


Middleware, manejo de errores y logs

 gmoralesc.gitbooks.io/creando-apis-con-node-js-express-y-mongodb/content/configurando-el-proyecto-con-express/middleware-manejo-de-errores-y-logs.html

Antes de continuar vamos a mezclar el *feature* anterior y crear una rama para trabajar:

```
git checkout master
git merge feature/04-environment-and-configuration
git checkout -b feature/05-handle-errors-and-logs
```

Hasta el momento la única **ruta** que tenemos configurada en nuestro proyecto con *express* es `/` (La raíz del proyecto) y con el método GET, pero ¿Qué sucede si tratamos de acceder otra ruta?. Pues en este momento no tenemos configurado como manejar esta excepción, en términos de *express* no existe ningún "*Middleware*" que la procese, por lo tanto terminará en un error no manejado por el usuario.

Si vamos al navegador y tratamos de acceder a una ruta como `localhost:3000/posts` obtendremos un resultado generico cómo: `Cannot GET /posts` y un código HTTP 404 que indica que el recurso no fue encontrado, pero nosotros podemos controlar esto con *express*, primero analizemos el fragmento de código que define la ruta:

```
app.get('/', (req, res) => {
  res.send('Hello World');
});
```

1. La aplicación (`app`) adjunta una función *callback* para cuando se realice una petición a la ruta `/` con el método GET.
2. Esta función *callback* recibe 2 argumentos: el objeto de la petición (request) abreviado `req` y el objeto de la respuesta (response) abreviado `res`
3. Una vez se cumplen los criterios establecidos en la definición de la ruta la función de *callback* es ejecutada
4. Se envía la respuesta por parte del servidor (`res.send('Hello world')`) el flujo se detiene y el servidor queda escuchando por más llamados.

Express Middleware

"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."

(Tomado de Utilización de middleware)

Esto quiere decir que nuestro *callback* en realidad es una función *middleware* de *express* y tiene la siguiente firma de parámetros:

```
app.get('/', (req, res, next) => {
  res.send('Hello World');
});
```

Dónde `next` es otra función que al invocarse permite continuar el flujo que habíamos mencionado antes para que otras funciones *middleware* pueden continuar con el procesamiento de la petición.

Esto quiere decir que podemos agregar otro *middleware* al final del flujo para capturar si la ruta no fue procesada por ningún *middleware* definido anteriormente. Modificamos el archivo `server/index.js` de la siguiente manera:

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

const app = express();

app.get('/', (req, res, next) => {
  res.send('Hello World');
});

app.use( (req, res, next) => {
  res.status(404);
  res.json({
    error: 'Error. Route not found'
  });
});

module.exports = app;
```

Nótese varias cosas importantes en el fragmento de código anterior:

1. Se ha agregado la función `next` al *middleware* de la ruta raíz, pero realmente no lo estamos utilizando dentro de la función, pero esto es con el fin hacer concordancia con la firma que tienen las funciones *middleware* (Mas adelante la editaremos)
2. Le hemos indicado a la aplicación (`app`) utilizar una función *middleware*, nuevamente podemos saber que es una función *middleware* de *express* por su firma, revisando el número de parámetros.
3. Estamos estableciendo el código HTTP de la respuesta con `res.status(404)` el cual equivalente a una página no encontrada, en este caso una ruta o recurso.
4. Estamos respondiendo explícitamente un objeto tipo `json` sin necesidad de establecer el `Content-Type`, pues *express* lo hace automáticamente por nosotros a `application/json` y adicional convierte el objeto literal de JavaScript en un objeto JSON como tal.

Guardamos el archivo (*nodemon* reiniciará nuestra aplicación automáticamente) y ahora si vamos nuevamente al navegador e invocamos nuevamente la ruta

`localhost:3000/posts` obtendremos el mensaje que establecimos en el *middleware* que se convierte en la última función de *middleware* de todas las declaradas anteriormente en esta aplicación de *express*:

```
{
  error: 'Error. Route not found'
}
```

Para hacer una analogía a la definición de los *middlewares* es como un sistema de tuberías donde la petición es el agua que esta corriendo, si el *middleware* coincide con la petición este detiene el flujo y no deja seguir pasando el agua (al menos que se abra la llave con la función *next* para que continúe), o de lo contrario el agua sigue fluyendo hasta llegar al final que es el *middleware* genérico para capturar todas las peticiones que no se pudieron procesar antes.

Manejo de errores

Ahora surge la siguiente pregunta, ¿Cómo podemos controlar los errores que sucedan en las diferentes rutas?, *express* permite definir una función *middleware* con una firma de parámetros un poco diferente a las anteriores, introducimos el siguiente código al final del archivo `server/index.js`:

```
...

app.use( (err, req, res, next) => {
  const {
    statusCode = 500,
    message,
  } = err;

  res.status(statusCode);
  res.json({
    message,
  });
});

module.exports = app;
```

Los ... en el código indica que hay código anterior en el archivo

Podemos ver varias cosas en el código anterior:

1. El *middleware* que captura errores en *express* comienza por el parámetro de error abreviado como `err`
2. Obtenemos el `statusCode` del objeto `err`, si no trae ninguno lo establecemos en 500 por defecto que significa *Internal Server Error*, así como el `message` que este trae.
3. Finalmente establecemos el `statusCode` en la respuesta (`res`) y enviamos de vuelta un JSON con el mensaje del error.

Ahora cada vez que invoquemos un error en cualquier *middleware* tendremos uno que lo capture y procese, podríamos aprovechar esto, por ejemplo, para guardar en los *logs* información relevante del error.

Para terminar cambiamos nuevamente la definición de la ruta raíz de la siguiente manera:

...

```
app.get('/', (req, res, next) => {  
  res.json({  
    message: 'Welcome to the API'  
  });  
});
```

...

Nótese que ahora estamos devolviendo un archivo json, pues estamos creando un API.

Antes de continuar guardemos nuestro progreso

```
git add .  
git commit -m "Add middleware for not found routes and errors"
```

Utilizando Morgan y Winston para los logs

Existen muchas librerías que son de mucha utilidad así como *express*, entre ellos están Morgan y Winston, el primero de estos nos ayuda a registrar el acceso a todas las rutas e imprimirlas en la Terminal dándonos un detalle de que peticiones se están enviando al servidor y el segundo nos permite crear un *log* personalizado para imprimir en la Terminal e inclusive guardarlos en archivos si así lo queremos.

Procedemos a instalar las dependencias

```
npm install -S morgan winston
```

Y modificamos el archivo `server/index.js` :

```
const express = require('express');
const morgan = require('morgan');
const logger = require('winston');

const app = express();

app.use(morgan('common'));

app.get('/', (req, res, next) => {
  logger.info('API root');
  res.json({
    message: 'Welcome to the API',
  });
});

app.use((req, res, next) => {
  logger.info('Route not found');
  res.status(404);
  res.json({
    error: 'Route not found',
  });
});

app.use((err, req, res, next) => {
  logger.error(`Error: ${err}`);
  res.status(500);
  res.json({
    error: `${err}`,
  });
});

module.exports = app;
```

Si abrimos el navegador e invocamos diferentes rutas en la dirección del navegador podremos ver los diferentes peticiones que llegan al servidor y adicionalmente los mensajes que imprimimos en la consola mediante los *logs*, esto es muy útil también en el momento del desarrollo.

Mas información:

- [Morgan](#)
- [Winston](#)