Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Proyecto 1

C# to Python

Manual Técnico

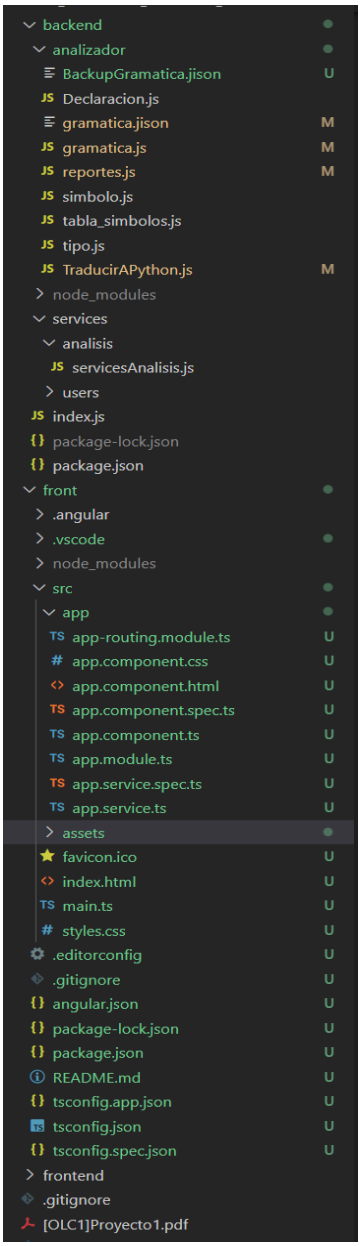
Erick Ivan Mayorga Rodríguez

201901758

Entorno de Desarrollo

- Lenguajes:
  JavaScript
- IDE: Visual Studio Code
- Librería para gramática: JISON

Estructura:

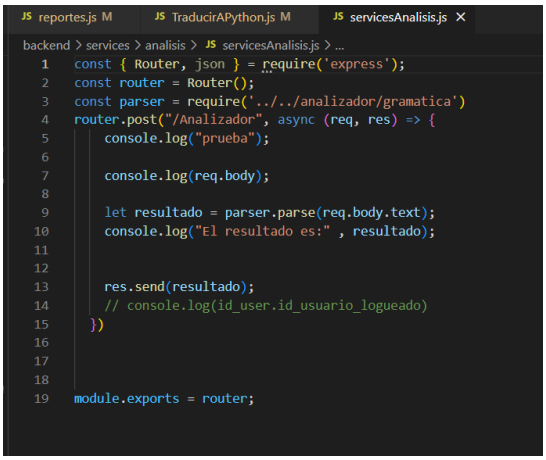


El proyecto en cuestión se divide en dos partes principales, FrontEnd y backend, como se puede inferir por su nombre, el backend contiene la gramática y todos los servicios a los cuales se va a consultar información, además de la lógico de traducción a implementar en la solución.

Del lado de FrontEnd se consumen los EndPoints de la aplicación de backend y se manda información por medio de la vista principal de las consolas.

Entre las clases principales para el entendimiento de la lógica del proyecto se encuentran:

- ServicesAnalisis: en esta clase se encuentra en el directorio backend/services/análisis, en esta se encuentra el endpoint hacia el cual se va a realizar la consulta para mandar la información que va a ser analizada por la gramática.



- Gramatica.jison: En esta clase como su nombre indica, se encuentra toda la información referente a la gramática.

En dicha clase se encuentran los objetos javascript utilizados para el análisis lexico y semántico.

```
/**
 * Ejemplo mi primer proyecto con Jison utilizando Nodejs en Ubuntu
 */

/* Definición Léxica */


%{
    const TraducirAPython = require('./TraducirAPython.js')
    const Reportes = require('./reportes.js');
    const Declaracion = require('./Declaracion.js');
    const SymbolTable = require('./tabla_simbolos.js');
    const Type = require('./tipo.js')
    var reportes = new Reportes();
    var tabla_simbolo = new SymbolTable(null);
    tabla_simbolo.reportes = reportes;

    var instrucciones = [];
    var traducir = new TraducirAPython();

%}

%lex
%options case-insensitive
%%
```

