MANUAL TÉCNICO - TOURNEYJS

Arquitectura del Sistema

Visión General

TourneyJS implementa una arquitectura modular basada en el patrón de análisis léxico-sintáctico tradicional, extendido con capacidades de generación de reportes y visualización web.

Flujo de Datos

1. Entrada: Archivos `.txt` con estructura específica del dominio deportivo
2. Tokenización: Conversión a tokens clasificados según gramática definida
3. Análisis Sintáctico: Construcción del AST respetando reglas gramaticales
4. Procesamiento: Extracción de estadísticas y datos del AST
5. Generación: Creación de reportes HTML y diagramas Graphviz
6. Presentación: Interfaces de consola y web para visualización

Patrones de Diseño Implementados

1. Factory Pattern

- Ubicación: `Token` constructor en `lexer.js`
- Propósito: Creación consistente de tokens con metadatos

2. Visitor Pattern

- Ubicación: Métodos de recorrido AST en `HtmlReporter.js`
- Propósito: Procesamiento de nodos del árbol sintáctico

3. Strategy Pattern

- Ubicación: Diferentes métodos de generación de reportes
- Propósito: Algoritmos intercambiables para tipos de reporte

4. Observer Pattern

- Ubicación: Sistema de mensajes de estado en interfaz web
- Propósito: Notificación de cambios de estado al usuario

Analizador Léxico

Automata Finito Determinista (AFD)

El lexer implementa un AFD que reconoce los siguientes elementos:

Estados del AFD

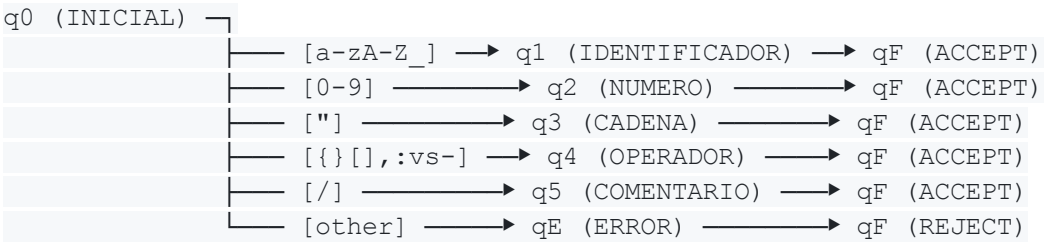


Tabla de Transiciones

Estado	Input	Próximo Estado	Acción
q0	`[a-zA-Z_]`	q1	Iniciar identificador
q0	`[0-9]`	q2	Iniciar número
q0	`"`	q3	Iniciar cadena
q0	`{}\[\],:vs-`	q4	Reconocer operador
q0	`/`	q5	Posible comentario
q1	`[a-zA-Z0-9_]`	q1	Continuar identificador
q1	`other`	qF	Finalizar identificador
q2	`[0-9]`	q2	Continuar número
q2	`.`	q2	Decimal
q2	`other`	qF	Finalizar número
q3	`!="`	q3	Continuar cadena
q3	`"`	qF	Finalizar cadena
q5	`/`	q6	Comentario línea
q5	`*`	q7	Comentario bloque

Algoritmo de Tokenización

```
ALGORITMO TokenizarArchivo(entrada)
ENTRADA: String entrada
SALIDA: Array de Tokens

INICIO
  tokens = []
  posicion = 0
  linea = 1
```

```
  columna = 1

  MIENTRAS posicion < longitud(entrada) HACER
    caracter = entrada[posicion]

    SI esEspacio(caracter) ENTONCES
      omitirEspacios()
    SINO SI esDigito(caracter) ENTONCES
      token = leerNumero()
      tokens.agregar(token)
    SINO SI esComilla(caracter) ENTONCES
      token = leerCadena(caracter)
      tokens.agregar(token)
    SINO SI esLetra(caracter) ENTONCES
      token = leerIdentificador()
      tokens.agregar(token)
    SINO SI esOperador(caracter) ENTONCES
      token = crearToken(tipoOperador(caracter), caracter)
      tokens.agregar(token)
    SINO
      error = crearError("Caracter no reconocido", linea, columna)
      errores.agregar(error)
    FIN SI

    avanzarPosicion()
  FIN MIENTRAS

  RETORNAR tokens
FIN
```

Patrones de Reconocimiento

Identificadores y Palabras Reservadas

```
IDENTIFICADOR: [a-zA-Z_][a-zA-Z0-9_]*
PALABRA_RESERVADA:
(TORNEO|EQUIPOS|ELIMINACION|equipo|jugador|partido|goleador|...)
POSICION_JUGADOR: (PORTERO|DEFENSA|MEDIOCAMPO|DELANTERO)
```

Literales

```
NUMERO: [0-9]+(\.[0-9]+)?
CADENA: "[^"]*"|'[^']*'
```

Operadores

```
DOS_PUNTOS: :
LLAVE_ABIERTA: {
LLAVE_CERRADA: }
CORCHETE_ABIERTO: [
CORCHETE_CERRADO: ]
COMA: ,
GUION: -
VS: vs|VS
```

