Lectura - API REST (Parte I)

Express-fileupload

Competencias

- Reconocer el uso del paquete express-fileupload para carga de archivos alojados en el servidor.
- Desarrollar una ruta para la carga de archivos usando el paquete express-fileupload.

Introducción

En este capítulo conocerás y aprenderás a utilizar express-fileupload, otro paquete creado para ser utilizado con el framework Express y que nos ofrece la posibilidad de cargar archivos en nuestro servidor a partir de una consulta HTTP.

En sesiones anteriores aprendimos a manejar ficheros con el módulo nativo File System, no obstante los archivos los creamos con el método writeFile y nacían de la lógica interna de nuestras aplicaciones. Integrando el paquete express-fileupload en nuestros servidores podremos recibir archivos enviados directamente desde el cliente a través de un formulario HTML, además de poder manipular sus atributos y restringir el proceso de carga definiendo un límite de Bytes por archivo.

Gestión de archivos

En el desarrollo de aplicaciones puede surgir la necesidad de trabajar no solo con datos, sino también con archivos y como bien sabes, Node cuenta con un módulo integrado de forma nativa (File System) para la lectura y escritura de ficheros, no obstante, nunca está demás pensar en cómo agilizar y/o potenciar nuestros servidores para poder ofrecer un servicio más completo.

Si hablamos de gestión de archivos, se debe mencionar la carga y descarga de ficheros o documentos multimedia, en especial ahora que todo está migrando o ya está migrado a la nube. Una de las tantas ventajas de desarrollar con el framework Express, es poder contar con una inmensa variedad de plugins en NPM, que fueron creados particularmente para ayudarnos con esta y muchas otras tareas, entre las diferentes alternativas tenemos el paquete express-fileupload.

¿Qué es express-fileupload?

En el mundo Node, podemos encontrar muchos paquetes con diferentes funciones que sirven como complemento de una aplicación en el lado del servidor. Express-fileupload, es un plugin que sirve principalmente para dotar a nuestro servidor de la funcionalidad "Upload File".

El atractivo principal de este paquete, se da por su objeto de configuración que nos permite por ejemplo definir un peso límite en la carga de un archivo que se pretende alojar en el servidor a partir de una consulta HTTP, además nos provee de propiedades con las que podremos manipular varios de los atributos de los archivos, previo a su almacenamiento.

Integración

Para poder integrar este paquete, debe ser instalado previamente, así que utiliza el siguiente comando por la terminal para iniciar nuestro proyecto y descargar su repositorio con NPM:

```
npm init -y
npm install --save express-fileupload
```

Con el paquete instalado en los node_modules, podemos proceder con la integración en nuestro servidor con Express, el cual solo amerita incluirlo como middleware y definir su objeto de configuración, tal y como te muestro en el siguiente código:

```
const expressFileUpload = require('express-fileupload');
app.use( expressFileUpload(<objeto de configuración>));
```

¿Qué estamos haciendo acá? Dentro del método "use" ejecutamos la instancia de nuestro paquete express-fileupload, ¿Por qué? Sucede que así como la instancia de Express, la importación de este paquete no nos devuelve un objeto sino una función, la cual al ser llamada, recibe como parámetro un objeto de configuración con varias propiedades con las que podremos personalizar el uso de este plugin.

Configuración

El archivo de configuración se declara como argumento en la ejecución de la instancia de express-fileupload, este objeto de configuración cuenta con varias propiedades pero en esta lectura ocuparemos las siguientes:

- **limits:** Es un objeto en el que podemos definir a través del atributo "fileSize" la cantidad máxima de Bytes que permitirá el paquete para la carga de archivos.

- **abortOnLimit:** Es una propiedad que recibe un valor booleano y prohíbe la carga de archivos cuando se sobrepasa el límite definido en el atributo "limits".

- **responseOnLimit:** Su valor será un string que será devuelto al cliente cuando se sobrepase el peso del archivo que se esté intentando cargar.

Si deseas conocer todas las propiedades que acepta este objeto de configuración, revisa el documento **Material Apoyo Lectura - Configuración del paquete express-fileupload** ubicado en "Material Complementario". Dentro de este encontrarás todas las propiedades que puedes definir en el archivo de configuración y tendrás una idea clara del alcance funcional de este paquete.

