CONTENIDO ESTÁTICO

Tiempo estimado: 15min

Una de las primeras cosas que tenemos que aclarar es la diferencia entre contenido estático y dinámico. El objeto de esta lección es presentar cómo servir contenido estático desde una aplicación Express.

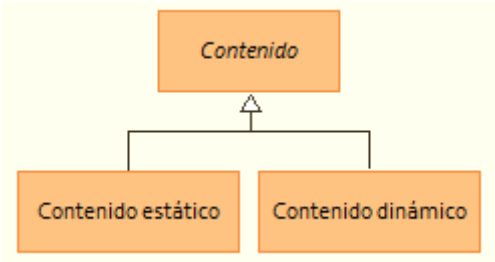
Comenzamos la lección introduciendo el concepto de contenido y la diferencia entre contenido estático y dinámico. A continuación, presentamos el componente de *middleware* serve-static para servir contenido estático. Lo que ayuda a no tener que implementar esta lógica nosotros mismos, aumentando así nuestra productividad. Además, se presenta el concepto de caché HTTP y algunas cabeceras de respuesta configurables en serve-static. Finalmente, se muestra el *middleware* serve-favicon para servir el favicono del sitio o aplicación web.

Al finalizar la lección, el estudiante sabrá:

- Cómo configurar la aplicación Express para que sirva contenido estático.
- Cómo utilizar el componente de *middleware* serve-static para servir contenido estático.
- Cómo configurar cabeceras HTTP como Cache-Control, Last-Modified y Accept-Ranges en serve-static.
- Cómo configurar el *middleware* serve-favicon para servir el favicono del sitio o aplicación web.

INTRODUCCIÓN

El contenido (*content*) es la secuencia de bytes que forman un recurso web, generalmente, un archivo de texto, un documento HTML o una imagen. Las aplicaciones pueden servir o publicar tanto contenido estático como dinámico.



El contenido estático (*static content*) es aquel que la aplicación puede servir directamente de disco. Mientras que el contenido dinámico (*dynamic content*) aquel que requiere su generación cada vez que se solicita.

Hay organizaciones que utilizan un servidor web como Apache o Nginx para servir directamente el contenido estático de una aplicación web, delegando el contenido dinámico en la aplicación Express. Otras prefieren que la aplicación web sirva tanto el contenido estático como el dinámico. Para gustos, colores.

MIDDLEWARE SERVE-STATIC

Podemos utilizar el *middleware* serve-static para configurar carpetas de la aplicación cuyo contenido es estático. La idea es indicarle al *middleware* que cuando entre una petición web, compruebe si el recurso existe en alguno de los directorios que le hemos indicado y, si así es, lo sirva directamente. En caso de no estarlo, dejará pasar la petición por el flujo de procesamiento con intención de que lo genere la aplicación dinámicamente u otro componente estáticamente.

Para poder utilizar esta función de *middleware*, debemos importarla y registrarla. No hay que olvidar añadir el paquete serve-static a las dependencias de la aplicación, esto es, a la propiedad dependencies del archivo package.json.

Generalmente, se conoce formalmente como directorio público (*public directory*) aquel que contiene contenido estático. Por convenio, se utiliza el directorio public de la aplicación. En aplicaciones pequeñas e incluso medianas, suele haber un único directorio público. Las grandes pueden tener, si es necesario, varios.

Función serve-static

La función serve-static devuelve una función de *middleware* para servir un determinado directorio público. Tiene la siguiente signatura:

```
function serve-static(root, options) : function
```

Parámetro	Tipo de datos	Descripción
root	string	Directorio público.
options	object	Opciones de servicio: • acceptRanges (boolean). ¿Permitir el envío de recursos parciales? Valor predeterminado: true. • cacheControl (boolean). ¿Activar la cabecera de respuesta Cache-Control? Valor predeterminado: true. • etag (boolean). ¿Generar la cabecera ETag? Valor predeterminado: true. • extensions (string[]). Extensiones a añadir a los recursos solicitados para encontrar el archivo a servir. Ejemplo: ["html", "htm"]. Valor predeterminado: ninguno. • index (boolean o string). ¿Enviar el archivo index.html cuando se solicite un directorio? true, sí; false, no; o bien el archivo a remitir. Valor predeterminado: index.html. • lastModified (boolean). ¿Enviar la cabecera Last-Modified? Valor predeterminado: true. • maxAge (number). Número de milisegundos que puede cachear el cliente el archivo. Valor predeterminado: 0. • redirect (boolean). ¿Redirigir a / cuando el recurso solicitado sea un directorio? Valor predeterminado: true. • setHeaders (function). Función que debe invocar la función de *middleware* para fijar las cabeceras de la respuesta: fn(res, path, stat). res representa el objeto respuesta; path, la ruta del archivo a remitir; y stat, el objeto stat del archivo a remitir.

Registro de la función de middleware

Recordemos que la función serve-static devuelve una función de *middleware* con la que servir un determinado directorio público. Es por tanto la función devuelta lo que hay que registrar en la aplicación mediante el método use(). Si tenemos varios directorios públicos, invocaremos la función una vez para cada directorio, registrando las distintas funciones devueltas en la aplicación según su orden de prioridad.

