



# Introducción a Node

HTTP



***Implementar un servidor web de contenidos estáticos y dinámicos utilizando motores de plantillas acordes al entorno Node Express para dar solución a un problema.***

- Unidad 1:  
Introducción a Node
- Unidad 2:  
Node y el gestor de paquetes
- Unidad 3:  
Persistencia



Te encuentras aquí



## ¿Qué aprenderás en esta sesión?

- *A distinguir casos de uso de GET, POST, PUT y DELETE para el consumo y envío de datos a un servidor.*

¿Alguna vez te has  
preguntado cómo funciona  
la comunicación entre tu  
navegador y los servidores  
cuando accedes a un sitio  
web?



***/\* Protocolo HTTP \*/***

# 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.



GET



PUT



POST

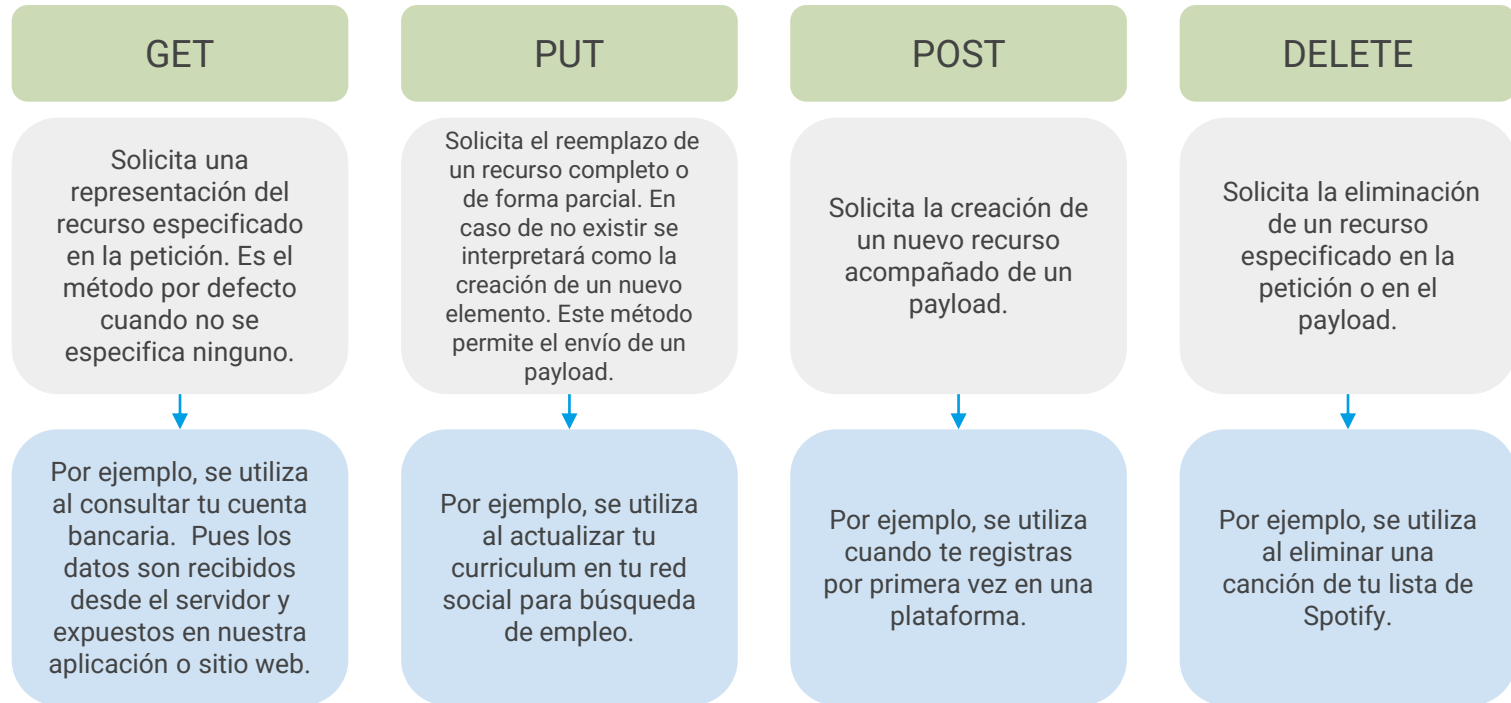


DELETE



# Protocolo HTTP

## Verbos para consultas HTTP



**/\* Capturando parámetros de una  
consulta \*/**



# 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" incluye entre sus propiedades el atributo "query", con el que podremos obtener los parámetros de la URL que recibamos en una petición a nuestro servidor.

