6-11-2021

Manual Técnico

SYSCOMPILER – Proyecto 2

EDUARDO RENÉ AGUSTIN MENDOZA
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1 - N

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

Contenido

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

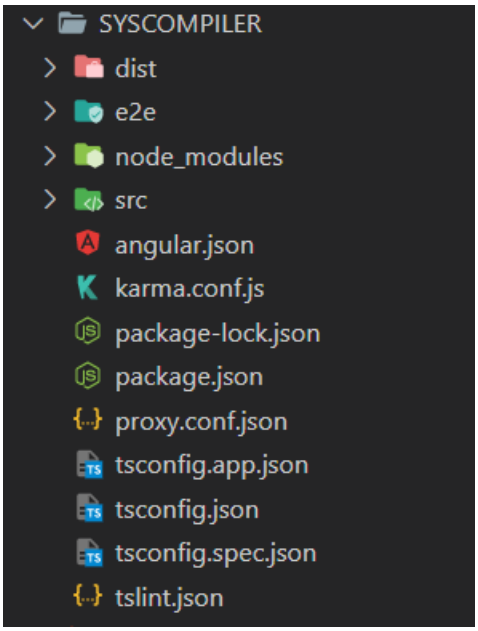
Requerimientos del Sistema

- Sistema Operativo requerido: Windows 8 o superiores.
- Memoria RAM: 4 GB o superiores.
- Procesador: Intel Celeron o superiores (Recomendado iCore i5 de 9na Generación en adelante)
- Tipo de sistema: 64bits
- Lenguaje de Programación: JavaScript/Typescript
- Librería de Análisis Léxico y Sintáctico: Jison
- Requerimientos de ejecución:
  - Visual Studio Code (Editor de código)
  - Angular CLI Versión 12.2.8
  - Node js Versión 14.18.0
  - Package Manager NPM Versión 6.14.8
  - Express

Clases Utilizadas.

-Para el Front End

Utilizamos los componentes propios de Angular en donde tenemos lo siguiente:



Esta carpera la generamos al crear un nuevo proyecto en angular con el comando ng new <Nombre del proyecto>, al tener esto nos genera la carpeta de src donde tendremos los componentes de la app, ahí dentro tendremos los componentes de la pagina (.html), el diseño (.css), la lógica(.ts), los módulos(.ts) y el service(.ts)

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

app.component.html

En este apartado tenemos la página principal en donde están etiquetas de los botones, los editores los cuales se utilizó la librería de ngx-monaco-editor propia de angular para su llamada se utilizó el comando options, también se utilizaron los ngModel para obtener los campos requeridos y utilizarlos en la lógica así mismo la función (click) para llamar a los métodos.

app.component.css

En este tenemos los estilos de cada una de las etiquetas utilizadas en la página

app.component.ts

En este archivo tendremos todos los métodos para realizar los envíos al back end y la configuración de algunos botones para visualizar, los cuales fueron:

- EditorOptions y ConsoleOptions
  Aquí se configura los editores.

```
EditorOptions = {
  theme: "vs-dark",
  automaticLayout: true,
  scrollBeyondLastLine: false,
  fontSize: 16,
  minimap: {
    enabled: true
  },
  language: 'java'
}

ConsoleOptions = {
  theme: "vs-dark",
  readOnly: true,
  automaticLayout: true,
  scrollBeyondLastLine: false,
  fontSize: 16,
  minimap: {
    enabled: true
  },
  language: 'markdown'
}
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- newWindow()
  Este solo abre una nueva pestaña en el navegador.

- closeWindow()
  Este cierra la pestaña en donde estemos ubicados.

- Compilar()
  En este se manda lo que se encuentre en la entrada por medio de una petición post al servidor para poder analizarlo, si se tiene una respuesta se mostrara en consola lo que se espera y en las tablas los tokens como los lexemas

```
Compilar() {
  if (this.entrada != "") {
    const x = { "input": this.entrada }
    this.appService.compile(x).subscribe(
      data => {
        console.log('Procesando Datos');
        this.salida = data.output;
        this.simbolos = data.arreglo_simbolos;
        this.errores = data.arreglo_errores;
      },
      error => {
        console.log('Hubo un error :(', error);
        this.simbolos = [];
        this.errores = [];
        if (error.error) {
          if (error.error.output)
            this.salida = error.error.output;
          else if (error.error.message)
            this.salida = error.error.message;
          else
            this.salida = error.error;
        }
        else {
          this.salida = "Error en la entrada.\nIngrese otra entrada.";
        }
      }
    );
  } else
    this.salida = "Entrada vacía. \n Ingresa una nueva entrada.";
}
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- DrawAST()
  De la misma manera que en el compile, aquí mandamos una petición post para el analizar la tabla de símbolos, ya que de esta parte es donde se toman los tokens para poder realizarlo.

```
DrawAST() {
  this.simbolos = [];
  this.errores = [];
  if (this.entrada != "") {
    const x = { "input": this.entrada }
    this.appService.getAST(x).subscribe(
      data => {
        saveAs(data, "AST");
        this.salida = "Generando AST y Descargandolo... :D!";
        console.log('AST recibido');
      },
      error => {
        console.log('Error en la generacion del AST :(', error);
        this.salida = "Ocurrió un error al analizar la entrada.\nNo se generó el AST."
      }
    );
  } else
    alert("Entrada vacía. No se puede generar el AST.");
}
```

