**

Entrega II

1. **Qual o framework?**

A solução utiliza o LangChain como a principal framework para desenvolver o agente de inteligência artificial. O LangChain é uma biblioteca Python projetada para facilitar a construção de aplicações baseadas em modelos de linguagem (LLMs), permitindo a integração com dados externos, ferramentas, e memória contextual. No caso do seu código, o LangChain é combinado com outras bibliotecas complementares para criar um agente que processa perguntas em linguagem natural e retorna resultados de consultas SQL em notas fiscais.

1. **Como a solução foi estruturada:**



* **Agente 1 (Aquisição):** Obtém notas fiscais eletrônicas (NF-e) por meio de arquivos zip
* **Agente 2 (Extração e Treinamento):** Processa os documentos adquiridos, aplicando OCR para extrair dados e utilizando um Large Language Model (LLM) para aprender novos layouts. Armazena os dados extraídos em banco de dados, otimizando a precisão na captura de informações fiscais.
* **Base de Conhecimento:** Armazena os dados estruturados gerados pelo Agente 2 em banco de dados, servindo como repositório central para consultas.
* **Agente 3 (Interação e Resposta):** Integra um LLM para responder a perguntas dos usuários, acessando a Base de Conhecimento e retornando informações fiscais em linguagem natural.
* **Interface do Usuário:** Permite interação com o Agente 3, recebendo respostas baseadas nos dados processados.

Essa arquitetura automatiza o ciclo completo de aquisição, extração e consulta de documentos fiscais, garantindo eficiência e precisão.

**Detalhes técnicos da framework:**

* **LangChain:**
* **Versão:** Usa langchain-google-genai, que integra o modelo Gemini da Google (ChatGoogleGenerativeAI).
* **Componentes Utilizados:**
* **PromptTemplate:** Define modelos de prompts para estruturar as entradas do LLM, com variáveis como {pergunta} e {df}.
* **JsonOutputParser com Pydantic:** Valida as saídas do LLM, garantindo que sigam formatos estruturados (ex.: Resposta com campo resposta: str ou Query com campo query: str).
* **Cadeias:** Combina PromptTemplate, LLM, e parser em pipelines para processar entradas e saídas de forma encadeada.

**Modelo de linguagem**

* ChatGoogleGenerativeAIcom o modelo gemini-2.0-flash, configurado com temperature=0.5 para respostas balanceadas entre precisão e criatividade.

**Outras bibliotecas complementares**

* **Pandas:** Manipula dataframes para carregar arquivos .csv, validar dados, e retornar resultados de queries SQL**.**
* **SQLAlchemy:** Gerencia conexões com o banco de dados SQLite (nfs\_data.db) e executa queries SQL.
* **Pydantic:** Define modelos de dados (Query, Resposta) para validação estruturada das saídas do LLM.

* **Zipfile e re:** Bibliotecas padrão do Python para descompactar arquivos .zip e usar expressões regulares na validação de nomes de arquivos.
* **Integração com Gemini:** A API do Gemini é acessada via ChatGoogleGenerativeAI

**Ambiente de execução**

* Há suporte para integração com um frontend, que fornece pergunta e diretorio, possibilitando a utilização em aplicações web ou desktop.

**Funcionalidades do Frontend**

* + **Upload de Arquivo ZIP:** O usuário faz o upload de um arquivo .zip contendo os **arquivos CSV de cabeçalho e itens das NF-es**.
  + **Campo de Pergunta (Linguagem Natural):** Permite que o usuário digite perguntas abertas sobre os dados contidos no ZIP, como por exemplo:  
    "Qual é a chave de acesso da nota 3510129?"
  + **Botão de Consulta:** Ao clicar em **"🔍 Consultar"**, o sistema:
  1. **Valida o upload**:  
     Se o arquivo ZIP não for enviado ou se a pergunta estiver em branco, o sistema exibe mensagens de erro claras.
  2. **Executa a análise com IA:**  
     Um **spinner de carregamento** aparece durante o processamento.
  3. **Apresenta o Resultado:**
     + Caso haja problemas nos arquivos (ex: falta de CSV de cabeçalho ou itens), o usuário é informado com mensagens de erro específicas.
     + Caso a IA conclua que os dados não são suficientes para responder, o sistema exibe um aviso do tipo **"Nenhum dado encontrado"**.
     + Se a consulta for bem-sucedida, o resultado da query é exibido em forma de **tabela interativa dentro do Streamlit**.
* **Feedback em Tempo Real:**  
  Todas as mensagens de erro, sucesso e status são mostradas de forma clara, usando os componentes nativos do Streamlit (st.error(), st.warning(), st.success(), st.dataframe(), etc).

1. **Link para o Github**

<https://github.com/ajndantas/I2A2-Grupo_01>

Obs: As instruções de instalação se encontram no README do GitHub

1. **Perguntas e respostas:**