Ejercicio guiado: Subiendo una imagen al servidor

Construir un servidor que integre el paquete express-fileupload y disponibilice una ruta **POST** para alojar imágenes en un directorio local con un peso máximo de 5 MB. La carga de archivos se deberá hacer a través de un formulario HTML que contenga lo siguiente:

- Un input que tenga definido en su atributo type el valor "file", esto permitirá que el usuario pueda seleccionar un archivo desde el navegador.

- Un botón que al ser presionado dispara la acción del formulario enviando el archivo a nuestro servidor.

Para iniciar el ejercicio y considerando que tenemos nuestro paquete instalado necesitamos un servidor base con Express, para esto ocupa el siguiente código:

```
const express = require('express');
const app = express();
app.listen(3000);
```

Con nuestro servidor creado y el paquete instalado, sigue los pasos para realizar este ejercicio progresivo en donde integraremos el paquete express-fileupload en nuestro servidor con express:

- ***Paso 1:*** Guardar en una constante de nombre "expressFileUpload" la dependencia "express-fileupload".

```
const expressFileUpload = require('express-fileupload');
```

- ***Paso 2:*** Al igual que Handlebars, los paquetes creados para ser usados con Express se integran a través del método "use" de nuestra constante "app". El cual recordemos, es usado para la creación de middlewares que nuestro servidor reconocerá al ser levantado.

  Para terminar la integración definamos el objeto de configuración con las siguientes propiedades y valores:

  - **limits:** 5 millones (5.000.000) de Bytes, es decir 5MB.
  - **abortOnLimit:** Lo definiremos en "true" para abortar el proceso en caso de que se sobrepase el límite
  - **responseOnLimit:** Un mensaje que diga "El peso del archivo que intentas subir supera el límite permitido"

Nuestro middleware quedaría de la siguiente manera:

```
app.use( expressFileUpload({
   limits: { fileSize: 5000000 },
   abortOnLimit: true,
   responseOnLimit: "El peso del archivo que intentas subir supera el
limite permitido",
 })
);
```

- ***Paso 3:*** Crear una ruta get para la devolución de un formulario que permita la selección de un archivo y declare en su atributo "method", que enviará su contenido ocupando el método POST.

```
app.get("/", (req, res) => {
  res.send(`
    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="foto" required>
        <button> Upload </button>
   </form>
    `);
});
```

Hay un par de consideraciones importante que debes tener en cuenta con este formulario:

- El input de tipo "file" debe incluir también el atributo "name" para poder ser identificado por el servidor. En nuestro caso definamos como nombre la palabra "foto" puesto que estrenaremos este plugin subiendo una foto al servidor.

- En la etiqueta form, debe declarar un atributo "enctype" cuyo valor debe ser "multipart/form-data". Esta declaración permitirá que nuestro formulario pueda enviar archivos y que estos sean reconocidos por el servidor.

Ahora, abre el navegador y consulta tu servidor, deberás encontrar el formulario HTML que devolvemos en la ruta raíz. En este deberás seleccionar cualquier archivo de tu sistema operativo, por ejemplo, una foto en formato png.

Haciendo lo anterior deberás tener algo como lo que te muestro en la siguiente imagen:

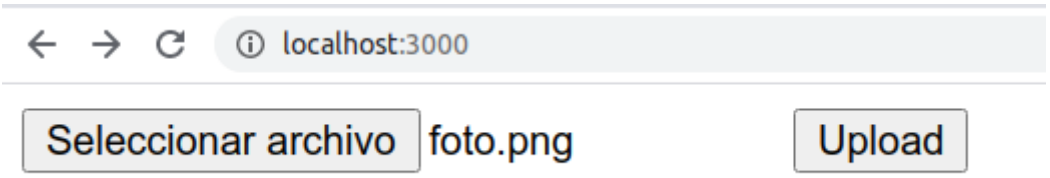


Imagen 1. Muestra del navegador con los inputs para la carga de archivos.
Fuente: Desafío Latam

- ***Paso 4:*** Crear una ruta POST que:

    - A través de un destructuring extraiga la propiedad "foto" de la "files" ubicada dentro del objeto request almacenando su valor en una constante. Lo que hacemos con esto, es guardar en una variable la instancia de la foto que fue

subida en el formulario y que está siendo recibida a través de la consulta HTTP.

