

Selvin Raúl Chuquiej Andrade

202405516

LFP B+

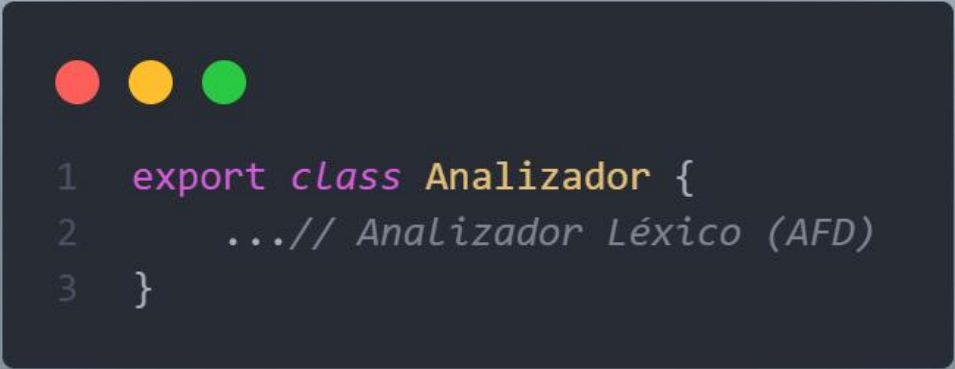
# MANUAL TECNICO

## Proyecto 1

El sistema está basado en una arquitectura modular implementada en Node.js con Express. Se divide en:

- Rutas (Routes): definen los endpoints disponibles en la API (lexicalAnalyze.routes.js).
- Controladores (Controllers): reciben las solicitudes HTTP, gestionan el flujo de análisis y devuelven los resultados (lexicalAnalyze.controller.js).
- Modelo de Análisis (Model): contiene el núcleo del analizador léxico y las funciones auxiliares para generación de estadísticas y detección de errores (analizador.js, token.js).
- Visualización: se apoya en Graphviz para representar gráficamente los brackets (graphviz.js).

#### Analizador.js:



```
1  export class Analizador {  
2      ...// Analizador Léxico (AFD)  
3  }
```

Esta clase Analizador implementa el analizador léxico para archivos de torneos deportivos. Recorre el texto carácter por carácter usando un autómata finito determinista (AFD), identifica tokens válidos (palabras reservadas, atributos, números, cadenas y símbolos), y registra errores léxicos. Al finalizar, genera la lista de tokens, errores, el bracket de partidos, estadísticas deportivas, goleadores y la información general del torneo.

- **generarBracket():**

```

1  function generarBracket(tokens) {
2      const fases = ['octavos', 'cuartos', 'semifinal', 'final'];
3      let faseActual = '';
4      const bracket = [];
5      let i = 0;
6
7      while (i < tokens.length) {
8          const token = tokens[i];
9
10         // Detecta la fase actual
11         if (token.tipo === 'ATRIBUTO' && fases.includes(token.lexema)) {
12             faseActual = token.lexema;
13         }
14
15         // Detecta un partido
16         if (token.tipo === 'RESERVADA' && token.lexema === 'partido') {
17             let partido = { fase: faseActual, partido: '', resultado: '', ganador: '' };
18
19             // Busca los equipos
20             let equipoA = tokens[i + 2]?.lexema || '';
21             let equipoB = tokens[i + 4]?.lexema || '';
22
23             // Busca el resultado
24             let resultado = '';
25             for (let j = i; j < tokens.length; j++) {
26                 if (tokens[j].tipo === 'RESERVADA' && tokens[j].lexema === 'resultado') {
27                     resultado = tokens[j + 2]?.lexema || '';
28                     break;
29                 }
30             }
31
32             const [golesA, golesB] = resultado.split('-').map(Number);
33             if (golesA > golesB) {
34                 partido.ganador = equipoA;
35             } else if (golesB > golesA) {
36                 partido.ganador = equipoB;
37             } else {
38                 partido.ganador = 'Empate';
39             }
40
41             partido.partido = `${equipoA}` + ATRIBUTOS[ATRIBUTOS.indexOf('vs')] + `${equipoB}`;
42             partido.resultado = resultado;
43
44             bracket.push(partido);
45         }
46         i++;
47     }
48     return bracket;
49 }

```

Esta función recorre la lista de tokens generados por el analizador léxico y construye el arreglo de partidos (bracket) del torneo. Detecta la fase actual (octavos, cuartos, semifinal, final) y, para cada partido, identifica los equipos participantes y el resultado. Determina el ganador comparando los goles de cada equipo y almacena toda la información relevante en un objeto. El resultado es un arreglo de objetos, cada uno representando un partido con su fase, equipos, resultado y ganador. Esto permite estructurar el torneo para generar el grafo y los reportes.

