



6-11-2021

Manual de Usuario

SYSCOMPILER – Proyecto 2

EDUARDO RENÉ AGUSTIN MENDOZA - 201801627
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 1 - N

Contenido

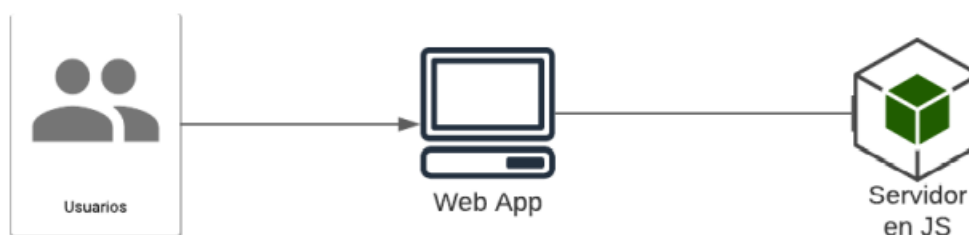
Requerimientos para la aplicación.	2
Arquitectura del proyecto.	2
Interfaz de la Aplicación.	3
Sintaxis del lenguaje.	4
Declaración de una variable.....	4
Funciones	4
Métodos.	5
Llamadas	5
Función WriteLine	6
Start With	6
Casteos.....	6
Incrementos y Decrementos.	6
Estructuras de datos.	6
□ Vector.....	6
□ Listas Dinámicas.	7
Palabra reservada.....	8
Descripción	8
Sentencias de Control.....	9
Sentencias Cíclicas.....	11
Sentencias de Transferencias.....	12
Funciones Nativas	13
Tabla de tipos	14
Tabla de Operaciones Relacionales.	15
Tabla de Operadores Lógicos.....	16
Montar el servidor y la Web App	16
Anexos	24

Requerimientos para la aplicación.

- Sistema Operativo requerido: Windows 8 o superiores.
- Memoria RAM: 4GB o superiores.
- Requerimientos de Ejecución: Tener instalado Visual Studio Code (Recomendado), Angular, Node.js y Express.
- Procesador: Intel Celeron o Superiores (Recomendables versiones de 9na Generación en adelante).
- Tipo de Sistema: Recomendado 64bits.

Arquitectura del proyecto.

Arquitectura Cliente-Servidor

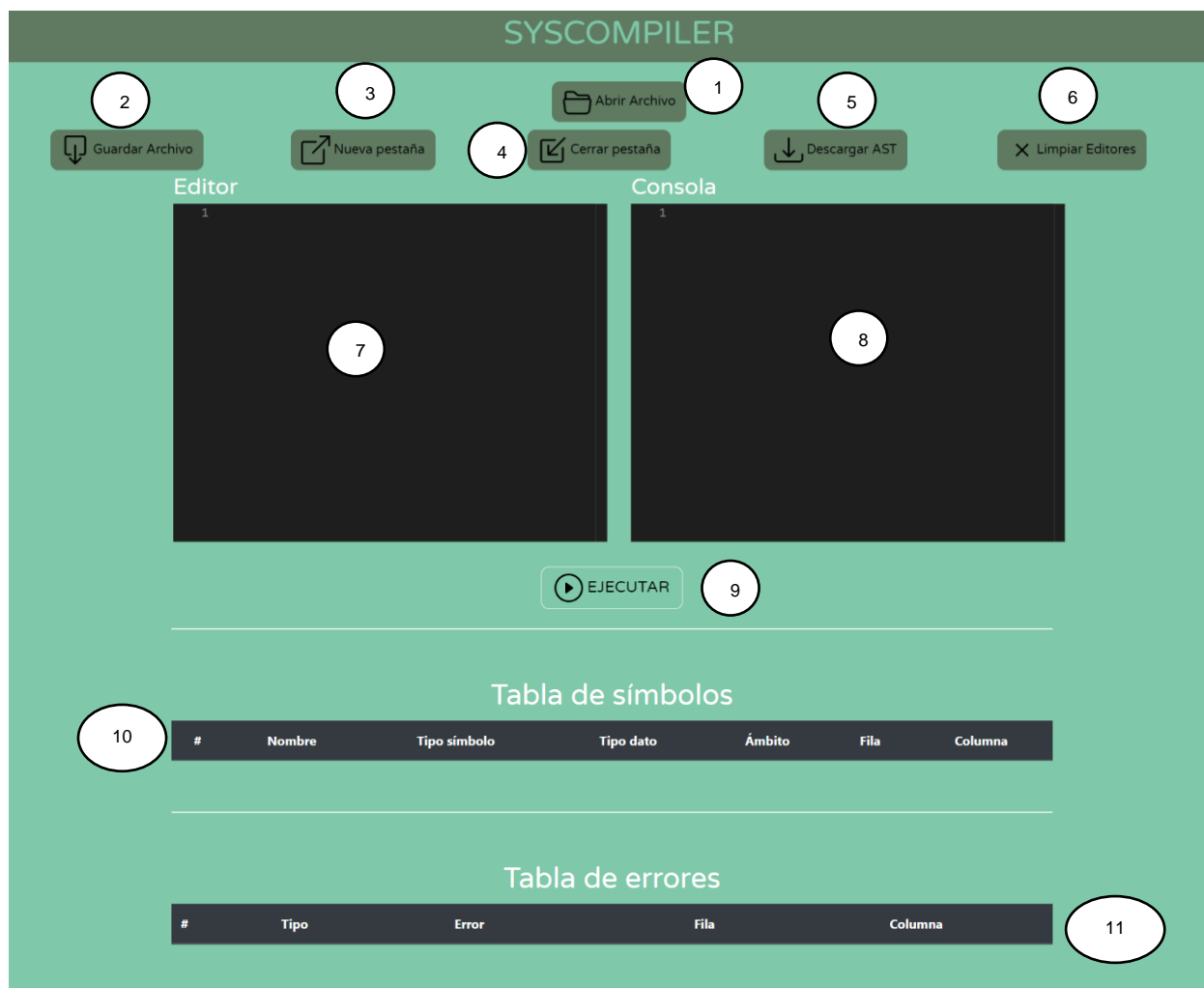


Dentro de nuestra aplicación trabajamos, para la parte del servies (Back-end) decidimos utilizar Express js y para la parte de la web app/cliente (Front-end) utilizamos el framework de Angular, el cual dentro del proyecto se separo por dos carpetas donde:

```
> server
> SYSCOMPILER
> .gitignore
> package-lock.json
> README.md
```

Syscompiler es la parte del Front-End, mientras que dentro de la carpeta de sever, además de encontrar la lógica del programa, se encuentra la conexión hacía la Web App

Interfaz de la Aplicación.



1. **Abrir Archivo:** Se escogerán los archivos con extensión .cs
2. **Guardar Archivo:** Se podrá guardar lo que se tenga escrito en el apartado de editor o un archivo en balnco.
3. **Nueva Pestaña:** Abre una nueva ventana en el explorador.
4. **Cerrar Pestaña:** Se cerrará la pestaña en la cual nos encontremos.
5. **Descargar AST:** Se obtendrá un PDF/JPG del árbol sintáctico generado por el lenguaje analizado
6. **Limpiar Editores:** Borrara todo lo que se encuentre en consola y editor
7. **Editor:** Es el área en donde se puede colocar el lenguaje.
8. **Consola:** Mostrara el resultado de las instrucciones colocadas en el editor.
9. **Ejecutar:** Tomara toda la sintaxis del lenguaje colocado en el editor y lo analizara de manera que se la respuesta se mostrara en las tablas y consola
10. **Tabla de Símbolos:** Mostrara la tabla de todos los tokens y lexemas analizados
11. **Tabla de Errores:** Mostrara si se encuentra algún error en la sintaxis analizada.

