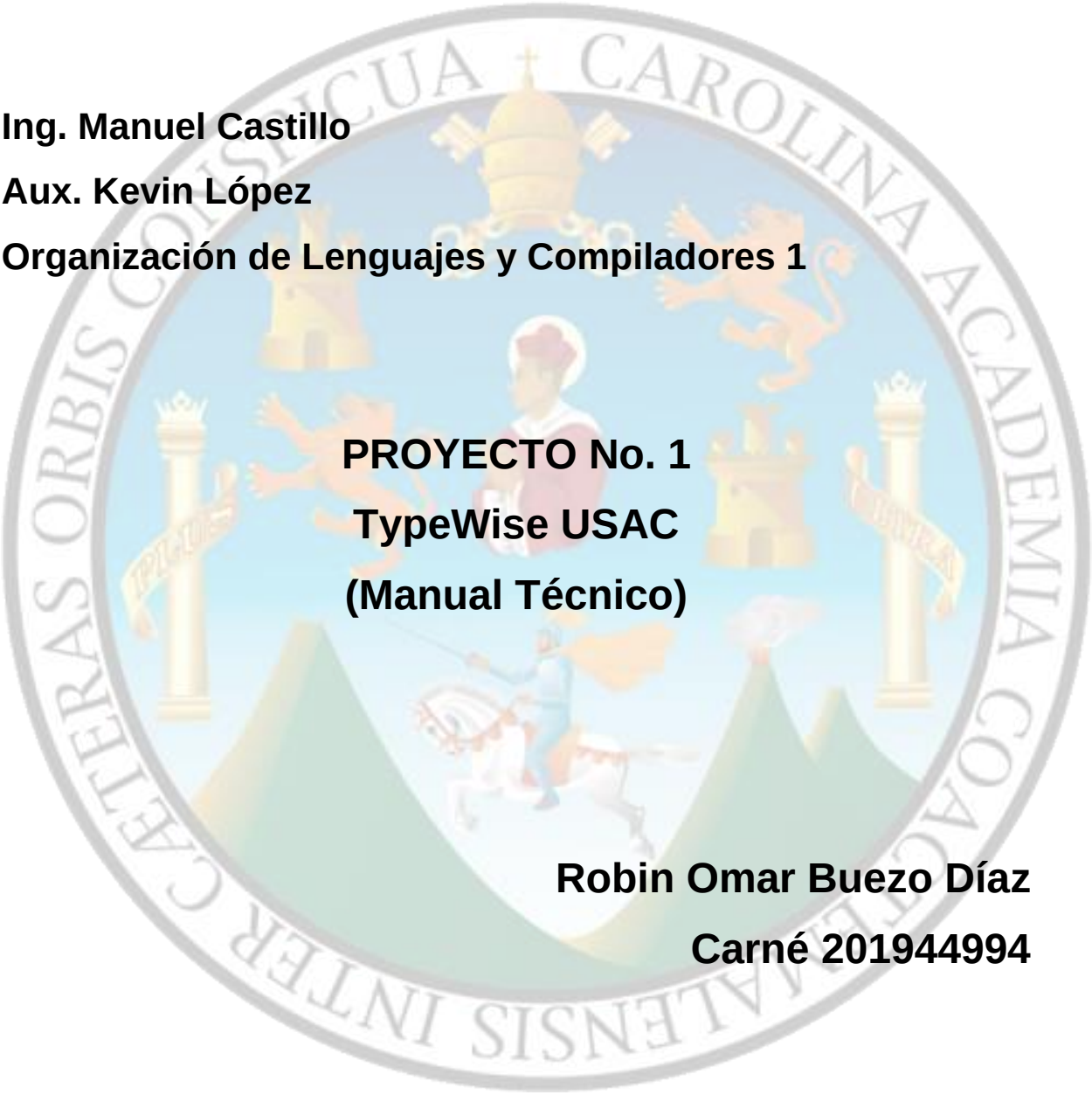
Universidad de San Carlos de Guatemala -USAC-

Facultad de Ingeniería

Ingeniería en Ciencias y Sistemas



Ing. Manuel Castillo

Aux. Kevin López

Organización de Lenguajes y Compiladores 1

PROYECTO No. 1

TypeWise USAC

(Manual Técnico)

Robin Omar Buezo Díaz

Carné 201944994

Miércoles 03 de mayo de 2023.