- Crear una constante y ocupar el destructuring para extraer propiedad “name” de la constante creada en el punto anterior. Esta propiedad almacenará el nombre y formato(extensión) de la foto que estamos recibiendo.
- Utilizar el método “mv” de la constante “foto” y definir los siguientes parámetros:
  - **Primer parámetro:** Este parámetro declara la ruta en donde almacenaremos el archivo, lo correcto será tener una carpeta particularmente dedicada a almacenar los archivos que vamos a recibir en nuestro servidor, para esto crea una carpeta con nombre “archivos”. Con la carpeta creada, podemos definir este parámetro concatenando el nombre de la carpeta y del archivo, el cual recordemos está almacenado en la constante “name”.
  - **Segundo parámetro:** Es una función callback que recibe como parámetro el posible error de este proceso.

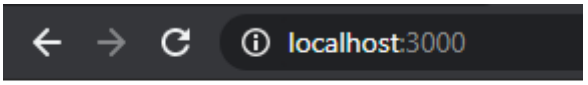
    En su bloque de código utiliza el método “send” para responderle al usuario un mensaje indicando que el archivo fue cargado con éxito. La ruta POST quedaría de la siguiente manera:

```
app.post("/", (req, res) => {
  const { foto } = req.files;
  const { name } = foto;
  foto.mv(`${__dirname}/archivos/${name}`, (err) => {
    res.send("Archivo cargado con éxito");
  });
});
```

Como puedes observar, estamos usando algunas propiedades nuevas como el req.files. Esta propiedad no pertenece por defecto al objeto request de Express, pero la podemos ocupar gracias a la integración del paquete express-fileupload en nuestro servidor.

El método “mv” es otra añadidura de nuestro paquete y sirve para mover el archivo hacia el path declarado en el primer parámetro. Para conocer todas las propiedades que nos ofrece el paquete express-fileupload, revisa el documento **Material Apoyo Lectura - Manipulación de atributos**, ubicado en “Material Complementario”.

Es momento de probar nuestro desarrollo y estrenar el paquete express-fileupload. Consulta tu servidor desde el navegador e intenta subir un archivo, deberás obtener la respuesta que te muestro en la siguiente imagen:



Archivo cargado con éxito

Imagen 2. Respuesta en el navegador luego de subir un archivo.
Fuente: Desafío Latam

Excelente, esto quiere decir que el servidor procesó el archivo y lo guardó en la carpeta "archivos", pero ¿Cómo podemos realmente comprobar que esto sucedió? Dirígete a tu árbol de archivos y deberás ver la foto que cargaste desde el formulario html, tal y como te muestro en la siguiente imagen:

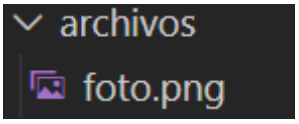
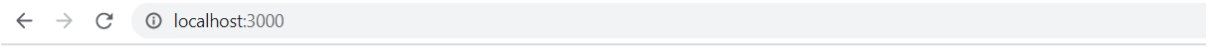


Imagen 3. Comprobación de que el archivo fue alojado con éxito en el servidor.
Fuente: Desafío Latam

Ahora solo falta probar el límite que definimos en la configuración de nuestro paquete, para ello, intenta subir un archivo que exceda los 5MB y deberás obtener como respuesta lo que te muestro en la siguiente imagen:



El peso del archivo que intentas subir supera el limite permitido

Imagen 4. Respuesta del servidor al recibir un archivo que excede el peso definido.
Fuente: Desafío Latam

Ejercicio propuesto (1)

Desarrollar un servidor que permita la carga de archivos usando el paquete express-fileupload y los almacene en una carpeta de nombre "uploads", además de tener definido un límite de 2 MB por archivo.

Postman y body-parser

Competencias

- Probar el despliegue de un proyecto REST para Upload File utilizando el utilitario POSTMAN.
- Integrar el paquete body-parser para la manipulación del payload enviado desde una aplicación cliente.

Introducción

Cuando desarrollamos nos encontramos constantemente con la necesidad de saber si estamos cumpliendo con los objetivos previstos. Esto se consigue fácilmente en el frontend con un debugger o imprimiendo los valores de interés por consola. En el caso del backend y específicamente en el desarrollo de servidores, necesitamos realizar consultas HTTP para probar el correcto funcionamiento de nuestras rutas.

POSTMAN ha sido de gran utilidad a lo largo de la carrera, gracias a su cómodo uso, facilidad para simular una aplicación cliente y probar nuestras API REST, sin embargo, siempre lo hemos usado para consultas que contienen datos en formato JSON. Ahora nos encontramos con la necesidad de probar la funcionalidad Upload file de nuestro servidor, ¿Es posible que POSTMAN nos ayude también a probar esta funcionalidad? La respuesta es sí, y en este capítulo aprenderás cómo hacerlo.