Sintaxis del lenguaje.

La sintaxis del lenguaje será parecida a la de Java con ciertas modificaciones, hay que tener en claro que al ser parecida la sintaxis debemos de recordar que al finalizar cada una de las palabras, numero y/o signos de agrupación debemos de colocar “;” (punto y coma) y también debemos de recordar que nuestro lenguaje tiene el encapsulamiento de sentencias por lo tanto las instrucciones se deben de colocar dentro de “{” “}” al igual que las instrucciones de un método o función por ultimo recordamos al usuario que el lenguaje es case-insensitive esto quiere decir que no toma las mayúsculas y minúsculas.

Declaración de una variable.

Para la declaración de variables se contará de 4 formas diferentes las cuales se conformará de un tipo, identificador y punto y coma, en algunos casos de una expresión:

1. <TIPO> (ver tabla 1 sección de tipos) IDENTIFICADOR (Nombre de la variable) punto y coma.
2. <TIPO> IDENTIFICADOR, IDENTIFICADOR,;
3. <TIPO> IDENTIFICADOR = <EXPRESION>(Donde la expresión puede ser una cadena, un booleano, un entero, un decimal o algún otro identificador)
4. <TIPO> IDENTIFICADOR, IDENTIFICADOR, = <EXPRESION>;

Funciones

Una función es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros. Para este lenguaje las funciones serán declaradas definiendo primero su tipo, luego un identificador para la función, seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no utilice parámetros).

Cabe a destacar que no habrá sobrecarga de funciones y métodos dentro de este lenguaje por lo que solo puede existir una función o método con el id declarado por lo que si se crea otra función o método con un id previamente utilizado esto debe de generar un error de tipo semántico.

```
<TIPO> IDENTIFICADOR (PARAMETROS){  
    INSTRUCCIONES  
}
```

PARAMETROS = <TIPO> ID, <TIPO> ID ...

Métodos.

Un método también es una subrutina de código que se identifica con un nombre, tipo y un conjunto de parámetros, aunque a diferencia de las funciones estas subrutinas no deben de retornar un valor. Para este lenguaje los métodos serán declarados haciendo uso de la palabra reservada 'void' al inicio, luego se indicará el identificador del método, seguido de una lista de parámetros dentro de paréntesis (esta lista de parámetros puede estar vacía en el caso de que la función no utilice parámetros).

Cada parámetro debe estar compuesto por su tipo seguido de un identificador, para el caso de que sean varios parámetros se debe utilizar comas para separar cada parámetro y en el caso de que no se usen parámetros no se deberá incluir nada dentro de los paréntesis. Luego de definir el método y sus parámetros se declara el cuerpo del método, el cual es un conjunto de instrucciones delimitadas por llaves {}

```
void IDENTIFICADOR (PARAMETROS){  
    INSTRUCCIONES  
}
```

PARAMETROS = <TIPO> ID, <TIPO> ID ...

Llamadas

La llamada a una función específica la relación entre los parámetros reales y los formales y ejecuta la función. Los parámetros se asocian normalmente por posición, aunque, opcionalmente, también se pueden asociar por nombre. Si la función tiene parámetros formales por omisión, no es necesario asociarles un parámetro real.

La llamada a una función devuelve un resultado que ha de ser recogido, bien asignándolo a una variable del tipo adecuado, bien integrándolo en una expresión.

La sintaxis de las llamadas de los métodos y funciones serán la misma

IDENTIFICADOR (PARAMETROS);

PARAMETROS = <TIPO> ID, <TIPO> ID ...

Función WriteLine

Esta función nos permite imprimir expresiones con valores únicamente de tipo entero, doble, booleano, cadena y carácter.

WRITELINE (EXPRESION);

Start With

Para poder ejecutar todo el código generado dentro del lenguaje, se utilizará la sentencia START WITH para indicar que método o función es la que iniciará con la lógica del programa.

START WITH IDENTIFICADOR ;

START WITH IDENTIFICADOR (PARAMETROS);

Casteos

Se podrán realizar casteos dentro del lenguaje ya que en algunas ocasiones necesitaremos utilizar varios tipos de datos o poder hacer operaciones entre estos, para lo cual constara de (<TIPO>) seguido de <EXPRESION> y punto y coma.

El lenguaje aceptará los siguientes casteos:

- **Int a double**
- **Double a Int**
- **Int a String**
- **Int a Char**
- **Double a String**
- **Char a int**
- **Char a double**

Incrementos y Decrementos.

Estos nos ayudaran a realizar sumas o restas continuas de un valor de uno en uno, para lo cual se podrá hacer dicha acción de la siguiente manera:

<EXPRESION> + +; Esto aumentara en uno un entero o decimal.

<EXPRESION> - -; Esto disminuirá en uno un entero o decimal.

Estructuras de datos.

– Vector.

Este tipo de estructura es estática ya que tendrá un tamaño definido, se podrán usar solo de un solo tipo y el lenguaje solo permitirá vectores de una sola dimensión así mismo recordemos que la posición de un vector esta dada de la

siguiente manera **N-1** es decir que si queremos acceder a la posición 1 sería de la siguiente manera `posVector[0]` con esto tendríamos el primer valor, al igual tendremos 2 maneras de poder declararlo

1. Declaración de tipo 1.

En esta se podrá declarar el vector de una manera en donde se le coloca explícitamente la dimensión de su tamaño de la siguiente manera,

`<TIPO> IDENTIFICADOR [] = new <TIPO> [ENTERO];`

`—→ int vecotr1 [] = new int [4];`

2. Declaración de tipo 2.

En este se podrá declara el vector con los datos ingresados directamente, así mismo es como se le dará su tamaño:

`<TIPO> IDENTIFICADOR [] = {LISTA DE VALORES};`

`—→ int vecotr2 [] = {12, 15, 35, 44};`

`—→ string vecotr3 [] = {"H", "O", "L", "A"};`

Se puede acceder a un vector de la siguiente manera:

`IDENTIFICADOR [POSICION VECTOR];`

`string posV = vector3[0];`

Así mismo se puede modificar una posición del vector de la siguiente manera:

`vector3[0] = "Hola"`

`vector3[1] = "Mundo"`

Al imprimir el vector tendríamos la siguiente salida en consola, utilizando `writeline`:

`Writeline(vector3);`

En consola veríamos ("Hola", "Mundo", "L", "A")

— Listas Dinámicas.

Este tipo de estructura no crece de una manera fija si no que de una manera iterativa y puede almacenar N elementos de un solo tipo, su manera de declarar es de la siguiente manera:

`DynamicList <TIPO> IDENTIFICADOR = new DynamicList <TIPO>;`

`DynamicList <int> lista2 = new DynamicList <int>;`

Dentro del lenguaje, existen palabras nativas (Reservadas) las cuales son obligatorias en la estructura de este para su funcionamiento correcto en la siguiente tabla mostraremos cuales son las palabras reservadas las cuales se deben de usar, así mismo como las sentencias de control y cíclicas, como poder imprimir, con que palabra iniciar el programa, etc.