- **calcularEstadisticas():**

```

1  function calcularEstadisticas(brackets, tokens) {
2      const equipos = {};
3      const ordenFases = {
4          'octavos': 1,
5          'cuartos': 2,
6          'semifinal': 3,
7          'final': 4
8      };
9
10     for (let i = 0; i < tokens.length; i++) {
11         const token = tokens[i];
12         if (token.tipo === 'RESERVADA' && token.lexema === 'equipo') {
13             const nombreEquipo = tokens[i + 2]?.lexema || '';
14             if (nombreEquipo && !equipos[nombreEquipo]) {
15                 equipos[nombreEquipo] = {
16                     equipo: nombreEquipo,
17                     partidosJugados: 0,
18                     ganados: 0,
19                     perdidos: 0,
20                     golesAFavor: 0,
21                     golesEnContra: 0,
22                     diferencia: 0,
23                     faseAlcanzada: 'N/A'
24                 };
25             }
26         }
27     }
28
29     brackets.forEach(partido => {
30         const [equipoA, equipoB] = partido.partido.split(' vs ');
31         const [golesA, golesB] = partido.resultado.split('-').map(Number);
32
33         // Suma partidos jugados
34         equipos[equipoA].partidosJugados += 1;
35         equipos[equipoB].partidosJugados += 1;
36
37         // Suma goles
38         equipos[equipoA].golesAFavor += golesA;
39         equipos[equipoA].golesEnContra += golesB;
40         equipos[equipoB].golesAFavor += golesB;
41         equipos[equipoB].golesEnContra += golesA;
42
43         // Diferencia de goles
44         equipos[equipoA].diferencia = equipos[equipoA].golesAFavor - equipos[equipoA].golesEnContra;
45         equipos[equipoB].diferencia = equipos[equipoB].golesAFavor - equipos[equipoB].golesEnContra;
46
47         // Ganados y perdidos
48         if (golesA > golesB) {
49             equipos[equipoA].ganados += 1;
50             equipos[equipoB].perdidos += 1;
51         } else if (golesB > golesA) {
52             equipos[equipoB].ganados += 1;
53             equipos[equipoA].perdidos += 1;
54         }
55
56         // Fase alcanzada (puedes actualizar según la lógica de tu torneo)
57         equipos[equipoA].faseAlcanzada = partido.fase;
58         equipos[equipoB].faseAlcanzada = partido.fase;
59     });
60
61     const equiposArray = Object.values(equipos);
62     equiposArray.sort((a, b) => ordenFases[b.faseAlcanzada] - ordenFases[a.faseAlcanzada]);
63     return equiposArray;
64 }

```

Esta función calcula las estadísticas deportivas de cada equipo participante. Primero, identifica todos los equipos y crea un objeto para cada uno con sus datos iniciales. Luego, recorre el bracket de partidos y actualiza para cada equipo: partidos jugados,

ganados, perdidos, goles a favor, goles en contra, diferencia de goles y la última fase alcanzada. Finalmente, ordena los equipos según la fase alcanzada y devuelve un arreglo con las estadísticas completas, útil para reportes y rankings.