Además, aprenderás a integrar un paquete que comúnmente vemos en proyectos como fiel acompañante de Express, se trata de body-parser, un middleware que formateara el payload recibido en una consulta HTTP a formato JSON, ¿En dónde estarán estos datos? Podremos acceder a ellos por medio de una propiedad “body” que se integrará dentro del objeto “request”. Con estos conocimientos podrás agilizar el desarrollo de tus servidores que sirvan la funcionalidad File Upload, además de manipular metadatos de la misma consulta y lograr procesar peticiones más complejas.

Upload file al servidor desde POSTMAN

Para agilizar el desarrollo de una API REST podemos utilizar POSTMAN y hacer pruebas simulando una aplicación cliente. Esto ya lo hemos hecho en sesiones anteriores aunque nunca bajo una temática de archivos.

Ahora que ocupamos el paquete express-fileupload, podemos hacer simulaciones de tipo Upload File desde POSTMAN. Probablemente esto te suene un poco raro, pero nuestra herramienta de simulación de consultas no solo es utilizada para obtener o enviar datos en formato JSON, también nos ofrece la posibilidad de enviar archivos emulando un formulario HTML y en el siguiente ejercicio guiado te mostraré cómo esto es posible.

Ejercicio guiado: Haciendo Upload File con POSTMAN

Probar el despliegue del servidor creado en el capítulo anterior, utilizando el utilitario POSTMAN para hacer una consulta POST que simule el comportamiento de un formulario HTML. El objetivo será probar que el archivo que enviemos se esté alojando en la carpeta “archivos”.

Para iniciar con este ejercicio, abre POSTMAN y prepara una consulta POST al servidor en su ruta raíz tal y como se ve en la siguiente imagen:

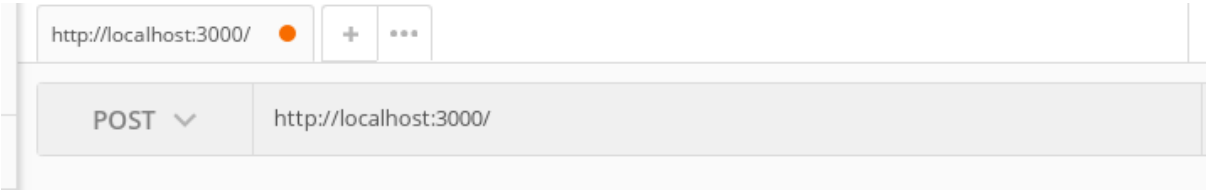


Imagen 5. Preparando consulta POST en POSTMAN para hacer upload file.
Fuente: Desafío Latam

Ahora sigue las siguientes instrucciones:

1. Para simular un formulario HTML selecciona en la pestaña body la opción de “form-data”.
2. Para adjuntar una imagen, debemos escribir como “key” el nombre que representará a nuestro archivo en la consulta, esto equivale a la declaración del atributo name en los input de HTML.
3. Usa el desplegable de la “key” y selecciona “File” en vez de “Text”, verás que aparecerá el botón para adjuntar un archivo en la columna “Value”.
4. Haz click en el botón y selecciona el archivo en tu computador.
5. Realiza la consulta y recibirás como respuesta lo predefinido en el servidor.

En la siguiente imagen te muestro el resultado de los 5 pasos. Observa que la respuesta que recibiremos es la misma que obtuvimos con el navegador y es señal de que todo salió bien.

