🧪 Guía Completa de Pruebas de Caja Negra - Sistema RAG Aconex

📑 Tabla de Contenidos

🎯 ¿Qué son las Pruebas de Caja Negra?

Las **pruebas de caja negra** (black box testing) validan el comportamiento del sistema desde el punto de vista del usuario, **sin conocer la implementación interna**:

Características:

✅ **Se enfoca en**: Entradas → Salidas
✅ **Valida**: Comportamiento esperado según especificaciones
✅ **No analiza**: Código interno, algoritmos, estructuras de datos
✅ **Simula**: Uso real del sistema por parte de usuarios

Diferencia con Caja Blanca:

Aspecto	Caja Negra ✅	Caja Blanca ❌
Conocimiento	Solo interfaz pública	Implementación interna
Enfoque	Funcionalidad completa	Lógica y rutas de código
Validación	Input → Output	Cobertura de líneas
Mantenimiento	Independiente de cambios internos	Requiere actualización con cambios

---

📊 Pruebas Implementadas

El proyecto cuenta con **22 pruebas de caja negra** distribuidas en 4 escenarios:

1️⃣ **Chat Conversacional RAG** (7 tests)

Test	Archivo	Tipo	Descripción
`test_chat_with_document_context`	`test_chat.py`	✅ Positivo	Chat con documentos relevantes

`test_chat_without_relevant_documents`	`test_chat.py`	⚠️ Negativo	Chat sin contexto disponible
`test_save_chat_history`	`test_chat.py`	✅ Positivo	Guardar historial de conversación
`test_get_chat_history`	`test_chat.py`	✅ Positivo	Recuperar historial de usuario
`test_chat_with_empty_question`	`test_chat.py`	⚠️ Negativo	Pregunta vacía
`test_save_chat_history_database_error`	`test_chat.py`	⚠️ Negativo	Error de BD al guardar
`test_get_chat_history_no_results`	`test_chat.py`	⚠️ Negativo	Usuario sin historial

2️⃣ **Búsqueda Semántica** (4 tests)

Test	Archivo	Tipo	Descripción
`test_semantic_search_basic`	`test_search.py`	✅ Positivo	Búsqueda con resultados
`test_semantic_search_with_project_filter`	`test_search.py`	✅ Positivo	Filtrado por proyecto
`test_semantic_search_empty_query`	`test_search.py`	⚠️ Negativo	Query vacía
`test_semantic_search_invalid_project_id`	`test_search.py`	⚠️ Negativo	Proyecto inexistente

3️⃣ **Upload de Documentos** (4 tests)

Test	Archivo	Tipo	Descripción
`test_extract_text_from_txt`	`test_upload.py`	✅ Positivo	Extracción de texto plano
`test_generate_document_id_unique`	`test_upload.py`	✅ Positivo	Generación de IDs únicos
`test_extract_text_file_not_found`	`test_upload.py`	⚠️ Negativo	Archivo inexistente
`test_extract_text_invalid_encoding`	`test_upload.py`	⚠️ Negativo	Encoding corrupto

4️⃣ **Ingesta de Datos** (4 tests)

Test	Archivo	Tipo	Descripción
`test_normalize_doc_complete`	`test_ingest.py`	✅ Positivo	Normalización completa
`test_iter_docs_from_file_json_and_ndjson`	`test_ingest.py`	✅ Positivo	Lectura JSON/NDJSON
`test_normalize_doc_missing_fields`	`test_ingest.py`	⚠️ Negativo	Campos faltantes
`test_iter_docs_invalid_json`	`test_ingest.py`	⚠️ Negativo	JSON malformado

5️⃣ **Utilidades Core** (3 tests)

Test	Archivo	Tipo	Descripción
`test_simple_chunk_with_overlap`	`test_utils.py`	✅ Positivo	Chunking con overlap
`test_get_db_connection_success`	`test_utils.py`	✅ Positivo	Conexión a BD exitosa
`test_simple_chunk_invalid_parameters`	`test_utils.py`	⚠️ Negativo	Parámetros inválidos

🚀 Cómo Ejecutar las Pruebas

Prerequisitos

```
# 1. Activar entorno virtual
& .\.venv311\Scripts\Activate.ps1

# 2. Verificar instalación de dependencias
pip install -r requirements-test.txt
```

Opciones de Ejecución

✅ **Ejecutar TODAS las pruebas**

```
# Opción 1: Usar script helper
.\run_tests.ps1 all

# Opción 2: Comando directo
pytest tests/ -v
```

Salida esperada:

```
tests/test_chat.py::test_chat_with_document_context PASSED          [ 14%]
tests/test_chat.py::test_chat_without_relevant_documents PASSED     [ 28%]
tests/test_search.py::test_semantic_search_basic PASSED             [ 42%]
...
======================== 22 passed in 5.23s ========================
```

