

**Universidad de San Carlos de Guatemala**  
**Centro Universitario de Occidente**  
**División de Ingeniería**  
**Carrera: Ingeniería en Ciencias y Sistemas**  
**Curso: Laboratorio, Lenguajes Formales y de Programación**  
**Auxiliar: Carlos Pac**  
**Docente: Ing. Bryan Monzón**



**Trabajo: Manual Técnico, práctica 1**

**Estudiante: Jefferson Salvador Gómez Menchú**  
**[jeffersongomez201830892@cunoc.edu.gt](mailto:jeffersongomez201830892@cunoc.edu.gt)**

**Registro académico: 201830892**

**Quetzaltenango 06-03-2025**

## **Objetivos**

### **Objetivo General**

Comprender los principios básicos de Java Script, HTML y CSS, y aplicarlos para crear una solución de software de un analizador léxico.

### **Objetivos específicos**

- Identificar una solución acorde a una necesidad.
- Realizar una toma de requerimientos basado en una necesidad previamente identificada.
- Implementar una solución a través de programación web.
- Explicar el código utilizado tanto en Javascript como en HTML

## **Explicación de la Arquitectura y Tecnologías utilizadas**

### **Arquitectura:**

#### **1. FrontEnd (HTML)**

- Index.html: maquetación del dashboard principal y sus componentes
- Styles.css: diseño y personalización de los componentes modulares (divs, container, headers, body, footer, tables) (no se hizo uso de bootstrap).

#### **2. BackEnd (JavaScript)**

- **Estructura de archivos:** Donde se encuentran el punto de entrada que es el app.js, donde están las rutas debidas para conectar con el FrontEnd.

### **Tecnologías utilizadas:**

#### **1. FrontEnd:**

- HTML 5
- CSS
- LiveServer

#### **2. BackEnd:**

- Javascript

## FrontEnd:

En esta parte del código se maquetó lo que es el encabezado, la carga de archivos y la sección de botones, cada apartado tiene su respectivo div, esto con el objetivo que durante la personalización no haya tanto problema con los elementos y cada uno sea independiente.

Se crea el laber que nos permitiera la carga de archivos dentro de nuestra aplicación web, así como también se maquetan los botones que nos ayudaran con las funcionalidades completas de la página.

```
<> index.html • # styles.css JS app.js
Analizador Lexico > <> index.html > html > head
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Analizador Léxico</title>
7      <!-- vínculo que permite la personalización por medio de CSS-->
8      <link rel="stylesheet" href="styles.css">
9  </head>
10 <body>
11     <div class="container">
12         <!-- Encabezado -->
13         <header>
14             <h1>Analizador Léxico - Práctica 1</h1>
15         </header>
16
17         <main>
18             <!-- Sección de carga de archivo -->
19             <section class="seccion-de-entrada">
20                 <div class="control-de-archivo">
21                     <div class="subir-archivo">
22                         <label for="cargarArchivo" class="label-archivo">Seleccionar archivo</label>
23                         <input type="file" id="cargarArchivo" accept=".txt">
24                     </div>
25                     <!-- Sección de botones -->
26                     <div class="accion-de-botones">
27                         <button class="btn btn-limpiar" onclick="limpiarTexto()">Limpiar texto</button>
28                         <button class="btn btn-analizar" onclick="analizarTexto()">Analizar texto</button>
29                         <button class="btn btn-guardar" onclick="guardarTexto()">Guardar/Descargar texto</button>
30                     </div>
31                 </div>
32
33                 <div class="text-area-container">
34                     <textarea id="ingresarTexto" placeholder="Si no tiene un archivo, ingrese algún texto acá para evaluar."></textarea>
35                 </div>
36
37                 <div class="buscar-container">
38                     <input type="text" id="buscarPatron" placeholder="Ingrese una palabra para buscar un patrón">
39                     <button class="btn btn-buscar" onclick="buscarPatron()">Buscar</button>
40                 </div>
41
42                 <div id="mostrarResultados" class="mostrar-resultados"></div>
43             </section>
44         </main>
45     </div>
46 </body>
47 </html>
```

