Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en ciencias de sistemas.

Manual de técnico

Nombre del estudiante

José Luis Espinoza Jolón - 202202182

Ing.(a). Kevin Lajpop

Guatemala, Abril 21 de 2024

Introducción

Descripción del programa

Aplicar los conocimientos sobre la fase de análisis léxico y sintáctico de un compilador para la realización de un intérprete sencillo, con las funcionalidades principales para que sea funcional.

Lenguaje para utilizar

Para este tipo de proyecto utilizamos typescript.

Librería a utilizadas

- **Jison :** Jison es un generador de analizadores sintácticos (parser) diseñado para ayudar en la creación de intérpretes y compiladores. Es similar a Bison (Berkeley Yacc), pero está escrito en JavaScript y puede utilizarse tanto en el lado del servidor como en el navegador. Jison toma una gramática en forma de especificaciones de gramáticas libres de contexto y genera un parser en JavaScript que puede ser utilizado para analizar y procesar código fuente. Es una herramienta útil para proyectos que requieren análisis sintáctico de lenguajes de programación u otros formatos estructurados.

Donde se descargar: Link para descargar la librería https://gerhobbelt.github.io/jison/docs/

- **Graphviz-react:**

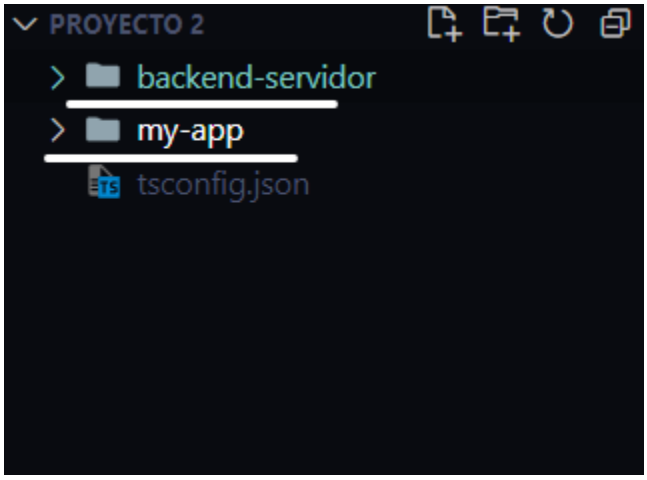
es una biblioteca de React que permite renderizar gráficos generados por Graphviz en aplicaciones web. Graphviz es una herramienta ampliamente utilizada para generar diagramas y visualizaciones de grafos a partir de descripciones de grafos en un lenguaje de texto llamado DOT.

