Universidad Politecnica de Chiapas

Ingeniería en Software

Analizador Lexico

Programacion Concurrente

David Antonio Gómez González
Luis Alberto Batalla González

Matrículas
221197
221189

Cuatrimestre: 7

Grupo: A

Tuxtla Gutierrez, Chiapas, México

Informe Técnico: Analizador Léxico de HTML

Equipo de Desarrollo

March 15, 2025

1 Introducción

Este informe describe la arquitectura, diseño y estrategias de implementación de un sistema de análisis léxico para archivos HTML. El sistema consta de tres componentes principales: un **backend** en Python (Flask), un **analizador léxico** y un **frontend** en React. El objetivo del sistema es permitir a los usuarios cargar archivos HTML, analizar su estructura léxica, validar su sintaxis y atributos, y visualizar los resultados de manera clara y organizada.

2 Arquitectura del Sistema

La arquitectura del sistema se divide en tres capas principales:

2.1 Frontend (React)

El frontend es una aplicación web desarrollada en React que permite a los usuarios cargar archivos HTML y visualizar los resultados del análisis léxico. Se comunica con el backend mediante solicitudes HTTP.

2.2 Backend (Flask)

El backend es un servidor desarrollado en Flask que recibe los archivos HTML enviados desde el frontend, los procesa utilizando el analizador léxico, y devuelve los resultados en formato JSON.

2.3 Analizador Léxico

El analizador léxico es un módulo en Python que procesa el contenido de los archivos HTML, identifica tokens (etiquetas, atributos, texto, etc.), y valida la estructura y atributos del HTML.

3 Diseño del Autómata

El autómata del analizador léxico está diseñado para reconocer los tokens definidos en el lenguaje HTML. Utiliza expresiones regulares para identificar patrones en el texto y clasificar los tokens.

3.1 Diagrama de Estados del Autómata

El autómata tiene los siguientes estados principales:

- **Inicio**: Estado inicial del autómata.
- **Etiqueta de Apertura**: Reconocimiento de etiquetas de apertura (e.g., ‘¡div¿‘).
- **Etiqueta de Cierre**: Reconocimiento de etiquetas de cierre (e.g., ‘¡/div¿‘).
- **Atributo**: Reconocimiento de atributos (e.g., ‘href=”...”‘).
- **Texto**: Reconocimiento de texto entre etiquetas.
- **Comentario**: Reconocimiento de comentarios (e.g., ‘¡!– comentario –¿‘).
- **Espacio en Blanco**: Reconocimiento de espacios en blanco.
- **Error**: Estado de error cuando se encuentra un carácter no reconocido.

