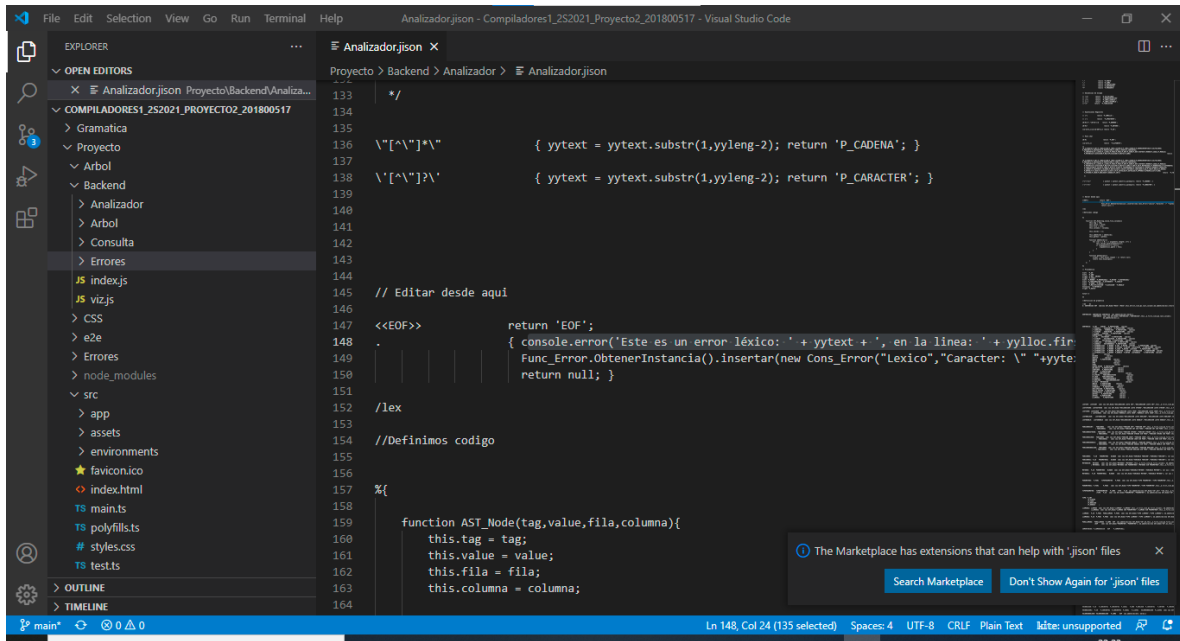
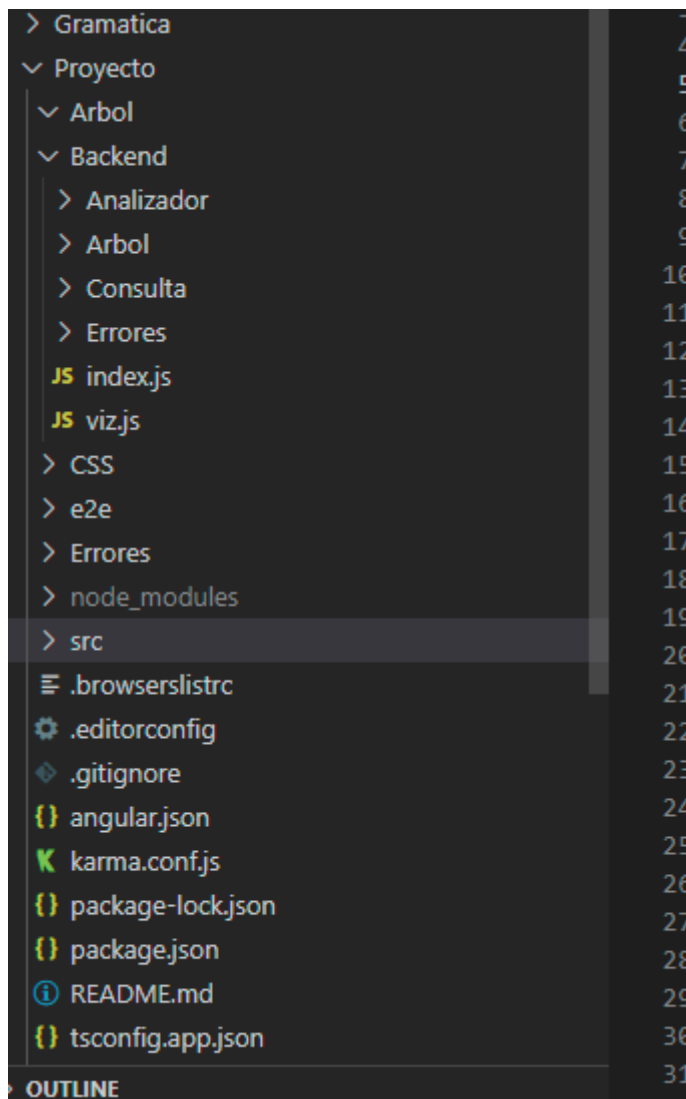