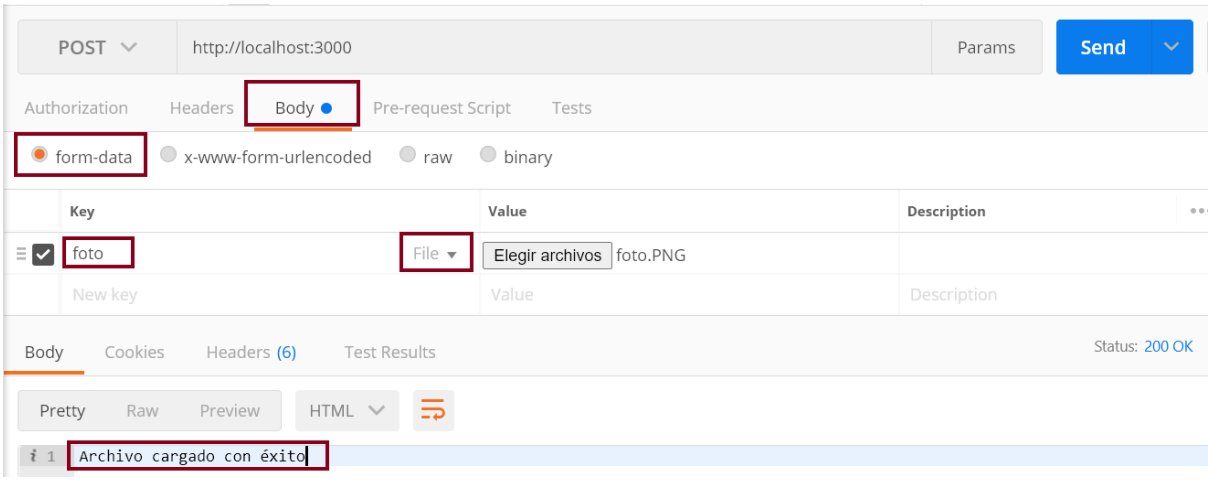


Imagen 6. Envío de una imagen al servidor con POSTMAN.
Fuente: Desafío Latam

Excelente, ahora sabemos cómo probar con POSTMAN nuestro servidor y el paquete express-fileupload para hacer Upload File, sin embargo, en la práctica real es muy poco común que los archivos que subimos no estén acompañados de otros metadatos, es decir, solo la subida del archivo no dice demasiado.

Entendiendo que estamos usando una consulta POST para enviar archivos, podemos aprovechar este método para enviar más datos, pero hay un detalle importante a considerar y es que nuestro servidor no está programado para recibir un payload enviado desde una aplicación cliente. En sesiones anteriores aprendimos a hacer esto con Node puro, ahora con Express notarás que podemos conseguir el mismo resultado rápidamente integrando el paquete body-parser que veremos a continuación:

El paquete body-parser

Es un paquete NPM muy popular que podemos instalar e integrar a nuestro servidor con Express, que nos permitirá recibir y manipular el payload enviado desde una aplicación cliente. Su función es formatear automáticamente el JSON que recibimos en una consulta a un objeto manipulable y disponibilizar una propiedad “body” dentro del objeto request.

Para instalar este paquete debemos utilizar la siguiente línea de comando por la terminal:

```
npm i --save body-parser
```

Una vez instalado, podemos integrarlo como un middleware en nuestro servidor, tal y como te muestro en el siguiente código:

```
const bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

Las propiedades “urlencoded” y “extended” sirven para declarar una versión extendida de este middleware. En caso de estar activada reconocerá y procesará otros tipos de datos, sin embargo, para nuestro ejercicio y la mayoría de las prácticas laborales no es necesario activarlo, porque normalmente manejamos, enviamos y recibimos datos en formato JSON. En base a lo anterior, definimos esta propiedad con el valor “false” y ejecutamos el método json() de la instancia bodyParser para el reconocimiento y procesamiento de este formato.

Ejercicio guiado: Más que solo un archivo

Desarrollar un servidor que aloje archivos mp3 en una carpeta llamada “canciones” y defina 10 MB como peso máximo para la carga de archivos, cuyos nombres serán definidos por diferentes inputs de un formulario HTML. Se debe utilizar el paquete body-parser para manipular el payload en formato JSON enviado desde el cliente, el cual contendrá específicamente los campos: nombre, artista y álbum. Finalmente estos atributos deberán ser concatenados en un mismo String para ser asignados como nombre del archivo mp3. Sigue los siguientes pasos para realizar este ejercicio guiado:

- ***Paso 1:*** Crear un servidor con Express e importar los paquetes express-fileupload y body-parser.

- ***Paso 2:*** Integrar el paquete body parser usando el método “use” de la constante “app”.

- ***Paso 3:*** Integrar el paquete express-fileupload definiendo 5MB como límite del peso de las canciones. Agrega un mensaje que indique que el límite fue superado.

- ***Paso 4:*** Crear una ruta **GET /** que devuelva un formulario HTML con 4 inputs para el ingreso de: nombre de la canción, artista y álbum, junto al archivo mp3 correspondiente a la canción.

- ***Paso 5:*** Crear una ruta **POST /** que almacene el archivo recibido dentro de una carpeta “canciones”, con un nombre compuesto por la concatenación de los 3 campos tipo texto recibidos, el formato para esta concatenación debe ser el siguiente: **<nombre de la canción> - <Nombre del artista> (<Nombre del álbum>)**. Por ejemplo “De música ligera - Soda Stereo (Canción Animal)”.

