"""

Unit tests for document chunker.

This module contains tests for the DocumentChunker class,

ensuring proper text splitting and chunk management.

"""

import pytest

from unittest.mock import Mock, patch

from src.text\_processing.chunker import DocumentChunker

from src.utils.exceptions import ChunkingError

from langchain.schema import Document

class TestDocumentChunker:

"""Test suite for DocumentChunker."""

@pytest.fixture

def chunker(self):

"""Create a DocumentChunker instance."""

return DocumentChunker(chunk\_size=100, chunk\_overlap=20)

@pytest.fixture

def sample\_documents(self):

"""Create sample documents for testing."""

return [

Document(

page\_content="This is a long document that needs to be chunked. " \* 10,

metadata={"source": "doc1.pdf", "page": 1}

),

Document(

page\_content="Another document with different content. " \* 8,

metadata={"source": "doc2.pdf", "page": 1}

)

]

def test\_initialization(self):

"""Test chunker initialization with various parameters."""

# Valid initialization

chunker = DocumentChunker(chunk\_size=500, chunk\_overlap=50)

assert chunker.chunk\_size == 500

assert chunker.chunk\_overlap == 50

# Invalid overlap

with pytest.raises(ValueError, match="chunk\_overlap must be less than chunk\_size"):

DocumentChunker(chunk\_size=100, chunk\_overlap=100)

def test\_process\_documents(self, chunker, sample\_documents):

"""Test processing documents into chunks."""

chunks = chunker.process(sample\_documents)

# Verify chunks were created

assert len(chunks) > len(sample\_documents)

# Verify chunk metadata

for i, chunk in enumerate(chunks):

assert 'chunk\_index' in chunk.metadata

assert chunk.metadata['chunk\_index'] == i

assert 'chunk\_size' in chunk.metadata

assert chunk.metadata['chunk\_size'] == len(chunk.page\_content)

assert chunk.metadata['chunking\_method'] == 'recursive\_character'

def test\_process\_empty\_documents(self, chunker):

"""Test processing empty document list."""

chunks = chunker.process([])

assert chunks == []

def test\_chunk\_size\_constraint(self, chunker, sample\_documents):

"""Test that chunks respect size constraints."""

chunks = chunker.process(sample\_documents)

for chunk in chunks:

# Allow for some flexibility due to word boundaries

assert len(chunk.page\_content) <= chunker.chunk\_size + 20

def test\_chunk\_overlap(self):

"""Test that chunks have proper overlap."""

# Create chunker with specific parameters

chunker = DocumentChunker(chunk\_size=50, chunk\_overlap=10)

# Create document with predictable content

content = "ABCDEFGHIJ" \* 10 # 100 characters

doc = Document(page\_content=content, metadata={})

chunks = chunker.process([doc])

# Verify overlap between consecutive chunks

for i in range(len(chunks) - 1):

chunk1\_end = chunks[i].page\_content[-10:]

chunk2\_start = chunks[i + 1].page\_content[:10]

# Check if there's overlap (may not be exact due to splitting logic)

assert len(chunks) > 1 # Should create multiple chunks

def test\_chunk\_text\_method(self, chunker):

"""Test chunking a single text string."""

text = "This is a test text. " \* 20

metadata = {"source": "test"}

chunks = chunker.chunk\_text(text, metadata)

assert len(chunks) > 0

assert all(chunk.metadata["source"] == "test" for chunk in chunks)

def test\_update\_parameters(self, chunker):

"""Test updating chunker parameters."""

# Update chunk size

chunker.update\_parameters(chunk\_size=200)

assert chunker.chunk\_size == 200

# Update chunk overlap

chunker.update\_parameters(chunk\_overlap=40)

assert chunker.chunk\_overlap == 40

# Update both

chunker.update\_parameters(chunk\_size=300, chunk\_overlap=60)

assert chunker.chunk\_size == 300

assert chunker.chunk\_overlap == 60

def test\_process\_with\_exception(self, chunker):

"""Test handling of exceptions during processing."""

with patch.object(chunker.text\_splitter, 'split\_documents', side\_effect=Exception("Test error")):

with pytest.raises(ChunkingError, match="Failed to chunk documents"):

chunker.process([Document(page\_content="Test", metadata={})])

def test\_custom\_separators(self):

"""Test chunker with custom separators."""

custom\_separators = ["\n\n", "\n", ". ", " "]

chunker = DocumentChunker(

chunk\_size=100,

chunk\_overlap=20,

separators=custom\_separators

)

assert chunker.separators == custom\_separators

def test\_preserve\_metadata(self, chunker, sample\_documents):

"""Test that original metadata is preserved in chunks."""

chunks = chunker.process(sample\_documents)

# Check that original metadata is preserved

for chunk in chunks:

assert 'source' in chunk.metadata

assert chunk.metadata['source'] in ["doc1.pdf", "doc2.pdf"]

@patch('src.text\_processing.chunker.logger')

def test\_logging\_statistics(self, mock\_logger, chunker, sample\_documents):

"""Test that chunking statistics are logged."""

chunks = chunker.process(sample\_documents)

# Verify logging was called

assert mock\_logger.info.called

# Check for statistics logging

stats\_logged = False

for call in mock\_logger.info.call\_args\_list:

if "Chunking statistics" in str(call):

stats\_logged = True

break

assert stats\_logged

def test\_chunk\_metadata\_structure(self, chunker):

"""Test the structure of chunk metadata."""

doc = Document(page\_content="Test content " \* 20, metadata={"original": "metadata"})

chunks = chunker.process([doc])

for chunk in chunks:

# Original metadata preserved

assert chunk.metadata.get("original") == "metadata"

# Chunk-specific metadata added

assert "chunk\_params" in chunk.metadata

assert chunk.metadata["chunk\_params"]["size"] == chunker.chunk\_size

assert chunk.metadata["chunk\_params"]["overlap"] == chunker.chunk\_overlap