



# MANUAL TÉCNICO

TourneyJS: Analizador  
Léxico para Gestión de  
Torneos Deportivos

LFP

CARNET 202407596

NOMBRE: Jefferson Ajcúc

# MANUAL TÉCNICO — PROYECTO1 TOURNEYJS: ANALIZADOR LÉXICO PARA GESTIÓN DE TORNEOS DEPORTIVOS )

Alcance: Analizador léxico del lenguaje de torneos +  
generación de reportes HTML y bracket en DOT y SVG.

Tecnologías: JS puro (navegador + Node para CLI). Sin  
frameworks ni generadores de analizadores.

Estructura de carpetas (raíz = Proyecto1/):

documentacion/ ← aquí guarda PDF del Manual Técnico y de  
Usuario

entradas/ ← .txt de prueba

plantillas/ ← tokens\_template.html, errores\_template.html

pruebas/ ← casos de prueba (txt) y resultados esperados

salidas/ ← HTML generados por CLI

src/ ← núcleo (léxico, reportes, graphviz, plantillas)

vista\_consola/ ← CLI para generar tokens/errores

Vista\_web/ ← index.html + app.js (UI)

# 1. Lenguaje soportado

## 1.1 Tokens

- Palabras reservadas: TORNEO, EQUIPOS, ELIMINACION, EQUIPO, JUGADOR, GOLEADOR, FASE, PARTIDO, VS, CUARTOS, SEMIFINAL, FINAL, LOCAL, VISITANTE.
- Atributos: NOMBRE, EQUIPOS, JUGADORES, POSICION, NUMERO, EDAD, RESULTADO, GOLEADORES, MINUTO, PAIS, SEDE.
- Identificador: letra/\_/acentos inicial; continúa con letra/dígito/\_/acentos.
- Número: 0-9 (enteros).
- Cadena: "... " (sin escapes).
- Símbolos de un carácter: { } [ ] ( ) , : ; -.
- Espacios y saltos: se ignoran.

## 1.2 Reglas importantes

- Maximal munch: siempre se consume el lexema más largo posible.
- Prioridad de clasificación (al cerrar un identificador):
- Si  $UPPER(\text{lexema}) \in \text{RESERVED} \rightarrow \text{Palabra Reservada}$
- Si  $UPPER(\text{lexema}) \in \text{ATTRIBUTES} \rightarrow \text{Atributo}$
- Si no,  $\rightarrow$  Identificador
- Errores léxicos (proyecto 1):
- Carácter no reconocido.
- Cadena sin cierre.
- (Errores de “falta de símbolo esperado”, etc., son sintácticos y no se validan en P1)

## 2. Diseño del AFD

### 2.1 Definición formal

- $A = (\Sigma, Q, \delta, q_0, F)$
- $\Sigma$ : letras A-Z/a-z, ÁÉÍÓÚáéíóúÑñ, \_, dígitos, " , { } [ ] ( ) , : ; - , espacio y saltos.
- $Q$ : {S, IN\_ID, IN\_NUM, IN\_STR, STR\_OK}.
- $q_0$ : S.
- $F$ : aceptación al cortar en IN\_ID, IN\_NUM, STR\_OK y símbolos de 1 char.

### 2.2 Transiciones (resumen)

- $S \rightarrow$
  - $" \rightarrow \text{IN\_STR}$
  - $0-9 \rightarrow \text{IN\_NUM}$
  - $\text{letra}/\_/\text{acentos} \rightarrow \text{IN\_ID}$
  - $\{ \} [ ] ( ) , : ; - \rightarrow \text{token inmediato}$
  - $\text{espacio/salto} \rightarrow S$
  - $\text{otro} \rightarrow \text{error}$
  - $\text{IN\_STR} \rightarrow " \rightarrow \text{STR\_OK}$  (acepta "Cadena"); cualquier otro  $\rightarrow \text{IN\_STR}$ ; EOF sin "  $\rightarrow \text{error}$ .
  - $\text{IN\_NUM} \rightarrow \text{dígito} \rightarrow \text{IN\_NUM}$ , de lo contrario cortar (token "Número").
  - $\text{IN\_ID} \rightarrow \text{letra}|\text{dígito}|\_|\text{acentos} \rightarrow \text{IN\_ID}$ , de lo contrario cortar y reclasificar.
- Complejidad:  $O(n)$  tiempo,  $O(1)$  memoria adicional.

## 3. Implementación (archivos y responsabilidades)

### 3.1 Léxico — src/lexer\_core.js

Responsabilidad: convertir el texto en { tokens, errors } con (línea, columna).

