

## Introducción a Node (Parte II)

### Servidores con Node

#### Competencia

- Implementar un servidor básico con Node y el módulo http para habilitar una ruta de comunicación con una aplicación cliente

#### Introducción

Los servidores en el mundo del desarrollo de aplicaciones, juegan un rol tremendamente importante al servir como anfitrión en el backend, encargándose de las comunicaciones necesarias para satisfacer las consultas generadas del lado del cliente.

Node como ya sabes es en pocas palabras JavaScript corriendo en el lado del servidor, esto quiere decir, que con la misma tecnología que usamos para hacer programación orientada a eventos en el navegador, la podemos usar para crear funciones que sirvan como middleware entre una petición y el almacén de datos de un sistema informático.

En este capítulo, aprenderás a crear un servidor con el módulo “http” de Node, generar diferentes rutas para diferentes objetivos, que ofrezcan al cliente la posibilidad de gestionar archivos y lograr en sus contenidos la persistencia de información, la cual es una necesidad infalible hoy en día en la industria que exige cada vez más contenido dinámico.

## El módulo http

El módulo http es un módulo nativo de Node, que nos permite entre tantas cosas crear servidores y realizar consultas a recursos externos. Ya que es un módulo propio de nuestro entorno no es necesario instalarlo ni descargarlo, sino que bastará con importarlo en una variable para poder empezar a ocupar sus métodos y propiedades.

En esta lectura solo necesitaremos los métodos “createServer” y “listen”, los cuales sirven para crear un servidor(createServer) y definir el puerto que ocuparemos para disponer el servidor(listen), no obstante existen muchos más métodos que puedes conocer en su documentación oficial que encontrarás en el siguiente [link](#).

## Ejercicio guiado: Mi primer servidor

Levantemos un servidor que escuche las consultas por medio del puerto 8080, ¿Y cómo verificaremos que esté andando bien? Generamos un mensaje por consola que solo se ejecutará si el servidor es consultado, por lo que si vemos este mensaje por consola tendremos la certeza que todo salió bien.

¡Manos al código!, crea un archivo index.js y realiza los siguientes pasos para crear tu primer servidor con Node:

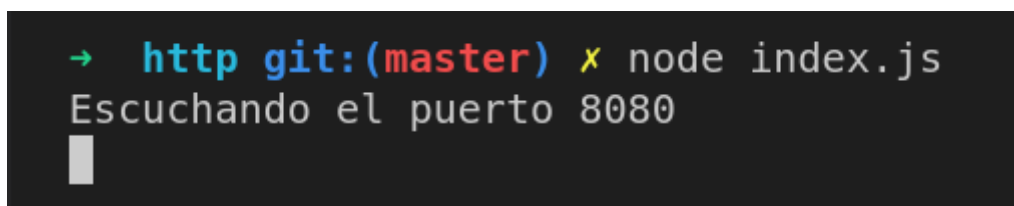
- **Paso 1:** Importar el módulo http en una variable.
- **Paso 2:** Usar el método createServer pasándole como parámetro una función callback con los siguientes parámetros: req, res.
- **Paso 3:** Dentro de la función callback imprime por consola el mensaje “¿Aló? ¿Quién habla?”
- **Paso 4:** Concatenar al método createServer el método “listen”, pasándole como primer parámetro el número 8080, este número representará el puerto por el cual el servidor estará oyendo peticiones, y como segundo parámetro una función de flecha que imprima por consola el mensaje “Escuchando el puerto 8080”.

El código quedaría como te muestro a continuación:

```
// Paso 1
const http = require('http')

// Paso 2
http
  .createServer(function (req, res) {
    // Paso 3
    console.log('¿Aló? ¿Quién habla?')
  })
  .listen(8080, () => console.log('Escuchando el puerto 8080')) // Paso 4
```

- **Paso 5:** Ahora debes abrir la terminal y correr el archivo creado con la línea de comando “**node index.js**” para levantar nuestro servidor y al hacerlo obtendrás la siguiente respuesta por la terminal.



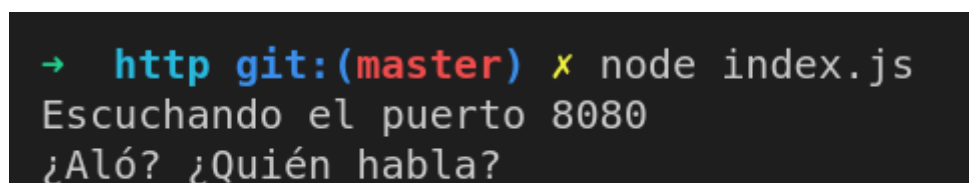
```
→ http git:(master) x node index.js
Escuchando el puerto 8080
```

Imagen 1. Mensaje por consola del servidor encendido y escuchando el puerto 8080.