En el siguiente apartado se muestran como se encuentra la maquetacion del área de reportes, con sus respectivas tablas

```
<!-- Seccion de reportes-->
<section class="seccion-reportes">
  <div class="area-reporte">
    <h2>Reporte de Errores</h2>
    <div class="table-container">
      <table id="reporteErrores">
        <thead>
          <tr>
            <th>Error</th>
            <th>Descripción</th>
          </tr>
        </thead>
        <tbody></tbody>
      </table>
    </div>
  </div>

  <div class="area-reporte">
    <h2>Reporte de Tokens</h2>
    <div class="table-container">
      <table id="reporteTokens">
        <thead>
          <tr>
            <th>Token</th>
            <th>Tipo</th>
          </tr>
        </thead>
        <tbody></tbody>
      </table>
    </div>
  </div>
</div>
```

Por último aparece el footer, donde va la información del desarrollador y también el vínculo que permite la conexión con javascript

```
<!-- Footer-->
<footer>
  <p> 2025 Jefferson Gómez [201830892]- Lenguajes Formales y de Programación</p>
</footer>
</div>
<!-- Vínculo que permite conexión con javascript-->
<script src="app.js"></script>
</body>
</html>
```

## Estilos css:

En este apartado únicamente se usaron estilos básicos para poder dar personalización a la página y que sea un poco más dinámica y amigable con el usuario, en este apartado se encuentran las configuraciones globales o principales, se dejan colores predeterminados para usar más adelante, así como los bordes, sombras, y fondo. También se tienen en cuenta fuentes de tipo de letra que nos pueden ayudar a estilizar de mejor manera la aplicación

```
:root {
  --primary-color: #3498db;
  --secondary-color: #2c3e50;
  --text-color: #333;
  --light-text: #ecf0f1;
  --border-color: #bdc3c7;
  --metallic-light: #e0e0e0;
  --metallic-dark: #7f8c8d;
  --card-bg: rgba(255, 255, 255, 0.9);
  --shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  --transition: all 0.3s ease;
}

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: var(--text-color);
  background: linear-gradient(135deg, var(--metallic-light) 0%, var(--metallic-dark) 100%);
  background-attachment: fixed;
  min-height: 100vh;
}

.container {
  max-width: 1200px;
  margin: 0 auto;
  padding: 20px;
}

h1,
h2 {
  color: var(--secondary-color);
  margin-bottom: 20px;
}
```

Acá se observa la manera en que está estilizada tanto el área del header, y la sección de cargar archivo, con los márgenes, color de fondo, display y justificación del contenido

```
header {
  margin-bottom: 30px;
  padding: 20px;
  background-color: var(--card-bg);
  border-radius: 10px;
  box-shadow: var(--shadow);
}

main {
  display: grid;
  gap: 30px;
}

.seccion-de-entrada {
  background-color: var(--card-bg);
  border-radius: 10px;
  padding: 20px;
  box-shadow: var(--shadow);
}

.control-de-archivo {
  display: flex;
  flex-wrap: wrap;
  gap: 15px;
  margin-bottom: 20px;
  justify-content: space-between;
}

.subir-archivo {
  position: relative;
  overflow: hidden;
}

.label-archivo {
  display: inline-block;
  padding: 10px 15px;
  background-color: var(--secondary-color);
  color: var(--light-text);
  border-radius: 5px;
}
```

Acá se observa la estilización de los botones que nos serviran para aplicar funcionalidad en la aplicación. Se observa la configuracion del textArea

```
.btn {
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-weight: 600;
  transition: var(--transition);
  color: var(--light-text);
}

.btn-limpiar {
  background-color: #34495e;
}

.btn-analizar {
  background-color: #34495e;
}

.btn-guardar {
  background-color: #084823;
}

.btn-buscar {
  background-color: #34495e;
}

.btn:hover {
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
}

.text-area-container {
  margin-bottom: 20px;
}

textarea {
  width: 100%;
  height: 200px;
  padding: 15px;
}
```



Acá se tiene la personalizacion de la seccion de reportes, al igual que sus tablas, para que muestre la inforacion idonea y luzca estetico

```
.seccion-reportes {
  display: grid;
  gap: 20px;
}

.area-reporte {
  background-color: var(--card-bg);
  border-radius: 10px;
  padding: 20px;
  box-shadow: var(--shadow);
}

.table-container {
  overflow-x: auto;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 10px;
}

table,
th,
td {
  border: 1px solid var(--border-color);
}

th {
  background-color: var(--secondary-color);
  color: var(--light-text);
  padding: 12px 15px;
  text-align: left;
}
```