```
// Paso 1
const express = require("express");
const app = express();
const expressFileUpload = require("express-fileupload");
const bodyParser = require("body-parser");
app.listen(3000);
// Paso 2
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

// Paso 3
app.use(
  expressFileUpload({
    limits: { fileSize: 10000000 },
    abortOnLimit: true,
    responseOnLimit:
      "El peso de la cancion que intentas subir supera el limite
permitido",
  })
);

// Paso 4
app.get("/", (req, res) => {
  res.send(`
    <form method="POST" action="" enctype="multipart/form-data">
        <input type="text" name="nombre" required placeholder="Nombre">
        <input type="text" name="artista" required
placeholder="Artista">
        <input type="text" name="album" required placeholder="Album">
        <input type="file" name="cancion" required>
        <button> Upload! </button>
   </form>
    `);
});

// Paso 5
app.post("/", (req, res) => {
  const { cancion } = req.files;
  const { nombre, artista, album } = req.body;
  const name = `${nombre} - ${artista} (${album})`;
  cancion.mv(`${__dirname}/canciones/${name}.mp3`, (err) => {
    res.send("Archivo cargado con éxito");
  });
});
```

Ahora consulta al servidor desde el navegador e ingresa los campos del formulario, como puedes ver en la siguiente imagen de referencia:

Imagen 7. Árbol de archivos para ejercicio final de lectura.

Presiona el botón para enviar el formulario, deberás recibir el mensaje “Archivo cargado con éxito por el navegador” al igual que el ejercicio anterior y esto es excelente, porque significa que el servidor no tuvo problemas en almacenar el archivo. Podemos comprobarlo revisando la carpeta “canciones”, en donde deberás ver el mp3 que subiste en el Formulario HTML, tal como te muestro en la siguiente imagen:

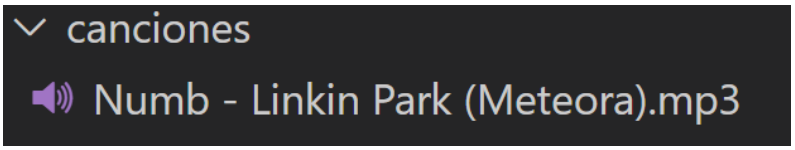


Imagen 8. Árbol de archivos para ejercicio final de lectura.
Fuente: Desafío Latam

Ejercicio propuesto (2)

Desarrollar un servidor que permita la carga de imágenes cuyo nombre y extensión sea definido por inputs de un formulario HTML. El peso máximo de imágenes permitido debe ser 3 MB.

Eliminando archivos

Competencia

- Desarrollar una ruta para eliminar los archivos cargados con el paquete express-fileupload.

Introducción

Cuando desarrollamos un servidor con la funcionalidad de Upload File, tarde o temprano aparece la necesidad no solo de seguir guardando nuevos archivos, sino también de eliminarlos. Para esto podemos ocupar el módulo File System y su método “unlink”, que nos sirve para eliminar un archivo cuya ubicación es declarada como argumento de este método.

En este capítulo práctico, realizaremos una pequeña galería de imágenes que será importada en el HTML, usando la misma carpeta en la que estamos almacenando archivos, gracias al método “static” de Express. El objetivo final será poder eliminar las imágenes haciendo click sobre ellas, una vez finalizado el ejercicio estarás listo para empezar a desarrollar sistemas que permitan no solo el envio y recibo de texto, sino también la carga, consumo y eliminación de imágenes y/o archivos alojados en el servidor con el paquete express-fileupload.

Eliminando archivos del servidor

Con el paquete express-fileupload hemos podido cargar archivos a nuestro servidor, pero ¿Qué sucede si necesitamos eliminarlos? Para esto podemos ocupar el módulo File System y su método “unlink”, especificando la dirección y el archivo que queremos eliminar.

Ejercicio guiado: Eliminando archivos

Construir un servidor que devuelva en su ruta raíz una aplicación cliente con 3 imágenes, que al ser presionadas emitan una consulta **DELETE** a la ruta **/imagen/:nombre**, siendo el parámetro “nombre” equivalente al id de cada etiqueta “img”. La aplicación cliente y las imágenes de este ejercicio las podrás encontrar en el **Apoyo Lectura - Eliminando archivos**, ubicado entre los archivos de esta sesión.

Crear una carpeta “public” y arrastrar dentro la carpeta “imágenes” que encontrarás también en el apoyo lectura.

Sigue los siguientes pasos para la solución de este ejercicio:

- ***Paso 1:*** Crear un servidor con Express e importar el módulo File System.
- ***Paso 2:*** Publicar la carpeta “public” usando los métodos “static” y “use”.
- ***Paso 3:*** Crear una ruta **GET /** que devuelva el documento “index.html” del apoyo lectura.
- ***Paso 4:*** Crear una ruta **DELETE /imagen/:nombre** que reciba el nombre de una imagen como parámetro y la elimine usando el método “unlink” de File System. Es importante que sepas que este método, es independiente de la publicación de un directorio local.