El presente documento tiene como finalidad mostrar al usuario administrador la funcionalidad y construcción del software para que entienda su creación y pueda dar solución a cualquier error que pueda presentarse.

Se explican el flujo y las diferentes partes que constituyen el software y como debemos de interactuar con este para poder sacarle el máximo provecho de forma óptima.

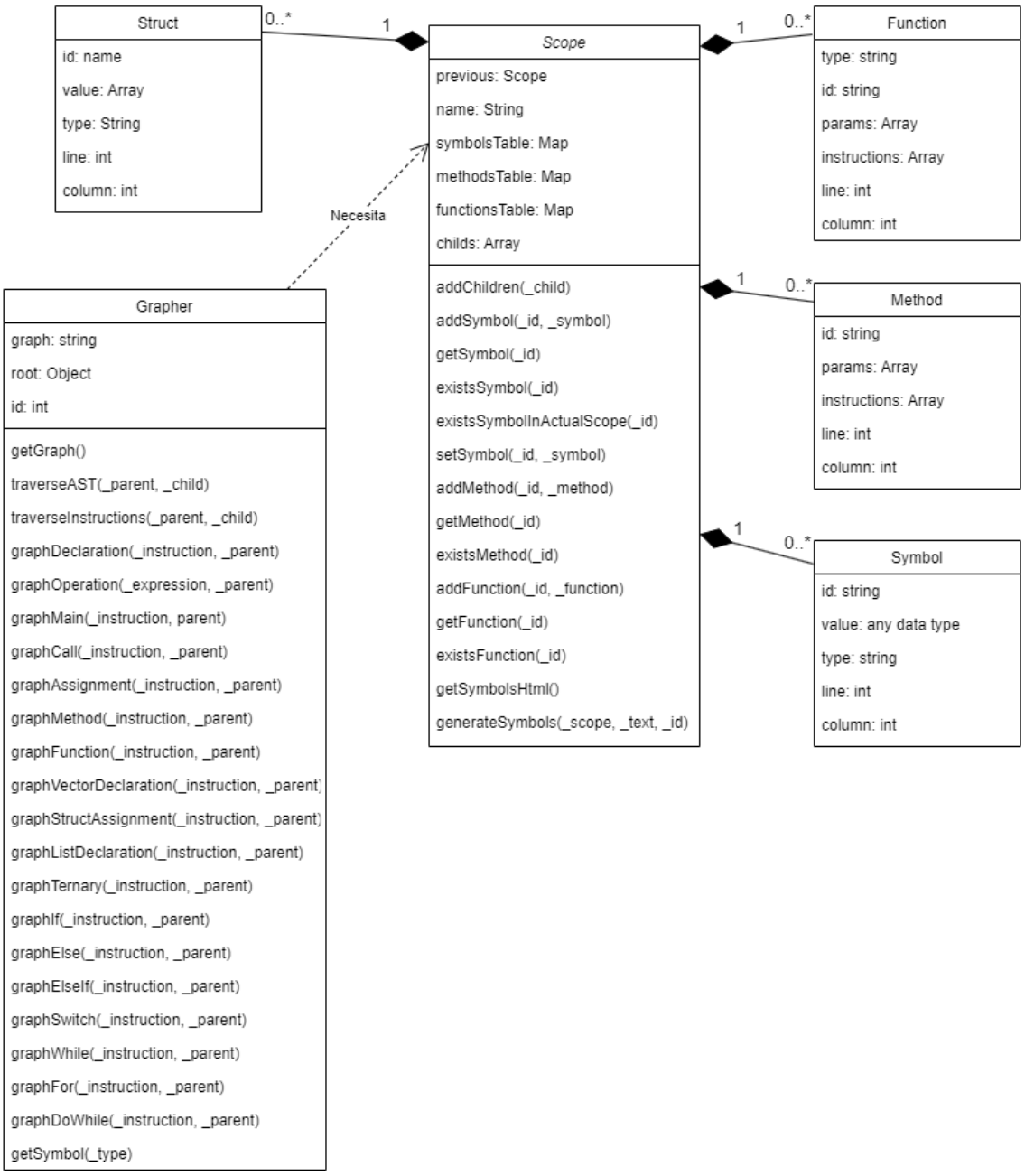
PARADIGMA DE PROGRAMACIÓN

Para la creación de este software se utilizó el paradigma de Programación Orientada a Objetos, ya que esto da una mejor facilidad a la hora de manejar el archivo que se está manipulando a lo largo de toda la ejecución, como también el poder encapsular los objetos y luego poder utilizar los mismos objetos en las diferentes ventanas que se utilizan a lo largo del programa.

NOMENCLATURA

Identificador	Regla	Ejemplo
Clases	Sustantivos, primera letra mayúscula	MyClass class
Funciones	Verbos, primera letra de cada palabra en mayúscula	ejecutar() cargarArchivo()
Atributos de clases	Minúsculas	atributo1 atributo2
Variables	En minúsculas y deben empezar con una letra	variableuno variabledos

DIAGRAMA DE CLASES



TOKENS

Rint = int

Rdouble = double

Rboolean = boolean

Rchar = char

Rstring = string

inc = ++

dec = --

sum = +

sub = -

mul = *

div = /

pow = ^

mod = %

tern = ?

equals = ==

diff = !=

lessEq = <=

less = <

greaterEq = >=