- GuardarArchivo()
  En este se genera un archivo ya sea en blanco o toma lo que esta en el apartado de entrada y lo guarda en un archivo con extensión .cs

```
GuardarArchivo() {
  var f = document.createElement('a');
  f.setAttribute('href', 'data:text/plain;charset=utf-8,' + encodeURIComponent(this.entrada));
  f.setAttribute('download', this.fname ? this.fname.replace("C:\\fakepath\\", "") : 'Code.sc');
  if (document.createEvent) {
    var event = document.createEvent('MouseEvents');
    event.initEvent('click', true, true);
    f.dispatchEvent(event);
  }
  else {
    f.click();
  }
  console.log('Archivo Guardado!');
}
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- openDialog() y readerFile()
  En openDialog vamos a tomar el id de la etiqueta en donde tenemos nuestro input de tipo file, esto para que en el readerFile podamos extraer la información del archivo que escojamos y lo podamos mostrar en nuestra entrada, tener en cuenta que solo aceptara archivos con extensión .sc

```
// Abre el cuadro de dialogo
openDialog() {
  document.getElementById("fileInput")!.click();
}

// Lee el archivo
readFile(event: any) {
  let input = event.target;
  let reader = new FileReader();
  reader.onload = () => {
    var text = reader.result;
    if (text) {
      this.entrada = text.toString();
    }
  }
  reader.readAsText(input.files[0]);
  this.salida = '';
  console.log('File opened!')
}
```

app.service.ts

En este archivo, vamos a construir e inyectar el servidor esto para poder mandar las peticiones a nuestro back end, creamos una variable para poder tener nuestra ruta hacia el servidor y en el constructor inyectamos los servicios http para las respuestas

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class AppService {

  _url = "http://localhost:3080";

  constructor(private http: HttpClient) { }
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- Compile
  Siendo una petición POST enviamos un parámetros el cual será nuestra entrada esto para poder analizarla.

```
compile(input: any) {
  return this.http.post<any>(this._url + '/compile', input);
}
```

- getAST
  De la misma manera que el compile, será un POST y enviaremos la tabla de símbolos para poder procesarla y generar el AST y poderlo visualizar.

