Informe de Optimización "Green Software"

Proyecto: Servicio de Textos (textService.js) Fecha: 14 de noviembre de 2025 Asunto: Análisis comparativo de eficiencia energética y de recursos entre getAllTextsByOwner (Original) y getPaginatedTextsByOwner (Mejorado).

1. **Resumen Ejecutivo**

   Este informe detalla la optimización "verde" aplicada a la función de listado de textos del textService. La función original, getAllTextsByOwner, presentaba un diseño ineficiente que consumía una cantidad excesiva de recursos (CPU, memoria y red) al escalar.

   La nueva implementación, getPaginatedTextsByOwner, aplica dos principios clave de **Green Software**:

   - **Proyección de Datos:** Solicitar solo la información estrictamente necesaria.
   - **Paginación:** Limitar la cantidad de datos procesados en una sola solicitud.

   El resultado es una reducción drástica en el consumo de recursos, haciendo la aplicación más rápida, escalable y ecológicamente eficiente.

2. **Análisis Comparativo del Código**

   Código Original (Ineficiente)

   El endpoint de listado de textos utilizaba esta función de servicio:

```
// textService.js (Versión Original)
const getAllTextsByOwner = async (userId) => {
   try {
      return await Text.find({ owner: userId }).sort({ createdAt: -1 });
   } catch (error) {
      // ...
   }
};
```

   Problemas de Eficiencia (Anti-Patrón "Verde"):

   - Falta de Paginación (.find() sin .limit()): Si un usuario tiene 5,000 textos, esta consulta intenta cargar los 5,000 documentos de la base de datos.
   - Falta de Proyección (Sin .select()): Para cada uno de esos 5,000 documentos, carga el documento completo, incluyendo el campo content. Si cada content tiene un promedio de 100 KB, la consulta intentaría procesar y transferir 500 MB de datos solo para mostrar una lista de nombres de archivo.

   Código Mejorado (Eficiente - "Green Software")

   La función fue refactorizada para aceptar parámetros de paginación y aplicar una proyección:

```
// textService.js (Versión Mejorada)
const getPaginatedTextsByOwner = async (userId, skip, limit) => {
   try {
      // 1. Conteo eficiente para el frontend
      const total = await Text.countDocuments({ owner: userId });
```

```
      // 2. Aplicación de principios "Verdes"
      const texts = await Text.find(
        { owner: userId },            // Filtro
        'filename createdAt status', // <-- PROYECCIÓN (Software Verde)
        { skip: skip, limit: limit }   // <-- PAGINACIÓN (Software Verde)
      ).sort({ createdAt: -1 });

      return { texts, total };
    } catch (error) {
      // ...
    }
};
```

Soluciones de Eficiencia ("Principios Verdes"):

- **Paginación** (.skip(0).limit(10)): La consulta solo extrae 10 documentos de la base de datos, sin importar si el usuario tiene 10 o 10 millones. El trabajo del servidor es constante y mínimo.
- **Proyección** ('filename createdAt status'): La consulta excluye explícitamente el campo pesado content. Solo transfiere unos pocos bytes de metadatos por documento.

3. Evaluación de Impacto (Comparativa)

El siguiente cuadro compara el impacto en los recursos para un escenario de un usuario con 5,000 textos consultando la primera página (10 items).

Métrica de Recursos	Código Original (Ineficiente)	Código Mejorado (Eficiente)	Impacto "Verde"
Carga de CPU (Base de Datos)	**Extrema**. Debe leer 5,000 documentos completos del disco.	**Mínima**. Lee solo 10 documentos (o índices) y los metadatos.	**Reducción > 99%** en el consumo de CPU del servidor de BD.
Transferencia de Red (BD -> App)	**~500 MB**. (5,000 docs * 100KB c/u)	**< 2 KB**. (10 docs * ~150 bytes c/u)	**Menor uso de ancho de banda.** Reduce la energía consumida por la red.
Consumo de Memoria (App Node.js)	**Peligroso (~500 MB)**. Carga todos los datos en memoria. Riesgo de "Out of Memory" (OOM) y caída del servicio.	**Mínimo (< 2 KB)**. Solo carga los 10 items necesarios para la respuesta.	**Menor huella de memoria**. Permite que el servidor maneje más usuarios concurrentes.
Latencia (Respuesta al	**Muy Lenta** (Segundos o	**Instantánea** (Milisegundos).	**Mejora drástica de la UX**. Menos

Usuario)	Minutos). El usuario espera a que se procesen 500MB.		tiempo de procesamiento = menos energía.
Escalabilidad	**Nula**. El sistema colapsa al crecer la data del usuario.	**Alta**. El rendimiento es constante (O(k)) sin importar el total de datos (N).	El sistema es sostenible a largo plazo.

4. Conclusión

La refactorización de getAllTextsByOwner a getPaginatedTextsByOwner es una optimización crítica de "Green Software". Al implementar paginación y proyección, transformamos una operación peligrosa e ineficiente (O(N)) en una operación ligera y constante (O(k)).

Este cambio reduce drásticamente el consumo de CPU, memoria y red, lo que se traduce directamente en un menor costo energético (y monetario) del servidor y una aplicación más rápida y estable para el usuario.