Subida de archivos al servidor

Subida de archivos al servidor

{desafío} latam_

Implementar la funcionalidad de subida de archivos a un servidor utilizando Express de acuerdo al entorno Node.js.

- Unidad 1:
  API REST
- Unidad 2:
  Subida de archivos al servidor
- Unidad 3:
  JWT

Te encuentras aquí

{desafío}
latam_

¿Qué aprenderás en esta sesión?

- *Reconoce los paquetes requeridos para la implementación de upload en un servidor Express.*

- *Implementa funcionalidad de subida de archivos en un servidor Express.*

{desafío} latam_

¿Porqué es importante probar las funcionalidades de nuestras aplicaciones?

{desafío} latam_

/* ¿Qué es el testing? */

¿Qué es el testing?

El testing en pocas palabras se puede definir como el proceso de probar que algo que se supone que cumple un objetivo, efectivamente cumpla con este. Es un universo paralelo al mundo del software en el que hacemos aplicaciones que evalúan la efectividad de otras aplicaciones.

En Javascript, contamos con varias herramientas para hacer tests, de las más populares tenemos JEST y Mocha js.

/* Mocha js */

Mocha js

Mocha js, es un framework de Javascript para hacer pruebas unitarias, muy conocido y utilizado en la industria actualmente, con más de 4 millones de descargas semanalmente en su paquete de NPM. Las pruebas de Mocha js se ejecutan en serie, lo que permite informes flexibles y precisos, al tiempo que asigna las excepciones no detectadas a los casos de prueba correctas.

Fuente: Mocha Js

Mocha js

Por sí solo, es capaz de hacer varias funcionalidades. Sin embargo, sus capacidades se amplían con la importación de 2 plugins en particular: Chai y Sinon.

En esta sesión estaremos usando Chai para hacer la consulta al servidor y obtener la data con el método GET. Si quieres saber más de Mocha js, puedes profundizar conocimientos en su sitio oficial.

/* Describe, it y expect, los métodos del testing */

Describe, it y expect, los métodos del testing

Entender testing es más sencillo luego que entiendes para qué sirven estos 3 métodos:

- **Describe**: Este método crea un bloque que agrupa varias pruebas relacionadas, se representa comúnmente como una "suite de pruebas"

- **It**: Esta es una prueba unitaria en sí misma. Por lo general, su argumento es un string que nombra al test unitario y esta será su descripción al momento de hacer la corrida de tests.

- **Expect:** La función expect se utiliza cada vez que desea probar un valor, por ejemplo “2 + 2” se espera que sea igual a 4. Es normalmente el último método en ejecutarse dentro de un test unitario y su argumento es el valor a evaluar. Mocha contempla una gran serie de propiedades descriptivas que indican que es lo que se está evaluando.

Ejemplo básico

Veamos como se ve un test básico, pero antes, deberás seguir una serie de pasos. Primero instalar mocha js y chai, para eso usa el siguiente comando:

```
npm i mocha chai --save
```

Ahora debes asegurarte que en tu **package.json** esté escrito este script:

```
"scripts": {
    "test": "mocha"
  },
```

Ejemplo básico

Recordando que necesitaremos probar nuestra API REST y que esta es un servidor en sí, necesitas exportar el servidor como una variable, por lo que dirígete al código del servidor y guarda en una constante el servidor, deberás exportarla como módulo, como se muestra en la siguiente imagen:

```
1   const url = require("url");
2   const http = require("http");
3   const fs = require("fs");
4   const { v4: uuidv4 } = require("uuid");
5   const server = http
6   >   .createServer(function (req, res) {···
54      .listen(3000);
55
56  module.exports = server;
```

Y por último, deberás crear una carpeta llamada “test” y dentro un archivo llamado “test.js”.

Ejercicio guiado:

Mi primer testing

Ejercicio guiado

Mi primer testing

Desarrollar una aplicación en Node que use la librería Mocha para testear la sumatoria de 2 números, para esto sigue los siguientes pasos:

- **Paso 1**: Importar el paquete chai.

- **Paso 2**: Utilizar el método describe y pasa como primer parámetro la descripción general del test y como segundo parámetro será una función callbacks en donde escribiremos los tests.

Ejercicio guiado

Mi primer testing

- **Paso 3**: Utilizar el método it pasando como primer parámetro la descripción del test que probará la sumatoria de 2 números y como segundo parámetro será una función callbacks en donde escribiremos las instrucciones de este test unitario

- **Paso 4**: Crear 2 variables numéricas y una que sea igual a la suma de las dos anteriores.