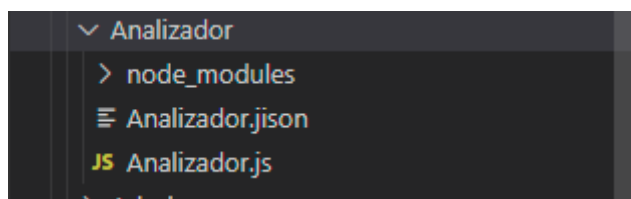
# Manual Tecnico



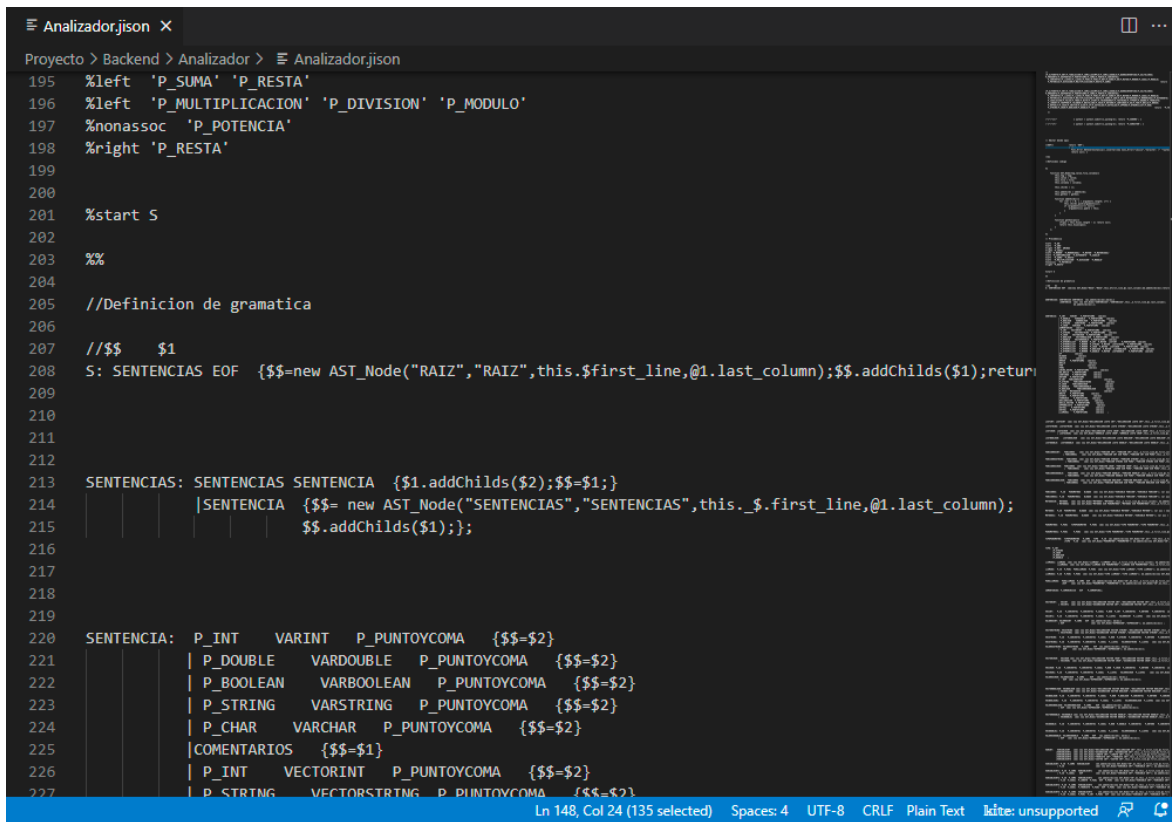
Esta es la estructura del programa. El programa fue escrito en javascript utilizando angular como apoyo para la creación del programa, json para escribir el interprete, y visual studio code como herramienta utilizada para el ordenamiento y escritura de todos los archivos.



Esta es el área de los archivos, donde en la carpeta del backend como su nombre lo indica se encuentra todo lo relacionado con el código interno. Y en la carpeta src es lo que se conoce como frontend.

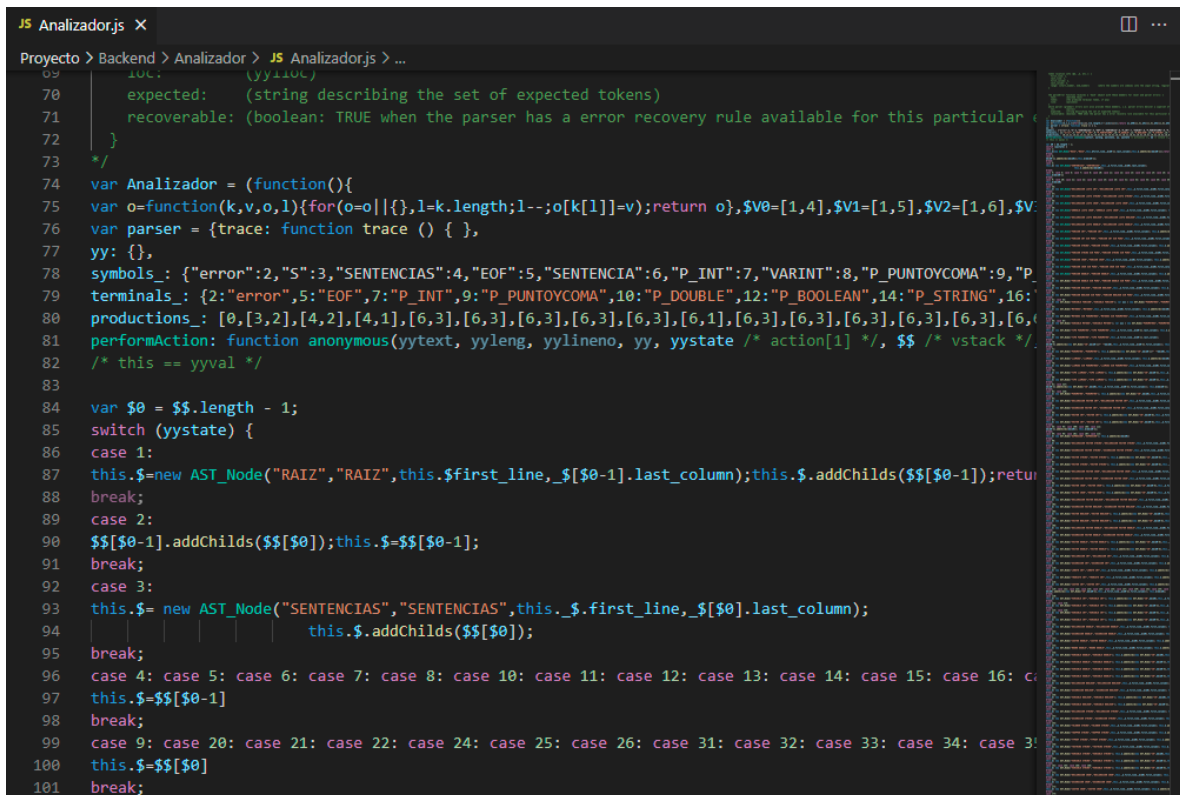


En la carpeta del backend se encuentra una subcarpeta denominada analizador donde la carpeta node\_modules son las librerías json y se encuentran los archivos del interprete json.



