"""

Unit tests for QA chain implementation.

This module tests the question-answering chain functionality

including retrieval and generation.

"""

import pytest

from unittest.mock import Mock, patch, MagicMock

import time

from src.retrieval.qa\_chain import QAChain, DEFAULT\_QA\_PROMPT

from src.utils.exceptions import RetrievalError

from langchain.schema import Document

from langchain.prompts import PromptTemplate

class TestQAChain:

"""Test suite for QAChain."""

@pytest.fixture

def mock\_llm(self):

"""Create a mock LLM."""

return Mock()

@pytest.fixture

def mock\_vector\_store(self):

"""Create a mock vector store."""

mock\_store = Mock()

mock\_retriever = Mock()

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

return mock\_store

@pytest.fixture

def qa\_chain(self, mock\_llm, mock\_vector\_store):

"""Create a QA chain instance."""

with patch('src.retrieval.qa\_chain.RetrievalQA'):

chain = QAChain(

llm=mock\_llm,

vector\_store=mock\_vector\_store,

chain\_type="stuff",

return\_source\_documents=True

)

return chain

def test\_initialization\_default\_params(self, mock\_llm, mock\_vector\_store):

"""Test QA chain initialization with default parameters."""

with patch('src.retrieval.qa\_chain.RetrievalQA') as mock\_retrieval\_qa:

chain = QAChain(llm=mock\_llm, vector\_store=mock\_vector\_store)

assert chain.llm == mock\_llm

assert chain.vector\_store == mock\_vector\_store

assert chain.chain\_type == "stuff"

assert chain.return\_source\_documents is True

assert chain.verbose is False

assert chain.retriever\_kwargs == {"k": 4}

# Verify retriever was created

mock\_vector\_store.as\_retriever.assert\_called\_once\_with(k=4)

def test\_initialization\_custom\_params(self, mock\_llm, mock\_vector\_store):

"""Test QA chain initialization with custom parameters."""

custom\_prompt = "Custom prompt: {context} {question}"

retriever\_kwargs = {"k": 10, "fetch\_k": 20}

with patch('src.retrieval.qa\_chain.RetrievalQA'):

chain = QAChain(

llm=mock\_llm,

vector\_store=mock\_vector\_store,

chain\_type="map\_reduce",

return\_source\_documents=False,

prompt\_template=custom\_prompt,

retriever\_kwargs=retriever\_kwargs,

verbose=True

)

assert chain.chain\_type == "map\_reduce"

assert chain.return\_source\_documents is False

assert chain.verbose is True

assert chain.retriever\_kwargs == retriever\_kwargs

assert chain.prompt.template == custom\_prompt

@patch('src.retrieval.qa\_chain.RetrievalQA')

def test\_chain\_creation\_success(self, mock\_retrieval\_qa\_class, mock\_llm, mock\_vector\_store):

"""Test successful chain creation."""

mock\_chain = Mock()

mock\_retrieval\_qa\_class.from\_chain\_type.return\_value = mock\_chain

chain = QAChain(llm=mock\_llm, vector\_store=mock\_vector\_store)

# Verify RetrievalQA was called correctly

mock\_retrieval\_qa\_class.from\_chain\_type.assert\_called\_once()

call\_args = mock\_retrieval\_qa\_class.from\_chain\_type.call\_args

assert call\_args.kwargs["llm"] == mock\_llm

assert call\_args.kwargs["chain\_type"] == "stuff"

assert call\_args.kwargs["return\_source\_documents"] is True

assert "prompt" in call\_args.kwargs["chain\_type\_kwargs"]

@patch('src.retrieval.qa\_chain.RetrievalQA')

def test\_chain\_creation\_failure(self, mock\_retrieval\_qa\_class, mock\_llm, mock\_vector\_store):

"""Test chain creation failure handling."""

mock\_retrieval\_qa\_class.from\_chain\_type.side\_effect = Exception("Chain error")

with pytest.raises(RetrievalError, match="Failed to create QA chain"):

QAChain(llm=mock\_llm, vector\_store=mock\_vector\_store)

def test\_run\_success(self, qa\_chain):

"""Test successful query execution."""

# Setup mock chain response

mock\_response = {

"result": "The answer is 42.",

"source\_documents": [

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

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

]

}

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.return\_value = mock\_response

# Run query

query = "What is the meaning of life?"

result = qa\_chain.run(query)

# Verify response structure

assert result["query"] == query

assert result["answer"] == "The answer is 42."

assert len(result["source\_documents"]) == 2

assert "metadata" in result

assert result["metadata"]["num\_source\_documents"] == 2

assert result["metadata"]["chain\_type"] == "stuff"

assert "processing\_time" in result["metadata"]

# Verify chain was called

qa\_chain.chain.invoke.assert\_called\_once\_with({"query": query})

def test\_run\_empty\_response(self, qa\_chain):

"""Test handling of empty response."""

