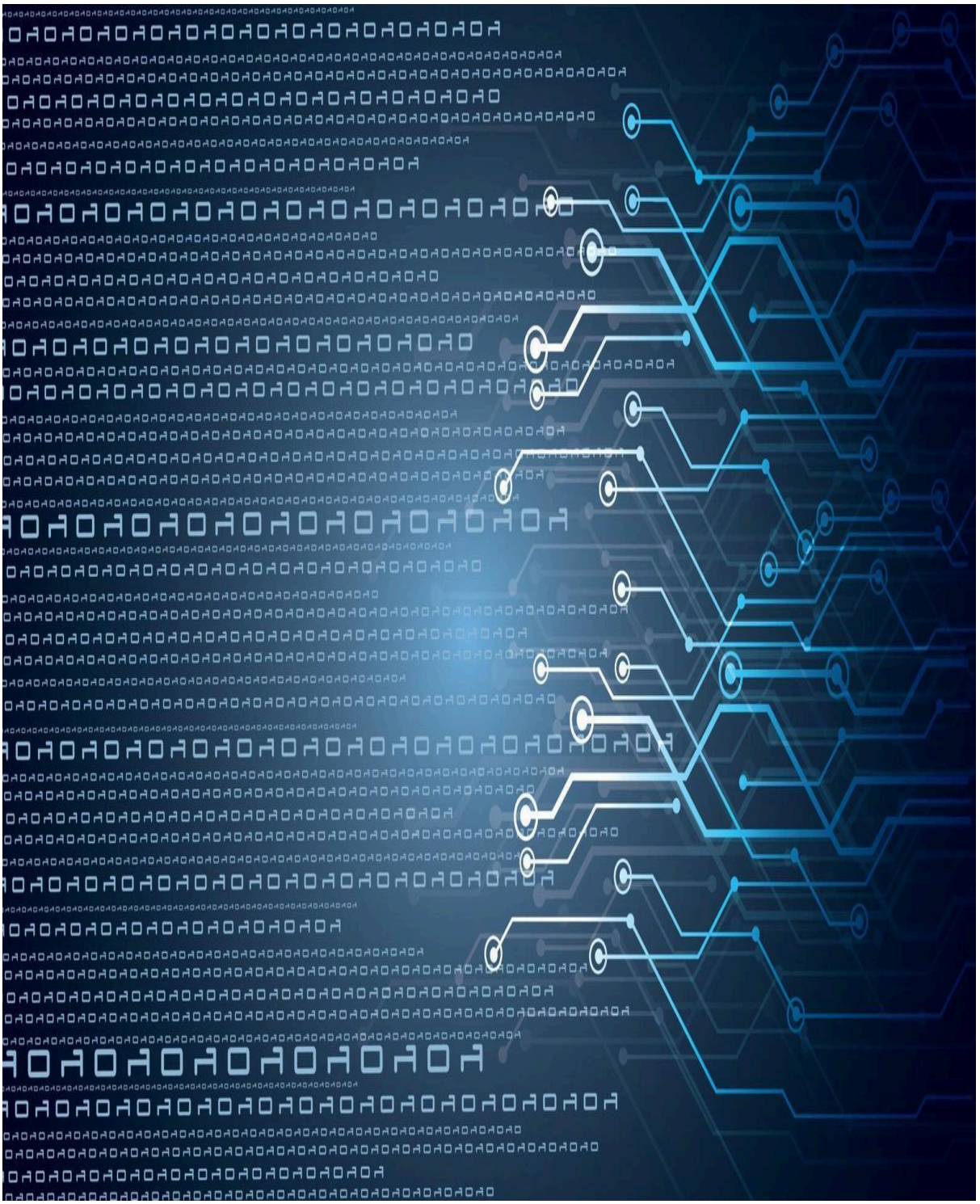
Manual Tecnico
Analizador Léxico



Frontend

Index.html

1. Descripción General

- **Propósito**: `index.html` es el punto de entrada principal del analizador léxico. Proporciona la interfaz de usuario para cargar, editar, analizar y guardar texto, así como para visualizar reportes y realizar búsquedas.
- **Tecnologías**: HTML, CSS, JavaScript (a través de archivos externos).
- **Funcionalidad**:
    - Carga de archivos de texto.
    - Edición de texto en un textarea.
    - Visualización de resultados del análisis léxico.
    - Generación y visualización de reportes (tokens, errores, recuento).
    - Búsqueda de patrones en el texto.
    - Guardar el texto editado en un archivo.

2. Estructura y Componentes

- **HTML**:
    - Contenedor principal (`div class="container"`) para organizar los elementos de la interfaz.
    - Botones para cargar archivos, analizar texto, guardar texto y ver reportes.
    - Dos áreas de texto (`textarea`): una para la entrada del usuario y otra para la salida del análisis.
    - Selectores para elegir el tipo de reporte a visualizar.
    - Campos de entrada y botones para la búsqueda de patrones.
    - Inputs ocultos para la funcionalidad de seleccionar y guardar archivos.
    - Divs para mostrar resultados de análisis y búsqueda.
- **CSS**:
    - Estilos para la apariencia general de la página (fuentes, colores, márgenes, etc.).
    - Diseño responsivo para adaptarse a diferentes tamaños de pantalla.
    - Estilos específicos para los componentes de la interfaz (botones, áreas de texto, etc.).
- **JavaScript**:
    - Se incluyen varios archivos JavaScript externos (`selectorArchivo.js`, `Token.js`, `ErrorLexico.js`, `AnalizadorLexico.js`, `poblarTabla.js`, `Analizadores.js`, `Busqueda.js`, `GuardarArchivo.js`, `index.js`) que manejan la lógica de la aplicación.
    - Manejo de eventos para los botones y campos de entrada.
    - Llamadas a las funciones de los archivos JavaScript externos para realizar el análisis léxico, la búsqueda y la generación de reportes.
    - Manejo de la funcionalidad de guardar archivos.

