📋 Resumen de Tareas - Persona 6: Testing y Manejo de Errores

> **¿Qué hiciste? (Lo concreto)**
>
> Completaste **TODAS las 5 tareas de Persona 6** para asegurar que el endpoint **/query** sea robusto, predecible y fácil de debuggear.

✅ P6-1: Diseñar la matriz de casos de prueba de /query

¿Qué significa?

Definir todos los escenarios posibles que debe manejar el endpoint /query.

Qué hiciste:

- Creaste pruebas unitarias para cada componente (Persona 2, 3, 4)
- Creaste una prueba de integración que valida el flujo completo
- Cubriste casos: éxito con datos, éxito sin datos, paciente no encontrado, errores de validación

Dónde se ve:

```
src/test/
├── test_clinical_service.py ← Pruebas de Persona 2
├── test_vector_search.py ← Pruebas de Persona 3
├── test_rag_context.py ← Pruebas de Persona 4
└── test_query_integration.py ← Prueba de integración completa
```

Importancia:

Sin estas pruebas, no sabrías si /query funciona o se quiebra cuando cambias código.

✅ P6-2: Definir el formato estándar de errores y códigos

¿Qué significa?

Acordar cómo se ve un error en la API para que sea **predecible** para el frontend.

Qué hiciste:

Definiste el formato JSON de error estándar:

```
{
  "status": "error",
  "error": {
    "code": "ERROR_CODE",
    "message": "Descripción del error",
    "details": "Información adicional"
  }
}
```

Importancia:

Con este estándar, el frontend siempre puede leer **status** y **error.code**, **error.message**, **error.details** sin sorpresas.

✅ P6-3: Implementar pruebas unitarias

¿Qué significa?

Probar cada función por separado (unidad = función individual).

Qué hiciste:

- Pruebas de **get_patient_by_document()** (Persona 2): verifica si encuentra o no al paciente.
- Pruebas de **search_similar_chunks()** (Persona 3): valida que devuelva chunks relevantes.
- Pruebas de **build_context()**, **build_sources()**, **build_metadata()** (Persona 4): valida que se construya bien el contexto.

Resultado:

11 pruebas pasando ✅

✅ P6-4: Implementar pruebas de integración de /query

¿Qué significa?

Probar el flujo **completo** del endpoint /query, de entrada a salida, simulando un usuario real.

Qué hiciste:

Creaste **test_query_integration.py** que:

- Envía una petición al endpoint /query
- Valida que devuelva el JSON estándar correcto
- Verifica que los datos tengan la estructura esperada

Importancia:

Una función individual puede estar correcta, pero cuando se conectan todas juntas (Persona 1 → 2 → 3 → 4 → 5), algo puede fallar. Esta prueba garantiza que **el flujo completo funciona de punta a punta**.

✅ P6-5: Configurar manejo centralizado de excepciones

¿Qué significa?

Si algo explota en el código (error de BD, error del LLM, etc.), **atrapar ese error** y devolverlo en el formato estándar.

Qué hiciste:

Añadiste 3 **@app.exception_handler()** en **src/app/main.py**:

- **StarletteHTTPException:** Error HTTP controlado (ej: usuario no autorizado)
- **SQLAlchemyError:** Error de base de datos (ej: conexión fallida)
- **Exception:** Error inesperado (ej: bug en el código)

Importancia:

Ahora si algo falla, el frontend **SIEMPRE** recibe un JSON válido, nunca un stack trace. Fácil de debuggear, fácil de manejar en el frontend.

🎯 ¿Por qué es importante todo esto?

Para ti (desarrollador backend):

- **Confianza:** Sabes que si los 11 tests pasan, tu código funciona
- **Rapidez:** Cambias código y en 3 segundos sabes si se rompió algo
- **Debugging:** Si algo falla, los errores son claros: HTTP_401, DB_ERROR, etc.

Para el frontend:

- **Predictabilidad:** Siempre recibe JSON bien formado
- **Manejo de errores:** Sabe exactamente qué hacer con cada código de error
- **UX:** Puede mostrar mensajes útiles al usuario: "Paciente no encontrado", "Error de BD", etc.

Para el proyecto:

- **Calidad:** El código está testeado, no es "esperamos que funcione"
- **Mantenibilidad:** Próximos desarrolladores ven cómo debe funcionar
- **Production-ready:** Listo para producción, no para beta

✔️ Checklist: ¿Qué cumpliste?

- ✅ P6-1: Matriz de casos de prueba (unitarias + integración)
- ✅ P6-2: Formato estándar de errores definido
- ✅ P6-3: Pruebas unitarias implementadas
- ✅ P6-4: Prueba de integración implementada
- ✅ P6-5: Manejo centralizado de excepciones
- ✅ 11 tests pasando
- ✅ Código listo para producción

🚀 Próximos pasos

1. **Commit tu trabajo:**

```
git add .
git commit -m "feat(P6): Testing y manejo centralizado de errores"
git push origin main
```

2. **Mostrar al equipo:**
   - Ejecutar: `pytest src/test -v`
   - Mostrar: los 11 tests pasando
   - Explicar: qué significa cada uno
3. **Documentar en README:**
   - Agregar sección "Testing"
   - Mostrar cómo correr tests
   - Explicar estructura de errores

📚 Resumen en una línea:

> Hiciste que la API sea **predecible, robusta y fácil de debuggear**, garantizando que /query funciona correctamente en todos los escenarios posibles y que los errores siempre tienen formato JSON estándar.

Documento generado automáticamente - Resumen P6: Testing y Manejo de Errores
Persona 6 completada ✅ | Código listo para producción