Fuente: Desafío Latam

Como puedes notar este es el mensaje que estamos mandando por consola en la función callback del método listen de http. Observa bien la terminal y fijate que parece que ha quedado un proceso sin terminar, pero no te preocupes que esto es lo que estamos esperando porque indica que el servidor está activo y atento a consultas.

Ahora si abres tu navegador de preferencia o usas el comando “curl” en otra terminal para consultar la dirección <http://localhost:8080/> obtendrás en la terminal del servidor lo que te muestro en la siguiente imagen:



```
→ http git:(master) x node index.js
Escuchando el puerto 8080
¿Aló? ¿Quién habla?
```

Imagen 2. Mensaje por consola recibido por la recepción de una consulta al servidor.

Fuente: Desafío Latam

¿Vas entendiendo cómo funciona esto? Estoy seguro que sí. Ahora que tenemos creado un servidor, es momento de generar diferentes rutas de comunicación, porque con lo que tenemos hasta ahora, obtendremos el mismo resultado siempre que consultemos la url base que es <http://localhost:8080/>, es decir, si concatenamos algún path como /usuarios o /cimas no obtendremos nada diferente.

Las rutas son diferenciadas por la URL que se está consultando, y ¿Cómo capturamos la URL que están consultando? Para esto tenemos los parámetros request (consulta) y response (respuesta) que fueron los que seteamos en el callback del método createServer como req y res.

Ya que hablamos de una “consulta”, usaremos el parámetro request en su propiedad “url”, esta nos devuelve la url que fue usada para consultar el servidor (sin tomar en cuenta la url base) y teniendo este dato podemos crear condicionales que evalúen si alguna URL es igual a un String definido. Posterior a la lógica de consultas es importante cerrar la consulta usando el método “end” del parámetro “res” porque de lo contrario el cliente no sabrá que la consulta no necesita más tiempo, porque no va a procesar nada más y quedaría en espera sin necesidad.

## Ejercicio guiado: ¿Qué fecha es hoy?

Realizar el siguiente ejercicio para generar las rutas de un servidor, titulado: **¿Qué fecha es hoy?**, en el que se requiere desarrollar un servidor en Node que disponibilice una ruta para consultar la fecha actual.

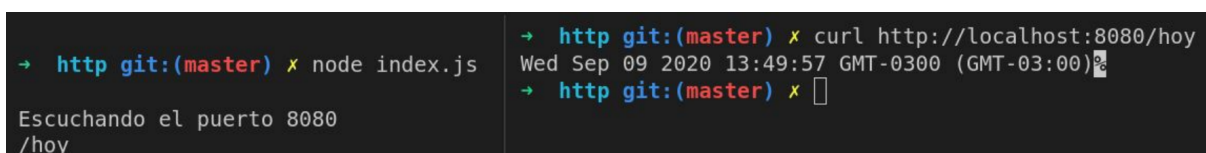
- **Paso 1:** Imprimir por consola el parámetro req en su propiedad url.
- **Paso 2:** Guardar en una variable la url recibida en el parámetro req.
- **Paso 3:** Crear un condicional que evalúe si la url es igual a “/hoy”.
- **Paso 4:** Dentro del bloque de código del condicional usa el parámetro res con un método llamado write, que sirve para devolver al cliente algún mensaje. Este siempre debe ser tipo String, por lo que deberás usar las template literales para concatenar un nuevo objeto Date de JavaScript.
- **Paso 5:** Terminar la consulta con el método “end” del parámetro res.

A continuación te muestro como debe quedar el código de nuestro servidor:

```
const http = require('http')

http
  .createServer(function (req, res) {
    // Paso 1
    console.log(req.url)
    // Paso 2
    const url = req.url
    // Paso 3
    if (url == '/hoy') {
      // Paso 4
      res.write(`${new Date()}`)
    }
    // Paso 5
    res.end()
  })
  .listen(8080, () => console.log('Escuchando el puerto 8080'))
```

- **Paso 6:** Ahora solo falta hacer una consulta al servidor con la siguiente dirección <http://localhost:8080/hoy>, en mi caso lo haré con el comando **curl** en una terminal partida dentro de Visual Studio Code, tú puedes hacer lo mismo o usar tu navegador para consultar el servidor. El resultado te lo muestro en la siguiente imagen:



The screenshot shows a terminal window with two panes. The left pane shows the command `node index.js` being executed, with the output `Escuchando el puerto 8080` and `/hoy`. The right pane shows the command `curl http://localhost:8080/hoy` being executed, with the output `Wed Sep 09 2020 13:49:57 GMT-0300 (GMT-03:00)`.