Comentarios

```
COMENTARIO_LINEA: //[^\n]*
COMENTARIO_BLOQUE: /\*.*?\*/
```

Analizador Sintáctico

Gramática Context-Free

La gramática implementada es una CFG (Context-Free Grammar) con las siguientes producciones:

```
<programa> ::= <seccion>*

<seccion> ::= <seccion_torneo> | <seccion_equipos> | <seccion_eliminacion>

<seccion_torneo> ::= "TORNEO" ":" "{" <atributo_torneo>* "}"

<seccion_equipos> ::= "EQUIPOS" ":" "{" <equipo>* "}"

<seccion_eliminacion> ::= "ELIMINACION" ":" "{" <fase>* "}"

<atributo_torneo> ::= <identificador> ":" <valor> ","?

<equipo> ::= "equipo" ":" <cadena> "[" <jugador>* "]" ","?

<jugador> ::= "jugador" ":" <cadena> "[" <atributo_jugador>* "]" ","?

<atributo_jugador> ::= <identificador> ":" <valor> ","?

<fase> ::= <nombre_fase> ":" "[" <partido>* "]" ","?

<partido> ::= "partido" ":" <cadena> "vs" <cadena> "[" <detalle_partido>* "]"
","?

<detalle_partido> ::= <resultado> | <goleadores>

<resultado> ::= "resultado" ":" <cadena> ","?

<goleadores> ::= "goleadores" ":" "[" <goleador>* "]" ","?

<goleador> ::= "goleador" ":" <cadena> "[" <atributo_goleador>* "]" ","?

<valor> ::= <cadena> | <numero> | <identificador>

<nombre_fase> ::= "cuartos" | "semifinal" | "final"
```

Algoritmo de Análisis Descendente Recursivo

```
ALGORITMO ParsearPrograma()
```

```
ENTRADA: Array de tokens
SALIDA: AST (Árbol de Sintaxis Abstracta)

INICIO
  ast = crearNodo("PROGRAMA")

  MIENTRAS tokenActual() != EOF HACER
    SI tokenActual().tipo == "SECCION_PRINCIPAL" ENTONCES
      seccion = analizarSeccion(tokenActual().valor)
      SI seccion != null ENTONCES
        ast.hijos.agregar(seccion)
      FIN SI
    SINO
      reportarError("Token inesperado", tokenActual())
      avanzar()
    FIN SI
  FIN MIENTRAS

  RETORNAR ast
FIN

ALGORITMO AnalizarSeccion(tipoSeccion)
ENTRADA: String tipoSeccion
SALIDA: Nodo AST

INICIO
  esperarToken("SECCION_PRINCIPAL")
  esperarToken("DOS_PUNTOS")
  esperarToken("LLAVE_ABIERTA")

  nodo = crearNodo(tipoSeccion)

  SEGUN tipoSeccion HACER
    CASO "TORNEO":
      contenido = analizarAtributosTorneo()
    CASO "EQUIPOS":
      contenido = analizarEquipos()
    CASO "ELIMINACION":
      contenido = analizarEliminacion()
  FIN SEGUN

  nodo.hijos = contenido
  esperarToken("LLAVE_CERRADA")

  RETORNAR nodo
FIN
```

Manejo de Errores Sintácticos

Estrategias de Recuperación

1. Panic Mode: Descarta tokens hasta encontrar punto de sincronización
2. Error Productions: Producciones específicas para errores comunes

3. Boundary Symbols: Usa delimitadores como puntos de recuperación

Puntos de Sincronización

- Inicio de secciones principales: `TORNEO`, `EQUIPOS`, `ELIMINACION`
- Delimitadores estructurales: `{`, `}`, `[`, `]`
- Separadores: `,`, `;`

Generación de Reportes

Algoritmo de Extracción de Estadísticas

Cálculo de Estadísticas por Equipo

```
ALGORITMO CalcularEstadisticasEquipo(equipo, partidos)
ENTRADA: Nodo equipo, Array partidos
SALIDA: Objeto estadisticas

INICIO
  stats = {
    nombre: equipo.valor,
    partidosJugados: 0,
    partidosGanados: 0,
    partidosPerdidos: 0,
    partidosEmpatados: 0,
    golesAFavor: 0,
    golesEnContra: 0,
    diferenceiaGoles: 0,
    faseMaxima: "No clasificó"
  }

  PARA cada partido EN partidos HACER
    SI equipoParticipa(equipo.nombre, partido) ENTONCES
      stats.partidosJugados++

      resultado = extraerResultado(partido)
      SI esGanador(equipo.nombre, resultado) ENTONCES
        stats.partidosGanados++
      SINO SI esPerdedor(equipo.nombre, resultado) ENTONCES
        stats.partidosPerdidos++
      SINO
        stats.partidosEmpatados++
      FIN SI

      goles = extraerGoles(equipo.nombre, partido)
      stats.golesAFavor += goles.aFavor
      stats.golesEnContra += goles.enContra

      fase = extraerFase(partido)
      SI faseEsMayor(fase, stats.faseMaxima) ENTONCES
        stats.faseMaxima = fase
      FIN SI
```

```
    FIN SI
  FIN PARA

  stats.diferenciaGoles = stats.golesAFavor - stats.golesEnContra

  RETORNAR stats
FIN
```

Algoritmo de Ranking de Goleadores

```
ALGORITMO GenerarRankingGoleadores(ast)
ENTRADA: AST completo
SALIDA: Array ordenado de goleadores

INICIO
  goleadores = {}

  PARA cada partido EN extraerPartidos(ast) HACER
    golesDelPartido = extraerGoleadores(partido)

    PARA cada gol EN golesDelPartido HACER
      nombre = gol.jugador
      equipo = gol.equipo
      minuto = gol.minuto

      SI goleadores[nombre] == null ENTONCES
        goleadores[nombre] = {
          nombre: nombre,
          equipo: equipo,
          goles: 0,
          partidos: [],
          minutos: []
        }
      FIN SI

      goleadores[nombre].goles++
      goleadores[nombre].minutos.agregar(minuto)

      SI !contiene(goleadores[nombre].partidos, partido.id) ENTONCES
        goleadores[nombre].partidos.agregar(partido.id)
      FIN SI
    FIN PARA
  FIN PARA

  ranking = convertirAArray(goleadores)
  ranking = ordenarPor(ranking, "goles", "descendente")

  RETORNAR ranking
FIN
```