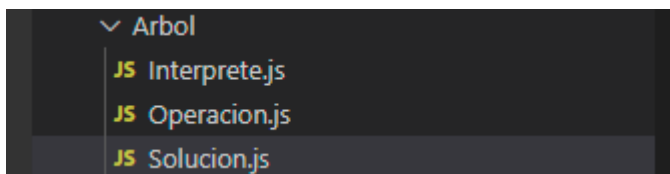
```
195 %left 'P_SUMA' 'P_RESTA'
196 %left 'P_MULTIPLICACION' 'P_DIVISION' 'P_MODULO'
197 %nonassoc 'P_POTENCIA'
198 %right 'P_RESTA'
199
200
201 %start S
202
203 %%
204
205 //Definicion de gramatica
206
207 //$$ $1
208 S: SENTENCIAS EOF {$$=new AST_Node("RAIZ","RAIZ",this.$first_line,@1.last_column);$.addChilds($1);return}
209
210
211
212
213 SENTENCIAS: SENTENCIAS SENTENCIA {$$.addChilds($2);$$=$1;}
214 | SENTENCIA {$$= new AST_Node("SENTENCIAS","SENTENCIAS",this.$first_line,@1.last_column);
215 | $.addChilds($1);};
216
217
218
219
220 SENTENCIA: P_INT VARINT P_PUNTOYCOMA {$$=$2}
221 | P_DOUBLE VARDOUBLE P_PUNTOYCOMA {$$=$2}
222 | P_BOOLEAN VARBOOLEAN P_PUNTOYCOMA {$$=$2}
223 | P_STRING VARSTRING P_PUNTOYCOMA {$$=$2}
224 | P_CHAR VARCHAR P_PUNTOYCOMA {$$=$2}
225 | COMENTARIOS {$$=$1}
226 | P_INT VECTORINT P_PUNTOYCOMA {$$=$2}
227 | P_STRING VECTORSTRING P_PUNTOYCOMA {$$=$2}
```

El archivo Analizador.json es la estructura json de todo el interprete.

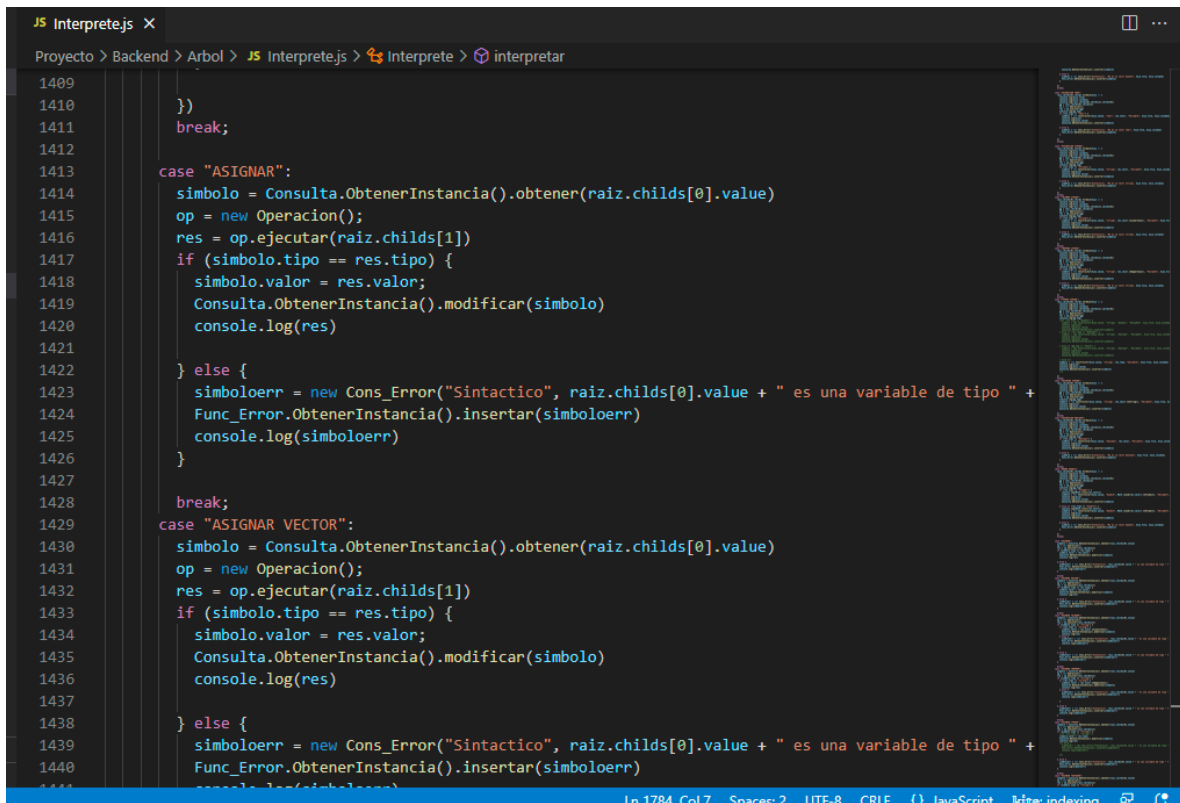


```
JS Analizador.js X
Proyecto > Backend > Analizador > JS Analizador.js > ...
69 100.1 (yy100)
70 expected: (string describing the set of expected tokens)
71 recoverable: (boolean: TRUE when the parser has a error recovery rule available for this particular
72 }
73 */
74 var Analizador = (function(){
75 var o=function(k,v,o,l){for(o=o||{},l=k.length;l-->o[k[l]]=v);return o},$V0=[1,4],$V1=[1,5],$V2=[1,6],$V
76 var parser = {trace: function trace () { },
77 yy: {}},
78 symbols_: {"error":2,"S":3,"SENTENCIAS":4,"EOF":5,"SENTENCIA":6,"P_INT":7,"VARINT":8,"P_PUNTOYCOMA":9,"P
79 terminals_: {2:"error",5:"EOF",7:"P_INT",9:"P_PUNTOYCOMA",10:"P_DOUBLE",12:"P_BOOLEAN",14:"P_STRING",16:
80 productions_: [0,[3,2],[4,2],[4,1],[6,3],[6,3],[6,3],[6,3],[6,3],[6,1],[6,3],[6,3],[6,3],[6,3],[6,3],[6,
81 performAction: function anonymous(yytext, yyleng, yylineno, yy, yystate /* action[1] */, $$ /* vstack */
82 /* this == yyval */
83
84 var $0 = $.length - 1;
85 switch (yystate) {
86 case 1:
87 this.$=new AST_Node("RAIZ","RAIZ",this.$first_line,$[$0-1].last_column);this.$.addChilds($[$0-1]);retu
88 break;
89 case 2:
90 $[$0-1].addChilds($[$0]);this.$=$[$0-1];
91 break;
92 case 3:
93 this.$= new AST_Node("SENTENCIAS","SENTENCIAS",this._$.first_line,$[$0].last_column);
94 this.$.addChilds($[$0]);
95 break;
96 case 4: case 5: case 6: case 7: case 8: case 10: case 11: case 12: case 13: case 14: case 15: case 16: c
97 this.$=$[$0-1]
98 break;
99 case 9: case 20: case 21: case 22: case 24: case 25: case 26: case 31: case 32: case 33: case 34: case 3
100 this.$=$[$0]
101 break;
```