Imagen 3. Recibiendo respuesta del servidor por una consulta a la ruta /Hoy.

Fuente: Desafío Latam

Acá podemos observar varias cosas:

1. En el lado izquierdo obtuvimos el mensaje por consola que por defecto saldrá cada vez que levantamos el servidor indicando que estará escuchando el puerto 8080.
2. Debajo tenemos escrito `"/hoy"` y esto es gracias al `console.log()` que hicimos en el paso 1.
3. En el lado derecho recibimos como respuesta la fecha actual con todo lo que nos entrega el objeto `Date` de JavaScript.

¡Excelente! Hemos logrado hasta ahora crear un servidor y generar una ruta para obtener la fecha actual.

## Ejercicio propuesto (1)

Basado en el ejercicio para obtener la fecha de hoy, crea un servidor con Node que disponibilice una ruta `/saludo` que devuelva el mensaje `"Hola! ¿Cómo estás?"`.

## Verbos para consultas HTTP

### Competencia

- Distinguir casos de uso de GET, POST, PUT y DELETE para el consumo y envío de datos a un servidor

### Introducción

La comunicación de aplicaciones bajo el protocolo HTTP está compuesta de verbos que representan las acciones que se quiere realizar en una petición, por ese motivo cuando consultamos datos de una API usamos el método GET, que significa “obtener” en español.

En este capítulo, aprenderás que hacen los verbos GET, POST, PUT y DELETE y casos donde se utilizarían, de esta manera entenderás la diferencia entre ellos para poder próximamente desarrollar servidores que acaten estos verbos.

## El protocolo HTTP

Por sus siglas en inglés Hypertext Transfer Protocol (Protocolo de transferencia de hipertexto) es un protocolo que permite una transmisión de archivos hipermedia, entre estos se pueden mencionar HTML, JSON y como tal cualquier texto plano. HTTP es básicamente la base comunicacional del desarrollo de aplicaciones bajo la arquitectura cliente-servidor y una de sus características más resaltantes es que es un protocolo que no conserva estado, es decir, simplemente se realiza, no persiste ningún dato en su utilidad.

Este protocolo goza de varios métodos diferentes para definir el tipo de solicitud que se esté realizando, no obstante los más comunes son los expuestos en la siguiente imagen:



Imagen 4. Diagrama de los métodos básicos del protocolo HTTP.  
Fuente: Desafío Latam

La definición oficial de estos 4 métodos de peticiones te las muestro a continuación:

- **GET:** Solicita una representación del recurso especificado en la petición, es el método por defecto cuando no se especifica ninguno.
- **PUT:** Solicita el reemplazo de un recurso completo o de forma parcial. En caso de no existir se interpretará como la creación de un nuevo elemento. Este método permite el envío de un payload.
- **POST:** Solicita la creación de un nuevo recurso acompañado de un payload.
- **DELETE:** Solicita la eliminación de un recurso especificado en la petición o en el payload.



¿Te ha quedado clara la definición de cada método? Espero que sí pero asegurémonos que sea así y hagamos unos breves ejercicios:

- ¿Al consultar tu cuenta bancaria qué método se estaría ocupando? Correcto! el método GET pues los datos son recibidos desde el servidor y expuestos en nuestra aplicación o sitio web.
- ¿Cuándo actualizas tu curriculum en tu red social para búsqueda de empleo, qué método crees que se esté usando? ¡Muy bien! el método PUT, puesto que tú curriculum sería solo un campo dentro del registro completo de tu perfil, así que sería solo un reemplazo de información de forma parcial o específica.
- ¿Cuándo te registras por primera vez en una plataforma? Exactamente, sería el método POST, ya que este método creará un registro con tus datos. No olvides que el método PUT también podría hacer esto, pero no sería una buena práctica darle uso como si se tratara de una petición POST, porque podría generar confusiones e inconsistencias en una auditoría de peticiones.
- ¿Cuando eliminas una canción de tu lista de spotify que método crees que se esté usando? Por supuesto, lo más probable es que sea el método DELETE.

Como puedes ver sus objetivos son claramente distintos y según lo que el cliente necesite, se ocupará el método correspondiente para una mejor comunicación con el servidor. No obstante, para esta lectura no trabajaremos con payloads por lo que vamos a trabajar sólo con el método GET, generando diferentes rutas que simulan los 4 métodos expuestos en la imagen anterior, en donde la consulta en sí tendrá los datos que se quieran enviar al servidor expuestos como parámetros.

