1. Servidor Backend

```
import express from 'express';
import fileUpload from 'express-fileupload';
import path from 'path';
import { getPokemonInfo } from './pokenapi';
import { calcularIV, seleccionarMejores, PokemonConIV } from './ivCalculator';
import { analyzeText, AnalysisResult } from './lexer';

interface PokemonData {
  nombre: string;
  salud: number;
  ataque: number;
  defensa: number;
}

const app = express();
const PORT = 3001;

app.use(express.static(path.join(__dirname, 'public')));
app.use(fileUpload());
```

```
// Ruta raíz
Tabnine | Edit | Test | Explain | Document
app.get('/', async (_req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

// Ruta para subir archivo
Tabnine | Edit | Test | Explain | Document
app.post('/upload', async (req, res) => {    Refactor this function
  try {
    const file = req.files?.['archivo'] as fileUpload.UploadedFile;
    if (!file || Array.isArray(file)) {    Empty block statement.


    }

    const content = file.data.toString('utf-8');
    console.log('Contenido recibido:', content);

    // Usar el nuevo analizador con detección de errores
    const analysis: AnalysisResult = analyzeText(content);
    console.log('Tokens generados:', analysis.tokens);
    console.log('Errores encontrados:', analysis.errors);
```

```
const pokemonesExtraidos: PokemonData[] = [];

// Solo procesar si no hay errores críticos
if (analysis.errors.length === 0) {
  // Extraer datos desde los tokens
  for (let i = 0; i < analysis.tokens.length; i++) {
    if (
      analysis.tokens[i].lexema === 'Pokemon' &&
      analysis.tokens[i + 2]?.tipo === 'Cadena de Texto' &&
      analysis.tokens[i + 4]?.lexema === 'Datos'
    ) {
      const nombre = analysis.tokens[i + 2].lexema.replace(/"/g, '');
      const salud = parseInt(analysis.tokens[i + 6]?.lexema || '0');
      const ataque = parseInt(analysis.tokens[i + 8]?.lexema || '0');
      const defensa = parseInt(analysis.tokens[i + 10]?.lexema || '0');

      pokemonesExtraidos.push({ nombre, salud, ataque, defensa });
    }
  }
```

```
console.log('Pokémon extraídos:', pokemonesExtraidos);

// Llamar a la PokeAPI y calcular IV solo si hay pokémon extraídos
const datosConIV: PokemonConIV[] = [];

for (const p of pokemonesExtraidos) {
  console.log(`Buscando información para: ${p.nombre}`);
  const data = await getPokemonInfo(p.nombre);
  if (data) {
    const iv = calcularIV(p.salud, p.ataque, p.defensa);
    datosConIV.push({ ...data, ...p, iv });
    console.log(`Pokémon procesado: ${p.nombre}, IV: ${iv}`);
  } else {
    console.log(`No se pudo obtener información para: ${p.nombre}`);
  }
}

console.log('Datos con IV:', datosConIV);

// Seleccionar mejores 6 sin repetir tipo
const seleccionados = seleccionarMejores(datosConIV);
```

```
    // Enviar respuesta final
    res.json({
      tokens: analysis.tokens,
      errors: analysis.errors,
      seleccionados,
      debug: {
        pokemonesExtraidos,
        totalProcesados: datosConIV.length,
        erroresEncontrados: analysis.errors.length
      }
    });

  } else {
    // Si hay errores, solo devolver el análisis léxico
    res.json({
      tokens: analysis.tokens,
      errors: analysis.errors,
      seleccionados: [],
      debug: {
        pokemonesExtraidos: [],
        totalProcesados: 0,
        erroresEncontrados: analysis.errors.length,
```

```
  } catch (error) {
    console.error('Error procesando archivo:', error);
    res.status(500).json({
      error: 'Error interno del servidor',
      message: error instanceof Error ? error.message : 'Error desconocido',
