Tarea 1 Grupo 8

Guillermo Achondo
Tomás Concha
Ana Marín

Contenido

01 Desafío

02 Supuestos

03 Demo

04 Funciones

Desafío

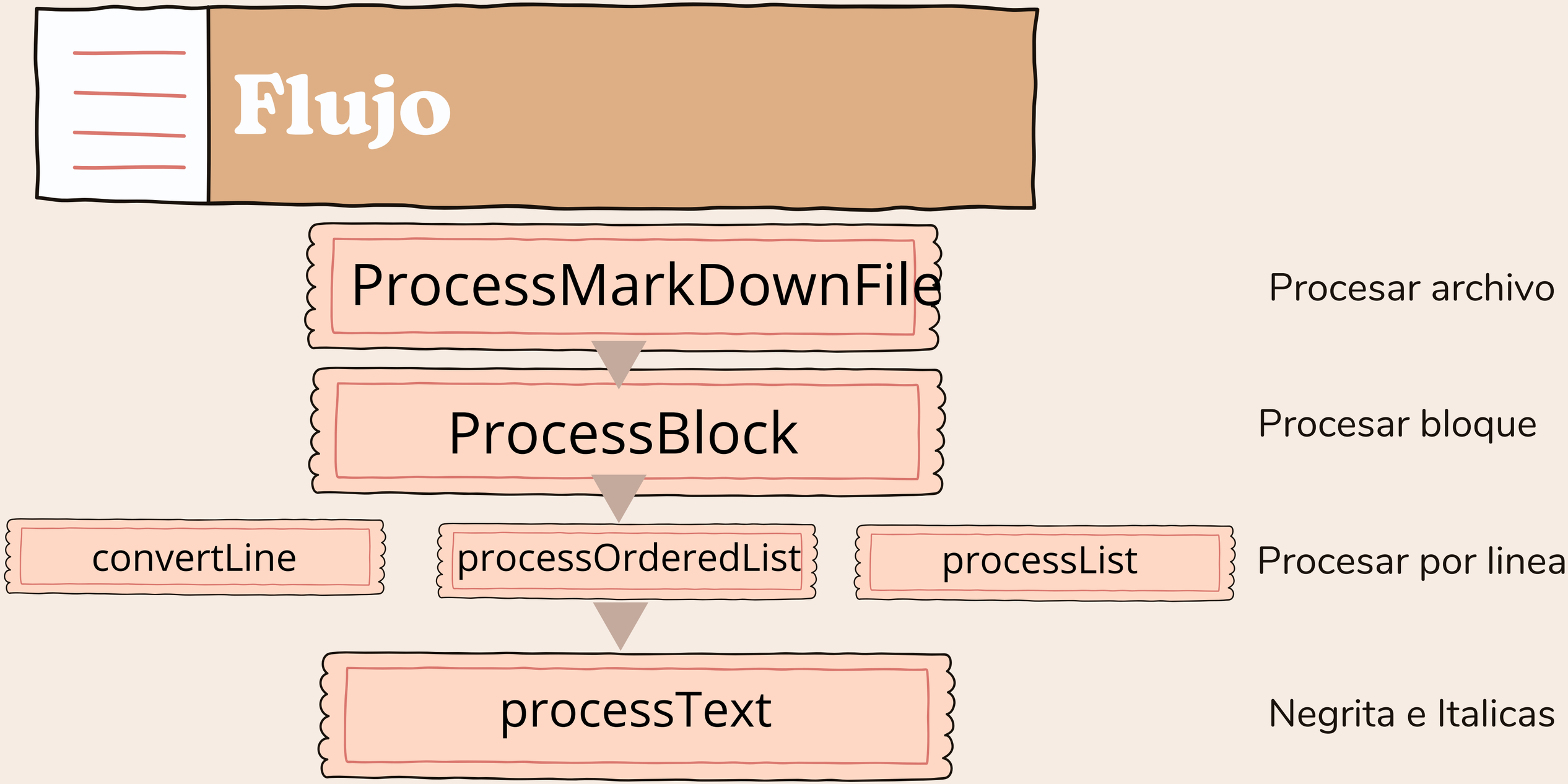
Transformar un archivo de texto en formato de markdown en un archivo html que se vea lo más parecido posible