```
getAST(input: any) {
  return this.http.post(this._url + '/AST_report', input, {
    responseType: 'blob',
  });
}
```

app.module.ts

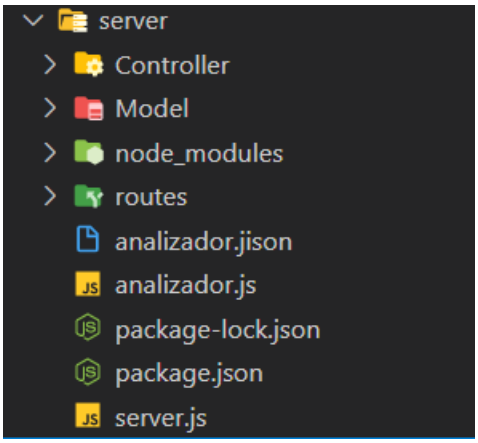
En este archivo tendremos las importaciones necesarias de los módulos que se utilizaran

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { AppService } from './app.service';
import { HttpClientModule } from '@angular/common/http';
import { MonacoEditorModule, MONACO_PATH } from '@materia-ui/ngx-monaco-editor';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    HttpClientModule,
    MonacoEditorModule
  ],
  providers: [
    AppService,
    {
      provide: MONACO_PATH,
      useValue: 'https://unpkg.com/monaco-editor@0.19.3/min/vs'
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- Para el Back End

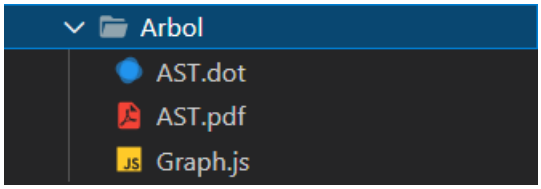
Esta carpeta tendrá tanto la lógica como las conexiones y también el analizador Jison.



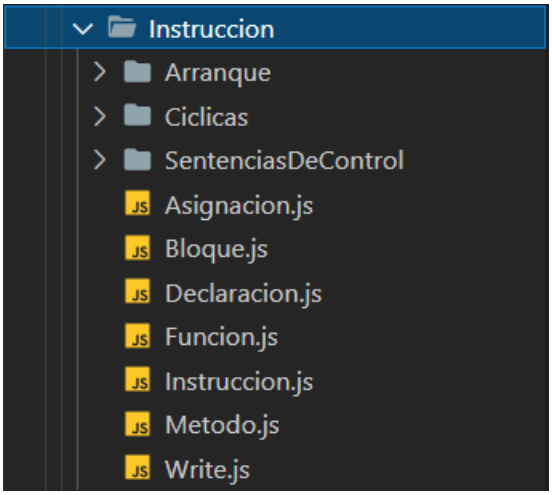
Dentro de la carpeta tenemos los controladores, las clases enumeradas, las clases que darán sentido a las instrucciones, las rutas para la conexión así mismo el analizador y el server

Controller.

- **Árbol.**
  Se encuentra el archivo donde iremos generando el árbol nodo por nodo, dentro de encontraremos los métodos para cada una de las instrucciones del lenguaje.



- **Instrucción.**
  Dentro de esta carpeta se encuentran las acciones que se deben de realizar, así mismo las carpetas de Arranque, Cíclicas y Sentencias de control.



- Arranque: Tiene la sentencia de Start With que dará como inicio al lenguaje.
- Ciclas: Tiene todas las sentencias que generan bucles y se iteran n veces (While, Do While, For).
- De Control: Tiene todas las sentencias que controlan un flujo de datos (If, If-Else, If-Else If , Switch - Case).

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

- **Principales(Clase Enumerada).**
  En esta carpeta se encuentran las clases enumeradas las cuales se heredarán en la mayoría de los archivos ya que estos tendrán los tipos de operaciones, de valores, de datos y los tipos primitivos.

```
const TIPO_INSTRUCCION = {
    PRINT: 'IMPRIMIR',
    DECLARACION: 'DECLARACION',
    ASIGNACION: 'ASIGNACION',
    WHILE: 'WHILE',
    FOR: 'FOR',
    DOWHILE: 'DO-WHILE',
    IF: 'IF',
    IF_ELSE: 'IF-ELSE',
    ELSE_IF: 'ELSE-IF',
    CASO: 'CASO-SWITCH',
    SWITCH: 'SWITCH',
    TERNARIO: 'OPERADOR-TERNARIO',
    CASTEO: 'CASTEO',
    ACCESO: 'ACCESO-EDD',
    BREAK: 'TRANSFERENCIA-BREAK',
    CONTINUE: 'TRANSFERENCIA-CONTINUE',
    TO_LOWER: 'TO-LOWER',
    TO_UPPER: 'TO-UPPER',
    LENGTH: 'LENGTH',
    TRUNCATE: 'TRUNCATE',
    ROUND: 'ROUND',
    TYPEOF: 'TYPEOF',
    TOSTRING: 'TOSTRING',
    TOCHARLIST: 'TOCHARARRAY',
    NUEVO_METODO: 'METODO',
    NUEVA_FUNCION: 'FUNCION',
    LLAMADA: 'LLAMADA-METODO/FUNCION',
    START_WITH: 'START-WITH',
    RETURN: 'RETURN'
}

module.exports = TIPO_INSTRUCCION
```

```
const TIPO_DATO = {
    ENTERO: 'ENTERO',
    DOBLE: 'DOBLE',
    BOOLEANO: 'BOOLEANO',
    CARACTER: 'CARACTER',
    CADENA: 'CADENA',
    VECTOR: 'VECTOR',
    LISTA: 'LISTA'
}

