

MANUAL TECNICO

PRACTICA 1 POKEAYUDA - MANUAL TECNICO



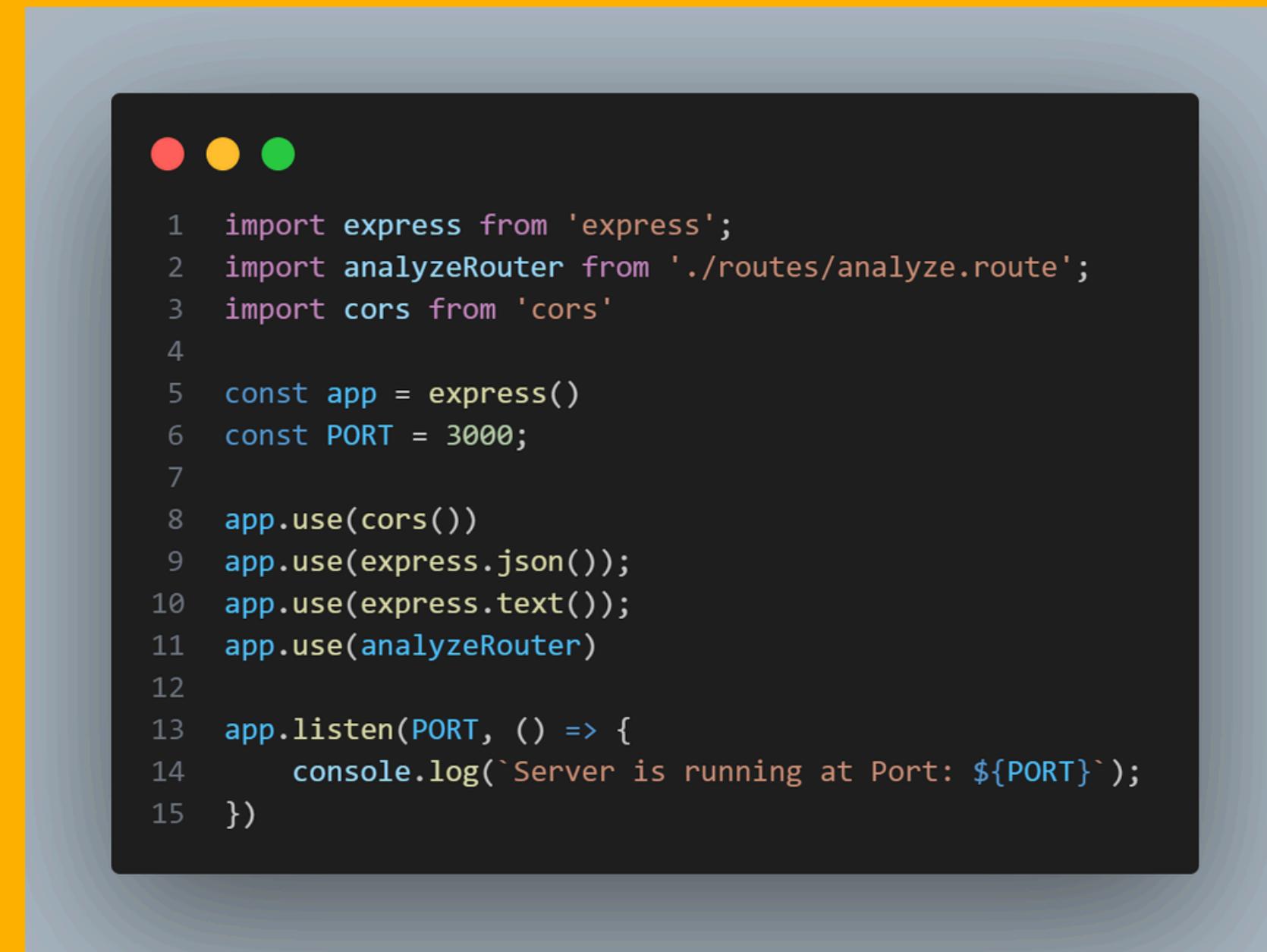


INTRODUCCION

Este es un manual tecnico que muestra como entender el codigo de programacion de la pagina web de PokeAyuda, mostrando sus funcionalidades y la logica que tiene en su backend.

BACKEND

index: El archivo principal del sistema de backend esta dado por el index que contiene lo siguiente en codigo; Hago uso de “cors” y tambien de express para el manejo mas facil de mi proyecto, express para los servicios y cors para poder usar esta api de backend en un entorno global, y uso express con json para el manejo de respuestas, tambien instancio el uso de las rutas que estan en otro archivo, por ultimo levanto el servidor en el puerto 3000.



```
● ● ●

1 import express from 'express';
2 import analyzeRouter from './routes/analyze.route';
3 import cors from 'cors'
4
5 const app = express()
6 const PORT = 3000;
7
8 app.use(cors())
9 app.use(express.json());
10 app.use(express.text());
11 app.use(analyzeRouter)
12
13 app.listen(PORT, () => {
14   console.log(`Server is running at Port: ${PORT}`);
15 })
```

BACKEND

Token: En la parte del archivo de Token hago un constructor con sus variables locales privadas que me ayudaran a crear una estructura sobre que es un token y tambien un enum Type que me va servir para llevar un orden sobre que tipos de Token puedo tener en mi programa, tambien tengo dos Getters que me sirven en otra parte del codigo para poder acceder a los lexemas y los tipos de token.

```
1 enum Type {  
2     UNKNOW,  
3     PAR_OPEN, // (  
4     PAR_CLOSE, // )  
5     SEMICOLON, // ;  
6     EQUAL, // =  
7     RESERVERD_WORD,  
8     NUMBER, // number  
9     STRING, // string  
10    COLON, // :  
11    BRACK_OPEN, // [  
12    BRACK_CLOSE, // ]  
13    BRACE_OPEN, // {  
14    BRACE_CLOSE, // }  
15    QUOTE, // "  
16    ASSIGN  
17 }
```

```
1 class Token {  
2  
3     private row: number;  
4     private column: number;  
5     private lexeme: string;  
6     private typeToken: Type;  
7  
8     constructor(typeToken: Type, lexeme: string, row: number, column: number) {  
9         this.typeToken = typeToken;  
10        this.lexeme = lexeme;  
11        this.row = row;  
12        this.column = column;  
13    }  
14  
15    getLexeme(): string {  
16        return this.lexeme;  
17    }  
18  
19    getType(): Type {  
20        return this.typeToken;  
21    }  
22 }
```