## Parámetros en una consulta HTTP

### Competencia

- Construir un servidor con Node que extraiga los parámetros de una consulta para procesarlos en una función

### Introducción

Los servidores son creados para funcionar como aplicaciones que reciban datos de otra aplicación, comúnmente de una aplicación cliente. Estos datos pueden ser enviados como query strings (parámetros de una url), como un payload definiendo el cuerpo de la consulta en formato JSON o escrito dentro de código XML, aunque este último hoy en día está prácticamente obsoleto.

En este capítulo, verás cómo extraer los parámetros de una consulta, para próximamente poder usar esos datos dentro de tus operaciones programadas en el servidor, como rutas independientes.

## Capturando parámetros de una consulta

Por defecto, cuando realizamos una consulta a un servidor y no especificamos qué tipo de consulta es, estamos usando el método GET. Este método permite enviar datos por medio de parámetros en la url, también conocidos como Query Strings.

Para hacer uso de las query strings en una consulta basta con concatenarle a la URL el símbolo “?” seguido del nombre del parámetro, asignando su valor con el símbolo “=”. Si necesitas mandar más de 1 parámetro, puedes ocupar el símbolo “&” y directamente después hacer mención del próximo parámetro.

El parámetro “request” no trae esta opción como tal, sino que debemos importar un módulo propio de Node llamado “url” el cual nos proporciona utilidades para la resolución y el análisis de URL. Para este ejercicio será necesario ocupar su método “parse” pasándole como parámetro la url que se está consultando en el servidor y extraer de esto la propiedad “query”.

## Ejercicio guiado: Consultando por RUT

Realizar el siguiente ejercicio llamado “**Consultando por RUT**”, en el que se requiere desarrollar un servidor en Node, que disponibilice una ruta para consultar la existencia de una persona en una base de datos, por medio de su rut recibido como una query string. Para simular la base de datos crearemos un arreglo con un único objeto que tendrá los datos del usuario de este ejercicio.

Basado en el servidor creado anteriormente para obtener la fecha de hoy, realiza los siguientes pasos para hacer los cambios correspondientes y lograr realizar este nuevo ejercicio:

- **Paso 1:** Importar el módulo “url” en una variable.
- **Paso 2:** Almacenar en una constante los parámetros recibidos en la consulta y envíalos por consola.
- **Paso 3:** Crear un arreglo con un único usuario con las propiedades: rut, nombre y apellido.
- **Paso 4:** Crear una ruta /usuarios. En esta ocasión debemos usar el método includes de JavaScript puesto que la url al tener parámetros no será igual a la que definimos como ruta sino que tendrá concatenada los query strings.

- **Paso 5:** Usar el método `find` para encontrar un usuario dentro del arreglo creado en donde la propiedad `rut` coincida con el `rut` recibido como parámetro. Guarda el resultado en una variable `"usuario"`.
- **Paso 6:** Usar un operador ternario para evaluar si el `"usuario"` es `true`, en caso de ser así devolver con el método `write` del parámetro `request` `"¡Usuario encontrado! Nombre: ... - Apellido: ..."` (cambia los puntos suspensivos por el nombre y apellido del usuario), en caso contrario devuelve `"No existe ninguna persona registrada con ese RUT"`.

Realizando todos los pasos el código te debería quedar de la siguiente manera :

```
const http = require('http')
// Paso 1
const url = require('url')
http
.createServer(function (req, res) {
  // Paso 2
  const params = url.parse(req.url, true).query
  console.log(params)
  // Paso 3
  let users = [
    {
      rut: '123456789',
      nombre: 'Pat',
      apellido: 'Morita',
    },
  ]
  // Paso 4
  if (req.url.includes('/usuarios')) {
    // Paso 5
    let usuario = users.find((u) => u.rut == params.rut)
    // Paso 6
    usuario
      ? res.write(`¡Usuario encontrado! Nombre: ${usuario.nombre} -
Apellido: ${usuario.apellido}`)
      : res.write('No existe ninguna persona registrada con ese RUT')
  }
  res.end()
})
.listen(8080, () => console.log('Escuchando el puerto 8080'))
```

- **Paso 7:** Si levantas el servidor y realizas una consulta con la siguiente dirección <http://localhost:8080/usuarios?rut=123456789> obtendrás lo que te muestro en la siguiente imagen.

```
→ http git:(master) x curl http://localhost:8080/usuarios?rut=123456789
¡Usuario encontrado! Nombre: Pat - Apellido: Morita
```

Imagen 5. Recibiendo mensaje de éxito del servidor.  
Fuente: Desafío Latam

Como puedes notar, hemos recibido el mensaje de éxito pues si existe un usuario con el rut indicado en la query string.

