"""

PDF document loader implementation.

This module provides functionality to load and process PDF documents

for the RAG pipeline.

"""

from typing import List, Any, Optional

from pathlib import Path

import logging

from langchain\_community.document\_loaders import PyPDFLoader

from langchain.schema import Document

from .base import BaseDocumentLoader

from ..utils.exceptions import PDFProcessingError, DataLoadingError

from ..utils.logging\_config import get\_logger, LogContext

logger = get\_logger(\_\_name\_\_)

class PDFDocumentLoader(BaseDocumentLoader):

"""

PDF document loader using LangChain's PyPDFLoader.

This loader handles PDF files and converts them into Document objects

suitable for further processing in the RAG pipeline.

"""

def \_\_init\_\_(self, extract\_images: bool = False):

"""

Initialize the PDF loader.

Args:

extract\_images: Whether to extract images from PDFs

"""

self.extract\_images = extract\_images

logger.info("Initialized PDFDocumentLoader",

extra={"extract\_images": extract\_images})

def load(self, file\_path: Path) -> List[Document]:

"""

Load a single PDF document.

Args:

file\_path: Path to the PDF file

Returns:

List of Document objects

Raises:

PDFProcessingError: If the PDF cannot be processed

DataLoadingError: If the file cannot be loaded

"""

with LogContext(logger, file\_path=str(file\_path)):

logger.info(f"Loading PDF document: {file\_path}")

# Validate file path

if not isinstance(file\_path, Path):

file\_path = Path(file\_path)

if not file\_path.exists():

raise DataLoadingError(f"File not found: {file\_path}")

if not file\_path.suffix.lower() == '.pdf':

raise DataLoadingError(f"Not a PDF file: {file\_path}")

try:

# Load the PDF using PyPDFLoader

loader = PyPDFLoader(str(file\_path))

documents = loader.load()

# Add metadata

for doc in documents:

doc.metadata['source\_type'] = 'pdf'

doc.metadata['file\_name'] = file\_path.name

doc.metadata['file\_path'] = str(file\_path)

logger.info(f"Successfully loaded {len(documents)} pages from {file\_path.name}")

return documents

except Exception as e:

logger.error(f"Error loading PDF: {str(e)}", exc\_info=True)

raise PDFProcessingError(f"Failed to process PDF {file\_path}: {str(e)}")

def load\_multiple(self, file\_paths: List[Path]) -> List[Document]:

"""

Load multiple PDF documents.

Args:

file\_paths: List of paths to PDF files

Returns:

List of all Document objects from all PDFs

Raises:

DataLoadingError: If no documents could be loaded

"""

logger.info(f"Loading {len(file\_paths)} PDF documents")

all\_documents = []

failed\_files = []

for file\_path in file\_paths:

try:

documents = self.load(file\_path)

all\_documents.extend(documents)

except (PDFProcessingError, DataLoadingError) as e:

logger.warning(f"Failed to load {file\_path}: {str(e)}")

failed\_files.append(str(file\_path))

if not all\_documents and failed\_files:

raise DataLoadingError(

f"Failed to load any documents. Failed files: {', '.join(failed\_files)}"

)

logger.info(

f"Loaded {len(all\_documents)} pages from {len(file\_paths) - len(failed\_files)} files. "

f"Failed: {len(failed\_files)}"

)

return all\_documents

def validate\_document(self, document: Document) -> bool:

"""

Validate a loaded document.

Args:

document: Document to validate

Returns:

True if valid, False otherwise

"""

if not document.page\_content or not document.page\_content.strip():

logger.warning("Document has empty content")

return False

if len(document.page\_content) < 10:

logger.warning(f"Document content too short: {len(document.page\_content)} chars")

return False

return True