3. Funcionalidad Detallada

- **Carga de Archivos**:
  - El botón "Seleccionar Archivo" activa un input de tipo `file` oculto.
  - El usuario selecciona un archivo de texto, y su contenido se carga en el `textarea` de entrada.
- **Edición de Texto**:
  - El usuario puede editar el texto directamente en el `textarea` de entrada.
- **Análisis Léxico**:
  - El botón "Analizar Texto" activa la función `analizarTexto()` en `index.js`.
  - Esta función llama a las funciones del `AnalizadorLexico.js` para realizar el análisis y mostrar los resultados en el `div` de salida.
- **Visualización de Reportes**:
  - El selector `select id="seleccionReporte"` permite elegir el tipo de reporte (tokens, errores, recuento).
  - El botón "Ver Reporte" abre el archivo HTML correspondiente en una nueva ventana.
- **Búsqueda de Patrones**:
  - El campo de entrada `input id="buscar"` permite ingresar una palabra o patrón a buscar.
  - El botón "Buscar" activa la función `buscar()` en `Busqueda.js`, que resalta las coincidencias en el `textarea` de entrada y muestra los resultados en el `div id="resultadosBusqueda"`.
- **Guardar Texto**:
  - El botón "Guardar Texto" activa la funcionalidad de guardar el texto del textarea, llamando a la función `abrirGuardarComo` del archivo `GuardarArchivo.js`
- **Estilos**:
  - Se utiliza CSS para dar un aspecto visual agradable y organizado a la interfaz.

ReporteToken.html

1. Descripción General

- **Propósito**: `ReporteToken.html` muestra una tabla con los tokens detectados por el analizador léxico, junto con sus lexemas, columnas y filas correspondientes.
- **Tecnologías**: HTML, CSS, JavaScript (a través de `poblarTabla.js`).
- **Funcionalidad**:
  - Generación dinámica de una tabla HTML con información de los tokens.
  - Visualización de los tokens, lexemas, columnas y filas en una tabla.

2. Estructura y Componentes

- **HTML**:
    - Contenedor principal (`div class="container"`) para organizar los elementos de la página.
    - Título de la página ("Reporte de Tokens").
    - Tabla HTML (`table id="tablaTokens"`) con encabezados para Token, Lexema, Columna y Fila.
    - El cuerpo de la tabla (`tbody`) se llena dinámicamente con datos de tokens.
- **CSS**:
    - Estilos para la apariencia general de la página (fuentes, colores, márgenes, etc.).
    - Estilos para la tabla, incluyendo bordes, relleno y colores de fondo.
    - Estilos para los encabezados y celdas de la tabla.
- **JavaScript**:
    - Se utiliza el archivo externo `poblarTabla.js` para generar dinámicamente las filas de la tabla con los datos de los tokens.
    - La función `cargarTokens()` se llama al cargar la página (`onload="cargarTokens()"`) para iniciar la generación de la tabla.

3. Funcionalidad Detallada

- **Carga de Datos**:
    - La función `cargarTokens()` en `poblarTabla.js` recupera los datos de los tokens generados por el analizador léxico.
    - Los datos de los tokens se utilizan para crear dinámicamente filas (`<tr>`) y celdas (`<td>`) en la tabla HTML.
- **Visualización de la Tabla**:
    - La tabla muestra los tokens, lexemas, columnas y filas correspondientes en un formato tabular organizado.
    - Los estilos CSS mejoran la legibilidad y apariencia de la tabla.

ReporteErrores.html

1. Descripción General

- **Propósito**: `ReporteErrores.html` muestra una tabla con los errores léxicos detectados por el analizador, junto con la cadena errónea, la columna y la fila donde se encontraron.
- **Tecnologías**: HTML, CSS, JavaScript (a través de `poblarTabla.js`).
- **Funcionalidad**:
    - Generación dinámica de una tabla HTML con información de los errores léxicos.
    - Visualización de la cadena errónea, la columna y la fila en una tabla.

2. Estructura y Componentes

- **HTML**:
    - Contenedor principal (`div class="container"`) para organizar los elementos de la página.
    - Título de la página ("Reporte de Errores").
    - Tabla HTML (`table id="tablaErrores"`) con encabezados para Cadena, Columna y Fila.
    - El cuerpo de la tabla (`tbody`) se llena dinámicamente con datos de errores.
- **CSS**:
    - Estilos para la apariencia general de la página (fuentes, colores, márgenes, etc.).
    - Estilos para la tabla, incluyendo bordes, relleno y colores de fondo.
    - Estilos específicos para los encabezados y celdas de la tabla, con énfasis en el color rojo para indicar errores.
- **JavaScript**:
    - Se utiliza el archivo externo `poblarTabla.js` para generar dinámicamente las filas de la tabla con los datos de los errores.
    - La función `cargarErrores()` se llama al cargar la página (`onload="cargarErrores()"`) para iniciar la generación de la tabla.