El Archivo Analizador.js se genera automáticamente del archivo.json al correrlo en consola utilizando el comando “jison Analizador.json”

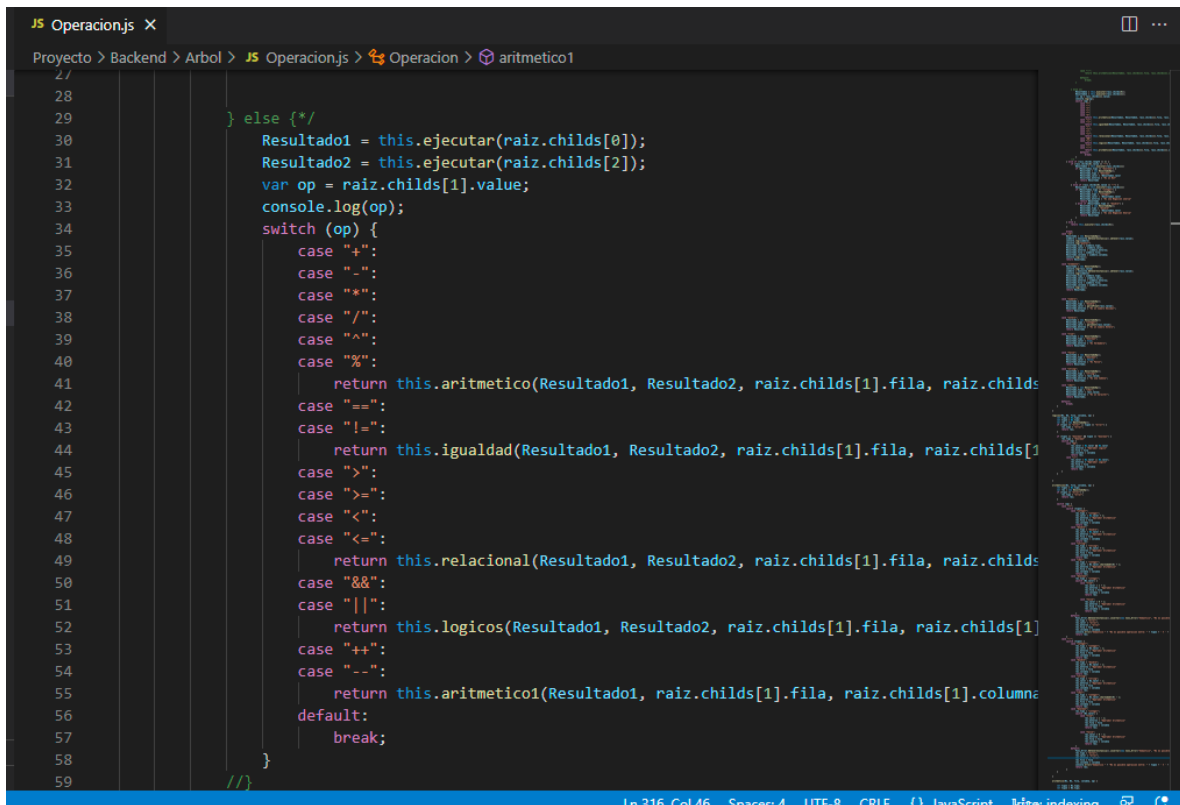


Otra subcarpeta del backend es la llamada Arbol, en esta se interpreta u opera todas las sentencias y operaciones vistas en el analizador.



