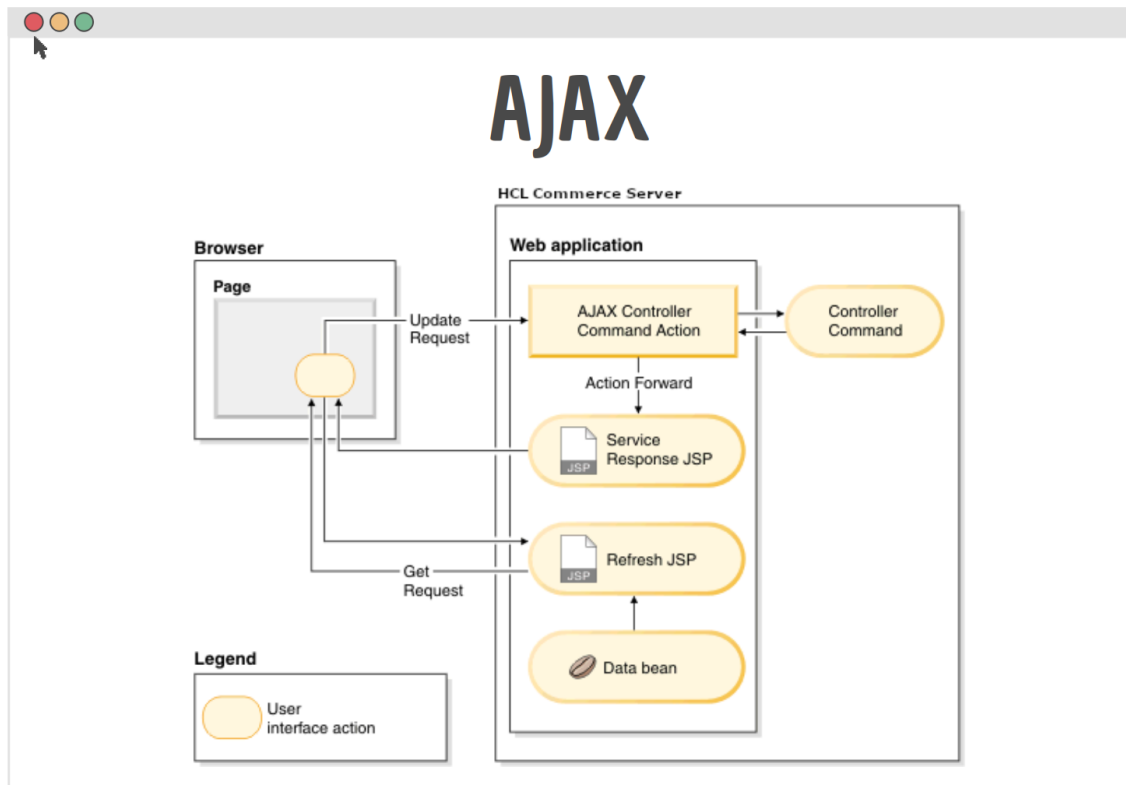




# PROGRAMACIÓN WEB 2



## Profesor(a):

Carlo Jose Luis Corrales Delgado

## Estudiante:

Mamani Anahua, Victor Narciso  
Mamani Huarsaya, Jorge Luis  
Quispe Marca Edysson Darwin  
Velarde Saldaña Jhossep Fabritzio  
Zuñiga Villacorta Peter Sebastian

## Repositorios GitHub:

[https://github.com/jorghee/tarea\\_ajax](https://github.com/jorghee/tarea_ajax)  
[https://github.com/jorghee/ajax\\_markdown](https://github.com/jorghee/ajax_markdown)

18 de mayo, 2024

## Peticiones AJAX a un servidor node.js

### Creando el directorio que contiene los archivos Markdwon

Empezamos copiando archivos de ejemplo .md al directorio files\_markdown. Este directorio es el que responde a las peticiones AJAX.

```
1 $ ls --tree
2 files_markdown
3     calculator_with_java.md
4     example_readme_github.md
5     learning_github.md
6     project_ajax_regions.md
7     server_with_nodejs.md
```