3. Funcionalidad Detallada

- **Carga de Datos**:
    - La función `cargarErrores()` en `poblarTabla.js` recupera los datos de los errores léxicos generados por el analizador.
    - Los datos de los errores se utilizan para crear dinámicamente filas (`<tr>`) y celdas (`<td>`) en la tabla HTML.
- **Visualización de la Tabla**:
    - La tabla muestra la cadena errónea, la columna y la fila correspondientes en un formato tabular organizado.
    - Los estilos CSS mejoran la legibilidad y apariencia de la tabla, destacando los errores con el color rojo.

RecuentoTokens.html

1. Descripción General

- **Propósito**: `RecuentoTokens.html` muestra una tabla con el recuento de cada lexema detectado por el analizador léxico.
- **Tecnologías**: HTML, CSS, JavaScript (a través de `RecuentoTokens.js`).
- **Funcionalidad**:
    - Generación dinámica de una tabla HTML con el recuento de lexemas.
    - Visualización de cada lexema y su cantidad correspondiente en una tabla.

2. Estructura y Componentes

- **HTML**:
  - Contenedor principal (`div class="container"`) para organizar los elementos de la página.
  - Título de la página ("Recuento de Tokens").
  - Tabla HTML (`table id="tablaRecuentoLexemas"`) con encabezados para Lexema y Cantidad.
  - El cuerpo de la tabla (`tbody`) se llena dinámicamente con datos del recuento de lexemas.
- **CSS**:
  - Estilos para la apariencia general de la página (fuentes, colores, márgenes, etc.).
  - Estilos para la tabla, incluyendo bordes, relleno y colores de fondo.
  - Estilos específicos para los encabezados y celdas de la tabla, con énfasis en el color verde para indicar recuento.
- **JavaScript**:
  - Se utiliza el archivo externo `RecuentoTokens.js` para generar dinámicamente las filas de la tabla con los datos del recuento de lexemas.
  - La generación de la tabla se realiza al cargar la página.

3. Funcionalidad Detallada

- **Carga de Datos**:
  - El código JavaScript en `RecuentoTokens.js` recupera los datos del recuento de lexemas generados por el analizador léxico.
  - Los datos del recuento se utilizan para crear dinámicamente filas (`<tr>`) y celdas (`<td>`) en la tabla HTML.
- **Visualización de la Tabla**:
  - La tabla muestra cada lexema y su cantidad correspondiente en un formato tabular organizado.
  - Los estilos CSS mejoran la legibilidad y apariencia de la tabla, destacando el recuento con el color verde.

Backend

Token.js

1. Descripción General

- **Propósito**: `Token.js` define la clase `Token` que se utiliza para representar los tokens identificados por el analizador léxico. Cada instancia de `Token` almacena información sobre el tipo, valor y posición del token en el texto de entrada.
- **Funcionalidad**:
    - Definición de la clase `Token` con atributos para tipo, valor y posición.
    - Constructor para inicializar las instancias de `Token`.

2. Estructura y Componentes

- **JavaScript**:
    - Clase `Token` con atributos:
        - `tipo`: Tipo del token (identificador, número, operador, etc.).
        - `valor`: Valor del token (el lexema correspondiente).
        - `posición`: Objeto con la columna y fila donde se encontró el token.
    - Constructor `constructor(tipo, valor, columna, fila)` para crear instancias de `Token`.

3. Funcionalidad Detallada

- **Clase Token**:
    - La clase `Token` se utiliza para crear objetos que representan los tokens identificados por el analizador léxico.
    - Cada objeto `Token` almacena el tipo, valor y posición del token en el texto de entrada.
- **Constructor**:
    - El constructor `constructor(tipo, valor, columna, fila)` inicializa los atributos de la instancia de `Token` con los valores proporcionados.
    - El atributo `posición` es un objeto que almacena la columna y fila donde se encontró el token.

ErrorLexico.js

1. Descripción General

- **Propósito**: `ErrorLexico.js` define la clase `ErrorLexico` que se utiliza para representar los errores léxicos identificados por el analizador. Cada instancia de `ErrorLexico` almacena información sobre el símbolo erróneo y su posición en el texto de entrada.
- **Funcionalidad**:
    - Definición de la clase `ErrorLexico` con atributos para el símbolo erróneo y su posición.
    - Constructor para inicializar las instancias de `ErrorLexico`.

2. Estructura y Componentes

- **JavaScript**:
    - Clase `ErrorLexico` con atributos:
        - `símbolo`: Símbolo erróneo encontrado en el texto de entrada.
        - `posición`: Objeto con la columna y fila donde se encontró el error.
    - Constructor `constructor(simbolo, columna, fila)` para crear instancias de `ErrorLexico`.

3. Funcionalidad Detallada

- **Clase ErrorLexico**:
    - La clase `ErrorLexico` se utiliza para crear objetos que representan los errores léxicos identificados por el analizador.
    - Cada objeto `ErrorLexico` almacena el símbolo erróneo y su posición en el texto de entrada.
- **Constructor**:
    - El constructor `constructor(simbolo, columna, fila)` inicializa los atributos de la instancia de `ErrorLexico` con los valores proporcionados.
    - El atributo `posición` es un objeto que almacena la columna y fila donde se encontró el error.

