

# **MANUAL TECNICO**

**PROYECTO 2 TRANSPILADOR - MANUAL TECNICO**

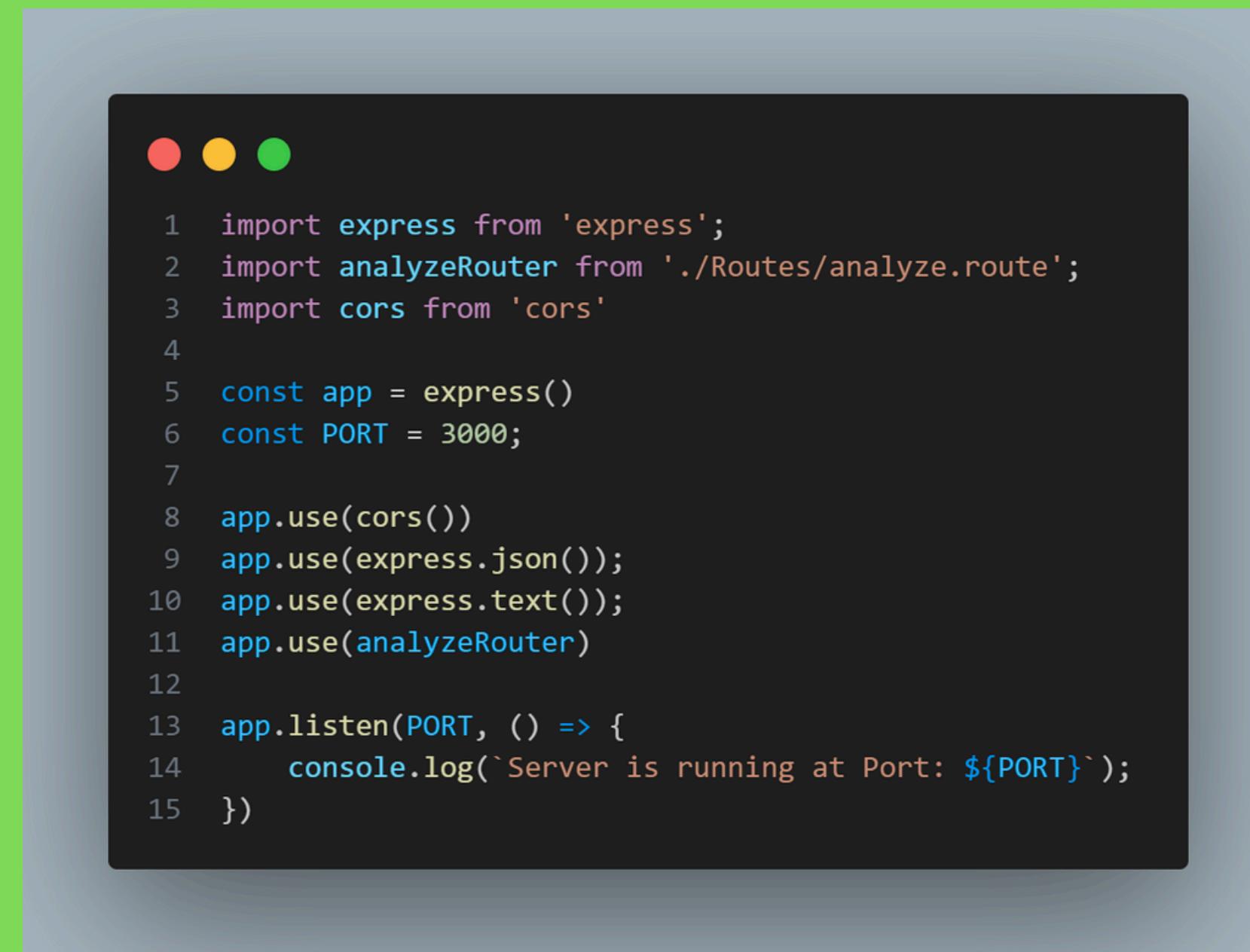


# INTRODUCCION

Este es un manual tecnico que muestra como entender el codigo de programacion de la pagina web de Transpilador, mostrando sus funcionalidades y la logica que tiene en su backend.

# BACKEND

**index:** El archivo principal del sistema de backend esta dado por el index que contiene lo siguiente en codigo; Hago uso de “cors” y tambien de express para el manejo mas facil de mi proyecto, express para los servicios y cors para poder usar esta api de backend en un entorno global, y uso express con json para el manejo de respuestas, tambien instancio el uso de las rutas que estan en otro archivo, por ultimo levanto el servidor en el puerto 3000.



```
● ● ●

1 import express from 'express';
2 import analyzeRouter from './Routes/analyze.route';
3 import cors from 'cors'
4
5 const app = express()
6 const PORT = 3000;
7
8 app.use(cors())
9 app.use(express.json());
10 app.use(express.text());
11 app.use(analyzeRouter)
12
13 app.listen(PORT, () => {
14   console.log(`Server is running at Port: ${PORT}`);
15 })
```

# BACKEND

```
1 enum Type {  
2     UNKNOW,  
3     KEY_O,  
4     KEY_C,  
5     BRA_O,  
6     BRA_C,  
7     PAR_O,  
8     PAR_C,  
9     SEMICOLON,  
10    COMMA,  
11    PERIOD,  
12    ASSIGN,  
13    PLUS,  
14    MINUS,  
15    MULT,  
16    DIV,  
17    INC,  
18    DEC,  
19    EQUAL,  
20    DIFF,  
21    LESS,  
22    GREATER,  
23    LESS_EQ,  
24    GREATER_EQ,  
25    IDENTIFIER,  
26    INTEGER,  
27    DECIMAL,  
28    COMMENT,  
29    MULTICOMMENT,  
30    STRING,  
31    CHAR,  
32    R_USING,  
33    R_SYSTEM,  
34    R_PUBLIC,  
35    R_CLASS,  
36    R_STATIC,  
37    R_VOID,  
38    R_MAIN,  
39    R_STRING,  
40    R_INT,  
41    R_FLOAT,  
42    R_CHAR,  
43    R_BOOL,  
44    R_FALSE,  
45    R_TRUE,  
46    R_CONSOLE,  
47    R_WITELINE,  
48    R_IF,  
49    R_ELSE,  
50    R_FOR  
51 }
```

**Token:** En la parte del archivo de Token hago un constructor con sus variables locales privadas que me ayudaran a crear una estructura sobre que es un token y tambien un enum Type que me va servir para llevar un orden sobre que tipos de Token puedo tener en mi programa, tambien tengo dos Getters que me sirven en otra parte del codigo para poder acceder a los lexemas y los tipos de token.

```
1 class Token {  
2     private row: number;  
3     private column: number;  
4     private lexeme: string;  
5     private typeToken: Type;  
6     private typeTokenString: string;  
7  
8     constructor(typeToken: Type, lexeme: string, row: number, column: number) {  
9         this.typeToken = typeToken;  
10        this.typeTokenString = Type[typeToken];  
11        this.lexeme = lexeme;  
12        this.row = row;  
13        this.column = column;  
14    }  
15    getLexeme(): string {  
16        return this.lexeme;  
17    }  
18    getType(): Type {  
19        return this.typeToken;  
20    }  
21 }  
22  
23 export { Type, Token };
```