### Creando el servidor con Express

En esta ocasión vamos a utilizar todos los siguientes paquetes para poder resolver direcciones a archivos y directorios, escribir y leer archivos en directorios y lo más importante es el uso del paquete **markdown-it** que se encarga de hacer la conversión de markdown a html.

```
1 const path = require("path");
2 const fs = require("fs");
3 const MarkdownIt = require("markdown-it");
4 const bp = require("body-parser");
5 const express = require("express");
6 const app = express();
7 const md = new MarkdownIt();
8 const MARKDOWN_DIR = path.resolve(__dirname, "files_markdown");
9
10 app.use(bp.json());
11 app.use(express.static("public"));
```

Como se observa, hemos declarado una variable estática **MARKDOWN\_DIR** que almacena la dirección completa al directorio con el cual vamos a trabajar. Además, se ha configurado para que el servidor sirva archivos estáticos desde el directorio **public** con el objetivo de organizar mejor nuestro proyecto, ya que aquí se encontrarán los archivos html, css y los scripts.

Ahora podemos servir el archivo **index.html** para que nuestro proyecto pueda visualizarse.

```
1 /**
2  * Creamos el servidor, request maneja las solicitudes que hacemos y
3  * response es el objeto que maneja las respuestas que envía el servidor.
4  */
5 app.get('/', (request, response) => {
6   response.sendFile(path.resolve(__dirname, "public/index.html"));
7 });
```

### Implementando la función que lista los archivos Markdown

Lo primero que hacemos es configurar el servidor para que pueda manejar la petición y enviar una respuesta al navegador. Como ahora solo necesitamos leer, vamos a usar el paquete **fs** de node.js que nos otorga funciones para poder interactuar con el sistema de archivos del sistema operativo donde se encuentra levantado el servidor.

Podemos hacer que el servidor responda en formato **JSON** como es común hoy en día.

```
1 // Listar archivos Markdown
2 app.get("/files", (request, response) => {
3   fs.readdir(MARKDOWN_DIR, (error, files) => {
4     if (error) {
5       console.error("No se pudo leer los archivos:", error);
6       return;
7     }
8     response.json(files);
9   });
10 });
```

## Solicitud AJAX

Ahora que ya hemos configurado el servidor, podemos empezar a contruir nuestra petición AJAX, en este caso usaremos la función `fetch()` para poder manejar las promesas.

La solicitud la hacemos a la dirección en la cual ha sido configurada el servidor. Una vez que hecha la petición, el servidor envía los nombres de los archivos `.md` y con la función `then()` manejamos esta respuesta.

```
1 function listMarkdownFiles() {
2   fetch("/files")
3     .then(response => response.json())
4     .then(files => {
5       let fileList = "";
6       files.forEach(file => fileList += `<li>${file}</li>`);
7       document.getElementById("files").innerHTML = fileList;
8     })
9     .catch(error => console.error("Error al listar los archivos:", error));
10 }
11
12 export { listMarkdownFiles };
```

Entonces comenzamos iterando por cada valor, que en este caso son los nombres de los archivos, y generamos código html que se encargue de mostrar estos nombres como una lista. La lista que hemos generando con la etiqueta `li` la ponemos en el elemento de ID `files`, concluyendo así la petición ajax.




Figure 1: Lisrado de los archivos markdown

## Implemetando la función que visualiza un archivo Markdown

Del mismo modo, nosotros tenemos que implementar la lógica para decirle al servidor cómo debe manejar una solicitud. En este excepcional caso, nos daremos cuenta que necesitamos enviar un identificador que le permita al servidor identificar qué archivo se desea visualizar. Por lo tanto, la configuración debe recibir un parametro.

```
1 app.get('/files/:fileName', (req, res) => {  
2   const filename = req.params.fileName;  
3   const filePath = path.resolve(MARKDOWN_DIR, filename);  
4  
5   fs.readFile(filePath, 'utf8', (err, data) => {  
6     if (err) {  
7       return res.status(500).json({ error: "Unable to read file" });  
8     }  
9   })  
10 }
```

```
9     const htmlContent = md.render(data);
10     res.json({html: htmlContent});
11   });
12 };
```

Como se observa, el servidor recibe un parametro, dicho parametro es el nombre de archivo que se desea ver. Aqui lo complicado fue pensar cómo recuperar el nombre y enviarle al servidor. Despues veremos que agregando un evento a cada etiqueta **li** podemos enviar este identificador.