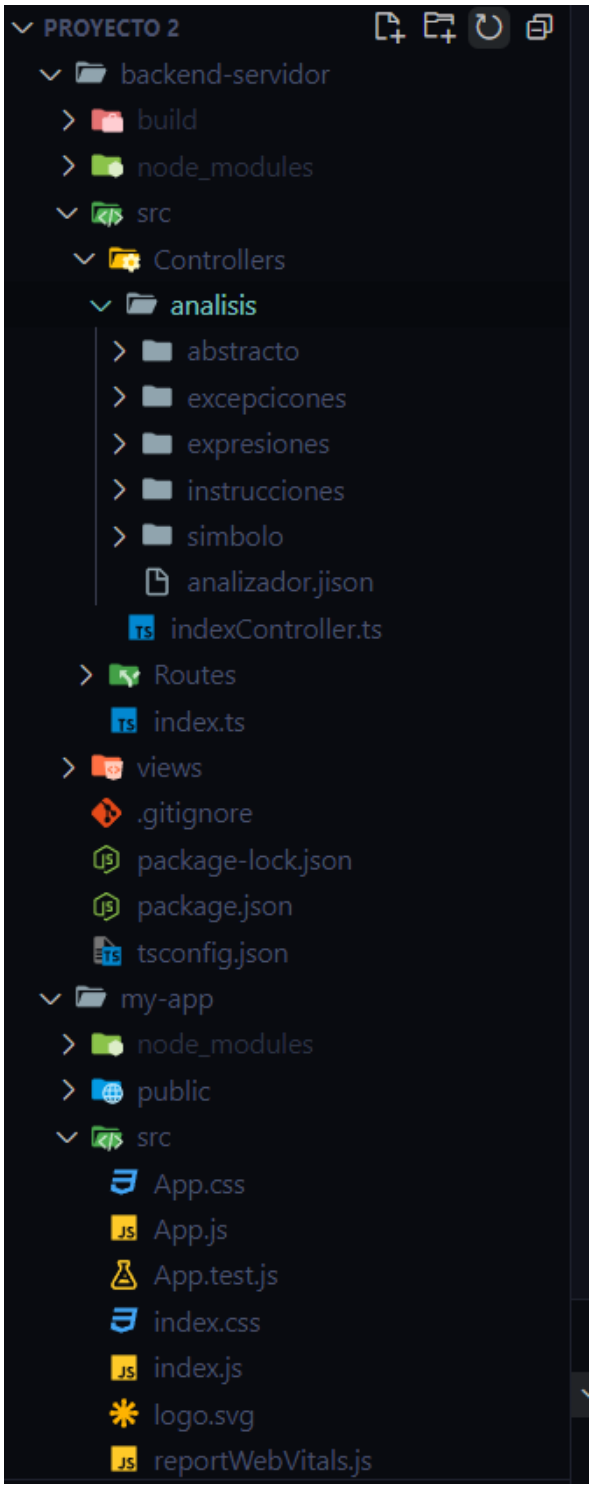
Donde se descarga:

Link https://www.npmjs.com/package/graphviz-react

Estructura del proyecto

La manera que organice el proyecto es de la siguiente forma:



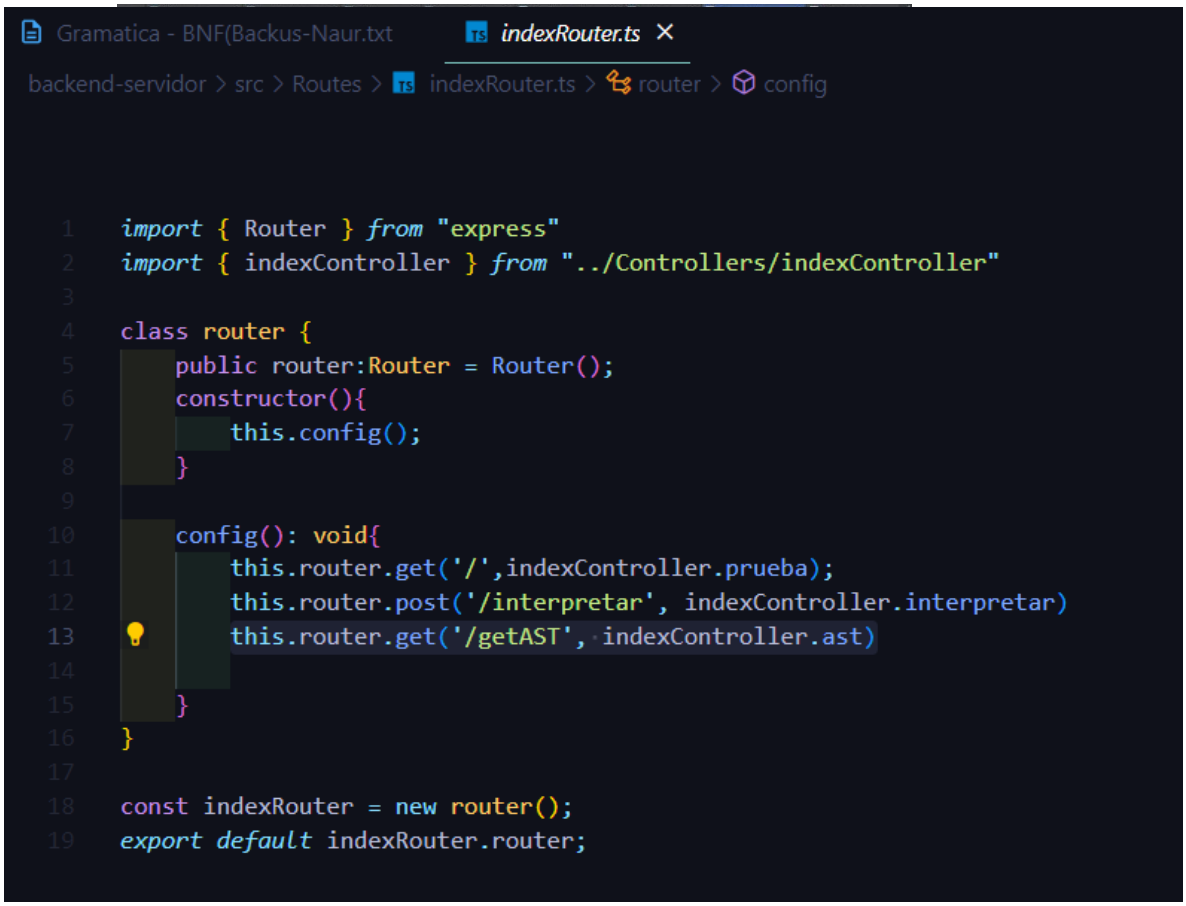


❖ **Backend-servicidor**

En esta parte se realizo toda la parte lógica del programa, donde se trabajo la gramática del lenguaje y las funcionalidades

Carpeta routes:

En esta parte es donde trabajamos la rutas para manejarla desde el fronted



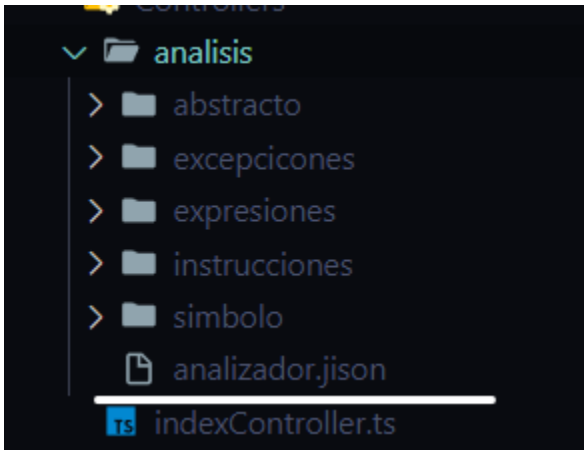
indexControle:

Donde realizamos la lectura de todo lo que viene de jison

```
class controller {
    public interpretar(req: Request, res: Response) {
        errores_list =  new Array<Errores>
        try {
            AstDot = ""
            let parser = require('./analisis/analizador')
            let ast = new Arbol(parser.parse(req.body.entrada))
            let tabla = new tablaSimbolo()
            tabla.setNombre("Ejemplo1")
            ast.setTablaGlobal(tabla)
            ast.setConsola("")

            let execute = null;

            //primer recorrdio
            for(let i of ast.getInstrucciones()){
                if(i instanceof Metodo || i instanceof Funcion){
                    i.id = i.id.toLocaleLowerCase()
                    ast.agregarFunciones(i)
                }
                if(i instanceof Declaracion){
                    i.interpretar(ast, tabla)
                    // manejo de errores
                }
                if (i instanceof Execute){
                    execute = i
                }

            }

            for (let i of errores_list) {
                ast.Print("---> "+i.getTipoError()+ ":" + i.getDescp() +" Fila: "+ i.getFila()+" Columna: "+ i.getColumna()+"\n" )
            }
```

Carpeta controller :

Donde se distribuyo las carpetas de cada parte del lenguaje y donde se realizo el jison que es la gramática



