Manual Técnico - Analizador Léxico de Lenguaje Pokémon

1. Descripción General del Proyecto

Este proyecto es una aplicación web desarrollada en **React + TypeScript + Vite** que implementa un analizador léxico para un lenguaje de dominio específico (DSL) diseñado para definir equipos de Pokémon. La aplicación permite escribir código en un editor personalizado, analizarlo léxicamente, visualizar los tokens generados y mostrar el equipo Pokémon resultante con información obtenida de la PokeAPI.

2. Arquitectura y Estructura del Proyecto

2.1 Estructura de Directorios

```
├── src/
│   ├── api/                 # Configuración de APIs externas
│   ├── components/          # Componentes React reutilizables
│   ├── context/            # Contextos de React para manejo de estado
│   ├── views/              # Vistas principales de la aplicación
│   ├── assets/             # Recursos estáticos
│   ├── lexer.ts            # Implementación del analizador léxico
│   ├── highlightHelper.ts  # Utilidades para destacado de sintaxis
│   └── App.tsx             # Componente principal de la aplicación
├── public/                 # Archivos públicos estáticos
└── package.json            # Configuración del proyecto
```

2.2 Tecnologías Utilizadas

- **React 19.1.0**: Framework de JavaScript para interfaces de usuario

- **TypeScript 5.8.3**: Superset tipado de JavaScript
- **Vite 6.3.5**: Herramienta de construcción y servidor de desarrollo
- **Axios 1.9.0**: Cliente HTTP para comunicación con APIs
- **ESLint**: Herramienta de análisis estático de código

3. Componentes Principales

3.1 Analizador Léxico (lexer.ts)

Función Principal: `lexer(input: string): LexerResult`**Tokens Reconocidos**:

- `RESERVED`: Palabra reservada "Jugador"
- `IDENTIFIER`: Identificadores (nombres de variables/estadísticas)
- `STRING`: Cadenas de texto delimitadas por comillas dobles
- `NUMBER`: Números enteros
- `LBRACE/RBRACE`: Llaves { y }
- `LPAREN/RPAREN`: Paréntesis (y)
- `LBRACKET/RBRACKET`: Corchetes [y]
- `ASSIGN`: Operador de asignación `:=`
- `EQUAL`: Signo igual =
- `SEMICOLON`: Punto y coma `;`
- `COLON`: Dos puntos `:`

Manejo de Errores:

- `UNKNOWN`: Caracteres no reconocidos
- `UNCLOSED_STRING`: Cadenas sin cerrar

3.2 Contexto Global (AppContext.tsx)

Gestiona el estado global de la aplicación utilizando React Context:

```
interface AppContextType {
  editorText: string;          // Texto del editor
  tokens: Token[];             // Tokens analizados
  errors: Token[];             // Errores léxicos
  analyzed: boolean;           // Estado de análisis
  // ... métodos de manipulación
}
```

3.3 Vistas Principales

HomeView (views/HomeView.tsx)

- Editor de texto con destacado de sintaxis
- Botón de análisis léxico
- Tabla de tokens resultante

TeamView (views/TeamView.tsx)

- Parseo de tokens para extraer información del equipo
- Integración con PokeAPI
- Cálculo de IVs (Individual Values)
- Selección automática del mejor equipo
- Visualización en formato cartas de TCG

ErrorLogsView (views/ErrorLogsView.tsx)

- Reporte detallado de errores léxicos
- Tabla de errores con ubicación y descripción

4. Integración con APIs

4.1 PokeAPI (api/pokemonApi.ts)

Endpoint: `https://pokeapi.co/api/v2/pokemon/{name}`

Funcionalidad:

- Obtiene información detallada de Pokémon
- Maneja errores 404 para Pokémon no encontrados
- Retorna datos de sprites, estadísticas, tipos y movimientos

5. Algoritmos Clave

5.1 Análisis Léxico

Implementa un analizador léxico con las siguientes características:

- Reconocimiento de patrones mediante expresiones regulares
- Seguimiento de posición (fila, columna)
- Manejo de caracteres especiales y unicode (ñ, acentos)
- Detección y reporte de errores

5.2 Parseo de Equipo Pokémon

```
// Estructura esperada del lenguaje:
Jugador: "nombre"{
  "pokemon"[tipo] := (
    [salud]=valor;
    [ataque]=valor;
    [defensa]=valor;
  )
}
```

5.3 Selección de Mejor Equipo

- Cálculo de IVs: `((salud + ataque + defensa) / 45) * 100`
- Priorización por tipo único (máximo 6 Pokémon)
- Ordenamiento por IVs descendente

6. Características Técnicas

6.1 Destacado de Sintaxis

- Resaltado en tiempo real usando overlays CSS
- Sincronización de scroll entre textarea y elemento de resaltado
- Codificación HTML segura para prevenir XSS

6.2 Manejo de Archivos

- Carga de archivos `.pklfp`
- Guardado con extensión personalizada
- Validación de formato al cargar

6.3 Estado y Navegación

- Sistema de navegación por pestañas
- Estado persistente durante la sesión
- Limpieza selectiva del editor

7. Configuración de Desarrollo

7.1 Scripts Disponibles

```
{
  "dev": "vite",           // Servidor de desarrollo
  "build": "tsc -b && vite build",  // Construcción de producción
  "lint": "eslint .",      // Análisis de código
  "preview": "vite preview" // Vista previa de build
}
```

7.2 Configuración TypeScript

- Configuración modular con tsconfig.app.json y tsconfig.node.json
- Tipado estricto habilitado
- Soporte para importaciones ES modules