```
// Paso 1
const express = require("express");
const app = express();
const fs = require("fs");
app.listen(3000);

// Paso 2
app.use(express.static("public"));

// Paso 3
app.get("/", (req, res) => {
  res.sendFile(__dirname + "/index.html");
});

// Paso 4
app.delete("/imagen/:nombre", (req, res) => {
  const { nombre } = req.params;
  fs.unlink(`${__dirname}/public/imagenes/${nombre}.jpg`, (err) => {
    res.send(`Imagen ${nombre} fue eliminada con éxito`);
  });
});
```

Ahora consulta el servidor con el navegador, deberás ver lo que te muestro en la siguiente imagen:

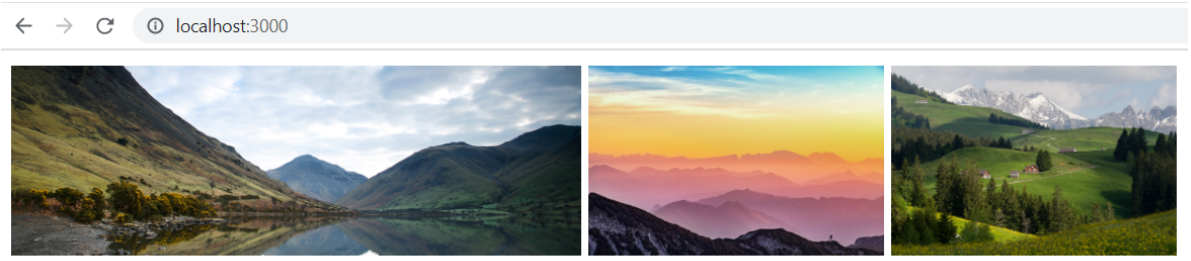


Imagen 9. Aplicación cliente disponibilizada en el Apoyo Lectura.
Fuente: Desafío Latam

Lo que vemos es la aplicación cliente mostrando las 3 imágenes que se encuentran en la dirección http://localhost:3000/. Presiona clic en la primera imagen y deberás ver lo que te muestro a continuación:

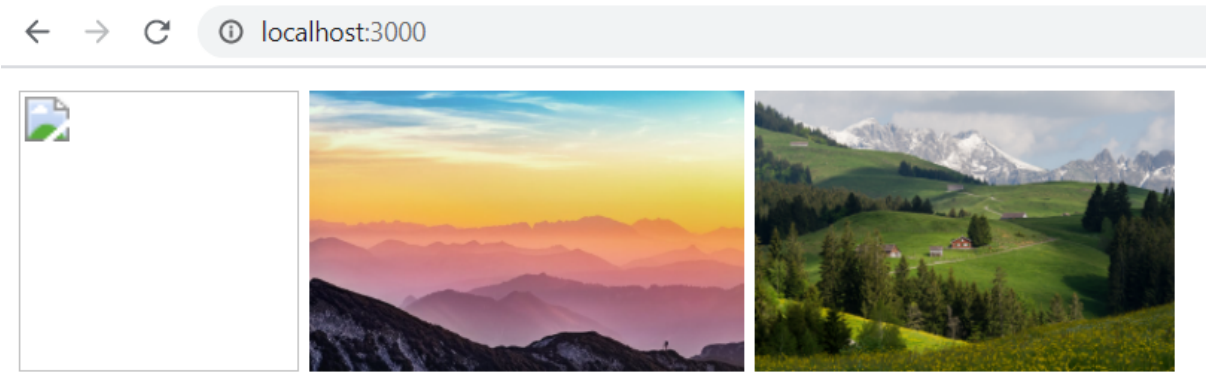


Imagen 10. Imagen eliminada del servidor.
Fuente: Desafío Latam

¡Excelente! la imagen fue eliminada del servidor y por eso dejó de mostrarse en la aplicación cliente.

Ejercicio propuesto (3)

Debate con tus compañeros la siguiente pregunta: **¿Cómo podemos unir los ejercicios de los 3 capítulos en una misma aplicación?**

Ejercicio propuesto (4)

Basado en el ejercicio guiado "Eliminando archivos", modifica la ruta **DELETE** para cambiar el nombre del parámetro "nombre" por "id", además de responder el mensaje "Lo siento, este archivo no existe en servidor" en caso de ocurrir un error en el intento de eliminar un archivo.

Resumen

En esta lectura revisamos cómo integrar la funcionalidad Upload File con el paquete express-fileupload. Aprendimos a probar nuestras rutas con el utilitario POSTMAN y finalizamos con un ejercicio guiado en donde repasamos cómo eliminar archivos ocupando el módulo File System pero ahora en un entorno express.

En síntesis hemos abordado los siguientes contenidos:

- Qué es express-fileupload y cómo integrarlo a un desarrollo con express para servir la funcionalidad Upload File.
- A través de un ejercicio guiado aprendimos a usar POSTMAN para hacer consultas POST, que incluyan archivos en su cuerpo simulando el comportamiento de un formulario HTML.
- Qué es body-parser y cómo integrarlo para formatear el payload de una petición emitida por una aplicación cliente a JSON.
- Cómo eliminar archivos con el método “unlink” pero ahora dentro de un entorno express.

Solución de los ejercicios propuesto

1. Desarrollar un servidor que permita la carga de videos recibidos por medio de un formulario HTML cuyo input está declarado con el nombre “video”. Deberás usar el paquete express-fileupload y desarrollar la lógica que almacene los archivos en una carpeta llamada “uploads”, además de tener definido un límite de 2 MB por archivo.

