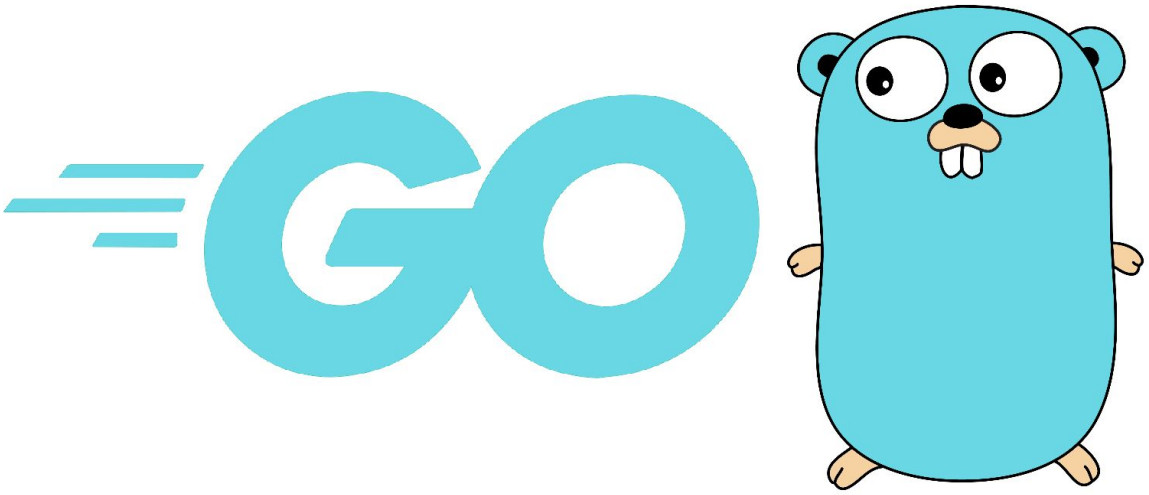
Manual Técnico

Entorno de desarrollo

Lenguaje de desarrollo: javascript

Para el desarrollo del servidor se usó NodeJs en su versión 12 y express 4.17

Para el desarrollo del cliente se hizo uso de Golang



El analizador léxico y Sintáctico usado es Jison

Ide utilizado: visual studio code

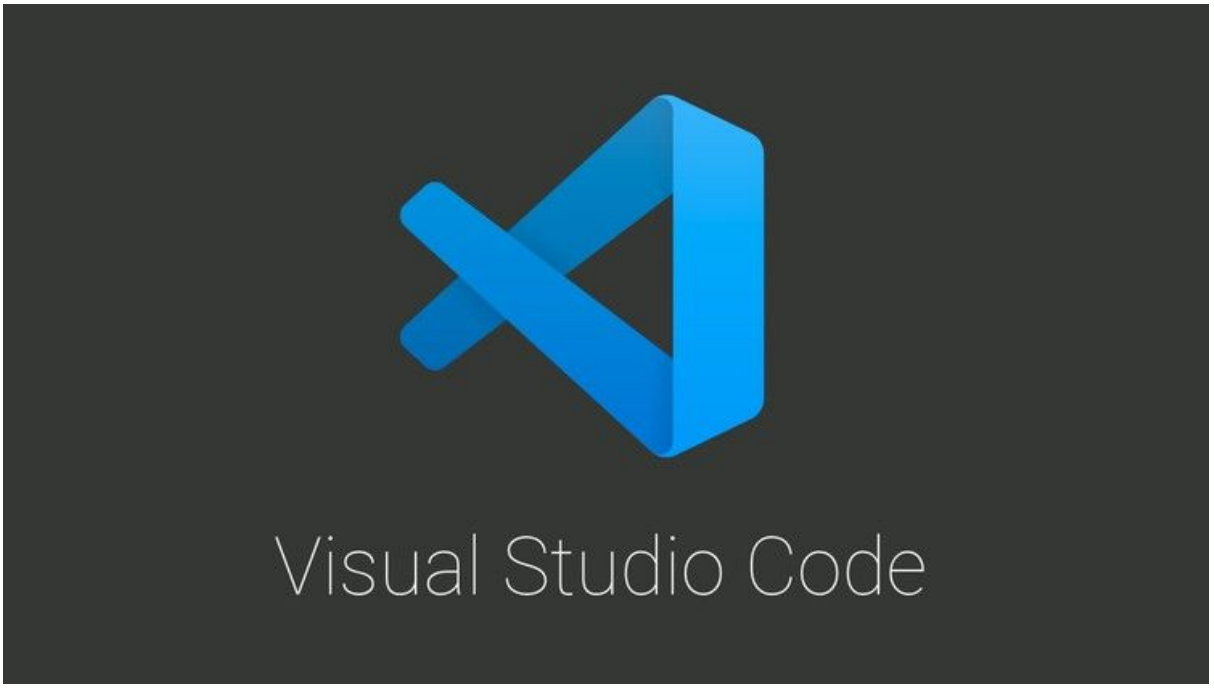


Diagrama de Clases/Estructuras

Gramática Análisis Lexico/Sintactico

Para definir y poder hacer uso símbolos o palabras reservadas es necesario retornar las con algún identificador en concreto el cual nos permitirá hacer uso de dichas reservadas en el análisis sintáctico.
Algunas de las palabras reservadas para nuestro lenguaje son las siguientes

```
//aritmeticos
"+"                 return 'suma';
"-"                 return 'resta';
"*"                 return 'multiplicacion';
"^"                 return 'potencia';
"/"                 return 'slash';
"%"                 return 'modulo';

//relacionales
">="                  return 'mayorigual';
"<="                  return 'menorigual';
"<"                   return 'menor';
">"                   return 'mayor';
"=="                  return 'identico';
"==="                 return 'referencias';
"!="                  return 'diferente';
```

Para poder reconcer una cadena por patrón hacemos uso de las expresiones regulares al igual que las palabras reservadas es necesario retornar las para su uso posterior.
Algunas expresiones regulares en nuestro lenguaje son las siguientes

```
/* Espacios en blanco */
[ \r\t\n]+                  {}

"'"[^]"'"                                   return 'caracter';
[0-9]+"."[0-9]+\b                             return 'decimal';
[0-9]+\b                                      return 'entero';
([\"]("\\\""|[^"])*[^\\][\"])|[\"][\"]         return 'cadena';
([a-zA-Z""])[a-z0-9A-Z""ñ""Ñ"]*               return 'id';
//([\"](("\\\"")|[^\"])*[^\\][\"])|[\"][\"]    return 'cadena';
```

Para no tener problemas de ambigüedad y que el parser no sepa por donde comenzar podemos definir jerarquía entre nuestras reservadas o expresiones regulares.