Palabra reservada	Descripción
WriteLine	Esta se debe de colocar antes de poder imprimir lo que deseamos seguido de paréntesis.
Start With	Esta será el motor de arranque para el lenguaje ya que esta tomará como principal una de las funciones o métodos, solo se debe de tener un Start With Start With método/función(pueden o no venir parámetros);
DynamicList	Así tomara el nombre al declarar una lista dinámica
Append	Append podrá agregar valores a la lista Append(nombrelista, valor a agregar);
getValue	getValue podrá acceder a una de las posiciones de la lista para poder ver su contenido - getValue(nombrelista, posiciónlista); - <TIPO> ID = getValue(lista, posición);
setValue	setValue podrá modificar el valor de un vector en una posición específica. setValue(nombrelista, posicionlista, valor);
toLower	Esta función recibe como parámetro una expresión de tipo cadena y retorna una nueva cadena con todas las letras minúsculas toLower(EXPRESION);
toUpper	Esta función recibe como parámetro una expresión de tipo cadena retorna una nueva cadena con todas las letras mayúsculas. toUpper(EXPRESION)

Sentencias de Control

Palabras Reservadas	Descripción
IF	<p>Esta sentencia solo se ejecuta si se cumple la condición, en dado caso no sean igual no entra en esta sección del código</p> <pre>if(CONDICION){ INSTRUCCIONES }</pre> <p>Donde la condición puede ser una operación u operación relacional (Ver tabla 2 sección de operaciones) o una expresión de tipo booleana junto a esto pueden venir con una operación lógica y las instrucciones pueden ser más sentencias, llamadas a métodos/funciones, declaraciones, etc.</p>
IF-ELSE	<p>En esta sentencia, le agregamos la palabra else el cual representa que, si no se cumple con la condición del if, se pasa al else ya que este puede realizar alguna instrucción que no esté especificada en el if</p> <pre>if(CONDICION){ INSTRUCCIONES }else{ INSTRUCCIONES }</pre>
IF-ELSE IF	<p>En esta sentencia hace la referencia a la condición “Si-Si No” ya que en cada bloque que encontremos hará la pregunta que si no se realiza esa condición pregunta si se puede realizar otra hasta encontrar un else que rompa el ciclo.</p> <pre>If (CONDICION) { INSTRUCCIONES } else if (CONDICION) { INSTRUCCIONES } else if (CONDICION) { INSTRUCCIONES } else { INSTRUCCIONES }</pre>
Switch	<p>Esta sentencia es una alternativa del if ya que va evaluando casos como si fueran los condicionales del if su estructura es la siguiente:</p> <pre>Switch (EXPRESION){</pre>

	LISTA DE CASOS , DEFAULT
Case-Default	<pre>} Estos serán los casos que estarán dentro del switch, la estructura completa del switch con los casos quedaría de la siguiente manera: switch (EXPRESION) { case EXPRESION: INSTRUCCIONES; BREAK; case EXPRESION: INSTRUCCIONES; BREAK; case EXPRESION: INSTRUCCIONES; BREAK; default: INSTRUCCIONES }</pre> <p>Donde las expresiones, pueden ser decimales, enteros, cadenas de texto o booleanos (break se explicará en las sentencias de transferencia) y el default es la sentencia que contiene las demás instrucciones si se llega a salir de un caso por medio del break</p>

Sentencias Cíclicas.

Palabras Reservadas

Descripción

While	Esta sentencia o bucle se ejecutará mientras la condición se cumpla, una vez no se cumpla nunca entrará a las instrucciones <pre>while (CONDICION){ INSTRUCCIONES }</pre>
Do-While	Esta sentencia es idéntica a la anterior, pero con la excepción de que si va a entrar a las instrucciones y luego evalúa la condición y si se cumple vuelve a entrar de lo contrario ya no lo hace, si su principal caracteriza es que ejecuta al menos una vez las instrucciones <pre>do{ INSTRUCCIONES } while (CONDICION);</pre>
For	Este bucle nos permite ejecutar n veces las instrucciones que necesitamos hasta que se cumpla la condición dentro de este, por lo cual su sintaxis es la siguiente: <pre>for (DECLARACION ASIGNACION; CONDICION; ACTUALIZACION){ INSTRUCCIONES }</pre>

Sentencias de Transferencias

Palabras Reservadas	Descripción
Break	Esta sentencia hace que salga del ciclo inmediatamente, es decir que lo que se encuentra después de esto ya no se tomara en cuenta y saldrá del ciclo
Continue	Esta sentencia deber de estar dentro de un ciclo si no dará error, su funcionalidad es detener un ejecución y pasar a la siguiente es decir pasa de una condición a otra
Return	Esta sentencia finaliza un bloque retornando el valor asignado, esto se utiliza principalmente para métodos o funciones.

Funciones Nativas

Palabra Reservada	Descripción
Lenght	Esta función recibe como parámetro un vector, una lista o una cadena y devuelve el tamaño de este. lenght(VALOR);
Truncate	Esta función recibe como parámetro un valor numérico. Permite eliminar los decimales de un número, retornando un entero. truncate(VALOR);
Round	Esta función recibe como parámetro un valor numérico. Permite redondear los números decimales según las siguientes reglas: Si el decimal es mayor o igual que 0.5, se aproxima al número superior Si el decimal es menor que 0.5, se aproxima al número inferior round(VALOR);
Typeof	Esta función retorna una cadena con el nombre del tipo de dato evaluado. typeof(VALOR)
toString	Esta función permite convertir un valor de tipo numérico o booleano en texto. toString(VALOR)
toCharArray	Esta función permite convertir una cadena en un arreglo de caracteres. toCharArray(VALOR)

Tabla de tipos

TIPO	DEFINICION	DESCRIPCION	EJEMPLO
Entero	Int	Este tipo de datos aceptará solamente números enteros.	1, 50, 100, 25552, etc.
Doble	Double	Admite valores numéricos con decimales.	1.2, 50.23, 00.34, etc.
Booleano	Boolean	Admite valores que indican verdadero o falso.	True, false
Caracter	Char	Tipo de dato que únicamente aceptará un único carácter, y estará delimitado por comillas simples. ''	'a', 'b', 'c', 'E', 'Z', '1', '2', '^', '%', ')', '=', '!', '&', '/', '\\', 'n', etc.
Cadena	String	Es un grupo o conjunto de caracteres que pueden tener cualquier carácter, y este se encontrará delimitado por comillas dobles. ""	"cadena1", "-- ** cadena 1"

Tabla de Operaciones Relacionales.

OPERADOR	DESCRIPCIÓN	EJEMPLO
==	Igualación: Compara ambos valores y verifica si son iguales: - Iguales= True - No iguales= False	1 == 1 "hola" == "hola" 25.5933 == 90.8883 25.5 == 20
!=	Diferenciación: Compara ambos lados y verifica si son distintos. - Iguales= False - No iguales= True	1 != 2, var1 != var2 25.5 != 20 50 != 'F' "hola" != "hola"
<	Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo. - Derecho mayor= True - Izquierdo mayor= False	(5/(5+5))<(8*8) 25.5 < 20 25.5 < 20 50 < 'F'
<=	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo. - Derecho mayor o igual= True - Izquierdo mayor= False	55+66<=44 25.5 <= 20 25.5 <= 20 50 <= 'F'
>	Mayor que: Compara ambos lados y verifica si el izquierdo es mayor que el derecho. - Derecho mayor= False - Izquierdo mayor= True	(5+5.5)>8.98 25.5 > 20 25.5 > 20 50 > 'F'
>=	Mayor o igual que: Compara ambos lados y verifica si el izquierdo es mayor o igual que el derecho. - Derecho menor o igual= True - Izquierdo menor= False	5-6>=4+6 25.5 >= 20 25.5 >= 20 50 >= 'F'

Tabla de Operadores Lógicos.