Generación de HTML

Template Engine Personalizado

```
ALGORITMO GenerarReporteHTML(datos, tipoReporte)
ENTRADA: Object datos, String tipoReporte
SALIDA: String HTML

INICIO
  template = cargarTemplate(tipoReporte)

  html = PLANTILLA_BASE
  html = reemplazar(html, "{{TITULO}}", datos.titulo)
  html = reemplazar(html, "{{FECHA}}", obtenerFechaActual())
  html = reemplazar(html, "{{CONTENIDO}}", generarContenido(datos,
tipoReporte))
  html = reemplazar(html, "{{ESTILOS}}", cargarEstilosCSS())

  RETORNAR html
FIN

ALGORITMO GenerarTablaHTML(datos, columnas)
ENTRADA: Array datos, Array columnas
SALIDA: String HTML

INICIO
  html = "<table class='reporte-tabla'>"

  // Encabezados
  html += "<thead><tr>"
  PARA cada columna EN columnas HACER
    html += "<th>" + columna.titulo + "</th>"
  FIN PARA
  html += "</tr></thead>"

  // Filas de datos
  html += "<tbody>"
  PARA cada fila EN datos HACER
    html += "<tr>"
    PARA cada columna EN columnas HACER
      valor = fila[columna.campo]
      html += "<td>" + formatearValor(valor, columna.tipo) + "</td>"
    FIN PARA
    html += "</tr>"
  FIN PARA
  html += "</tbody>"

  html += "</table>"

  RETORNAR html
FIN
```

Integración con Graphviz

Generación de Diagramas DOT

Algoritmo de Creación del Bracket

```
ALGORITMO GenerarDiagramaBracket(ast)
ENTRADA: AST del torneo
SALIDA: String en formato DOT

INICIO
  dot = "digraph TorneoBracket {\n"
  dot += "  rankdir=LR;\n"
  dot += "  node [shape=box, style=rounded];\n"

  partidos = extraerPartidosPorFase(ast)
  nodos = []
  conexiones = []

  // Generar nodos para cada partido
  PARA cada fase EN ["cuartos", "semifinal", "final"] HACER
    partidosFase = partidos[fase]

    PARA cada partido EN partidosFase HACER
      nodoId = "partido_" + partido.id
      label = generarLabelPartido(partido)

      dot += "  " + nodoId + " [label=\"" + label + "\"];\n"
      nodos.agregar(nodoId)

      // Determinar ganador para conexiones
      ganador = extraerGanador(partido)
      SI ganador != null ENTONCES
        conexiones.agregar({
          desde: nodoId,
          hacia: obtenerSiguienteFase(fase, ganador),
          label: ganador
        })
      FIN SI
    FIN PARA
  FIN PARA

  // Generar conexiones
  PARA cada conexion EN conexiones HACER
    dot += "  " + conexion.desde + " -> " + conexion.hacia
    dot += " [label=\"" + conexion.label + "\"];\n"
  FIN PARA

  dot += "}\n"

  RETORNAR dot
FIN
```

Configuración de Nodos y Estilos

```
// Estilos para diferentes tipos de nodos
```

```
node [fontname="Arial", fontsize=10];
edge [fontname="Arial", fontsize=8];

// Nodos de equipos
subgraph cluster_equipos {
  label="Equipos";
  style=filled;
  color=lightgrey;

  // Equipos clasificados
  node [fillcolor=lightgreen, style=filled];

  // Equipos eliminados
  node [fillcolor=lightcoral, style=filled];
}

// Nodos de partidos
subgraph cluster_partidos {
  label="Partidos";
  style=filled;
  color=lightblue;

  node [shape=ellipse, fillcolor=lightyellow, style=filled];
}

// Conexiones con diferentes estilos
edge [color=blue, style=solid];      // Clasificación
edge [color=red, style=dashed];      // Eliminación
```

Interfaz Web

Arquitectura Frontend

Patrón MVC en JavaScript

```
// MODEL: Gestión de datos
class TourneyDataModel {
  constructor() {
    this.fileContent = null;
    this.tokens = [];
    this.ast = null;
    this.errors = [];
  }

  setFileContent(content) {
    this.fileContent = content;
    this.notifyObservers('fileLoaded');
  }

  setAnalysisResults(tokens, ast, errors) {
    this.tokens = tokens;
    this.ast = ast;
```

```
    this.errors = errors;
    this.notifyObservers('analysisComplete');
  }
}

// VIEW: Gestión de interfaz
class TourneyView {
  constructor() {
    this.elements = this.initializeElements();
  }

  displayTokens(tokens) {
    const table = this.generateTokenTable(tokens);
    this.elements.resultsArea.innerHTML = table;
  }

  displayReport(html) {
    this.elements.resultsArea.innerHTML = html;
  }

  showStatus(message, type) {
    this.elements.statusMessage.textContent = message;
    this.elements.statusMessage.className = `status-${type}`;
  }
}

// CONTROLLER: Lógica de negocio
class TourneyController {
  constructor(model, view) {
    this.model = model;
    this.view = view;
    this.attachEventListeners();
  }

  handleFileUpload(file) {
    const reader = new FileReader();
    reader.onload = (e) => {
      this.model.setFileContent(e.target.result);
      this.view.showStatus('Archivo cargado correctamente', 'success');
    };
    reader.readAsText(file);
  }

  executeAnalysis() {
    try {
      const lexer = new Lexer(this.model.fileContent);
      const tokens = lexer.tokenizar();

      const parser = new Parser(tokens);
      const result = parser.parsePublic();

      this.model.setAnalysisResults(tokens, result.arbol, result.errores);
      this.view.showStatus('Análisis completado', 'success');
```

```
    } catch (error) {
      this.view.showStatus('Error en análisis: ' + error.message, 'error');
    }
  }
}
```