El siguiente ejemplo es la jerarquía que otorgamos en nuestro lenguaje además que se puede visualizar la como es otorgada la jerarquía entre más abajo se encuentran las reservadas/expresiones contarán con mayor precedencia.

```
%right igual
%left incremento
%left decremento
%left or
%left and
%left identico, diferente, referencias
%left mayor, menor, mayorigual, menorigual
%left suma, resta
%left multiplicacion, slash,modulo
%right potencia
%right not
```

Para poder definir código javascript o importar otras estructuras lo podemos hacer de la siguiente forma que es la manera en la que jison interpreta que es código a utilizar tanto en el scanner como en el parser.

```
%{
    const Node = require('./clases/Node');
    const Error = require('./clases/Error');
    temp = [];
%}
```

Respuesta a petición post

Para la respuesta es necesario que la entrada proporcionada por el cliente sea analisada la salida generara un Ast para poder mostrar en la página del cliente. también en el análisis se guarda la infromacion de la creacion de clases funciones y variables dicho arreglo es retornada como VARS, los errores como ERROR, el ast como AST

```
app.post('/compilar',(req,res) => {
    console.log(req.body);

    global.globerrores = [];
    global.globresults = [];
    global.globresultsFVC = [];

    if(globentradas.length > 1)
        globentradas = [];

    if(globentradas.length == 0)
    {
        globentradas.push(req.body.code);
        parser.parse(req.body.code);
    }else if(globentradas.length == 1)
    {
        globentradas.push(req.body.code);
        parser.parse(req.body.code);
    }
    //console.log(globerrores.length);
    //console.log(globresults.length);
    //console.log(globresultsFVC.length);
    //console.log(parser.RES.operar())
    //console.log(count.getOutput());
    //console.log(count.getError());
    res.send({ERROR: globerrores, AST: globresults[globresults.length -1], VARS: globresultsFVC});
})
```

Variables de retorno

para poder retornar y poder almacenar todas las clases funciones y variables encontradas en el parser se necesita hacer uso de varios arreglos en javascript que son los siguientes

```
global.globentradas = [];
global.globerrores = [];
global.globresults = [];
global.globresultsFVC = [];
```