Por convenio, el paquete serve-static que representa la función homónima se importa como serveStatic. He aquí un ejemplo ilustrativo:

```
//imports
import express from "express";
import serveStatic from "serve-static";
…

//registro
const app = express();
app.use(serveStatic(path.join(__dirname, "public")), {index: ["html"]});
…
```

La función de *middleware* debe ser una de las primeras funciones del ciclo de procesamiento. Si el recurso solicitado se encuentra en su directorio público, rellena la respuesta HTTP a remitir al cliente y cancela el resto del flujo de procesamiento, esto es, la pila de *middleware*. Pero si no lo encuentra, no cancela el flujo, ni propaga error. Simplemente, no hace nada y deja que las funciones siguientes rellenen la respuesta. De ahí que se

recomiende que se encuentre al comienzo del flujo de procesamiento.

Caché HTTP

En Internet, una de las cosas que más valoran los usuarios es la rapidez con que el navegador visualiza las páginas webs. Esto depende de muchas cosas. Una de ellas, la velocidad con que el navegador recibe los recursos de los servidores webs. Para ello, el protocolo HTTP permite que los clientes, los *proxies* y las aplicaciones dispongan de cachés.

Una caché (*cache*) no es más que un almacén o contenedor en el que se guarda datos accedidos con mucha frecuencia o recientemente. Su objetivo es mejorar su velocidad de acceso. Por ejemplo, un *proxy* puede almacenar en su caché los recursos más frecuentemente accedidos y remitirlos directamente sin tener que enviar las solicitudes clientes a las aplicaciones servidoras. Por otra parte, los navegadores disponen de una pequeña caché donde almacenar las últimas páginas accedidas por el usuario. De tal manera que pueden utilizar la copia local, en vez de solicitársela al servidor y esperar a que éste se la envíe de nuevo.

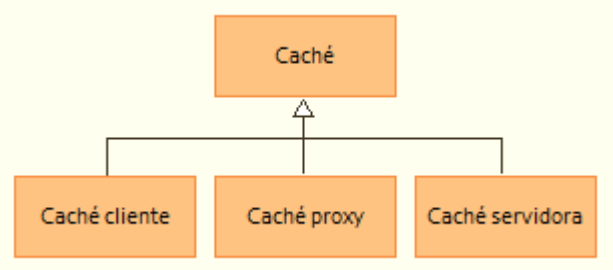
Por lo general, es más rápido el acceso a datos cacheados que la generación del recurso o bien la solicitud y envío de una copia. Esto es muy útil cuando se genera contenido dinámico donde una parte procede de una base de datos. También lo es con contenido estático, porque si el recurso se encuentra en la caché local, no hay que volver a solicitarlo y se puede servir directamente.

El principal objetivo de la caché es reducir la latencia (*latency*), el tiempo que transcurre desde que el usuario solicita el recurso y lo obtiene. En el 99% de los casos, suministrar un recurso directamente de una caché presenta una latencia menor que solicitárselo a su correspondiente servidor o aplicación web y esperar su recepción.

La operación mediante la cual se introduce un dato en la caché se conoce formalmente como cacheo (*caching*).

Ubicación de la caché

Cada vez que se solicita y sirve un recurso web, intervienen varios componentes o dispositivos webs, cada uno de los cuales puede disponer de su propia caché con objeto de responder más rápidamente.



La caché ubicada en el cliente se conoce como caché cliente (*client cache*) o caché local (*local cache*). La mantiene, generalmente, el navegador. En ella, guarda las últimas páginas accedidas por el usuario y la usa para obtener una copia cuando el usuario vuelve a pedir alguna de ellas.

Por otra parte, tenemos cachés en los dispositivos intermedios, generalmente, *proxies*. Un proxy web (*web proxy*) es una aplicación que intermedia entre los clientes y los servidores webs. Estos dispositivos suele tener la capacidad de mantener copias de los recursos que pasan por sus manos en una caché proxy (*proxy cache*). Cada vez que un *proxy* recibe una petición cliente, comprueba primero si dispone de una copia cacheada del recurso solicitado. Si así es, la sirve directamente sin remitírsela al servidor web. Si no lo es, se la remite al servidor web y espera su respuesta. Entonces, la cachea y se la envía al cliente. Así la próxima vez dispondrá de una copia del recurso y podrá servirla directamente.

Es muy importante tener claro que los *proxies no* siempre cachean lo que pasa por ellos. Pero su uso suele ser muy útil cuando se sabe que el contenido no cambiará y, por lo tanto, se puede almacenar en una caché *proxy*.

Finalmente, tenemos la caché servidora (*server cache*), en nuestro caso, una específica de la aplicación Express.

Es importante tener claro que serve-static no dispone de caché, pero *sí* puede notificar a las cachés *proxies* y clientes si el recurso remitido no cambiará en un determinado período de tiempo y, por lo tanto, pueden

cachearlo sin miedo a cambios inesperados.

