Nombre: Marco Sebastian Solares España
Carnet: 202004822
Curso: OLC1

Manual Tecnico

Lenguaje utilizado:
JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas. Desde actualizar fuentes de redes sociales a mostrar animaciones y mapas interactivos, las funciones de JavaScript pueden mejorar la experiencia del usuario de un sitio web. Como lenguaje de scripting del lado del servidor, se trata de una de las principales tecnologías de la World Wide Web. Por ejemplo, al navegar por Internet, en cualquier momento en el que vea un carrusel de imágenes, un menú desplegable “click-to-show” (clic para mostrar), o cambien de manera dinámica los elementos de color en una página web, estará viendo los efectos de JavaScript.

Herramientas utilizadas: Jison, React, Nodejs, Boostrap, Graphviz,hashmap.

Parser jison: para la realizacion del analizador lexico y sintactico se utilizo la herramienta jison que nos permite generar estos parsers.

```
    const sng = require('../Interprete/singleton/Manager.js');


%}

%left 'OR'
%left 'AND'
%left 'TERNARIO'
%left 'REL_IGUAL' 'DIFERENTE' 'MENOR' 'MENORIGUAL' 'MAYOR'  'MAYORIGUAL'
%left 'MAS' 'MENOS'
%left 'DIV' 'POR'
%left 'POTENCIA' 'MODULO'


%right 'NOT'

%nonassoc 'PARIZQ' 'PARDER'



// -------> Simbolo Inicial
%start inicio


%% // ------> Gramatica

inicio
    : listainstruccion EOF  {$$=$1; return $$;}
;

listainstruccion
    : listainstruccion instruccion   { $$ = $1; $$.push($2);}
```

Este lenguaje de programacion fue creado en dos partes una de lado cliente y otra de servidor.

Cliente: para el lado del cliente se utilizo la herramienta react para poder levantar un frontend y poder mandar o recibir cosas al backend.

```
export const Home = () => {
    const [entrada, setData] = useState('');

    const handleOpenFile = event => {
        const file = event.target.files[0]; // Obtiene el primer archivo se

        if (file && file.name.endsWith('.sc')) {
            const fileReader = new FileReader();


            fileReader.onload = () => {
                const fileContent = fileReader.result;
                // lo ponemos en el textarea
                document.querySelector('.text-area').value = fileContent;

                //guardar en el estado
                setData(fileContent);


                console.log('Contenido del archivo:', fileContent);
            };


            fileReader.readAsText(file);
        } else {
            alert('Por favor selecciona un archivo con extensión .sc');
        }
    };
```

```
return (


    <div className="screen">
        <nav class="navbar navbar-expand-lg navbar-light bg-light">
            <a class="navbar-brand" href="#">Navbar</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNa
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">

                    <li class="nav-item">
                        <button id="crearArchivoBtn" class="btn btn-light" onClick={createFile} type="button">Crear Archivo</button>
                    </li>
                    <li class="nav-item">
                        <input type="file" accept=".sc" style={{ display: 'none' }} onChange={handleOpenFile} id="fileInput" />
                        <label htmlFor="fileInput" className="btn btn-light" >Abrir Archivo</label>
                    </li>
```

Servidor: para el lado del servidor se utilizo js con node js para levantar un servidor backend para poder realizar toda la logica del programa, se utilizo el patron interprete para poder realizarlo.

En el patron interprete se divide en dos secciones. Instrucciones y expresiones

Instrucciones: para realizar una accion pero no de volver un resultado, como por ejemplo las funciones, ifs,for, while etc.

