Contenidos a Trabajar

1. MIDDLEWARES
2. MULTER

MIDDLEWARES

Las funciones de middleware son funciones que tienen acceso al objeto de solicitud (req), al objeto de respuesta (res) y a la siguiente función de middleware en el ciclo de solicitud/respuestas de la aplicación. La siguiente función de middleware se denota normalmente con una variable denominada next.

Las funciones de middleware pueden realizar las siguientes tareas:

- Ejecutar cualquier código.
- Realizar cambios en la solicitud y los objetos de respuesta.
- Finalizar el ciclo de solicitud/respuestas.
- Invocar la siguiente función de middleware en la pila.

Si la función de middleware actual no finaliza el ciclo de solicitud/respuestas, debe invocar next() para pasar el control a la siguiente función de middleware. De lo contrario, la solicitud quedará colgada.

Un Middleware tiene como propósito tomar dos piezas de la aplicación y conectarlas, como un puente en el que fluye la información. Normalmente decimos que una rutina de código tiene como propósito recibir información y retornarla transformada, la única característica especial de un Middleware es que la información la obtiene de otra función de código para luego enviársela a una función distinta más.

Los middlewares en Express se montan por múltiples razones, una de ellas por ejemplo es validar la información antes de que llegue a la rutina que enviará

respuesta hacia el cliente, también pueden usarse para hacer una consulta y guardar información antes de que pase a las funciones que responderán.

Un middleware en Express es una función, cuyo único distintivo es que recibe 3 argumentos:

```
function middleware(req,res,next){}
```

Los primeros dos argumentos, como cualquier función que responde respuestas del cliente contiene la información de la solicitud en el primer argumento Request, y el objeto Response como segundo argumento, que nos sirve para modificar la respuesta que se enviará al usuario.

El tercer argumento es muy importante, este es el que distingue un middleware de una función de respuesta. Este tercer argumento es una función que contiene el siguiente middleware o función a ejecutar luego de que el actual termine su ejecución.

Esta función next termina la ejecución de nuestro middleware y puede hacerlo de dos formas.

- Con éxito, la función next en este caso no recibe argumentos al ejecutarse, indicando al siguiente punto de la ejecución que todo salió bien.

- Con un error, el error se envía como argumento de la función, indicando al siguiente punto de la ejecución que algo salió mal y no puede continuar con la respuesta de la petición del cliente.

Todo salió bien:

```
function middleware(req,res,next){
   next();
}
```

Enviamos un error en el middleware

```
function middleware(req,res,next){
  if(user.permisos != "admin"){
          next(new Error('No tienes permisos para estar aquí'));
   }
}
```

Estas funciones se montan en el proceso de respuesta a una petición usando el método use del objeto **app**

```
const express = require('express');
const app = express();

function middleware(req,res,next){
   next();
}
app.use(next)  //Esto indica que antes de cualquier función de respuesta se debe ejecutar este middleware
```

O bien como parte de la respuesta de una ruta:

```
const express = require('express');
const app = express();

function middleware(req,res,next){
   next();
}

app.get('/', middleware, (req,res) => {
   /* Se ejecutará esta función luego del middleware */
});
```

En ambos casos, es posible que podamos colocar cuantos middlewares queramos definir, lo importante es que cada uno llame la función next, sin argumentos, para que el siguiente middleware se ejecute hasta llegar a la función de respuesta.

MULTER

Cuando un cliente web sube un archivo a un servidor, generalmente lo envía a través de un formulario y se codifica como `multipart/form-data`. Multer es un middleware para Express y Node.js que hace que sea fácil manipular este `multipart/form-data` cuando tus usuarios suben archivos.

¿Cómo funciona Multer?

Multer es un middleware para Express, este lee los archivos adjuntos provenientes de una request desde un formulario y antes de pasar al controlador, utilizamos Multer para guardar los archivos en el servidor.

A continuación instalamos la librería:

```
npm install multer
```

Crear el código del cliente

Necesitamos un archivo HTML con un formulario para poder enviar archivos desde el cliente.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
```

```
</head>
<body>
     <!--  Un solo archivo -->
        <form  action="/upload"  enctype="multipart/form-data"
method="POST">
    <input type="file" name="file" />
    <input type="submit" value="Subir un archivo"/>
    </form>


    <!-- Múltiples archivos -->

       <form  action="/upload"    enctype="multipart/form-data"
method="POST">
    Select images: <input type="files" name="files" multiple>
    <input type="submit" value="Subir archivos"/>
    </form>

</body>
</html>
```

Almacenamiento con Multer

Lo siguiente será definir una ubicación de almacenamiento para nuestros archivos. Multer ofrece la opción de almacenar archivos en el disco, como se muestra a continuación. Aquí, configuramos un directorio donde se guardarán todos nuestros archivos, y también les daremos a los archivos un nuevo identificador.

```
// Storage de archivos

var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, 'uploads')
  },
  filename: function (req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now())
  }
})

var upload = multer({ storage: storage })
```

Manejo de carga de archivos

Subiendo un solo archivo

En el archivo index.html, definimos un atributo de acción que realiza una petición POST. Ahora necesitamos crear un punto final en la aplicación Express. Como respuesta a la ruta encargada de recibir el archivo, realizamos lo siguiente:

```
app.post('/upload, upload.single('file'), (req, res, next) =>{
  const file = req.file
  if (!file) {
    const error = new Error('Please upload a file')
    error.httpStatusCode = 400
    return next(error)
  }
    res.send(file)
});
```

En el caso de **upload.single('file')** es el valor del atributo name del input donde cargamos el archivo en el HTML.

Subiendo múltiples archivos

Cargar varios archivos con Multer es similar a cargar un solo archivo, pero con algunos cambios.

```
// Subiendo más de 1 archivo a la vez
app.post('/upload, upload.array('files', 12), (req, res, next)
=> {
  const files = req.files
  if (!files) {
    const error = new Error('Please choose files')
    error.httpStatusCode = 400
    return next(error)
  }

    res.send(files)
});
```

En esta ocasión subimos usamos upload.array donde el primer argumento es el valor del atributo name del input y como segundo el número de archivos subidos.