BACKEND

LexicalAnalyzer: Esta parte del código es de las más importantes, aca es donde se maneja toda la lógica del AFD y como funciona el analizador léxico, lo que hago aca es primero instanciar variables privadas y un constructor que va consistir en dos arreglos, un estado para moverse, un carácter auxiliar y uno de fila y uno de columna para poder llevar un orden y un conocimiento. Por otra parte estan las funciones de addCharacter que agrega los caracteres actuales y los auxiliares para llevar orden, la función Clean que limpia caracteres y estados, la función de addToken que agrega los tokens a la lista de tokens y addError que agrega los errores a la lista de errores y por ultimo un getter que devuelve la lista de errores.

```
1 class LexicalAnalyzer {  
2  
3     private row: number;  
4     private column: number;  
5     private auxChar: string;  
6     private state: number;  
7     private tokenList: Token[];  
8     private errorList: Token[];  
9  
10    constructor() {  
11        this.row = 1;  
12        this.column = 1;  
13        this.auxChar = '';  
14        this.state = 0;  
15        this.tokenList = [];  
16        this.errorList = [];  
17    }
```

```
1  
2     private addCharacter(char: string) {  
3         this.auxChar += char;  
4         this.column++;  
5     }  
6  
7     private clean() {  
8         this.state = 0;  
9         this.auxChar = '';  
10    }  
11  
12    private addToken(type: Type, lexeme: string, row: number, column: number) {  
13        this.tokenList.push(new Token(type, lexeme, row, column));  
14    }  
15  
16    private addError(type: Type, lexeme: string, row: number, column: number) {  
17        this.errorList.push(new Token(type, lexeme, row, column));  
18    }  
19  
20    getErrorList() {  
21        return this.errorList;  
22    }  
23 }
```

BACKEND

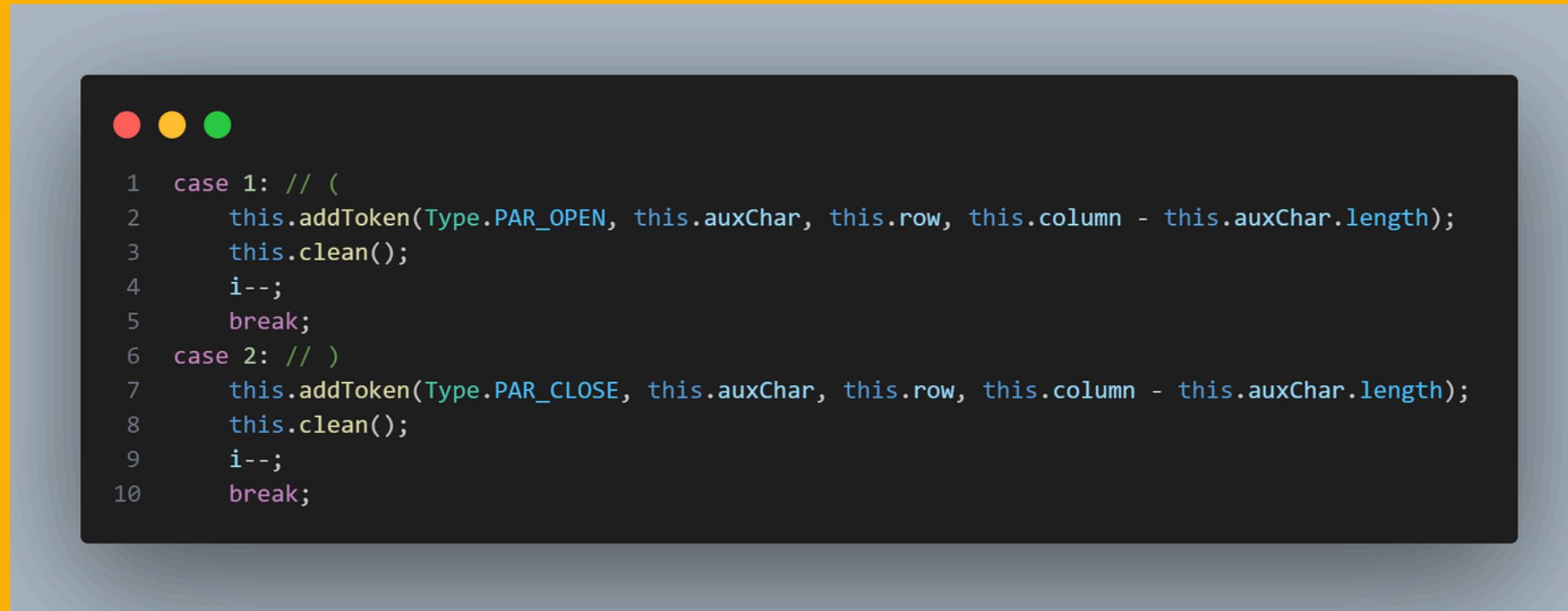
LexicalAnalyzer: Esta parte es muy tecnica, dicha funcion primero inicia con una instancia de un caracter y continua con un bucle for que va recorrer un condicional switch con determinados casos donde en el primer caso enumerado con el 0 va tener dentro otro switch el cual va tener todos lo inicios de palabras reservadas o transiciones principales de salen del estado cero en un AFD entonces dependiendo de en que parte del caso 0 inicia reconociendo lo mandara a ese estado, hay otros estados del estado 0 que ayudan por ejemplo para el espaciado y saltos de linea que son necesarios para saber las columnas y filas.

```
● ○ ●  
1 scanner(input: string) {  
2     let char: string;  
3     for (let i: number = 0; i < input.length; i++) {  
4         char = input[i];  
5  
6         switch (this.state) {  
7             case 0:  
8                 switch (char) {  
9                     case '(':  
10                        this.state = 1;  
11                        this.addCharacter(char);  
12                        break;  
13  
14                     case ')':  
15                        this.state = 2;  
16                        this.addCharacter(char);  
17                        break;  
18  
19  
20  
21  
22  
23  
24  
25  
26
```

```
● ○ ●  
1 case 'f':  
2     this.state = 58;  
3     this.column++;  
4     this.addCharacter(char);  
5     break;  
6  
7 case 'n':  
8     this.state = 63;  
9     this.column++;  
10    this.addCharacter(char);  
11    break;  
12  
13 case '\n':  
14     this.row++;  
15     this.column = 1;  
16     break;  
17 case '\r':  
18     if (input[i + 1] === '\n') {  
19         i++;  
20     }  
21     this.row++;  
22     this.column = 1;  
23     break;  
24 case '\t':  
25     this.column += 4;  
26     break;
```

BACKEND

Al momento de dirigirse a uno de los estados fuera del principal que es el cero dependera abran 3 casos para recibir valores y almacenarlos, por ejemplo en el caso 1 al identificar en el estado 0 que es un parentesis abierto ya lo reconoce como Token entonces ira al estado 1 y agregara el token de ‘(‘ a la lista de Tokens luego hace funcion del Clean y regresa las lienas usadas de i para el siguiente caracter a reconocer



