Universidad de San Carlos de Guatemala

Organización de lenguajes y compiladores 1

Sección “C”

Manual técnico

Nombres y Apellidos	Carné
José David Góngora Olmedo	202201444

Índice

Descripción 3
Lenguaje 3
Herramientas 3
Jison.....3
1.1 Backend 4
1.1.1 Build 5
1.1.2 nodo_modules.....5
1.1.3 src.....6
1.1.3.1 Index controller 6
1.1.3.2 Index router.....7
1.1.4 Analizador léxico.....7
1.1.5 Analizador sintáctico 8
1.2 Frontend.....9
1.2.1 Componentes 9

Descripción

Proyecto que consiste en crear un lenguaje de programación, un intérprete sencillo, con las funcionalidades principales para que sea funcional.

Lenguaje

- Javascript
- Typescript

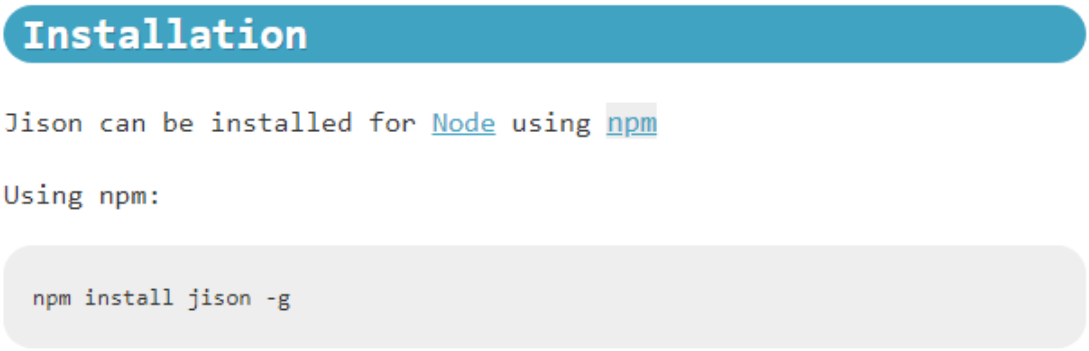
Herramientas

Se utilizaron librerías para el desarrollo del sistema para un mejor funcionamiento.

Jison

Jison toma como entrada una gramática libre de contexto y genera un archivo JavaScript capaz de analizar el lenguaje descrito por dicha gramática. A continuación, puede utilizar el script generado para analizar entradas y aceptar, rechazar o realizar acciones basadas en la entrada. Si está familiarizado con Bison o Yacc, u otros clones, está casi listo para empezar.

Instalación

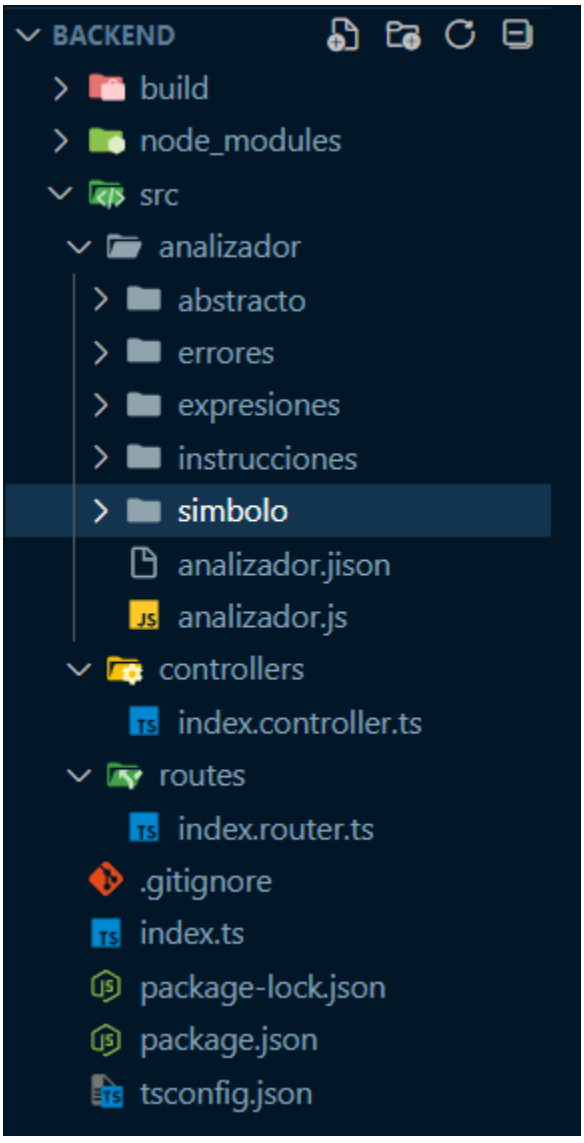


1. Proyecto

El proyecto está estructurado por paquetes o carpetas y está dividido por Backend y Frontend, donde cada carpeta contiene los archivos para cada parte del programa.

