Pila de middleware

Tiempo estimado: 10min

Una vez introducidos los conceptos de aplicación Express y de controladores de petición, uno de los primeros componentes a comprender es la pila de *middleware* o, lo que es lo mismo, el flujo de procesamiento de la aplicación.

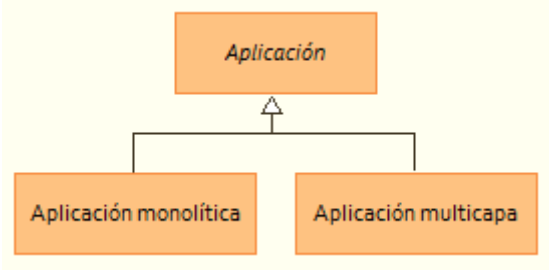
Comenzamos la lección distinguiendo los dos tipos básicos de aplicación, las monolíticas y las multicapa. Seguimos con los componentes de *middleware* y la pila de *middleware*. A continuación, mostramos cómo registrar funciones de *middleware* en la pila. Finalizamos describiendo cómo controlar los errores producidos durante el procesamiento de la petición HTTP en curso.

Al finalizar la lección, el estudiante sabrá:

- Qué es un componente de *middleware*.
- Qué es la pila de *middleware* o flujo de procesamiento.
- Cómo registrar funciones de *middleware* en la pila.
- Qué diferencia hay entre las funciones normales de *middleware* y las de control de errores.
- Cuándo se invoca las funciones normales y cuándo las de control de errores.

Introducción

Básicamente, el desarrollo de aplicaciones webs se puede hacer desde dos puntos de vista, de manera monolítica o multicapa. Una aplicación monolítica (*monolithic application*) es aquella en la que se desarrolla *todo* en un único componente, capa o controlador. En cambio, una aplicación multicapa (*multitier application*) es aquella que utiliza varios componentes, capas o controladores, cada uno de los cuales con una funcionalidad y procesamiento bien claro y definido.

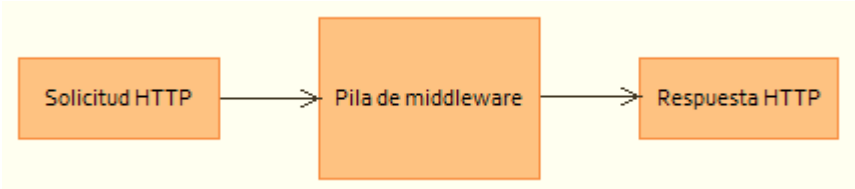


Actualmente, preferimos los entornos multicapa porque son más sencillos de desarrollar y, por encima de todo, de mantener y probar. Como no podía ser de otra manera, Express permite el desarrollo de aplicaciones multicapa y lo hace mediante el uso de la pila de *middleware*.

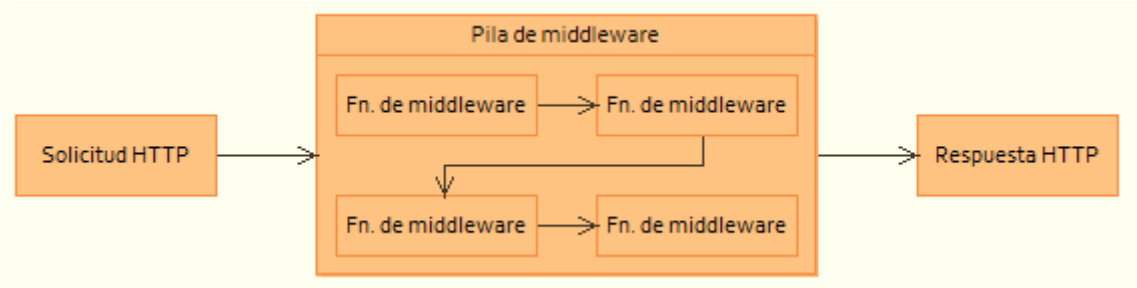
Antes de presentar la pila, hay que tener claro que es el *middleware*. Un componente de middleware (*middleware component*) no es más que el término formal con que se conoce a una pieza de software reutilizable. La cual realiza una determinada funcionalidad de procesamiento de las peticiones HTTP. Así, por ejemplo, tenemos componentes de *middleware* para llevar a cabo el proceso de autenticación, la aplicación de restricciones de seguridad, la publicación de contenido estático, etc.

Por su parte, la pila de middleware (*middleware stack*), también conocida como flujo de procesamiento (*processing flow*) o conducto (*pipeline*), contiene la secuencia de funciones de *middleware* que procesan, una detrás de otra, las peticiones HTTP recibidas de los clientes para construir, entre todas ellas, las respuestas HTTP a remitir como contestación. Todo hay que decirlo, algunas funciones no participan en la redacción de la respuesta como, por ejemplo, el *middleware* de registro de eventos que escribe en un archivo o en la salida estándar información sobre la solicitada en procesamiento.