OPERADOR	DESCRIPCIÓN	EJEMPLO	OBSERVACIONES
	OR: Compara expresiones lógicas y si al menos una es verdadera entonces devuelve verdadero en otro caso retorna falso	(55.5) bandera==true Devuelve true	bandera es true
&&	AND: Compara expresiones lógicas y si son ambas verdaderas entonces devuelve verdadero en otro caso retorna falso	(flag1) && ("hola" == "hola") Devuelve true	flag1 es true
!	NOT: Devuelve el valor inverso de una expresión lógica si esta es verdadera entonces devolverá falso, de lo contrario retorna verdadero.	!var1 Devuelve falso	var1 es true

Montar el servidor y la Web App

Para poder inicializar la aplicación debemos de seguir ciertos comando en consola ya que debemos de levantar el framework y servies por lo que debemos de seguir una serie de pasos que son los siguientes:

1. Al tener ambas carpetas, debemos de tener en cuenta quien es el servidor y quien es el cliente.

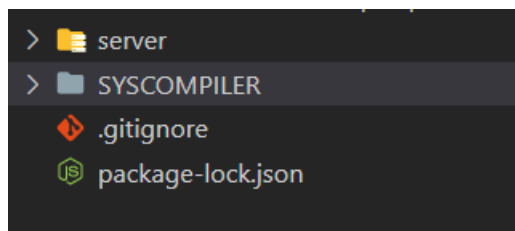


Ilustración 1 Carpetas del Proyecto

2. Iniciamos el Visual Studio Code y arrastramos la carpeta de nuestro proyecto hacia su inicio.

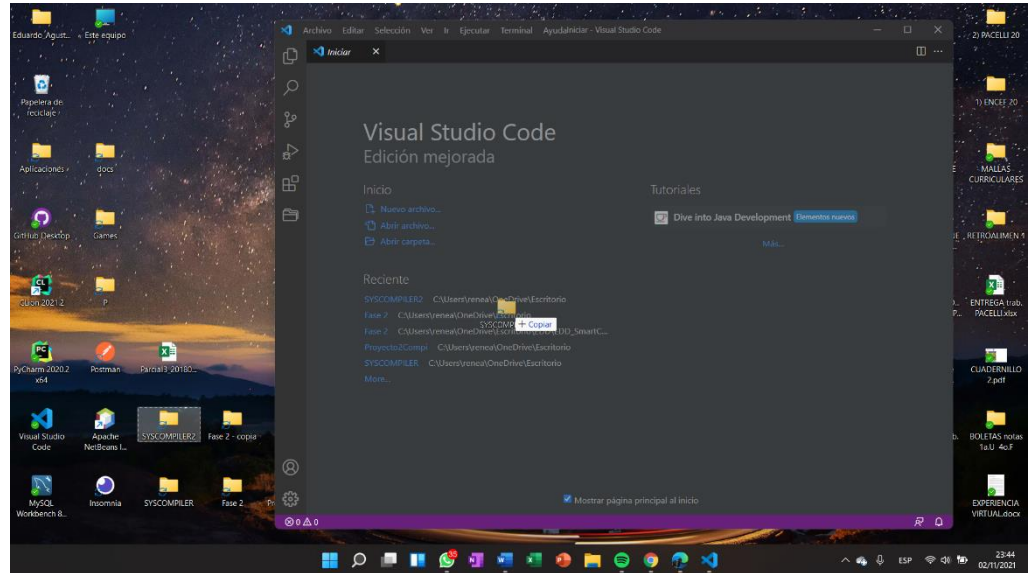
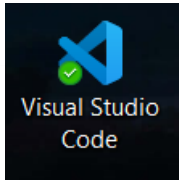


Ilustración 2 Pantalla principal del editor en blanco

3. Al arrastrar se nos abrirá la carpeta del lado izquierdo donde tendremos las carpetas que vendrán dentro de nuestro proyecto, posterior a eso nos dirigimos al lado superior de la ventana y seleccionamos la parte de terminal, después de eso nos desplegará un submenú en donde daremos clic en donde dice nueva terminal o utilizando el atajo Ctrl + ñ, el cual nos desplegará una terminal en la parte de abajo.