## Ejercicio propuesto (2)

Basado en el ejercicio de las query strings para saber la existencia de un usuario, crea un servidor con Node que por medio de una ruta “/switch” cambie el valor de una variable booleana por el valor de un parámetro recibido en la ruta.

Devuelve el mensaje “ON” si el valor de la variable booleana es exactamente igual a *true* y un “Off” si no lo es.

## Formularios HTML y servidores con Node

### Competencia

- Construir un servidor con Node que disponibilice cuatro rutas que permitan crear, leer, renombrar y eliminar archivos de texto almacenados dentro del servidor

### Introducción

Ahora que sabes como crear un servidor con el módulo http, también como funciona el protocolo HTTP y a extraer los parámetros de una consulta, es momento que aprendas a generar el proceso completo.

En este capítulo crearemos formularios HTML, que tendrán declarados en su atributo "action" las rutas correspondientes a la escritura, lectura, actualización y eliminación de archivos de texto que se alojarán en nuestro servidor. De esta manera estarás preparado para crear servidores que cumplan las cuatro operaciones de un CRUD.

## Formularios HTML y servidores con Node

Has llegado a un momento importante en la preparación de un full stack developer y es la sincronización o conexión de una aplicación cliente y una aplicación desarrollada en el lado del servidor.

A continuación, dividiremos el capítulo en 2, donde tendremos la “Lógica en el lado del cliente” que corresponderá al código html y la “Lógica en el lado del servidor” correspondiente al desarrollo con Node de nuestro servidor.

### Lógica en el lado del cliente

La gestión de archivos como ya has aprendido en la lectura anterior se maneja desde Node con el módulo File System (fs). ¿Qué pasaría si mezclamos esta herramienta con las comunicaciones por HTTP con el cliente? Correcto, permitiremos extender las capacidades de Node a todos los usuarios que ocupe las rutas definidas en el servidor. De esta manera podremos crear un CRUD controlado por peticiones desde el cliente.

En la mayoría de los sistemas bajo la arquitectura cliente-servidor, los datos son escritos por un usuario dentro de un formulario en HTML y al ejecutarse su método “submit”, la información es enviada por el protocolo HTTP al servidor.

### Ejercicio guiado: Mi repertorio

Realizar el siguiente ejercicio llamado “**Mi repertorio**”, en el que se requiere desarrollar un servidor en Node que disponibilice 4 rutas para la creación, lectura, renombramiento y eliminación de un archivo que contenga una lista con canciones.

Para aplicar la lógica en el lado del cliente, crea un documento HTML con 2 formularios:

- **Primer formulario:** Debe tener un input para la definición del nombre del archivo a crear y un texto para el contenido del mismo. Asegurate de declarar el atributo “name” en los campos de entrada puesto que este será el identificador dentro de las query strings. Además, el atributo “action” del formulario debe ser la siguiente <http://localhost:8080/crear>.
- **Segundo formulario:** Debe tener un input para la definición del nombre del archivo a consumir e igual que el primer formulario, debe tener el atributo “action” pero en esta ocasión la dirección será <http://localhost:8080/leer>.

```
<form action="http://localhost:8080/crear">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  Contenido:
  <textarea name="contenido"></textarea>
  <br />
  <button>Crear</button>
</form>
<hr />

<!-- Segundo Formulario -->
<form action="http://localhost:8080/leer">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>Leer</button>
</form>
```

## Lógica en el lado del servidor

Ahora que tenemos los formularios, lo siguiente será crear las rutas para diferenciar las funciones que queremos que cumpla nuestro servidor, esto lo lograremos simplemente con condicionales que evalúen cuál es la URL que se está usando para consultar al servidor. Para esto deberás crear un archivo `index.js` y seguir los siguientes pasos para desarrollar la lógica de nuestro servidor que permita crear y leer archivos de texto alojados en nuestro servidor.

- **Paso 1:** Importar el módulo `http` en una constante.
- **Paso 2:** Importar el módulo `url` en una constante.
- **Paso 3:** Importar el módulo `File System` en una constante.
- **Paso 4:** Crear un servidor con el método `createServer` del módulo `http` que esté disponible en el puerto 8080.
- **Paso 5:** Almacenar los parámetros de la consulta en una constante con el método `parse` del módulo `url` y su propiedad `query`.

