

# Unit Test

- [Template](#)
- [Unit test cho hệ thống RAG](#)
- [Repo mẫu cho hệ thống RAG có unit test](#)

# Template

```
## Repo: rag_unit_test_template
```

```
# 📁 Folder Structure
```

```
rag_unit_test_template/
```

```
├─ rag_pipeline.py
```

```
├─ retriever.py
```

```
├─ generator.py
```

```
├─ test/
```

```
│   └─ test_pipeline.py
```

```
│   └─ test_retriever.py
```

```
│   └─ test_generator.py
```

```
└─ requirements.txt
```

```
└─ README.md
```

```
# 📄 rag_pipeline.py
```

```
class RAGPipeline:
```

```
    def __init__(self, retriever, generator):
```

```
        self.retriever = retriever
```

```
        self.generator = generator
```

```
    def run(self, query):
```

```
        docs = self.retriever.retrieve(query)
```

```
        return self.generator.generate(query, docs)
```

```
# 📄 retriever.py
```

```
class SimpleRetriever:
```

```
    def __init__(self, corpus):
```

```
        self.corpus = corpus
```

```
    def retrieve(self, query):
```

```
return [doc for doc in self.corpus if query.lower().split()[0] in doc.lower()]
```

```
# test/generator.py
```

```
class DummyGenerator:
```

```
    def generate(self, query, documents):
```

```
        return f"Answering '{query}' using: {documents[0]}"
```

```
# test/test_pipeline.py
```

```
import pytest
```

```
from unittest.mock import MagicMock
```

```
from rag_pipeline import RAGPipeline
```

```
def test_rag_pipeline_returns_expected_output():
```

```
    mock_retriever = MagicMock()
```

```
    mock_retriever.retrieve.return_value = ["Mock doc"]
```

```
    mock_generator = MagicMock()
```

```
    mock_generator.generate.return_value = "Mock answer"
```

```
    pipeline = RAGPipeline(mock_retriever, mock_generator)
```

```
    result = pipeline.run("What is mock?")
```

```
    assert result == "Mock answer"
```

```
    mock_retriever.retrieve.assert_called_once_with("What is mock?")
```

```
    mock_generator.generate.assert_called_once_with("What is mock?", ["Mock doc"])
```

```
# test/test_retriever.py
```

```
from retriever import SimpleRetriever
```

```
def test_simple_retriever_returns_match():
```

```
    retriever = SimpleRetriever(["Paris is the capital of France."])
```

```
    docs = retriever.retrieve("What is the capital of France?")
```

```
    assert any("Paris" in doc for doc in docs)
```

```
# test/test_generator.py
```

```
from generator import DummyGenerator
```

```
def test_dummy_generator_formats_output():
```

```
    gen = DummyGenerator()
```

```
    result = gen.generate("What is AI?", ["AI is artificial intelligence."])
```

```
    assert "What is AI?" in result
```

```
    assert "AI is artificial intelligence." in result
```

```
# requirements.txt
```

```
pytest
```

```
# README.md
```

```
# RAG Unit Test Template
```

This repo demonstrates how to build a unit-tested RAG (Retrieval-Augmented Generation) system.

```
## Components
```

- SimpleRetriever: Retrieves documents based on keyword match
- DummyGenerator: Generates answer using context
- RAGPipeline: Combines retriever + generator

```
## Unit Tests
```

Run all tests with:

```
```bash
```

```
pytest test/ -v
```

```
```
```

You can expand this repo with real vector stores (FAISS/Chroma) or LLMs (Flan-T5) later.

# Unit test cho hệ thống RAG

Viết **unit test cho hệ thống RAG (Retrieval-Augmented Generation)** giúp đảm bảo rằng các thành phần chính như Retriever, Generator, và Data Pipeline hoạt động chính xác, độc lập và có thể kiểm soát được. Dưới đây là hướng dẫn thực hành cách viết **unit test** cho hệ thống RAG sử dụng Python (với `pytest`) và thư viện phổ biến như LangChain hoặc custom code.

## 1. Các thành phần cần kiểm thử

Hệ thống RAG thường gồm:

1. **Retriever** – Tìm kiếm các đoạn văn bản phù hợp từ kho dữ liệu.
2. **Generator** – Sinh câu trả lời dựa trên ngữ cảnh và câu hỏi.
3. **RAG Pipeline** – Tổng thể pipeline kết hợp cả retriever và generator.
4. **Post-processing** (tùy chọn) – Xử lý đầu ra của LLM.