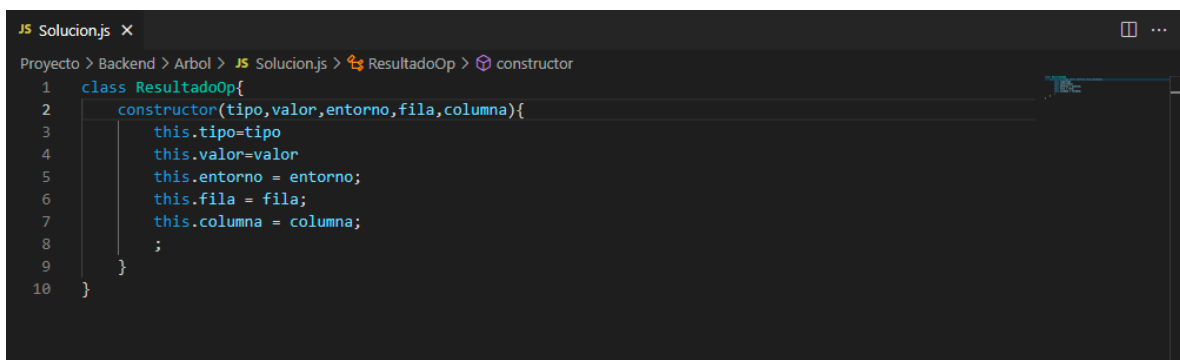
```
1409
1410
1411     break;
1412
1413     case "ASIGNAR":
1414         simbolo = Consulta.ObtenerInstancia().obtener(raiz.childs[0].value)
1415         op = new Operacion();
1416         res = op.ejecutar(raiz.childs[1])
1417         if (simbolo.tipo == res.tipo) {
1418             simbolo.valor = res.valor;
1419             Consulta.ObtenerInstancia().modificar(simbolo)
1420             console.log(res)
1421
1422         } else {
1423             simboloerr = new Cons_Error("Sintactico", raiz.childs[0].value + " es una variable de tipo " +
1424             Func_Error.ObtenerInstancia().insertar(simboloerr)
1425             console.log(simboloerr)
1426         }
1427
1428         break;
1429     case "ASIGNAR VECTOR":
1430         simbolo = Consulta.ObtenerInstancia().obtener(raiz.childs[0].value)
1431         op = new Operacion();
1432         res = op.ejecutar(raiz.childs[1])
1433         if (simbolo.tipo == res.tipo) {
1434             simbolo.valor = res.valor;
1435             Consulta.ObtenerInstancia().modificar(simbolo)
1436             console.log(res)
1437
1438         } else {
1439             simboloerr = new Cons_Error("Sintactico", raiz.childs[0].value + " es una variable de tipo " +
1440             Func_Error.ObtenerInstancia().insertar(simboloerr)
1441             console.log(simboloerr)
```

El archivo Interprete.js se utiliza para interpretar todas las sentencias utilizadas al momento de analizar código por medio del analizador.



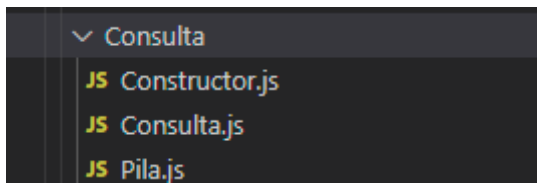
```
27
28
29 } else { /*
30     Resultado1 = this.ejecutar(raiz.childs[0]);
31     Resultado2 = this.ejecutar(raiz.childs[2]);
32     var op = raiz.childs[1].value;
33     console.log(op);
34     switch (op) {
35         case "+":
36         case "-":
37         case "*":
38         case "/":
39         case "^":
40         case "%":
41             return this.aritmetico(Resultado1, Resultado2, raiz.childs[1].fila, raiz.childs[1].columna);
42         case "=":
43         case "!=":
44             return this.igualdad(Resultado1, Resultado2, raiz.childs[1].fila, raiz.childs[1].columna);
45         case ">":
46         case ">=":
47         case "<":
48         case "<=":
49             return this.relacional(Resultado1, Resultado2, raiz.childs[1].fila, raiz.childs[1].columna);
50         case "&&":
51         case "||":
52             return this.logicos(Resultado1, Resultado2, raiz.childs[1].fila, raiz.childs[1].columna);
53         case "++":
54         case "--":
55             return this.aritmetico1(Resultado1, raiz.childs[1].fila, raiz.childs[1].columna);
56         default:
57             break;
58     }
59 }
```

El Archivo Operación.js como su nombre lo indica se encarga de operar correctamente todas las expresiones utilizadas al momento de analizarlas y prever que tengan buena estructura semántica.



```
1 class ResultadoOp{
2     constructor(tipo,valor,entorno,fila,columna){
3         this.tipo=tipo
4         this.valor=valor
5         this.entorno = entorno;
6         this.fila = fila;
7         this.columna = columna;
8     };
9 }
10 }
```

El archivo Solucion.js es un constructor temporal donde se almacenan temporalmente datos obtenidos de Operación.js esperando ser registrados en el futuro.



También se encuentra en el backend la subcarpeta de Consulta que ya se encarga de las funcionalidades principales para ser mostradas en el frontend.