# BACKEND

**LexicalAnalyzer:** Esta parte del código es de las más importantes, aca es donde se maneja toda la lógica del AFD y como funciona el analizador léxico, lo que hago aca es primero instanciar variables privadas y un constructor que va consistir en tres arreglos, donde uno de los arreglos contiene las palabras reservadas o identificadores un estado para moverse, un carácter auxiliar y uno de fila y uno de columna para poder llevar un orden y un conocimiento. Por otra parte estan las funciones de addCharacter que agrega los caracteres actuales y los auxiliares para llevar orden, la función Clean que limpia caracteres y estados, la función de addToken que agrega los tokens a la lista de tokens y addError que agrega los errores a la lista de errores y por ultimo un getter que devuelve la lista de errores.

```
1 type ReserverWords = {
2   lexeme: string;
3   token: Type;
4 }
5
6 class LexicalAnalyze {
7   private state: number;
8   private auxChar: string;
9   private row: number;
10  private column: number;
11  private tokenList: Token[];
12  private errorList: Token[];
13  private reservedWords: ReserverWords[];
14
15  constructor() {
16    this.row = 1;
17    this.column = 1;
18    this.auxChar = '';
19    this.state = 0;
20    this.tokenList = [];
21    this.errorList = [];
22    this.reservedWords = [
23      { lexeme: 'using', token: Type.R_USING },
24      { lexeme: 'System', token: Type.R_SYSTEM },
25      { lexeme: 'public', token: Type.R_PUBLIC },
```

```
1  private addCharacter(char: string) {
2    this.auxChar += char;
3    this.column++;
4  }
5
6  private clean() {
7    this.state = 0;
8    this.auxChar = '';
9  }
10
11 private addToken(type: Type, lexeme: string, row: number, column: number) {
12   this.tokenList.push(new Token(type, lexeme, row, column));
13 }
14
15 private addError(type: Type, lexeme: string, row: number, column: number) {
16   this.errorList.push(new Token(type, lexeme, row, column));
17 }
18
19 getErrorList() {
20   return this.errorList;
21 }
```

# BACKEND

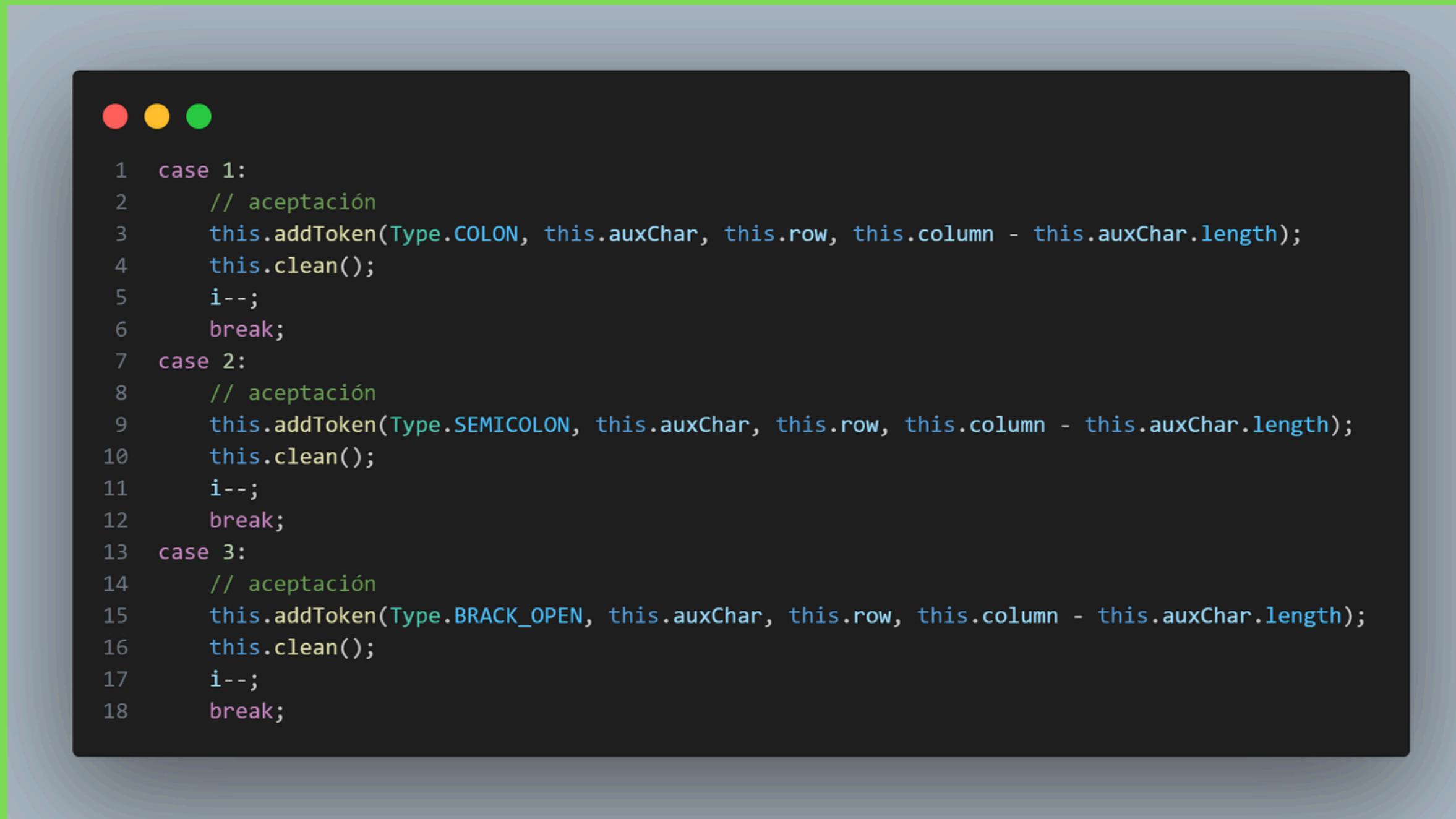
**LexicalAnalyzer:** Esta parte es muy tecnica, dicha funcion primero inicia con una instancia de un caracter y continua con un bucle for que va recorrer un condicional switch con determinados casos donde en el primer caso enumerado con el 0 va tener dentro otro switch el cual va tener todos lo inicios de palabras reservadas o transiciones principales de salen del estado cero en un AFD entonces dependiendo de en que parte del caso 0 inicia reconociendo lo mandara a ese estado, hay otros estados del estado 0 que ayudan por ejemplo para el espaciado y saltos de linea que son necesarios para saber las columnas y filas.

```
1  scanner(input: string) {  
2      input += '#';  
3      let char: string;  
4      for (let i: number = 0; i < input.length; i++) {  
5          char = input[i];  
6          switch (this.state) {  
7              case 0:  
8                  switch (char) {  
9                      case ':':  
10                         this.state = 1;  
11                         this.addCharacter(char);  
12                         break;  
13                     case ';':  
14                         this.state = 2;  
15                         this.addCharacter(char);  
16                         break;  
17                     case '[':  
18                         this.state = 3;  
19                         this.addCharacter(char);  
20                         break;
```

```
1  case '\t':  
2      this.column += 4;  
3      break;  
4  default:  
5      if (/[^a-zA-Z]/.test(char)) {  
6          // es una letra  
7          this.state = 9;  
8          this.addCharacter(char)  
9          continue;  
10     }  
11     if (/^\d/.test(char)) {  
12         // es un dígito  
13         this.state = 10;  
14         this.addCharacter(char);  
15         continue;  
16     }  
17     if (char == '#' && i == input.length - 1) {  
18         // Se termino el analisis  
19         console.log("Analyze Finished");  
20     } else {  
21         // Error Léxico  
22         this.addError(Type.UNKNOW, char, this.row, this.column);  
23         this.column++;  
24     }  
25 }
```

# BACKEND

Al momento de dirigirse a uno de los estados fuera del principal que es el cero dependera abran 3 casos para recibir valores y almacenarlos, por ejemplo en el caso 1 al identificar en el estado 0 que es un parentesis abierto ya lo reconoce como Token entonces ira al estado 1 y agregara el token de ‘(‘ a la lista de Tokens luego hace funcion del Clean y regresa las lienas usadas de i para el siguiente caracter a reconocer