greater = >

colon = :

and = &&

or = ||

not = !

parLeft = (

parRight =)

semiColon = ;

oBracke = {

cBracke = }

same = =

comma = ,

dot = .

oDouSquare = [[

cDouSquare =]]

oSquare = [

cSquare =]

Rif = if

Relse = else

Rswitch = switch

Rcase = case

Rdefault = default

Rwhile = while

Rfor = for

Rdo = do

Rbreak = break

Rcontinue = continue

Rreturn = return

Rvoid = void

Rprint = print

Rtrue = true

Rfalse = false

Rmain = main

Rlist = list

Rnew = new

Radd = add

RtoLower = toLower

RtoUpper = toUpper

Rlength = length

Rtruncate = truncate

Rround = round

Rtypeof = typeof

RtoString = toString

RtoCharArray = toCharArray

id = [a-zA-Z][a-zA-Z0-9_]*

string = ["'][^"']["']

double = [0-9]+(.[0-9]+)\b

int = [0-9]+\b

commentary = \s+|//.|/*[^]+([^/][^]*+)*/

GRAMÁTICA

<INI> ::= <INSTRUCTIONSBODY> EOF

<INSTRUCTIONSBODY> ::= <INSTRUCTIONSBODY> <BODY>

| <BODY>

<BODY> ::= <DEC_VAR> ';' | <ASIG_VAR> ';' | <METHODS> | <MAIN>

| <DEC_STRUCT> | <SET_STRUCT> | error ';'

<METHODS> ::= Rvoid id '(' ')' '{' <INSTRUCTIONS> '}'

| Rvoid id '(' <PARAMS> ')' '{' <INSTRUCTIONS> '}'

| <TYPE> id '(' ')' '{' <INSTRUCTIONS> '}'

| <TYPE> id '(' <PARAMS> ')' '{' <INSTRUCTIONS> '}'

<PARAMS> ::= <PARAMS> ',' <PARAM> | <PARAM>

<PARAM> ::= <TYPE> id

<CALL> ::= id '(' ')' | id '(' <PARAMS_CALL> ')'

<MAIN> ::= Rmain id '(' ')' ';' | Rmain id '(' <PARAMS_CALL> ')' ';'