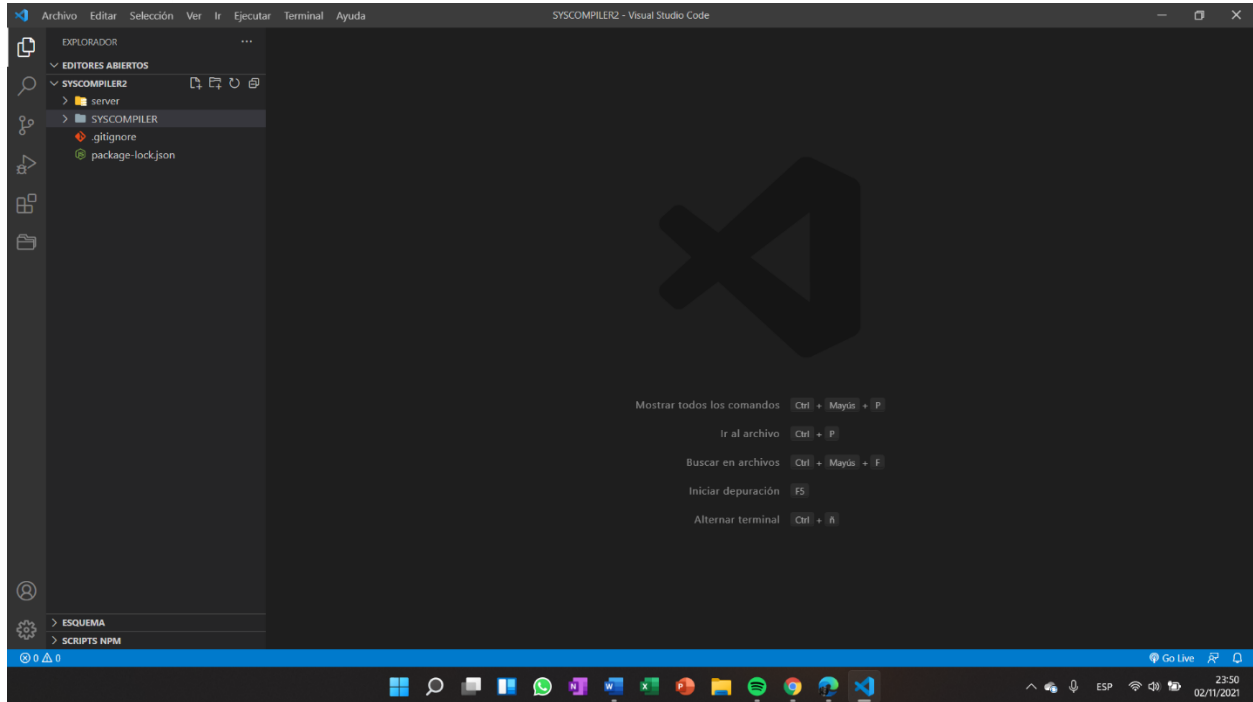


Ilustración 3 Pantalla Principal del editor al arrastrar el archivo

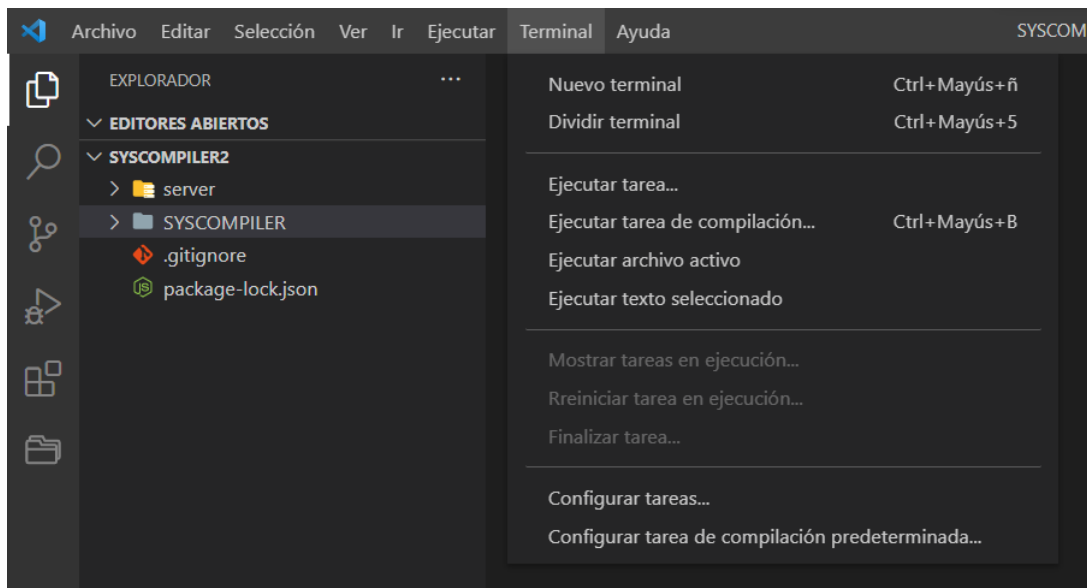


Ilustración 4 Activación de la consola

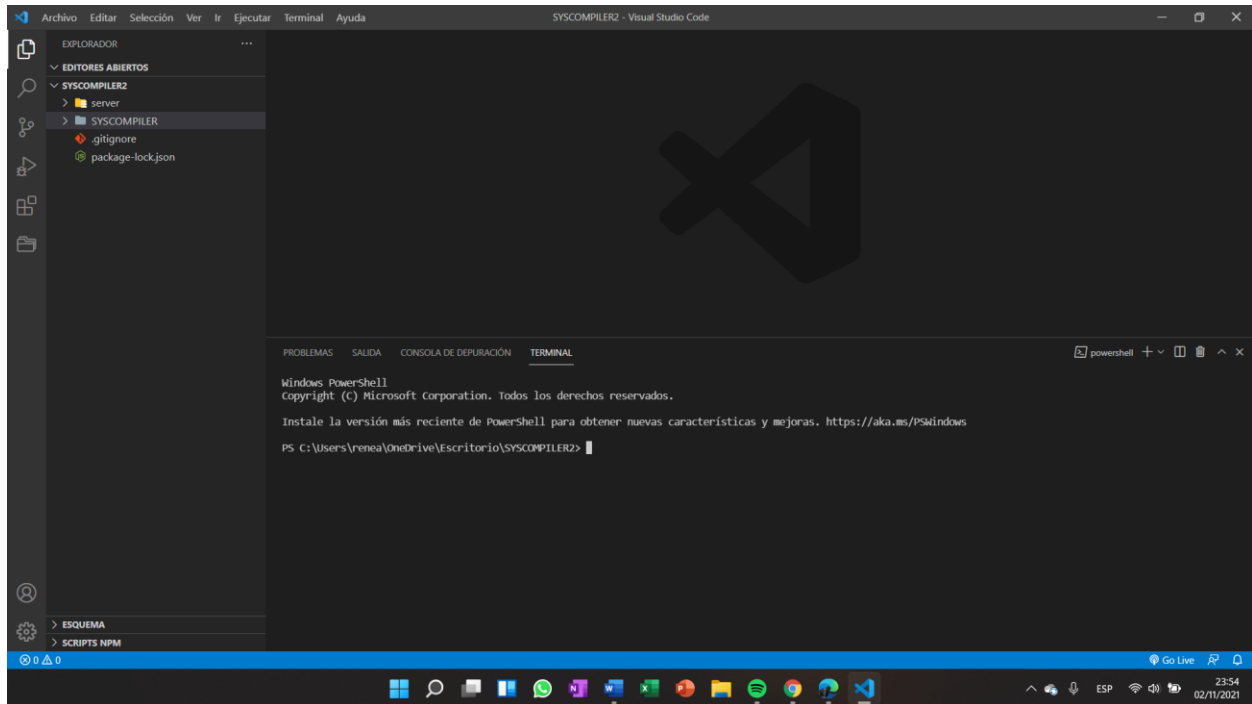
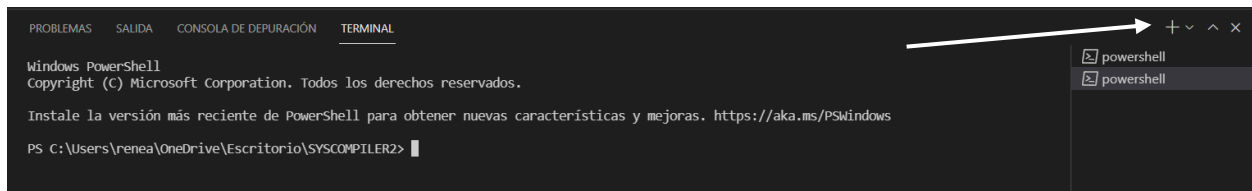
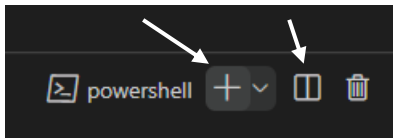
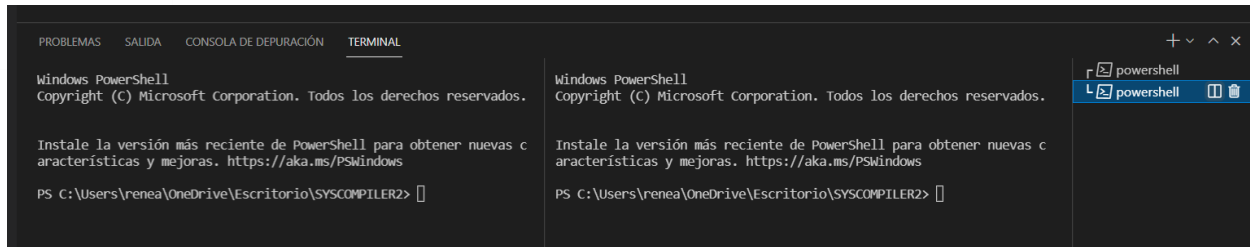


Ilustración 5 Vista de consola en el editor

4. Teniendo la consola desplegada nos mostrara la ruta de nuestro proyecto, en la esquina superior derecha de la consola veremos un + le daremos clic ya que necesitaremos de otra consola (Podemos darle clic también a la pestaña que está al lado izquierdo del bote de basura para poder tener conjuntas las dos consolas).