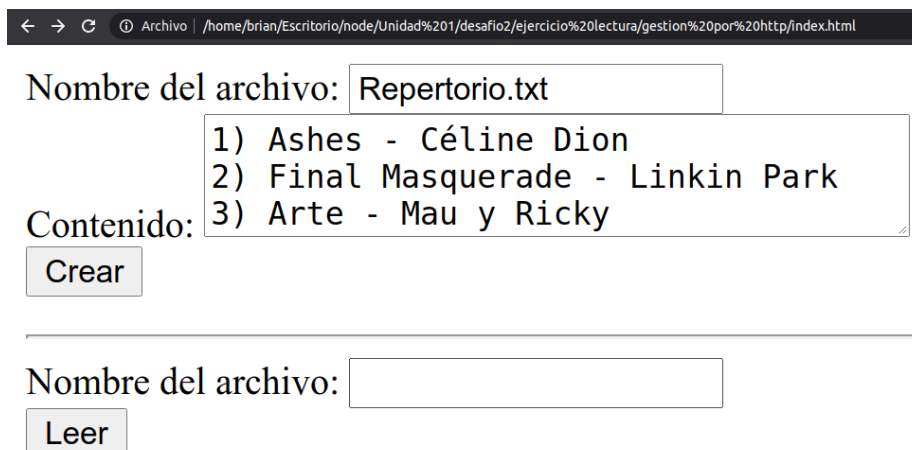


- **Paso 6:** Crear una ruta `"/crear"` que ejecute la creación de un archivo con el método `writeFile` del módulo `File System`, usando los parámetros `nombre` y `contenido` de la url expuestos en la url de la petición. Devuelve un mensaje de éxito al cliente.
- **Paso 7:** Crea una ruta `"/leer"` que use el método `readFile` del módulo `File System` para obtener el contenido del archivo cuyo nombre debe ser el obtenido por query string.

A continuación te muestro cómo quedará el código del servidor siguiendo los pasos previos:

```
// Paso 1
const http = require('http')
// Paso 2
const url = require('url')
// Paso 3
const fs = require('fs')
// Paso 4
http
.createServer(function (req, res) {
  // Paso 5
  const params = url.parse(req.url, true).query
  const nombre = params.nombre
  const contenido = params.contenido
  // Paso 6
  if (req.url.includes('/crear')) {
    fs.writeFile(nombre, contenido, () => {
      res.write('Archivo creado con éxito!')
      res.end()
    })
  }
  // Paso 7
  if (req.url.includes('/leer')) {
    fs.readFile(nombre, (err, data) => {
      res.write(data)
      res.end()
    })
  }
})
.listen(8080, () => console.log('Escuchando el puerto 8080'))
```

- **Paso 8:** Ahora veamos si realmente funciona, abre documento HTML con los formularios y escribe `Repertorio.txt` en el input y una lista de tus 3 canciones favoritas del momento en el textarea como te muestro en la siguiente imagen:



← → ↻ ⓘ Archivo | /home/brian/Escritorio/node/Unidad%201/desafio2/ejercicio%20lectura/gestion%20por%20http/index.html

Nombre del archivo:

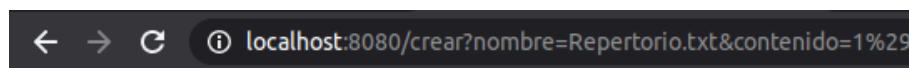
Contenido:

---

Nombre del archivo:

Imagen 6. Escribiendo valores en formulario para la creación del archivo Repertorio.txt.  
Fuente: Desafío Latam

Escrito el repertorio procedemos a presionar el botón “crear” y deberás obtener lo que te muestro en la imagen 7.



## Archivo creado con éxito!

Imagen 7. Respuesta del servidor con mensaje de éxito indicando que el archivo fue creado.  
Fuente: Desafío Latam

Ahora falta probar con el otro formulario si en efecto fue creado el archivo Repertorio.txt y su contenido es el indicado en el formulario anterior. Para esto vuelve a abrir el documento HTML pero ahora escribe el nombre del archivo en el input del segundo formulario. Te muestro un ejemplo en la siguiente imagen:



← → ↻ ⓘ Archivo | /home/brian/Escritorio/node/Unidad%201/desafio2/ejercicio%20lectura/gestion%20po

Nombre del archivo:

Contenido:

Crear

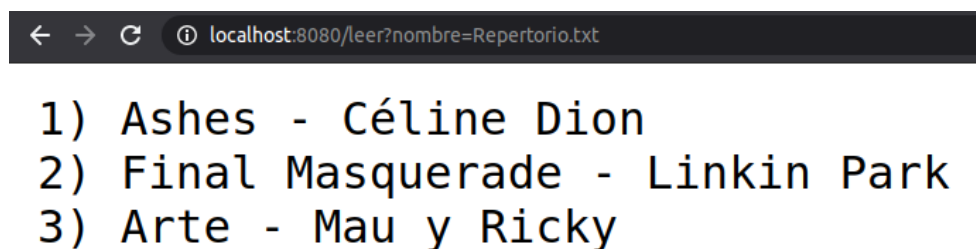
---

Nombre del archivo:

Leer

Imagen 8. Escribiendo valores en formulario para la creación del archivo Repertorio.txt.  
Fuente: Desafío Latam

Ahora haz click en el botón “leer” y deberás obtener lo que te muestro en la imagen 9.