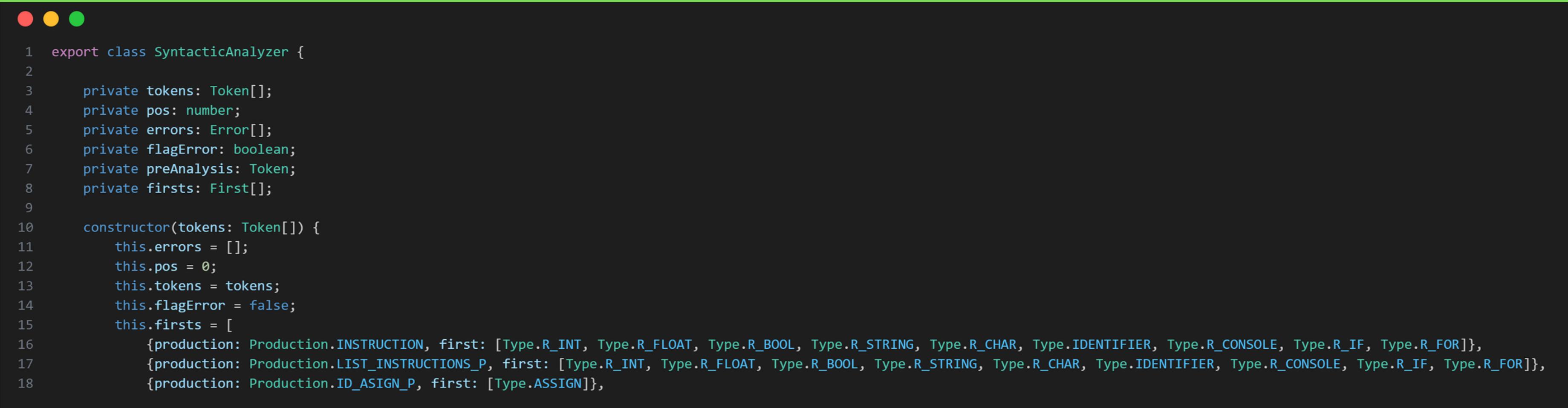


```
● ● ●

1 case 1:
2     // aceptación
3     this.addToken(Type.COLON, this.auxChar, this.row, this.column - this.auxChar.length);
4     this.clean();
5     i--;
6     break;
7 case 2:
8     // aceptación
9     this.addToken(Type.SEMICOLON, this.auxChar, this.row, this.column - this.auxChar.length);
10    this.clean();
11    i--;
12    break;
13 case 3:
14     // aceptación
15     this.addToken(Type.BRACK_OPEN, this.auxChar, this.row, this.column - this.auxChar.length);
16     this.clean();
17     i--;
18     break;
```

# BACKEND

**SyntacticAnalyzer:** En este archivo se crea todo lo que tiene que ver con analizar la estructura del código que le estamos mandando, me refiero a que aquí es que se valida por ejemplo si una asignación tiene la estructura esperada por una asignación tipo que la variable y su tipo estén bien estructuradas, esto únicamente en este caso para el lenguaje de C#, en el que se encarga de conocer bien la estructura del lenguaje y analizar que esté bien redactado sintácticamente.



The screenshot shows a code editor window with a dark theme. At the top left, there are three circular icons: a red one, a yellow one, and a green one. The main area contains the following code:

```
1 export class SyntacticAnalyzer {  
2  
3     private tokens: Token[];  
4     private pos: number;  
5     private errors: Error[];  
6     private flagError: boolean;  
7     private preAnalysis: Token;  
8     private firsts: First[];  
9  
10    constructor(tokens: Token[]) {  
11        this.errors = [];  
12        this.pos = 0;  
13        this.tokens = tokens;  
14        this.flagError = false;  
15        this.firsts = [  
16            {production: Production.INSTRUCTION, first: [Type.R_INT, Type.R_FLOAT, Type.R_BOOL, Type.R_STRING, Type.R_CHAR, Type.IDENTIFIER, Type.R_CONSOLE, Type.R_IF, Type.R_FOR]},  
17            {production: Production.LIST_INSTRUCTIONS_P, first: [Type.R_INT, Type.R_FLOAT, Type.R_BOOL, Type.R_STRING, Type.R_CHAR, Type.IDENTIFIER, Type.R_CONSOLE, Type.R_IF, Type.R_FOR]},  
18            {production: Production.ID_ASIGN_P, first: [Type.ASSIGN]}],
```

# BACKEND

```
1  public parser() {
2      this.blockUsing();
3      this.class();
4  }
5
6  private blockUsing() {
7      this.expect(Type.R_USING);
8      this.expect(Type.R_SYSTEM);
9      this.expect(Type.SEMICOLON);
10 }
11
12 private class() {
13     this.expect(Type.R_PUBLIC);
14     this.expect(Type.R_CLASS);
15     this.expect(Type.IDENTIFIER);
16     this.expect(Type.KEY_O);
17     this.blockMain();
18     this.expect(Type.KEY_C);
19 }
20
21 private blockMain() {
22     this.expect(Type.R_STATIC);
23     this.expect(Type.R_VOID);
24     this.expect(Type.R_MAIN);
25     this.expect(Type.PAR_O);
26     this.expect(Type.R_STRING);
27     this.expect(Type.BRA_O);
28     this.expect(Type.BRA_C);
29     this.expect(Type.IDENTIFIER);
30     this.expect(Type.PAR_C);
31     this.expect(Type.KEY_O);
32     this.listInstructions();
33     this.expect(Type.KEY_C);
34 }
```

```
1  private type() {
2      this.expect(this.preAnalysis.getType());
3  }
4
5  private listId() {
6      this.idAsign();
7      this.listIdP();
8  }
9
10 private idAsign() {
11     this.expect(Type.IDENTIFIER);
12     this.idAsignP();
13 }
14
15 private idAsignP() {
16     if (this.isFirst(Production.ID_ASSIGN_P)) {
17         this.expect(Type.ASSIGN);
18         this.expression();
19     }
20 }
21
22 private listIdP() {
23     if (this.isFirst(Production.LIST_ID_P)) {
24         this.expect(Type.COMMA);
25         this.idAsign();
26         this.listIdP();
27     }
28 }
```

```
1  private print() {
2      this.expect(Type.R_CONSOLE);
3      this.expect(Type.PERIOD);
4      this.expect(Type.R_WRITELINE);
5      this.expect(Type.PAR_O);
6      this.expression();
7      this.expect(Type.PAR_C);
8      this.expect(Type.SEMICOLON);
9  }
10
11 private instrIf() {
12     this.expect(Type.R_IF);
13     this.expect(Type.PAR_O);
14     this.expression();
15     this.expect(Type.PAR_C);
16     this.expect(Type.KEY_O);
17     this.listInstructions();
18     this.expect(Type.KEY_C);
19     this.instIfP();
20 }
21
22 private instIfP() {
23     if (this.isFirst(Production.INST_IF_P)) {
24         this.expect(Type.R_ELSE);
25         this.expect(Type.KEY_O);
26         this.listInstructions();
27         this.expect(Type.KEY_C);
28     }
29 }
30 }
```

# BACKEND

**Error:** Aca se manejan una interface para los errores que va detectar el analizador sintactico y asi llevar un mejor control de los errores del mismo.



```
1  export class Error {
2      private row;
3      private column;
4      private lexeme: string;
5      private description: string;
6
7      constructor(lexeme:string, description: string, row: number, column: number) {
8          this.lexeme = lexeme;
9          this.description = description;
10         this.row = row;
11         this.column = column;
12     }
13
14     getLexeme(): string {
15         return this.lexeme;
16     }
17
18     getDescription(): string {
19         return this.description;
20     }
21
22     getRow(): number {
23         return this.row;
24     }
25
26     getColumn(): number {
27         return this.column;
28     }
29 }
```

# BACKEND

**Instruction:** Esta interaz ayuda a llevar el orden de la mayoria de estructuras en donde a cada estructura se le asigna una fila y columna en la cual debe acoplarse mas su transpilador que es el codigo asignado.

```
● ● ●  
1 export interface Instruction {  
2     row: number;  
3     column: number;  
4     // Método Abstracto  
5     transpiler(): string;  
6 }  
7  
8 }
```