- Goleadores():

```
1  function goleadores(tokens) {
2      const jugadorEquipo = {}
3      let equipoActual = ''
4
5      for (let i = 0; i < tokens.length; i++) {
6          const token = tokens[i];
7          if (token.tipo === 'RESERVADA' && token.lexema === 'equipo') {
8              equipoActual = tokens[i + 2]?.lexema || '';
9          }
10         if (token.tipo === 'RESERVADA' && token.lexema === 'jugador') {
11             const jugador = tokens[i + 2]?.lexema || '';
12             if (jugador) {
13                 jugadorEquipo[jugador] = equipoActual;
14             }
15         }
16     }
17
18     const lista = {};
19     for (let i = 0; i < tokens.length; i++) {
20         const token = tokens[i];
21         if (token.tipo === 'RESERVADA' && token.lexema === 'goleador') {
22             const jugador = tokens[i + 2]?.lexema || '';
23             const equipo = jugadorEquipo[jugador] || '';
24             let minuto = '';
25             for (let j = i; j < tokens.length; j++) {
26                 if (tokens[j].tipo === 'ATRIBUTO' && tokens[j].lexema === 'minuto') {
27                     minuto = tokens[j + 2]?.lexema || '';
28                     break;
29                 }
30             }
31
32             if (!lista[jugador]) {
33                 lista[jugador] = { equipo, goles: 1, minutos: [minuto] };
34             } else {
35                 lista[jugador].goles += 1;
36                 lista[jugador].minutos.push(minuto);
37             }
38         }
39     }
40
41     const goleadoresArray = Object.entries(lista).map(([jugador, datos], idx) => ({
42         posicion: idx + 1,
43         jugador,
44         equipo: datos.equipo,
45         goles: datos.goles,
46         minutos: datos.minutos.join(', ')
47     })).sort((a, b) => b.goles - a.goles).map((g, idx) => ({ ...g, posicion: idx + 1 }));
48
49     return goleadoresArray;
50 }
```

Esta función identifica los jugadores que han anotado goles en el torneo. Relaciona cada jugador con su equipo y cuenta la cantidad de goles y los minutos en que fueron anotados. Recorre los tokens buscando los eventos de gol y suma la información en un objeto por jugador. Al final, genera un arreglo ordenado por cantidad de goles, mostrando el nombre, equipo, goles y minutos de cada goleador.

- **infoTorneo():**

```
1 function infoTorneo(tokens, estadisticas, goleadores) {
2   let nombreTorneo = '';
3   let sede = '';
4   let edades = [];
5
6   for (let i = 0; i < tokens.length; i++) {
7     const token = tokens[i];
8     if (token.tipo === 'ATRIBUTO' && token.lexema === 'nombre') {
9       nombreTorneo = tokens[i + 2].lexema || '';
10    }
11    if (token.tipo === 'ATRIBUTO' && token.lexema === 'sede') {
12      sede = tokens[i + 2].lexema || '';
13    }
14    if (token.tipo === 'ATRIBUTO' && token.lexema === 'edad') {
15      const edad = Number(tokens[i + 2].lexema);
16      if (!isNaN(edad) && edad > 0) {
17        edades.push(edad);
18      }
19    }
20  }
21
22  const totalEquipos = estadisticas.length;
23  const totalPartidos = estadisticas.reduce((sum, equipo) => sum + equipo.partidosJugados, 0) / 2;
24  const totalGoles = goleadores.reduce((sum, goleador) => sum + goleador.goles, 0);
25  const promedioGoles = totalPartidos ? (totalGoles / totalPartidos).toFixed(2) : 0;
26  const edadPromedio = edades.length > 0 ? (edades.reduce((a, b) => a + b, 0) / edades.length).toFixed(2) : 'N/A';
27  const faseActual = estadisticas[0].faseAlcanzada || 'N/A';
28
29  return [
30    { estadistica: 'Nombre del Torneo', valor: nombreTorneo },
31    { estadistica: 'Sede', valor: sede },
32    { estadistica: 'Total de Equipos', valor: totalEquipos },
33    { estadistica: 'Total de Partidos', valor: totalPartidos },
34    { estadistica: 'Total de Goles', valor: totalGoles },
35    { estadistica: 'Promedio de Goles por Partido', valor: promedioGoles },
36    { estadistica: 'Edad Promedio de Jugadores', valor: edadPromedio + ' años' },
37    { estadistica: 'Fase Actual', valor: faseActual }
38  ];
39 }
```

Esta función extrae y calcula información general del torneo. Obtiene el nombre del torneo, la sede, la cantidad total de equipos, partidos y goles, el promedio de goles por partido, la edad promedio de los jugadores y la fase actual del torneo. Toda esta información se presenta en un arreglo de objetos, cada uno con una estadística y su valor, para mostrar en reportes generales y en la interfaz.