← → ↻ ⓘ localhost:8080/leer?nombre=Repertorio.txt

1) Ashes - Céline Dion  
2) Final Masquerade - Linkin Park  
3) Arte - Mau y Ricky

Imagen 9. Mensaje recibido del servidor con el contenido del archivo Repertorio.txt.  
Fuente: Desafío Latam

Excelente, hasta ahora hemos logrado tratar la siglas C y R del CRUD, es decir, hemos logrado crear y leer un archivo desde el cliente basado en rutas del servidor. ¿Qué falta por hacer? Correcto, faltan las siglas U y D para actualizar y eliminar.

Para esto agrega en el documento HTML 2 formularios especificando las ruta y los campos correspondientes. En el siguiente código te muestro como quedaría:

```
<!-- Tercer Formulario -->
<form action="http://localhost:8080/renombrar">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>Renombrar</button>
</form>

<!-- Cuarto Formulario -->
<form action="http://localhost:8080/eliminar">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>eliminar</button>
</form>
```

Nota como va variando el valor del atributo "action" en función del objetivo a lograr con el envío de esos datos.

Ahora a nivel del servidor solo falta incluir las rutas "renombrar" y "eliminar" en donde usaremos el método rename y el unlink de File System respectivamente. Sigue las siguientes instrucciones para la construcción de estas rutas.

- **Paso 1:** Crear una ruta "/renombrar" que procese el método rename del módulo fileSystem especificando el nombre del archivo devolviendo en su callback un mensaje de éxito.
- **Paso 2:** Crear una ruta "/eliminar" que procese el método unlink del módulo File System especificando el nombre del archivo devolviendo en su callback un mensaje de éxito.

El código de las rutas quedaría como te muestro en el siguiente código.

```
// Paso 1
if (req.url.includes('/renombrar')) {
  fs.rename('Repertorio.txt', nombre, (err, data) => {
    res.write(`Archivo Repertorio.txt renombrado por ${nombre}`)
    res.end()
  })
}

// Paso 2
if (req.url.includes('/eliminar')) {
  fs.unlink(nombre, (err, data) => {
    res.write(`Archivo ${nombre} eliminado con éxito`)
    res.end()
  })
}
```

Ahora abre el documento HTML y escribe cómo se llamará ahora el archivo Repertorio.txt, en mi caso le pondré “Renombrado.txt” así como te muestro en la siguiente imagen.

Nombre del archivo:

Renombrar

Nombre del archivo:

eliminar

Imagen 10. Escribiendo el nuevo nombre del archivo Repertorio.txt.

Fuente: Desafío Latam

Al presionar el botón “Renombrar” deberás recibir el mensaje que te muestro en la imagen 11.

← → ↻ ⓘ localhost:8080/renombrar?nombre=Renombrado.txt

Archivo Repertorio.txt renombrado por Renombrado.txt

Imagen 11. Mensaje recibido por el navegador indicando que el archivo fue renombrado.

Fuente: Desafío Latam

Si revisas el archivo propiamente en la carpeta del servidor podrás comprobar que en efecto el nombre del archivo fue cambiado.

¡Muy bien! Ahora solo falta probar la ruta para eliminar el archivo, vuelve a la pestaña en donde está renderizado el documento HTML y escribe en el cuarto formulario el nombre “Renombrado.txt” o el que hayas decidido así como te lo muestro en la siguiente imagen:

---

Nombre del archivo:

---

Nombre del archivo:

Imagen 12. Escribiendo el nombre del archivo Renombrado.txt para eliminarlo.

Fuente: Desafío Latam

Al presionar el botón “eliminar” deberás entonces recibir el mensaje de éxito desde el servidor, tal y como te muestro en la imagen 13:

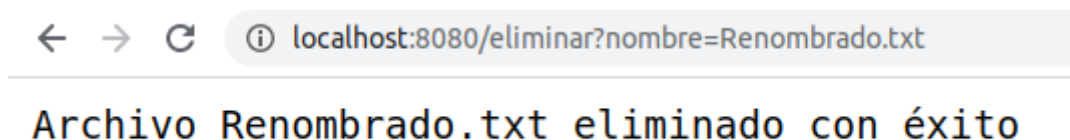


Imagen 13. Mensaje recibido por el navegador indicando que el archivo fue eliminado.