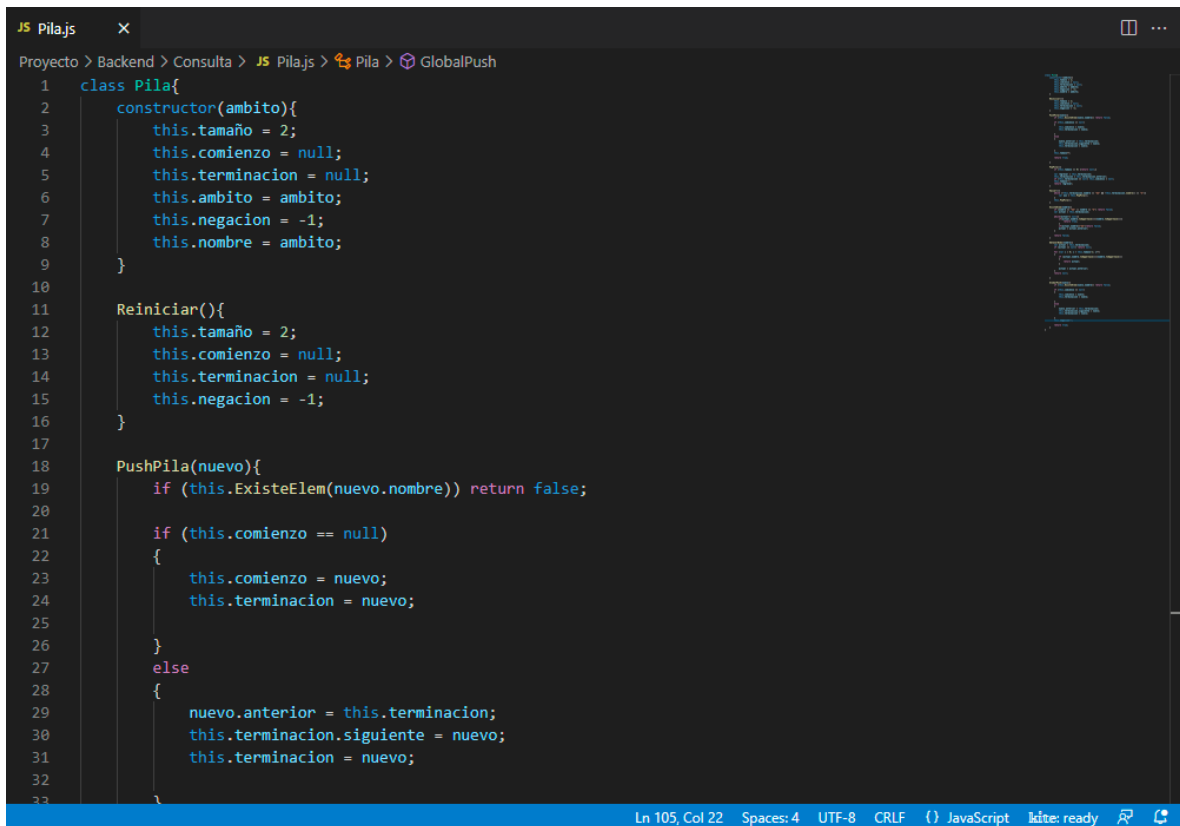
```
JS Constructor.js X
Proyecto > Backend > Consulta > JS Constructor.js > Constructor
1  class Constructor{
2      constructor(nombre, tipo, valor, entorno, fila, columna){
3          this.nombre = nombre;
4          this.tipo = tipo;
5          this.valor = valor;
6          this.entorno = entorno;
7          this.fila = fila;
8          this.columna = columna;
9      }
10 }
```

El archivo Constructor.js como indica el nombre es el constructor donde se registran los datos obtenidos en la subcarpeta Arbol y listos para ser mostrados.

```
JS Consulta.js X
Proyecto > Backend > Consulta > JS Consulta.js > Consulta > <function> > TablaSimbolo > ObtenerSimbolos > simbolos.forEach() callback
1  var Consulta = (function () {
2
3      var instancia;
4
5      class TablaSimbolo {
6          constructor() {
7              this.simbolos = [];
8          }
9
10         insertar(sim) {
11             this.simbolos.push(sim);
12         }
13
14         ObtenerSimbolos() {
15
16             var texto = " ";
17
18             texto += `<html><head><title>Tabla de Simbolos</title><style>
19             table {
20                 border-collapse: collapse;
21                 width: 100%;
22             }
23
24             th, td {
25                 text-align: left;
26                 padding: 8px;
27             }
28
29             tr:nth-child(even){background-color: #C4F3E5}
30
31             th {
32                 background-color: #D3F3EE;
33                 color: white;
```

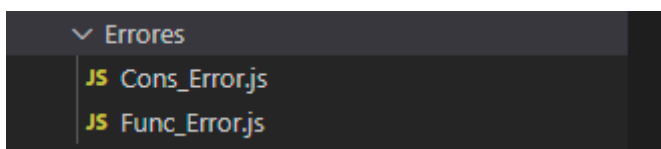
El archivo Consulta.js es donde se realizan todas las instrucciones sobre que acciones realizar con los datos obtenidos por medio del analizador, como la generación de la tabla de símbolos.





```
1 class Pila{
2   constructor(ambito){
3     this.tamaño = 2;
4     this.comienzo = null;
5     this.terminacion = null;
6     this.ambito = ambito;
7     this.negacion = -1;
8     this.nombre = ambito;
9   }
10
11   Reiniciar(){
12     this.tamaño = 2;
13     this.comienzo = null;
14     this.terminacion = null;
15     this.negacion = -1;
16   }
17
18   PushPila(nuevo){
19     if (this.ExisteElem(nuevo.nombre)) return false;
20
21     if (this.comienzo == null)
22     {
23       this.comienzo = nuevo;
24       this.terminacion = nuevo;
25     }
26     else
27     {
28       nuevo.anterior = this.terminacion;
29       this.terminacion.siguiente = nuevo;
30       this.terminacion = nuevo;
31     }
32   }
33 }
```

El archivo Pila.js es una configuración mas detallada sobre las acciones de los datos obtenidos, como agregar, modificar o quitar ciertos nodos al momento de analizar



