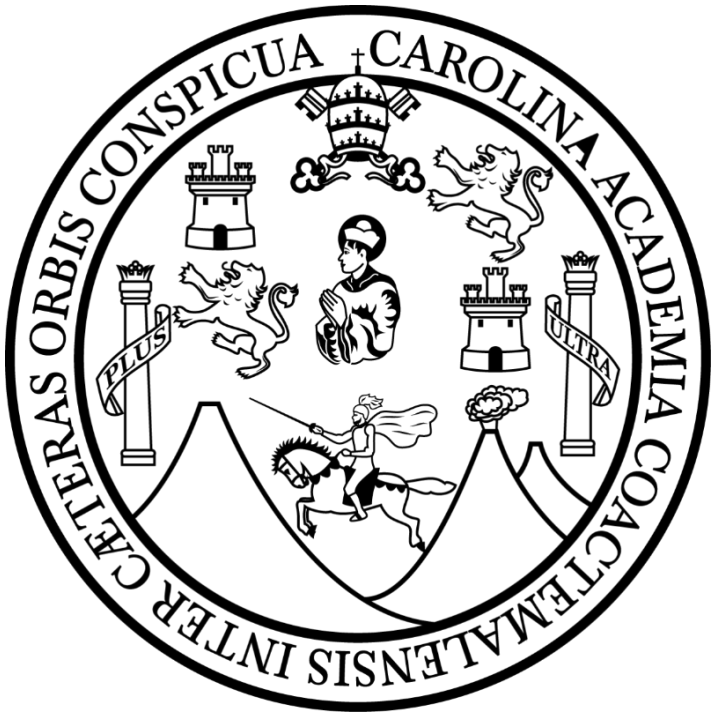
Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Curso: Lenguajes Formales
Sección: B-
Ing. Zulma Aguirre
Tutor Académico:



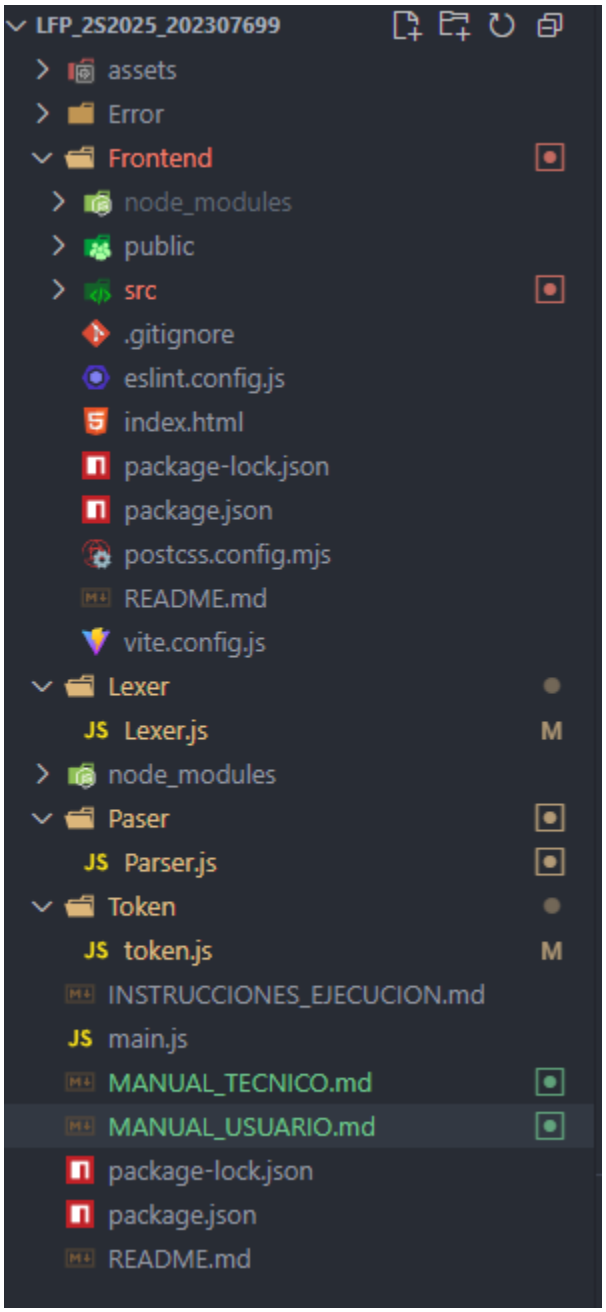
Proyecto#2

Nombre: Juan Carlos Humberto Reyes Chavarria

Carné: 202307699

Arquitectura del sistema:

JavaBridge sigue una arquitectura Cliente-Servidor con separación de responsabilidades:



Componentes del Sistema

1. Token (Token.js)

Propósito: Define la estructura de tokens y palabras reservadas.

Clase Token:

```
export class Token{
    constructor(type, value, line, column){
        this.type = type;
        this.value = value;
        this.line = line;
        this.column = column;
    }
}
```

Palabras Reservadas:

```
export const ReservedWords = {
    "public": "PUBLIC",
    "class": "CLASS",
    "static": "STATIC",
    "void": "VOID",
    "main": "MAIN",
    "String": "STRING_TYPE",
    "args": "ARGS",
    "int": "INT_TYPE",
    "double": "DOUBLE_TYPE",
    "float": "FLOAT_TYPE",
    "char": "CHAR_TYPE",
    "boolean": "BOOLEAN_TYPE",
    "true": "TRUE",
    "false": "FALSE",
    "if": "IF",
    "else": "ELSE",
    "for": "FOR",
    "while": "WHILE",
    "System": "SYSTEM",
    "out": "OUT",
    "println": "PRINTLN"
};
```

Simbolos:

```
export const Symbols = {
    "{": "LLAVE_IZQ",
    "}": "LLAVE_DER",
    "(": "PAR_IZQ",
    ")": "PAR_DER",
    "[": "CORCHETE_IZQ",
    "]": "CORCHETE_DER",
    ";": "SEMICOLON",
    ",": "COMMA",
    ".": "DOT",
    "=": "EQUAL",
    "+": "PLUS",
    "-": "MINUS",
    "*": "MULTIPLY",
    "/": "DIVIDE",
    "==": "EQUAL_EQUAL",
    "!=": "NOT_EQUAL",
    ">": "GREATER",
    "<": "LESS",
    ">=": "GREATER_EQUAL",
    "<=": "LESS_EQUAL",
    "++": "INCREMENT",
    "--": "DECREMENT"
};
```

🔍 Análisis Léxico

Lexer (Lexer.js)

Responsabilidades:

Tokenizar el código fuente Java
Identificar palabras reservadas, identificadores, literales
Reconocer operadores y símbolos
Detectar errores léxicos
Manejar comentarios (// y /* */)

Algoritmo Principal:

```
analizar() {
    while (this.pos < this.texto.length) {
        let char = this.texto[this.pos];

        if (char === " " || char === "\t") { this.avanzar(); continue; }
        if (char === "\n") { this.linea++; this.columna = 1; this.avanzar(); continue; }

        if (this.esLetra(char)) {
            this.recorrerIdentificador();
            continue;
        }

        if (this.esDigito(char)) {
            this.recorrerNumero();
            continue;
        }

        if (char === '"') {
            this.recorrerCadena();
            continue;
        }

        if (char === "'") {
            this.recorrerCaracter();
            continue;
        }

        if (char === "/" && this.texto[this.pos + 1] === "/") {
            this.recorrerComentarioLinea();
            continue;
        }

        if (char === "/" && this.texto[this.pos + 1] === "*") {
            this.recorrerComentarioBloque();
            continue;
        }
```