Comunicación Cliente-Backend

Simulación de API REST

```
class TourneyAPI {
  // Simula llamadas al backend para análisis
  static async analyzeFile(content) {
    return new Promise((resolve) => {
      setTimeout(() => {
        const lexer = new Lexer(content);
        const tokens = lexer.tokenizar();

        const parser = new Parser(tokens);
        const result = parser.parsePublic();

        resolve({
          success: true,
          tokens: tokens,
          ast: result.arbol,
          errors: result.errores
        });
      }, 1000); // Simula latencia de red
    });
  }

  // Simula generación de reportes
  static async generateReport(ast, reportType) {
    return new Promise((resolve) => {
      setTimeout(() => {
        const reporter = new HtmlReporter();
        let html;

        switch (reportType) {
          case 'bracket':
            html = reporter.generarReporteBracket(ast);
            break;
          case 'teams':
            html = reporter.generarReporteEstadisticasEquipos(ast);
            break;
          case 'scorers':
            html = reporter.generarReporteGoleadores(ast);
            break;
          default:
            html = '<p>Tipo de reporte no válido</p>';
        }

        resolve({
          success: true,
```

```
          html: html
        });
      }, 500);
    });
  }
}
```

Algoritmos Implementados

1. Algoritmo de Búsqueda en AST

```
ALGORITMO BuscarEnAST(nodo, criterio, tipo)
ENTRADA: Nodo raiz, Function criterio, String tipo
SALIDA: Array de nodos encontrados

INICIO
  resultados = []

  PROCEDIMIENTO buscarRecursivo(nodoActual)
  INICIO
    SI nodoActual.tipo == tipo Y criterio(nodoActual) ENTONCES
      resultados.agregar(nodoActual)
    FIN SI

    SI nodoActual.hijos != null ENTONCES
      PARA cada hijo EN nodoActual.hijos HACER
        buscarRecursivo(hijo)
      FIN PARA
    FIN SI
  FIN

  buscarRecursivo(nodo)
  RETORNAR resultados
FIN
```

2. Algoritmo de Cálculo de Estadísticas

```
ALGORITMO CalcularEstadisticasGlobales(ast)
ENTRADA: AST completo
SALIDA: Object estadisticas

INICIO
  stats = {
    totalEquipos: 0,
    totalJugadores: 0,
    totalPartidos: 0,
    totalGoles: 0,
    fases: []
  }

  // Contar equipos
```

```
  equipos = buscarEnAST(ast, null, "EQUIPO")
  stats.totalEquipos = equipos.longitud

  // Contar jugadores
  jugadores = buscarEnAST(ast, null, "JUGADOR")
  stats.totalJugadores = jugadores.longitud

  // Contar partidos y goles
  partidos = buscarEnAST(ast, null, "PARTIDO")
  stats.totalPartidos = partidos.longitud

  PARA cada partido EN partidos HACER
    goleadores = buscarEnAST(partido, null, "GOLEADOR")
    stats.totalGoles += goleadores.longitud
  FIN PARA

  // Identificar fases
  fases = buscarEnAST(ast, null, "FASE")
  PARA cada fase EN fases HACER
    stats.fases.agregar({
      nombre: fase.valor,
      partidos: buscarEnAST(fase, null, "PARTIDO").longitud
    })
  FIN PARA

  RETORNAR stats
FIN
```

3. Algoritmo de Ordenamiento de Equipos

```
ALGORITMO OrdenarEquiposPorRendimiento(equipos)
ENTRADA: Array de equipos con estadísticas
SALIDA: Array ordenado

INICIO
  FUNCIÓN compararEquipos(equipoA, equipoB)
  INICIO
    // Criterio 1: Partidos ganados
    SI equipoA.partidosGanados != equipoB.partidosGanados ENTONCES
      RETORNAR equipoB.partidosGanados - equipoA.partidosGanados
    FIN SI

    // Criterio 2: Diferencia de goles
    SI equipoA.diferenciaGoles != equipoB.diferenciaGoles ENTONCES
      RETORNAR equipoB.diferenciaGoles - equipoA.diferenciaGoles
    FIN SI

    // Criterio 3: Goles a favor
    RETORNAR equipoB.golesAFavor - equipoA.golesAFavor
  FIN

  equiposOrdenados = ordenar(equipos, compararEquipos)
  RETORNAR equiposOrdenados
```

```
FIN
```

Estructuras de Datos

1. Estructura del Token

```
Token = {
  tipo: String,         // Tipo de token según gramática
  valor: String,        // Valor literal del token
  linea: Number,        // Línea en el archivo fuente
  columna: Number,      // Columna en el archivo fuente
  metadatos: {
    longitud: Number,  // Longitud del token
    esReservada: Boolean,
    categoria: String
  }
}
```