**lexicalAnalyze.controller.js:**

```

1  'use strict'
2  import { Analizador } from '../model/analizador.js';
3  import { generarDot } from './graphviz.js';
4
5  export function analizarArchivo(req, res) {
6      const { contenido } = req.body;
7      if (!contenido) {
8          return res.status(400).json({ error: 'No se proporcionó contenido del archivo.' });
9      }
10
11     const analizador = new Analizador(contenido);
12     const resultado = analizador.analizar();
13
14     const nombreTorneo = resultado.informacionTorneo?.find(e => e.estadistica === 'Nombre del Torneo')?.valor || '';
15     const sede = resultado.informacionTorneo?.find(e => e.estadistica === 'Sede')?.valor || '';
16     const fases = resultado.bracket ? [...new Set(resultado.bracket.map(p => p.fase))] : [];
17     const dot = generarDot(nombreTorneo, sede, fases, resultado.bracket);
18
19
20     //console.log(resultado.errores);
21     //console.log(resultado.tokens);
22     return res.json({
23         tokens: resultado.tokens,
24         errores: resultado.errores,
25         bracket: resultado.bracket,
26         estadisticas: resultado.estadisticas,
27         goleadores: resultado.listaGoleadores,
28         torneos: resultado.informacionTorneo,
29         dot
30     });
31 };

```

Esta función `analizarArchivo` es el controlador principal del backend que procesa la petición del frontend. Recibe el contenido del archivo enviado por el usuario, crea una instancia del analizador léxico, ejecuta el análisis y obtiene los tokens, errores, el bracket, estadísticas, goleadores e información del torneo. Luego, genera el grafo DOT con los datos obtenidos y responde al frontend con todos los resultados en formato JSON para su visualización y reporte.

### graphviz.js:

```

1  export function generarDot(nombreTorneo = '', sede = '', fases = [], bracket = []) {
2      ...//Logica para
3  }

```

La función `generarDot` construye el grafo del torneo en formato DOT para Graphviz. Recibe el nombre del torneo, sede, fases y el bracket de partidos. Genera los nodos de cada equipo y partido, agrupados por fase (cuartos, semifinal, final), y asigna colores: verde para ganadores y rojo para perdedores. Crea las conexiones entre nodos según el avance de los equipos, usando líneas normales para ganadores y líneas punteadas

para perdedores. El resultado es un string DOT que representa visualmente el desarrollo y resultados del torneo.

### token.js:

```
1  'use strict';
2
3  export class Token {
4    constructor(lexema, tipo, fila, columna) {
5      this.lexema = lexema;
6      this.tipo = tipo;
7      this.fila = fila;
8      this.columna = columna;
9    }
10 }
11
12 export const RESERVADAS = [
13   "TORNEO",
14   "EQUIPOS",
15   "ELIMINACION",
16   "equipo",
17   "jugador",
18   "partido",
19   "resultado",
20   "goleador"
21 ];
22
23 export const ATRIBUTOS = [
24   "nombre",
25   "sede",
26   "equipos",
27   "posicion",
28   "numero",
29   "edad",
30   "octavos",
31   "cuartos",
32   "semifinal",
33   "final",
34   "vs",
35   "goleadores",
36   "minuto"
37 ];
38
39 export const SIMBOLOS = {
40   "{": "LLAVE_IZQUIERDA",
41   "}": "LLAVE_DERECHA",
42   "[": "CORCHETE_IZQUIERDO",
43   "]": "CORCHETE_DERECHO",
44   ",": "COMA",
45   ".": "DOS_PUNTOS",
46   ";": "PUNTO_Y_COMA",
47   "(": "PARENTESIS_IZQUIERDO",
48   ")": "PARENTESIS_DERECHO",
49   "-": "GUION"
50 };
51
52
```

El archivo define la clase Token, que representa cada elemento léxico identificado en el análisis, almacenando su valor, tipo y posición en el archivo. También contiene los arreglos RESERVADAS y ATRIBUTOS, que listan las palabras clave y atributos reconocidos por el analizador, y el objeto SIMBOLOS, que asocia cada símbolo especial con su tipo. Estos recursos permiten clasificar y validar los tokens durante el análisis léxico del archivo de entrada.



