Tarea: Avance Unidad II Trabajo final de asignatura TFA

Se pide presentar su avance del Trabajo final de Asignatura correspondiente a la Unidad 2.

Se debe presentar:

1. **Un documento o informe con el avance del TFA respecto a la Unidad 2**
2. **El código fuente Unidad II respectivo.**

Se debe respetar la fecha límite de presentación del avance.

Estudiante: Jaime David Mendoza Orosco

Arquitectura General

1. Frontend (React)

- **Componentes principales:**
    - **Home Page:**
        - Campo para ingresar el enlace a la publicación.
        - Botón para iniciar el análisis.
        - Indicador de progreso del análisis.
    - **Resultados:**
        - Tabla o lista de comentarios analizados, con indicadores (ofensivo/no ofensivo).
        - Opciones para filtrar y gestionar los comentarios marcados como ofensivos.
    - **Configuración:**
        - Configuración de patrones ofensivos adicionales (personalización).
        - Opciones de idioma o sensibilidad.
    - **Versión móvil:**
        - Interfaz responsiva que replica las funcionalidades de la versión web.
- **Librerías recomendadas:**
    - Axios: para interactuar con el backend.
    - React Router: para la navegación.
    - Material-UI o Tailwind CSS: para estilos.
    - Redux o Context API: para gestionar el estado.

---

2. Backend (Node.js + Express)

- **Endpoints principales:**
    - **GET /patterns:**
        - Devuelve patrones de lenguaje ofensivo almacenados.
    - **POST /analyze:**
        - Recibe un enlace de la publicación.
        - Extrae los comentarios usando una API de terceros o scraping (verificar permisos y límites legales de uso).
        - Devuelve un análisis detallado de cada comentario.
    - **POST /patterns:**
        - Permite añadir nuevos patrones ofensivos.
    - **GET /results:**
        - Devuelve resultados previos de análisis (si es necesario).
- **Procesamiento:**
    - Implementación de autómatas:
        - Crear un módulo que implemente DFA (Autómatas Deterministas Finitos) para buscar palabras ofensivas.
        - Alternativa: utilizar RegEx para patrones más complejos.
    - **Pipeline de análisis:**
        - Scraping o acceso a API -> Preprocesamiento del texto -> Análisis con autómatas -> Marcado de comentarios.
- **Tecnologías recomendadas:**
    - Node.js + Express para el servidor.
    - Puppeteer o Cheerio para scraping (si las APIs no son una opción).
    - MongoDB para almacenar patrones y resultados.
    - JWT para autenticar usuarios (administradores de contenido).

---

3. Base de Datos (MongoDB o PostgreSQL)

- **Tablas/colecciones principales:**
    - **Comentarios analizados:**
        - `id_comentario`, `contenido`, `ofensivo (bool)`, `usuario`, `timestamp`.
    - **Patrones ofensivos:**
        - `id_patron`, `palabra`, `descripcion`, `fecha_creacion`.
    - **Usuarios:**
        - `id_usuario`, `nombre`, `email`, `password (hashed)`, `rol`.

---

4. Análisis de Comentarios

- **Pipeline detallado:**
    1. **Scraping/API:**

        - Extraer los comentarios usando herramientas específicas para plataformas como Facebook o Instagram.
    2. **Preprocesamiento:**
        - Limpiar datos: eliminar emojis, normalizar texto.
        - Tokenización: dividir el texto en palabras para facilitar el análisis.
    3. **Análisis léxico:**
        - Utilizar un DFA para detectar patrones ofensivos predefinidos.
        - Generar un reporte con comentarios marcados como ofensivos.

---

5. Módulo de Integración de APIs (Scraping)

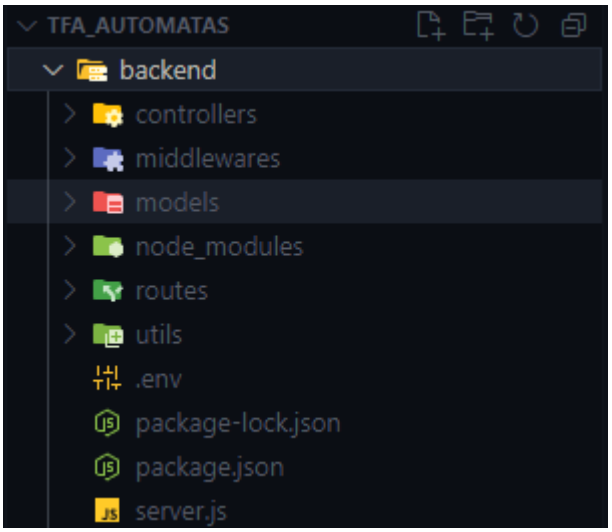
- Facebook Graph API: para comentarios de publicaciones (requiere permisos).
- Instagram Graph API: para datos de publicaciones y comentarios.
- En caso de no contar con permisos API:
    - Utilizar Puppeteer o Selenium para scraping automatizado.
    - **Nota:** Verificar los términos de uso de estas plataformas para evitar bloqueos.

---

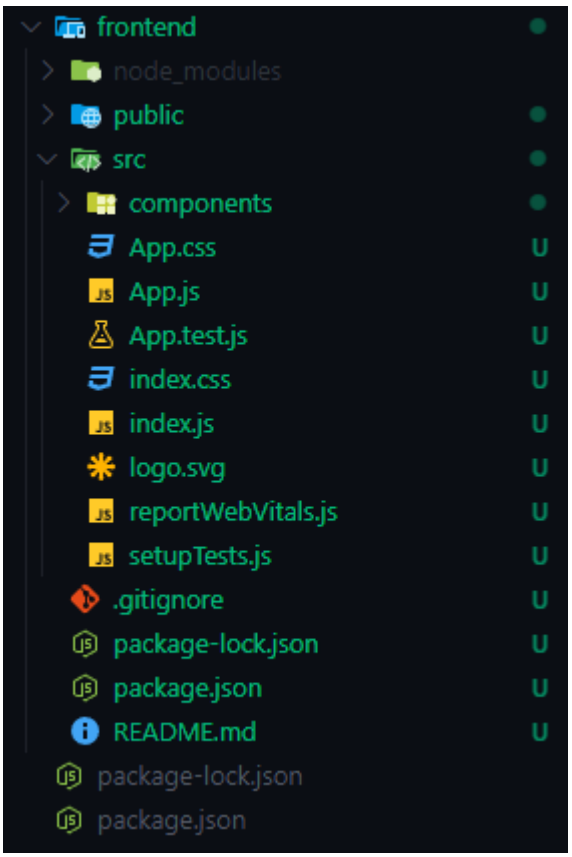
6. Seguridad

- Encriptación de contraseñas y tokens con bcrypt y JWT.
- Sanitización de entradas para evitar ataques de inyección SQL o XSS.
- Configurar CORS para proteger el backend.
- Implementar HTTPS.

ESTRUCTURA DEL BACKEND:



ESTRUCTURA DEL FRONTEND:



Conclusiones:

1. El desarrollo del analizador léxico demostró la eficiencia de combinar la teoría de autómatas con tecnologías modernas como Node.js y React, logrando un sistema funcional capaz de identificar comentarios ofensivos en tiempo real.

2. La implementación de una arquitectura modular y escalable permitió integrar fácilmente tanto la interfaz web como el backend, garantizando una experiencia de usuario intuitiva y resultados precisos en la detección de lenguaje ofensivo.