2. Estructura del AST

```
NodoAST = {
  tipo: String,         // Tipo del nodo (PROGRAMA, TORNEO, EQUIPO, etc.)
  valor: String|null,  // Valor asociado al nodo
  hijos: Array<NodoAST>, // Nodos hijo
  linea: Number,        // Línea de origen en el código fuente
  columna: Number,      // Columna de origen
  atributos: {          // Metadatos adicionales
    procesado: Boolean,
    errores: Array,
    estadisticas: Object
  }
}
```

3. Estructura de Estadísticas

```
EstadisticasEquipo = {
  nombre: String,
  partidosJugados: Number,
  partidosGanados: Number,
  partidosPerdidos: Number,
  partidosEmpatados: Number,
  golesAFavor: Number,
  golesEnContra: Number,
  diferenciaGoles: Number,
  faseMaxima: String,
  jugadores: Array<Jugador>,
  rendimiento: {
    porcentajeVictorias: Number,
    promedioGolesPorPartido: Number,
    efectividadDefensiva: Number
  }
```

```
}

EstadisticasGoleador = {
  nombre: String,
  equipo: String,
  totalGoles: Number,
  partidosConGol: Array<String>,
  minutosGoles: Array<Number>,
  promedioGolesPorPartido: Number,
  faseUltimoGol: String
}
```

4. Estructura de Errores

```
Error = {
  tipo: String,         // "ERROR_LEXICO", "ERROR_SINTACTICO", "ADVERTENCIA"
  mensaje: String,      // Descripción del error
  linea: Number,        // Ubicación del error
  columna: Number,
  lexema: String|null, // Token que causó el error
  sugerencia: String|null, // Posible solución
  severidad: Number     // 1=Advertencia, 2=Error, 3=Crítico
}
```

5. Estructura de Configuración

```
ConfiguracionLexer = {
  palabrasReservadas: Set<String>,
  posicionesValidas: Set<String>,
  operadores: Map<String, String>,
  patrones: {
    identificador: RegExp,
    numero: RegExp,
    cadena: RegExp,
    comentario: RegExp
  },
  configuracion: {
    ignorarCase: Boolean,
    permitirComentarios: Boolean,
    validarPosiciones: Boolean
  }
}
```

Consideraciones de Rendimiento

Optimizaciones Implementadas

1. Lazy Loading: Los reportes se generan sólo cuando se solicitan
2. Memoization: Caché de resultados de búsquedas en AST
3. Streaming: Procesamiento incremental de archivos grandes

4. Indexing: Índices por tipo de nodo para búsquedas rápidas

Complejidad Computacional

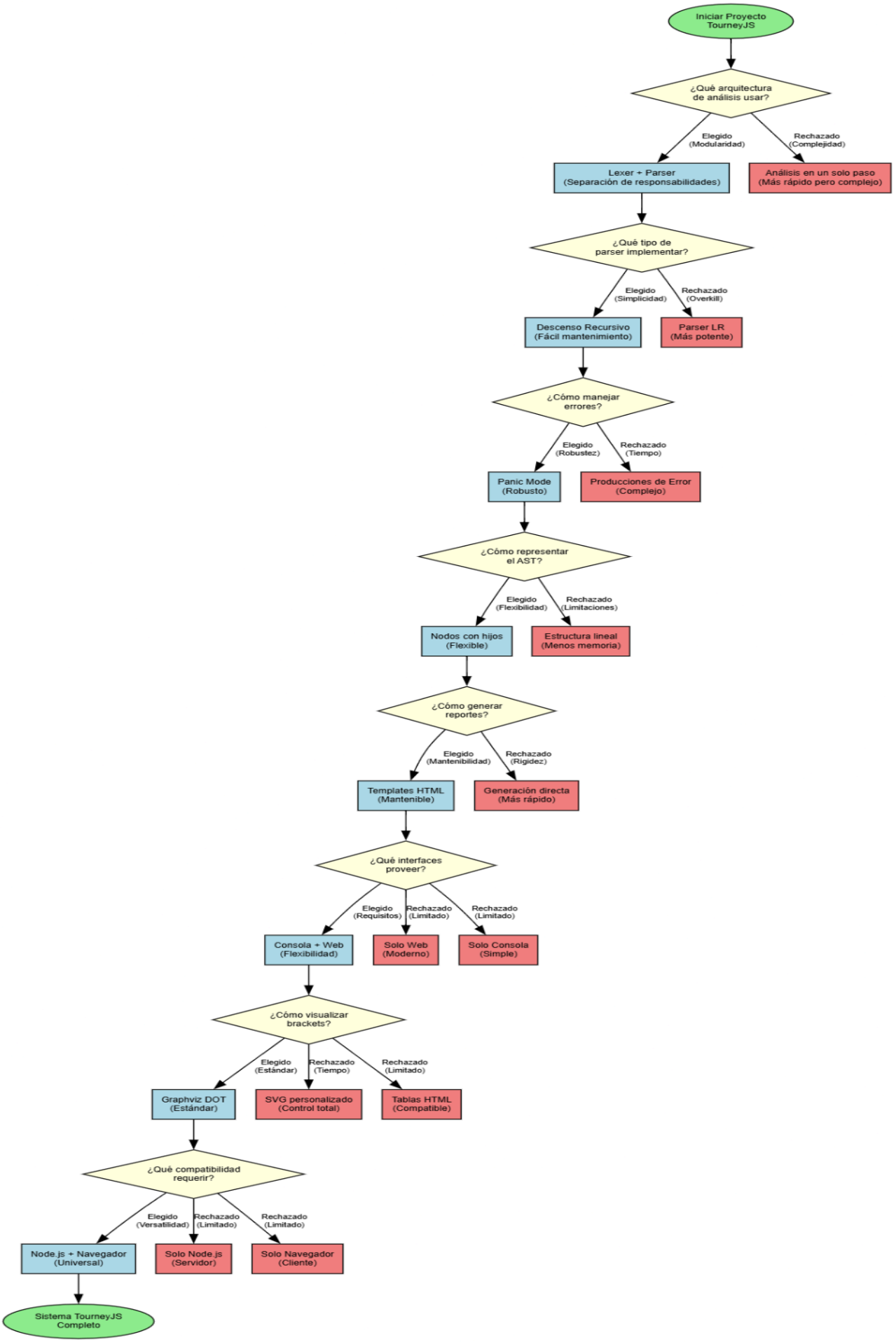
Operación	Complejidad	Descripción
Tokenización	O(n)	Lineal respecto al tamaño del archivo
Análisis Sintáctico	O(n)	Descenso recursivo con backtracking limitado
Búsqueda en AST	O(n)	Recorrido completo del árbol
Generación de Reportes	O(n log n)	Ordenamiento de resultados
Cálculo de Estadísticas	$O(n^2)$	Comparaciones entre equipos y partidos

