He analizado los cuatro archivos que me proporcionaste (documentos\_widget.py, editar\_documento\_dialog.py, documentos\_controller.py, y documentos\_db.py) para comprender el flujo de actualización de documentos y las posibles inconsistencias que podrían estar causando problemas.

Aquí te presento un **Plan de Trabajo Ordenado y Lógico** para resolver el problema de refactorización y asegurar la correcta funcionalidad del módulo de documentos, especialmente en la edición:

**Diagnóstico General**

Se observan principalmente dos áreas de inconsistencia que podrían generar errores o un comportamiento inesperado durante la edición de documentos:

1. **Nomenclatura de Campos de Fecha:** Hay una falta de uniformidad en el nombre del campo de fecha de "subida" o "creación" del documento a través de las capas. El modelo de base de datos (Documento en DocumentosDB) usa fecha\_subida, pero la capa de la base de datos (get\_documento\_por\_id en DocumentosDB) lo retorna como fecha\_creacion en el diccionario, y el controlador (get\_document\_for\_edition) lo mapea de igual forma, mientras que editar\_documento en el controlador espera fecha\_subida\_str.
2. **Manejo de proceso\_id:** Aunque el campo proceso\_id existe en el modelo Documento y es aceptado por algunos métodos en el controlador y la base de datos, no se observa una clara gestión de este campo en la interfaz de usuario (editar\_documento\_dialog.py) para su edición. Esto puede llevar a que el valor de proceso\_id se pierda o no se actualice correctamente.

**Plan de Trabajo Detallado**

Implementaremos el plan en fases para asegurar que cada cambio se realice de manera incremental y controlada.

**Fase 1: Estandarización del Campo de Fecha (Prioridad Alta)**

El objetivo es usar un nombre único y consistente para el campo de la fecha del documento en todas las capas. Proponemos estandarizarlo a **fecha\_subida** ya que es el nombre del atributo en el modelo de base de datos Documento.

1. **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_db.py**
   * **get\_documento\_por\_id**: Modificar la clave del diccionario retornado para que sea **"fecha\_subida"** en lugar de "fecha\_creacion".

Python

# Antes

# "fecha\_creacion": documento.fecha\_subida.date() ...

# Después

"fecha\_subida": documento.fecha\_subida.date() if isinstance(documento.fecha\_subida, datetime) else documento.fecha\_subida,

* + **update\_documento**: Eliminar la línea específica que mapea fecha\_creacion a fecha\_subida. Con la estandarización, fecha\_subida debería ser el nombre de clave directamente pasado en kwargs.

Python

# Eliminar esta parte si 'fecha\_subida' ya se pasa directamente como clave

# if key == 'fecha\_creacion':

# setattr(documento, 'fecha\_subida', value)

* + **actualizar\_documento**: Asegurarse de que el parámetro que se pasa a update\_documento sea fecha\_subida y no nueva\_fecha\_creacion.

Python

# Antes

# fecha\_subida=nueva\_fecha\_creacion

# Después

fecha\_subida=nueva\_fecha\_creacion # Asegurarse de que 'nueva\_fecha\_creacion' en la lógica se entienda como 'fecha\_subida' en DB

1. **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_controller.py**
   * **get\_document\_for\_edition**: Modificar la clave del diccionario de retorno para que sea **'fecha\_subida'** en lugar de 'fecha\_creacion'.

Python

# Antes

# 'fecha\_subida': str(documento\_dict['fecha\_creacion']),

# Después

'fecha\_subida': str(documento\_dict['fecha\_subida']), # Asegurarse que el dict del DB ya tiene 'fecha\_subida'

* + **editar\_documento**: Asegurarse de que el parámetro fecha\_subida\_str se pase con el nombre correcto fecha\_subida al diccionario updated\_data. Actualmente, esto parece estar correcto, solo verificar la consistencia semántica.

1. **Refactorizar SELECTA\_SCAM/modulos/documentos/editar\_documento\_dialog.py**
   * **Poblar QDateEdit**: Cuando se cargan los datos del documento para edición, asegurarse de que el valor para poblar el QDateEdit se obtiene de la clave correcta **'fecha\_subida'** del diccionario documento\_data.

Python

# Ejemplo (ajusta a tu implementación actual de carga de datos):

# self.date\_upload\_edit.setDate(documento\_data['fecha\_subida'])

* + **Obtener datos para emisión**: Al recopilar los datos del diálogo para emitir la señal de actualización, asegurarse de que el valor de la fecha se asigne a la clave **'fecha\_subida'** en el diccionario de datos.