5. Al tener esto escogeremos la manera en la que nos guste más, al tener listo ambas consolas ejecutaremos los comandos **cd** y **npm** los cuales **cd** es para moverse entre carpetas y el **npm** es nativo de Node.js para poder ejecutar los servicios y el cliente. **OJO** si al descargar la carpeta del proyecto no te aparecen los **node_modules** dentro de las carpetas del Front (SYSCOMPILER) o Back (SERVER) ejecutar el siguiente comando dentro de las carpetas para poder instalar todas las dependencias de Angular.

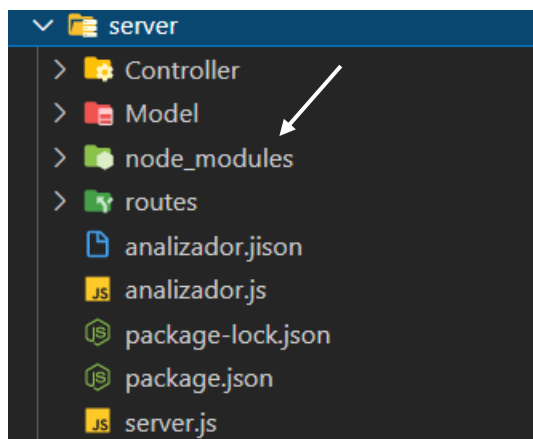


Ilustración 6 Carpeta **node_modules** en Back end

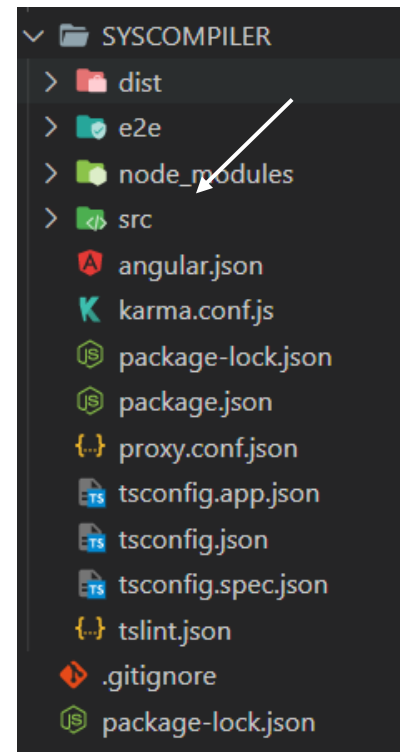
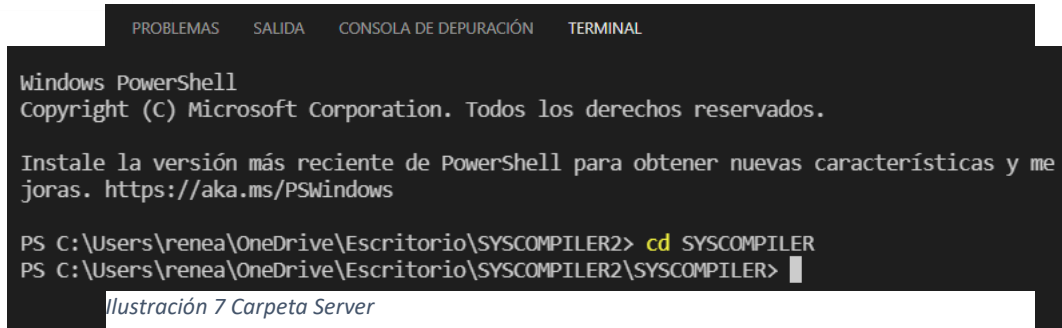


Ilustración 6.1 Carpeta **node_modules** en Front end

Si en caso no existieran esas carpetas nos moveremos a su respectivo directorio en la consola son los siguientes comandos.

- Server



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

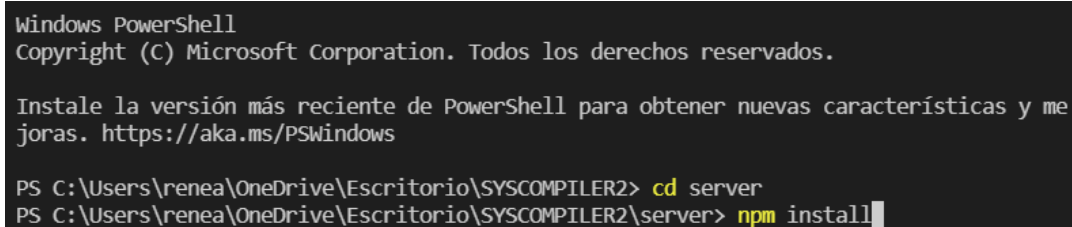
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2> cd SYSCOMPILER
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\SYSCOMPILER> |
```

Ilustración 7.1 Carpeta Syscompiler

- SYSCOMPILER

Se ejecuta el siguiente comando para instalarlos módulos, este comando se debería de realizar en ambas carpetas



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

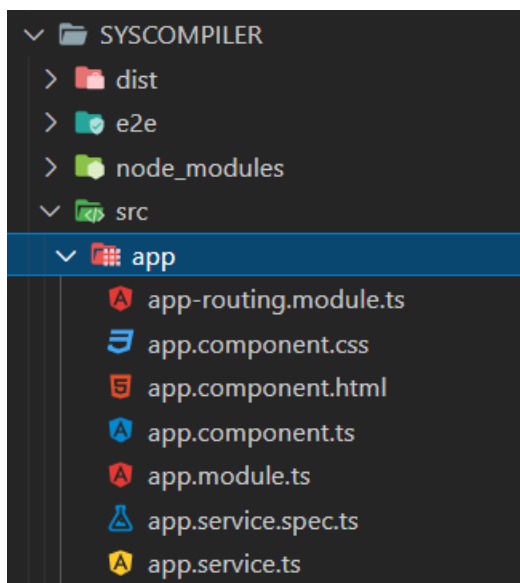
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2> cd server
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\server> npm install
```

Ilustración 8 Instalar dependencias de Angular

Al darle enter, se instalarán las dependencias y listo ya podemos empezar a montar todo el proyecto.

6. Al tener instalado todo esto, en el server nos quedaremos en esta carpeta y aquí ejecutaremos el comando **npm start** y en el syscompiler nos moveremos hasta la carpeta app en donde tenemos la página web y sus componentes, pero para llegar hasta esa carpeta nos moveremos de carpeta en carpeta hasta poder llegar.



