Renderizado de vistas:

Dado que cuando le peguemos a una ruta en muchas ocasiones vamos a querer renderizar archivos, por ejemplo un formulario, una **primera forma** que utilizaremos será uso del módulo “fs” para esto:

```
const servidor = http.createServer((req, res) =>{
    const parsedUrl = url.parse(req.url, true);

    if(req.method == "GET" && parsedUrl.pathname == "/agregar-registros"){
        // Mostrar formulario para agregar registro
        res.writeHead(200, { 'Content-Type': 'text/html' });
        fs.readFile('form.html', (err, data) => {
            if (err) {
                res.writeHead(500);
                return res.end('Error cargando el formulario.');
            }
            res.end(data);
        });
    }

});
```

Para el caso anterior, configuramos nuestro servidor para que cuando un usuario accede a la ruta “/agregar-registros”, la cual es de tipo “GET”, entonces la respuesta de esa solicitud, identificada con el parámetro “res”, será la lectura del archivo “form.html”. Finalmente la línea “res.end(data)” significa que no se procesará más información y se procederá a renderizar el formulario.

No está de más indicar que “url.parse” retorna un objeto con las diferentes partes de la URL. El parámetro “true” de la función indica que también se tendrán en cuenta los parámetros de la consulta.

Nótese la importancia de la función “readFile” para leer archivos y presentarlos al usuario (cliente en el navegador).

Otra forma de renderizar una vista sería no a través de un archivo sino directamente escribiendo la respuesta.

```
    if (req.method === 'GET' && parsedUrl.pathname === '/mostrar-registros') {
        res.writeHead(200, { 'Content-Type': 'text/html' });
```

```
        res.write('<h1>Ejemplo formulario</h1>');
        res.write('<ul>');
        for (let i = 0; i < 5; i++) {
            res.write(`<li>Listado de registros existentes: ${i}</li>`);
        }
        res.write('</ul>');
        res.end();
    }
```

Como se puede apreciar, a nuestra respuesta (identificada en la variable “res”) le escribimos el contenido con el método “write” y luego todos los elementos del archivo .html que se quiere renderizar. Este método (“write”) generará HTML dinámico.

Procesamiento de información:

Cuando queramos que nuestro servidor procese información enviada por el usuario, por ejemplo en un formulario, debemos indicar que la solicitud es de tipo “POST”:

```
    if (req.method === 'POST' && parsedUrl.pathname === '/agregar-registros')
{
        // Recibir datos del formulario
        let body = '';
        req.on('data', chunk => {
            body += chunk.toString();
        });
        req.on('end', () => {
            const nombre = new URLSearchParams(body).get('nombre');

            // Resto de código para tratar la información, por ejemplo guardarla

            res.writeHead(302, { 'Location': '/mostrar-registros' });
            res.end();
        });
    }
```

“req.on” nos permitirá capturar los datos enviados por el usuario (cuerpo de la solicitud). Los datos se procesan en partes dado que se reciben en fragmentos y de forma asíncrona (por eso el uso de “chunk”).

Cuando llega una solicitud POST, Node.js no recibe el cuerpo de la solicitud completo de

inmediato. En lugar de eso, recibe los datos en fragmentos o "chunks". Por eso se escucha el evento 'data', que se activa cada vez que llega un fragmento de datos. De todas formas no es obligatorio usar “chunk”, también podríamos a cada fragmento de nuestro contenedor de información “data”, ir iterando cada una de sus partes / fragmentos.

```
let body = '';

req.on('data', data => {
    body += data; // Concatenando los fragmentos de datos
});
```

“data” en sí es un evento para recibir los fragmentos de data, y la variable “data” puede ser cualquier otro nombre.

Por último en nuestro evento “end”, “URLSearchParams” se encarga de analizar la variable “body” y obtiene el valor del campo que se desea.