- **Paso 5**: Utilizar el método expect que se encuentra en la instancia del paquete chai pasando como parámetro la variable de suma, seguidamente ocupa las propiedades “to” y “equal” de forma concatenada, siendo esta segunda un método al que le deberás pasar el valor que esperas que tenga la variable suma.

Ejercicio guiado

Mi primer testing

```
// Paso 1
const chai = require("chai");

// Paso 2
describe("Primer encuentro con el testing", function () {
  // Paso 3
  it("Probando una sumatoria", function () {
    // Paso 4
    let num1 = 2;
    let num2 = 2;
    let suma = num1 + num2;
    // Paso 5
    chai.expect(suma).to.equal(4);
  });
});
```

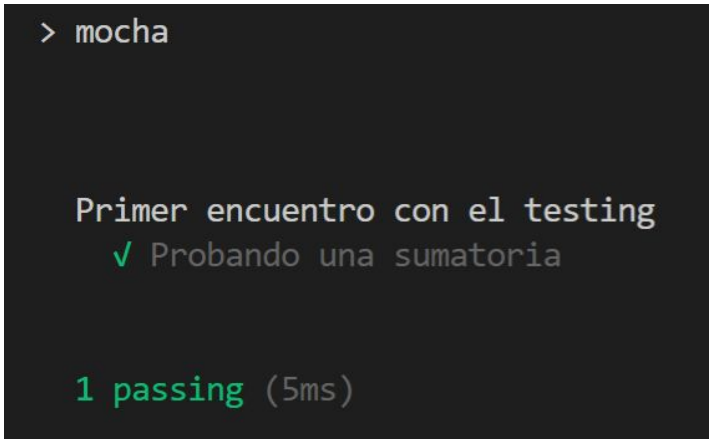
Ejercicio guiado

Mi primer testing

Ahora para correr este test debes utilizar el siguiente comando:

```
npm run test
```

Y recibirás lo que se muestra en la siguiente imagen por terminal:



Ejercicio propuesto

Ejercicio propuesto

Basado en el test para probar la sumatoria de 2 números, realiza un test de la multiplicación de 2 números.

/* Probando mi API REST */

Probando mi API REST

Ahora es momento de acercarnos al mundo real, donde no probamos algo que estamos 100% seguros que pasará, sino que evaluamos una funcionalidad de nuestra aplicación, en este caso nuestro servidor/API REST.

Para esto, haremos uso de los pasos que seguiste anteriormente pues usaremos el servidor como un módulo importado y usaremos “chai” con una extensión llamada “chai-http” que debes instalar usando el siguiente comando:

```
npm i chai-http
```

Probando mi API REST

Esta dependencia nos ayudará a simular una consulta HTTP a nuestro servidor.
¿Cómo es posible esto? Sigue estos pasos para incluir a nuestra aplicación el plugin de chai-http con el que podremos hacer test a nuestro servidor usando el protocolo HTTP:

- **Paso 1**: Importar el paquete chai-http.
- **Paso 2**: Importar el servidor del archivo index.js o del documento en el que hayas creado tu servidor.
- **Paso 3**: Instalar el plugin de chai-http usando un método llamado “use” de la instancia del paquete chai y como parámetro escribe la instancia de chai-http.

Probando mi API REST

- **Paso 4**: Crear una suite de test y un test unitario con las descripciones correspondientes a la prueba de nuestra API REST.

- **Paso 5**: Utilizar un método de chai llamado “request” pasándole como parámetro el servidor.

- **Paso 6**: Concatenarle al método anterior otro método llamado “get” en la que pasarás como parámetro la ruta que quieres testear del servidor, en este caso “/bicicletas”

Probando mi API REST

- **Paso 7**: Concatenarle al método anterior otro método llamado “end” en el que recibirás como parámetro una función callback que contiene el error y la data de la consulta al servidor

- **Paso 8**: Guardar en una variable la data de la respuesta en su propiedad “text” parseada con el JSON.parse()

- **Paso 9**: Utilizar el método expect de la instancia de chai pasándole como parámetro la data parseada, posteriormente concatenarle a esta instrucción las propiedades “to”, “have” y “property”, siendo esta última un método que recibe el nombre de la propiedad que esperas que contenga la data

Probando mi API REST

- **Paso 10:** Utilizar el método expect de la instancia de chai pasándole como parámetro la data parseada en su propiedad “bicicletas”, posteriormente concatenarle a esta instrucción las propiedades “to”, “be” y “an”, siendo esta última un método que recibe el nombre del tipo de dato que esperas que sea esa propiedad, en este caso queremos que sea un “array”