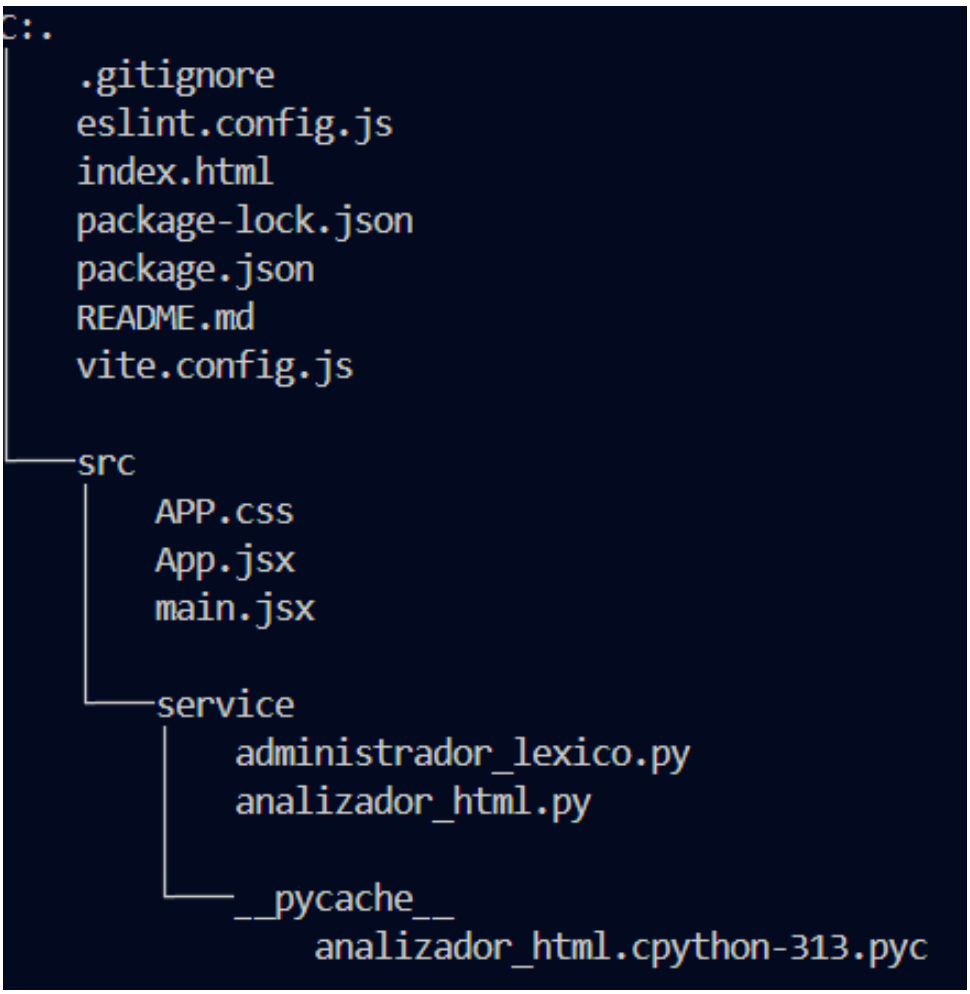

Figure 1: Diagrama de la arquitectura del sistema.

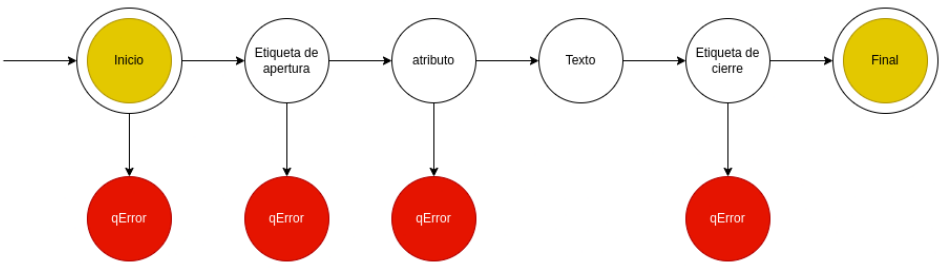

Figure 2: Diagrama de estados del autómata.

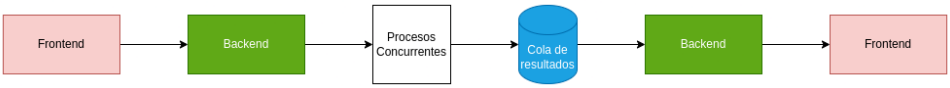

Figure 3: Diagrama de flujo del procesamiento concurrente.

4 Estrategia de Concurrencia

Para mejorar la eficiencia del sistema, se implementó una estrategia de concurrencia en el backend utilizando el módulo `multiprocessing` de Python. Esta estrategia permite procesar múltiples archivos HTML de manera simultánea, aprovechando los recursos del sistema.

4.1 Procesamiento Concurrente

- **Cola de Resultados**: Se utiliza una cola (`Queue`) para almacenar los resultados de cada proceso.
- **Procesos Independientes**: Cada archivo se procesa en un proceso independiente, lo que permite ejecutar múltiples tareas en paralelo.
- **Sincronización**: El backend espera a que todos los procesos terminen antes de recoger los resultados y devolverlos al frontend.

5 Decisiones de Diseño

- **Expresiones Regulares**: Se utilizaron expresiones regulares para la identificación de tokens debido a su eficiencia y facilidad de implementación.
- **Validación de Atributos**: Se definió un diccionario de atributos permitidos para cada etiqueta, lo que permite una validación precisa de los atributos.
- **Interfaz de Usuario**: Se eligió React para el frontend debido a su capacidad para crear interfaces dinámicas y reactivas.
- **Concurrencia**: Se implementó concurrencia en el backend para mejorar el rendimiento al procesar múltiples archivos.

6 Conclusión

El sistema desarrollado permite a los usuarios cargar archivos HTML, analizar su estructura léxica, validar su sintaxis y atributos, y visualizar los resultados de manera clara y organizada. La arquitectura del sistema, el diseño del autómata y la estrategia de concurrencia implementada garantizan un procesamiento eficiente y una experiencia de usuario satisfactoria.