"""

Unit tests for vector store implementations.

This module tests the ChromaVectorStore functionality including

document storage, retrieval, and persistence.

"""

import pytest

from unittest.mock import Mock, patch, MagicMock

from pathlib import Path

import tempfile

import shutil

from src.vector\_store.chroma\_store import ChromaVectorStore

from src.utils.exceptions import VectorStoreError

from langchain.schema import Document

class TestChromaVectorStore:

"""Test suite for ChromaVectorStore."""

@pytest.fixture

def mock\_embedding\_model(self):

"""Create a mock embedding model."""

mock\_model = Mock()

mock\_model.embed\_documents.return\_value = [[0.1, 0.2, 0.3], [0.4, 0.5, 0.6]]

mock\_model.embed\_query.return\_value = [0.1, 0.2, 0.3]

return mock\_model

@pytest.fixture

def temp\_persist\_dir(self):

"""Create a temporary directory for persistence."""

temp\_dir = tempfile.mkdtemp()

yield temp\_dir

# Cleanup

shutil.rmtree(temp\_dir, ignore\_errors=True)

@pytest.fixture

def vector\_store(self, mock\_embedding\_model, temp\_persist\_dir):

"""Create a vector store instance."""

return ChromaVectorStore(

embedding\_model=mock\_embedding\_model,

persist\_directory=str(temp\_persist\_dir),

collection\_name="test\_collection"

)

def test\_initialization(self, mock\_embedding\_model):

"""Test vector store initialization."""

store = ChromaVectorStore(

embedding\_model=mock\_embedding\_model,

collection\_name="test",

collection\_metadata={"description": "Test collection"}

)

assert store.embedding\_model == mock\_embedding\_model

assert store.collection\_name == "test"

assert store.collection\_metadata["description"] == "Test collection"

assert store.\_vectorstore is None # Not initialized until documents added

@patch('src.vector\_store.chroma\_store.Chroma')

def test\_add\_documents\_first\_time(self, mock\_chroma\_class, vector\_store, mock\_embedding\_model):

"""Test adding documents for the first time."""

# Create mock Chroma instance

mock\_chroma = Mock()

mock\_chroma\_class.from\_documents.return\_value = mock\_chroma

# Test documents

documents = [

Document(page\_content="Test content 1", metadata={"source": "test1.pdf"}),

Document(page\_content="Test content 2", metadata={"source": "test2.pdf"})

]

# Add documents

vector\_store.add\_documents(documents)

# Verify Chroma.from\_documents was called

mock\_chroma\_class.from\_documents.assert\_called\_once\_with(

documents=documents,

embedding=mock\_embedding\_model,

persist\_directory=vector\_store.persist\_directory,

collection\_name="test\_collection",

collection\_metadata={},

ids=None

)

# Verify vector store is now initialized

assert vector\_store.\_vectorstore == mock\_chroma

@patch('src.vector\_store.chroma\_store.Chroma')

def test\_add\_documents\_to\_existing(self, mock\_chroma\_class, vector\_store):

"""Test adding documents to existing vector store."""

# Setup existing vector store

mock\_chroma = Mock()

vector\_store.\_vectorstore = mock\_chroma

# Test documents

documents = [

Document(page\_content="New content", metadata={"source": "new.pdf"})

]

# Add documents

vector\_store.add\_documents(documents)

# Verify add\_documents was called on existing store

mock\_chroma.add\_documents.assert\_called\_once\_with(documents, ids=None)

# Verify from\_documents was NOT called

mock\_chroma\_class.from\_documents.assert\_not\_called()

def test\_add\_documents\_empty\_list(self, vector\_store):

"""Test adding empty document list."""

vector\_store.add\_documents([])

# Should not create vector store for empty list

assert vector\_store.\_vectorstore is None

@patch('src.vector\_store.chroma\_store.Chroma')

def test\_add\_documents\_with\_ids(self, mock\_chroma\_class, vector\_store, mock\_embedding\_model):

"""Test adding documents with custom IDs."""

mock\_chroma = Mock()

mock\_chroma\_class.from\_documents.return\_value = mock\_chroma

documents = [

Document(page\_content="Test 1", metadata={}),

Document(page\_content="Test 2", metadata={})

]

ids = ["doc1", "doc2"]

vector\_store.add\_documents(documents, ids=ids)

# Verify IDs were passed

mock\_chroma\_class.from\_documents.assert\_called\_once()

call\_args = mock\_chroma\_class.from\_documents.call\_args

assert call\_args.kwargs["ids"] == ids

@patch('src.vector\_store.chroma\_store.Chroma')

def test\_add\_documents\_error\_handling(self, mock\_chroma\_class, vector\_store):

"""Test error handling when adding documents fails."""

mock\_chroma\_class.from\_documents.side\_effect = Exception("Database error")

documents = [Document(page\_content="Test", metadata={})]

with pytest.raises(VectorStoreError, match="Failed to add documents"):

vector\_store.add\_documents(documents)

def test\_similarity\_search\_not\_initialized(self, vector\_store):

"""Test similarity search when vector store not initialized."""

with pytest.raises(VectorStoreError, match="Vector store not initialized"):

vector\_store.similarity\_search("test query")

def test\_similarity\_search\_success(self, vector\_store):

"""Test successful similarity search."""