Analizadores.js

1. Descripción General

- **Propósito**: `Analizadores.js` define la clase `Analizadores` con métodos estáticos para identificar diferentes tipos de caracteres en un texto. Estos métodos se utilizan para clasificar los caracteres como letras, dígitos, operadores, signos de puntuación, delimitadores y espacios en blanco, facilitando el análisis léxico.
- **Funcionalidad**:
    - Métodos estáticos para verificar si un carácter es una letra, dígito, operador, signo de puntuación, delimitador, salto de línea o espacio en blanco.
    - Utilización de `switch` y arrays para la identificación de caracteres.

2. Estructura y Componentes

- **JavaScript**:
    - Clase `Analizadores` con métodos estáticos:
        - `esLetra(char)`: Verifica si un carácter es una letra (mayúscula o minúscula).
        - `esDigito(char)`: Verifica si un carácter es un dígito (0-9).
        - `esOperador(char)`: Verifica si un carácter es un operador.
        - `esPuntuacion(char)`: Verifica si un carácter es un signo de puntuación.
        - `esAgrupacion(char)`: Verifica si un carácter es un delimitador.
        - `esSaltoDeLinea(char)`: Verifica si un carácter es un salto de línea.
        - `esEspacio(char)`: Verifica si un carácter es un espacio en blanco.

3. Funcionalidad Detallada

- **Métodos de Verificación**:
    - Cada método estático recibe un carácter como entrada y devuelve `true` si el carácter pertenece a la categoría correspondiente, o `false` en caso contrario.
    - Los métodos `esLetra` y `esDigito` utilizan una sentencia `switch` para comparar el carácter con una lista de caracteres válidos.
    - Los métodos `esOperador`, `esPuntuacion` y `esAgrupacion` utilizan arrays para almacenar los caracteres válidos y realizan una búsqueda lineal para verificar la pertenencia.
    - Los métodos `esSaltoDeLinea` y `esEspacio` realizan una comparación directa con los caracteres `\n` y respectivamente.

AnalizadorLexico.js

1. Descripción General

- **Propósito**: `AnalizadorLexico.js` define la clase `AnalizadorLexico` que realiza el análisis léxico de un texto dado. Identifica y clasifica tokens (identificadores, números, operadores, etc.) y detecta errores léxicos.
- **Funcionalidad**:
  - Análisis léxico de un texto.
  - Identificación de tokens y errores léxicos.
  - Almacenamiento de tokens y errores en arrays.
  - Mantenimiento de la posición, columna y fila durante el análisis.

2. Estructura y Componentes

- **JavaScript**:
  - Clase `AnalizadorLexico` con atributos:
    - `texto`: Texto a analizar.
    - `tokens`: Array de tokens identificados.
    - `errores`: Array de errores léxicos detectados.
    - `posición`: Posición actual en el texto.
    - `columna`: Columna actual.
    - `fila`: Fila actual.
    - `enError`: Bandera para indicar si se ha encontrado un error.
  - Métodos:
    - `constructor(texto)`: Inicializa la instancia con el texto a analizar.
    - `analizar()`: Realiza el análisis léxico del texto.
    - `analizarIdentificador()`: Analiza un identificador.
    - `analizarNumeroODecimal()`: Analiza un número o decimal.
    - `analizarOperador(char)`: Analiza un operador.
    - `analizarError()`: Analiza un error léxico.
    - `agregarToken(tipo, valor, columna, fila)`: Agrega un token al array de tokens.
    - `imprimirTokens()`: Imprime los tokens identificados.
    - `imprimirErrores()`: Imprime los errores léxicos detectados.

3. Funcionalidad Detallada

- **Análisis Léxico**:
  - El método `analizar()` recorre el texto carácter por carácter.
  - Utiliza los métodos de `Analizadores` para clasificar los caracteres.
  - Llama a los métodos de análisis específicos (identificador, número, operador, error) según el tipo de carácter.
  - Mantiene la posición, columna y fila durante el análisis.
- **Identificación de Tokens y Errores**:

    - Los métodos de análisis específicos identifican y crean instancias de `Token` y `ErrorLexico`.
    - Los tokens y errores se almacenan en los arrays `tokens` y `errores`.
- **Manejo de Errores**:
    - La bandera `enError` se utiliza para indicar si se ha encontrado un error.
    - El método `analizarError()` captura y almacena los errores léxicos.

Index.js

1. Descripción General

- **Propósito**: `index.js` maneja la lógica principal de la interfaz de usuario del analizador léxico. Incluye funciones para analizar texto, mostrar reportes, buscar patrones, guardar texto y ajustar la altura del textarea.
- **Funcionalidad**:
    - Análisis léxico del texto ingresado por el usuario.
    - Generación y visualización de reportes de tokens y errores.
    - Búsqueda de patrones en el texto.
    - Guardado del texto editado en un archivo.
    - Ajuste dinámico de la altura del textarea.

2. Estructura y Componentes

- **JavaScript**:
    - Manejo de eventos `DOMContentLoaded` para asegurar que el código se ejecute después de que el DOM esté completamente cargado.
    - Funciones globales para interactuar con la interfaz de usuario:
        - `abrirGuardarComo()`: Abre el diálogo para guardar el texto editado.
        - `analizarTexto()`: Realiza el análisis léxico del texto ingresado.
        - `verReporte()`: Muestra el reporte seleccionado (tokens, errores, recuento).
        - `buscar()`: Busca patrones en el texto y muestra los resultados.
        - `copiarTexto()`: Copia el texto del textarea al área de salida.
        - `ajustarTextarea()`: Ajusta la altura del textarea dinámicamente.