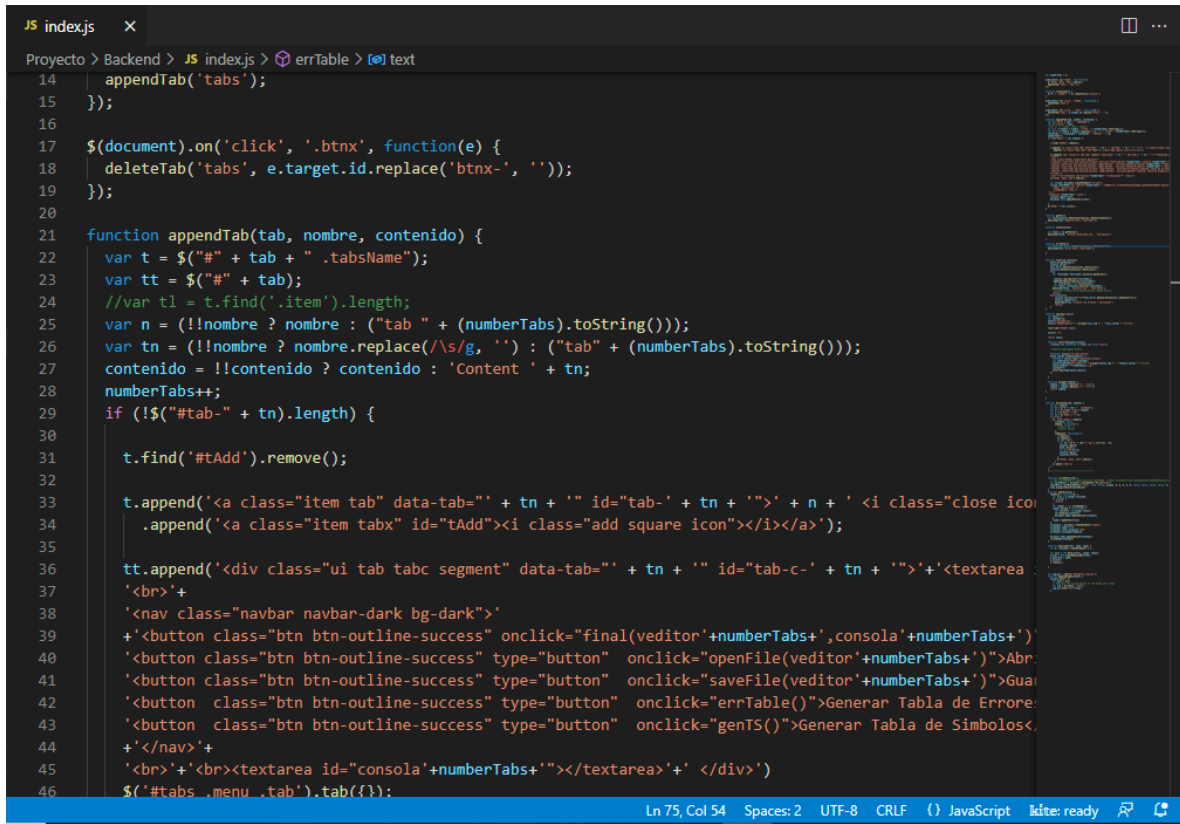
La ultima subcarpeta vista en el backend es la de los Errores y como se menciona en el nombre se encarga de registrar todos los errores encontrados al ejecutar.

```
JS Cons_Error.js X
Proyecto > Backend > Errores > JS Cons_Error.js > Cons_Error
1 class Cons_Error{
2     constructor(tipo, descripcion, fila, columna){
3         this.tipo = tipo;
4         this.descripcion = descripcion;
5         this.fila = fila;
6         this.columna = columna;
7         this.siguiente = null;
8         this.anterior = null;
9     }
10 }
```

El archivo Cons\_Error.js es el constructor encargado de registrar todos los errores tanto léxico como sintácticos y pasando por los semánticos.

```
JS Func_Error.js X
Proyecto > Backend > Errores > JS Func_Error.js > Func_Error > <function> > Listado > insertar
1 var Func_Error = (function () {
2     var instancia;
3
4     class Listado {
5         constructor() {
6             this.errorSec = [];
7             this.inicio = null;
8             this.final = null;
9         }
10
11         insertar(Error) {
12             this.errorSec.push(Error)
13
14             if (this.inicio == null) {
15                 this.inicio = Error;
16                 this.final = Error;
17                 return;
18             }
19
20             this.final.siguiente = Error;
21             Error.anterior = this.final;
22             this.final = Error;
23             console.log(this.final);
24         }
25
26
27         ObtenerError() {
28             //var texto1 = "";
29
30             var prueba = this.inicio;
31
32             var texto = " ";
33 }
```

El archivo Func\_Error.js se encarga de ejecutar todas las acciones requeridas para el manejo de errores como insertarlos o generar la tabla de errores.