Puntos clave del código (coloca estos fragmentos en el manual si necesitas citas):

(A) Bucle principal AFD

Archivo: src/lexer\_core.js — función `lex(text)` → bucle `while (i < text.length)` con dispatch por clase de carácter.

```
// [CITA] AFD — bucle de escaneo y dispatch por clase de
while(i < text.length) {
  const ch = peek();
  // WS → advance
  // símbolo 1 char → emitir token
  // '"' → IN_STR
  // dígito → IN_NUM
  // letra/_/acentos → IN_ID
  // otro → error léxico
}
```

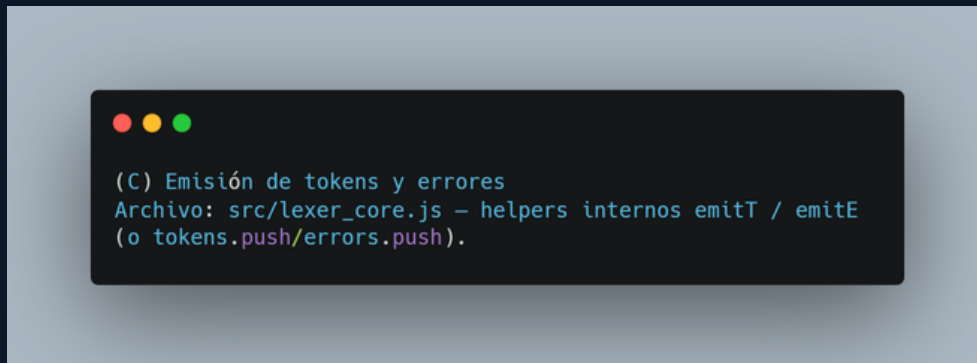
(B) Clasificadores sin regex

Archivo: src/lexer\_core.js — helpers `isWS`, `isDigit`, `isLetter`, `isIdChar`.

```
// [CITA] Clasificadores sin regex
function isWS(ch){ /* ' ', '\t', '\n', '\r' */ }
function isDigit(ch){ /* '0'..'9' */ }
function isLetter(ch){ /* A..Z, a..z, ÁÉÍÓÚáéíóúÑñ, '_' */ }
function isIdChar(ch){ return isLetter(ch) || isDigit(ch); }
```

### (C) Emisión de tokens y errores

Archivo: `src/lexer_core.js` — helpers internos `emitT` / `emitE` (o `tokens.push/errors.push`).



### Notas de mantenimiento:

- Añadir una reservada/atributo = meterla al Set (sin tocar el AFD).
- Acentos soportados en `isLetter`.
- Si reemplazaste por la versión “AFD explícito” que te di, las firmas y el flujo se mantienen.

## 3.2 Reportes — `src/report_core.js`

Responsabilidad: construir un modelo desde los tokens (no desde texto) y producir filas para HTML.

Flujo:

#### 1. `analyzeTokens(tokens)`:

- Extrae nombre y sede (atributos).
- Reconoce EQUIPO/JUGADOR (+edad) y calcula estadísticos.
- Reconoce fases CUARTOS/SEMIFINAL/FINAL.
- Reconoce PARTIDO "A" VS "B" [resultado: "x-y"; goleador: "..."].
- Aplica resultados: jugados, ganados, GF/GC, diferencia.

#### 2. `bracketRows` / `statsRows` / `scorersRows` / `infoRows`:

convierten el modelo a filas para las plantillas.



## D) Firma pública y retornos

Archivo: src/report\_core.js — API:

```
// [CITA] API pública de reportes
const model = TourneyReports.analyzeTokens(tokens);
TourneyReports.bracketRows(model);
TourneyReports.statsRows(model);
TourneyReports.scorersRows(model);
TourneyReports.infoRows(model);
```

## (E) Heurística de score

Archivo: src/report\_core.js — utils scoreFrom("x-y").

```
// [CITA] Reconocimiento de marcador y aplicación a equipos
const s = scoreFrom(cadenaScore); // {a,b}
if (s) applyScore(teamA, teamB, s);
```

## 3.3 HTML “reportes” — src/html\_templates.js

Responsabilidad: encapsular el look de las tablas en HTML.

### (F) API de plantillas

Archivo: src/html\_templates.js — funciones:

```
3.3 HTML “reportes” — src/html_templates.js

Responsabilidad: encapsular el look de las tablas en HTML.

(F) API de plantillas
Archivo: src/html_templates.js — funciones:
```


## 3.4 Bracket (DOT + SVG) — src/graphviz\_builder.js

Responsabilidad:

- buildBracketDOT(model): produce .dot compatible con Graphviz.
- buildBracketSVG(model): produce SVG para mostrar la gráfica sin dependencias.