3. Funcionalidad Detallada

- **Análisis Léxico**:
    - La función `analizarTexto()` obtiene el texto del textarea, crea una instancia de `AnalizadorLexico`, realiza el análisis y almacena los tokens y errores en `sessionStorage`.
- **Visualización de Reportes**:

    - La función `verReporte()` obtiene la selección del usuario y abre el archivo HTML correspondiente para mostrar el reporte.
- **Búsqueda de Patrones**:
    - La función `buscar()` utiliza la clase `Búsqueda` para buscar patrones en el texto y muestra los resultados en el área de salida.
- **Guardado de Texto**:
    - La función `abrirGuardarComo()` utiliza la clase `GuardarTexto` para guardar el texto editado en un archivo.
- **Ajuste del Textarea**:
    - La función `ajustarTextarea()` ajusta la altura del textarea dinámicamente para adaptarse al contenido.

PoblarTabla.js

1. Descripción General

- **Propósito**: `PoblarTabla.js` contiene funciones para generar dinámicamente tablas HTML con los datos de tokens y errores léxicos almacenados en `sessionStorage`.
- **Funcionalidad**:
    - Generación de tablas HTML para mostrar tokens y errores léxicos.
    - Recuperación de datos de `sessionStorage`.
    - Limpieza de tablas antes de agregar nuevos datos.

2. Estructura y Componentes

- **JavaScript**:
    - Funciones para poblar tablas:
        - `poblarTablaTokens(tokens)`: Genera una tabla con los datos de los tokens.
        - `poblarTablaErrores(errores)`: Genera una tabla con los datos de los errores léxicos.
    - Funciones para cargar datos:
        - `cargarTokens()`: Recupera los tokens de `sessionStorage` y los muestra en la tabla.
        - `cargarErrores()`: Recupera los errores de `sessionStorage` y los muestra en la tabla.

3. Funcionalidad Detallada

- **Poblar Tablas**:
    - Las funciones `poblarTablaTokens()` y `poblarTablaErrores()` reciben un array de tokens o errores como entrada.

    - Obtienen la referencia a la tabla HTML correspondiente (`tablaTokens` o `tablaErrores`).
    - Limpian el contenido de la tabla antes de agregar nuevos datos (`tabla.innerHTML = '';`).
    - Iteran sobre el array de tokens o errores y crean filas (`<tr>`) y celdas (`<td>`) para cada elemento.
    - Asignan los valores de los tokens o errores a las celdas correspondientes.
- **Cargar Datos**:
    - Las funciones `cargarTokens()` y `cargarErrores()` recuperan los datos de `sessionStorage` utilizando `JSON.parse()`.
    - Verifican si los datos existen antes de llamar a las funciones `poblarTablaTokens()` o `poblarTablaErrores()`.

RecuentoTokens.js

1. Descripción General

- **Propósito**: `RecuentoTokens.js` contiene funciones para contar la frecuencia de aparición de cada lexema y generar una tabla HTML con el recuento.
- **Funcionalidad**:
    - Contar la frecuencia de aparición de cada lexema.
    - Generar una tabla HTML con el recuento de lexemas.
    - Recuperar datos de `sessionStorage`.

2. Estructura y Componentes

- **JavaScript**:
    - Funciones para contar lexemas:
        - `contarLexemas(tokens)`: Cuenta la frecuencia de aparición de cada lexema.
    - Funciones para poblar tablas:
        - `poblarTablaRecuentoLexemas(conteoLexemas)`: Genera una tabla con el recuento de lexemas.
    - Funciones para cargar datos:
        - `cargarRecuentoLexemas()`: Recupera los tokens de `sessionStorage`, cuenta los lexemas y los muestra en la tabla.

3. Funcionalidad Detallada

- **Contar Lexemas**:
    - La función `contarLexemas(tokens)` recibe un array de tokens como entrada.
    - Utiliza un objeto (`conteoLexemas`) para almacenar el recuento de cada lexema.

    - Itera sobre el array de tokens y actualiza el recuento de cada lexema en el objeto.
- **Poblar Tabla**:
    - La función `poblarTablaRecuentoLexemas(conteoLexemas)` recibe un objeto con el recuento de lexemas como entrada.
    - Obtiene la referencia a la tabla HTML correspondiente (`tablaRecuentoLexemas`).
    - Limpia el contenido de la tabla antes de agregar nuevos datos (`tabla.innerHTML = '';`).
    - Itera sobre las propiedades del objeto `conteoLexemas` y crea filas (`<tr>`) y celdas (`<td>`) para cada lexema y su recuento.
    - Asigna los valores de los lexemas y recuentos a las celdas correspondientes.
- **Cargar Recuento**:
    - La función `cargarRecuentoLexemas()` recupera los tokens de `sessionStorage` utilizando `JSON.parse()`.
    - Verifica si los tokens existen antes de llamar a las funciones `contarLexemas()` y `poblarTablaRecuentoLexemas()`.
    - Llama a `contarLexemas()` para obtener el recuento de lexemas.
    - Llama a `poblarTablaRecuentoLexemas()` para mostrar el recuento en la tabla.
- **Carga de Página**:
    - La función `cargarRecuentoLexemas()` se llama al cargar la página (`window.onload = cargarRecuentoLexemas;`).