📊 Ejecutar con reporte de cobertura

```
# Generar reporte HTML
.\run_tests.ps1 cov

# O manualmente:
pytest tests/ --cov=app --cov-report=html --cov-report=term-missing -v
```

Ver reporte:

```
# Abrir en navegador
Start-Process htmlcov\index.html
```

🎯 Ejecutar por categoría

```
# Solo tests de Chat
pytest tests/test_chat.py -v

# Solo tests de Búsqueda
pytest tests/test_search.py -v

# Solo tests de Upload
pytest tests/test_upload.py -v

# Solo tests de Ingesta
pytest tests/test_ingest.py -v

# Solo tests de Utilidades
pytest tests/test_utils.py -v
```

-

⚡ Ejecución rápida (paralelo)

```
# Usar todos los cores disponibles
.\run_tests.ps1 fast

# O manualmente:
pytest tests/ -n auto -v
```

🔍 Ejecutar solo tests que fallaron

```
# Re-ejecutar últimos tests fallidos
.\run_tests.ps1 failing

# O manualmente:
pytest tests/ --lf -v
```

📝 Modo verbose con salida detallada

```
# Máximo detalle
pytest tests/ -vv -s

# Con traceback completo
pytest tests/ -v --tb=long
```

📈 Resultados Esperados

✅ Ejecución Exitosa

```
============================= test session starts ==========================
platform win32 -- Python 3.11.0, pytest-9.0.1

collected 22 items

tests/test_chat.py::test_chat_with_document_context PASSED          [  4%]
tests/test_chat.py::test_chat_without_relevant_documents PASSED     [  9%]
tests/test_chat.py::test_save_chat_history PASSED                   [ 13%]
tests/test_chat.py::test_get_chat_history PASSED                    [ 18%]
tests/test_chat.py::test_chat_with_empty_question PASSED            [ 22%]
tests/test_chat.py::test_save_chat_history_database_error PASSED    [ 27%]
tests/test_chat.py::test_get_chat_history_no_results PASSED         [ 31%]

tests/test_search.py::test_semantic_search_basic PASSED             [ 36%]
tests/test_search.py::test_semantic_search_with_project_filter PASSED [ 40%]
```

```
tests/test_search.py::test_semantic_search_empty_query PASSED       [ 45%]
tests/test_search.py::test_semantic_search_invalid_project_id PASSED [ 50%]

tests/test_upload.py::test_extract_text_from_txt PASSED             [ 54%]
tests/test_upload.py::test_generate_document_id_unique PASSED       [ 59%]
tests/test_upload.py::test_extract_text_file_not_found PASSED       [ 63%]
tests/test_upload.py::test_extract_text_invalid_encoding PASSED     [ 68%]

tests/test_ingest.py::test_normalize_doc_complete PASSED            [ 72%]
tests/test_ingest.py::test_iter_docs_from_file_json_and_ndjson PASSED [ 77%]
tests/test_ingest.py::test_normalize_doc_missing_fields PASSED      [ 81%]
tests/test_ingest.py::test_iter_docs_invalid_json PASSED            [ 86%]

tests/test_utils.py::test_simple_chunk_with_overlap PASSED          [ 90%]
tests/test_utils.py::test_get_db_connection_success PASSED          [ 95%]
tests/test_utils.py::test_simple_chunk_invalid_parameters PASSED    [100%]

======================== 22 passed in 5.23s ========================
```

📊 Resumen de Cobertura

```
---------- coverage: platform win32, python 3.11.0 -----------
Name                    Stmts   Miss  Cover   Missing
-------------------------------------------------------
app/__init__.py            12      0   100%
app/api.py                145     18    88%   45-52, 89-95
app/analytics.py           67      8    88%   34-38, 67-70
app/ingest.py              98     12    88%   67-72, 145-150
app/search_core.py        123     15   88%   89-95, 201-208
app/upload.py              87     11    87%   56-62, 134-139
app/utils.py               45      3    93%   78-80
-------------------------------------------------------
TOTAL                     577     67    88%
```

📋 Casos de Prueba Detallados

🔵 Escenario 1: Chat Conversacional RAG

Test 1.1: Chat con Documentos Relevantes ✅

Archivo: `tests/test_chat.py::test_chat_with_document_context`

Propósito: Validar el flujo completo de RAG (Retrieval-Augmented Generation)

Entrada (Input):

```
request = ChatRequest(
    question="¿Qué incluye el plan maestro de arquitectura?",
    max_context_docs=5,
    session_id="test-session-001"
)
```