The screenshot shows a terminal window with a dark theme. At the top left, there are three colored dots: red, yellow, and green. Below them is a block of Java code:

```
1 case 1: // (
2     this.addToken(Type.PAR_OPEN, this.auxChar, this.row, this.column - this.auxChar.length);
3     this.clean();
4     i--;
5     break;
6 case 2: // )
7     this.addToken(Type.PAR_CLOSE, this.auxChar, this.row, this.column - this.auxChar.length);
8     this.clean();
9     i--;
10    break;
```

BACKEND

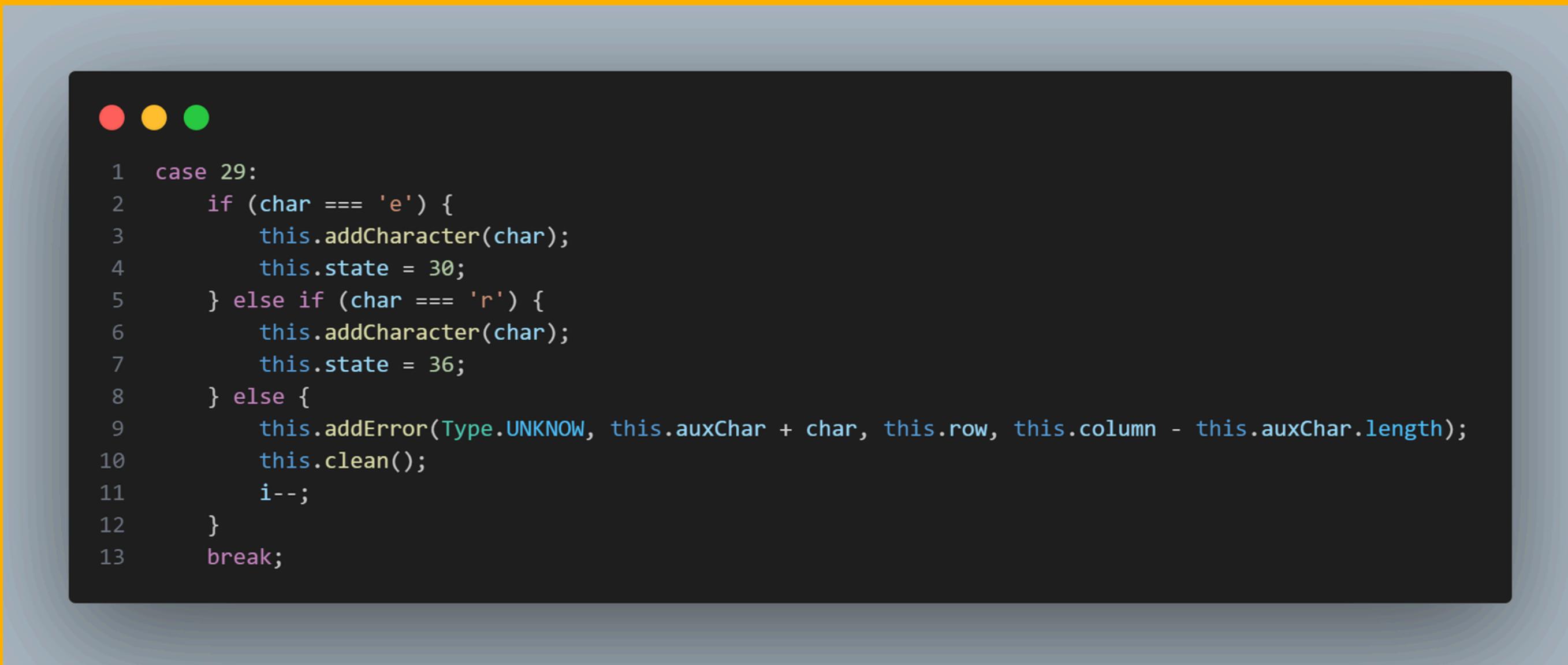
La segunda opcion es que mi Token tenga mas de una opcion, en donde se valida que si despues de ‘:’ sigue un ‘=’ entonces va tomar los dos como un unico Token, pero si despues de ‘:’ no continua un ‘=’ entonces argegara solo ‘:’ a la lista de Tokens.

```
1 case 5: // :
2     if (char == '=') {
3         this.state = 69;
4         this.addCharacter(char);
5         continue;
6     }
7     this.addToken(Type.COLON, this.auxChar, this.row, this.column - this.auxChar.length);
8     this.clean();
9     i--;
10    break;
```

```
1 case 69:
2     this.addToken(Type.ASSIGN, this.auxChar, this.row, this.column - this.auxChar.length);
3     this.clean();
4     i--;
5     break;
```

BACKEND

Tambien esta la opcion donde para evitar la ambiguedad de las letras de palabras reservadas que salen desde el estado 0 entonces se usa condicional if y else if para saber en que direccion se va dirigir, si lo siguiente es una ‘ e ‘ entonces va tomar ruta hacia ese estado, y si no entonces verifica si lo que sigue es una ‘ r ‘ entonces ira a la ruta del estado 36 y si no es ninguna entonces lo agregara a la lista de errores. Y asi con todas las opciones para el analizador de lexema.



```
1 case 29:
2     if (char === 'e') {
3         this.addCharacter(char);
4         this.state = 30;
5     } else if (char === 'r') {
6         this.addCharacter(char);
7         this.state = 36;
8     } else {
9         this.addError(Type.UNKNOW, this.auxChar + char, this.row, this.column - this.auxChar.length);
10        this.clean();
11        i--;
12    }
13 break;
```

BACKEND

ProcessToken: En este archivo se crea una interface de los pokemon y una interface de jugadores para tener una estructura de cada uno y manejarlos de mejor manera. luego tenemos una funcion que es la principal para recolectar jugadores y sus pokemons donde hacemos una llamada a una lista de tipo jugadores luego entramos en un while que recorre la cantidad de tokens y luego los recorre y cuando encunetra un token igual a Jugador entonces lo agrega a la lista de jugadores y toma su nombre para instanciarlo luego crea el objeto de jugador con dos listas una de sus pokemons y otra con los top de sus pokemons.

```
● ● ●  
1 interface Pokemon {  
2     nombre: string;  
3     tipo: string;  
4     salud: number;  
5     ataque: number;  
6     defensa: number;  
7     ivs: number;  
8 }  
9  
10 interface Jugador {  
11     nombre: string;  
12     pokemons: Pokemon[];  
13     topPokemons: Pokemon[];  
14 }
```

```
● ● ●  
1 export const processTokens = (tokens: Token[]): Jugador[] => {  
2     const jugadores: Jugador[] = [];  
3     let i = 0;  
4     while (i < tokens.length) {  
5         const token = tokens[i];  
6         if (token.getLexeme() === 'Jugador') {  
7             const jugadorNombreToken = tokens[i + 2];  
8             const jugadorNombre = jugadorNombreToken.getLexeme().replace(/\//g, '');  
9             const jugador: Jugador = {  
10                 nombre: jugadorNombre,  
11                 pokemons: [],  
12                 topPokemons: []  
13             };  
14         }  
15     }  
16     return jugadores;  
17 }
```