Expiración

La expiración (*expiration*) es el intervalo de tiempo que se puede mantener un recurso cacheado. Transcurrido éste, se considera que la copia ya no es válida. El dispositivo tendrá que volver a solicitar el recurso al servidor web si el usuario vuelve a accederlo.

¿Quién fija este período de tiempo? Como no puede ser de otra manera, quien publica el recurso, esto es, el servidor web o, en nuestro caso, la aplicación Express.

Cabecera Cache-Control

Mediante la cabecera HTTP Cache-Control nuestra aplicación Express puede notificar si se puede cachear el recurso y durante cuánto tiempo puede hacerse.

serve-cache proporciona dos opciones de configuración: cacheControl y maxAge. Mediante cacheControl, se indica si serve-static debe añadir esta cabecera cuando sirva un recurso estático. Mientras que con maxAge, el período de expiración.

Cabecera Last-Modified

Un servidor web puede comunicar al cliente la fecha de la última modificación de un recurso mediante la cabecera HTTP Last-Modified:

```
Last-Modified: Fecha-HTTP
```

Cuando la opción lastModified de serve-static se encuentra activa, el *middleware* añadirá a la respuesta esta cabecera, usando como valor la fecha de última modificación del archivo remitido.

HTTP presenta cabeceras adicionales mediante las cuales un cliente puede hacer solicitudes como: *sírveme el recurso si se ha modificado posteriormente a la fecha que te indico*. De esta manera, pregunta si puede utilizar una copia cacheada cuyo período de expiración ha vencido, siempre y cuando el recurso no haya sufrido cambios.

Cabecera Accept-Ranges

HTTP permite que los clientes soliciten el contenido completo de un recurso o sólo parte de él. Los servidores webs deben poder servir como mínimo los recursos al completo. Si proporcionan la funcionalidad de publicación parcial, pueden comunicárselo a los clientes mediante la cabecera Accept-Ranges:

```
Accept-Ranges: bytes|none
```

Mediante el valor bytes, el servidor le comunica a los clientes que puede servir contenido parcial de un recurso. Mediante none, que no lo hace.

serve-static puede servir total o parcialmente un archivo. Para configurar la publicación parcial, debe añadir la cabecera Accept-Ranges a sus respuestas. Mediante la opción acceptRanges fijamos si debe hacerlo. Si se fija a false, no la añadirá y sólo servirá recursos al completo.

Middleware serve-favicon

Un favicono (*favicon*) es una pequeña imagen asociada a un sitio web que muestran principalmente los navegadores en la pestaña de la página. Generalmente, es de 16 x 16 píxels. He aquí unos ejemplos:

Se trata del archivo favicon.*, generalmente, favicon.ico o favicon.png. Este archivo se indica en la página web mediante un elemento <link> como el siguiente:

```
<link rel="icon" type="image/png" href="/images/favicon.png">
```

Generalmente, los navegadores solicitan automáticamente el archivo favicon.ico para cada recurso web que solicitan. Cuando reciben una página web, si ésta indica el elemento <link> solicitará el favicono ahí indicado, en vez de favicon.ico.

Aunque el favicono es estático y, por lo tanto, se puede servir mediante el *middleware* serve-static, por lo general se prefiere serve-favicon. Principalmente, porque este *middleware* cachea el favicono en la memoria principal, por lo que servirlo será más rápido que si hay que recurrir a disco constantemente.

Función serve-favicon

La función serve-favicon devuelve una función de *middleware* para servir el favicono de la aplicación. Recordemos generalmente el archivo favicon.ico o favicon.png. Su signatura es como sigue:

```
function serve-favicon(path) : function
function serve-favicon(path, options) : function
```

Parámetro	Tipo de datos	Descripción
path	string	Ruta del archivo favicono.
options	object	Opciones de servicio: • maxAge (number). Período de expiración en milisegundos. Valor predeterminado: un año.

La opción maxAge tiene como objeto indicar el período de expiración a adjuntar en la cabecera Cache-Control de la respuesta, tal como vimos anteriormente. Como el favicono es un archivo que no suele cambiar con frecuencia, se recomienda que los dispositivos lo almacenen en su caché para evitar que la aplicación web tenga que responder contínuamente a las peticiones de los navegadores sobre el favicono.

Registro de la función de middleware

Al igual que serve-static, serve-favicon no es una función de *middleware* propiamente dicha. Es una función que devuelve la función de *middleware* a registrar en la aplicación. Se recomienda hacerlo al comienzo del flujo de procesamiento. Antes que serve-static.

Por convenio, el paquete serve-favicon se importa como serveFavicon o simplemente favicon. He aquí un ejemplo ilustrativo:

```
//imports
import express from "express";
import serveFavicon from "serve-favicon";
…

//registro
const app = express();
app.use(serveFavicon(path.join(__dirname, "public/images/favicon.png")), {maxAge: 43200000});
app.use(serveStatic(path.join(__dirname, "public")));
…
```