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.return\_value = {

"result": "",

"source\_documents": []

}

result = qa\_chain.run("Test query")

assert result["answer"] == ""

assert result["source\_documents"] == []

assert result["metadata"]["num\_source\_documents"] == 0

def test\_run\_error\_handling(self, qa\_chain):

"""Test error handling during query execution."""

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.side\_effect = Exception("LLM error")

with pytest.raises(RetrievalError, match="QA chain execution failed"):

qa\_chain.run("Test query")

def test\_run\_batch\_success(self, qa\_chain):

"""Test batch query processing."""

queries = ["Question 1?", "Question 2?", "Question 3?"]

# Mock successful responses

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.side\_effect = [

{"result": f"Answer {i}", "source\_documents": []}

for i in range(1, 4)

]

results = qa\_chain.run\_batch(queries)

assert len(results) == 3

for i, result in enumerate(results):

assert result["query"] == queries[i]

assert result["answer"] == f"Answer {i+1}"

def test\_run\_batch\_partial\_failure(self, qa\_chain):

"""Test batch processing with some failures."""

queries = ["Q1?", "Q2?", "Q3?"]

# Mock mixed responses (success, failure, success)

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.side\_effect = [

{"result": "Answer 1", "source\_documents": []},

Exception("Query 2 failed"),

{"result": "Answer 3", "source\_documents": []}

]

results = qa\_chain.run\_batch(queries)

assert len(results) == 3

assert results[0]["answer"] == "Answer 1"

assert results[1]["answer"] is None

assert "error" in results[1]

assert results[2]["answer"] == "Answer 3"

def test\_get\_relevant\_documents(self, qa\_chain):

"""Test retrieving documents without generation."""

expected\_docs = [

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

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

]

qa\_chain.retriever.get\_relevant\_documents.return\_value = expected\_docs

docs = qa\_chain.get\_relevant\_documents("test query")

assert docs == expected\_docs

qa\_chain.retriever.get\_relevant\_documents.assert\_called\_once\_with("test query")

def test\_get\_relevant\_documents\_error(self, qa\_chain):

"""Test error handling in document retrieval."""

qa\_chain.retriever.get\_relevant\_documents.side\_effect = Exception("Retrieval error")

with pytest.raises(RetrievalError, match="Document retrieval failed"):

qa\_chain.get\_relevant\_documents("test query")

def test\_update\_retriever\_kwargs(self, qa\_chain, mock\_vector\_store):

"""Test updating retriever parameters."""

with patch('src.retrieval.qa\_chain.RetrievalQA'):

# Update parameters

qa\_chain.update\_retriever\_kwargs(k=10, fetch\_k=20)

# Verify parameters updated

assert qa\_chain.retriever\_kwargs["k"] == 10

assert qa\_chain.retriever\_kwargs["fetch\_k"] == 20

# Verify new retriever created

assert mock\_vector\_store.as\_retriever.call\_count == 2 # Initial + update

mock\_vector\_store.as\_retriever.assert\_called\_with(k=10, fetch\_k=20)

@patch('src.retrieval.qa\_chain.logger')

def test\_logging\_metrics(self, mock\_logger, qa\_chain):

"""Test that metrics are logged."""

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.return\_value = {

"result": "Test answer",

"source\_documents": [

Document(page\_content="A" \* 100, metadata={}),

Document(page\_content="B" \* 200, metadata={})

]

}

qa\_chain.run("Test query")

# Check metrics were logged

assert mock\_logger.info.called

# Find the metrics log call

metrics\_logged = False

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

if "QA chain metrics" in str(call):

metrics\_logged = True

# Check that metrics include expected fields

extra = call[1].get('extra', {})

assert 'query\_length' in extra

assert 'answer\_length' in extra

assert 'num\_sources' in extra

assert 'processing\_time\_seconds' in extra

break

assert metrics\_logged

def test\_processing\_time\_measurement(self, qa\_chain):

"""Test that processing time is measured correctly."""

qa\_chain.chain = Mock()

qa\_chain.chain.invoke.return\_value = {

"result": "Answer",

"source\_documents": []

}

# Mock time to control processing time

with patch('time.time') as mock\_time:

mock\_time.side\_effect = [100.0, 101.5] # 1.5 seconds

result = qa\_chain.run("Query")

assert result["metadata"]["processing\_time"] == 1.5

def test\_custom\_prompt\_template(self, mock\_llm, mock\_vector\_store):

"""Test using custom prompt template."""

custom\_template = """

Use the context to answer the question.

Context: {context}

Question: {question}

Answer in one sentence.

"""

with patch('src.retrieval.qa\_chain.RetrievalQA') as mock\_retrieval\_qa:

chain = QAChain(

llm=mock\_llm,

vector\_store=mock\_vector\_store,

prompt\_template=custom\_template

)

assert chain.prompt.template == custom\_template

assert isinstance(chain.prompt, PromptTemplate)

assert set(chain.prompt.input\_variables) == {"context", "question"}