SelectorArchivo.js

1. Descripción General

- **Propósito**: `SelectorArchivo.js` define la clase `SelectorArchivo` que maneja la funcionalidad de cargar archivos de texto en el analizador léxico. Permite al usuario seleccionar un archivo desde su sistema local y carga su contenido en el área de texto del editor y en el área de salida.
- **Funcionalidad**:
    - Permite seleccionar un archivo de texto desde el sistema local.
    - Carga el contenido del archivo en el textarea del editor y en el área de salida.
    - Inicializa el evento change del input file.

2. Estructura y Componentes

- **JavaScript**:
    - Clase `SelectorArchivo` con atributos:
        - `inputFile`: Elemento input de tipo file.
        - `textArea`: Elemento textarea del editor.

        - `outputArea`: Elemento textarea de salida.
    - Métodos:
        - `constructor(inputFileId, textAreaId, outputAreaId)`: Inicializa la instancia con los IDs de los elementos HTML.
        - `init()`: Inicializa el evento change del input file.
        - `cargarArchivo(event)`: Lee el contenido del archivo y lo carga en el textarea del editor y en el área de salida.
    - Función global:
        - `seleccionarArchivo()`: Simula un clic en el input de tipo `file`.
    - Manejo de eventos `DOMContentLoaded` para inicializar la clase.

3. Funcionalidad Detallada

- **Selección de Archivo**:
    - La función `seleccionarArchivo()` simula un clic en el input de tipo `file` (`inputArchivo`), que está oculto.
    - Cuando el usuario selecciona un archivo, se activa el evento `change` del input.
- **Carga de Contenido**:
    - El evento `change` llama a la función `cargarArchivo()`, que recibe el evento como parámetro.
    - Se utiliza un objeto `FileReader` para leer el contenido del archivo como texto.
    - El contenido leído se asigna al valor del textarea del editor (`textArea.value`) y al área de salida (`outputArea.value`).
- **Inicialización**:
    - La clase `SelectorArchivo` se inicializa cuando el DOM está completamente cargado.
    - El constructor obtiene las referencias a los elementos HTML y llama al método `init()`.
    - El método `init()` agrega un listener al evento `change` del input file.

GuardarArchivo.js

1. Descripción General

- **Propósito**: `GuardarArchivo.js` define la clase `GuardarTexto` que proporciona un método estático para guardar texto en un archivo descargable.
- **Funcionalidad**:
  - Permite guardar un texto en un archivo con nombre especificado por el usuario.
  - Asegura que el archivo tenga la extensión ".txt".
  - Crea un enlace descargable para el archivo.

2. Estructura y Componentes

- **JavaScript**:
  - Clase `GuardarTexto` con método estático:
    - `guardarComoArchivo(texto, nombreArchivo)`: Crea un archivo descargable con el texto proporcionado.

3. Funcionalidad Detallada

- **Guardar Texto**:
  - El método `guardarComoArchivo()` recibe el texto y el nombre del archivo como parámetros.
  - Verifica si el nombre del archivo tiene la extensión ".txt". Si no, la agrega.
  - Crea un objeto `Blob` con el texto y el tipo MIME "text/plain".
  - Crea una URL para el objeto `Blob` utilizando `URL.createObjectURL()`.
  - Crea un elemento `<a>` (enlace) con la URL y el atributo `download` establecido al nombre del archivo.
  - Agrega el enlace al cuerpo del documento y simula un clic para iniciar la descarga.
  - Elimina el enlace del documento y libera la URL del objeto `Blob`.

Busqueda.js

1. Descripción General

- **Propósito**: `Busqueda.js` define la clase `Búsqueda` que proporciona un método estático para buscar un patrón en un texto y resaltar las coincidencias.
- **Funcionalidad**:
  - Busca un patrón en un texto.
  - Resalta las coincidencias con una etiqueta `<span>` y la clase "highlight".
  - Cuenta el número de coincidencias encontradas.
  - Reemplaza los saltos de línea (`\n`) con etiquetas `<br>`.