Los tokens recibidos para el análisis léxico

```
[\n\t\s\r]+                         {} // Espacios en blanco
"//".*                              {} // Comentario de una linea //
[/][*][^*]*[*]+([^/*][^*]*[*]+)*[/]     {} // Comentario multilinea

//Reservadas res == reservada

"++"                return 'incremento'
"--"                return 'reduccion'
"int"               return 'int';
"double"            return 'double';
"char"              return 'char';
"bool"              return 'bool';
"string"            return 'resstring';
"void"              return 'void';

"null"              return 'null';
"integer"           return 'integer';

"true"              return 'true';
"false"             return 'false';

"if"                return 'if_sen';
"else"              return 'else';
"switch"            return 'switch'
"for"               return 'for';
"while"             return 'while';
"do"                return 'do';
"Console.Write"     return 'imprimir';
"return"            return 'return';
"continue"          return 'continue';
"case"              return "case"
"default"           return "default"
"break"             return "break"

"+"                 return 'mas';
"-"                 return 'menos';
"*"                 return 'por';
"/"                 return 'dividido';

"&&"                return 'and'
"||"                return 'or'
"!"                 return 'not'
">"                 return 'mayor'
"<"                 return 'menor'
">="                return 'mayorigual'
"<="                return 'menorigual'
"=="                return 'igualigual'
"!="                return 'diferente'
```

```
//simbolos
"{"             return 'llaveaper';
"}"             return 'llavecierre';
"("             return 'parentesisaper';
")"             return 'parentesiscierre';
","             return 'coma';
"."             return 'punto';
"="             return 'igual';
";"             return 'puntocoma'
":"             return 'dospuntos'



/* Espacios en blanco */


([a-zA-Z_])[a-z0-9A-Z_ñÑ]*      return 'id';
[0-9]+("."[0-9]+)\b         return 'decimal'
[0-9]+\b                    return 'entero'

\'((\\\')|[^\n\'])\'    { yytext = yytext.substr(1,yyleng-2); return 'caracter'; }

[\"]("\\n"|"\\r"|"\\t"|"\\'"|"\\\""|"\\\\"|[^\"])*[\"]   {yytext = yytext.substring(1,yytext.length-1)
                                                          yytext = yytext.replace(/\\n/g, '\n')
                                                          yytext = yytext.replace(/\\r/g, '\r')
                                                          yytext = yytext.replace(/\\t/g, '\t')
                                                          yytext = yytext.replace(/\\\'/g, '\'')
                                                          //yytext = yytext.replace(/\\\"/g, '\"')
                                                          //yytext = yytext.replace(/\\\\/g, '\\')
                                                          return'cadena';}


<<EOF>>                 return 'EOF';

.                       { reportes.putError({lexema:yytext, fila: this._$.first_line, columna:this._$.first_column, tipo: "LEXICO" }) }
/lex
```