**Produtction:** Este enum contienen identificadores de tipo para los distintos tipos de bloques de codigos que se encuentran en el analizador sintactico como por ejemplo INST\_FOR que es un bloque de codigo de una sentencia “for”.

```
● ● ●  
1 export enum Production {  
2     PROGRAM,  
3     BLOCK_USING,  
4     CLASS,  
5     BLOCK_MAIN,  
6     LIST_INSTRUCTIONS,  
7     LIST_INSTRUCTIONS_P,  
8     INSTRUCTION,  
9     DECLARATION,  
10    TYPE,  
11    LIST_ID,  
12    ID_ASIGN,  
13    ID_ASIGN_P,  
14    LIST_ID_P,  
15    ASSIGNATION,  
16    PRINT,  
17    INST_IF,  
18    INST_IF_P,  
19    INST_FOR,  
20    FIRST_BLOCK_FOR,  
21    THIRD_BLOCK_FOR,  
22    THIRD_BLOCK_FOR_P,  
23    INCREMENT,  
24    DECREMENT,  
25    EXPRESSION,  
26    ARITHMETIC,  
27    RELATIONAL,  
28    ARITHMETIC_P,  
29    TERM,  
30    TERM_P,  
31    FACTOR  
32 }
```

# BACKEND

**First:** Este es un tipado que ayuda para asignar los identificadores de bloques de instrucciones con los posibles tokens que pueden llegar a tener.

```
● ● ●  
1 import { Type } from "../Analyzer/Token";  
2 import { Production } from "./Production";  
3  
4 export type First = {  
5     production: Production;  
6     first: Type[];  
7 }
```

**IdDec:** Este tipado es una estructura la cual ayuda para asignar los valores a los identificadores y así llevar un mejor control de ambos.

```
● ● ●  
1 import { Instruction } from "../models/abstract/Instruction"  
2  
3 export type IdDec = {  
4     id: Instruction;  
5     value: Instruction | undefined;  
6 }
```

# BACKEND

```
● ● ●  
1 import { Instruction } from "../abstract/Instruction";  
2  
3 export class Arithmetic implements Instruction {  
4  
5     row: number;  
6     column: number;  
7     private exp1: Instruction | undefined;  
8     private exp2: Instruction | undefined;  
9     private operator: string;  
10    private flag: boolean;  
11  
12    constructor(exp1: Instruction | undefined, exp2: Instruction | undefined, operator: string, row: number, column: number){  
13        this.row = row;  
14        this.column = column;  
15        this.exp1 = exp1;  
16        this.exp2 = exp2;  
17        this.operator = operator;  
18        this.flag = false;  
19    }  
20  
21    transpiler(): string {  
22        let left: string = '';  
23        let right: string = '';  
24  
25        if (this.exp1) {  
26            left = this.exp1.transpiler();  
27        }  
28  
29        if (this.exp2) {  
30            right = this.exp2.transpiler();  
31        }  
32  
33        return this.flag ? `(${left} ${this.operator} ${right})` : `${left} ${this.operator} ${right}`;  
34    }  
35  
36    public setFlag(flag: boolean) {  
37        this.flag = flag;  
38    }  
39}  
40 }
```

**Expresions:** Asi como este archivo de codigo que esta orientado a “Arithmetic” tambien hay otros 3: “Identifier”, “Primitive”, “Relational”, en donde cada una es similar a este bloque de codigo, donde la idea principal es que se estructure bien el como van a estar los bloques de codigo asignado con variables entre otras cosas por ejemplo este codigo hace realciones de expresiones aritmeticas, evalua dos expresiones por medio de suma resta, etc. y entonces logra armar la estructura pero para otro lenguaje.

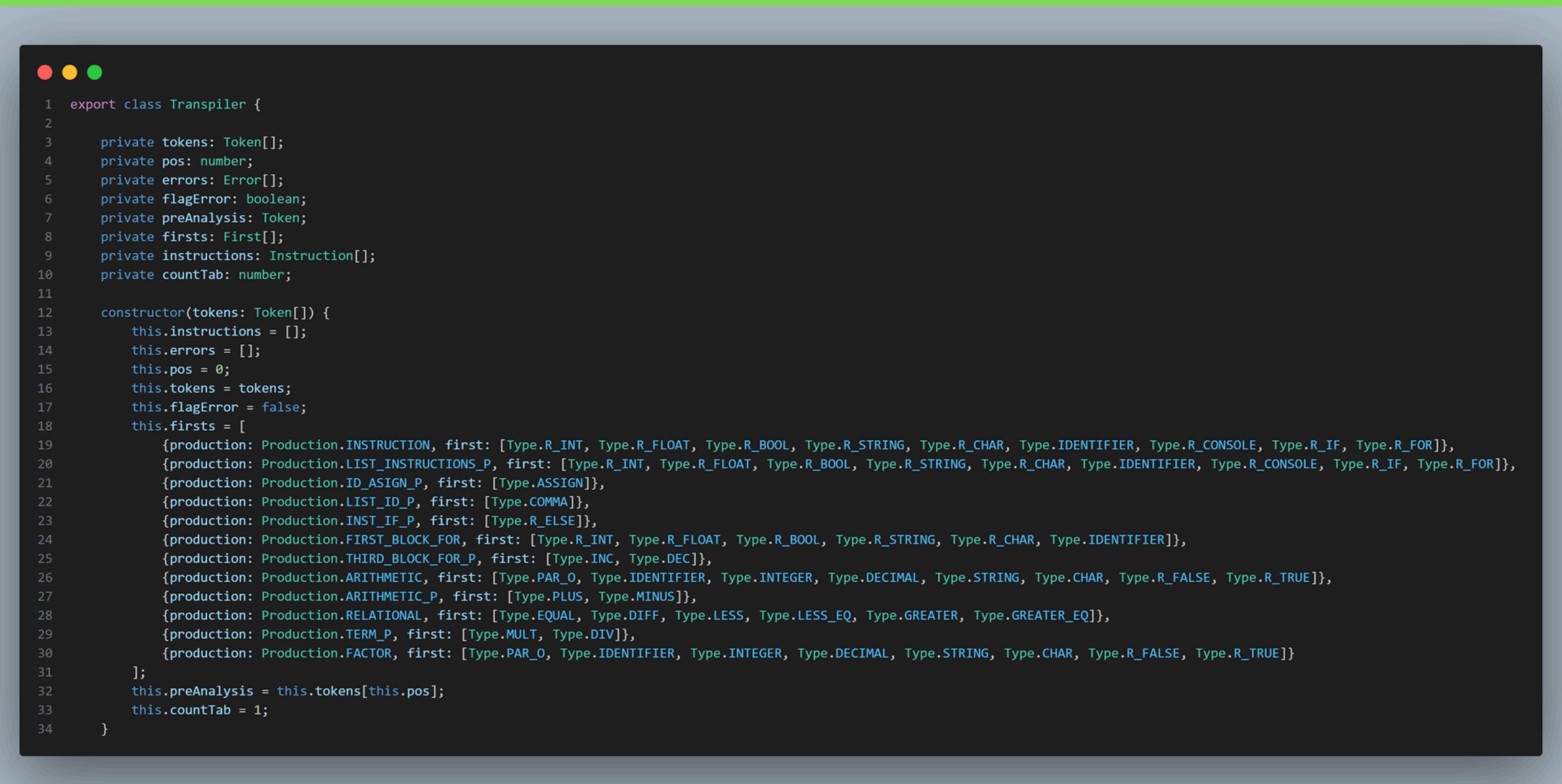
# BACKEND

```
1 import { Instruction } from "../abstract/Instruction"
2
3 export class Assigntation implements Instruction {
4
5     row: number;
6     column: number;
7     private id: string;
8     private exp: Instruction | undefined;
9
10    constructor(id: string, exp: Instruction | undefined, row: number, column: number) {
11        this.row = row;
12        this.column = column;
13        this.id = id;
14        this.exp = exp;
15    }
16
17    transpiler(): string {
18
19        let exp: string = '';
20
21        if (this.exp) {
22            exp = this.exp.transpiler();
23        }
24
25        return `${this.id} = ${exp};\n`;
26    }
27}
28}
```

**Instructions:** Este es parecido al anterior codigo donde se estructura una estructura para asignacion de variables, en donde esta asociada a su id y a su expresion, asi como este hay otros 4: “Declaration”, “For”, “If”, “Print”. Con estos metodos se busca crear la estructura de por como su nombre lo infica dichas sentencias de codigo pero en otro lenguaje de programacion.