BACKEND

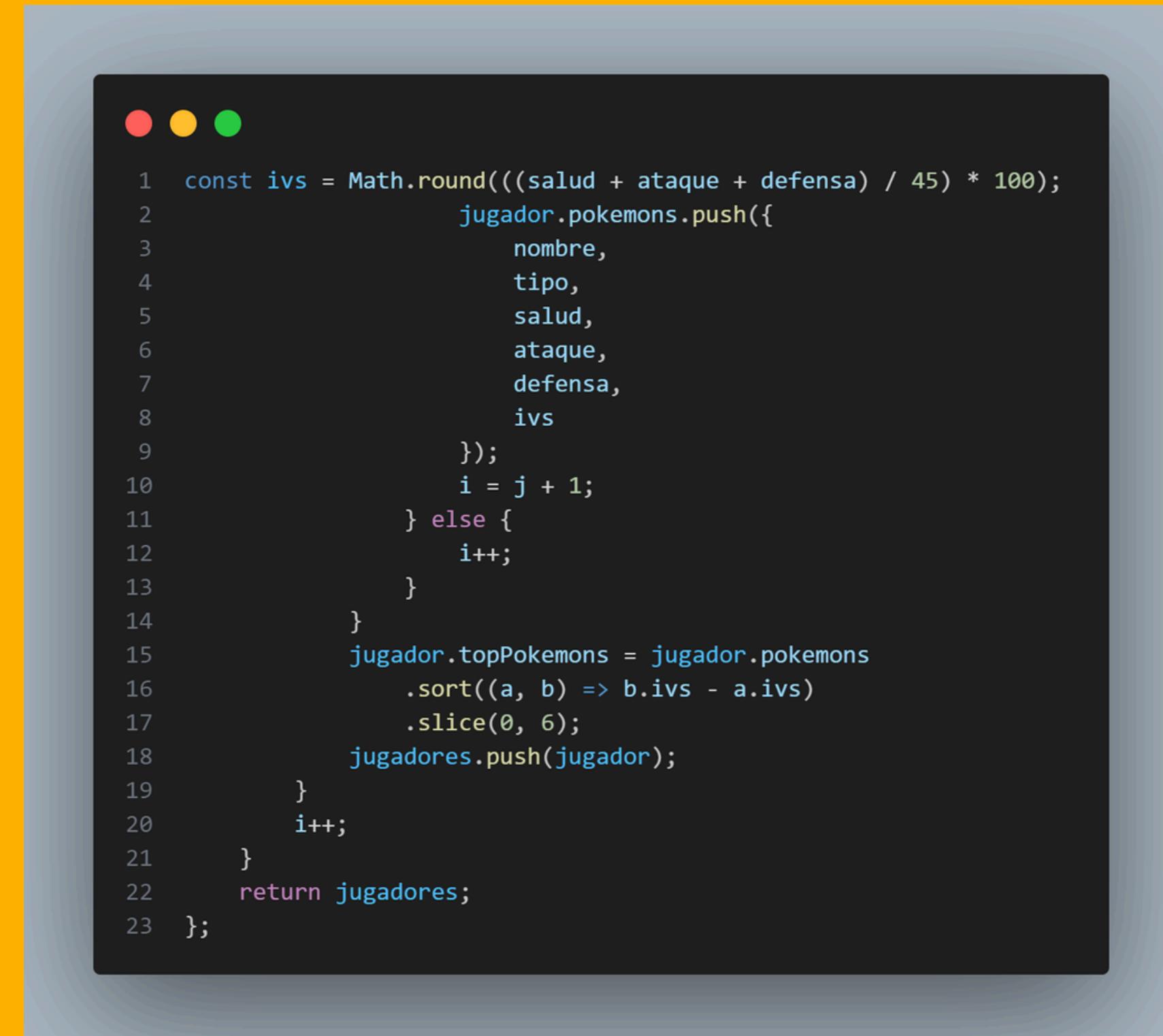
Luego continua con otro while donde busca cuando cierra un bloque de jugadores y hace un if donde busca si el tipo de token es igual a un string si es asi entonces toma ese token y lo mete en una variable que sera el nombre del pokemon, luego hace lo mismo con tipo, despues se instancia salud, ataque y defensa para evitar errores, busca el bloque de stats y obtiene los tokens que no sean de las tres variables y luego los obtiene los valores de las tres variables y las instancia.



```
1 while (i < tokens.length && tokens[i].getLexeme() !== '}') {
2     if (tokens[i].getType() === Type.STRING) {
3         const nombre = tokens[i].getLexeme().replace(/\//g, '');
4         const tipo = tokens[i + 2]?.getLexeme().replace(/\[\]/g, '') ?? 'desconocido';
5         let salud = 0, ataque = 0, defensa = 0;
6         let j = i;
7         while (j < tokens.length && tokens[j].getLexeme() !== '(') j++;
8         j++;
9         while (j < tokens.length && tokens[j].getLexeme() !== ')') {
10            if (
11                tokens[j]?.getLexeme() === '[' &&
12                tokens[j + 1] && typeof tokens[j + 1].getLexeme() === 'string' &&
13                tokens[j + 2]?.getLexeme() === ']' &&
14                tokens[j + 3]?.getLexeme() === '=' &&
15                tokens[j + 4]?.getType() === Type.NUMBER
16            ) {
17                const key = tokens[j + 1].getLexeme().toLowerCase();
18                const value = parseInt(tokens[j + 4].getLexeme());
19                if (key === 'salud') salud = value;
20                else if (key === 'ataque') ataque = value;
21                else if (key === 'defensa') defensa = value;
22                j += 6;
23            } else {
24                j++;
25            }
26        }
27    }
28}
```

BACKEND

Por ultimo se calcula los ivs para identificar a los mejores pokemons y los agrega a la lista de los pokemon de cada jugador, despues los ordena segun los ivs y los agrega a la lista de top pokemons y por ultimo retorna a los jugadores.