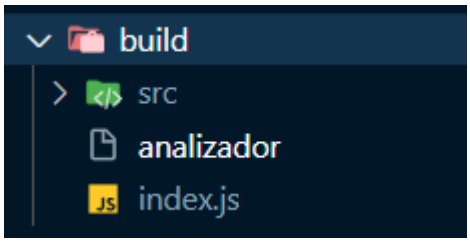
1.1 Backend

Esta carpeta contiene todos los archivos para el analizador y donde está toda la lógica.



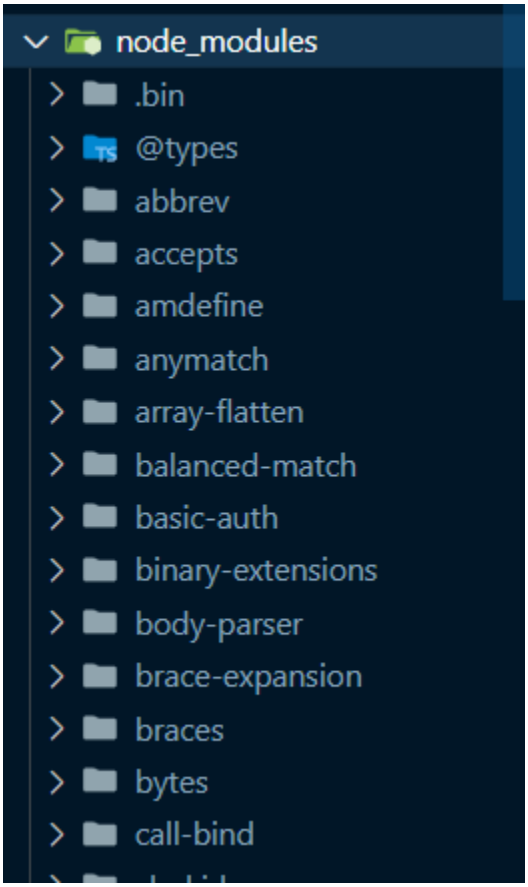
1.1.1 Build

Esta carpeta contiene todos los archivos ts pero parseados a js para poder ser ejecutados.



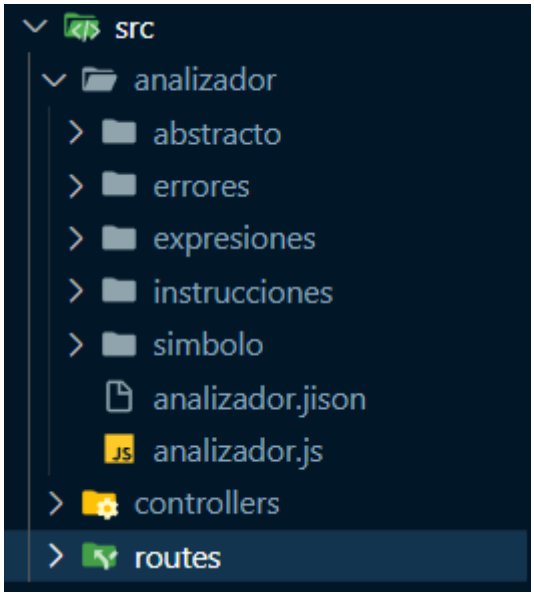
1.1.2 nodo_modules

Esta carpeta contiene toda la información de las librerías utilizadas.



1.1.3 src

Esta carpeta contiene toda la lógica del analizador, y sus carpetas donde se guardan las expresiones, instrucciones y todas las clases del Interprete.



1.1.3.1 Index controller

En este archivo se manejan todas las peticiones para hacerle al backend y devolvemos una salida o recibimos una entrada.

```
export let lista_errores: Array<Errores> = []
export let dot: string = ""

class Controller {

    public analizar(req: Request, res:Response) {
        lista_errores = new Array<Errores>
        try {
            let parser = require('../analizador/analizador.js')
            let ast = new Arbol(parser.parse(req.body.entrada))
            let tabla = new TablaSimbolos()
            tabla.setNombre("Tabla simbolos")
            ast.setTablaGlobal(tabla)
            ast.setConsola("")
            let execute = null

            let contador = Cont.getInstancia()

            dot = "digraph ast{\n"
            dot += "nINICIO[label=\"INICIO\"];\n"
            dot += "nINSTRUCCIONES[label=\"INSTRUCCIONES\"];\n"
            dot += "nINICIO->nINSTRUCCIONES;\n"

            for (let error of lista_errores) {
                ast.actualizarConsola((<Errores>error).obtenerError())
            }

            for(let i of ast.getInstrucciones()) {
                if(i instanceof Metodo || i instanceof Funcion) {
                    i.id = i.id.toLocaleLowerCase()
                    ast.addFuncion(i)
                }
            }
```