Por ultimo se ve el diseño del footer, y de las secciones que requieren un diseño responsivo para que los divs que los almacenan no sufran ningun movimiento cambio brusco despues de la maquetacion

```
/* Footer */
footer {
  text-align: center;
  margin-top: 40px;
  padding: 20px;
  color: #121212;
  font-size: 0.9rem;
  font-weight: bold;
}

/* Responsive Design */
@media (min-width: 768px) {
  .seccion-reportes {
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
  }
}

@media (max-width: 767px) {
  .control-de-archivo {
    flex-direction: column;
  }

  .accion-de-botones {
    width: 100%;
    justify-content: space-between;
  }

  h1 {
    font-size: 2rem;
  }
}
```

**BackEnd:**

**Endpoints:**

**Puntos de acceso que conectan JavaScript con HTML**

```
index.html  # styles.css  JS app.js  X
Analizador Lexico > JS app.js > esIdentificador
1  //Variables globales, para llamar a los id desde html y generar END POINTS
2  let ingresarTexto = document.getElementById('ingresarTexto');
3  let cargarArchivo = document.getElementById('cargarArchivo');
4  let reporteErrores = document.getElementById('reporteErrores');
5  let reporteTokens = document.getElementById('reporteTokens');
6  let conteoLexemas = document.getElementById('conteoLexemas');
7  let mostrarResultados = document.getElementById('mostrarResultados');
8
```

Funciones que permiten la subida de una archivo por medio de un selector de archivos gracias a la implementacion de FileReader y también la funcion de limpiar texto, donde simplemente se emplea todos los valores como vacios, para poder reiniciar el proceso de analizar

```
//Evento para boton de cargar archivo
cargarArchivo.addEventListener('change', function(event) {
    let file = event.target.files[0];
    if (file) {
        let reader = new FileReader();
        reader.onload = function(e) {
            ingresarTexto.value = e.target.result;
        };
        reader.readAsText(file);
    }
});

//Funcion para limpiar texto
function limpiarTexto() {
    ingresarTexto.value = '';
    cargarArchivo.value = '';
    buscarPatron.value = '';
    reporteErrores.innerHTML = '';
    reporteTokens.innerHTML = '';
    conteoLexemas.innerHTML = '';
    mostrarResultados.innerHTML = '';
}
```

Por medio de ciclos anidados podemos establecer la funcion de analizar texto, ya que se establece un for que por medio de una varibale va evaluando linea por línea y carácter por carácter ya que la misma cadena ha sido separada por medio de la funcion split, entonces se puede evaluar cada carácter, y por medio del while anidado se va tomando cada caracter como una columna diferente para poder realizar las evaluaciones necesarias, se toman instancias de funciones que más adelante profundizaremos, esto con el fin de evaluar el tipo de token que es cada carácter o en su defecto si tiene cierto tipo de estructura, la cadena completa se establece como un token determinado. Por ultimo se realiza una validacion para determinar si hay errores se genera un reporte de errores, de lo contrario genera un reporte de tokens y lexemas.

```
index.html # styles.css JS app.js X
Analizador Lexico > JS app.js > ...
33 //Funcion para analizar texto
34 function analizarTexto() {
35     let texto = ingresarTexto.value;
36     let linea = texto.split('\n');
37     let errores = [];
38     let tokens = [];
39     let lexemas = [];
40
41     for (let i = 0; i < linea.length; i++) {
42         let line = linea[i];
43         let columna = 0;
44
45         while (columna < line.length) {
46             let char = line[columna];
47
48             //No se toman en cuenta los espacios en blanco
49             if (char === ' ' || char === '\t') {
50                 columna++;
51                 continue;
52             }
53
54             // Se Identifican los tokens
55             if (esLetra(char)) {
56                 // El token es un identificador
57                 let token = esIdentificador(line, columna);
58                 tokens.push({ token: 'Identificador', lexema: token.lexema, line: i + 1, columna: columna + 1 });
59                 lexemas[token.lexema] = (lexemas[token.lexema] || 0) + 1;
60                 columna += token.length;
61             } else if (esDigito(char)) {
62                 // El token es un número o decimal
63                 let token = verificarNumero(line, columna);
64                 if (token.type === 'Número Entero') {
65                     tokens.push({ token: 'Número Entero', lexema: token.lexema, line: i + 1, columna: columna + 1 });
66                 } else {
67                     tokens.push({ token: 'Decimal', lexema: token.lexema, line: i + 1, columna: columna + 1 });
68                 }
69                 lexemas[token.lexema] = (lexemas[token.lexema] || 0) + 1;
70                 columna += token.length;
71             } else if (esSimbolo(char)) {
72                 // El token es un símbolo de los tipos operadores, puntuación o agrupación
73                 let token = tiposSimbolo(line, columna);
74                 if (token.type) {
75                     tokens.push({ token: token.type, lexema: token.lexema, line: i + 1, columna: columna + 1 });
76                     lexemas[token.lexema] = (lexemas[token.lexema] || 0) + 1;
77                     columna += token.length;
78                 } else {
79                     errores.push({ simbolo: char, line: i + 1, columna: columna + 1 });
80                     columna++;
81                 }
82             } else {
83                 //Si el carácter no forma parte del lenguaje que se indica entonces se reporta como error.
84                 errores.push({ simbolo: char, line: i + 1, columna: columna + 1 });
85                 columna++;
86             }
87         }
88     }
89
90     if (errores.length > 0) {
91         generarReporteErrores(errores);
92     } else {
93         generarReporteToken(tokens);
94         generarConteoLexemas(lexemas);
95     }
96 }
```