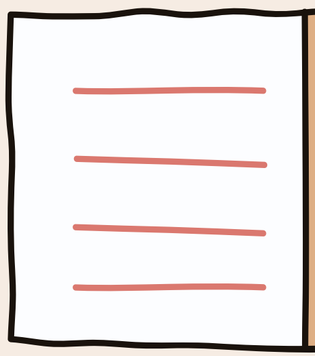
Supuestos

- Un párrafo se escribe en una sola línea.
- Buenas prácticas de MarkDown
- Recibe archivo de texto md

DEMO

Flujo


ProcessMarkDownFile
Procesar archivo
ProcessBlock
Procesar bloque
convertLine
processOrderedList
processList
Procesar por linea
processText
Negrita e Italicas



Separar Markdown

```
4   export function ProcessMarkdownFile(markdownFilePath, outpuFileName="result"){
5       const content = readFileSync(markdownFilePath, {encoding:'utf8', flag:'r'});
6       const contentByBlocks = content
7           .split('\n\n')
8           .map((l) =>l.split("\n"))
9           .filter(n => JSON.stringify(n) !== '[""]' )
10          .map(block => block.filter(line => line));
11      const htmlFileContent = contentByBlocks
12          .map(block => processBlock(block))
13          .map(htmlBlock => htmlBlock.join(""))
14          .join("");
15      writeFile(`output/${outpuFileName}.html`,htmlFileContent , (err) => {
16          if (err) throw err;
17      })
18      return console.log(`File succesfully saved at output/${outpuFileName}.html`)
19  };
```

Procesar Bloque

```
49  // Funcion para procesar un bloque
50  export function processBlock(lines) {
51      if (lines[0].trim().split(" ")[0] === "*") {
52          //procesar lista
53          return processList(lines);
54      }
55      else if (lines[0].trim().split(" ")[0] === "1.") {
56          //procesar lista ordenada
57          return processOrderedList(lines);
58      }
59      else {
60          //procesar lineas
61          return lines.map(line => convertLine(line));
62      }
63  };
```

Procesar Lista

```
31    //Funcion para procesar una lista
32    const processList = (lines) => {
33        const listLines = lines.map(line => {
34            return processText(line.replace('* ', `    <li>`) + '</li>\n');
35        });
36        const result = ["<ul>\n"].concat(listLines).concat(["</ul>\n"]);
37        return result;
38    };
```

Procesar Lista Ordenada

```
40    //Funcion para procesar una lista ordenada
41  const processOrderedList = (lines) => {
42      const listLines = lines.map((value,index) => {
43        return processText(value.replace(`${index+1}. `, `    <li>`) + '</li>\n');
44      });
45      const result = ["<ol>\n"].concat(listLines).concat(["</ol>\n"]);
46      return result;
47    };
```

Procesar Lineas

arr.slice([inicio [, fin]])

```
3   // Función por linea (recibe una linea y devuelve una linea)
4   const convertLine = (line) => {
5       // Línea horizontal
6       if (line.match(/^---$/)) {
7           return '<hr>\n';
8       }
9       const proccesedLine = processText(line);
10      const firstChar = proccesedLine.split(" ")[0];
11      if (countHashtags(firstChar) === firstChar.length){
12          return `<h${firstChar.length}>${proccesedLine.slice(firstChar.length+1)}</h${firstChar.length}>\n`;
13      }
14      else{
15          return `<p>${proccesedLine}</p>\n`;
16      }
17  };
18
19  //Funcion para contar #
20  const countHashtags = (str) => _.countBy(str)["#"] || 0;
```

Lodash
.countBy

Procesar Texto

Negrita: __(.*)__

(.*)

Itálica: _(.*)_

(.)*

```
22    //Funcion para procesar texto, reemplazando negrita y italica
23    const processText = (text) => {
24        return text
25        .replace(/\*\*(.*)\*\*/g, '<strong>$1</strong>')
26        .replace(/__(.*)__/g, '<strong>$1</strong>')
27        .replace(/\*(.*)\*/g, '<em>$1</em>')
28        .replace(/_(.*)_/g, '<em>$1</em>');
29    }
```

RegEx

g: global

(.*): anything

Conclusiones