# Setup mock vector store

mock\_vectorstore = Mock()

expected\_results = [

Document(page\_content="Result 1", metadata={"score": 0.9}),

Document(page\_content="Result 2", metadata={"score": 0.8})

]

mock\_vectorstore.similarity\_search.return\_value = expected\_results

vector\_store.\_vectorstore = mock\_vectorstore

# Perform search

results = vector\_store.similarity\_search("test query", k=2)

# Verify

assert results == expected\_results

mock\_vectorstore.similarity\_search.assert\_called\_once\_with(

query="test query",

k=2,

filter=None

)

def test\_similarity\_search\_with\_filter(self, vector\_store):

"""Test similarity search with metadata filter."""

mock\_vectorstore = Mock()

mock\_vectorstore.similarity\_search.return\_value = []

vector\_store.\_vectorstore = mock\_vectorstore

filter\_dict = {"source": "specific.pdf"}

vector\_store.similarity\_search("query", k=5, filter=filter\_dict)

mock\_vectorstore.similarity\_search.assert\_called\_once\_with(

query="query",

k=5,

filter=filter\_dict

)

def test\_similarity\_search\_with\_score(self, vector\_store):

"""Test similarity search with scores."""

mock\_vectorstore = Mock()

expected\_results = [

(Document(page\_content="Result 1", metadata={}), 0.95),

(Document(page\_content="Result 2", metadata={}), 0.85)

]

mock\_vectorstore.similarity\_search\_with\_score.return\_value = expected\_results

vector\_store.\_vectorstore = mock\_vectorstore

results = vector\_store.similarity\_search\_with\_score("test query", k=2)

assert results == expected\_results

assert len(results) == 2

assert all(isinstance(score, float) for \_, score in results)

def test\_persist\_success(self, vector\_store):

"""Test successful persistence."""

mock\_vectorstore = Mock()

vector\_store.\_vectorstore = mock\_vectorstore

vector\_store.persist()

mock\_vectorstore.persist.assert\_called\_once()

def test\_persist\_no\_directory(self, mock\_embedding\_model):

"""Test persist when no directory configured."""

store = ChromaVectorStore(

embedding\_model=mock\_embedding\_model,

persist\_directory=None

)

store.\_vectorstore = Mock()

# Should not raise error, just skip

store.persist()

store.\_vectorstore.persist.assert\_not\_called()

def test\_persist\_error\_handling(self, vector\_store):

"""Test error handling during persistence."""

mock\_vectorstore = Mock()

mock\_vectorstore.persist.side\_effect = Exception("Disk full")

vector\_store.\_vectorstore = mock\_vectorstore

with pytest.raises(VectorStoreError, match="Failed to persist"):

vector\_store.persist()

@patch('src.vector\_store.chroma\_store.Chroma')

def test\_load\_success(self, mock\_chroma\_class, vector\_store, mock\_embedding\_model):

"""Test successful loading from disk."""

mock\_chroma = Mock()

mock\_chroma.\_collection.count.return\_value = 100

mock\_chroma\_class.return\_value = mock\_chroma

vector\_store.load()

mock\_chroma\_class.assert\_called\_once\_with(

persist\_directory=vector\_store.persist\_directory,

embedding\_function=mock\_embedding\_model,

collection\_name="test\_collection"

)

assert vector\_store.\_vectorstore == mock\_chroma

def test\_load\_no\_directory(self, mock\_embedding\_model):

"""Test loading when no persist directory."""

store = ChromaVectorStore(

embedding\_model=mock\_embedding\_model,

persist\_directory=None

)

with pytest.raises(VectorStoreError, match="No persist directory configured"):

store.load()

def test\_load\_directory\_not\_exists(self, vector\_store):

"""Test loading when directory doesn't exist."""

vector\_store.persist\_directory = "/nonexistent/path"

with pytest.raises(VectorStoreError, match="Persist directory does not exist"):

vector\_store.load()

def test\_delete\_collection(self, vector\_store):

"""Test deleting collection."""

mock\_vectorstore = Mock()

vector\_store.\_vectorstore = mock\_vectorstore

vector\_store.delete\_collection()

mock\_vectorstore.delete\_collection.assert\_called\_once()

assert vector\_store.\_vectorstore is None

def test\_clear\_persist\_directory(self, temp\_persist\_dir):

"""Test clearing persist directory."""

# Create some files in the directory

test\_file = Path(temp\_persist\_dir) / "test.txt"

test\_file.write\_text("test")

store = ChromaVectorStore(

embedding\_model=Mock(),

persist\_directory=str(temp\_persist\_dir)

)

store.clear\_persist\_directory()

assert not Path(temp\_persist\_dir).exists()

def test\_as\_retriever(self, vector\_store):

"""Test getting retriever from vector store."""

mock\_vectorstore = Mock()

mock\_retriever = Mock()

mock\_vectorstore.as\_retriever.return\_value = mock\_retriever

vector\_store.\_vectorstore = mock\_vectorstore

retriever = vector\_store.as\_retriever(search\_kwargs={"k": 10})

assert retriever == mock\_retriever

mock\_vectorstore.as\_retriever.assert\_called\_once\_with(search\_kwargs={"k": 10})

def test\_as\_retriever\_not\_initialized(self, vector\_store):

"""Test getting retriever when store not initialized."""

with pytest.raises(VectorStoreError, match="Vector store not initialized"):

vector\_store.as\_retriever()

@patch('src.vector\_store.chroma\_store.logger')

def test\_logging\_stats(self, mock\_logger, vector\_store):

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

mock\_vectorstore = Mock()

mock\_collection = Mock()

mock\_collection.count.return\_value = 50

mock\_vectorstore.\_collection = mock\_collection

vector\_store.\_vectorstore = mock\_vectorstore

# Trigger stats logging

vector\_store.\_log\_store\_stats()

# Check that count was logged

assert mock\_logger.info.called

log\_messages = [call[0][0] for call in mock\_logger.info.call\_args\_list]

assert any("50 documents" in msg for msg in log\_messages)