Acá se muestran los caracteres permitidos cada cada tipo de token, de está manera tenemos una mejor precision en nuestro análisis

```
// Funciones para verificar tipos de caracteres que pertenece a cada tipo de token
function esLetra(char) {
    return (char >= 'a' && char <= 'z') || (char >= 'A' && char <= 'Z');
}

function esDigito(char) {
    return char >= '0' && char <= '9';
}

function esSimbolo(char) {
    const simbolos = ['+', '-', '*', '/', '%', '=', '<', '>', '!', '&', '|', '(', ')', '[', ']', '{', '}', '.', ',', ';', ':'];
    return simbolos.includes(char);
}
```

Acá se realizan las validaciones respectivas para saber el tipo de token que es el carácter o en su defecto la cadena de datos, por medio de un while que recorre la longitud de la palabra o numero y en caso de que lo sea se incremente un contador, para llevar la contabilidad de los lexemas

```
// Función para saber si un token es identificador
function esIdentificador(line, inicio) {
    let lexema = '';
    let i = inicio;
    while (i < line.length && (esLetra(line[i]) || esDigito(line[i]))) {
        lexema += line[i];
        i++;
    }
    return { lexema: lexema, length: lexema.length };
}

// Función para saber si un token es número entero o decimal
function verificarNumero(line, inicio) {
    let lexema = '';
    let i = inicio;
    let esDecimal = false;

    while (i < line.length && (esDigito(line[i]) || line[i] === '.')) {
        if (line[i] === '.') {
            if (esDecimal) break; // Solo se permite un punto para verificar si es decimal
            esDecimal = true;
        }
        lexema += line[i];
        i++;
    }

    return {
        lexema: lexema,
        length: lexema.length,
        type: esDecimal ? 'Decimal' : 'Número Entero'
    };
}
```

Por el contrario, si el carácter es representado por un símbolo la validación se realiza por medio de un switch que nos permite una estructura más organizada y una estética visual, así como una mejor comprensión en el proceso de desarrollo de código

```
144 // Funcion para saber el tipo de simbolo al que pertenece el token
145 function tiposSimbolo(line, inicio) {
146     let char = line[inicio];
147     let nextChar = line[inicio + 1];
148     let lexema = char;
149     let type = null;
150
151     switch (char) {
152         case '<':
153         case '>':
154         case '=':
155         case '!':
156             if (nextChar === '=') {
157                 lexema += nextChar;
158                 type = 'Operador Relacional';
159             } else if (char === '=') {
160                 type = 'Operador de Asignación';
161             } else {
162                 type = 'Operador Relacional';
163             }
164             break;
165
166         case '&':
167         case '|':
168             if (nextChar === char) {
169                 lexema += nextChar;
170                 type = 'Operador Lógico';
171             }
172             break;
173
174         case '+':
175         case '-':
176         case '*':
177         case '/':
178         case '%':
179             type = 'Operador Aritmético';
180             break;
181
182         case '(':
183         case ')':
184         case '[':
185         case ']':
186         case '{':
187         case '}':
188             type = 'Agrupación';
189             break;
190
191         case '.':
192         case ',':
193         case ';':
194         case ':':
195             type = 'Puntuación';
196             break;
197
198         default:
199             break;
200     }
201
202     return {
203         lexema: lexema,
204         length: lexema.length,
205         type: type
206     };
207 }
```

En este apartado se encuentran los reportes generados unicamente en formato de tabla, con maquetacion de html, con los respectivos requisitos de la práctica y cada dato acorde a lo que se pide.

```
//Funciones para poder generar los reportes de errores, tokens y lexemas

function generarReporteErrores(errores) {
    reporteErrores.innerHTML = '<tr><th>Simbolo</th><th>Fila</th><th>Columna</th></tr>';
    errores.forEach(error => {
        let fila = '<tr><td>${error.simbolo}</td><td>${error.line}</td><td>${error.columna}</td></tr>';
        reporteErrores.innerHTML += fila;
    });
    reporteTokens.innerHTML = '';
    conteoLexemas.innerHTML = '';
}

function generarReporteToken(tokens) {
    reporteTokens.innerHTML = '<tr><th>Token</th><th>Lexema</th><th>Fila</th><th>Columna</th></tr>';
    tokens.forEach(token => {
        let fila = '<tr><td>${token.token}</td><td>${token.lexema}</td><td>${token.line}</td><td>${token.columna}</td></tr>';
        reporteTokens.innerHTML += fila;
    });
    reporteErrores.innerHTML = '';
}

function generarConteoLexemas(lexemas) {
    conteoLexemas.innerHTML = '<tr><th>Lexema</th><th>Cantidad</th></tr>';
    for (let lexema in lexemas) {
        let fila = '<tr><td>${lexema}</td><td>${lexemas[lexema]}</td></tr>';
        conteoLexemas.innerHTML += fila;
    }
}
```

En el apartado de patrones se redirige al html para poder conocer el valor que se buscara y de igual forma que al realizar las validaciones, se procede a buscar carácter por carácter, se declara una variable que va a ser la encargada de subrayar los caracteres encontrados y que coincidan, por medio de ciclos, al igual que anteriormente iremos comparando linea por linea y carácter por carácter, esto nos ayuda a tener más precision con la busqueda de patrones.

```
//Funcion para buscar patrones
function buscarPatron() {
    let patron = document.getElementById('buscarPatron').value;
    let texto = ingresarTexto.value;

    //En caso de que el buscador de patron esté vacío no realiza ninguna acción
    if (patron.length === 0) {
        mostrarResultados.innerHTML = "<p>Ingrese un patrón para buscar.</p>";
        return;
    }

    let linea = texto.split('\n');
    let cont = 0;
    let textoSubrayado = '';

    for (let i = 0; i < linea.length; i++) {
        let line = linea[i];
        let lineaResaltada = '';
        let j = 0;
        // Procesar cada línea carácter por carácter
        while (j < line.length) {
            if (j <= line.length - patron.length) {
                let encontrado = true;
                for (let pos = 0; k < patron.length; k++) {
                    if (line[j + pos] !== patron[pos]) {
                        encontrado = false;
                        break;
                    }
                }

                if (encontrado) {
                    lineaResaltada += '<span style="background-color: gray;">' + patron + '</span>';
                    cont++;
                    j += patron.length;
                    continue;
                }

                // Si no hay coincidencia, añadimos el carácter actual
                lineaResaltada += line[j];
                j++;
            }
        }
        // Añadimos la línea procesada al resultado, preservando la estructura original
        textoSubrayado += '<div>' + lineaResaltada + '</div>';
    }

    // Mostrar resultados
    let resultHTML = '<p><strong>Patron buscado:</strong> ${patron}</p>';
    resultHTML += '<p><strong>No. Repeticiones:</strong> ${cont} veces</p>';
    resultHTML += '<div><strong>Texto resaltado:</strong><br>${textoSubrayado}</div>';

    mostrarResultados.innerHTML = resultHTML;
}
```



Por último tenemos la función de guardar y descargar el archivo de entrada de texto si así se desea, para esta función se hizo uso de Blob, que permite la descarga de archivos, así como el conocimiento de su tamaño y de la función “URL.createObjectURL”, cuya función es extraer la información del Blob y de esta manera descargar el archivo a la computadora. Después de hacer click en el botón establecido, luego simplemente con referencias, se le brinda un nombre predeterminado para que quede almacenado en nuestra computadora.

```
//Funcion para poder guardar y descargar la entrada de texto en caso que así se quiera

function guardarTexto() {
  let texto = ingresarTexto.value;
  let blob = new Blob([texto], { type: 'text/plain' }); //Permite la descarga del archivo editado o generado en el editor de texto
  let url = URL.createObjectURL(blob);
  let a = document.createElement('a');
  a.href = url;
  a.download = 'texto_analizado_correctamente.txt';
  a.click();
  URL.revokeObjectURL(url);
}
```