Pila de Middleware (Práctica)

Tiempo estimado: 25min

El objeto de esta práctica es asentar y consolidar los conceptos presentados en la parte teórica de la lección.

Al finalizarla, el estudiante:

- Habrá escrito funciones de *middleware*.
- Habrá registrado funciones de *middleware* en la pila de *middleware*.
- Habrá trabajado con funciones de *middleware* normales y de control de errores.

Objetivos

En esta práctica, vamos a crear una aplicación Express que muestre, por un lado, cómo registrar funciones de *middleware* en una aplicación Express y, por otro lado, cómo trabajar con la función de continuación de flujo next().

Creación del proyecto

Para comenzar, vamos a crear el proyecto de la aplicación.

1. Abrir una consola.
2. Crear el directorio de la práctica.
3. Ir al directorio de la práctica.
4. Crear el archivo package.json con el siguiente contenido:

```
{
  "name": "express-app",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "start": "node ./app.js"
  },
  "dependencies": {
    "express": "*"
  }
}
```

5. Crear el archivo app.js con el siguiente contenido:

```
"use strict";

//imports
const express = require("express");
const http = require("http");

//app
const app = express();
const index = "<!doctype html>\n" +
              "<html>\n" +
              "<head><title>Express app</title></head>\n" +
              "<body><p>¡Hola Mundo!</p></body>\n" +
              "</html>";

//config app
app.get("*", function(req, res) { res.send(index); });
```



```
//listen
http.createServer(app).listen(8080, function() {
  console.log("Listening...");
});
```

6. Instalar dependencias:

```
> npm install
```

7. Iniciar la aplicación Express:

```
> npm start
```

8. Abrir el navegador.
9. Ir a http://localhost:8080.

   Debe aparecer el mensaje ¡Hola Mundo!

Registro de funciones normales de middleware

En este punto, vamos a jugar un poco con las funciones normales de *middleware*, recordemos, aquellas que se ejecutan siempre que todo va bien. La idea es crear tres funciones. La primera mostrará por la consola información sobre la petición HTTP bajo procesamiento; la segunda añadirá el texto Hello a la respuesta HTTP; mientras que la tercera World! Con esto en mente, observe que varias funciones pueden participar en la generación de la respuesta. No siendo necesario que toda la respuesta la genere la misma función. Y además, si es necesario, una función no tiene por qué hacer nada sobre la respuesta.

1. Editar el archivo app.js.
2. Suprimir la definición de la constante index.
3. Suprimir lo siguiente:

```
//config app
app.get("*", function(req, res) { res.send(index); });
```

4. Añadir las siguientes funciones de *middleware* :

```
//logger
app.use(function(req, res, next) {
  console.log(req.ip, req.method, req.originalUrl);
  next();
});

//genera Hello
app.use(function(req, res, next) {
  console.log("genera Hello");
  res.setHeader("Content-Type", "text/plain");
  res.write("Hello");
  next();
});

//genera World!
app.use(function(req, res, next) {
  console.log("genera World!");
  res.write("World!");
  res.end();
  next();
});
```

5. Guardar cambios.
6. Ir a la consola.
7. Detener la aplicación.
8. Arrancarla de nuevo:

```
>  npm start
```

9. Ir al navegador.
10. Solicitar de nuevo http://localhost:8080.

Debe aparecer el mensaje HelloWorld!

11. Ir a la consola.
12. Revisar los mensajes que han mostrado las tres funciones.

    Es muy probable que aparezca una petición del recurso /favicon.ico. Es normal. Más adelante en el curso, haremos hincapié en él.

Pruebas con la función next()

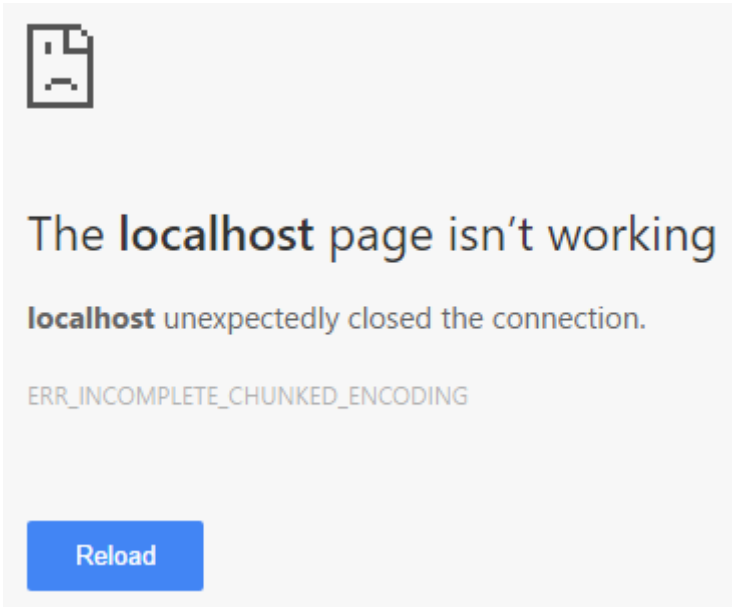
A continuación, vamos a ver qué ocurre cuando una función de *middleware* no invoca el parámetro next():

1. Editar el archivo app.js.
2. Suprimir la invocación del parámetro next() en la función que genera Hello:

```
//genera Hello
app.use(function(req, res, next) {
  console.log("genera Hello");
  res.setHeader("Content-Type", "text/plain");
  res.write("Hello");
});
```

3. Guardar cambios.
4. Ir a la consola.
5. Reiniciar la aplicación.
6. Ir al navegador y solicitar de nuevo http://localhost:8080.