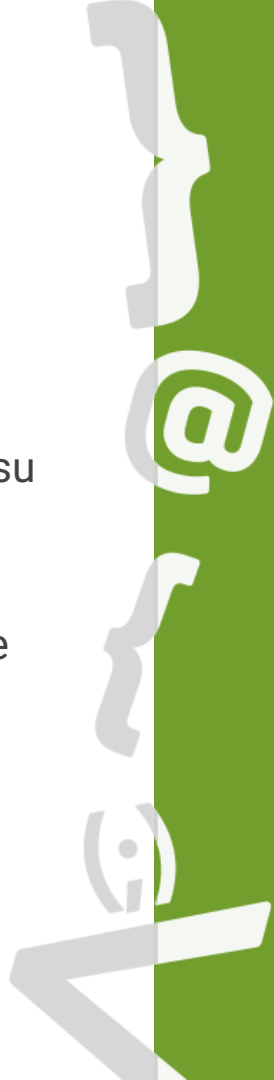
# Demostración "Consultando por tu RUT"



## Ejercicio guiado

### *Consultando por tu 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.
- Realiza los siguientes pasos para realizar este nuevo ejercicio:



# Ejercicio guiado

*Consultando por tu RUT*

- **Paso 1:** Crea una ruta GET /usuarios
- **Paso 2:** A partir del atributo query del objeto req, usar el destructuring para acceder al RUT de la Query String recibida en la URL.
- **Paso 3:** Utiliza el método find para intentar encontrar un usuario con un RUT igual al recibido.
- **Paso 4:** En caso de encontrar un usuario, devolver el nombre y apellido del mismo, de lo contrario, informar que no se encontró ningún usuario con el RUT recibido.



```
const users = [
  {
    rut: '123456789',
    nombre: 'Pat',
    apellido: 'Morita',
  },
]
// Paso 1
app.get("/usuarios", (req, res) => {
  // Paso 2
  const { rut } = req.query
  // Paso 3
  const usuarioEncontrado = users.find(u => u.rut == rut)
  // Paso 4
  if (usuarioEncontrado) {
    const { nombre, apellido } = usuarioEncontrado
    res.send(`¡Usuario encontrado! Nombre: ${nombre} - Apellido: ${apellido}`)
  } else {
    res.send(`No se encontró ningún usuario con el rut ${rut}`)
  }
})
```



## Ejercicio guiado

### Consultando por tu RUT

Ahora utilizamos el comando “curl” en la terminal para realizar una consulta HTTP a la ruta que hemos creado, así como se muestra en la siguiente imagen.

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

Como puedes notar, hemos recibido el mensaje de éxito, porque sí existe un usuario con el RUT indicado en la query string. Además, del atributo query del objeto req en las rutas, también disponemos de la propiedad params para obtener los parámetros de una consulta con payload, aunque eso lo veremos más adelante.

# **/\* Formularios HTML y servidores con Express js \*/**

# Formularios HTML y servidores con Express js

## *Lógica en el lado del cliente*

- La gestión de archivos se maneja desde Node con el módulo File System (fs). ¿Qué pasaría si mezclamos esta herramienta con las comunicaciones por HTTP con el cliente? Permitiremos extender las capacidades de Node a todos los usuarios que ocupen 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.