# BACKEND

**Transpiler:** Por ultimo para la logica esta el transpilador, que es el encargado de pasar el lenguaje de C# a TypeScript, este esta encargado de transcribir todo ese codigo al lenguaje del cual se esta haciendo referencia, si se presta atencion toda su estructura es muy parecida a la del SyntacticAnalyzer con la diferencia que este lo que hace es tomar los bloques de codigo y transformarlos a bloques de codigo de estructura de lenguaje de TypeScript.



The screenshot shows a code editor window with a dark theme. At the top left, there are three circular icons: red, yellow, and green. The code editor displays a class named 'Transpiler' with the following content:

```
1 export class Transpiler {
2
3     private tokens: Token[];
4     private pos: number;
5     private errors: Error[];
6     private flagError: boolean;
7     private preAnalysis: Token;
8     private firsts: First[];
9     private instructions: Instruction[];
10    private countTab: number;
11
12    constructor(tokens: Token[]) {
13        this.instructions = [];
14        this.errors = [];
15        this.pos = 0;
16        this.tokens = tokens;
17        this.flagError = false;
18        this.firsts = [
19            {production: Production.INSTRUCTION, first: [Type.R_INT, Type.R_FLOAT, Type.R_BOOL, Type.R_STRING, Type.R_CHAR, Type.IDENTIFIER, Type.R_CONSOLE, Type.R_IF, Type.R_FOR]},
20            {production: Production.LIST_INSTRUCTIONS_P, first: [Type.R_INT, Type.R_FLOAT, Type.R_BOOL, Type.R_STRING, Type.R_CHAR, Type.IDENTIFIER, Type.R_CONSOLE, Type.R_IF, Type.R_FOR]},
21            {production: Production.ID_ASIGN_P, first: [Type.ASSIGN]},
22            {production: Production.LIST_ID_P, first: [Type.COMMA]},
23            {production: Production.INST_IF_P, first: [Type.R_ELSE]},
24            {production: Production.FIRST_BLOCK_FOR, first: [Type.R_INT, Type.R_FLOAT, Type.R_BOOL, Type.R_STRING, Type.R_CHAR, Type.IDENTIFIER]},
25            {production: Production.THIRD_BLOCK_FOR_P, first: [Type.INC, Type.DEC]},
26            {production: Production.ARITHMETIC, first: [Type.PAR_0, Type.IDENTIFIER, Type.INTEGER, Type.DECIMAL, Type.STRING, Type.CHAR, Type.R_FALSE, Type.R_TRUE]},
27            {production: Production.ARITHMETIC_P, first: [Type.PLUS, Type_MINUS]},
28            {production: Production.RELATIONAL, first: [Type.EQUAL, Type.DIFF, Type.LESS, Type.LESS_EQ, Type.GREATER, Type.GREATER_EQ]},
29            {production: Production.TERM_P, first: [Type.MULT, Type.DIV]},
30            {production: Production.FACTOR, first: [Type.PAR_0, Type.IDENTIFIER, Type.INTEGER, Type.DECIMAL, Type.STRING, Type.CHAR, Type.R_FALSE, Type.R_TRUE]}
31        ];
32        this.preAnalysis = this.tokens[this.pos];
33        this.countTab = 1;
34    }
}
```

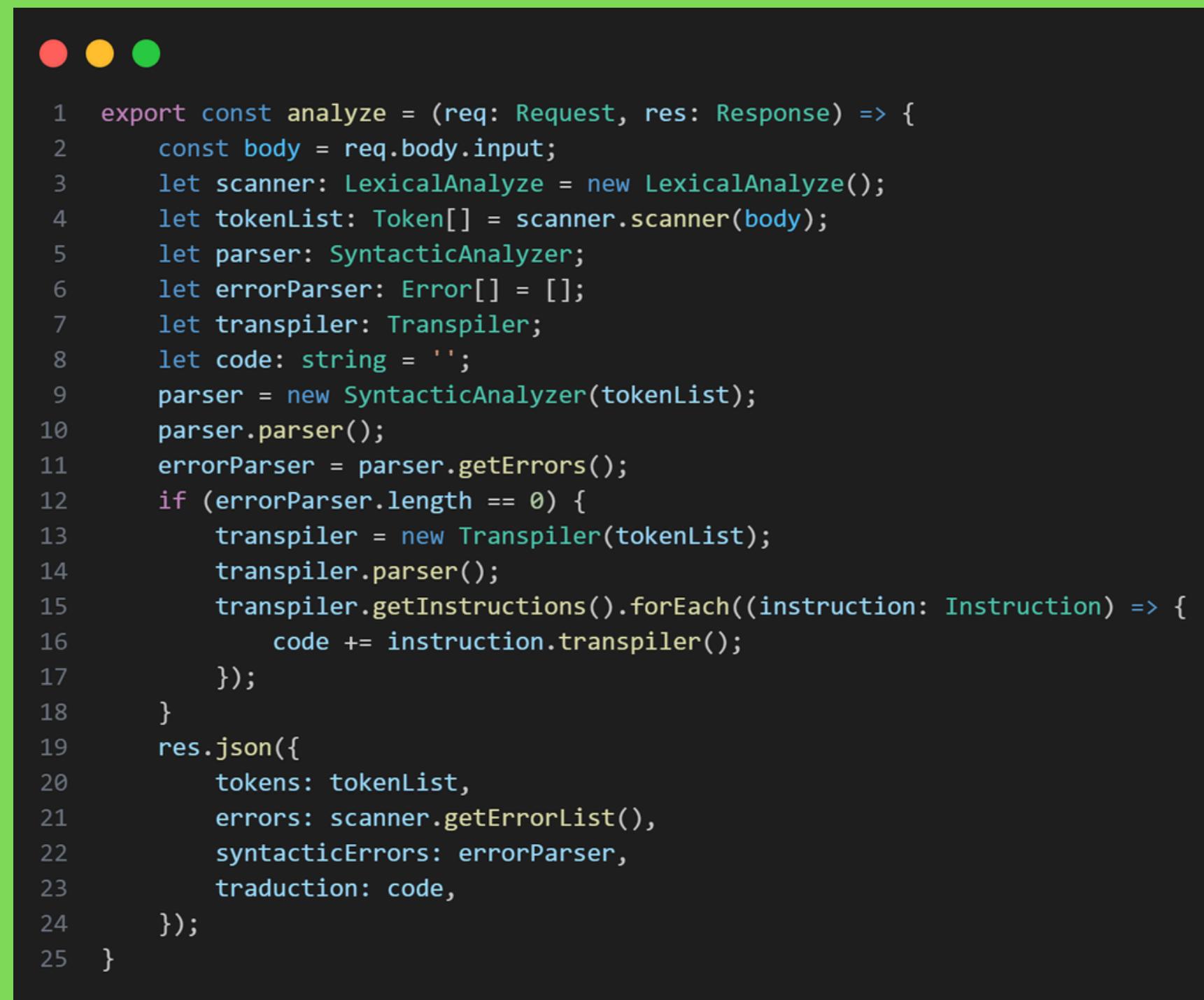
# BACKEND

```
1  private class() {
2      this.expect(Type.R_PUBLIC);
3      this.expect(Type.R_CLASS);
4      this.expect(Type.IDENTIFIER);
5      this.expect(Type.KEY_O);
6      this.blockMain();
7      this.expect(Type.KEY_C);
8  }
9
10 private blockMain() {
11     this.expect(Type.R_STATIC);
12     this.expect(Type.R_VOID);
13     this.expect(Type.R_MAIN);
14     this.expect(Type.PAR_O);
15     this.expect(Type.R_STRING);
16     this.expect(Type.BRA_O);
17     this.expect(Type.BRA_C);
18     this.expect(Type.IDENTIFIER);
19     this.expect(Type.PAR_C);
20     this.expect(Type.KEY_O);
21     this.instructions = this.listInstructions();
22     this.expect(Type.KEY_C);
23 }
24
```

```
1  private instruction(): Instruction | undefined {
2      switch(this.preAnalysis.getType()) {
3          case Type.R_INT:
4          case Type.R_STRING:
5          case Type.R_FLOAT:
6          case Type.R_BOOL:
7          case Type.R_CHAR:
8              return this.declaration();
9          case Type.IDENTIFIER:
10             return this.assigntion();
11         case Type.R_CONSOLE:
12             return this.print();
13         case Type.R_IF:
14             return this.instrIf();
15         case Type.R_FOR:
16             return this.instrFor();
17         default:
18
19             if (this.flagError) return;
20
21             const firsts: First | undefined = this.firsts.find(first => first.production === Production.INSTRUCTION);
22             this.addError(this.preAnalysis, firsts ? firsts.first : []);
23             break;
24     }
25 }
26
27 private declaration(): Instruction {
28     let type: Type = this.preAnalysis.getType();
29     let row: number = this.preAnalysis.getRow();
30     let column: number = this.preAnalysis.getColumn();
31
32     this.type();
33     let listIds: IdDec[] = this.listId();
34     this.expect(Type.SEMICOLON);
35
36     return new Declaration(type, listIds, row, column);
37 }
```