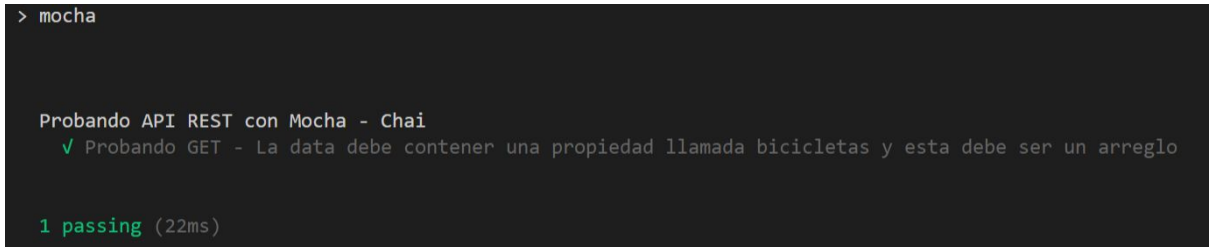
Probando mi API REST

```
const chai = require('chai')
// Paso 1
const chaiHttp = require('chai-http')
// Paso 2
const server = require('../index')
// Paso 3
chai.use(chaiHttp)
// Paso 4
describe('Probando API REST con Mocha - Chai', function () {
    it('Probando GET - La data debe contener una propiedad llamada bicicletas y esta debe ser un arreglo',
function () {
    // Paso 5
    chai
      .request(server)
    //   Paso 6
      .get('/bicicletas')
    //   Paso 7
      .end(function (err, res) {
        // Paso 8
        let data = JSON.parse(res.text)
        // Paso 9
        chai.expect(data).to.have.property('bicicletas')
        // Paso 10
        chai.expect(data.bicicletas).to.be.an('array')
      })
  })
})
```

{desafío} latam_

Probando mi API REST

Ahora corre el test y deberás recibir lo que muestra la siguiente imagen.



¡Excelente! Ha pasado nuestro test y con esto comprobaremos cada vez que queramos que nuestra API REST está devolviendo los datos de las bicicletas correctamente.

Si quieres profundizar más sobre la API de chai, en su sitio oficial encontrarás toda la información que necesites.

/* Gestión de archivos */

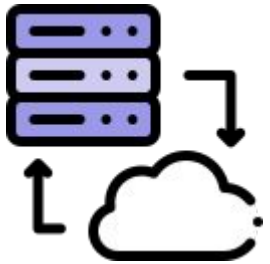
Gestión de archivos

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.

Gestión de archivos

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.

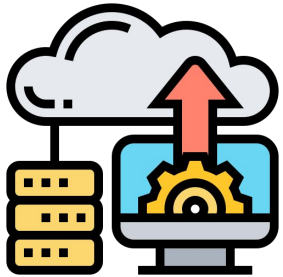


/* ¿Qué es express-fileupload? */

¿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”.



¿Qué es express-fileupload?

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
```

¿Qué es express-fileupload?

Integración

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 es express-fileupload?

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 sesión 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.

Ejercicio guiado:

Subiendo una imagen al servidor

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.

Ejercicio guiado

Subiendo una imagen al 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);
```

Ejercicio guiado

Subiendo una imagen al servidor

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');
```

{desafío} latam_

Ejercicio guiado

Subiendo una imagen al servidor

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”

Ejercicio guiado

Subiendo una imagen al servidor

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",
 })
);
```