Y la gramática creada para el posterior análisis sintáctico, el cual realiza la tarea de traducir el código de C# a Python.

```
INI
    : INSTRUCCIONES EOF {
        instrucciones = $1;
        let resultado = ""
        for (var i = 0; i < instrucciones.length; i++){
            resultado += $1[i]
        }

        return {
            salida: resultado,
            errores: reportes,
            consolaPrint: traducir.contenidoPrint,
        };
    }
;

INSTRUCCIONES
    : INSTRUCCIONES INSTRUCCION  {$$ = $1; $$.push($2);}
    | INSTRUCCION  {$$ = [$1];}
    | error  { console.error('Este es un error sintáctico: ' + yytext + ', en la linea: ' + this._$.first_line + ', en la columna: ' + this._$.first_
            reportes.putError({lexema:yytext, fila: this._$.first_line, columna:this._$.first_column, tipo: "SINTACTICO" })
           }
;

INSTRUCCION
    : DECLARACION_VAR  puntocoma    {$$ = $1 + "\n"; console.log("declarar variable", $1);}
    | DECLARACION_FUNCION           {$$ = $1 + "\n";}
    | DECLARACION_METODO            {$$ = $1 + "\n";}
    | ASIGNACIONVAR puntocoma       {$$ = $1;}
    | IF_INS                        {$$ = $1;}
    | SWITCH                        {$$ = $1;}
    | FOR                           {$$ = $1;}
    | WHILE                         {$$ = $1;}
    | PRINT puntocoma               {$$ = $1;}
    | RETURN puntocoma              {$$ = $1;}
    | BREAK                 {$$ = $1;}
    | CONTINUE puntocoma            {$$ = $1;}
;

DECLARACION_VAR
     : TYPE id igual EXPRESION          {$$ = $2 +  $3 + $4 ; console.log("declarando variable", $1);}
     | TYPE id coma DECLARACION_VAR     { $$ = $2 +  $3 + $4 ;} // falta colocar comillas o caracteres si es el tipo de caracter e identacion
     | TYPE id                          {$$ = $2 + " = " + "null" ;}
;

ASIGNACIONVAR
: id igual EXPRESION {$$ = $1 +  $2 + $3 ;console.log("asignando variable", $1)}
;

DECLARACION_FUNCION
    : TYPE id parentesisaper parentesiscierre llaveaper INSTRUCCIONES llavecierre           {$$ = traducir.funcionYMetodoVacio($2, $6);}
    | TYPE id parentesisaper PARAMETROS parentesiscierre llaveaper INSTRUCCIONES llavecierre {$$ = traducir.funcionYMetodo($2,$4, $7);}
;
```

```
DECLARACION_METODO
    : void id parentesisaper parentesiscierre llaveaper INSTRUCCIONES llavecierre           {$$ = traducir.funcionYMetodoVacio($2, $6);}
    | void id parentesisaper PARAMETROS parentesiscierre llaveaper INSTRUCCIONES llavecierre {$$ = traducir.funcionYMetodo($2,$4, $7);}
;

// if
IF_INS
 : INS_IF {$$ = $1}
 | INS_IF MULTI_ELSE {$$ = $1 + $2}
 ;

 INS_IF
 : if_sen parentesisaper EXPRESION parentesiscierre llaveaper  INSTRUCCIONES  llavecierre {$$ = traducir.sentenciaIf($3, $6); console.log("terminando if", $3);}
 ;


MULTI_ELSE
 : MULTI_ELSE ELSE { $$ = $1 + $2}
 | ELSE            { $$ = $1}
 ;

 ELSE
 : else llaveaper INSTRUCCIONES llavecierre {$$ = traducir.sentenciaElse($3, "else:");}
| else if llaveaper INSTRUCCIONES llavecierre {$$ = traducir.sentenciaElse($3, "elif:");}
 ;

 // switch
SWITCH
//: default dospuntos  {$$ = $1; console.log("terminando case")}
//:   SWITCHCASES {$$ = $1}
: switch parentesisaper EXPRESION parentesiscierre llaveaper SWITCHCASES {$$ = traducir.sentenciaSwitch() + $6}
//: switch parentesisaper EXPRESION parentesiscierre llaveaper  { console.log("terminando switch", $3)}
// $$ = traducir.sentenciaSwitch() + $6;  id igual EXPRESION puntocoma break
;

SWITCHCASES
//: CASES llavecierre{$$ = $1}
: CASES DEFAULT {$$ = $1 + $2}
;

CASES
: CASES CASE {$$ = $1 + $2; console.log("terminando switch", $2)}
| CASE {$$ = $1; console.log("terminando case", $1)}
;

CASE
: case entero dospuntos id igual EXPRESION puntocoma break puntocoma { $$ = traducir.casesParaSwitch($2, $4 , $6)}
;

DEFAULT
: default dospuntos INSTRUCCIONES llavecierre {$$ = $1}
;
```