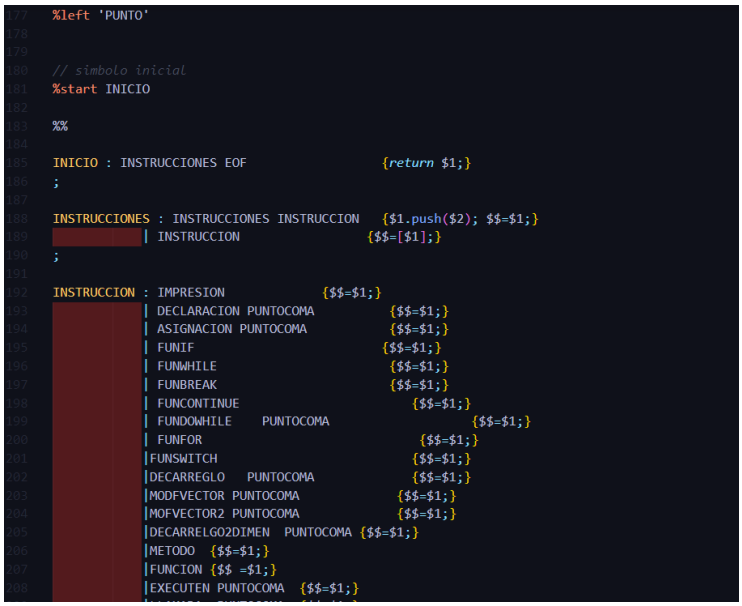
lexico:

Se realizo todos los tokens del lenguaje

```
"//".*      //comentario de una linea
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]    //multilinea

//palabras reservadas
"cout"              return 'IMPRIMIR'
"int"                   return 'INT'
"double"                return 'DOUBLE'
"std::String"               return 'STRING'
"true"              return 'TRUE'
"false"             return 'FALSE'
"char"              return 'CHAR'
"bool"              return 'BOOLEAN'
"pow"               return 'POW'
"tolower"           return 'TOLOWER'
"toupper"           return 'TOUPPER'
"round"             return 'ROUND'
"std::toString"             return 'TOSTRING'
"if"                return 'IF'
"while"             return 'WHILE'
"break"             return 'BREAK'
"typeof"            return 'TYPEOF'
"length"            return 'LENGTH'
"else"              return 'ELSE'
"do"                return 'DO'
"for"               return 'FOR'
"continue"          return 'CONTINUE'
"switch"            return 'SWITCH'
"case"              return 'CASE'
"default"           return 'DEFAULT'
"new"               return 'NEW'
"execute"               return 'EXECUTE'
"void"              return 'VOID'
"c_str"             return 'CSTR'
```

sintax:

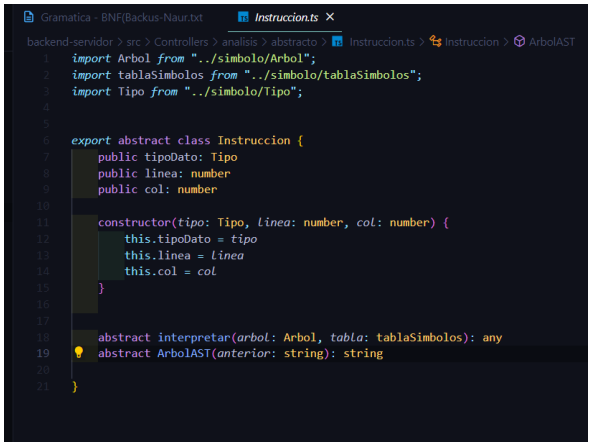
Se realizo las producciones y la gramática



```
%left 'PUNTO'


// simbolo inicial
%start INICIO

%%

INICIO : INSTRUCCIONES EOF                  {return $1;}
;

INSTRUCCIONES : INSTRUCCIONES INSTRUCCION   {$1.push($2); $$=$1;}
              | INSTRUCCION                 {$$=[$1];}
;

INSTRUCCION : IMPRESION             {$$=$1;}
            | DECLARACION PUNTOCOMA         {$$=$1;}
            | ASIGNACION PUNTOCOMA          {$$=$1;}
            | FUNIF                         {$$=$1;}
            | FUNWHILE                      {$$=$1;}
            | FUNBREAK                      {$$=$1;}
            | FUNCONTINUE                       {$$=$1;}
            | FUNDOWHILE    PUNTOCOMA                   {$$=$1;}
            | FUNFOR                          {$$=$1;}
            |FUNSWITCH                        {$$=$1;}
            |DECARREGLO   PUNTOCOMA           {$$=$1;}
            |MODFVECTOR PUNTOCOMA            {$$=$1;}
            |MOFVECTOR2 PUNTOCOMA            {$$=$1;}
            |DECARRELGO2DIMEN  PUNTOCOMA {$$=$1;}
            |METODO  {$$=$1;}
            |FUNCION {$$ =$1;}
            |EXECUTEN PUNTOCOMA  {$$=$1;}
```