   El navegador parecerá que se ha quedado colgado. En Chrome, más tarde o temprano, recibiremos un mensaje como el siguiente:



7. Ir a la consola y comprobar qué mensajes aparecen.

   Sólo debe aparecer el que informa del recurso solicitado y el de la función que genera Hello. O sea, los mensajes de las dos primeras funciones. Esto se debe a que el motor de *middleware* está esperando que la segunda función invoque el parámetro next().

8. Editar el archivo app.js.
9. Dejar la función, que genera Hello, como sigue:

```
//genera Hello
app.use(function(req, res, next) {
  console.log("Hello");
```

```
  res.setHeader("Content-Type", "text/plain");
  res.write("Hello");

  setTimeout(function() {
    next();
  }, 5000);
});
```

Lo que estamos haciendo es añadir un retardo de cinco segundos, tras el cual, la función invocará el parámetro next().

10. Guardar cambios.
11. Ir a la consola.
12. Reiniciar la aplicación.
13. Ir al navegador y solicitar http://localhost:8080.

    Esta vez, la respuesta llega pero cinco segundos más tarde, por la espera que hemos añadido. Esto indica algo muy sencillo: el motor invoca una función de *middleware* y espera a que ésta ejecute next() para continuar. De esta manera, una función de *middleware* puede ejecutar código asíncronamente y, cuando ha terminado, invocar next() para que el motor continúe con la siguiente función de la pila.

Registro de funciones de control de errores

Para finalizar, vamos a registrar funciones de *middleware* que controlen los errores propagados por la pila de *middleware* mediante el parámetro next():

1. Editar el archivo app.js.
2. Modificar las funciones de *middleware* que tenemos por lo siguiente:

```
//logger
app.use(function(req, res, next) {
  console.log(req.ip, req.method, req.originalUrl);
  next();
});

//genera Hello
app.use(function(req, res, next) {
  console.log("genera Hello");
  res.setHeader("Content-Type", "text/plain");
  res.write("Hello");
  next();
});

//control de errores #1
app.use(function(err, req, res, next) {
  console.log("control de errores #1");
  next(err);
});

//genera World!
app.use(function(req, res, next) {
  console.log("genera World");
  res.write("World!");
  res.end();
});

//control de errores #2
app.use(function(err, req, res, next) {
  console.log("control de errores #2");
});
```

Por buenas prácticas, se recomienda ubicar las funciones de control de error al final, después de las normales.

3. Guardar cambios.

4. Ir a la consola.
5. Reiniciar la aplicación.
6. Ir al navegador.
7. Solicitar http://localhost:8080.
8. Ir a la consola y comprobar que no se ha ejecutado ninguna de las funciones de control de errores. Como no se ha invocado next() con error, el motor de *middleware* las omiten.
9. Editar el archivo app.js y modificar la siguiente función:

```
//genera World!
app.use(function(req, res, next) {
  console.log("genera World");
  res.write("World!");
  res.end();
  next("mensaje de error");
});
```

10. Guardar cambios.
11. Ir a la consola.
12. Reiniciar la aplicación.
13. Solicitar http://localhost:8080.

    A pesar de haberse producido un error, la respuesta se ha generado y enviado al cliente.

    Por otra parte, observe que se ha ejecutado la función de control de errores #2, no desde la #1. Es importante tener claro que no existe un flujo normal y otro de control de errores. Sólo hay uno. Lo que ocurre es que si se invoca next(), sin argumentos, el motor de *middleware* pasa a la siguiente función normal registrada en la pila, omitiendo toda función de control de errores que se encuentre. Mientras que si se ejecuta con argumento, a la siguiente de control de errores de la pila, omitiéndose cualquier función normal que se encuentre.
14. Editar el archivo app.js.
15. Añadir la invocación next() con el error al final de la segunda función de control de errores :

```
//control de errores #2
app.use(function(err, req, res, next) {
  console.log("control de errores #2");
  next(err);
});
```

16. Guardar cambios.
17. Ir a la consola.
18. Reiniciar la aplicación.
19. Ir al navegador.
20. Solicitar http://localhost:8080.

    Ahora, como la aplicación no puede encontrar otra función de control de errores en la pila, la aplicación captura el error y lo muestra en la consola.
21. Ir al archivo app.js.
22. Modificar la siguiente función:

```
//control de errores #2
app.use(function(err, req, res, next) {
  console.log("control de errores #2");
  next();
});
```

23. Añadir la siguiente función debajo de logger:

```
app.use(function(req, res, next) {
  throw new Error("mensaje de error.");
});
```

24. Guardar cambios.
25. Ir a la consola.
26. Reiniciar la aplicación.
27. Ir al navegador.
28. Solicitar http://localhost:8080.

    Debemos recibir como respuesta un mensaje con código de error 500 Internal Server Error y además la pila de traza del error.
29. Ir a la consola y comprobar que se han ejecutado los controladores de error.
30. Cerrar todo.