# Demostración "Mi repertorio"



# Ejercicio guiado

## *Mi repertorio*

Realizar el siguiente ejercicio llamado “Mi repertorio”, en el que se requiere desarrollar un servidor con Express Js 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 formularios:

- **Primer formulario:** Debe tener un input para la definición del nombre del archivo a crear y un textarea para el contenido del mismo. Asegúrate 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>
```



# Formularios HTML y servidores con Express js

## *Lógica en el lado del servidor*

- Ahora que tenemos los dos primeros formularios, lo siguiente será crear las rutas para diferenciar las funciones que queremos que cumpla nuestro 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 él.

# Ejercicio guiado

## *Mi repertorio*

- **Paso 1:** Crea una ruta GET /crear.
- **Paso 2:** Crea 2 variables con el nombre y contenido recibidos por Query Strings.
- **Paso 3:** Utiliza File System para crear un archivo con el nombre y contenido recibidos, notificando posteriormente a la aplicación cliente que se ha creado un archivo con éxito.
- **Paso 4:** Crea una ruta GET /leer.
- **Paso 5:** Crea 1 variable con el nombre recibido por Query Strings.
- **Paso 6:** Utiliza File System para leer un archivo con el nombre recibido enviando su contenido a la aplicación cliente que está realizando la consulta HTTP.



```
// Paso 1
app.get("/crear", (req, res) => {
  // Paso 2
  const { nombre, contenido } = req.query
  // Paso 3
  fs.writeFile(nombre, contenido, () => {
    res.send('Archivo creado con éxito!')
  })
})

// Paso 4
app.get("/leer", (req, res) => {
  // Paso 5
  const { nombre } = req.query
  // Paso 6
  fs.readFile(nombre, (err, data) => {
    res.send(data)
  })
})
```



# Ejercicio guiado

## Mi repertorio

- **Paso 7:** Ahora veamos si realmente funciona. Abre el documento HTML con los formularios y escribe “Repertorio.txt” en el input. Luego haz una lista de tus 3 canciones favoritas del momento en el textarea, como te mostramos en la siguiente imagen:



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

Nombre del archivo:

Contenido:

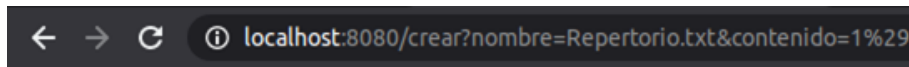
---

Nombre del archivo:

## Ejercicio guiado

### *Mi repertorio*

Escrito el repertorio, procedemos a presionar el botón “crear” y deberás obtener lo que te mostramos en la siguiente imagen:



Archivo creado con éxito!

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.

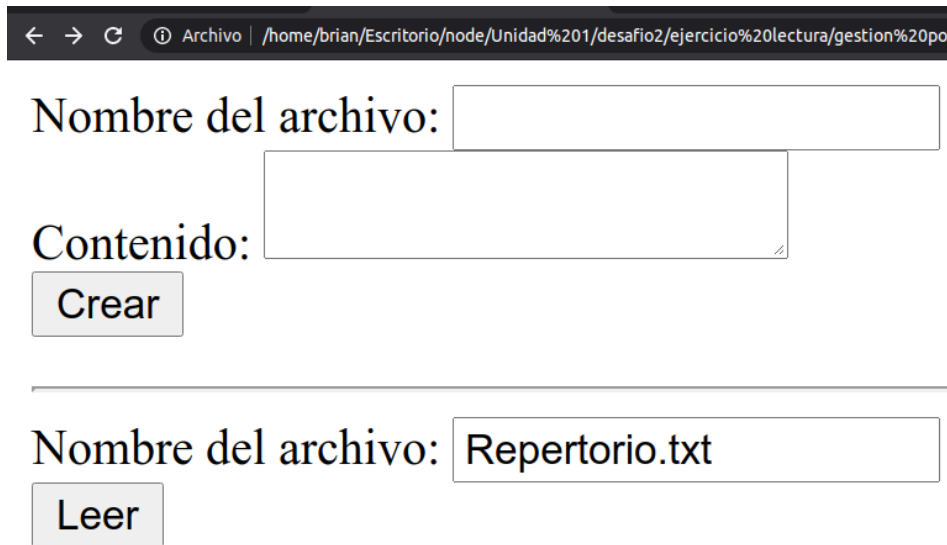




# Ejercicio guiado

## Mi repertorio

Por ejemplo, como se ve en la siguiente imagen:



The screenshot shows a web browser window with a dark address bar containing the text: Archivo | /home/brian/Escritorio/node/Unidad%201/desafio2/ejercicio%20lectura/gestion%20po. Below the address bar, there are two main sections. The first section has a label 'Nombre del archivo:' followed by a text input field, and below that, a larger text area labeled 'Contenido:'. A 'Crear' button is positioned below the text area. The second section, separated by a horizontal line, has a label 'Nombre del archivo:' followed by a text input field containing the text 'Repertorio.txt'. A 'Leer' button is positioned below this input field.

Nombre del archivo:

Contenido:

Crear

---

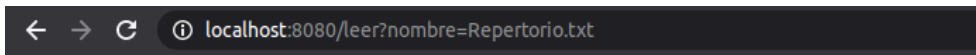
Nombre del archivo:

Leer

# Ejercicio guiado

## *Mi repertorio*

Ahora haz clic en el botón “leer” y deberás obtener lo que se muestra en la imagen:



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

Excelente, hasta ahora hemos logrado tratar las 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? Faltan las siglas U y D para actualizar y eliminar.

# Ejercicio guiado

## *Mi repertorio*

Para esto agrega en el documento HTML 2 formularios especificando las rutas y los campos correspondientes.

- **Tercer formulario:** Debe tener un input para la definición del nuevo nombre del archivo que creamos en el primer formulario y su atributo “action” será <http://localhost:8080/renombrar>.
- **Cuarto formulario:** Debe tener un input para escribir el nombre del archivo, queremos eliminar y su atributo “action” será <http://localhost:8080/elminiar>.