Estructura:

```
export class Parser {
  constructor(tokens) {
    this.tokens = tokens;
    this.pos = 0;
    this.errors = [];
    this.pythonCode = "";
    this.indent = "";
    this.className = this.extractClassName(tokens);
  }

  extractClassName(tokens) {
    for (let i = 0; i < tokens.length; i++) {
      if (tokens[i].type === "CLASS" && tokens[i + 1]?.type === "IDENTIFICADOR") {
        return tokens[i + 1].value;
      }
    }
    return "Unknown";
  }

  analizar() {
    while (this.pos < this.tokens.length) {
      const token = this.tokens[this.pos];

      switch (token.type) {
        case "INT_TYPE":
        case "DOUBLE_TYPE":
        case "FLOAT_TYPE":
        case "CHAR_TYPE":
        case "BOOLEAN_TYPE":
          this.declaracionVariable();
          break;

        case "IF":
          this.traducirIf();
          break;
```

Métodos Principales:

```
declaracionVariable() {
  const tipo = this.tokens[this.pos].type;
  const tipoOriginal = this.tokens[this.pos].value;
  this.pos++;

  const id = this.tokens[this.pos];
  if (!id || id.type !== "IDENTIFICADOR") {
    this.errors.push(
      new Error(
        "Sintáctico",
        id?.value || "EOF",
        "Se esperaba un identificador",
        id?.line || 0,
        id?.column || 0
      )
    );
    return;
  }

  this.pos++;

  const siguiente = this.tokens[this.pos];

  if (siguiente?.value === ";") {
    let valorDefecto = "None";
    if (tipo === "INT_TYPE" || tipo === "DOUBLE_TYPE" || tipo === "FLOAT_TYPE") {
      valorDefecto = "0";
    } else if (tipo === "BOOLEAN_TYPE") {
      valorDefecto = "False";
    } else if (tipo === "STRING_TYPE") {
      valorDefecto = '""';
```

Traduce declaraciones de variables Java a Python.

```
asignacionVariable() {
  const id = this.tokens[this.pos];
  this.pos++;

  const siguiente = this.tokens[this.pos];

  if (siguiente?.value === "++" || siguiente?.type === "INCREMENT") {
    this.pythonCode += `${this.indent}${id.value} += 1\n`;
    this.pos++;
    if (this.tokens[this.pos]?.value === ";") this.pos++;
    return;
  }

  if (siguiente?.value === "--" || siguiente?.type === "DECREMENT") {
    this.pythonCode += `${this.indent}${id.value} -= 1\n`;
    this.pos++;
    if (this.tokens[this.pos]?.value === ";") this.pos++;
    return;
  }

  if (!siguiente || siguiente.value !== "=") {
    this.pos--;
    return;
  }
  this.pos++;

  let expresion = "";
  while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ";") {
    const tok = this.tokens[this.pos];
    if (tok.type === "STRING") {
      expresion += `"${tok.value}" `;
    } else if (tok.type === "CHAR") {
      const charValue = tok.value.replace(/'/g, '');
      expresion += `"${charValue}" `;
    } else if (tok.value === "true" || tok.type === "TRUE") {
      expresion += "True ";
    } else if (tok.value === "false" || tok.type === "FALSE") {
      expresion += "False ";
```

Traduce asignaciones y operadores de incremento/decremento.

```
traducirFor() {
  this.pos++;
  if (this.tokens[this.pos]?.value !== "(") return;
  this.pos++;

  this.pos++;
  const variable = this.tokens[this.pos++];
  this.pos++;
  const inicio = this.tokens[this.pos++];
  this.pos++;

  this.pos++;
  this.pos++;
  const limite = this.tokens[this.pos++];
  this.pos++;

  this.pos++;
  if (this.tokens[this.pos]?.value === "++") this.pos++;
  if (this.tokens[this.pos]?.value === "+") this.pos++;

  if (this.tokens[this.pos]?.value === ")") this.pos++;
  if (this.tokens[this.pos]?.value === "{") this.pos++;

  this.pythonCode += `${this.indent}for ${variable.value} in range(${inicio.value}, ${limite.value} + 1)
  this.indent += "    ";

  while (
    this.pos < this.tokens.length &&
    this.tokens[this.pos].value !== "}"
  ) {
    this.traducirLineaBloque();
  }

  this.indent = this.indent.slice(0, -4);
  if (this.tokens[this.pos]?.value === "}") this.pos++;
}
```