      tokens: [],
      errors: [],
      seleccionados: []
    });
  }
);

abnine | Edit | Test | Explain | Document
pp.listen(PORT, () => {
  console.log(`Servidor corriendo en http://localhost:${PORT}`);
);
```

2. Lexer

```typescript
export interface Token {
  lexema: string;
  tipo: string;
  fila: number;
  columna: number;
}

export interface ErrorToken {
  lexema: string;
  tipo: string;
  descripcion: string;
  fila: number;
  columna: number;
}

export interface AnalysisResult {
  tokens: Token[];
  errors: ErrorToken[];
}
```

```typescript
Tabnine | Edit | Test | Fix | Explain | Document
export function analyzeText(input: string): AnalysisResult {
  const tokens: Token[] = [];
  const errors: ErrorToken[] = [];
  const lines = input.split('\n');

  for (let fila = 0; fila < lines.length; fila++) {
    const line = lines[fila];
    let columna = 1;
    let i = 0;

    while (i < line.length) {
      const char = line[i];

      // Saltar espacios en blanco
      if (char === ' ' || char === '\t') {
        i++;
        columna++;
        continue;
      }
```

```
// Cadenas de texto
if (char === '"' || char === "'") {
  const startCol = columna;
  let lexema = char;
  i++;
  columna++;

  while (i < line.length && line[i] !== char) {
    lexema += line[i];
    i++;
    columna++;
  }

  if (i < line.length) {
    lexema += char;
    i++;
    columna++;
    tokens.push({ lexema, tipo: 'Cadena de Texto', fila: fila + 1, columna: startCol });
  } else {
    errors.push({
      lexema,
```

```
// Símbolos compuestos
if (char === ':' && i + 1 < line.length && line[i + 1] === '=') {
  tokens.push({ lexema: ':=', tipo: 'Operador de Asignación', fila: fila + 1, columna });
  i += 2;
  columna += 2;
  continue;
}

// Símbolos simples
if (simbolosValidos.includes(char)) {
  let tipoSimbolo = 'Símbolo';
  if (char === '{' || char === '}') tipoSimbolo = 'Delimitador de Bloque';
  else if (char === '(' || char === ')') tipoSimbolo = 'Delimitador de Grupo';
  else if (char === '[' || char === ']') tipoSimbolo = 'Delimitador de Atributo';
  else if (char === ':') tipoSimbolo = 'Separador';
  else if (char === ';') tipoSimbolo = 'Terminador';
  else if (char === '=') tipoSimbolo = 'Operador';

  tokens.push({ lexema: char, tipo: tipoSimbolo, fila: fila + 1, columna });
  i++;
```

```
// Identificadores y palabras
if (/[a-zA-Z_]/.test(char)) {
  const startCol = columna;
  let lexema = '';

  while (i < line.length && /[a-zA-Z0-9_]/.test(line[i])) {    Use conc
    lexema += line[i];
    i++;
    columna++;
  }

  let tipo = 'Identificador';
  if (palabrasReservadas.includes(lexema)) {
    tipo = 'Palabra Reservada';
  }

  tokens.push({ lexema, tipo, fila: fila + 1, columna: startCol });
  continue;
}
```

```
    // Caracteres no reconocidos
    errors.push({
      lexema: char,
      tipo: 'Carácter inválido',
      descripcion: `Carácter '${char}' no reconocido en el lenguaje PKLFP`,
      fila: fila + 1,
      columna
    });

    i++;
    columna++;
  }
}
```

```
// Pila para balanceamiento de símbolos
const stack: { symbol: string, fila: number, columna: number }[] = [];
const pairs: { [key: string]: string } = { '(': ')', '[': ']', '{': '}' };

for (let i = 0; i < tokens.length; i++) {
  const token = tokens[i];

  // Validar estructura de Jugador
  if (token.lexema === 'Jugador') {
    if (i + 1 >= tokens.length || tokens[i + 1].lexema !== ':') {
      errors.push({
        lexema: token.lexema,
        tipo: 'Error sintáctico',
        descripcion: 'Falta ":" después de "Jugador"',