```
<!-- 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>
```



# Ejercicio guiado

## *Mi repertorio*

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:** Crea una ruta GET /renombrar.
- **Paso 2:** Crea 1 variable con el nombre recibido por Query Strings.
- **Paso 3:** Utiliza File System para renombrar un archivo con el nombre recibido, notificando posteriormente a la aplicación cliente que se ha renombrado con éxito.



# Ejercicio guiado

## Mi repertorio

- **Paso 4:** Crea una ruta GET /eliminar.
- **Paso 5:** Crea 1 variable con el nombre recibido por Query Strings.
- **Paso 6:** Utiliza File System para eliminar un archivo con el nombre recibido notificando a la aplicación cliente que la operación fue realizada con éxito.

```
// Paso 1
app.get("/renombrar", (req, res) => {
  // Paso 2
  const { nombre } = req.query
  // Paso 3
  fs.rename('Repertorio.txt', nombre, (err, data) => {
    res.send(`Archivo Repertorio.txt renombrado por
    ${nombre}`)
  })
})

// Paso 4
app.get("/eliminar", (req, res) => {
  // Paso 5
  const { nombre } = req.query
  // Paso 6
  fs.unlink(nombre, (err, data) => {
    res.send(`Archivo ${nombre} eliminado con éxito`)
  })
})
```

# Ejercicio guiado

## *Mi repertorio*

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:

---

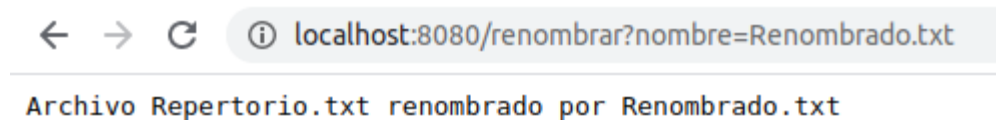
Nombre del archivo:



# Ejercicio guiado

## *Mi repertorio*

Al presionar el botón “Renombrar” deberás recibir el mensaje que se muestra en la siguiente imagen



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



# Ejercicio guiado

## *Mi repertorio*

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 se muestra en la siguiente imagen:

---

Nombre del archivo:

---

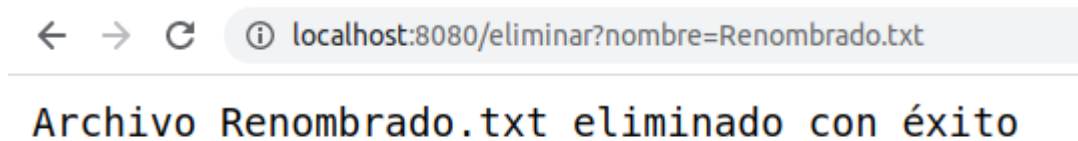
Nombre del archivo:



# Ejercicio guiado

## *Mi repertorio*

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



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

¿Existe algún concepto que  
no hayas comprendido?





## Próxima sesión...

- *Guía de ejercicios - Introducción a Node (I)*

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