module.exports = TIPO_DATO
```

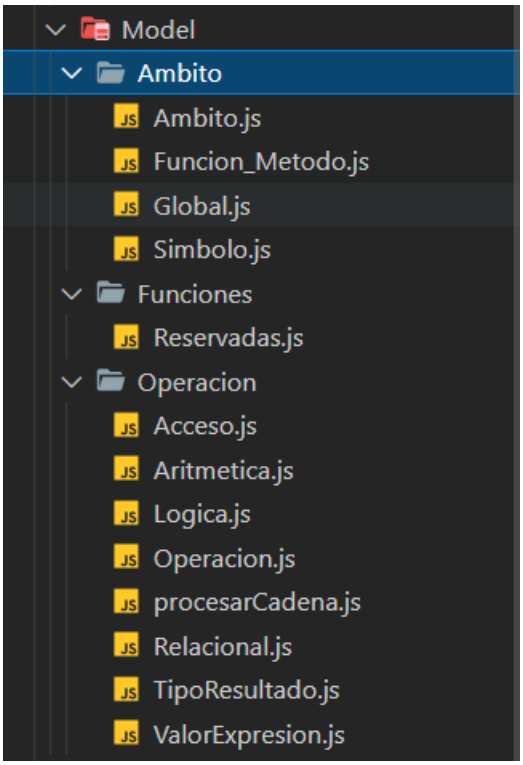
```
const TIPO_VALOR = {
    ENTERO: 'VAL_ENTERO',
    DOBLE: 'VAL_DOBLE',
    CADENA: 'VAL_CADENA',
    CARACTER: 'VAL_CARACTER',
    BOOLEANO: 'VAL_BOOLEANO',
    IDENTIFICADOR: 'VAL_IDENTIFICADOR'
}

module.exports = TIPO_VALOR
```

```
const TIPO_OPERACION = {
    SUMA: 'SUMA',
    RESTA: 'RESTA',
    MULTIPLICACION: 'MULTIPLICACION',
    DIVISION: 'DIVISION',
    POTENCIA: 'POTENCIA',
    MODULO: 'MODULO',
    NEGACION: 'NEGACION',
    IGUALIGUAL: 'IGUALIGUAL',
    DIFERENTE: 'DIFERENTE',
    MENOR: 'MENOR',
    MENORIGUAL: 'MENORIGUAL',
    MAYOR: 'MAYOR',
    MAYORIGUAL: 'MAYORIGUAL',
    OR: 'OR',
    AND: 'AND',
    NOT: 'NOT',
    MAX: 2000
}
module.exports = TIPO_OPERACION;
```

Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

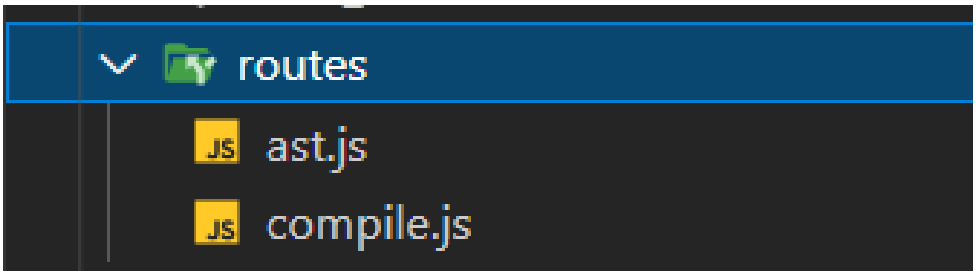
Model.

Dentro de esta carpeta encontraremos las carpetas de las palabras reservadas, los ámbitos y las operaciones



Routes.

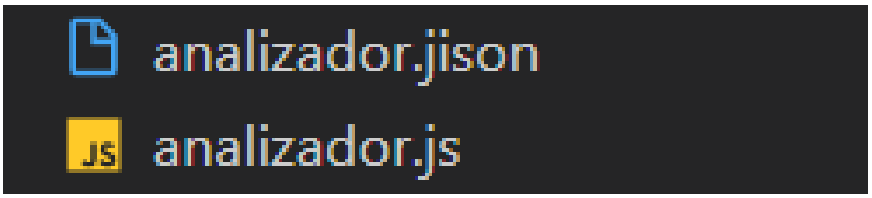
Este archivo contendrá dos archivos que conectaran al server, estas funcionaran como el cliente ya que de aquí serán las llamadas hacia las demás clases.



Soporte Técnico: rene.agus67@gmail.com
Tel. 41194827

Analizador.jison

Si hablamos de Jison veremos que es un lenguaje que facilita la creación de un interprete en JavaScript, ya que en este se vera una sintaxis a un léxico y sintáctico ya que Jison traduce nuestros tokens, expresiones y producciones a js, el comando para compilar es Jison analizador.jison esto por si se hacen modificaciones en las producciones o tokens



Server.js

Este archivo es él comunicador entre el back y front ya que aquí se encuentran los endpoints donde se harán las peticiones.

```
const express = require('express');
const app = express(),
  bodyParser = require("body-parser");
port = 3080;
let cors = require('cors');

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

const parser = require('./analizador')
require('./routes/compile.js')(parser, app)
require('./routes/ast.js')(parser, app)

app.get('/', (req, res) => {
  res.send('Hello from server!');
});

app.listen(port, () => {
  console.log(`Server listening on the port: ${port}`);
});
```