Para esto debemos de ejecutar el comando **cd** y nos moveremos de syscompiler hacia **src** y de **src** nos moveremos hacia **app** en **app** ejecutaremos el comando **npm start** y listo la página por defecto se ejecutará en el puerto 4500

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2> cd SYSCOMPILER
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\SYSCOMPILER> cd src
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\SYSCOMPILER\src> cd app
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\SYSCOMPILER\src\app> npm start
```

Como tal ejecutaremos ambos comandos de start

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2> cd server
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\server> npm start

> server@1.0.0 start C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\server
> node server.js

Server listening on the port: 3080

runtime.js      | runtime      | 6.15 kB
                | Initial Total | 3.98 MB

Build at: 2021-11-03T06:23:02.280Z - Hash: 9de35e055945dd7e382f - Time: 11042ms

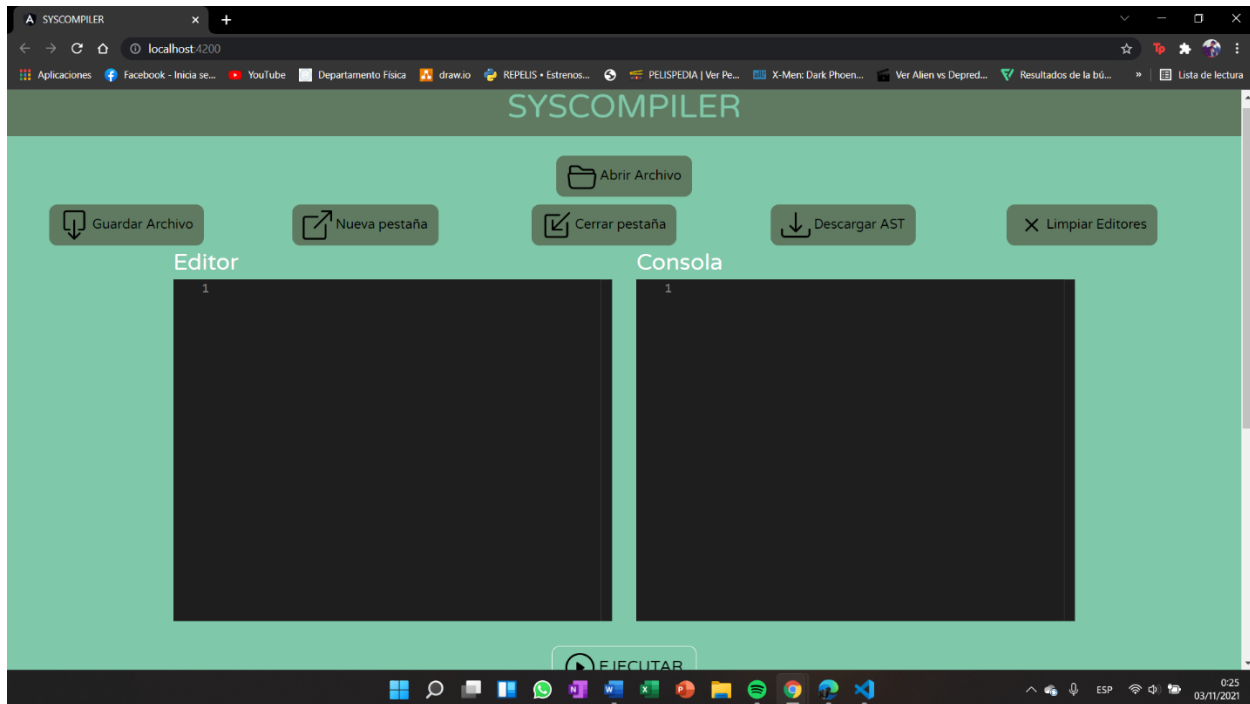
Warning: C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\SYSCOMPILER\src\app\app.component.ts depends on 'file-saver'. CommonJS or AMD dependencies can cause optimization bailouts.
For more info see: https://angular.io/guide/build#configuring-commonjs-dependencies

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

√ Compiled successfully.

```

Como tal al levantar ambos, nos aparecerá esto, posterior a esto podemos copiar la ruta localhost:4200 y pegarla en el navegador de mejor preferencia.



Y así es como estaremos ya dentro de la aplicación, posterior a esto ya podemos ir probando la sintaxis descrita anteriormente, las pruebas estarán en la parte de anexo para que se pueda visualizar el resultado.

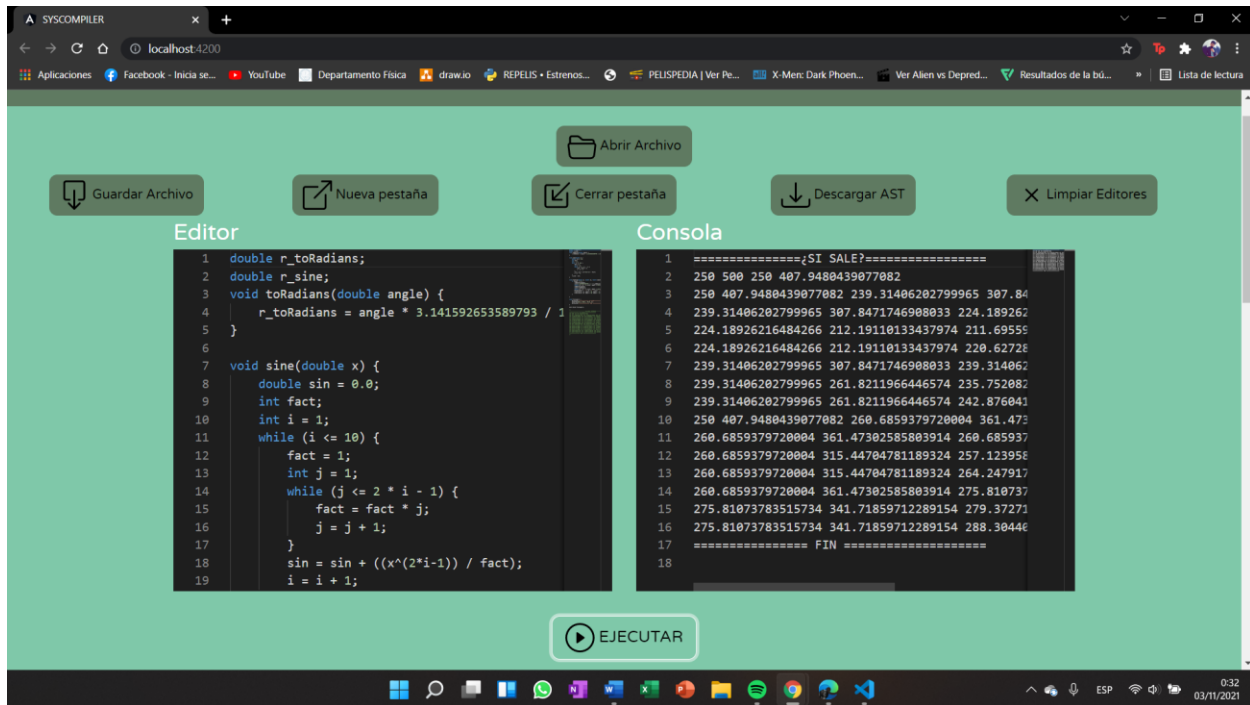
Para poder parar tanto el servidor como el cliente debemos de presionar ctrl + c y luego presionar la letra s con esto podremos detener ambas terminales


```

}
¿Desea terminar el trabajo por lotes (S/N)? s
PS C:\Users\renea\OneDrive\Escritorio\SYSCOMPILER2\server>

```

Anexos



The screenshot shows the SYSCompiler web IDE interface. The top bar includes navigation icons and a browser address bar showing 'localhost:4200'. Below the top bar are buttons for 'Abrir Archivo', 'Guardar Archivo', 'Nueva pestaña', 'Cerrar pestaña', 'Descargar AST', and 'Limpiar Editores'. The main area is split into two panels: 'Editor' and 'Console'. The 'Editor' panel contains the following C code:

```

1 //Este es el primer archivo de prueba
2 /*En este archivo se verifica el funcionamiento
3 ciclo while, así como la sentencia if y else
4 */
5
6 void metodo1(){
7     //llamada del metodo
8     figura1(10);
9 }
10 void figura1(int n) {
11     String cadenaFigura = "";
12     double i;
13     i=3*n/2;
14     //iniciando dibujo
15     while(i<n){
16         cadenaFigura = "";
17         double j;
18         j=3*n/2;
19         while(j<=3*n){

```

The 'Console' panel shows the output of the program, which is a diamond shape made of asterisks. The 'EJECUTAR' button is located at the bottom center of the interface.