Salida Esperada (Output):

```
response = ChatResponse(
    question="¿Qué incluye el plan maestro de arquitectura?",
    answer="Basándome en la documentación técnica... [respuesta generada]",
    sources=[
        {"id": "DOC-ARQ-001", "title": "Plan Maestro", "score": 0.87}
    ],
    context_used="[snippets de documentos relevantes]",
    session_id="test-session-001"
)
```

Validaciones de Caja Negra:

- ✅ Respuesta contiene información del contexto
- ✅ Lista de sources no está vacía
- ✅ Context_used tiene contenido sustancial (> 100 caracteres)
- ✅ Session_id se preserva o genera

Ejecutar:

```
pytest tests/test_chat.py::test_chat_with_document_context -v
```

Test 1.2: Chat sin Documentos Relevantes ⚠️

Archivo: `tests/test_chat.py::test_chat_without_relevant_documents`

Propósito: Validar que el sistema maneja apropiadamente cuando NO hay contexto

Entrada (Input):

```
request = ChatRequest(
    question="¿Cuál es la receta del pastel de chocolate?",  # Fuera de cont
    max_context_docs=5
)
```

Salida Esperada (Output):

```
response = ChatResponse(
    answer="No encuentro información relevante en los documentos disponibles
    sources=[],  # Lista vacía
    context_used=""  # String vacío
)
```

Validaciones de Caja Negra:

- ✅ Sistema no crashea
- ✅ Respuesta indica "no encuentro información"
- ✅ Lista de sources está vacía
- ✅ No genera "alucinaciones" sin contexto

Ejecutar:

```
pytest tests/test_chat.py::test_chat_without_relevant_documents -v
```

🔵 Escenario 2: Búsqueda Semántica

Test 2.1: Búsqueda Básica ✅

Archivo: `tests/test_search.py::test_semantic_search_basic`

Propósito: Validar búsqueda semántica con ranking híbrido

Entrada (Input):

```
query = "construcción sismo resistente"
project_id = None
top_k = 10
```

Salida Esperada (Output):

```
results = [
    {
        "document_id": "DOC-001",
        "title": "Manual de Construcción Sísmica",
        "snippet": "Normas NSR-10...",
        "score": 0.78  # Entre 0 y 1
    },
    ...
]
```

Validaciones de Caja Negra:

- ✅ Retorna lista de documentos
- ✅ Cada documento tiene score entre 0 y 1
- ✅ Documentos ordenados por relevancia (score descendente)
- ✅ No más de `top_k` resultados

Ejecutar:

```
pytest tests/test_search.py::test_semantic_search_basic -v
```

Test 2.2: Búsqueda con Filtro de Proyecto ✅

Archivo: `tests/test_search.py::test_semantic_search_with_project_filter`

Propósito: Validar aislamiento multi-tenant (multi-tenancy)

Entrada (Input):

```
query = "arquitectura educativa"
project_id = "PROYECTO-EDUCATIVO"  # Filtro específico
top_k = 20
```

Salida Esperada (Output):

```
results = [
    {"document_id": "EDU-001", "project_id": "PROYECTO-EDUCATIVO", ...},
    {"document_id": "EDU-002", "project_id": "PROYECTO-EDUCATIVO", ...},
    # TODOS los resultados del mismo proyecto
]
```

Validaciones de Caja Negra:

- ✅ Todos los resultados pertenecen al proyecto especificado
- ✅ No se mezclan documentos de otros proyectos
- ✅ Aislamiento de datos garantizado

Ejecutar:

```
pytest tests/test_search.py::test_semantic_search_with_project_filter -v
```

🔵 Escenario 3: Upload de Documentos

Test 3.1: Extracción de Texto TXT ✅

Archivo: `tests/test_upload.py::test_extract_text_from_txt`

Propósito: Validar extracción básica de texto plano

Entrada (Input):

```
archivo_txt = "documento.txt"
contenido = """Manual de Seguridad en Construcción
Procedimientos EPP..."""
```

Salida Esperada (Output):

```
texto_extraido = "Manual de Seguridad en Construcción\nProcedimientos EPP...
```

Validaciones de Caja Negra:

- ✅ Texto extraído contiene palabras clave del archivo
- ✅ Longitud del texto es sustancial (> 50 caracteres)
- ✅ Encoding UTF-8 preservado (caracteres especiales)

Ejecutar:

```
pytest tests/test_upload.py::test_extract_text_from_txt -v
```

Test 3.2: Generación de IDs Únicos ✅

Archivo: `tests/test_upload.py::test_generate_document_id_unique`

Propósito: Validar generación de identificadores MD5

Entrada (Input):

```
filename = "manual.txt"
content = "Contenido del documento"
```