<PARAMS_CALL> ::= <PARAMS_CALL> ',' <EXPRESSION> | <EXPRESSION>

<DEC_VAR> ::= <TYPE> id | <TYPE> id same <EXPRESSION>

<ASIG_VAR> ::= id same <EXPRESSION> | id inc | id dec

<TYPE> ::= Rint | Rdouble | Rchar | Rboolean | Rstring

<INSTRUCTIONS> ::= <INSTRUCTIONS> <INSTRUCTION> | <INSTRUCTION>

<INSTRUCTION> ::= <DEC_VAR> | <ASIG_VAR> | <DEC_STRUCT>

| <SET_STRUCT> | <PRINT> | <IF> | <SWITCH> | <WHILE> | <FOR>

| <DO_WHILE> | <CALL> | Rbreak ';' | Rcontinue ';' | <RETURN> | error ';'

<PRINT> ::= Rprint '(' <EXPRESSION> ')' ';'

<IF> ::= Rif '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}'

| Rif '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}' Relse '{' <INSTRUCTIONS> '}'

| Rif '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}' <ELSEIF>

| Rif '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}' <ELSEIF> Relse '{' <INSTRUCTIONS> '}'

<ELSEIF> ::= <ELSEIF> <EIF> | <EIF>

<EIF> ::= Relse Rif '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}'

<SWITCH> ::= Rswitch '(' <EXPRESSION> ')' '{' <CASES> <DEFAULT> '}'

| Rswitch '(' <EXPRESSION> ')' '{' <CASES> '}'

| Rswitch '(' <EXPRESSION> ')' '{' <DEFAULT> '}'

<CASES> ::= <CASES> <CASE> | <CASE>

<CASE> ::= Rcase <EXPRESSION> ':' <INSTRUCTIONS>

<DEFAULT>: Rdefault ':' <INSTRUCTIONS>

<WHILE> ::= Rwhile '(' <EXPRESSION> ')' '{' <INSTRUCTIONS> '}'

<FOR> ::= Rfor '(' <DEC_VAR> ';' <EXPRESSION> ';' <ASIG_VAR> ')' '{'

<INSTRUCTIONS> '}'

| Rfor '(' <ASIG_VAR> ';' <EXPRESSION> ';' <ASIG_VAR> ')' '{' <INSTRUCTIONS> '}'

<DO_WHILE> ::= Rdo '{' <INSTRUCTIONS> '}' Rwhile '(' <EXPRESSION> ')' ';'

<RETURN> ::= Rreturn <EXPRESSION> ';' | Rreturn ';'

<DEC_STRUCT> ::= <TYPE> '[' ']' id same Rnew <TYPE> '[' <EXPRESSION> ']' ';'

| <TYPE> '[' ']' id same '{' <LIST_VALUES> '}' ';'

|Rlist '<' <TYPE> '>' id same Rnew Rlist '<' <TYPE> '>' ';'

|Rlist '<' <TYPE> '>' id same <EXPRESSION> ';'

<LIST_VALUES> ::= <LIST_VALUES> ',' <EXPRESSION> | <EXPRESSION>

<SET_STRUCT> ::= id '[' <EXPRESSION> ']' same <EXPRESSION> ';'

| id '.' Radd '(' <EXPRESSION> ')' ';'

| id '[[' <EXPRESSION> ']]' same <EXPRESSION> ';'

<EXPRESSION> ::= <EXPRESSION> '?' <EXPRESSION> ':' <EXPRESSION>

| <EXPRESSION> '+' <EXPRESSION>

| <EXPRESSION> '-' <EXPRESSION>

| <EXPRESSION> '*' <EXPRESSION>

| <EXPRESSION> '/' <EXPRESSION>

| <EXPRESSION> '^' <EXPRESSION>

| <EXPRESSION> '%' <EXPRESSION>

| <EXPRESSION> '<' <EXPRESSION>

| <EXPRESSION> '>' <EXPRESSION>

| <EXPRESSION> '<=' <EXPRESSION>

| <EXPRESSION> '>=' <EXPRESSION>

| <EXPRESSION> '==' <EXPRESSION>

| <EXPRESSION> '!=' <EXPRESSION>

| <EXPRESSION> '&&' <EXPRESSION>

| <EXPRESSION> '||' <EXPRESSION>

| '(' <TYPE> ')' <EXPRESSION>

| RtoLower '(' <EXPRESSION> ')'