{desafío} latam_

Ejercicio guiado

Subiendo una imagen al servidor

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>
    `);
});
```

Ejercicio guiado

Subiendo una imagen al servidor

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.

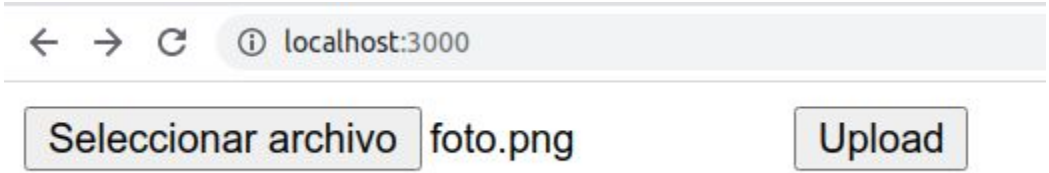
- En la etiqueta form, debe declarar un atributo “enctype” cuyo valor debe ser “multipart/form-data”. Esta declaración permitirá que nuestro formulario pueda enviar archivos y que estos sean reconocidos por el servidor.

Ejercicio guiado

Subiendo una imagen al 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:



Ejercicio guiado

Subiendo una imagen al servidor

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.

Ejercicio guiado

Subiendo una imagen al servidor

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.

Ejercicio guiado

Subiendo una imagen al servidor

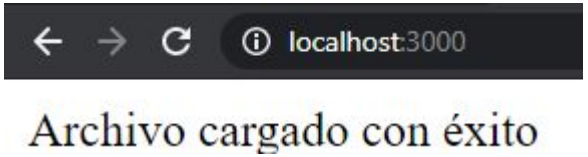
En su bloque de código utiliza el método “send” para responderle al usuario un mensaje indicando que el archivo fue cargado con éxito. La ruta POST quedaría de la siguiente manera:

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

Ejercicio guiado

Subiendo una imagen al servidor

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:

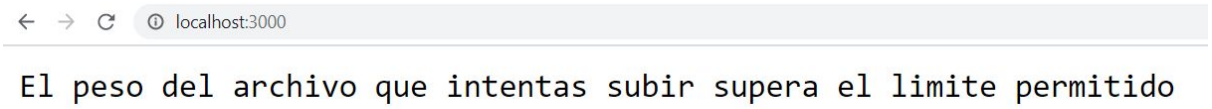


Excelente, esto quiere decir que el servidor procesó el archivo y lo guardó en la carpeta “archivos”, pero ¿Cómo podemos realmente comprobar que esto sucedió? Dirígete a tu árbol de archivos y deberás ver la foto que cargaste desde el formulario html, tal y como te muestro en la siguiente imagen:

Ejercicio guiado

Subiendo una imagen al servidor

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:



/* Upload file al servidor desde POSTMAN */

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 será primera vez que trabajamos con una temática de archivos.

Ahora que sabemos ocupar 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

{desafío} latam_

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:

Ejercicio guiado

Haciendo Upload File con POSTMAN

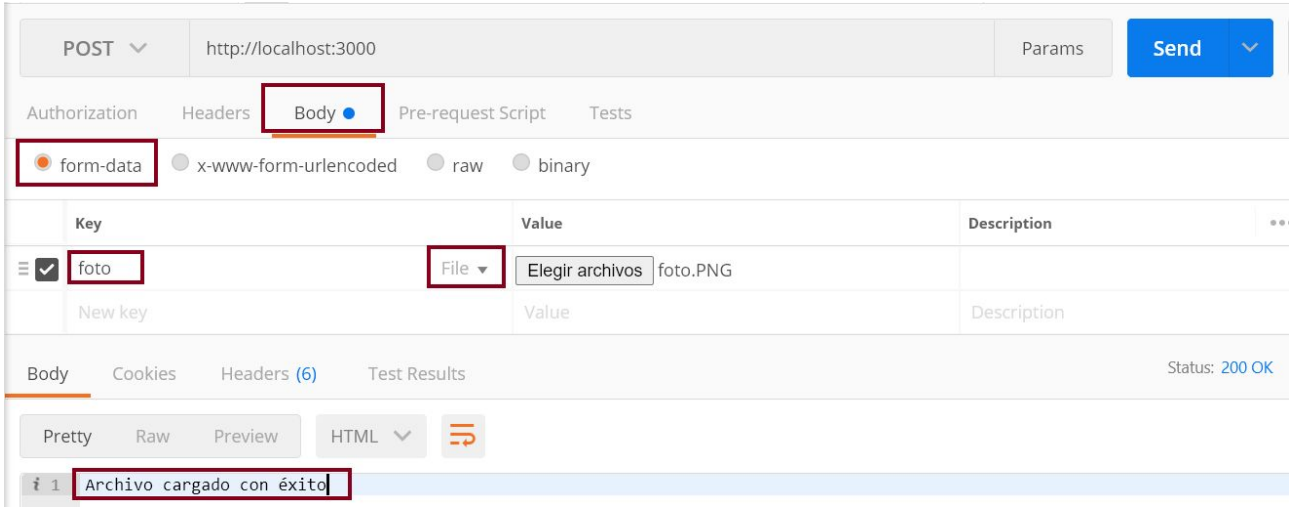
Ahora sigue las siguientes instrucciones:

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.

Ejercicio guiado

Haciendo Upload File con POSTMAN

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.



{desafío} latam_

Ejercicio guiado:

Más que solo un archivo

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 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.

Ejercicio guiado

Más que solo un archivo

Paso 1: Crear un servidor con Express e importar el paquete express-fileupload.

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

Ejercicio guiado

Más que solo un archivo

Paso 2: 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 2
app.use(
  expressFileUpload({
    limits: { fileSize: 10000000 },
    abortOnLimit: true,
    responseOnLimit:
      "El peso de la cancion que intentas subir supera el limite permitido",
  })
);
```

Ejercicio guiado

Más que solo un archivo

Paso 3: 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 3
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>
    `);
});
```

Ejercicio guiado

Más que solo un archivo

Paso 4: 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 4
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");
  });
});
```

{desafío} latam_

Ejercicio guiado

Más que solo un archivo

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

{desafío} latam_

Ejercicio guiado

Más que solo un archivo

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:

/* Eliminando archivos del servidor */

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

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 **Archivo Apoyo - Subida de archivos al servidor**, ubicado entre los archivos de esta sesión.

