aiBA

Documentation - Artificial Intelligence Bidding Assistant.

RESUME:

“aiBA” is an artificial intelligence assistant developed to optimize analysis and query processes in PDF documents. It combines natural language processing with machine learning, using the OpenAI GPT-4 model combined with the simple RAG technique, to interpret user input and generate responses.

Integrated with a vector storage system, aiBA performs semantic searches in documents, creating personalized reports in .docx format to meet analysis demands.

1. Requisitos para Configuração

Antes de instalar e testar o código, certifique-se de ter o ambiente correto configurado:

Pré-requisitos

- **Python 3.9 ou superior**
- Dependências listadas no arquivo requirements.txt, incluindo:
  - Flask
  - LangChain
  - FAISS
  - OpenAI Python SDK
  - PyMuPDF
  - dotenv

Configuração Inicial

1. Clone o repositório:

   bash
   git clone <link-do-repositorio>
   cd <pasta-do-repositorio>

2. Crie e ative um ambiente virtual:

   bash
   python3 -m venv venv

   source venv/bin/activate # Linux/macOS
   venv\Scripts\activate   # Windows

3. Instale as dependências:

   bash
   pip install -r requirements.txt

4. Configure as variáveis de ambiente no arquivo. env:

   UPLOAD_FOLDER=uploads
   OUTPUT_FOLDER=outputs
   FRONT_OP=frontend/index.html
   OPENAI_API_KEY=<sua-chave-openai>

5. Crie os diretórios necessários:

   bash
   mkdir uploads outputs

---

2. Como Executar

1. Execute a aplicação Flask:

   ```
   bash
   python app.py
   ```

2. Acesse a aplicação em seu navegador:
   - URL padrão: http://127.0.0.1:5000/

---

3. Fluxo de Dados

1. **Envio de Arquivos**:
   - O endpoint /upload aceita arquivos PDF, salvando-os no diretório especificado por UPLOAD_FOLDER.
2. **Processamento do Documento**:
   - O arquivo PDF é carregado e analisado usando o **PyMuPDFLoader**, que extrai o conteúdo do documento.
3. **Armazenamento Vetorial**:
   - Os dados extraídos são indexados em uma base vetorial **FAISS**, permitindo buscas semânticas.
4. **Consulta e Resposta**:
   - Uma consulta é feita ao modelo para buscar informações relevantes no documento indexado.
   - O modelo GPT-4, com um prompt customizado, gera uma resposta baseada na consulta.
5. **Geração de Relatórios**:
   - O relatório é formatado em um arquivo .docx usando a função criar_docx e salvo no diretório OUTPUT_FOLDER.
6. **Download do Relatório**:
   - O endpoint /download/<filename> permite o download do relatório gerado.

---

4. Descrição das Funções

Funções do Código

1. **carregar_documentos(file_path)**
   - Carrega e processa documentos PDF do caminho especificado.
   - Retorna o conteúdo dos documentos como uma lista de objetos.
2. **gerar_resposta(message, dados)**
   - Usa a cadeia LLM do LangChain para gerar respostas baseadas na entrada do usuário e nos dados carregados.
3. **criar_docx(conteudo, nome_arquivo_original)**
   - Formata e salva o conteúdo gerado em um arquivo .docx.
   - Inclui cabeçalho, rodapé e personalização de layout.

Rotas da API

1. **GET /**
   - Renderiza a interface HTML da aplicação, caso definida.
   - Exibe uma mensagem de erro se o arquivo HTML não for encontrado.
2. **POST /upload**
   - Aceita um arquivo PDF, processa-o e gera um relatório baseado nos dados.
   - Retorna a URL para download do relatório gerado.
3. **GET /download/<filename>**
   - Permite o download do relatório .docx pelo nome.

---

5. Lógica Completa

1. O usuário faz upload de um arquivo PDF para /upload.
2. O documento é analisado e seu conteúdo indexado em FAISS.
3. O sistema gera uma consulta semântica para buscar informações relevantes.
4. O modelo GPT-4 usa um prompt customizado para gerar a resposta final.
5. O relatório é criado em .docx e disponibilizado para download.