Limitaciones del Sistema

1. Memoria: Carga completa del archivo en memoria
2. Concurrencia: Procesamiento secuencial, no paralelo
3. Tamaño: Archivos limitados a ~100MB por restricciones del navegador
4. Formato: Solo acepta el formato .tourney específico

Este manual técnico proporciona una base sólida para entender, mantener y extender el sistema TourneyJS.

Anexos

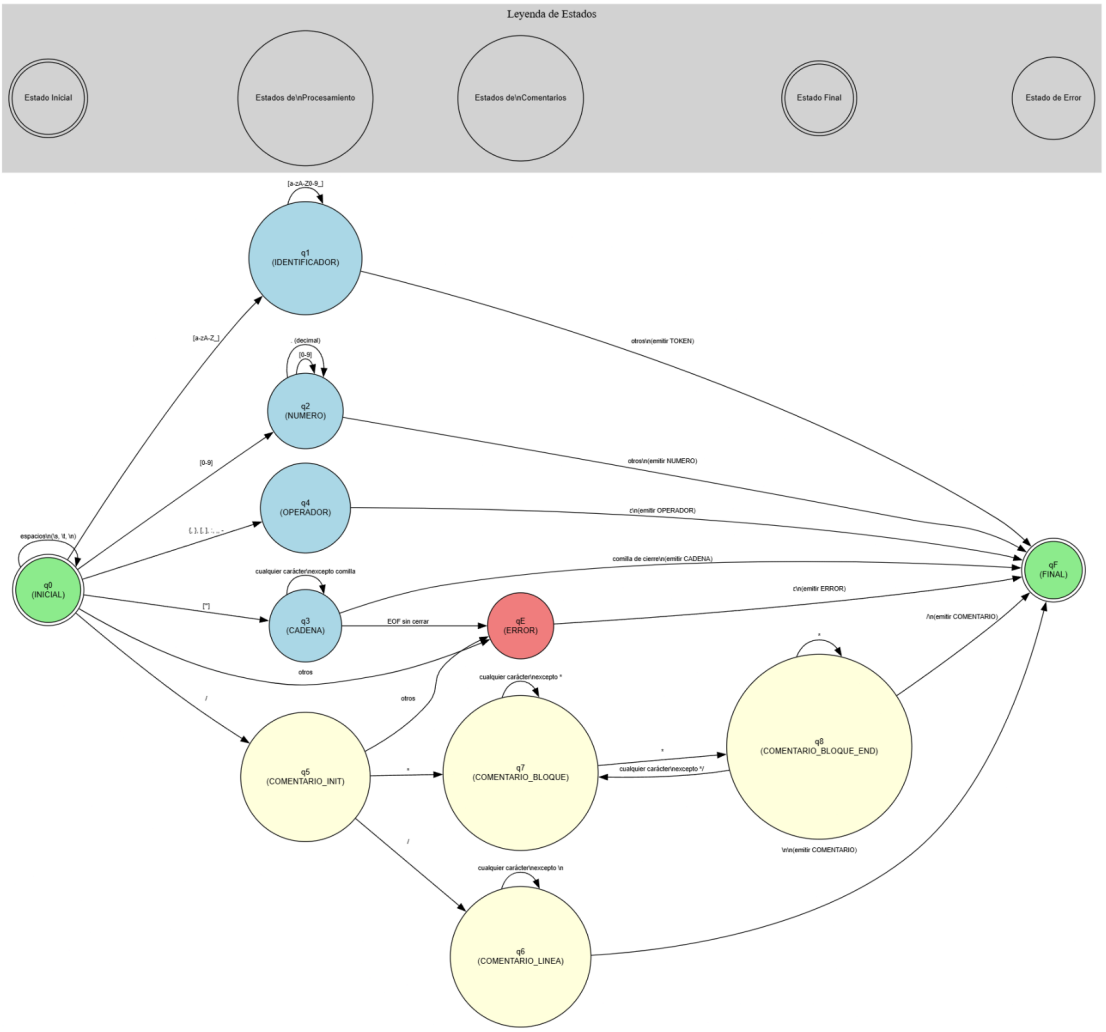

Iniciar Proyecto TourneyJS
¿Qué arquitectura de análisis usar?
Elegido (Modularidad)
Rechazado (Complejidad)
Lexer + Parser (Separación de responsabilidades)
Análisis en un solo paso (Más rápido pero complejo)
¿Qué tipo de parser implementar?
Elegido (Simplicidad)
Rechazado (Overkill)
Descenso Recursivo (Fácil mantenimiento)
Parser LR (Más potente)
¿Cómo manejar errores?
Elegido (Robustez)
Rechazado (Tiempo)
Panic Mode (Robusto)
Producciones de Error (Complejo)
¿Cómo representar el AST?
Elegido (Flexibilidad)
Rechazado (Limitaciones)
Nodos con hijos (Flexible)
Estructura lineal (Menos memoria)
¿Cómo generar reportes?
Elegido (Mantenibilidad)
Rechazado (Rigidez)
Templates HTML (Mantenible)
Generación directa (Más rápido)
¿Qué interfaces proveer?
Elegido (Requisitos)
Rechazado (Limitado)
Rechazado (Limitado)
Consola + Web (Flexibilidad)
Solo Web (Moderno)
Solo Consola (Simple)
¿Cómo visualizar brackets?
Elegido (Estándar)
Rechazado (Tiempo)
Rechazado (Limitado)
Graphviz DOT (Estándar)
SVG personalizado (Control total)
Tablas HTML (Compatible)
¿Qué compatibilidad requerir?
Elegido (Versatilidad)
Rechazado (Limitado)
Rechazado (Limitado)
Node.js + Navegador (Universal)
Solo Node.js (Servidor)
Solo Navegador (Cliente)
Sistema TourneyJS Completo