Traduce bucles for a range() de Python.

```
traducirWhile() {
  this.pos++;
  if (!this.tokens[this.pos] || this.tokens[this.pos].value !== "(") {
    this.errors.push(
      new Error("Sintáctico", "while", "Se esperaba '(' después de while", 0, 0)
    );
    return;
  }

  this.pos++;
  let condicion = "";

  while (this.pos < this.tokens.length && this.tokens[this.pos].value !== ")") {
    const tok = this.tokens[this.pos];
    const val = tok.value;

    if (val === "=" && condicion.trim().endsWith("=")) {
      this.pos++;
      continue;
    }

    if (tok.type === "CHAR") {
      const charValue = val.replace(/'/g, '');
      condicion += `"${charValue}" `;
    } else if (tok.value === "true" || tok.type === "TRUE") {
      condicion += "True ";
    } else if (tok.value === "false" || tok.type === "FALSE") {
      condicion += "False ";
    } else {
      condicion += val + " ";
    }
    this.pos++;
  }
```

Traduce bucles while a Python.

Traducción:

Reglas de traducción:

Java	Python	Notas
`int x = 5;`	`x = 5`	Sin tipo explícito
`double y = 3.14;`	`y = 3.14`	Sin tipo explícito
`char c = 'A';`	`c = "A"`	Comillas simples → dobles
`boolean b = true;`	`b = True`	Capitalización
`x++;`	`x += 1`	Operador de incremento
`x--;`	`x -= 1`	Operador de decremento
`if (x > 5) { }`	`if x > 5:`	Sin paréntesis, con `:`
`for (int i=0; i<=10; i++)`	`for i in range(0, 11):`	range() en Python
`while (x < 10) { }`	`while x < 10:`	Sin paréntesis, con `:`
`System.out.println(x);`	`print(x)`	Función print
`"texto" + var`	`"texto" + str(var)`	Conversión explícita

🌐 API Backend

Servidor Express (main.js)

Configuración:
const app = express();
const PORT = 4000;

app.use(express.json());
app.use(cors());
Endpoint Principal:
POST /analizar
Request:
{
  "code": "public class Main { ... }"
}
Response:
{
  "tokens": [
    { "type": "PUBLIC", "value": "public", "line": 1, "column": 1 },
    ...
  ],
  "lexicalErrors": [
    { "type": "Léxico", "value": "@", "message": "Carácter no reconocido", "line": 5, "column": 8 }
  ],
  "syntaxErrors": [
    { "type": "Sintáctico", "value": "}", "message": "Se esperaba ';'", "line": 10, "column": 5 }
  ],
  "pythonCode": "x = 10\nprint(x)"
}