(G) Parámetros de layout editables


Archivo: src/graphviz\_builder.js — constantes en buildBracketSVG:



```
// [CITA] Ajuste rápido de layout
const nodeW = 200, nodeH = 70; // tamaño de cajas
const colGap = 140, rowGap = 40; // separaciones
```

(H) API pública

Archivo: src/graphviz\_builder.js — firmas:



```
// [CITA] Firmas
TourneyGraph.buildBracketDOT(model);
TourneyGraph.buildBracketSVG(model);
```

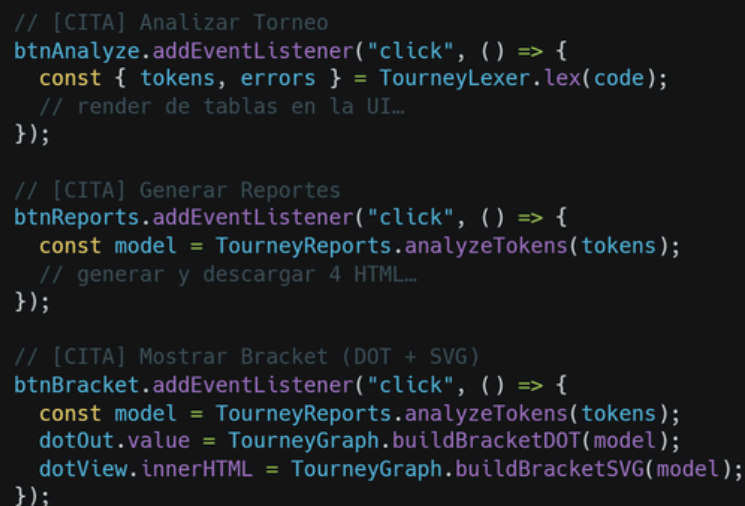


### 3.5 UI (Web) — Vista\_web/index.html + Vista\_web/app.js

Responsabilidad: Interfaz minimalista (B/N), flujo: Cargar → Analizar → Reportes → Bracket.

(I) Eventos principales

Archivo: Vista\_web/app.js — handlers de botones:



```
// [CITA] Analizar Torneo
btnAnalyze.addEventListener("click", () => {
  const { tokens, errors } = TourneyLexer.lex(code);
  // render de tablas en la UI...
});

// [CITA] Generar Reportes
btnReports.addEventListener("click", () => {
  const model = TourneyReports.analyzeTokens(tokens);
  // generar y descargar 4 HTML...
});

// [CITA] Mostrar Bracket (DOT + SVG)
btnBracket.addEventListener("click", () => {
  const model = TourneyReports.analyzeTokens(tokens);
  dotOut.value = TourneyGraph.buildBracketDOT(model);
  dotView.innerHTML = TourneyGraph.buildBracketSVG(model);
});
```

### 3.6 CLI (consola) — vista\_consola/cli.js

Responsabilidad: Generar salidas/tokens.html y salidas/errores.html a partir de un .txt.


(J) Uso



```
node vista_consola/cli.js entradas/ejemplo.txt
# genera: salidas/tokens.html salidas/errores.html
```

## (K) Pipeline CLI

Archivo: vista\_consola/cli.js — flujo: leer → lex → inyectar en plantillas → escribir.



```
// [CITA] CLI: pipeline
const text = read(inFile);
const { tokens, errors } = lex(text);
// tokens_template/errores_template → reemplazar
{{TABLE_ROWS}}
```

## 4. Errores léxicos (tabla de reporte)

Cada error incluye: No., Lexema, Tipo (Token inválido), Descripción, Línea, Columna.

- Carácter no reconocido → se reporta con el carácter en lexeme.
- Cadena sin cierre → lexeme = prefijo leído ("...), desc: "Cadena sin cierre".

## 5. Pruebas

### 5.1 Plan de pruebas

- Happy path: ejemplos completos que generan los 4 reportes.
- Cadenas sin cierre.
- Identificadores con acentos.
- Marcadores "x-y" válidos y "Pendiente".
- Fases: solo CUARTOS/SEMIFINAL/FINAL (no octavos).

## 6. Mantenimiento / Extensión

- Agregar palabra reservada/atributo: añadir al Set en `src/lexer_core.js`.
- Nuevos símbolos de 1 char: añadir a `SYMBOLS`.
- Más fases: ampliar `PHASES` en `src/graphviz_builder.js` y en `report_core.js`.
- Más estadísticas: derivar del model de `report_core.js`.