Python

# Ejemplo (ajusta a tu implementación actual de recopilación de datos):

# 'fecha\_subida': self.date\_upload\_edit.date().toString('yyyy-MM-dd HH:mm:ss')

**Fase 2: Manejo de proceso\_id (Decisión y Implementación)**

Necesitamos decidir si proceso\_id debe ser editable a través de este diálogo.

1. **Decisión y Diseño (Comunicar al equipo)**:
   * **Opción A (Editable)**: Si proceso\_id debe ser editable, se debe añadir un widget (ej. QLineEdit o QComboBox si es un ID de una lista de procesos) en editar\_documento\_dialog.py.
     + **Refactorizar SELECTA\_SCAM/modulos/documentos/editar\_documento\_dialog.py**:
       - Añadir el widget para proceso\_id al UI.
       - Poblar el widget con el proceso\_id actual del documento.
       - Incluir el valor del widget en la señal documento\_editado\_signal.
     + **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_controller.py**:
       - Asegurar que get\_document\_for\_edition incluye proceso\_id en el diccionario retornado.
       - Asegurar que editar\_documento reciba y pase el proceso\_id actualizado a documentos\_db.update\_documento.
     + **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_db.py**:
       - Asegurar que update\_documento maneja correctamente la actualización del proceso\_id si se envía en kwargs. (La implementación actual parece manejarlo bien).
   * **Opción B (No Editable)**: Si proceso\_id no es editable en este diálogo (se mantiene su valor original o se gestiona en otro lugar):
     + **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_controller.py**:
       - En editar\_documento, asegurarse de que el proceso\_id que se pasa al update\_documento de la DB sea el original del documento si no se desea modificar. Esto implica obtener el documento antes para obtener el proceso\_id original o asegurarse de que editar\_documento\_dialog lo devuelva inalterado. Si el diálogo no lo envía, se pasará None, lo cual sobrescribiría el valor existente.
       - Considera eliminar proceso\_id como parámetro de editar\_documento si no se va a usar para actualizarlo, o pasarlo como un parámetro opcional con su valor actual.

**Fase 3: Robustez y Manejo de Errores**

Mejorar la experiencia de usuario y la estabilidad de la aplicación.

1. **Refactorizar SELECTA\_SCAM/modulos/documentos/editar\_documento\_dialog.py**
   * **Validación de Entrada**: Implementar validaciones básicas (ej. nombre no vacío) antes de emitir la señal de edición. Deshabilitar el botón "Guardar" si los campos requeridos no son válidos.
   * **Feedback al Usuario**: Proporcionar mensajes de error o advertencia claros si la validación falla.
2. **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_controller.py**
   * **Manejo de Errores de editar\_documento**: Asegurarse de que los raise en los except bloques de editar\_documento sean manejados por la vista, o que se capturen y se devuelva un False para indicar el fallo.
   * **Registro (Logging)**: Añadir más mensajes de registro detallados en editar\_documento y get\_document\_for\_edition para facilitar la depuración, incluyendo los valores de los datos que se intentan guardar o recuperar.
3. **Refactorizar SELECTA\_SCAM/modulos/documentos/documentos\_widget.py**
   * **Manejo de Respuestas del Controlador**: En handle\_edit\_document, asegurarse de que se gestiona adecuadamente el valor de retorno de self.controller.editar\_documento. Si devuelve False o lanza una excepción, mostrar un QMessageBox.critical para informar al usuario que la edición falló.
   * **Actualización de Vista**: Asegurarse de que la vista se refresca correctamente (self.ejecutar\_busqueda()) después de una edición exitosa.

**Fase 4: Pruebas Exhaustivas**

Una vez implementados los cambios, es crucial realizar pruebas para asegurar que todo funciona como se espera y no se han introducido nuevas regresiones.

1. **Pruebas Unitarias**:
   * Para DocumentosDB.update\_documento: Probar con datos válidos, nulos, formatos de fecha incorrectos (si es posible inyectarlos).
   * Para DocumentosController.editar\_documento: Probar con diferentes escenarios de entrada, incluyendo el manejo de proceso\_id según la decisión tomada en la Fase 2.
2. **Pruebas de Integración (End-to-End)**:
   * Probar la edición de un documento desde la interfaz de usuario (DocumentosWidget -> EditarDocumentoDialog).
   * Verificar que los datos se actualizan correctamente en la base de datos y que se reflejan en la interfaz después de la edición.
   * Probar la edición con y sin el campo proceso\_id (según la Fase 2).
   * Verificar el comportamiento con entradas inválidas en el diálogo (si se implementó validación).

Este plan te proporcionará una guía clara y paso a paso para refactorizar y mejorar la funcionalidad de edición de documentos, asegurando la consistencia de los datos y una mejor experiencia de usuario.

¿Tienes alguna pregunta o quieres que profundicemos en algún punto específico de este plan?