Salida Esperada (Output):

```
document_id = "a1b2c3d4e5f67890123456789012345678901234ab"  # Hash MD5 (32 chars)
```

Validaciones de Caja Negra:

- ✅ ID tiene exactamente 32 caracteres hexadecimales
- ✅ Solo contiene [0-9a-f]
- ✅ Cambios en filename/content generan IDs diferentes
- ✅ IDs son reproducibles con mismos inputs

Ejecutar:

```
pytest tests/test_upload.py::test_generate_document_id_unique -v
```

🔵 Escenario 4: Ingesta de Datos

Test 4.1: Normalización Completa ✅

Archivo: `tests/test_ingest.py::test_normalize_doc_complete`

Propósito: Validar transformación de documentos Aconex

Entrada (Input):

```
documento_aconex = {
    "DocumentId": "200076-CCC02-PL-AR-000400",
    "project_id": "PROJ-TEST-001",
    "metadata": {
        "Title": "Plan Maestro",
        "Number": "200076-CCC02-PL-AR-000400",
        "Category": "Arquitectura",
        "DateModified": "2024-01-15T10:30:00Z"
    },
    "full_text": "Contenido técnico..."
}
```

Salida Esperada (Output):

```
documento_normalizado = {
    "document_id": "200076-CCC02-PL-AR-000400",
    "project_id": "PROJ-TEST-001",
    "title": "Plan Maestro",
    "category": "Arquitectura",
    "body_text": "Plan Maestro\n\nContenido técnico...",
    "date_modified": datetime(2024, 1, 15, 10, 30, 0)
}
```

Validaciones de Caja Negra:

- ✅ Todos los campos se extraen correctamente
- ✅ `date_modified` es tipo `datetime`, no string
- ✅ `body_text` concatena título + contenido
- ✅ Prioridad de `project_id` correcta

Ejecutar:

```
pytest tests/test_ingest.py::test_normalize_doc_complete -v
```

Test 4.2: Lectura JSON y NDJSON ✅

Archivo: `tests/test_ingest.py::test_iter_docs_from_file_json_and_ndjson`

Propósito: Validar soporte para ambos formatos

Entrada 1 (JSON Array):

```
[
    {"DocumentId": "001", "metadata": {"Title": "Doc 1"}},
    {"DocumentId": "002", "metadata": {"Title": "Doc 2"}}
]
```

Entrada 2 (NDJSON):

```
{"DocumentId": "003", "metadata": {"Title": "Doc 3"}}
{"DocumentId": "004", "metadata": {"Title": "Doc 4"}}
```

Salida Esperada:

```
# Ambos formatos retornan lista de documentos
docs = [
    {"DocumentId": "001", ...},
    {"DocumentId": "002", ...}
]
```

Validaciones de Caja Negra:

- ✅ Detección automática de formato
- ✅ Parsing correcto de ambos formatos
- ✅ Líneas vacías en NDJSON se ignoran
- ✅ Número correcto de documentos extraídos

Ejecutar:

```
pytest tests/test_ingest.py::test_iter_docs_from_file_json_and_ndjson -v
```

🔧 Troubleshooting

❌ Error: ModuleNotFoundError: No module named 'app'

Causa: pytest no encuentra el módulo app

Solución:

```
# Verificar que estás en backend-acorag/
cd backend-acorag

# Verificar estructura
ls app/

# Ejecutar desde directorio correcto
pytest tests/ -v
```

❌ Error: fixture 'mock_model_loader' not found

Causa: `conftest.py` no se está cargando

Solución:

```
# Verificar que conftest.py existe
ls tests/conftest.py

# Asegurarse de que __init__.py existe
ls tests/__init__.py

# Re-ejecutar
pytest tests/ -v
```

❌ Error: FAILED tests/test_chat.py - ImportError: cannot import name 'ChatRequest'

Causa: Módulos de `app/` no disponibles

Solución:

```
# Verificar que app/ tiene los módulos
ls app/api.py
ls app/analytics.py

# Verificar Python Path
python -c "import sys; print('\n'.join(sys.path))"

# Instalar en modo editable
pip install -e .
```

⚠️ Warning: No coverage data collected

Causa: pytest-cov no instalado o mal configurado

Solución:

```
# Instalar pytest-cov
pip install pytest-cov

# Verificar instalación
pytest --version

# Ejecutar con cobertura
pytest tests/ --cov=app --cov-report=term
```

🐌 Tests muy lentos

Causa: Ejecución secuencial

Solución:

```
# Instalar pytest-xdist
pip install pytest-xdist

# Ejecutar en paralelo (todos los cores)
pytest tests/ -n auto

# O especificar número de workers
pytest tests/ -n 4
```