# BACKEND

**Constructor:** Por ultimo se encuentra el constructor que recopila toda la informacion obtenida tanto del analizador lexico, como del analizador sintactico y manda la informacion para que pueda trabajar el transpilador y su respuesta la devuelve al constructor para que este la guarde y la muestre cuando sea llamado desde las rutas.



```
● ● ●

1  export const analyze = (req: Request, res: Response) => {
2      const body = req.body.input;
3      let scanner: LexicalAnalyze = new LexicalAnalyze();
4      let tokenList: Token[] = scanner.scanner(body);
5      let parser: SyntacticAnalyzer;
6      let errorParser: Error[] = [];
7      let transpiler: Transpiler;
8      let code: string = '';
9      parser = new SyntacticAnalyzer(tokenList);
10     parser.parser();
11     errorParser = parser.getErrors();
12     if (errorParser.length == 0) {
13         transpiler = new Transpiler(tokenList);
14         transpiler.parser();
15         transpiler.getInstructions().forEach((instruction: Instruction) => {
16             code += instruction.transpiler();
17         });
18     }
19     res.json({
20         tokens: tokenList,
21         errors: scanner.getErrorList(),
22         syntacticErrors: errorParser,
23         traduction: code,
24     });
25 }
```

# BACKEND

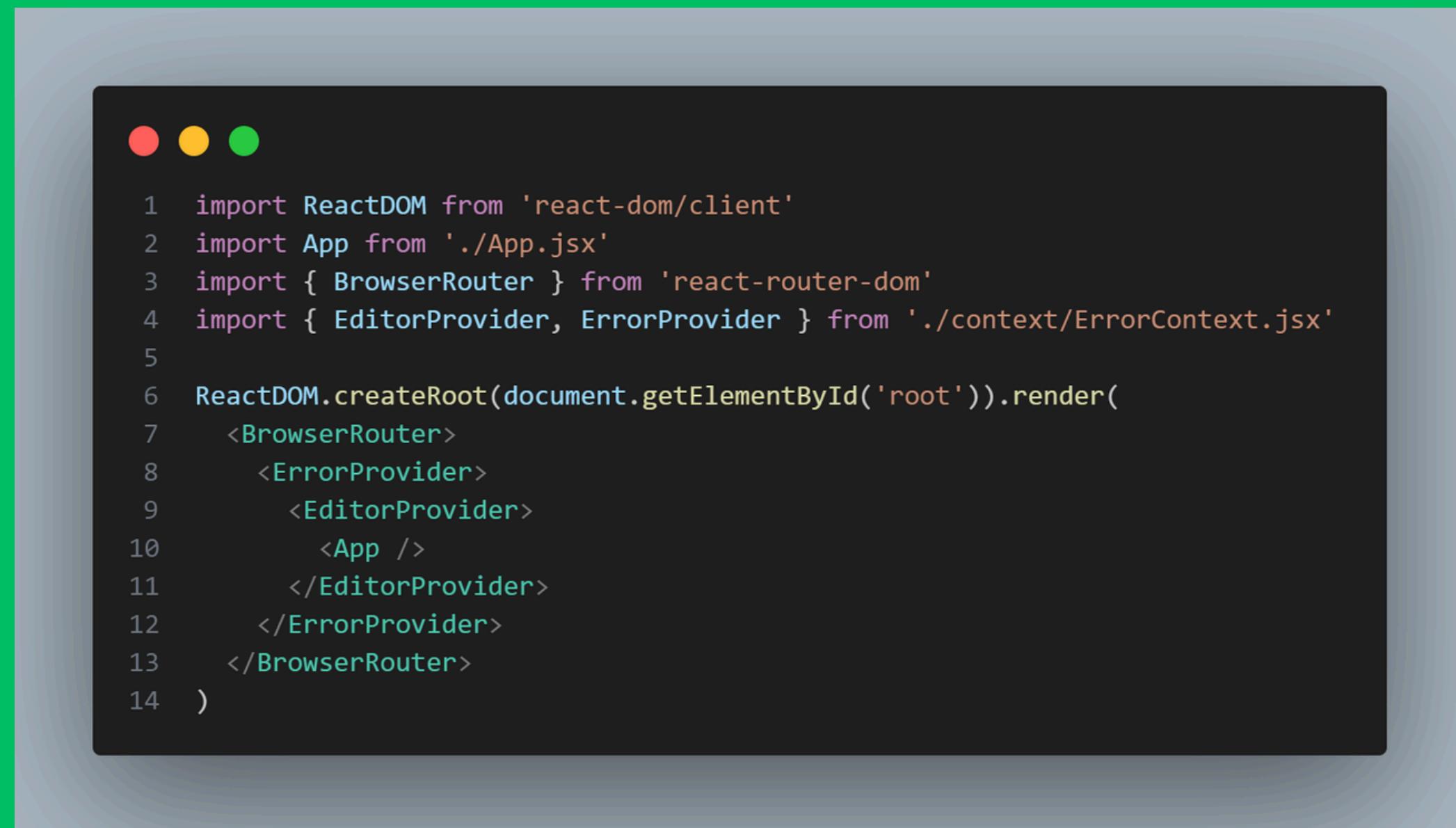
**Route:** Por ultimo en las rutas llamo a la funcion del controlador y le establezco una ruta en la cual se pueda usar en este caso '/analyze' y llamo a la funcion para que al entrar en esa ruta se hagan esas funciones.



```
1 import { analyze, } from '../Controllers/analyze.controller';
2 import { Router } from 'express';
3 const analyzeRouter = Router();
4
5 analyzeRouter.post('/analyze', analyze);
6
7 export default analyzeRouter;
```

# FRONTEND

**main:** En esta parte solo se estructura como va estar utilizandoce la verificacion de rutas en la web, para que se pueda usar React de la mejor manera.