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

Ejercicio guiado

Eliminando archivos

- **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.

Ejercicio guiado

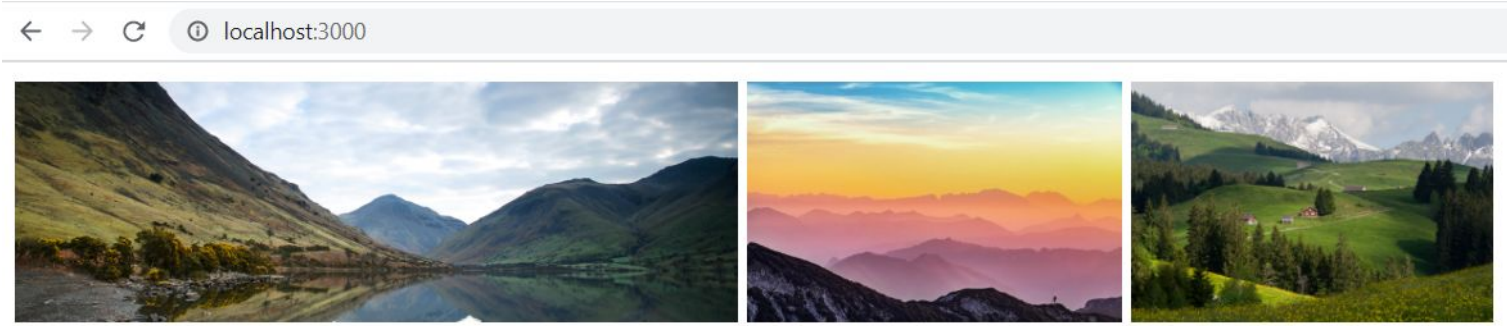
Eliminando archivos

```
// Paso 1
const express = require("express");
const app = express();
const fs = require("fs").promises;
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", async (req, res) => {
  const { nombre } = req.params;
  await fs.unlink(`${__dirname}/public/imagenes/${nombre}.jpg`);
  res.send(`Imagen ${nombre} fue eliminada con éxito`);
});
```

Ejercicio guiado

Eliminando archivos

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

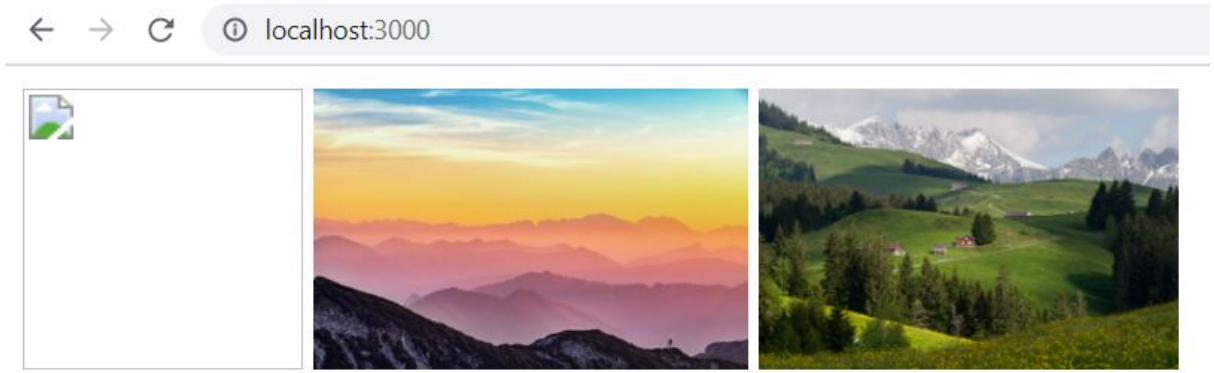


Ejercicio guiado

Eliminando archivos

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:



Resumen

- El testing en pocas palabras se puede definir como el proceso de probar que algo que se supone que cumple un objetivo, efectivamente cumpla con este.

- Mocha js, es un framework de Javascript para hacer pruebas unitarias.

- El testing dispone de 3 métodos:
  - Describe
  - it
  - expect

- Express-fileupload, es un plugin que sirve principalmente para dotar a nuestro servidor de la funcionalidad “Upload File”.

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales