# Manual Técnico

Proyecto 2

Kevin Steve Martinez Lemus
[OLC1] Universidad De San Carlos De Guatemala

# Índice

# I. Introducción

El presente documento describe los aspectos técnicos informáticos del sistema de información. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

## Objetivo

Instruir el uso adecuado del Sistema de Información, para el acceso oportuno y adecuado en la modificación de este, mostrando la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración y soporte de este.

## Requerimientos

- Tener un navegador.
- Se recomienda tener un editor de código (Visual Studio Code u otro) para ver y editar el código de este programa.

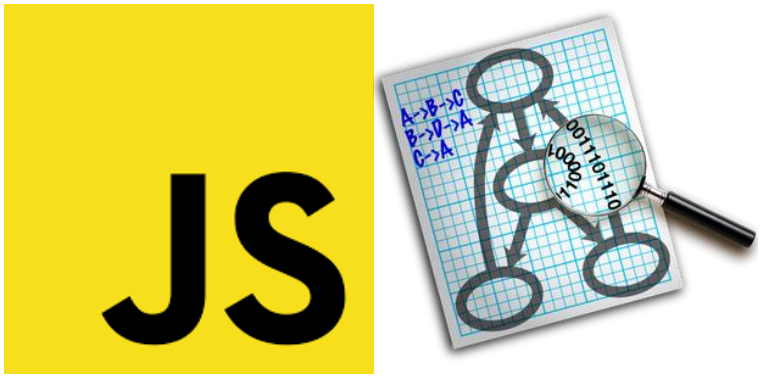
# II. Descripción Del Sistema

## 1. Descripción del Contenido del Sistema

Este es un sistema en el cual el usuario interactúa directamente con una interfaz gráfica específicamente en una página web, este programa permite la lectura de un código escrito por el usuario, lo analiza y crea autómatas con las expresiones regulares descritas, verifica entradas con las expresiones regulares, abre y guarda archivos .cst. Se trabajo con frontend y backend por separado.

## 2. Lenguajes Utilizados

Para la creación de este sistema todas las funcionalidades se realizaron en lenguaje de JAVA y también se utilizó Graphviz para la creación de reportes. Utilizando un enfoque al paradigma de programación POO (Programación Orientada a Objetos). Creando las diferentes estructuras de datos para almacenar la información.




Todo realizado en el IDE NetBeans 8.2:

Todo orientado a servidores web, utilizando react y express con NodeJs para su creación.

## 3. FrontEnd

Este fue realizado con la ayuda de react en el cual por medio de javascript se pudieron realizar las diferentes funciones que ayudan a tener una interfaz agradable al usuario, con ayuda de monaco para crear el editor de texto funcional, este se utilizo de la siguiente manera:

```
<Editor
  height="70vh"
  defaultLanguage="java"
  defaultValue="// CompScript :)"
  onMount={handleEditorDidMount}
  className={'rounded-xl'}
  theme="vs-dark"
/>
```

```
import Editor from '@monaco-editor/react';
```

Se utilizo Bootstrap para la mejora visual de la página web, teniendo así una mejor interfaz gráfica.

```
<!-- CSS only -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">
<!-- JavaScript Bundle with Popper -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
  crossorigin="anonymous"></script>
```

Para tener una conexión estable con el backend se utilizaron las peticiones fetch descritas así:

```
function ejecutar() {
  var obj = { 'codigo': editorRef.current.getValue() }

  fetch(`http://localhost:5000/ejecutar`, {
    method: 'POST',
    body: JSON.stringify(obj),
    headers: {
      'Content-Type': 'application/json',
      'Access-Control-Allow-Origin': '*',
    }
  })
    .then(res => res.json())
    .catch(err => {
      console.error('Error:', err)
      alert("Ocurrio un error, ver la consola")
    })
    .then(response => {
      console.log(response);
      document.getElementById('consola').textContent = response.message;
      ast = response.ast;
      simbolos = response.simbolos;
    })
}
```

## 4. BackEnd

El backend se realizo con NodeJs Express en el cual se creaban las peticiones de la siguiente manera:

```
app.get('/', (req, res) => {
    res.json({
        message: 'Hello World 2.0'
    });
});
```

Para ejecutar el código se utilizó la siguiente petición:

```
app.post('/ejecutar', function (req, res){
    //AST
    let ast;
    try {
        const entrada = req.body.codigo;
        ast = parser.parse(entrada.toString());
    } catch (e) {
        console.error(e);
        return;
    }
    const tsGlobal = new TS([],[]);
    guardarFunciones(ast,tsGlobal);
    var respuesta = procesarBloque(ast, tsGlobal, true).salida;

    //respuesta
    res.json({
        message: respuesta,
        ast: ast,
        simbolos: tsGlobal
    });

});
```

## 4.1 Gramática

Para poder procesar la gramática, se utilizo la herramienta de Jison, de la siguiente manera:

```
%lex

%options case-insensitive

%%

\s+                                 // se ignoran espacios en blanco
"//".*                              // comentario simple línea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/] // comentario multiple líneas

\"[^\"]*\"                          { yytext = yytext.substr(1,yyleng-2); return 'CADENA'; }
\'(\\(n|\"|\'|\\|t|r)|.)\'          { yytext = yytext.substr(1,yyleng-2); return 'CARACTER'; }
```

Al ejecutar el comando jison gramática.jison (nombre del archivo jison) se crea automáticamente una clase de js en la que se parsea una entrada y se crea un AST para poder ejecutar las instrucciones especificadas:

JS gramatica.js

Con ayuda de clases auxiliares como la de tabla de símbolos, la cual guarda todos los símbolos de la producción:

```
class TS{

    constructor (simbolos,funciones) {
        this._simbolos = simbolos;
        this._funciones = funciones;
    }
```

Y la clase de instrucciones la cual se implemento en el Jison para poder identificar las difernetes instrucciones que se pedían:

```
const instruccionesAPI = {
    nuevoOperacionBinaria: function(operandoIzq, operandoDer, tipo) {
        return nuevaOperacion(operandoIzq, operandoDer, tipo);
    },
    nuevoOperacionUnaria: function(operando, tipo) {
        return nuevaOperacion(operando, undefined, tipo);
    },
    nuevoValor: function(valor, tipo) {
        return {
            tipo: tipo,
            valor: valor
        }
    },
    nuevoImprimirLn: function(expresion) {
        return {
            tipo: TIPO_INSTRUCCION.IMPRIMIRLN,
            expresion: expresion
        };
    },
    nuevoImprimir: function(expresion) {
        return {
            tipo: TIPO_INSTRUCCION.IMPRIMIR,
            expresion: expresion
        };
    },
```