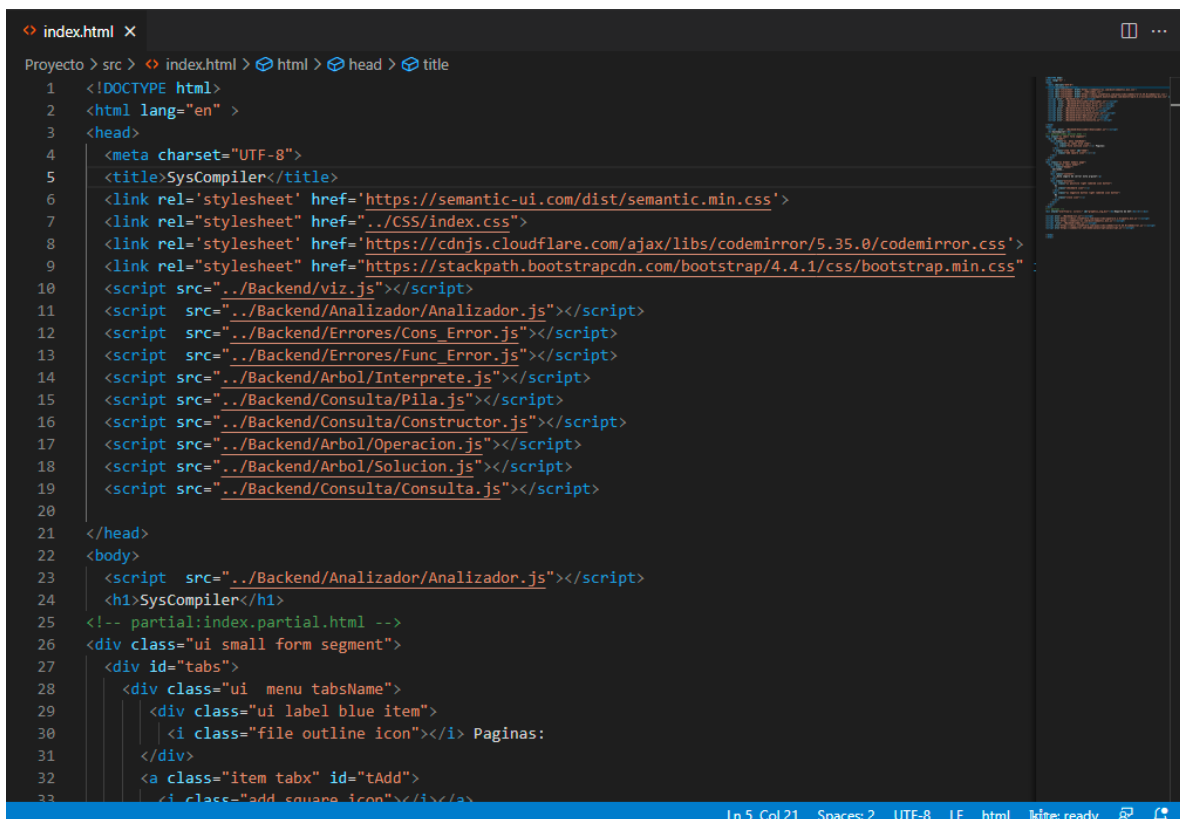


```
14 appendTab('tabs');
15 });
16
17 $(document).on('click', '.btnx', function(e) {
18   deleteTab('tabs', e.target.id.replace('btnx-', ''));
19 });
20
21 function appendTab(tab, nombre, contenido) {
22   var t = $("# " + tab + " .tabsName");
23   var tt = $("# " + tab);
24   //var tl = t.find('.item').length;
25   var n = (!!nombre ? nombre : ("tab " + (numberTabs).toString()));
26   var tn = (!!nombre ? nombre.replace(/\s/g, '') : ("tab" + (numberTabs).toString()));
27   contenido = !!contenido ? contenido : 'Content ' + tn;
28   numberTabs++;
29   if (!$("#tab-" + tn).length) {
30
31     t.find('#tAdd').remove();
32
33     t.append('<a class="item tab" data-tab="' + tn + '" id="tab-' + tn + '">' + n + ' <i class="close icon'
34       .append('<a class="item tabx" id="tAdd"><i class="add square icon"></i></a>');
35
36     tt.append('<div class="ui tab tabc segment" data-tab="' + tn + '" id="tab-c-' + tn + '">' + <textarea :
37       '<br>'+
38       '<nav class="navbar navbar-dark bg-dark">
39       +<button class="btn btn-outline-success" onclick="final(veditor'+numberTabs+',consola'+numberTabs+')"
40       +<button class="btn btn-outline-success" type="button" onclick="openFile(veditor'+numberTabs+')">Abr:
41       +<button class="btn btn-outline-success" type="button" onclick="saveFile(veditor'+numberTabs+')">Gua:
42       +<button class="btn btn-outline-success" type="button" onclick="errTable()">Generar Tabla de Errore:
43       +<button class="btn btn-outline-success" type="button" onclick="genTS()">Generar Tabla de Simbolos<
44       +</nav>'+
45       '<br>'+<br><textarea id="consola'+numberTabs+'"></textarea>' + </div>')
46     $('#tabs .menu .tab').tab({});
```

Un archivo almacenado en el backend es el index.js. Este archivo ya esta conectado directamente con el frontend, en se encuentran todas las funcionalidades que ve el usuario como la creación del arbol ast.



Ya para la cuestión del frontend encontramos diversos archivos y carpetas pero nosotros solo nos vamos a enfocar en index.html



```
index.html X
Proyecto > src > index.html > html > head > title
1 <!DOCTYPE html>
2 <html lang="en" >
3 <head>
4   <meta charset="UTF-8">
5   <title>SysCompiler</title>
6   <link rel="stylesheet" href="https://semantic-ui.com/dist/semantic.min.css">
7   <link rel="stylesheet" href=" ../CSS/index.css">
8   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/codemirror/5.35.0/codemirror.css">
9   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
10  <script src=" ../Backend/viz.js"></script>
11  <script src=" ../Backend/Analizador/Analizador.js"></script>
12  <script src=" ../Backend/Errores/Cons_Error.js"></script>
13  <script src=" ../Backend/Errores/Func_Error.js"></script>
14  <script src=" ../Backend/Arbol/Interprete.js"></script>
15  <script src=" ../Backend/Consulta/Pila.js"></script>
16  <script src=" ../Backend/Consulta/Constructor.js"></script>
17  <script src=" ../Backend/Arbol/Operacion.js"></script>
18  <script src=" ../Backend/Arbol/Solucion.js"></script>
19  <script src=" ../Backend/Consulta/Consulta.js"></script>
20
21 </head>
22 <body>
23   <script src=" ../Backend/Analizador/Analizador.js"></script>
24   <h1>SysCompiler</h1>
25   <!-- partial:index.partial.html -->
26   <div class="ui small form segment">
27     <div id="tabs">
28       <div class="ui menu tabsName">
29         <div class="ui label blue item">
30           <i class="file outline icon"></i> Paginas:
31         </div>
32         <a class="item tabx" id="tAdd">
33           <i class="add square icon"></i></a>
```

En index.html es la visualización básica del frontend que vera el usuario, además de enlaces que llevan directamente al backend; en especifico a index.js que es allí donde se encuentran todas sus funcionalidades