## 7. Despliegue / Ejecución

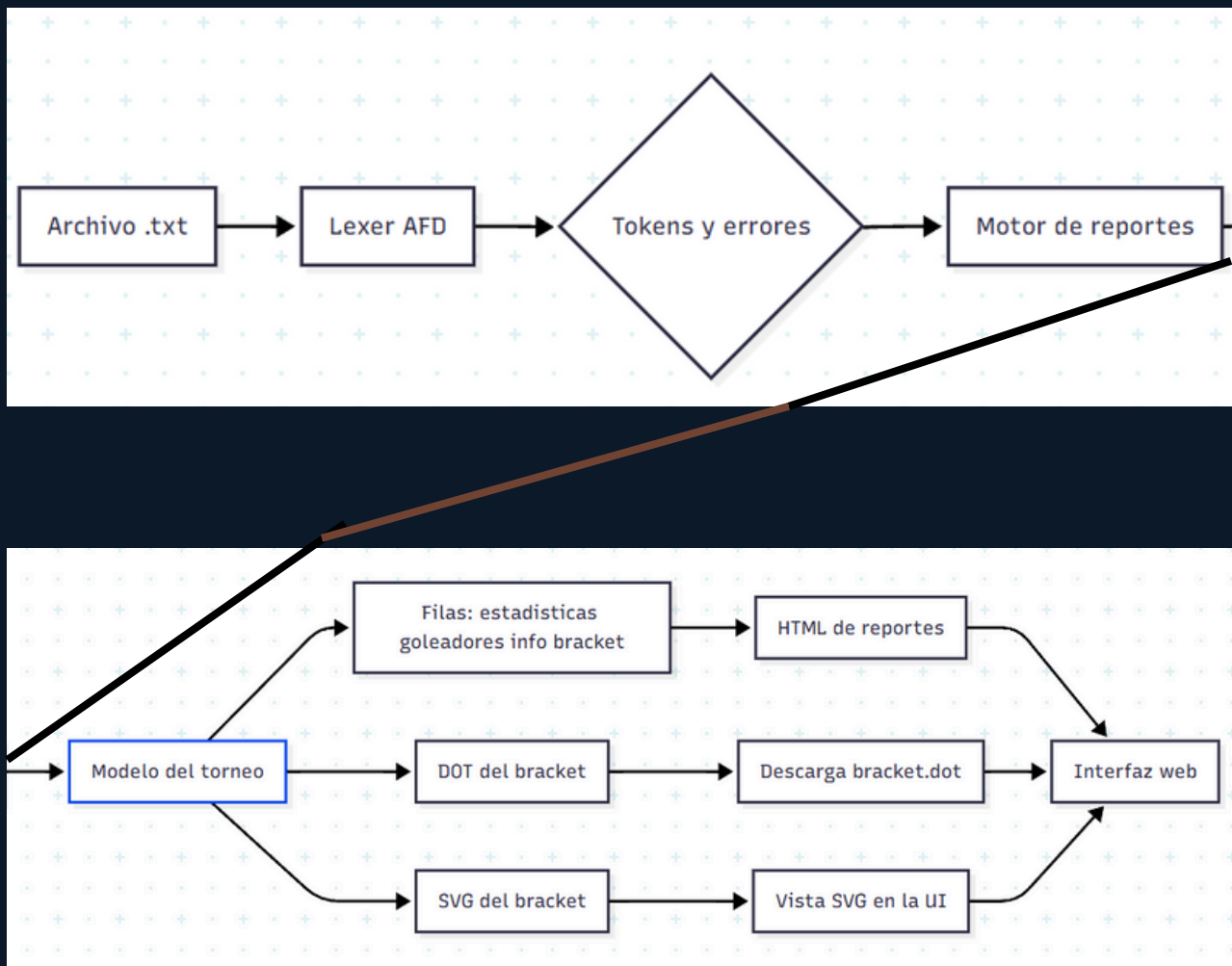
- Web: abrir `Vista_web/index.html` con Live Server de VS Code (o cualquier server estático).
- CLI: `node vista_consola/cli.js entradas/archivo.txt`.

Requisitos: Node 18+ para CLI y un navegador moderno para la UI.

## 8. Alineación con Rubrica

- Análisis léxico propio (AFD), sin generadores, carácter por carácter.
- Reportes HTML (4): Bracket, Estadísticas por equipo, Goleadores, Info general.
- Graphviz: `.dot` descargable + SVG en la UI.
- Interfaz simple (B/N) y segmentación por módulos.
- Documentación: este Manual Técnico + Manual de Usuario (pendiente si no lo subiste).

# DIAGRAMA DE FLUJO



# DIAGRAMA DE CLASES