La idea que se esconde bajo este sistema de *middleware* es que toda petición que reciba la aplicación pase por el flujo de funciones de *middleware* registradas en la pila y que, entre todas ellas, lleven a cabo su tratamiento, generándose la respuesta a remitir al cliente.



En Express, la pila está formada por funciones, conocidas formalmente como funciones de middleware (*middleware functions*) o controladores de petición (*request handlers*), disponibles a través de componentes de *middleware*. A modo de ejemplo, consideremos el componente serve-static. Se utiliza cuando deseamos que la aplicación sirva contenido estático. Este componente dispone de una función, *no middleware*, que recibe la ruta del directorio que contiene los archivos que puede servir estáticamente. Y devuelve la función de *middleware* que hay que registrar en el flujo de procesamiento para que así pueda servirlos cuando sea necesario.



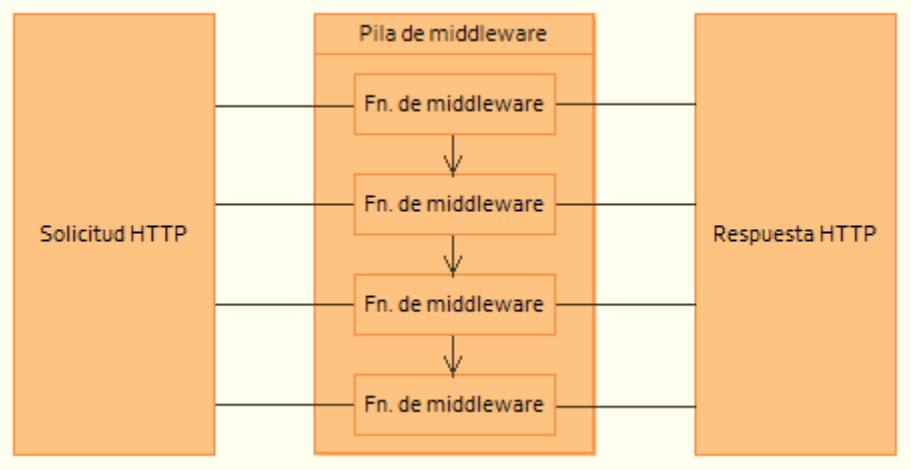
Tal como veremos a lo largo del curso, un componente de *middleware* puede ser básicamente dos cosas:

- Una función de *middleware* por sí misma.
- Una función que devuelve funciones de *middleware*.

En cualquier caso, lo importante a recordar es que en el flujo de procesamiento de la aplicación sólo debemos registrar funciones de *middleware* o controladores de petición.

El componente de la aplicación Express que se encarga de ejecutar ordenadamente las distintas funciones de *middleware* registradas en la pila de procesamiento, se conoce formalmente como motor de middleware (*middleware engine*).

Todos los componentes de *middleware* tienen acceso tanto a la petición en procesamiento como a la respuesta a remitir al cliente. Así pues, pueden analizar su contenido como, por ejemplo, sus cabeceras HTTP o el cuerpo del mensaje, y tras analizar la parte que les corresponde, generar, si es necesario, la parte de la respuesta HTTP asignada a su funcionalidad. Grosso modo, cuando se recibe una petición HTTP, la aplicación se la pasa al motor de *middleware*, el cual va invocando, una a una en orden de registro, las distintas funciones registradas. Tras finalizar la ejecución de la pila, el motor le pasa la respuesta HTTP generada por el *middleware* a la aplicación para su envío al cliente.



Como toda función de *middleware* o controlador de petición tiene acceso a la respuesta HTTP que se remitirá al cliente, está claro que podrá consultar cualquier modificación o añadidura que haya realizado cualquiera de los componentes anteriores de la pila. Por ejemplo, el *middleware* encargado de la generación de entradas en el registro de eventos sólo trabaja sobre la solicitud HTTP, no así sobre la respuesta.

Funciones de middleware

Como ya sabemos, una función de *middleware* es una función JavaScript que realiza una determinada funcionalidad de la aplicación. Es un controlador de petición. Puede trabajar sobre el objeto que representa la petición HTTP recibida del cliente y/o el objeto que representa la respuesta HTTP que la aplicación acabará remitiendo al cliente como contestación.

La función, al ser un controlador de petición, debe presentar la siguiente signatura:

```
function(req, res)
function(req, res, next)
```

Parámetro	Tipo de datos	Descripción
req	Request	Solicitud HTTP en procesamiento.
res	Response	Respuesta HTTP que se está generando.
next	function	Función que debe invocar el *middleware* para indicarle al motor de *middleware* que ejecute el siguiente componente de la pila de procesamiento: next([error]).