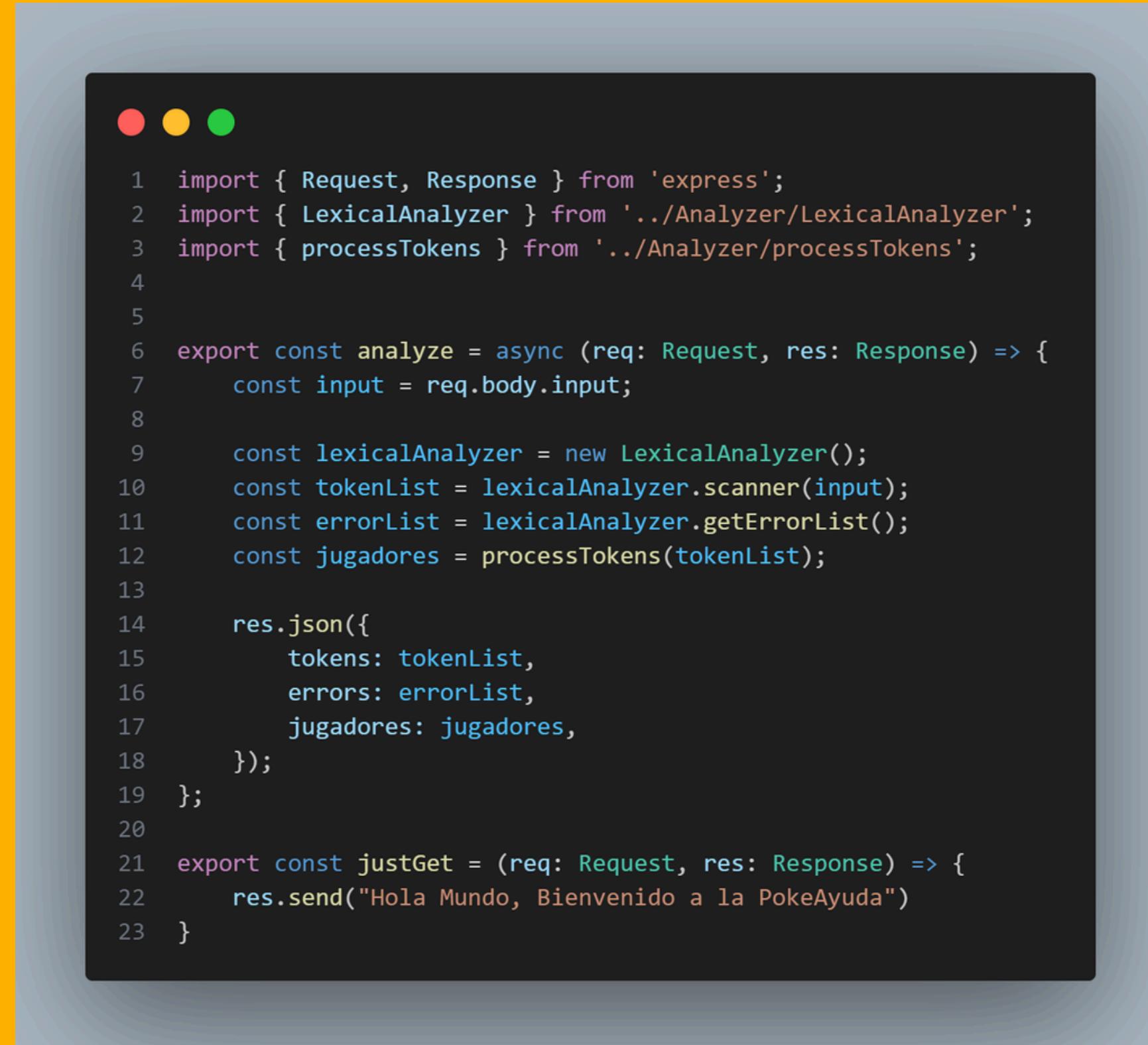


```
● ● ●

1 const ivs = Math.round(((salud + ataque + defensa) / 45) * 100);
2         jugador.pokemons.push({
3             nombre,
4             tipo,
5             salud,
6             ataque,
7             defensa,
8             ivs
9         });
10        i = j + 1;
11    } else {
12        i++;
13    }
14}
15jugador.topPokemons = jugador.pokemons
16    .sort((a, b) => b.ivs - a.ivs)
17    .slice(0, 6);
18jugadores.push(jugador);
19}
20i++;
21}
22return jugadores;
23};
```

BACKEND

Controlador: ahora lo que se hace aca es llamar a las funciones de LExicalAnalyzer y processTokens y luego se llaman a cada una de las funcinoes para recibir una respuesta luego de que se manden y reciban los tokens.



```
● ● ●

1 import { Request, Response } from 'express';
2 import { LexicalAnalyzer } from '../Analyzer/LexicalAnalyzer';
3 import { processTokens } from '../Analyzer/processTokens';
4
5
6 export const analyze = async (req: Request, res: Response) => {
7     const input = req.body.input;
8
9     const lexicalAnalyzer = new LexicalAnalyzer();
10    const tokenList = lexicalAnalyzer.scanner(input);
11    const errorList = lexicalAnalyzer.getErrorList();
12    const jugadores = processTokens(tokenList);
13
14    res.json({
15        tokens: tokenList,
16        errors: errorList,
17        jugadores: jugadores,
18    });
19}
20
21 export const justGet = (req: Request, res: Response) => {
22     res.send("Hola Mundo, Bienvenido a la PokeAyuda")
23 }
```

BACKEND

Por ultimo en las rutas llamo a la funcion del controlador y le establezco una ruta en la cual se pueda usar en este caso '/analyze' y llamo a la funcion para que al entrar en esa ruta se hagan esas funciones.

```
● ● ●  
1 import {Router} from 'express';  
2 import {analyze, justGet} from '../controllers/analyze.controller';  
3  
4 const analyzeRouter = Router();  
5  
6 analyzeRouter.get('/', justGet)  
7 analyzeRouter.post('/analyze', analyze);  
8  
9 export default analyzeRouter;
```

FRONTEND

main: En esta parte solo se estructura como va estar utilizandoce la verificacion de rutas en la web, para que se pueda usar React de la mejor manera.



A terminal window with three colored dots at the top (red, yellow, green). The window displays the following React code:

```
1 import ReactDOM from 'react-dom/client'
2 import App from './App.jsx'
3 import { BrowserRouter } from 'react-router-dom'
4 import { EditorProvider, ErrorProvider } from './context/ErrorContext.jsx'
5
6 ReactDOM.createRoot(document.getElementById('root')).render(
7   <BrowserRouter>
8     <ErrorProvider>
9       <EditorProvider>
10         <App />
11       </EditorProvider>
12     </ErrorProvider>
13   </BrowserRouter>
14 )
```

FRONTEND

App: Aca se crean las distintas rutas que se utilizan en el proyecto web.

```
● ● ●  
1 import { Route, Router, Routes } from 'react-router-dom'  
2 import { Inicial } from './Components/Inicial'  
3 import { Manuales } from './Components/Manuales'  
4 import { Reporte } from './Components/Reporte'  
5  
6 function App() {  
7  
8   return (  
9     <Routes>  
10      <Route path='/' element = {<Inicial />}></Route>  
11      <Route path='/Manuales' element = {<Manuales />}></Route>  
12      <Route path='/Reporte' element = {<Reporte />}></Route>  
13    </Routes>  
14  )  
15}  
16  
17 export default App
```

FRONTEND

NavBar: Aca se crea el Navbar que tiene la pagina web con sus rutas, donde la que mas tiene funcion es la de archivos que sirve para cargar un archivo al editor de texto.



