# Evaluación de la librería `better-profanity` y Flujo de Moderación Backend/Frontend

Equipo INTEGRA2

6 de octubre de 2025

## 1. Objetivo

Evaluar la viabilidad y efectividad de la librería `better-profanity` en un flujo de moderación de texto, considerando su integración en backend y frontend. Se realizará una prueba de concepto (POC) local utilizando un archivo de lista de palabras prohibidas (`profanity-list.txt`) personalizado.

## 2. Descripción

Se busca implementar y analizar el filtro de lenguaje ofensivo en una aplicación local, utilizando la librería `better-profanity`. El sistema debe leer un archivo externo con palabras prohibidas, y filtrar el texto ingresado por el usuario antes de procesarlo o almacenarlo. Además, se evalúa cómo este filtro puede integrarse en el flujo de moderación tanto en el backend (procesamiento y validación) como en el frontend (retroalimentación al usuario).

## 3. Prueba de Concepto (POC) Local

Supongamos que el archivo `profanity-list.txt` contiene una palabra prohibida por línea.

Listing 1: POC de uso con lista personalizada

```
from better_profanity import Profanity

# Leer el listado de palabras prohibidas
with open('profanity-list.txt', 'r', encoding='utf-8') as f:
    custom_words = [line.strip() for line in f if line.strip()]
```

```
profanity = Profanity()
profanity.load_censor_words(custom_words)

texto = "Este texto contiene palabras feas y groseras."
if profanity.contains_profanity(texto):
    print("Texto no permitido:", profanity.censor(texto))
else:
    print("Texto permitido:", texto)
```

**Pasos para probar:**

1. Instalar la librería: `pip install better-profanity`
2. Crear el archivo `profanity-list.txt` con palabras prohibidas.
3. Ejecutar el código anterior.

# 4. Pros y Contras Extendidos y Justificados

## Pros

- **Simplicidad de integración y uso:** La librería se instala fácilmente vía `pip` y su API es directa. Esto reduce la curva de aprendizaje y agiliza la incorporación en proyectos existentes, lo cual es clave para equipos con recursos limitados o plazos ajustados.
- **Personalización del diccionario:** Permite cargar listas propias de palabras prohibidas (`profanity-list.txt`), lo cual es esencial para adaptarse a los requerimientos lingüísticos y culturales específicos de cada aplicación o comunidad.
- **Código abierto y mantenido:** Al ser open source y estar activamente mantenida, facilita auditoría, colaboración y adaptación según las necesidades del proyecto. Esto brinda transparencia y confianza respecto al funcionamiento del filtro.
- **Buen rendimiento para textos cortos y medianos:** La librería está optimizada para validar textos en tiempo real, lo cual es fundamental en entornos web donde la experiencia de usuario depende de la rapidez de respuesta.

- **Opciones de censura flexibles:** Permite elegir entre censura total o parcial de palabras detectadas, pudiendo adaptar el feedback al usuario según el contexto (advertencia, bloqueo, sustitución, etc.).

### Contras

- **Limitación a palabras individuales:** El filtro funciona solo con palabras exactas, sin capacidad para detectar frases ofensivas o insultos compuestos. Esto puede dejar pasar expresiones ofensivas que no estén en el diccionario como una sola palabra.
- **Falta de soporte multilenguaje integrado:** Por defecto, el filtro está orientado al inglés, aunque se puede adaptar con una lista en español. Sin embargo, no incluye soporte avanzado para variantes idiomáticas, conjugaciones o regionalismos, lo que puede limitar su efectividad en comunidades diversas.
- **No detecta variantes creativas o alteraciones:** Usuarios malintencionados pueden eludir el filtro usando variantes como "p@labr4" o agregando espacios/símbolos. Esto limita la capacidad de moderación, requiriendo complementar con algoritmos de fuzzy matching o regex.
- **Ausencia de contexto semántico:** El filtro no analiza el contexto del mensaje, por lo que puede censurar palabras válidas en ciertos usos (ejemplo: términos médicos, educativos). Esto puede generar falsas alarmas y afectar la experiencia de usuario.
- **No es adecuado para flujos críticos de seguridad:** Para aplicaciones donde la moderación debe ser exhaustiva (por ejemplo, plataformas públicas de alto tráfico), puede ser insuficiente y requerir soluciones más avanzadas basadas en inteligencia artificial o moderación manual.

## 5. Justificación

La elección de `better-profanity` como primer filtro en el flujo de moderación se justifica por su rapidez de implementación, facilidad de personalización y bajo coste operativo. En el contexto de una POC o sistemas donde el riesgo reputacional es bajo, su uso es adecuado y eficiente. Sin embargo, para sistemas en producción, especialmente de acceso público, se recomienda complementar este filtro con técnicas adicionales y revisión manual, debido a las limitaciones descritas.

# 6. Flujo de Moderación Backend/Frontend

**Backend:**

- El texto se envía al servidor vía API.
- El backend aplica el filtro usando `better-profanity` y el listado personalizado.
- Si se detectan palabras prohibidas, se rechaza el mensaje y se informa al frontend.
- Si no se detectan, el texto se procesa normalmente (almacena, muestra, etc.).

**Frontend:**

- (Opcional) Validación preliminar con JavaScript para feedback rápido.
- Mostrar advertencia si el texto es bloqueado por el backend.
- Permitir al usuario editar y reenviar el texto.
- Siempre confiar en el filtrado del backend para seguridad.