Clases abstractas:

Para poder realizar el proyecto realizamos las clases abstractas, en cada función para el programa



Clases arbol:

Con esta clase se creo el árbol para poder manejar los datos

Clase tabla de simbol

Se creo la clase de símbolo para guardar los datos y poder manejarlo

Crear el Ast:

Se creo la siguiente clase stactic para poder manejarlo en todo nuestro proyecto

❖ **My-app:**

Clases app.js:

Se creo primero las funcionalidades de cada botón

```
import React, { useEffect, useState, useRef } from "react";
import './App.css';
import Editor from '@monaco-editor/react';
import { Graphviz } from 'graphviz-react';

function App() {
    const editorRef = useRef(null);
    const consolaRef = useRef(null);
    const [AST, obtenerAst] = useState("");

    const [archivos, setArchivos] = useState([]); // Estado para mantener los archivos
    const [archivoActual, setArchivoActual] = useState(null); // Estado para mantener

    function handleEditorDidMount(editor, id) {
        if (id === "editor") {
            editorRef.current = editor;
        } else if (id === "consola") {
            consolaRef.current = editor;
        }
    }

    function reporteAST(){
        fetch('http://localhost:4000/getAST', {
          method: 'GET',
          headers: {
            'Content-Type': 'application/json',
          },
        })
          .then(response => response.json())
          .then(data => {
            obtenerAst(data.AST);
            console.log(data.AST);
```

Se realizo la estructura del proyecto en html

```
return (
    <div className="App">
        <nav>
            <label className="logo">CompiScript+</label>
            <ul>
                <li><a href="#" onClick={crearArchivoEnBlanco}>Crear archivo en blanco</a></li>    The href attribute requires a valid valu
                {archivos.map(archivo => (
                    <li key={archivo.nombre}><a href="#" onClick={() => abrirArchivo(archivo.nombre)}>{archivo.nombre}</a></li>    The href
                ))}
                <li><label className="nav-link" htmlFor="fileInput">Abrir Archivos</label>
        <input id="fileInput" type="file" accept=".sc" style={{ display: 'none' }} onChange={arbir_archivo_nuevo} /></li>
                <li><a href="#" onClick={guardarArchivo}>Guardar archivo</a></li>    The href attribute requires a valid value to be access
                <li><a href="#" onClick={interpretar}>Ejecutar</a></li>    The href attribute requires a valid value to be accessible. Prov
                <li><a href="#">Reportes</a></li>    The href attribute requires a valid value to be accessible. Provide a valid, navigable
            </ul>
        </nav>

        <div className="editors-container">
            <h1>Entrada</h1>
            <div className="editor">
                <Editor height="90vh" width="100%" defaultLanguage="java" defaultValue="" theme="vs" onMount={(editor) => handleEditorDidMo
            </div>

            <h1>Consola</h1>
            <div className="editor-consola">
                <Editor height="90vh" width="100%" defaultLanguage="cpp" defaultValue="" theme="vs" options={{ readOnly: true }} onMount={(
            </div>
        </div>
```

Clases app.css:

Se creo todo el estilo de la pagina