Con el identificador ya recibido, podemos buscar el archivo en el correspondiente directorio y empezar a leerlo. Sin embargo, lo que hemos leído solo es texto con la extension **.md**, entonces, aqui usamos el paquete **markdown-it** que se encarga de hacer la conversion y que pueda ser visualizado por el navegador.

### Realizando la petición AJAX

Con el servidor configurado, podemos empezar creando la función que se encarga de hacer la petición. La función entonces envia el identificador al servidor y este responde con un texto formateado a html. Basicamente ese el funcionamiento de esta función la cual modifica la etiqueta **div** de ID content en la cual muestra el texto html.

```
1 export { showMarkdownFile };
2
3 function showMarkdownFile(fileName) {
4   fetch(`/files/${fileName}`)
5     .then(response => {
6       if (!response.ok) {
7         throw new Error(`HTTP error! status: ${response.status}`);
8       }
9       return response.json();
10     })
11     .then(htmlContent => {
12       document.getElementById("content").innerHTML = htmlContent.html;
13     })
14     .catch(error => console.error("Error al mostrar el archivo:", error));
15 }
16
17 export { showMarkdownFile };
```



Figure 2: Vista de un archivo específico

## Implementando la función que crea y almacena un archivo Markdown en el servidor

Finalmente necesitamos guardar en el servidor los archivos creados en la página web. Por lo tanto la función específica del paquete **fs** que usaremos es `writeFile()`.

El servidor necesita recibir el nombre del archivo y el contenido en sí del archivo, por ello recuperamos los valores enviados en el cuerpo de la solicitud y los usamos en el método mencionado del paquete **fs**

```
1 // Crear un archivo
2 app.post("/files", (request, response) => {
3   // Recuperamos los valores
4   const { filename, content } = request.body;
5   console.log(filename, content);
6 }
```

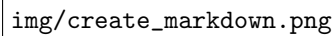
```
7   const filePath = path.resolve(MARKDOWN_DIR, filename);
8   fs.writeFile(filePath, content, error => {
9     if (error) {
10      console.log(error);
11      response.json({ error: "No se pudo crear el archivo" });
12    }
13
14    response.json({ message: "Archivo creado exitosamente" });
15  });
16  });
```

En este caso el servidor no devuelve nada específico que se deba usar en el cliente, solo necesitamos saber si el almacenamiento del archivo fue exitoso.

### Realizando la petición AJAX

Para poder crear y elegir un nombre para guardar el archivo, hemos creado 2 elementos: **input** y **textarea**, entonces necesitamos obtener los valores de estos elementos. Una vez recogidos los valores, los enviamos en el cuerpo de la solicitud ajax. Básicamente en eso consiste la solicitud, finalizando así las tres funcionalidades de esta página web.

```
1  import { listMarkdownFiles } from "./listMarkdown.js";
2
3  function createMarkdownFile() {
4    const filename = document.getElementById("newFileName").value;
5    const content = document.getElementById("newFileContent").value;
6    console.log(filename, content);
7
8    // Generamos los parametros de la solicitud
9    const request = {
10     method: "POST",
11     headers: {
12       "Content-Type": "application/json",
13     },
14
15     // Pasamos el objeto con los valores de filename y content
16     body: JSON.stringify({ filename, content }),
17   };
18
19   fetch("/files", request)
20     .then(response => response.json())
21     .then(data => {
22       console.log(data.message);
23       listMarkdownFiles();
24     })
25     .catch(error => console.error("Error al crear el archivo", error));
26 }
27
28 export { createMarkdownFile }
```



img/create\_markdown.png

Figure 3: Creando archivo y almacenando en el servidor

Como se ha observado en todas las funciones de petición AJAX, al final siempre exportábamos dicha función. Esta idea fue con el objetivo de organizar nuestro proyecto y hacer más manejable en cuanto a los errores.

```
1 import { listMarkdownFiles } from "./listMarkdown.js";
2 import { showMarkdownFile } from "./showMarkdown.js";
3 import { createMarkdownFile } from "./createMarkdown.js";
4
5 listMarkdownFiles();
6
7 document.getElementById('files').addEventListener('click', (event) => {
8   if (event.target.tagName === 'LI')
9     showMarkdownFile(event.target.textContent);
10 });
11
```



12

```
document.getElementById("createFile").addEventListener("click", createMarkdownFile);
```