servicesAnalisis.js | gramatica.jison M | BackupGramatica.jison U

backend > analizador > gramatica.jison

```
FOR
//:for id incremento {$$ = $2 + $3}
//for parentesisaper INICIOFOR puntocoma EXPRESION puntocoma ACTUALIZACION  {$$ = $1 + $2 + $3 + $4 + $5}
:for parentesisaper TYPE id igual EXPRESION puntocoma EXPRESION_FOR puntocoma ACTUALIZACION parentesiscierre llaveaper INSTRUCCIONES llavecierre {$$ = traducir.sentenciaFor($4, $6, $8, $13)}
;


INICIOFOR
: TYPE id igual EXPRESION {$$  = $2}
;

ACTUALIZACIONFOR
: EXPRESION puntocoma ACTUALIZACION { $$ = $1 + $2 + $3}
;


ACTUALIZACION
: id incremento { $$ = $1 + $2 }
| id reduccion { $$ = $1 + $2 }
;
//WHILE

WHILE
: while parentesisaper EXPRESION parentesiscierre llaveaper INSTRUCCIONES llavecierre {$$ = traducir.sentenciaWhile(true, $3, $6, "")}
| ASIGNACIONVAR do llaveaper INSTRUCCIONES llavecierre while parentesisaper EXPRESION parentesiscierre { $$ = traducir.sentenciaWhile(false, $3,$6, $1)}
;


//print
PRINT
: imprimir parentesisaper EXPRESION parentesiscierre {$$ = traducir.traducirPrint($3)}
;


//return

RETURN
: return EXPRESION {$$ = $1 + " " + $2}
| return {$$ = $1}
;

//break

BREAK
: break {$$ = $1}
;

//continue

CONTINUE
: continue {$$ = $1}
;

PARAMETROS
: PARAMETROS coma PARAMETRO { $$ = $1 + $2 + $3}
|PARAMETRO {$$ = $1}
;
```

```
PARAMETRO
: TYPE id { $$ = $2}
;



EXPRESION
    : menos EXPRESION %prec uMenos  { $$ = $2 *-1 }
    | EXPRESION mas EXPRESION       { $$ = $1 + " + " +  $3 }
    | EXPRESION menos EXPRESION     { $$ = $1 + " - " + $3 }
    | EXPRESION por EXPRESION       { $$ = $1 + " * " + $3}
    | EXPRESION dividido EXPRESION  { $$ = + $1 + " / " + $3 }
    | EXPRESION and EXPRESION       { $$ =  $1 + "&&" + $3; }
    | EXPRESION or EXPRESION          { $$ =  $1 + " || " + $3; }
    | EXPRESION menor EXPRESION       { $$ = $1 + " < " + $3; }
    | EXPRESION menorigual EXPRESION  { $$ = $1 + " <= " + $3 }
    | EXPRESION mayor EXPRESION       { $$ =  $1  + " > " + $3 }
    | EXPRESION mayorigual EXPRESION  { $$ = $1 + " >= " + $3 }
    | EXPRESION igualigual EXPRESION  { $$ = $1 + " == " + $3}
    | cadena                        { $$ = '"' + $1 + '"' }
    | entero                        { $$ = $1; }
    | decimal                       { $$ = $1}
    | true                          { $$ = $1 }
    | false                         { $$ = $1; }
    | caracter                      {$$ = "'" + $1 + "'" }
    | id                            {$$ = $1; console.log("identificador", $1);}
;

TYPE
    :int     {console.log("reconociendo string", $1); $$ = Type.ENTERO}
    | double {$$ = Type.DOUBLE}
    | bool   {$$ = Type.BOOLEANO}
    | resstring {$$ = Type.CADENA}
    | char   {$$ = Type.CARACTER}
;


EXPRESION_FOR
    : menos EXPRESION %prec uMenos  { $$ = $2 *-1 }
    | EXPRESION_FOR mas EXPRESION_FOR       { $$ =  $3; }
    | EXPRESION_FOR menos EXPRESION_FOR     { $$ =  $3; }
    | EXPRESION_FOR por EXPRESION_FOR       { $$ =  $3; }
    | EXPRESION_FOR dividido EXPRESION_FOR  { $$ = + $1 + " / " + $3 }
    | EXPRESION_FOR and EXPRESION_FOR       { $$ =  $1 + "&&" + $3; }
    | EXPRESION_FOR or EXPRESION _FOR         { $$ =  $1 + " || " + $3; }
    | EXPRESION_FOR menor EXPRESION_FOR      { $$ =  $3; }
    | EXPRESION_FOR menorigual EXPRESION_FOR  { $$ =  $3; }
    | EXPRESION_FOR mayor EXPRESION_FOR       { $$ =  $3; }
    | EXPRESION_FOR mayorigual EXPRESION_FOR  { $$ =  $3; }
    | EXPRESION_FOR igualigual EXPRESION_FOR  { $$ =  $3; }
    | cadena                        { $$ = '"' + $1 + '"' }
    | entero                        { $$ = $1; }
    | decimal                       { $$ = $1}
    | true                          { $$ = $1 }
    | false                         { $$ = $1; }
    | caracter                      {$$ = "'" + $1 + "'" }
    | id                            {$$ = $1; console.log("identificador", $1);}
```