DIAGRAMA DE CLASES

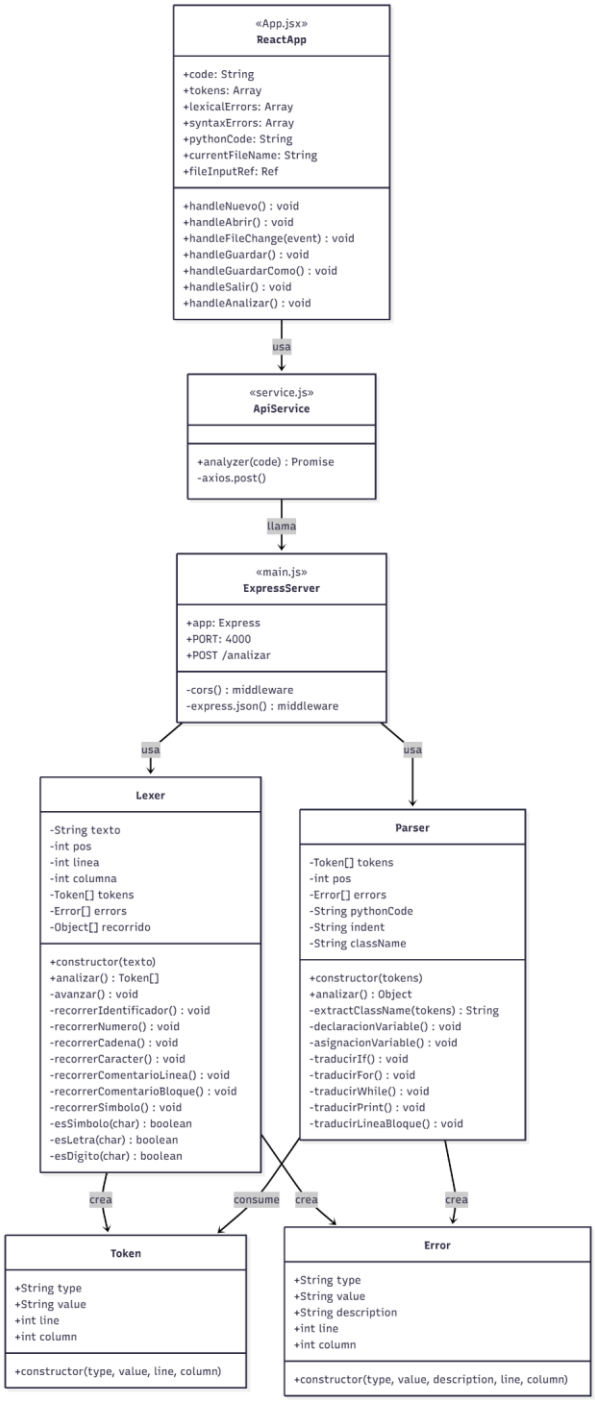
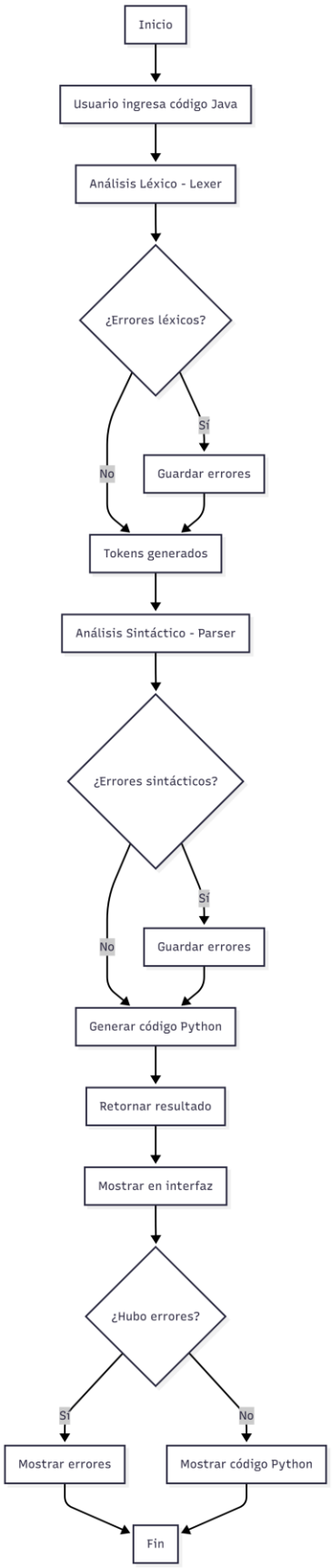


DIAGRAMA DE FLUJO



Desarrollo de Juan Reyes