The screenshot shows the SYSCompiler web IDE interface. The top bar includes navigation icons and a browser address bar showing 'localhost:4200'. Below the top bar are buttons for 'Abrir Archivo', 'Guardar Archivo', 'Nueva pestaña', 'Cerrar pestaña', 'Descargar AST', and 'Limpiar Editores'. The main area is split into two panels: 'Editor' and 'Console'. The 'Editor' panel contains the following C code:

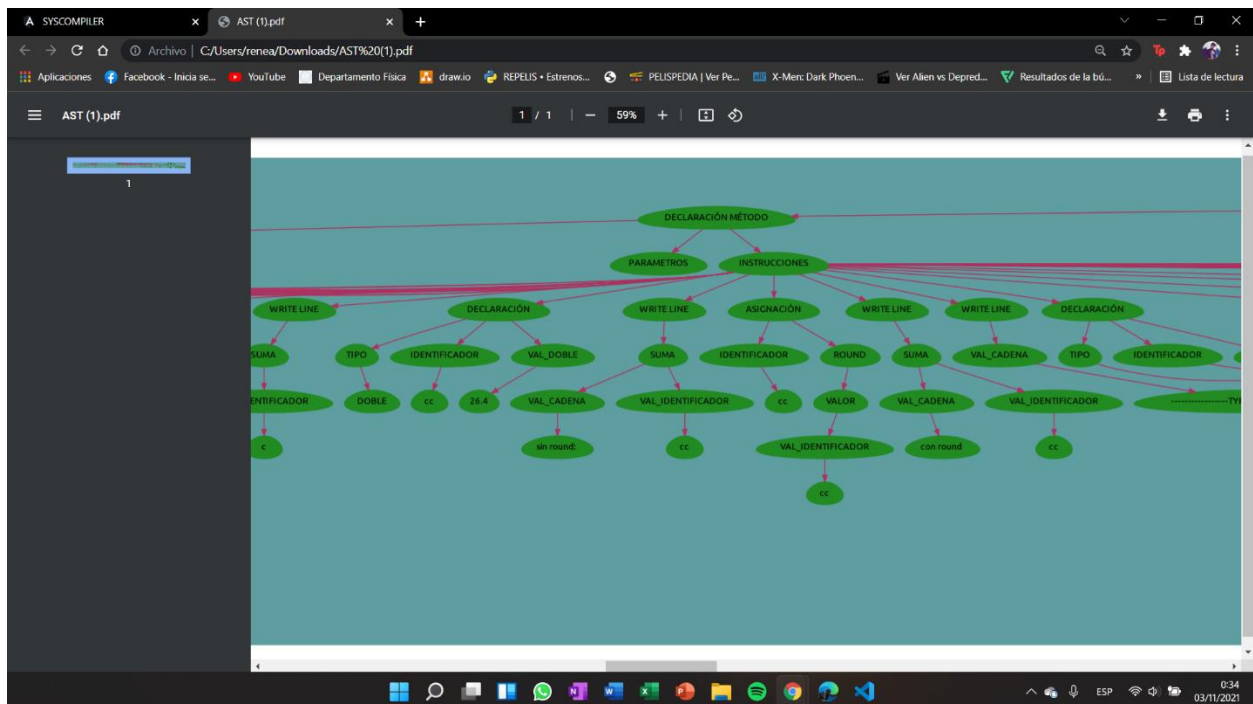
```

1 void funcionesEspecialesYNativas(){
2     int a = 15;
3     writeline("-----TOLOWER-----");
4     writeline("SIN TOLOWER");
5     writeline(tolower("CON TOLOWER"));
6     writeline("-----TOUPPER-----");
7     writeline("sin toupper");
8     writeline(toupper("con toupper"));
9     writeline("-----TRUNCATE-----");
10    double b=17.8;
11    writeline("sin truncate: "+b);
12    b=truncate(b);
13    writeline("con truncate "+b);
14    writeline("-----ROUND-----");
15    double c=26.5;
16    writeline("sin round: "+c);
17    c=round(c);
18    writeline("con round "+c);
19    double cc=26.4;

```

The 'Console' panel shows the output of the program, which includes the results of various standard library functions: TOLOWER, TOUPPER, TRUNCATE, ROUND, and TYPEOF. The 'EJECUTAR' button is located at the bottom center of the interface.

#	Nombre	Tipo símbolo	Tipo dato	Ámbito	Fila	Columna
1	funcionesEspecialesYNativas	Método	VOID	global	1	1
2	a	Variable	ENTERO	funcionesEspecialesYNativas	2	5
3	b	Variable	DOBLE	funcionesEspecialesYNativas	10	5
4	c	Variable	DOBLE	funcionesEspecialesYNativas	15	5
5	cc	Variable	DOBLE	funcionesEspecialesYNativas	19	5
6	x	Variable	CADENA	funcionesEspecialesYNativas	24	5
7	y	Variable	ENTERO	funcionesEspecialesYNativas	25	5
8	z	Variable	DOBLE	funcionesEspecialesYNativas	26	5
9	xx	Variable	CARACTER	funcionesEspecialesYNativas	27	5
10	yy	Variable	BOOLEANO	funcionesEspecialesYNativas	28	5
11	cadena	Variable	CADENA	funcionesEspecialesYNativas	35	5
12	numero	Variable	ENTERO	funcionesEspecialesYNativas	38	5
13	imprimirListaChar	Método	VOID	global	49	1
14	miLista	Parámetro	Dynamic List: CARACTER	imprimirListaChar	49	24



The screenshot displays the SYSCOMPILER web application interface. At the top, there's a header with the SYSCOMPILER logo. Below the header, there are several buttons: 'Abrir Archivo', 'Guardar Archivo', 'Nueva pestaña', 'Cerrar pestaña', 'Descargar AST', and 'Limpiar Editores'. The main area is divided into two panels: 'Editor' and 'Consola'.

Editor Panel:

```
1 void function(){
2     Writeline("Hola Mundo");
3     int vectorNumero [] = {2020,2021,2022};
4     writeline(vectorNumero);
5     string vector2[ ] = {"hola", "Mundo"};
6     string val = vector2[0];
7     writeline(val);
8     int vector [] = new int[4];
9     vector[0] = 1;
10    vector[1] = 2;
11    vector[2] = 3;
12    vector[3] = 4;
13    int val2 = vector[2];
14    writeline(val2);
15 }
16 start with function();
```

Consola Panel:

```
1 Hola Mundo
2 [ 2020, 2021, 2022 ]
3 hola
4 3
5
```

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 0:35 on 03/11/2021.