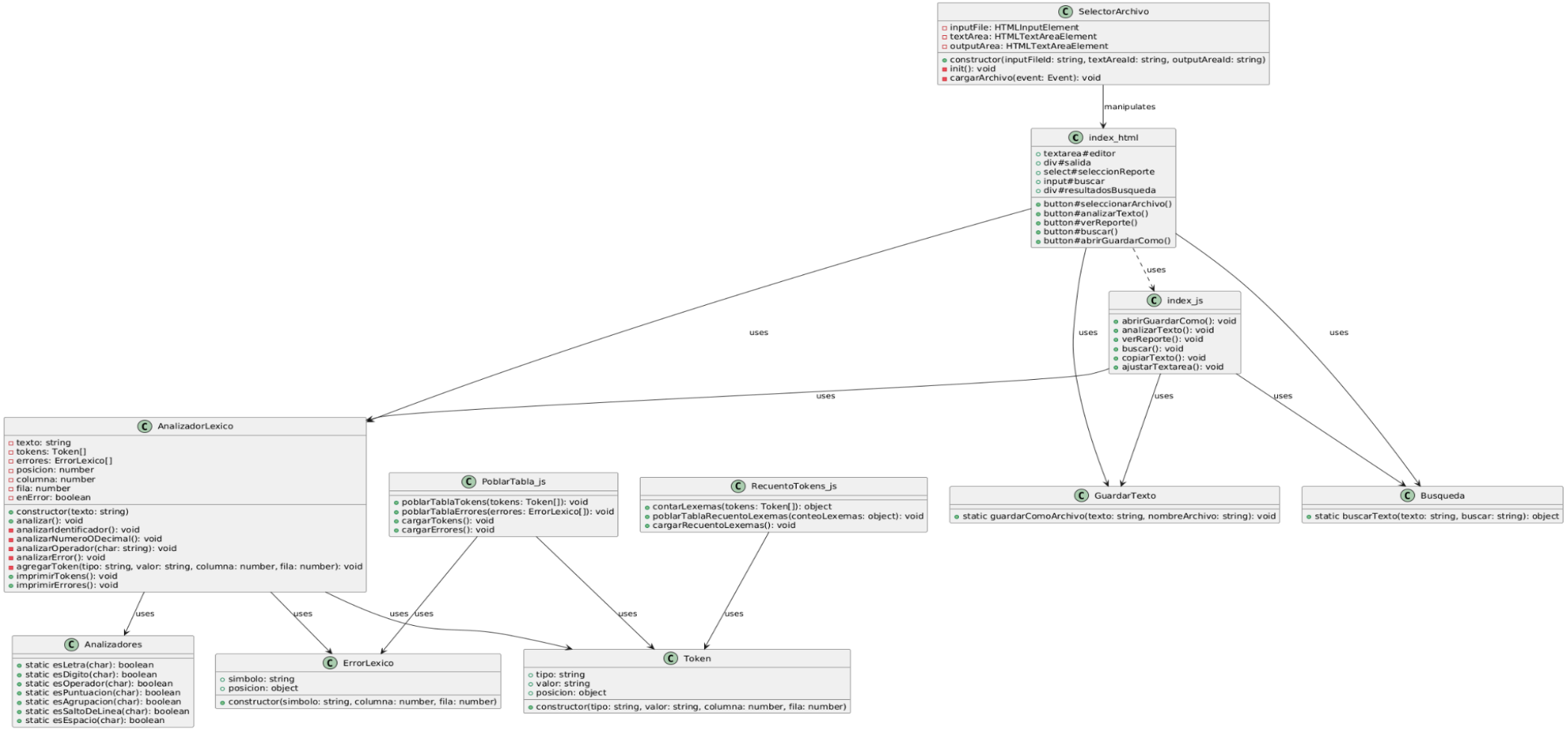
2. Estructura y Componentes

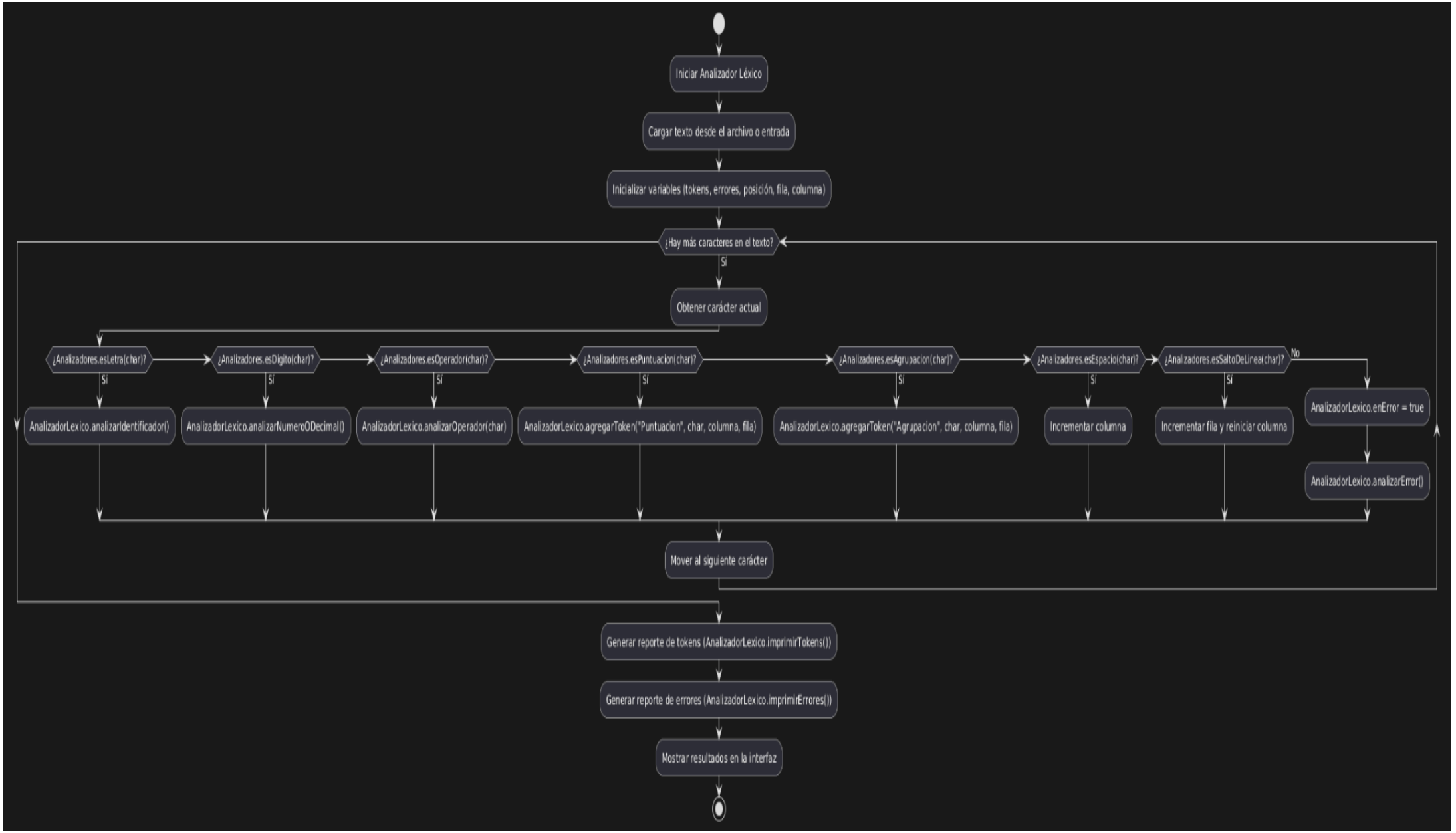
- **JavaScript**:
  - Clase `Búsqueda` con método estático:
    - `buscarTexto(texto, buscar)`: Busca el patrón en el texto y devuelve un objeto con el texto resaltado y el número de coincidencias.