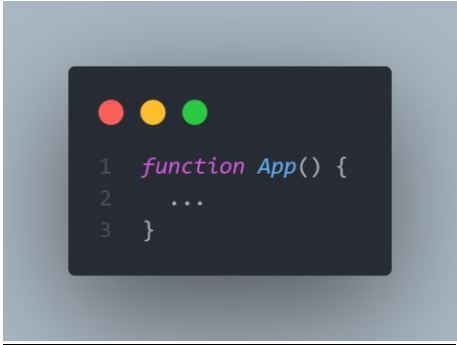
### lexicalAnalyze.routes.js:

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is written in JavaScript and defines an Express.js router for lexical analysis. It includes imports for 'express' and a controller function 'analizarArchivo', sets up a POST route for '/archivo', and exports the router as 'archivoRoutes'.

```
1 'use strict'
2
3 import express from 'express';
4 const router = express.Router();
5
6 import { analizarArchivo } from '../controller/lexicalAnalyze.controller.js';
7
8 router.post('/archivo', analizarArchivo);
9
10 export const archivoRoutes = router;
```

Este archivo define la ruta principal para el análisis léxico en el backend usando Express. Importa el controlador analizarArchivo y lo asocia a la ruta POST /archivo, permitiendo que el frontend envíe el contenido del archivo para ser procesado y recibir los resultados en formato JSON. La exportación archivoRoutes permite integrar esta ruta en el servidor principal.

### App.jsx:

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code shows the beginning of a function component named 'App' in React, with a function signature 'function App()' and an opening curly brace, followed by an ellipsis indicating more code.

```
1 function App() {
2   ...
3 }
```

Este componente App es el núcleo del frontend en React. Permite al usuario cargar un archivo .txt, visualizar su contenido, analizar los tokens y errores, generar reportes y mostrar el grafo del torneo. Gestiona el estado de la aplicación, realiza peticiones al backend para procesar el archivo y actualiza la interfaz según la vista seleccionada (contenido, tokens, errores, reporte o gráfico). Integra componentes para mostrar tablas de tokens, errores, reportes y el grafo generado con Graphviz.

### Reportes.jsx:

```

1  const Reportes = ({ brackets, estadísticas, goleadores, torneos }) => (
2    ...//Tabla Bracket Eliminación, Tabla Estadísticas por Equipo, Tabla Goleadores, Tabla Información general torneo
3  );

```

El componente Reportes muestra los reportes principales del torneo en el frontend. Recibe los datos procesados (brackets, estadísticas, goleadores, torneos) y los presenta en tablas separadas: partidos y ganadores por fase, estadísticas por equipo, ranking de goleadores y estadísticas generales del torneo. Permite al usuario visualizar de forma clara y ordenada toda la información relevante del torneo analizado.

### TablaErrores.jsx:

```

1  import React from 'react';
2  import '../css/Tablas.css';
3
4  const TablaErrores = ({ errores }) => (
5    <div className="contenedor-tabla">
6      <h2>Errores Lexicos: {errores.length}</h2>
7      <table className="tabla-tokens">
8        <thead>
9          <tr>
10             <th>No</th>
11             <th>Lexema</th>
12             <th>Tipo de Error</th>
13             <th>Descripción</th>
14             <th>Fila</th>
15             <th>Columna</th>
16          </tr>
17        </thead>
18        <tbody>
19          {errores.map((error, idx) => (
20            <tr key={idx}>
21              <td>{idx + 1}</td>
22              <td>{error.lexema}</td>
23              <td>{error.tipo}</td>
24              <td>{error.descripcion}</td>
25              <td>{error.fila}</td>
26              <td>{error.columna}</td>
27            </tr>
28          ))}
29        </tbody>
30      </table>
31    </div>
32  );

```

El componente TablaErrores muestra una tabla con los errores léxicos detectados durante el análisis del archivo. Recibe el arreglo de errores como propiedad y los

presenta con su número, lexema, tipo de error, descripción, fila y columna. Permite al usuario identificar fácilmente los problemas de sintaxis en el archivo cargado.