The screenshot shows a code editor window with a dark theme. At the top left, there are three small colored circles (red, yellow, green). The code itself is a React component named `PokeNavbar`. It imports React, useState, Link, and index.css. It uses useState to manage the responsiveness of the nav bar. The component returns a `div` with the id `myTopnav`, which contains a `ul` with `nav-pills` class. The `ul` contains several `a` tags: one active (Poke Ayuda), one linking to `/Home`, one linking to `/Reporte`, and one for `Archivos` which triggers a file selection dialog. There is also a link to `/Manuales` and a toggle button represented by a `fa fa-bars` icon.

```
1 import React, { useState } from 'react'
2 import { Link } from 'react-router-dom';
3 import './index.css'
4
5 export const PokeNavbar = ({ onFileClick }) => {
6   const [isResponsive, setIsResponsive] = useState(false);
7
8   const toggleNav = () => {
9     setIsResponsive(!isResponsive);
10 };
11
12 return (
13   <div className={`topnav${isResponsive ? " responsive" : ""}`} id="myTopnav">
14     <ul className="nav nav-pills">
15       <a className="a-style active">Poke Ayuda</a>
16       <a className='a-style' href='/>Home</a>
17       <Link to="/Reporte" className="a-style">Reporte Error</Link>
18       <a className='a-style' href="/" onClick={(e) => {
19         e.preventDefault();
20         onFileClick();
21       }}>Archivos</a>
22       <a className='a-style' href="/Manuales">Manuales</a>
23       <a className="icon" onClick={toggleNav}>
24         <i className="fa fa-bars icono" />
25       </a>
26     </ul>
27   </div>
28 )
29 }
30 }
```

FRONTEND

Api: Se importa la libreria de axios para manejar las peticiones al Backend, luego se crea la funcion que va llamar al controlador del backend donde se manda la ruta y los datos necesarios que seria el texto donde van los tokens y retorna los datos pero ya verificados.



```
1 import axios from 'axios'
2
3
4 export const pokeData = async(data) => {
5     try {
6         const response = await axios.post('http://localhost:3000/analyze', { input: data });
7         return response.data;
8     } catch (error) {
9         console.error("Error al analizar:", error);
10        return null;
11    }
12}
```

FRONTEND

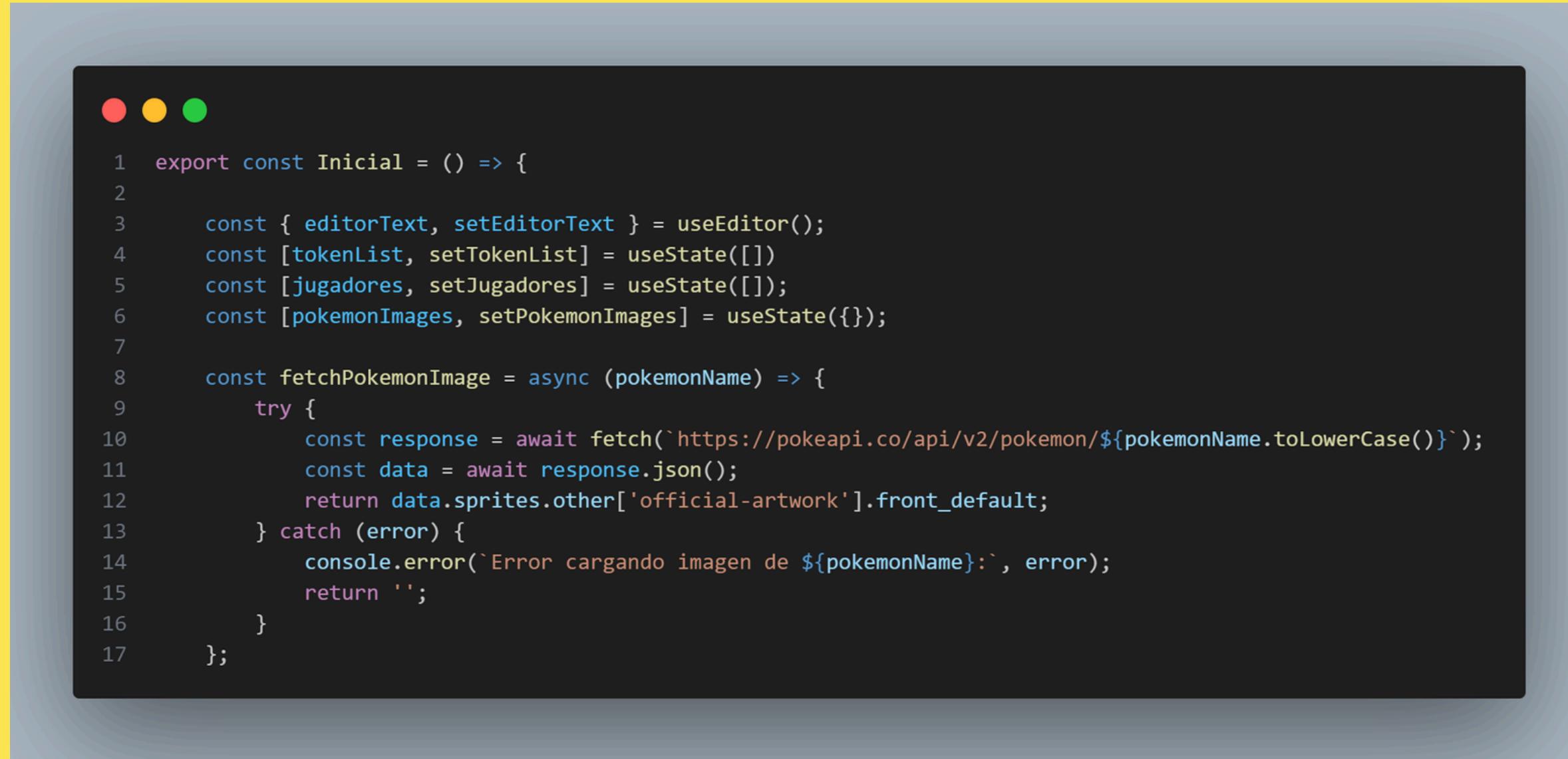
```
● ○ ●  
1 export const Reporte = () => {  
2   const { errors } = useError();  
3   return (  
4     <>  
5       <PokeNavbar />  
6       <div className="container mt-4">  
7         <h2>Reporte de Errores</h2>  
8         {errors.length === 0 ? (<p>No se encontraron errores.</p>) : (  
9           <table className="table">  
10             <thead>  
11               <tr>  
12                 <th>No.</th>  
13                 <th>Fila</th>  
14                 <th>Columna</th>  
15                 <th>Lexema</th>  
16                 <th>Token</th>  
17               </tr>  
18             </thead>  
19             <tbody>  
20               {errors.map((error, index) => (  
21                 <tr key={index}>  
22                   <td>{index + 1}</td>  
23                   <td>{error.row}</td>  
24                   <td>{error.column}</td>  
25                   <td>{error.lexeme}</td>  
26                   <td>{error.typeToken}</td>  
27                 </tr>  
28               ))}  
29             </tbody>  
30           </table>  
31         )}  
32       </div>  
33     </>  
34   );  
35 };
```