Registro de funciones de middleware

Mediante el registro de middleware (*middleware register*) se añade, al final de la pila de procesamiento, una función de *middleware*. Se realiza mediante el método use() de la aplicación:

```
use(fn)
use(route, fn)
```

Parámetro	Tipo de datos	Descripción
route	string	Ruta a la que se aplicará el componente de *middleware*. Si no se especifica, se asumirá que se debe ejecutar para toda petición.
fn	function	Función que implementa la lógica del componente de *middleware*.

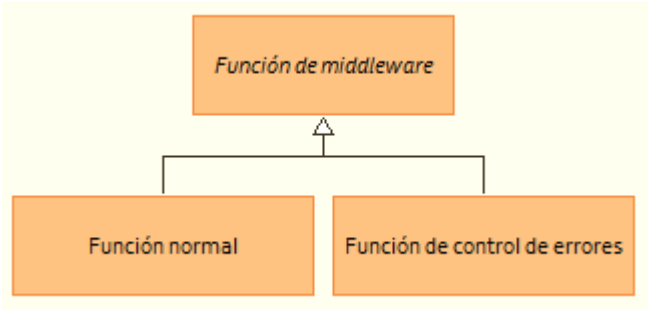
Con el registro de funciones de *middleware* lo que estamos haciendo es añadir o dotar de más funcionalidad a la aplicación.

Orden de registro

El orden en que se registra las funciones de *middleware* es importante. Si un determinado componente utiliza algo generado por otro, es necesario registrar primero la función de la que depende para que de esta manera el motor de *middleware* la invoque primero y, así, la segunda pueda acceder a cualquier objeto generado por éste.

Middleware de control de errores

Básicamente, hay dos tipos de funciones de *middleware*, las normales y las de control de errores.



Una función normal (*normal function*) es aquella que se ejecuta mientras *no* se produzca error. Las vistas hasta ahora. Tienen dos o tres parámetros: la solicitud, la respuesta y la función de continuación de flujo, next(). En cambio, una función de control de errores (*error-handling function*) es aquella que atiende y procesa un error comunicado a través de la función de continuación de flujo.

Las funciones de error se registran también mediante el método use() de la aplicación, pero tienen una signatura distinta de las normales:

```
function(error, req, res, next)
```

Parámetro	Tipo de datos	Descripción
error	object	Error propagado mediante una invocación next(error) anterior.
req	Request	Petición HTTP en procesamiento.
res	Response	Respuesta HTTP que se está generando.
next	function	Función que debe invocar la función de control de errores para indicarle al motor de *middleware* que ejecute la siguiente de la pila de procesamiento.

La función de continuación de flujo, next(), tiene un comportamiento distinto según se invoque con o sin argumento de error:

- Si no le pasamos ningún argumento, invocará la siguiente función normal registrada en la pila. Esto es así tanto si lo hacemos desde una función normal como desde una de control de errores.
- Si le pasamos un argumento, invocará la siguiente función de control de errores registrada en la pila. Esto es así tanto si lo hacemos desde una función normal como desde una de control de errores.

Generalmente, las funciones de control de errores se registran después de las funciones normales. He aquí un ejemplo ilustrativo:

```
app.use(function(req, res, next) {
  //normal
  next();
});

app.use(function(req, res, next) {
  //otra normal
});

app.use(function(err, req, res, next) {
  //control de errores
  next(err);
});

app.use(function(err, req, res, next) {
  //otra de control de errores
});
```

Si todo va bien en el flujo de procesamiento, las funciones de error *no* se ejecutan *nunca*. Recordemos, sólo cuando alguna función de *middleware* ejecuta la función next() con un argumento, el cual se considera como el error. Y ojo, si tenemos varias, para que se siga con la cadena de control de errores, es necesario que las

controladoras de error invoquen la función next() con el error, porque si se ejecuta sin error, se devolverá el flujo a las funciones normales. Si una deja de hacerlo, las funciones de error que le sigan *no* serán invocadas por el motor de *middleware*.

Finalmente, hay que decir que si la última función de error registrada en la pila invoca next() con el error, la aplicación mostrará el error por la consola.

Captura de excepciones

Cuando una función de *middleware*, sea cual sea su tipo, propaga un error mediante la sentencia throw, la aplicación Express lo captura. Finaliza el flujo de procesamiento normal. A continuación, el motor de *middleware* busca la función de control de errores en la pila que siga a aquella que propagó el error. Y finalmente, genera una respuesta HTTP con código de estado 500 Internal Server Error y se lo remite al cliente.