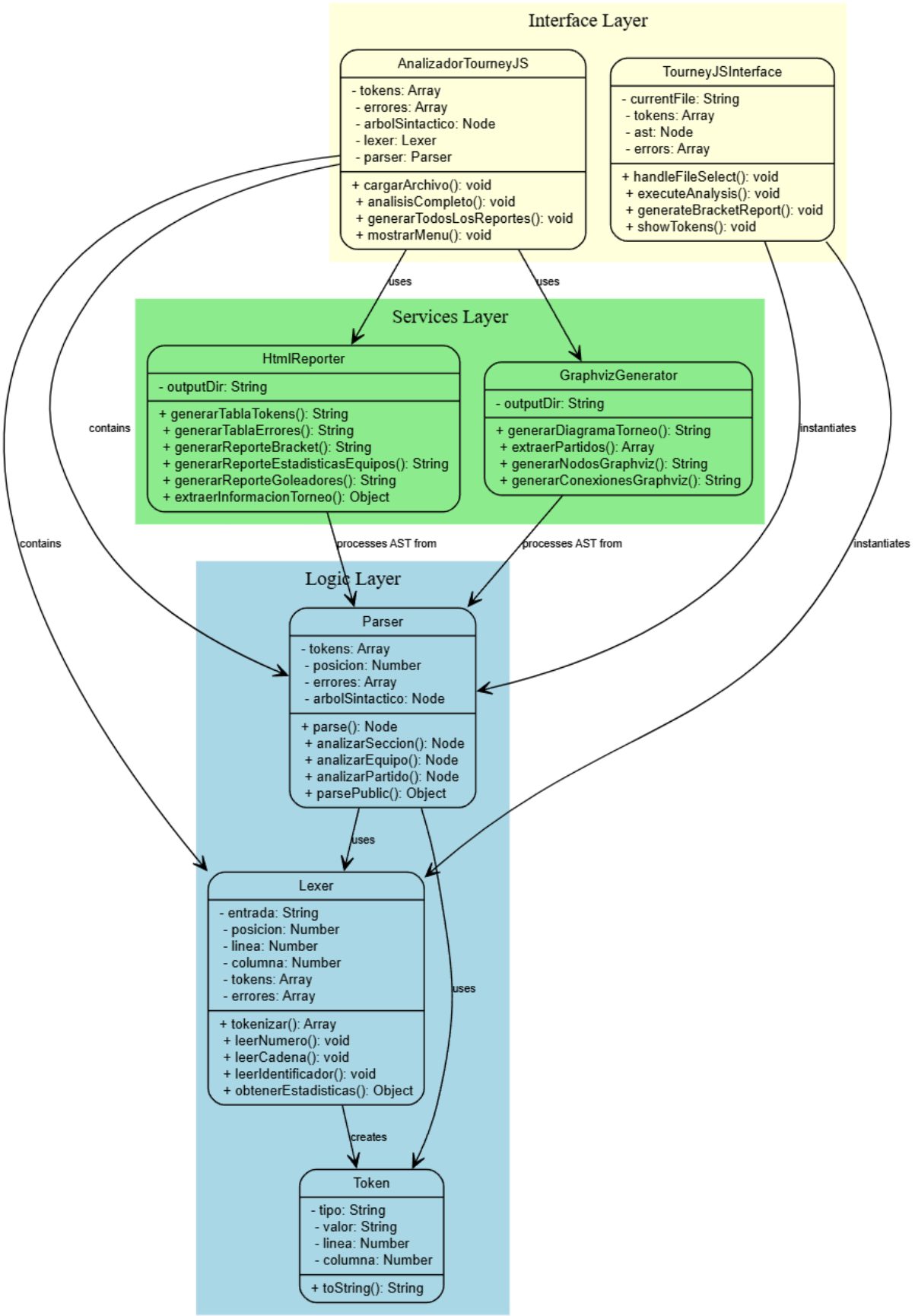
Algoritmo de Tokenización

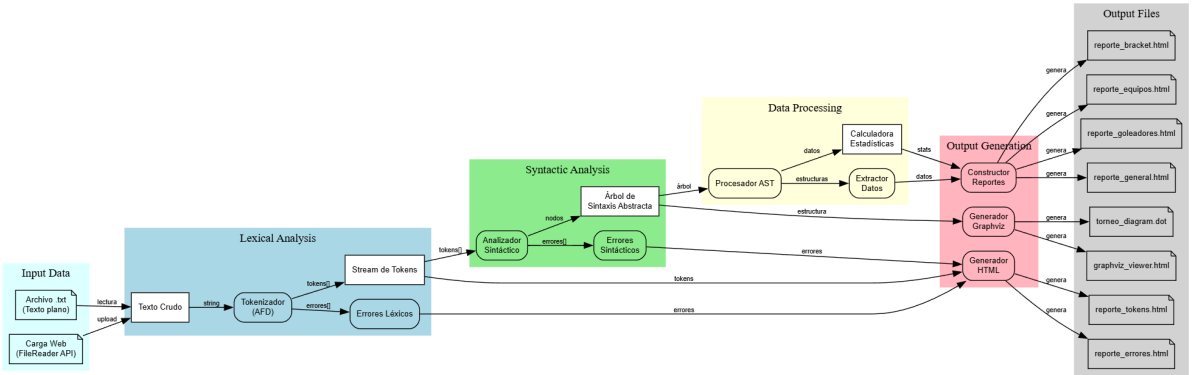
1. Leer carácter actual\l
2. Determinar transición según AFD\l
3. Cambiar al estado correspondiente\l
4. Acumular caracteres del token\l
5. Al llegar a estado final:\l
- Crear token con tipo y valor\l
- Agregar a lista de tokens\l
- Retornar a estado inicial\l
6. Si error: reportar y continuar\l
7. Repetir hasta EOF\l

Tokens Reconocidos

SECCION_PRINCIPAL: TORNEO, EQUIPOS, ELIMINACION\l
PALABRA_RESERVADA: equipo, jugador, partido, etc.\l
POSICION_JUGADOR: PORTERO, DEFENSA, etc.\l
IDENTIFICADOR: nombres definidos por usuario\l
CADENA: texto entre comillas\l
NUMERO: enteros y decimales\l
OPERADORES: {, }, [,], .. , -, vs\l
COMENTARIO: // y /* */\l
ERROR: caracteres no reconocidos\l
EOF: fin de archivo\l




Interface Layer
AnalizadorTourneyJS
- tokens: Array
- errores: Array
- arbolSintactico: Node
- lexer: Lexer
- parser: Parser
+ cargarArchivo(): void
+ analisisCompleto(): void
+ generarTodosLosReportes(): void
+ mostrarMenu(): void
TourneyJSInterface
- currentFile: String
- tokens: Array
- ast: Node
- errors: Array
+ handleFileSelect(): void
+ executeAnalysis(): void
+ generateBracketReport(): void
+ showTokens(): void
uses
uses
Services Layer
HtmlReporter
- outputDir: String
+ generarTablaTokens(): String
+ generarTablaErrores(): String
+ generarReporteBracket(): String
+ generarReporteEstadisticasEquipos(): String
+ generarReporteGoleadores(): String
+ extraerInformacionTorneo(): Object