| RtoUpper '(' <EXPRESSION> ')'

| Rlength '(' <EXPRESSION> ')'

| Rtruncate '(' <EXPRESSION> ')'

| Rround '(' <EXPRESSION> ')'

| Rtypeof '(' <EXPRESSION> ')'

| RtoString '(' <EXPRESSION> ')'

| RtoCharArray '(' <EXPRESSION> ')'

| '!' <EXPRESSION>

| '-' <EXPRESSION>

| '(' <EXPRESSION> ')'

| <CALL>

| id '[' <EXPRESSION> ']'

| id '[[' <EXPRESSION> ']]'

| double

| int

| Rtrue

| Rfalse

| string

| id

| char

METODOS PRINCIPALES

Assignment():

Este método se encarga de generar la lógica para guardar las instrucciones de una asignación.

Block():

Este método se encarga de generar la lógica ejecutar cada una de las instrucciones que puedan venir en un bloque de código de cada ámbito.

Call():

Este método se encarga de generar la lógica para guardar las instrucciones de una llamada ya sea a un método o una función.

DecFunction():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración de función.

Declaration():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración de variable.

DecMethod():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración de método.

DecParam():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración de parámetros.

StatementDoWhile():

Este método se encarga de generar la lógica para guardar las instrucciones de un ciclo Do While.

StatementFor():

Este método se encarga de generar la lógica para guardar las instrucciones de un ciclo For.

Global():

Este método se encarga de generar la lógica para guardar las instrucciones del ámbito global.

StatementIf():

Este método se encarga de generar la lógica para guardar las instrucciones de una sentencia If.

StatementIfElse():

Este método se encarga de generar la lógica para guardar las instrucciones de una sentencia If con Else.

StatementIfElseIf():

Este método se encarga de generar la lógica para guardar las instrucciones de una sentencia if que contenga If Else o también Else.

Instruction:

Este módulo se encarga de proporcionar las funciones al analizador sintáctico para crea el objeta de cada una de las instrucciones que puedan venir.

List():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración de Lista.

Main():

Este método se encarga de generar la lógica para guardar las instrucciones de una declaración del main.

Print():

Este método se encarga de generar la lógica para ejecutar la instrucción print en la consola.

Return():

Este método se encarga de generar la lógica para ejecutar la instruccion return.

StatementSwitch():

Este método se encarga de generar la lógica para guardar las instrucciones de una sentencia switch.

Vector():

Este método se encarga de generar la lógica para guardar las instrucciones que puedan corresponder a un Vector.

StatementWhile():

Este método se encarga de generar la lógica para guardar las instrucciones de un ciclo while.

Arithmetic():

Este método se encarga de generar la lógica para ejecutar las distintas operaciones aritméticas que el programa requiera.

Cast():

Este método se encarga de generar la lógica para ejecutar la operación cast cuando el programa requiera.

ExpressionValue():

Este método se encarga de retornar el valor de cualquier variable que el sistema requiera.

Logical():

Este método se encarga de generar la lógica para ejecutar las distintas operaciones lógicas que el programa requiera.

Native():

Este método se encarga de generar la lógica para ejecutar las distintas funciones nativas con las que el intérprete cuenta.

Operation():

Este método se encarga de direccionar la operación que el sistema solicita a cada una de las funciones correspondientes para la realización de dicha operación.

OpString():

Este método se encarga de retornar la cadena del valor solicitado por la función Print.

Relational():

Este método se encarga de generar la lógica para ejecutar las distintas operaciones relacionales que el programa requiera.

ResultType():

Este método se encarga de retornar el tipo de resultado que la operación tendrá en función de sus valores a operarse.