Fuente: Desafío Latam

Y listo, ¡Felicitaciones! Has logrado crear tu primer CRUD full stack. Ahora solo queda practicar mucho e inventar sin límites.

## Ejercicio propuesto (3)

Basado en el ejercicio de los formulario HTML y el servidor para la creación de Repertorio.txt, crea otro sistema tipo CRUD que sirva como registros de gastos, es decir, en este ejercicio el planteamiento del problema será un control financiero personal en donde la persona podrá registrar los gastos que tuvo esta semana, además de consultarlo, poder renombrar el archivo de texto llamado Gastos.txt y eliminarlo en su totalidad.

En el lado del servidor deberás crear las 4 rutas con su lógica correspondiente con el módulo file System.

Puedes ocupar los mismos formularios HTML del ejercicio de Repertorio.

## Resumen

A lo largo de esta lectura cubrimos:

- El módulo http en su método createServer para la creación de servidores.
- El protocolo HTTP.
- Casos de uso de los métodos básicos del protocolo HTTP.
- Obtención de los parámetros de una consulta GET (Query Strings).
- Envío de información de un formulario HTML al servidor.
- Gestión de archivos realizado desde las peticiones a las rutas creativas en el servidor.

## Solución de los ejercicios propuestos

1. Basado en el ejercicio para obtener la fecha de hoy, crea un servidor con Node que disponibilice una ruta /Saludo que devuelva el mensaje "Hola! ¿Cómo estás?".

```
const http = require('http')

http
  .createServer(function (req, res) {
    console.log(req.url)
    const url = req.url
    if (url === '/Saludo') {
      res.write(`Hola! ¿Cómo estás?`)
    }
    res.end()
  })
  .listen(8080, () => console.log('Escuchando el puerto 8080'))
```



2. Basado en el ejercicio de las query strings para saber la existencia de un usuario, crea un servidor con Node que por medio de una ruta `"/switch"` cambie el estado de una variable de tipo booleana por un parámetro (`true` o `false`) recibido capturado en la URL de la petición.

Devuelve el mensaje `"ON"` si el valor de la variable booleana es exactamente igual a `true` y un `"Off"` si no lo es

```
const http = require('http')
const url = require('url')
http
  .createServer(function (req, res) {
    let boolean = false
    const params = url.parse(req.url, true).query

    if (req.url.includes('/switch')) {
      let buleano = params.buleano
      buleano == 'true' ? (boolean = true) : (boolean = false)
      boolean ? res.write('ON') : res.write('OFF')
    }
    res.end()
  })
  .listen(8080, () => console.log('Escuchando el puerto 8080'))
```

3. Basado en el ejercicio de los formulario HTML y el servidor para la creación de Repertorio.txt, agrega 2 formularios para el renombramiento del archivo y para la eliminación del mismo.

En el lado del servidor deberás crear las rutas y la lógica correspondiente con el módulo File system.

```
<!-- Primer Formulario -->
<form action="http://localhost:8080/crear">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  Contenido:
  <textarea name="contenido"></textarea>
  <br />
  <button>Crear</button>
</form>
<hr />
<!-- Segundo Formulario -->
<form action="http://localhost:8080/leer">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>Leer</button>
</form>

<!-- Tercer Formulario -->
<form action="http://localhost:8080/renombrar">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>Renombrar</button>
</form>

<!-- Cuarto Formulario -->
<form action="http://localhost:8080/eliminar">
  Nombre del archivo:
  <input name="nombre" />
  <br />
  <button>eliminar</button>
</form>
```

```
const http = require('http')
const url = require('url')
const fs = require('fs')
http
.createServer(function (req, res) {
  const params = url.parse(req.url, true).query
  const nombre = params.nombre
  const contenido = params.contenido
  if (req.url.includes('/crear')) {
    fs.writeFile(nombre, contenido, () => {
      res.write('Archivo creado con éxito!')
      res.end()
    })
  }
  if (req.url.includes('/leer')) {
    fs.readFile(nombre, (err, data) => {
      res.write(data)
      res.end()
    })
  }
  if (req.url.includes('/renombrar')) {
    fs.rename('Gastos.txt', nombre, (err, data) => {
      res.write(`Archivo Gastos.txt renombrado por ${nombre}`)
      res.end()
    })
  }
  if (req.url.includes('/eliminar')) {
    fs.unlink(nombre, (err, data) => {
      res.write(`Archivo ${nombre} eliminado con éxito`)
      res.end()
    })
  }
})
.listen(8080, () => console.log('Escuchando el puerto 8080'))
```