Servidor Cliente

Para poder levantar y poner a escuchar nuestro servidor cliente es realizado con golang pero para esto necesitamos importar nuestra plantilla html y los archivos javascript y css que requiera utilizar este, para eso hacemos uso del siguiente código.

```go
import (
    "fmt"
    "html/template"
    "net/http"
)

func index(w http.ResponseWriter, r *http.Request) {

    t := template.Must(template.ParseFiles("index.html"))
    t.Execute(w, "")
}

func main() {
    http.Handle("/css/", http.StripPrefix("/css/", http.FileServer(http.Dir("css/"))))
    http.Handle("/fonts/", http.StripPrefix("/fonts/", http.FileServer(http.Dir("fonts/"))))
    http.Handle("/js/", http.StripPrefix("/js/", http.FileServer(http.Dir("js/"))))
    http.Handle("/codemirror/", http.StripPrefix("/codemirror/", http.FileServer(http.Dir("codemirror/"))))

    http.HandleFunc("/", index)

    fmt.Printf("Servidor escuchando en: http://localhost:8000/")
    http.ListenAndServe(":8000", nil)
}
```

como se puede observar nuestro servidor para el lado del cliente estará escuchando en el puerto 8000

Peticiones Cliente -> Servidor

Para esto hacemos uso de peticiones en xml en javascript el código necesario se encuentra en la carpeta js en el archivo index.js el cual será ampliado a continuación

Para la petición que necesita que sea analizado se captura el texto que se encuentra en el textarea mostrado en el tab seleccionado, para esto hacemos uso de las peticiones post de jquery, que son asíncronas. está necesita la url a la cual haremos la petición necesitamos también un json para enviar para saber si hay respuesta hacemos uso de status que nos devolverá un string de success si obtuvo respuesta de ser así si la petición es post tendremos la respuesta en la variable que indiquemos en este caso será data

```
var data1, data2 = null;
function send_request()
{
    var ta=document.getElementById(get_vent());
    var contenido=ta.value;//texto de vent actual

    var url = 'http://localhost:3000/compilar';

    $.post(url,{code: contenido}, function(data, status){
        if(status.toString() == "success"){

            if(data2 != null)…
            }

            if(data1 == null)…
            }
            else…
            }
            crear_reporte_errores(data.ERROR);
        }else{
            alert("Error en la conexion:" + status)
        }
    });
}
```

Generar Reportes Copia/Errores

Para esto en la información que retorna el servidor que realiza el análisis se retorna todas la variables encontradas, clases y funciones, este arreglo crea otros tres arreglos para variables, funciones y clases, luego de esto se guardan si es la primera vez que se solicita al servidor que analise la entrada seleccionada, de no ser la primera pasa a realizar la comparacion primero la de las variables para encontrar si las variables pertenecen tanto a la misma función y clase, de ser copia se agregan a una estructura que es la que genera el reporte de variable, la mecanica para la deteccion de funciones es similar a diferencia que verifica si tienen el mismo tipo de retorno si tuvieran, para las clases se hace el conteo de las funciones que posee y si la de ambas clases coinciden entonces se establece dicha clase como copia.

```
for(var i = 0; i < vars.length ; i++)
{
    for(var j = 0; j < vars1.length ; j++)
    {
        if(vars1[j].VALUE == vars[i].VALUE && vars1[j].PARENT == vars[i].PARENT)
        {
            add_to_table_rv(vars1[j].VALUE + " " + vars1[j].PARENT, vars[i].VALUE + " " + vars[j].PARENT);
            break;
        }
        //realizo la comparacion si son iguales las clases
    }
}
```

errores 9 of 14

```
    for(var i = 0; i < varsfunction.length ; i++)
    {
        if(find_and_compare_function(varsfunction[i].NOMBRE, i))
        {
            add_to_table_rf(varsfunction[i].VALUE + " " + varsfunction[i].PARENT, varsfunction[i].VALUE + " " + varsfunction[i].PARENT);
            break;
        }

    }
    for (var i = 0 ; i < varsclass.length ; i++)
    {
        for (var j = 0 ; j < varsclass1.length ; j++)
        {
            if(varsclass[i].VALUE == varsclass1[j].VALUE)
            {
                var number =  get_number_equal(varsclass[i].VALUE);
                if(number > 0)
                    add_to_table_rc(varsclass[j].VALUE + " " + varsclass[j].PARENT + "No. Funciones " + number, varsclass1[i].VALUE + " " + varsclass
                break;
            }
        }
    }
```

Para el reporte de errores simplemente se recorre el arreglo de errores retornado del análisis ejecutado por el servidor que maneja el scanner y parser nodejs

```
function crear_reporte_errores(errores)
{
    var rep_error = "<table>\n<tr>\n<th>Tipo Error</th>\n<th>Columna</th>\n<th>Linea</th>\n<th>Descripcion</th>\n</tr>\n";
    for(var i = 0 ; i < errores.length ; i++)
    {
        rep_error += "<tr>\n<td>"+ errores[i].type +"</td>\n<td>"+ errores[i].column +"</td>\n"+"</td>\n<td>"+ errores[i].row +"</td>\n"+"</td>\n<td>"+ erro
    }
    rep_error += "</table>\n";
    if(errores.length > 0)
        download("Errores.html", rep_error);
    else
        rep_error = "";
}
```