Reporte: En esta parte del reporte primero se llama al NavBar para que aparezca en la pagina luego se empieza a hacer un mapeo de los errores si es que encuentra, si no entonces mostrara un texto que dice que no hay errores, despues se hace una tabla en la cual empieza a mostrar todos los errores que puedan haber.

FRONTEND

Inicial: Aca primero usamos estados de variables para las distintas cosas que tenemos en el codigo, editorText maneja el editor de texto y sus funciones, TokenList es para los tokens que aparecen en la tabla, jugadores maneja lo relacionado con los distintos jugadores que se pueden cargar, y luego pokemonImages que maneja las funciones de las imagenes llamadas de un api.

La primera funcion sirve para llamar al api de pokemons que obtiene los datos de las imagenes de los pokemons basado en sus nombres, lo cual recibe como parametro, extrae la data en base al nombre del pokemon y devuelve la imagen.



```
1 export const Inicial = () => {
2
3     const { editorText, setEditorText } = useEditor();
4     const [tokenList, setTokenList] = useState([]);
5     const [jugadores, setJugadores] = useState([]);
6     const [pokemonImages, setPokemonImages] = useState({});
7
8     const fetchPokemonImage = async (pokemonName) => {
9         try {
10             const response = await fetch(`https://pokeapi.co/api/v2/pokemon/${pokemonName.toLowerCase()}`);
11             const data = await response.json();
12             return data.sprites.other['official-artwork'].front_default;
13         } catch (error) {
14             console.error(`Error cargando imagen de ${pokemonName}:`, error);
15             return '';
16         }
17     };
}
```

FRONTEND

setErrors obtiene los errores si es que los obtiene, la funcion refreshPage refresca la pagina, handleFieldLoad carga un archivo al programa y lo mete al textArea primero leyendolo y transcribiendo lo que encuentre, por ultimo handleFileClick abre el navegador de archivos que esta invisible.

```
● ● ●  
1 const { setErrors } = useError();  
2 const navigate = useNavigate();  
3  
4 const refreshPage = () => {  
5   window.location.reload();  
6 };  
7  
8 const handleFieldLoad = (event) => {  
9   const file = event.target.files[0];  
10  if (!file) return;  
11  const reader = new FileReader();  
12  reader.onload = (e) => {  
13    const content = e.target.result;  
14    console.log("Archivo cargado:", content);  
15    setEditorText(content);  
16  };  
17  reader.onerror = (e) => {  
18    console.error("Error al leer el archivo:", e);  
19  };  
20  reader.readAsText(file);  
21};  
22  
23  
24 const handleFileClick = () => {  
25   document.getElementById("fileInput").click();  
26};
```

FRONTEND

Aca es donde se obtiene todos los datos del textarea y se manda al api que se comunica con el backend, si al recibir la respuesta es positiva entonces metera partes de la respuesta en los estados ya instanciados, de lo contrario si encuentra que hay errores en los textos mostrara una alerta que dice que se encontraron errores, y al darle en OK lo mandara a los reportes.



```
1 const handlePokeData = async () => {
2     const result = await pokeData(editorText);
3     if (result) {
4         const errores = result.errors || [];
5         setErrors(errores);
6         if (errores.length > 0) {
7             setTokenList([]);
8             setJugadores([]);
9             Swal.fire({
10                 icon: 'error',
11                 title: 'Errores detectados',
12                 text: 'Se detectaron errores. Puedes verlos en la sección de Reporte.',
13                 confirmButtonText: 'OK'
14             }).then(() => {
15                 navigate('/Reporte');
16             });
17         } else {
18             setTokenList(result.tokens || []);
19             setJugadores(result.jugadores || []);
20         }
21     }
22};
```

FRONTEND

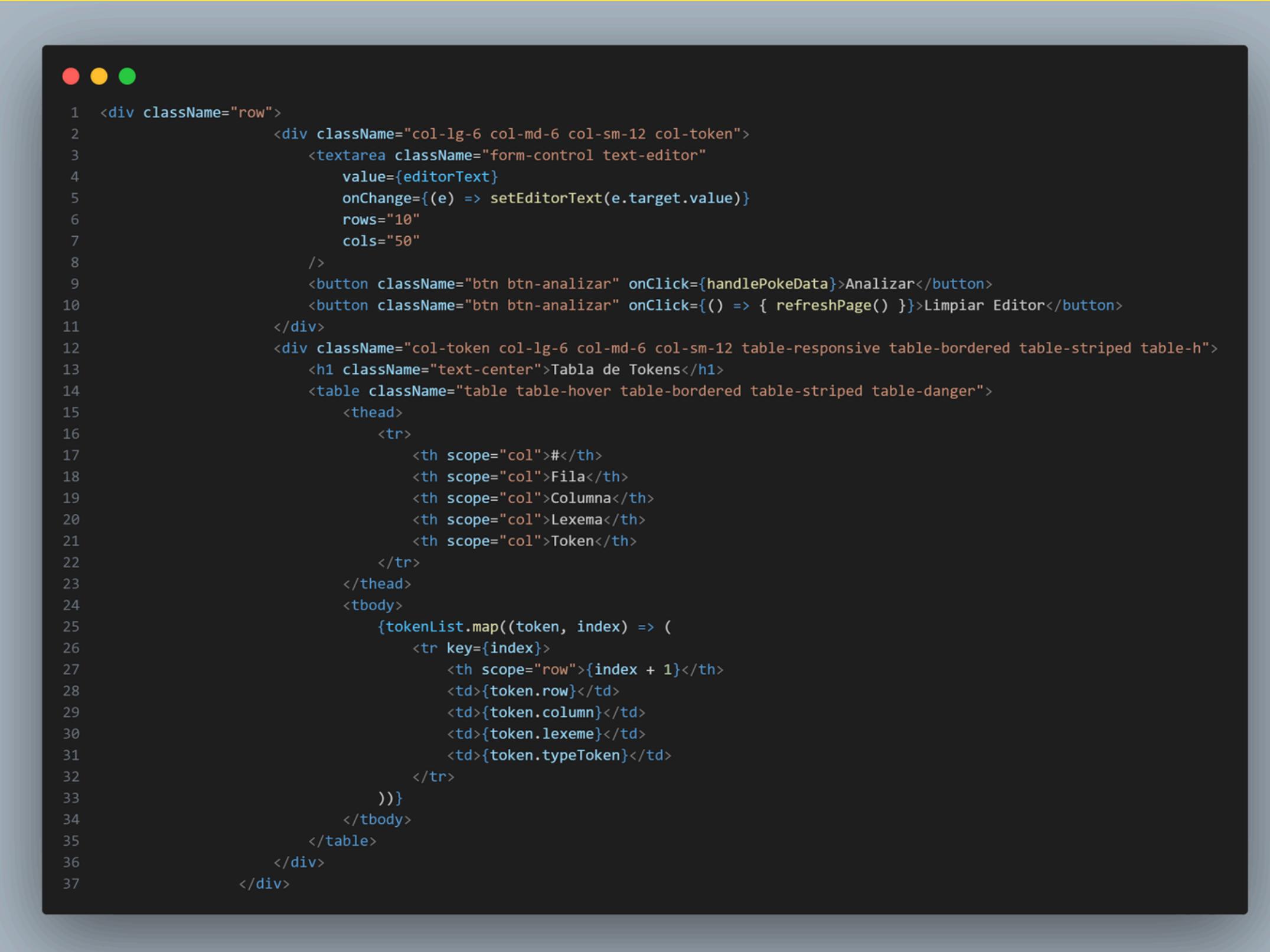
useEffect lo que hace es que carga las imagenes de pokemon y entonces cada que la pagina detecte que tiene algun jugador entonces va activar la funcion de cargar imagenes de pokemons.