Ternary():

Este método se encarga de generar la lógica para ejecutar la función Ternaria cada vez que el programa requiera.

Grapher.getGraph():

Este método se encarga de generar el reporte del árbol AST según los datos recopilados por el intérprete.

Scope.addChildren():

Este método se encarga de agregar un hijo a la lista de hijos del objeto Scope.

Scope.addSymbol():

Este método se encarga de agregar un símbolo a la tabla de símbolos del objeto Scope.

Scope.existsSymbol():

Este método se encarga de verificar la existencia de un símbolo en el objeto Scope o en los padres de este.

Scope.existsSymbolInActualScope():

Este método se encarga de verificar la existencia de un símbolo solamente en el propio objeto Scope.

Scope.addMethod():

Este método se encarga de agregar el método a la Tabla de Métodos del objeto Scope.

Scope.existsMethod():

Este método se encarga de verificar la existencia de un método en el objeto Scope o en los padres de este.

Scope.addFunction():

Este método se encarga de agregar la función a la Tabla de Funciones del objeto Scope.

Scope.existsFunction():

Este método se encarga de verificar la existencia de una función en el objeto Scope o en los padres de este.

Scope.getSymbolsHtml():

Este método se encarga de retornar el código html de la página que muestra la tabla se símbolos.

ENDPOINTS PRINCIPALES

analyzer(Post):

Este endpoint se encarga de ejecutar el intérprete en el código trasladado y generar el árbol AST como también los errores en caso de existir.

graphAST(Get):

Este endpoint se encarga de llamar a la función encargada de generar el reporte del árbol AST.

generateSymbols(Get):

Este endpoint se encarga de llamar a la función encargada de generar el archivo html que contiene la tabla de símbolos.

generateErrors(Get):

Este endpoint se encarga de generar el archivo html que contiene la tabla de errores.

HERRAMIENTAS

Para poder dar solución a los requerimientos anteriores se utilizó el lenguaje de programación JavaScript en conjunto con Jison para generar el backend de nuestro programa haciendo uso de su documentación por su versatilidad, fácil programación y su popularidad a nivel mundial. También se utilizó React con un poco de html para poder generar el frontend de nuestro programa.

Como herramienta de programación se utilizó Visual Studio Code por su amplia funcionalidad y las herramientas que brinda a los programadores al momento de programar con el lenguaje Java.

Para poder realizar los reportes se utilizó la herramienta Graphviz por facilidad para poder realizar gráficas y diagramas bastante profesionales y entendibles por medio de la programación.

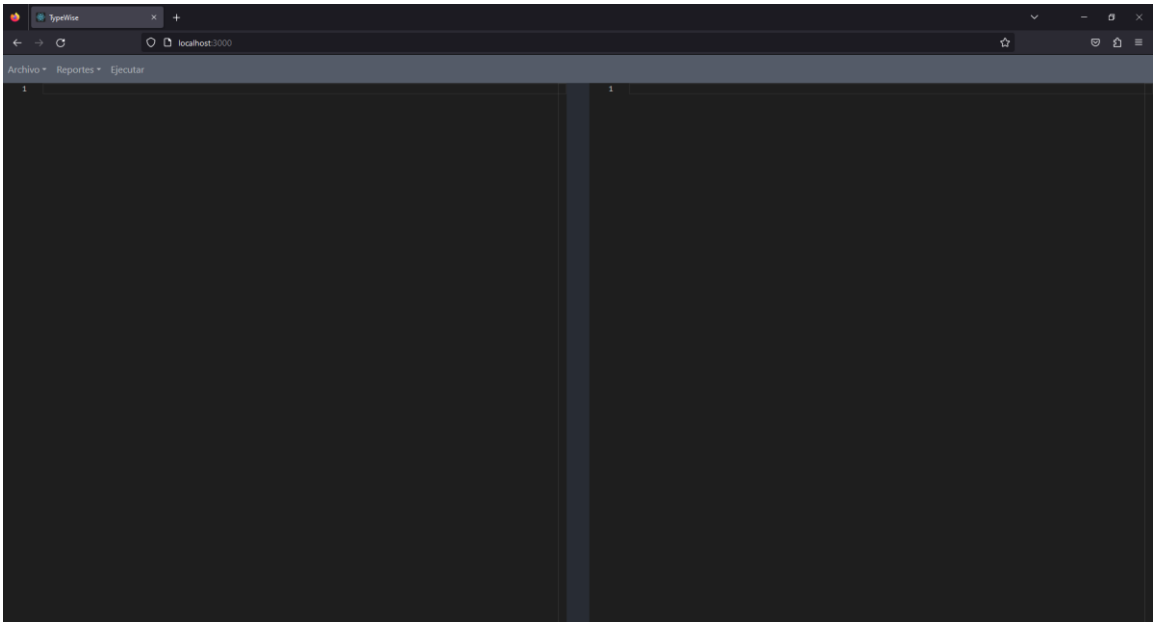
Para poder realizar los diagramas se utilizó la herramienta en línea diagram.net por sus plantillas que nos facilitan el hacer diagramas más profesionales.