1.1.3.2 Index router

En este archivo se manejan todas las rutas para hacer las peticiones al backend.

```
class router {
    public router: Router = Router()
    constructor() {
        this.config()
    }

    config(): void {
        this.router.post('/analizar', indexController.analizar)
        this.router.get('/obtenerErrores', indexController.getErrores)
        this.router.get('/getAST', indexController.getAST)
    }
}

const indexRouter = new router()
export default indexRouter.router
```

1.1.4 Analizador léxico

El analizador léxico es donde se verifican los lexemas de la entrada y se detectan los que no pertenecen al lenguaje.

```
%lex

%options case-insensitive
%x string

%%

\s+ // OMITE ESPACIOS

/* COMENTARIOS */

\/\/.* {}  // COMENTARIO UNA LINEA
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/] {} // COMEN

/* PALABRAS RESERVADAS */
"new" return "NEW";
"if" return "IF";
"else" return "ELSE";
"switch" return "SWITCH";
"case" return "CASE";
"default" return "DEFAULT";
"while" return "WHILE";
"for" return "FOR";
"do" return "DO";
"break" return "BREAK";
"continue" return "CONTINUE";
"return" return "RETURN";
```

Ejemplo de expresiones regulares

1.1.5 Analizador sintáctico

El analizador sintáctico es el que se encarga de validar el orden de los tokens si cumple con la gramática o sintaxis del lenguaje a analizar.

Ejemplo de una gramática para aceptar las sentencias del lenguaje:

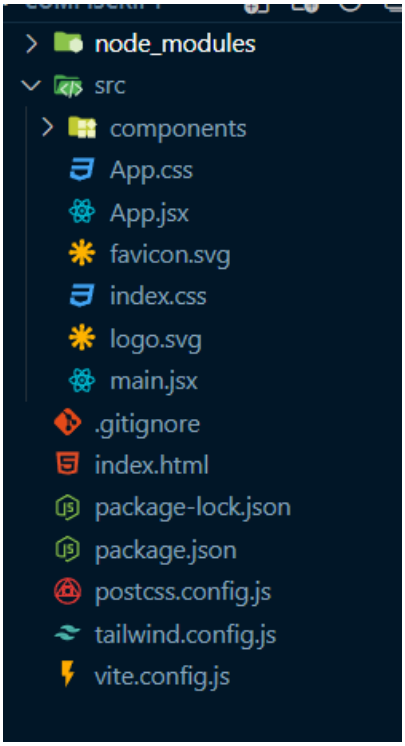
Crear árbol y agregar hijos para el análisis.

```
inicio : instrucciones EOF { return $1 }
;

instrucciones : instrucciones sentencias { if($2 != false) $1.push($2); $$ = $1 }
              | sentencias { $$ = ($1!= false) ? [$1] : []}
;

sentencias : declaracion { $$ = $1 }
           | imprimir { $$ = $1 }
           | asignacion PYC { $$ = $1 }
           | incre_decre PYC { $$ = $1 }
           | if_s { $$ = $1 }
           | while_s { $$ = $1 }
           | break_s { $$ = $1 }
           | continue_s { $$ = $1 }
           | do_while_s { $$ = $1 }
           | for_s { $$ = $1 }
           | switch_s { $$ = $1 }
           | vector_ud { $$ = $1 }
           | modificar_vud { $$ = $1 }
           | vector_dd { $$ = $1 }
```

Así mismo se puede observar cómo se trabaja el árbol sintáctico, donde se almacena cada árbol con sus hijos para un análisis más óptimo y fácil.

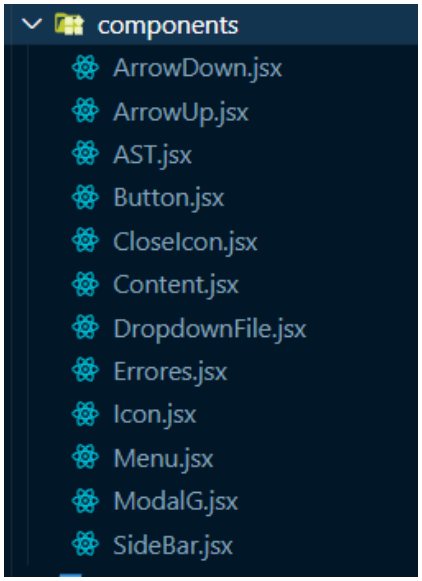
1.2 Frontend

En esta carpeta se encuentran todos los archivos para el cliente.



1.2.1 Componentes

En esa carpeta se encuentran todos los componentes que conforman la vista.



1.2.2 Peticiones

Asi se hacen las peticiones desde el frontend para recuperar las respuestas del backend.

```
const interpretar = () => {
    var entrada = editorRef.current.getValue()
    fetch('http://localhost:4000/analizar', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({ entrada: entrada }),
    }).then(response => response.json())
    .then(data => {
        consolaRef.current.setValue(data.respuesta)
        setDot(data.ast)
        console.log(data.ast)
        // setErrores(data.lista_errores)
    })
    .catch((error) => {
        alert("No sale compi")
        console.error('Error:', error)
    })
}
```