3. Funcionalidad Detallada

- **Búsqueda de Texto**:
  - El método `buscarTexto()` recibe el texto y el patrón a buscar como parámetros.
  - Utiliza un bucle `while` para recorrer el texto y busca coincidencias manualmente.
  - Utiliza un bucle `for` anidado para encontrar el índice de la primera aparición del patrón.
  - Si se encuentra una coincidencia, incrementa el contador de coincidencias y agrega el texto resaltado al resultado.
  - Si no se encuentran más coincidencias, agrega el resto del texto al resultado y sale del bucle.
  - Reemplaza los saltos de línea (`\n`) con etiquetas `<br>`.
  - Devuelve un objeto con el texto resaltado y el número de coincidencias.

Diagramas


SelectorArchivo
inputFile: HTMLInputElement
textArea: HTMLTextAreaElement
outputArea: HTMLTextAreaElement
constructor(inputFileId: string, textAreaId: string, outputAreaId: string)
init(): void
cargarArchivo(event: Event): void
manipulates
index_html
textarea#editor
div#salida
select#seleccionReporte
input#buscar
div#resultadosBusqueda
button#seleccionarArchivo()
button#analizarTexto()
button#verReporte()
button#buscar()
button#abrirGuardarComo()
uses
index_js
abrirGuardarComo(): void
analizarTexto(): void
verReporte(): void
buscar(): void
copiarTexto(): void
ajustarTextarea(): void
AnalizadorLexico
texto: string
tokens: Token[]
errores: ErrorLexico[]
posicion: number
columna: number
fila: number
enError: boolean
constructor(texto: string)
analizar(): void
analizarIdentificador(): void
analizarNumeroODecimal(): void
analizarOperador(char: string): void
analizarError(): void
agregarToken(tipo: string, valor: string, columna: number, fila: number): void
imprimirTokens(): void
imprimirErrores(): void
PoblarTabla_js
poblarTablaTokens(tokens: Token[]): void
poblarTablaErrores(errores: ErrorLexico[]): void
cargarTokens(): void
cargarErrores(): void
RecuentoTokens_js
contarLexemas(tokens: Token[]): object
poblarTablaRecuentoLexemas(conteoLexemas: object): void
cargarRecuentoLexemas(): void
GuardarTexto
static guardarComoArchivo(texto: string, nombreArchivo: string): void
Busqueda
static buscarTexto(texto: string, buscar: string): object
Analizadores
static esLetra(char): boolean
static esDigito(char): boolean
static esOperador(char): boolean
static esPuntuacion(char): boolean
static esAgrupacion(char): boolean
static esSaltoDeLinea(char): boolean
static esEspacio(char): boolean
ErrorLexico
simbolo: string
posicion: object
constructor(simbolo: string, columna: number, fila: number)
Token
tipo: string
valor: string
posicion: object
constructor(tipo: string, valor: string, columna: number, fila: number)


Iniciar Analizador Léxico
Cargar texto desde el archivo o entrada
Inicializar variables (tokens, errores, posición, fila, columna)
¿Hay más caracteres en el texto?
Sí
Obtener carácter actual
¿Analizadores.esLetra(char)?
Sí
AnalizadorLexico.analizarIdentificador()
¿Analizadores.esDigito(char)?
Sí
AnalizadorLexico.analizarNumeroODecimal()
¿Analizadores.esOperador(char)?
Sí
AnalizadorLexico.analizarOperador(char)
¿Analizadores.esPuntuacion(char)?
Sí
AnalizadorLexico.agregarToken("Puntuacion", char, columna, fila)
¿Analizadores.esAgrupacion(char)?
Sí
AnalizadorLexico.agregarToken("Agrupacion", char, columna, fila)
¿Analizadores.esEspacio(char)?
Sí
Incrementar columna
¿Analizadores.esSaltoDeLinea(char)?
Sí
Incrementar fila y reiniciar columna
No
AnalizadorLexico.enError = true
AnalizadorLexico.analizarError()
Mover al siguiente carácter
Generar reporte de tokens (AnalizadorLexico.imprimirTokens())
Generar reporte de errores (AnalizadorLexico.imprimirErrores())
Mostrar resultados en la interfaz