The image shows a terminal window with a dark background and three colored dots (red, yellow, green) at the top left. The terminal contains the following code:

```
1 import ReactDOM from 'react-dom/client'
2 import App from './App.jsx'
3 import { BrowserRouter } from 'react-router-dom'
4 import { EditorProvider, ErrorProvider } from './context/ErrorContext.jsx'
5
6 ReactDOM.createRoot(document.getElementById('root')).render(
7   <BrowserRouter>
8     <ErrorProvider>
9       <EditorProvider>
10      <App />
11    </EditorProvider>
12  </ErrorProvider>
13 </BrowserRouter>
14 )
```

# FRONTEND

App: Aca se crean las distintas rutas que se utilizan en el proyecto web.

```
● ● ●  
1 import { Route, Router, Routes } from 'react-router-dom'  
2 import { Inicial } from './Components/Inicial'  
3 import { Manuales } from './Components/Manuales'  
4 import { Reporte } from './Components/Reporte'  
5  
6 function App() {  
7  
8   return (  
9     <Routes>  
10      <Route path='/' element = {<Inicial />}></Route>  
11      <Route path='/Manuales' element = {<Manuales />}></Route>  
12      <Route path='/Reporte' element = {<Reporte />}></Route>  
13    </Routes>  
14  )  
15}  
16  
17 export default App
```

# FRONTEND

**NavBar:** NavBar: Aca se crea el Navbar que tiene la pagina web con sus rutas, donde la que mas tiene funcion es la de archivos que sirve para cargar un archivo al editor de texto.



```
1  export const PenumNavbar = ({ onFileClick }) => {
2      const [isResponsive, setIsResponsive] = useState(false);
3
4      const toggleNav = () => {
5          setIsResponsive(!isResponsive);
6      };
7      return (
8          <div className={`topnav${isResponsive ? " responsive" : ""}`}>
9              <ul className="nav nav-pills">
10                 <a className="a-style active">Penum USAC</a>
11                 <a className='a-style' href='/>Home</a>
12                 <Link to="/Reporte" className="a-style">Reporte Error</Link>
13                 <a className='a-style' href="/" onClick={(e) => {
14                     e.preventDefault();
15                     onFileClick();
16                 }}>Archivos</a>
17                 <a className='a-style' href="/Manuales">Manuales</a>
18                 <a className="icon" onClick={toggleNav}>
19                     <i className="fa fa-bars icono" />
20                 </a>
21             </ul>
22         </div>
23     )
24 }
```

# FRONTEND

**Api:** Se importa la libreria de axios para manejar las peticiones al Backend, luego se crea la funcion que va llamar al controlador del backend donde se manda la ruta y los datos necesarios que seria el texto donde van los tokens y retorna los datos pero ya verificados.



```
1 export const pensumData = async(data) => {
2     try {
3         const response = await axios.post('http://localhost:3000/analyze', { input: data });
4         return response.data;
5     } catch (error) {
6         console.error("Error al analizar:", error);
7         return null;
8     }
9 };
```

# FRONTEND

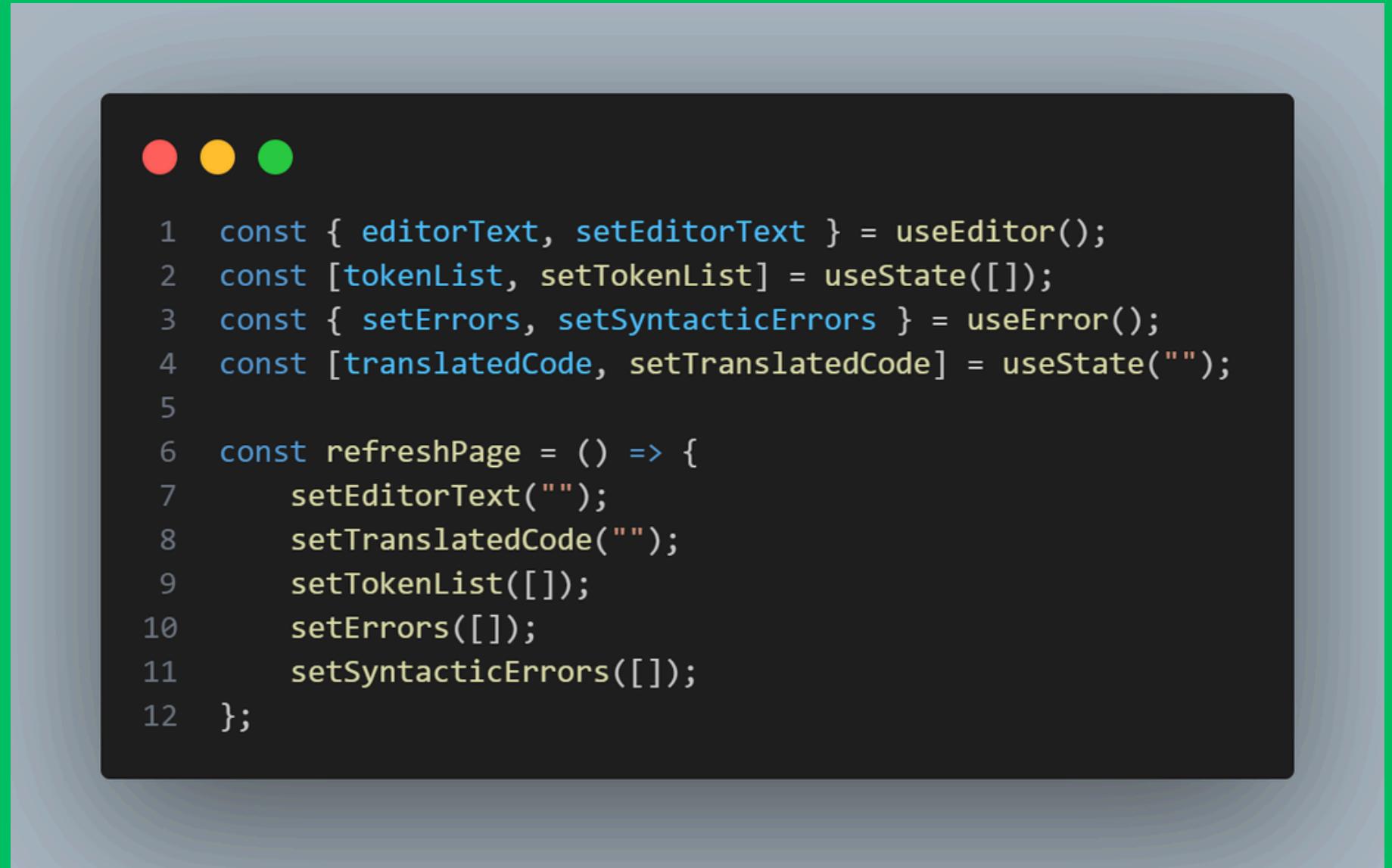


```
1 export const Reporte = () => {
2     const { errors } = useState();
3     return (
4         <>
5             <PensumNavbar />
6             <div className="container mt-4">
7                 <h2>Reporte de Errores</h2>
8                 {errors.length === 0 ? (<p>No se encontraron errores.</p>) : (
9                     <table className="table">
10                         <thead>
11                             <tr>
12                                 <th>No.</th>
13                                 <th>Fila</th>
14                                 <th>Columna</th>
15                                 <th>Lexema</th>
16                                 <th>Token</th>
17                             </tr>
18                         </thead>
19                         <tbody>
20                             {errors.map((error, index) => (
21                                 <tr key={index}>
22                                     <td>{index + 1}</td>
23                                     <td>{error.row}</td>
24                                     <td>{error.column}</td>
25                                     <td>{error.lexeme}</td>
26                                     <td>{error.typeToken}</td>
27                                 </tr>
28                             )));
29                         </tbody>
30                     </table>
31                 )}
32             </div>
33         </>
34     );
35 };
```

**Reporte:** En esta parte del reporte primero se llama al NavBar para que aparezca en la pagina luego se empieza a hacer un mapeo de los errores si es que encuentra, si no entonces mostrara un texto que dice que no hay errores, despues se hace una tabla en la cual empieza a mostrar todos los errores que puedan haber.