        fila: token.fila,
        columna: token.columna
      });
    }
  }
```

3. Iv

```
export interface PokemonConIV {
  nombre: string;
  sprite: string;
  tipos: string[];
  salud: number;
  ataque: number;
  defensa: number;
  iv: number;
}


// Función para calcular IV basándose en las estadísticas
Tabnine | Edit | Test | Explain | Document
export function calcularIV(salud: number, ataque: number, defensa: number): number {
  // Valores base aproximados para calcular IV
  // En un juego real, estos valores vendrían de la base de datos
  const saludMax = 200; // Valor máximo aproximado
  const ataqueMax = 200;
  const defensaMax = 200;
```

```
export function calcularIV(salud: number, ataque: number, defensa: number): number {
  // Calcular porcentaje de cada estadística
  const saludPorcentaje = Math.min((salud / saludMax) * 100, 100);
  const ataquePorcentaje = Math.min((ataque / ataqueMax) * 100, 100);
  const defensaPorcentaje = Math.min((defensa / defensaMax) * 100, 100);

  // Promedio de las tres estadísticas
  const iv = Math.round((saludPorcentaje + ataquePorcentaje + defensaPorcentaje) / 3);

  return Math.min(Math.max(iv, 0), 100); // Asegurar que esté entre 0 y 100
}

// Función para seleccionar los mejores 6 Pokémon sin repetir tipos
Tabnine | Edit | Test | Explain | Document
export function seleccionarMejores(pokemones: PokemonConIV[]): PokemonConIV[] {
  if (!pokemones || pokemones.length === 0) {
    return [];
  }

  // Ordenar por IV de mayor a menor
  const ordenados = [...pokemones].sort((a, b) => b.iv - a.iv);
```

```
const seleccionados: PokemonConIV[] = [];
const tiposUsados = new Set<string>();

for (const pokemon of ordenados) {
  // Verificar si alguno de los tipos del Pokémon ya fue usado
  const tieneNuevoTipo = pokemon.tipos.some(tipo => !tiposUsados.has(tipo));

  if (tieneNuevoTipo && seleccionados.length < 6) {
    seleccionados.push(pokemon);
    // Agregar todos los tipos del Pokémon al set
    pokemon.tipos.forEach(tipo => tiposUsados.add(tipo));
  }
}

// Si no tenemos 6 Pokémon únicos por tipo, completar con los mejores restantes
if (seleccionados.length < 6) {
  for (const pokemon of ordenados) {
    if (!seleccionados.includes(pokemon) && seleccionados.length < 6) {
      seleccionados.push(pokemon);
    }
  }
}
```

4. Pokenapi

```
interface TypeDetail {
  type: {
    name: string;
  };
}

interface SpriteDetail {
  front_default: string | null;
}

interface PokemonApiResponse {
  name: string;
  sprites: SpriteDetail;
  types: TypeDetail[];
}

export interface PokemonData {
  nombre: string;
  sprite: string;
  tipos: string[];
```

```
Tabnine | Edit | Test | Explain | Document
export async function getPokemonInfo(nombre: string): Promise<PokemonData | null> {
  try {
    const response = await axios.get<PokemonApiResponse>(
      `https://pokeapi.co/api/v2/pokemon/${nombre.toLowerCase()}`
    );

    const data = response.data;

    const sprite = data.sprites?.front_default ?? null;

    if (!sprite) {
      throw new Error('No se encontró la imagen del Pokémon');
    }

    const tipos = data.types.map((t) => t.type.name);

    return {
      nombre: data.name,
      sprite,
      tipos
    };
```

```
  } catch (error) {
    // Validación segura del error
    let errorMessage = 'Error desconocido';

    if (error instanceof Error) {
      errorMessage = error.message;
    } else if (typeof error === 'string') {
      errorMessage = error;
    }

    console.error(`No se pudo obtener el Pokémon ${nombre}:`, errorMessage);
    return null;
  }
}
```