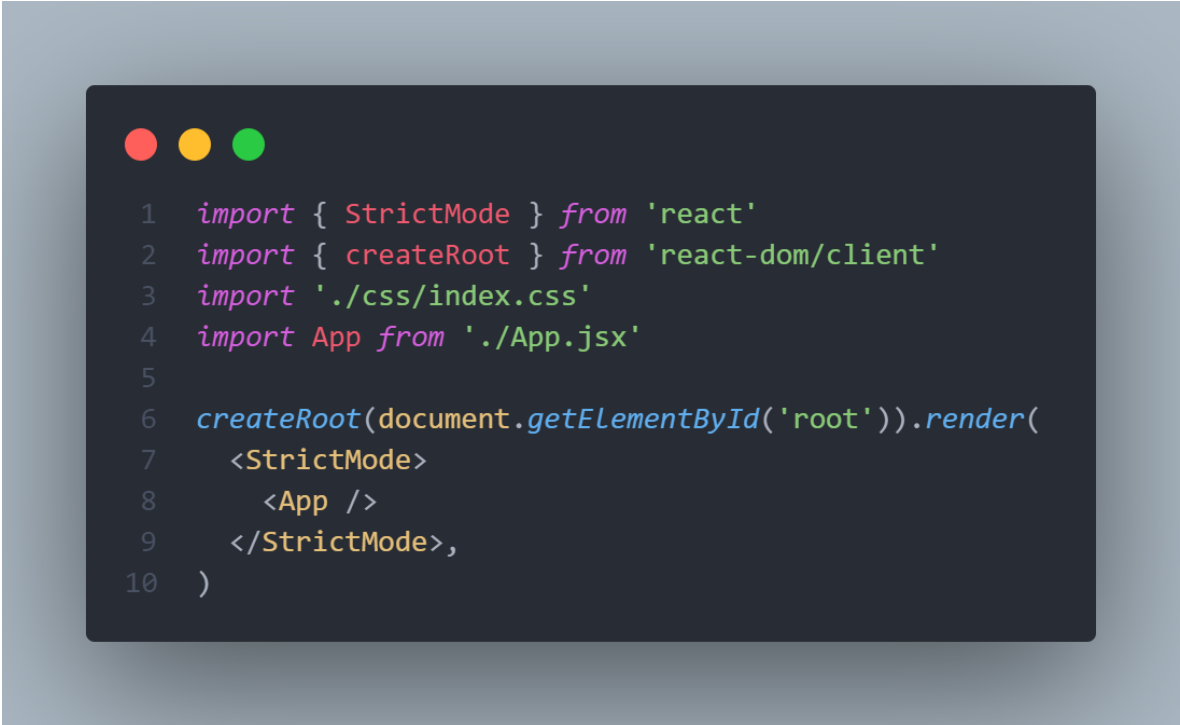
### TablaTokens.jsx:

```
1  import React from 'react';
2  import '../css/Tablas.css';
3
4  const TablaTokens = ({ tokens }) => (
5    <div className="contenedor-tabla">
6      <h2>Tokens Extraídos: {tokens.length}</h2>
7      <table className="tabla-errores">
8        <thead>
9          <tr>
10             <th>No</th>
11             <th>Lexema</th>
12             <th>Tipo</th>
13             <th>Fila</th>
14             <th>Columna</th>
15          </tr>
16        </thead>
17        <tbody>
18          {tokens.map((token, idx) => (
19            <tr key={idx}>
20              <td>{idx + 1}</td>
21              <td>{token.lexema}</td>
22              <td>{token.tipo}</td>
23              <td>{token.fila}</td>
24              <td>{token.columna}</td>
25            </tr>
26          ))}
27        </tbody>
28      </table>
29    </div >
30  );
31
32  export default TablaTokens;
```

El componente TablaTokens muestra una tabla con todos los tokens extraídos del archivo analizado. Recibe el arreglo de tokens como propiedad y los presenta con su

número, lexema, tipo, fila y columna. Permite al usuario visualizar de forma clara y ordenada los elementos léxicos reconocidos por el analizador.

### Main.jsx:



```
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import './css/index.css'
4  import App from './App.jsx'
5
6  createRoot(document.getElementById('root')).render(
7    <StrictMode>
8      <App />
9    </StrictMode>,
10  )
```

El archivo main.jsx es el punto de entrada del frontend en React. Importa el componente principal App y lo renderiza dentro del modo estricto (StrictMode) en el elemento raíz del HTML. También importa los estilos globales. Este archivo inicializa la aplicación y asegura que el componente App se muestre correctamente en el navegador.