Por último, se utilizó la herramienta de versionamiento GitHub. Para poder tener un mejor control sobre los cambios que se iban realizando en nuestro código y no tener el problema de perder funcionalidad si en caso algún cambio ocasionaba erros.

GUI PRINCIPALES

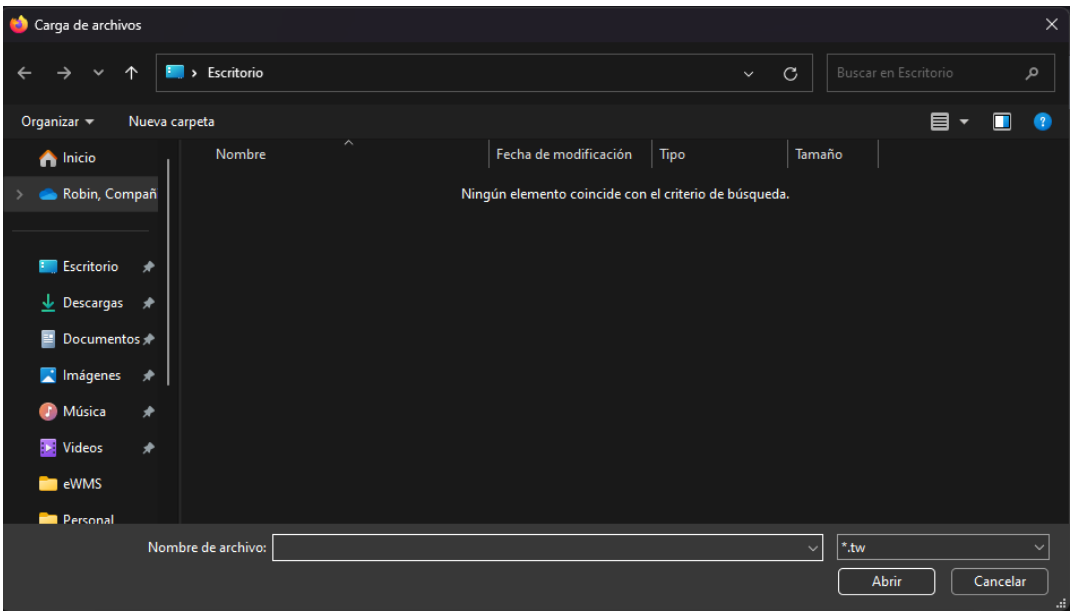
Ventana Principal

Esta es la ventana principal y en donde tendremos a nuestro editor de texto para poder manipular nuestro código, como también nuestra barra de menú para el resto de las opciones.



Abrir

Esta opción nos abrirá una ventana emergente desde donde podremos buscar y seleccionar el archivo que queremos cargar.



GLOSARIO

HTML: Es un lenguaje de marcado que nos permite indicar la estructura de nuestro documento mediante etiquetas.

TW: Extensión de los archivos que reconoce y genera el intérprete TypeWise.