## 2. Cấu trúc ví dụ RAG

Giả sử bạn có pipeline như sau:

```
class RAGPipeline:
    def __init__(self, retriever, generator):
        self.retriever = retriever
        self.generator = generator

    def run(self, query):
        documents = self.retriever.retrieve(query)
        return self.generator.generate(query, documents)
```

## 3. Cách viết unit test

### 3.1. Tạo file `test_rag.py`

```
import pytest
from unittest.mock import MagicMock
from rag_pipeline import RAGPipeline

def test_rag_pipeline_returns_expected_output():
    # Mock retriever
```

```

mock_retriever = MagicMock()
mock_retriever.retrieve.return_value = ["This is a test document."]

# Mock generator
mock_generator = MagicMock()
mock_generator.generate.return_value = "This is a generated answer."

# Create pipeline
pipeline = RAGPipeline(mock_retriever, mock_generator)
result = pipeline.run("What is this?")

# Assertions
mock_retriever.retrieve.assert_called_once_with("What is this?")
mock_generator.generate.assert_called_once_with("What is this?", ["This is a test document."])
assert result == "This is a generated answer."

```

### 3.2. Test retriever riêng biệt

```

def test_retriever_returns_relevant_docs():
    from my_retriever import SimpleRetriever
    retriever = SimpleRetriever(["Paris is the capital of France."])
    docs = retriever.retrieve("What is the capital of France?")
    assert any("Paris" in doc for doc in docs)

```

## 4. Công cụ và kỹ thuật nâng cao

- **pytest fixtures** để khởi tạo dữ liệu.
- **mocking LLM API calls** để tránh chi phí gọi thực tế.
- **test coverage** để kiểm tra phần nào chưa được test.
- **snapshot testing** để so sánh kết quả sinh tự động với mẫu.

## 5. Chạy test

```

pytest test_rag.py -v

```

## Gợi ý mở rộng

- Test tích hợp: Chạy full pipeline với vectordb thật (FAISS, Chroma).
- So sánh kết quả RAG vs non-RAG (benchmark chất lượng sinh).
- Kết hợp với **Promptfoo**, **LangSmith**, hoặc **TruLens** để test LLM đầu ra tự động.



# Repo mẫu cho hệ thống RAG có unit test

Cấu trúc thư mục repo: `rag-pipeline-example/`

```
rag-pipeline-example/  
├─ rag_pipeline/  
│  ├─ __init__.py  
│  ├─ retriever.py  
│  ├─ generator.py  
│  └─ pipeline.py  
├─ tests/  
│  ├─ __init__.py  
│  └─ test_pipeline.py  
├─ requirements.txt  
└─ README.md
```

## 1. `rag_pipeline/retriever.py`

```
class SimpleRetriever:  
    def __init__(self, documents):  
        self.documents = documents  
  
    def retrieve(self, query):  
        # Tìm văn bản chứa từ khoá  
        return [doc for doc in self.documents if any(word.lower() in doc.lower() for word in query.split())]
```

## 2. `rag_pipeline/generator.py`

```
class SimpleGenerator:  
    def generate(self, query, documents):  
        if not documents:  
            return "I don't know."  
        return f"Based on the documents, the answer to '{query}' is related to: {documents[0]}"
```

## 3. `rag_pipeline/pipeline.py`



```
class RAGPipeline:
    def __init__(self, retriever, generator):
        self.retriever = retriever
        self.generator = generator

    def run(self, query):
        docs = self.retriever.retrieve(query)
        return self.generator.generate(query, docs)
```

#### 4. tests/test\_pipeline.py

```
import pytest
from unittest.mock import MagicMock
from rag_pipeline.pipeline import RAGPipeline
from rag_pipeline.retriever import SimpleRetriever
from rag_pipeline.generator import SimpleGenerator

def test_pipeline_with_mock():
    retriever = MagicMock()
    generator = MagicMock()

    retriever.retrieve.return_value = ["Mock document"]
    generator.generate.return_value = "Mock answer"

    rag = RAGPipeline(retriever, generator)
    result = rag.run("What is this?")

    assert result == "Mock answer"
    retriever.retrieve.assert_called_once()
    generator.generate.assert_called_once()

def test_pipeline_with_real_components():
    retriever = SimpleRetriever([
        "Paris is the capital of France.",
        "Berlin is the capital of Germany."
    ])
    generator = SimpleGenerator()

    rag = RAGPipeline(retriever, generator)
    result = rag.run("What is the capital of France?")
```

```
assert "Paris" in result
```

5. requirements.txt

```
pytest
```

Bạn có thể thêm `langchain`, `transformers`, `faiss-cpu` nếu mở rộng thực tế.

6. README.md

```
# RAG Pipeline Example
```

```
This is a simple Retrieval-Augmented Generation pipeline with unit tests.
```

```
## Run tests
```

```
```bash
```

```
pip install -r requirements.txt
```

```
pytest tests/ -v
```

```
---
```

```
## ► Cách chạy
```

```
```bash
```

```
# Tải repo
```

```
git clone <repo-url> rag-pipeline-example
```

```
cd rag-pipeline-example
```

```
# Cài đặt & chạy test
```

```
pip install -r requirements.txt
```

```
pytest tests/
```