# FRONTEND

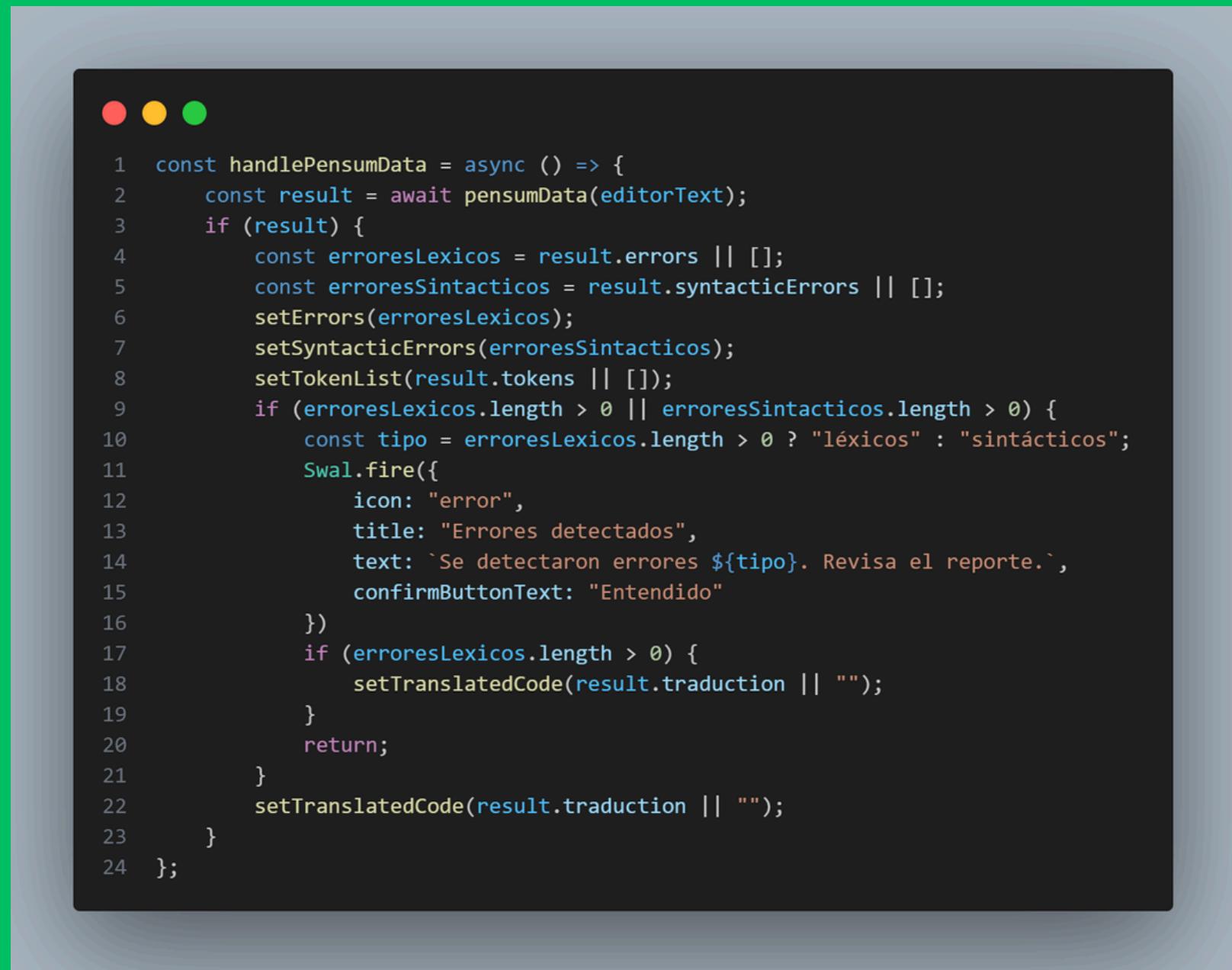
**Inicial:** Aca primero usamos estados de variables para las distintas cosas que tenemos en el codigo, luego tenemos la funcion de recargar la pagina para limpiar el editor, luego tambien esta la funcion de handleFileLoad que sirve para mostrar el input de archivos y permite cargar el archivo directo al textarea.



```
1 const { editorText, setEditorText } = useEditor();
2 const [tokenList, setTokenList] = useState([]);
3 const { setErrors, setSyntacticErrors } = useError();
4 const [translatedCode, setTranslatedCode] = useState("");
5
6 const refreshPage = () => {
7     setEditorText("");
8     setTranslatedCode("");
9     setTokenList([]);
10    setErrors([]);
11    setSyntacticErrors([]);
12};
```

# FRONTEND

Ahora la siguiente funcion manda lo que se encuentra en el textarea para el api para que se conecte con el backend y luego continuar con las funciones como si encuentra un error tanto lexico como sintactico, mandar la alerta y mostrar la lista de tokens y mostrar la traducion si en dado caso el error es lexico, si el error es sintactico no lo traducira pero si mostrara los errores, si no encuentra errores entonces mostrar los tokens y la traducion del lenguaje en typescript.



```
1 const handlePensumData = async () => {
2     const result = await pensumData(editorText);
3     if (result) {
4         const erroresLexicos = result.errors || [];
5         const erroresSintacticos = result.syntacticErrors || [];
6         setErrors(erroresLexicos);
7         setSyntacticErrors(erroresSintacticos);
8         setTokenList(result.tokens || []);
9         if (erroresLexicos.length > 0 || erroresSintacticos.length > 0) {
10             const tipo = erroresLexicos.length > 0 ? "léxicos" : "sintácticos";
11             Swal.fire({
12                 icon: "error",
13                 title: "Errores detectados",
14                 text: `Se detectaron errores ${tipo}. Revisa el reporte.`,
15                 confirmButtonText: "Entendido"
16             })
17             if (erroresLexicos.length > 0) {
18                 setTranslatedCode(result.traduction || "");
19             }
20             return;
21         }
22         setTranslatedCode(result.traduction || "");
23     }
24 };
```

# FRONTEND

```
 1 <div className="row">
 2   /* Editor de texto en C# */
 3   <div className="col-lg-6 col-md-6 col-sm-12 col-token">
 4     <h2>Editor de texto en C#</h2>
 5     <textarea
 6       className="form-control text-editor"
 7       value={editorText}
 8       onChange={(e) => setEditorText(e.target.value)}
 9       rows="10"
10       cols="50"
11     />
12     <button className="btn btn-analizar" onClick={handlePensemData}>
13       Analizar
14     </button>
15     <button className="btn btn-analizar" onClick={refreshPage}>
16       Limpiar Editor
17     </button>
18   </div>
19
20  /* Salida traducida */
21  <div className="col-lg-6 col-md-6 col-sm-12 col-token">
22    <h2>Salida Traducida a TypeScript</h2>
23    <textarea
24      className="form-control text-editor"
25      value={translatedCode}
26      readOnly
27      rows="10"
28      cols="50"
29    />
30  </div>
31 </div>
```

La siguiente parte corresponde al return del archivo inicial en donde se pone la parte del codigo que se muestra en pantalla, lo que hace es lo siguiente, de que el textarea que esta asignada para ser editada por el lengauje de C# entonces va recibir el codigo y al darle analizar este va empezar a tranpilar el codigo a typescript y a mostrar los tokens tambien, aparte tambien hay otro textarea en el cual se mostrara el codigo ya traducido a typescript osea ya transpilado.

# CONCLUSIÓN

- Esas son todas las funciones que tiene la pagina web diseñada para el usuario y su uso de ayuda para elegir sus cursos para aprobar.
- Muestra cada una de las funciones tanto en Backend como en Frontend y como usarlas y por si se quiere modificar el codigo en algun momento.

Nombre: Alberto Moisés Gerardo Lémus Alvarado

Carne: 202400999