The screenshot shows a code editor window with a dark theme. At the top left, there are three colored circular icons: red, yellow, and green. The code itself is written in a light-colored font on a dark background. It uses the `useEffect` hook to perform an asynchronous operation. The code is as follows:

```
1  useEffect(() => {
2      const loadPokemonImages = async () => {
3          const imageMap = {};
4          for (const jugador of jugadores) {
5              for (const pokemon of jugador.topPokemons) {
6                  const nombre = pokemon.nombre.toLowerCase();
7                  if (!imageMap[nombre]) {
8                      const imageUrl = await fetchPokemonImage(nombre);
9                      imageMap[nombre] = imageUrl;
10                 }
11             }
12         }
13         setPokemonImages(imageMap);
14     };
15     if (jugadores.length > 0) {
16         loadPokemonImages();
17     }
18 }, [jugadores]);
```

FRONTEND



```
1 <div className="row">
2     <div className="col-lg-6 col-md-6 col-sm-12 col-token">
3         <textarea className="form-control text-editor"
4             value={editorText}
5             onChange={(e) => setEditorText(e.target.value)}
6             rows="10"
7             cols="50"
8         />
9         <button className="btn btn-analizar" onClick={handlePokeData}>Analizar</button>
10        <button className="btn btn-analizar" onClick={() => { refreshPage() }}>Limpiar Editor</button>
11    </div>
12    <div className="col-token col-lg-6 col-md-6 col-sm-12 table-responsive table-bordered table-striped table-h">
13        <h1 className="text-center">Tabla de Tokens</h1>
14        <table className="table table-hover table-bordered table-striped table-danger">
15            <thead>
16                <tr>
17                    <th scope="col">#</th>
18                    <th scope="col">Fila</th>
19                    <th scope="col">Columna</th>
20                    <th scope="col">Lexema</th>
21                    <th scope="col">Token</th>
22                </tr>
23            </thead>
24            <tbody>
25                {tokenList.map((token, index) => (
26                    <tr key={index}>
27                        <th scope="row">{index + 1}</th>
28                        <td>{token.row}</td>
29                        <td>{token.column}</td>
30                        <td>{token.lexeme}</td>
31                        <td>{token.typeToken}</td>
32                    </tr>
33                )))
34            </tbody>
35        </table>
36    </div>
37</div>
```

En la parte del return donde se muestra lo visual primero empezamos con la parte del textArea y de los botones con sus respectivas funciones que son de analizar y de limpiar editor, despues tambien tiene la parte de la tabla de tokens en donde mapea las respuestas y muestra todos los tokens.

FRONTEND

```
1  <div className="row mt-4">
2    <h1 className="text-center">Jugadores</h1>
3    {jugadores.map((jugador, index) => (
4      <div className="col-12 mb-4" key={index}>
5        <div className="poke-card shadow-sm p-3">
6          <h4 className="text-center mb-3">{jugador.nombre}</h4>
7          <div className="row">
8            {jugador.topPokemons.map((pokemon, pIndex) => (
9              <div className="col-6 mb-3" key={pIndex}>
10                <div className="border rounded p-2 bg-light text-center">
11                  <img
12                    src={pokemonImages[pokemon.nombre.toLowerCase()]}
13                    alt={pokemon.nombre}
14                    style={{ width: '100px', height: '100px' }}
15                    className="mb-2"
16                  />
17                  <div>
18                    <strong>{pokemon.nombre}</strong> ({pokemon.tipo})<br />
19                    Salud: {pokemon.salud}<br />
20                    Ataque: {pokemon.ataque}<br />
21                    Defensa: {pokemon.defensa}<br />
22                    IVs: {pokemon.ivs}%
23                  </div>
24                </div>
25              </div>
26            )));
27            {jugador.topPokemons.length === 0 && (
28              <div className="col-12">
29                <p className="text-muted">Este jugador no tiene pokémon válidos.</p>
30              </div>
31            )}
32          </div>
33        </div>
34      </div>
35    )));
36  </div>
```

En esta parte del código es donde se muestran los jugadores, se toma el estado de los jugadores donde se guardaron las respuestas y empieza a mostrar por bloques cada uno de los jugadores y sus pokemons mostrando las imágenes y sus estadísticas tambien valida si el jugador no tiene pokemons y lo muestra con un texto.

FRONTEND

Por ultimo se pone un input donde se va abrir el navegador de archivos pero se pone como invisible para que aparezca únicamente cuando se presione el botón de archivos que está en el NavBar.

```
1 <input  
2   type="file"  
3   id="fileInput"  
4   accept=".pk1fp"  
5   style={{ display: 'none' }}  
6   onChange={handleFileLoad}  
7 />
```

CONCLUSIÓN

- Esas son todas las funciones que tiene la pagina web diseñada para el usuario y su uso de ayuda para elegir sus pokemon mas poderosos.
- Muestra cada una de las funciones tanto en Backend como en Frontend y como usarlas y por si se quiere modificar el codigo en algun momento.

Nombre: Alberto Moisés Gerardo Lémus Alvarado

Carne: 202400999