IF:

```
class If extends instruccion {
    constructor(condicion, instr, siNo,fila, columna) {
        super(condicion, instr, fila, columna);
        this.condicion = condicion;
        this.instr = instr;
        this.siNo = siNo;

    }

    interpretar(entorno) {
        let entornoIf = new Entorno(TipoInstruccion.IF, entorno);
        let cond = this.condicion.interpretar(entornoIf);
        let value ="";

        if (this.condicion.tipo != 'BOOL') {
            console.log('Error semantico: la condicion no es booleana');
            return "Error semantico: la condicion no es booleana";
        }
        if (String(this.condicion.valor).toLowerCase() === "true") {

            for (let i = 0; i < this.instr.length; i++) {
                let instruccion = this.instr[i];
                value += instruccion.interpretar(entornoIf);
                value += "\n";
                if (instruccion.tipo == TipoInstruccion.BREAK) {
                    this.tipo= TipoInstruccion.BREAK;
                    break;
                }else if (instruccion.tipo == TipoInstruccion.CONTINUE) {
                    this.tipo= TipoInstruccion.CONTINUE;
                    continue;
                }
            }

            return value;
            //GUARDAMOS ENTORNO
```

FOR:

```
interpretar(entorno){
    let entornoFor = new Entorno(TipoInstruccion.FOR,entorno);
    let declara = this.declaracion.interpretar(entornoFor);

    let cond = this.condicion.interpretar(entornoFor);

    let value ="";
    if(this.condicion.tipo != 'BOOL'){
        console.log('Error semantico: la condicion no es booleana');
        return this;
    }

    while(this.condicion.valor){
        let result = TipoInstruccion.FOR;
        for (let i =0;i<this.instr.length;i++){
            let instruccion = this.instr[i];
            value += instruccion.interpretar(entornoFor);
            value += "\n";
            if(instruccion.tipo ==  TipoInstruccion.BREAK){
                result = TipoInstruccion.BREAK;
                break;
            } else if (instruccion.tipo == TipoInstruccion.CONTINUE) {
                result = TipoInstruccion.CONTINUE;
                continue;
            }
        }
        if (result == TipoInstruccion.BREAK){

            break;
        } else if (result == TipoInstruccion.CONTINUE) {
```

Expresion: en cambio la expresion es un metodo del patron interprete que devuelve un resultado, como por ejemplo una operación aritmetica,relacional,logica etc.

Ejemplo relacionales:

```
if (this.op1.tipo == TipoDato.ENTERO && this.op2.tipo == TipoDato.ENTERO
    || this.op1.tipo == TipoDato.ENTERO && this.op2.tipo == TipoDato.DECIMAL
    || this.op1.tipo == TipoDato.DECIMAL && this.op2.tipo == TipoDato.ENTERO
    || this.op1.tipo == TipoDato.DECIMAL && this.op2.tipo == TipoDato.DECIMAL
    || this.op1.tipo == TipoDato.ENTERO && this.op2.tipo == TipoDato.CHAR
    || this.op1.tipo == TipoDato.CHAR && this.op2.tipo == TipoDato.ENTERO
    || this.op1.tipo == TipoDato.ENTERO && this.op2.tipo == TipoDato.CHAR
    || this.op1.tipo == TipoDato.CHAR && this.op2.tipo == TipoDato.CHAR
    || this.op1.tipo == TipoDato.BOOL && this.op2.tipo == TipoDato.BOOL
    || this.op1.tipo == TipoDato.CADENA && this.op2.tipo == TipoDato.CADENA
    || this.op1.tipo == TipoDato.DECIMAL && this.op2.tipo == TipoDato.CHAR
    || this.op1.tipo == TipoDato.CHAR && this.op2.tipo == TipoDato.DECIMAL) {

    switch (this.operador) {
        case "==":
            this.tipo = TipoDato.BOOL;
            this.valor = op1 == op2;
            return this.valor;
        case "!=":
            this.tipo = TipoDato.BOOL;
            this.valor = op1 != op2;
            return this.valor;
        case ">":
            this.tipo = TipoDato.BOOL;
            this.valor = op1 > op2;
            console.log(this.valor)
            return this.valor;
        case "<":
            this.tipo = TipoDato.BOOL;
            this.valor = op1 < op2;
            return this.valor;
```

Y asi funcionan todas las clases de expresiones e instrucciones dentro del programa.