GraphvizGenerator
- outputDir: String
+ generarDiagramaTorneo(): String
+ extraerPartidos(): Array
+ generarNodosGraphviz(): String
+ generarConexionesGraphviz(): String
contains
instantiates
contains
processes AST from
processes AST from
instantiates
Logic Layer
Parser
- tokens: Array
- posicion: Number
- errores: Array
- arbolSintactico: Node
+ parse(): Node
+ analizarSeccion(): Node
+ analizarEquipo(): Node
+ analizarPartido(): Node
+ parsePublic(): Object
uses
Lexer
- entrada: String
- posicion: Number
- linea: Number
- columna: Number
- tokens: Array
- errores: Array
+ tokenizar(): Array
+ leerNumero(): void
+ leerCadena(): void
+ leerIdentificador(): void
+ obtenerEstadisticas(): Object
uses
creates
Token
- tipo: String
- valor: String
- linea: Number
- columna: Number
+ toString(): String


Input Data
Archivo .txt (Texto plano)
Carga Web (FileReader API)
lectura
upload
Lexical Analysis
Texto Crudo
string
Tokenizador (AFD)
tokens[]
errores[]
Stream de Tokens
Errores Léxicos
Syntactic Analysis
tokens[]
Analizador Sintáctico
nodos
errores[]
Árbol de Sintaxis Abstracta
Errores Sintácticos
Data Processing
árbol
Procesador AST
datos
estructuras
Calculadora Estadísticas
Extractor Datos
stats
datos
estructura
errores
tokens
errores
Output Generation
Constructor Reportes
Generador Graphviz
Generador HTML
Output Files
genera
reporte_bracket.html
reporte_equipos.html
reporte_goleadores.html
reporte_general.html
torneo_diagram.dot
graphviz_viewer.html
reporte_tokens.html
reporte_errores.html