```
const express = require("express");
const app = express();
app.listen(3000);
const expressFileUpload = require("express-fileupload");

app.use(
  expressFileUpload({
    limits: { fileSize: 2000000 },
    abortOnLimit: true,
    responseOnLimit:
      "El peso del archivo que intentas subir supera el limite
permitido",
  })
);

app.get("/", (req, res) => {
  res.send(`
    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="video" required>
        <button> Upload </button>
   </form>
    `);
});

app.post("/", (req, res) => {
  let video = req.files.video;
  const { name } = video;
  video.mv(`${__dirname}/uploads/${name}`, (err) => {
    res.send("Archivo cargado con éxito");
  });
});
```

2. Desarrollar un servidor que permita la carga de imágenes cuyo nombre y extensión sea definido por inputs de un formulario HTML. El peso máximo de imágenes permitido debe ser 3 MB.

```
const express = require("express");
const app = express();
const expressFileUpload = require("express-fileupload");
const bodyParser = require("body-parser");

app.listen(3000);

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

app.use(
  expressFileUpload({
    limits: { fileSize: 3000000 },
    abortOnLimit: true,
    responseOnLimit:
      "El peso de la imagen que intentas subir supera el limite
permitido",
  })
);

app.get("/", (req, res) => {
  res.send(`
    <form method="POST" action="" enctype="multipart/form-data">
       <input type="text" name="nombre" required placeholder="Nombre">
       <input type="text" name="extension" required
placeholder="extension">
       <input type="file" name="imagen" required>
       <button> Subir imagen </button>
   </form>
    `);
});

app.post("/", (req, res) => {
  const { imagen } = req.files;
  const { nombre, extension } = req.body;
  imagen.mv(`${__dirname}/imagenes/${nombre}.${extension}`, (err) => {
    res.send("Imagen cargada con éxito");
  });
});
```

3. Debate con tus compañeros la siguiente pregunta: ¿Cómo podemos unir los ejercicios de los 3 capítulos en una misma aplicación?

   Creando un servidor con express que utilice el paquete express-fileupload y body-parser, devolviendo una aplicación cliente con un formulario para cargar archivos a un directorio local y disponibilizando una ruta DELETE que reciba como parámetro el nombre de un archivo y lo elimine con el método "unlink" de File System.

4. Basado en el ejercicio guiado "Eliminando archivos", modifica la ruta **DELETE** para cambiar el nombre del parámetro "nombre" por "id", además responder el mensaje "Lo siento, este archivo no existe en servidor" en caso de ocurrir un error en el intento de eliminar un archivo.

```
app.delete("/imagen/:id", (req, res) => {
  const { id } = req.params;
  fs.unlink(`${__dirname}/public/imagenes/${id}.jpg`, (err) => {
    err
      ? res.send("Lo siento, este archivo no existe  en servidor")
      : res.send(`Imagen ${id} fue eliminada con éxito`);
  });
});
```