Graficar AST: para graficar el arbol utilizamos una herramienta llamada graphviz que sirve para generar estos grafos, es muy importante utilzarlo para mostrar el arbol AST generado.

```
function graficarArbol(arbolitos) {
    contador = 1;
    cuerpo = '';
    graphAST('n0', arbolitos);
    let principal = `digraph arbolAST{
      n0[label="${arbolitos.valor.replace('"', '\\"')}"];
      ${cuerpo}
    }`;
    fs.writeFile('arbolAST.dot', principal, () => { });
    (0, child_process_1.exec)('dot -Tsvg arbolAST.dot -o arbolAST.svg', (error, stdout, stderr) => {
        if (error) {
            return;
        }
        if (stderr) {
            return;
        }
    });
    return principal;
}
module.exports.graficarArbol = graficarArbol;
```

```
Ssolares0, el mes pasado | 1 author (Ssolares0)
class NodoAst{
    constructor(valor){
        this.valor = valor;
        this.listaHijos = [];
    }

    agregarHijo(val){
        this.listaHijos.push(new NodoAst(val));
    }

    agregarHijoAST(hijo){
        if (hijo!==undefined){
            this.listaHijos.push(hijo);
        }
    }



}
exports.NodoAst = NodoAst;      Ssolares0, el mes pasado • cre
```

Archivo Manager: el archivo manager es un archivo creado para hacer la estructura de los htmls de los reportes de simbolos y errores, en este archivo hay varias funciones como por ejemplo agregar error o obtener error. Estas funciones generan una variable de texto con la estructura dentro apra luego mandarla al frontend y utilizarla para crear el html.

```
function addError(error) {
    errorStorage +=
        `<tr class="bg-gray-800 border-gray-700 hover:bg-gray-600">
            <th scope="row" class="py-4 px-6 font-medium whitespace-nowrap text-white">${error.title}</th>
            <td class="py-4 px-6">${error.description}</td>
            <td class="py-4 px-6">${error.fila}</td>
            <td class="py-4 px-6">${error.columna}</td>
            <td class="py-4 px-6">${error.error}</td>
        </tr>\n`;
    console.log("Error agregado");
}

function getError() {
    return `
    <table class="w-full text-sm text-left text-gray-400">
        <thead class="text-xs uppercase bg-gray-700 text-gray-400">
            <tr>
                <th scope="col" class="py-3 px-6">Tipo de error</th>
                <th scope="col" class="py-3 px-6">Descripción</th>
                <th scope="col" class="py-3 px-6">Línea</th>
                <th scope="col" class="py-3 px-6">Columna</th>
                <th scope="col" class="py-3 px-6">Error</th>
            </tr>
        </thead>
        <tbody>${errorStorage}</tbody>
    </table>`;
}
```

Index Controller: Este archivo del patron interprete sirve para manejar las rutas del backend y poder poner los endpoint para recibir las entradas y devolver una respuesta al backend.


```
function addError(error) {
    errorStorage +=
        `<tr class="bg-gray-800 border-gray-700 hover:bg-gray-600">
            <th scope="row" class="py-4 px-6 font-medium whitespace-nowrap text-white">${error.title}</th>
            <td class="py-4 px-6">${error.description}</td>
            <td class="py-4 px-6">${error.fila}</td>
            <td class="py-4 px-6">${error.columna}</td>
            <td class="py-4 px-6">${error.error}</td>
        </tr>\n`;
    console.log("Error agregado");
}

function getError() {
    return `
    <table class="w-full text-sm text-left text-gray-400">
        <thead class="text-xs uppercase bg-gray-700 text-gray-400">
            <tr>
                <th scope="col" class="py-3 px-6">Tipo de error</th>
                <th scope="col" class="py-3 px-6">Descripción</th>
                <th scope="col" class="py-3 px-6">Línea</th>
                <th scope="col" class="py-3 px-6">Columna</th>
                <th scope="col" class="py-3 px-6">Error</th>
            </tr>
        </thead>
        <tbody>${errorStorage}</tbody>
    </table>`;
}
```