Tiempo estimado: 15m

Una aplicación web no es más que un servidor de recursos mediante HTTP. Estos recursos se pueden organizar física o lógicamente mediante carpetas. Cuando el recurso es servido por el componente de *middleware* serve-static, la organización es física. Todo se organiza en forma jerárquica dentro del directorio público. En el caso de contenido dinámico, la organización debe de ser también posible. Y como no podía ser menos, Express así lo permite. La organización de recursos dinámicos es más fácil de hacer si se usan rutas, el objeto de esta lección.

Empezamos introduciendo los conceptos de encaminador y ruta. A continuación, presentamos cómo acceder al encaminador principal de la aplicación, así como la definición de encaminadores secundarios. Después, mostramos los distintos tipos de rutas: estáticas, dinámicas y parametrizadas. Finalizamos la lección con los módulos de rutas. Módulos que se utilizan, por convenio, para organizar las rutas de una aplicación y/o componente.

Al finalizar la lección, el estudiante sabrá:

- Qué es un encaminador.
- Cómo definir rutas.
- Qué tipos de rutas puede definir.
- Cómo organizar las rutas mediante módulos.

INTRODUCCIÓN

Cuando se sirve contenido dinámicamente, debemos indicar una función JavaScript para procesar la petición y genera la respuesta a remitir al cliente. Este tipo de funciones se conocen formalmente como controladoras (*handlers*), tal como hemos visto ya en el curso en varias ocasiones. Podemos usar la misma función controladora para toda la aplicación e incluso una para cada recurso o conjunto de recursos. Esto mejora el mantenimiento de la aplicación.

Para conseguirlo, se hace uso de encaminadores.

Un encaminador (*router*) es un objeto que invoca una u otra función controladora atendiendo al *path* del recurso accedido. Para ello, los encaminadores permiten definir lo que se conoce como rutas. Donde una ruta (*route*) no es más que el identificador local de un recurso o conjunto de recursos cuyas peticiones son procesadas por una función controladora. Algunos ejemplos de rutas son: /, /index.html, /band/the-national y /users/. Cada aplicación debe definir sus rutas específicas.

Toda ruta tiene:

- Un identificador (*path*), el texto que identifica el recurso dentro de la aplicación. Puede ser un identificador estático, identifica a un único recurso, o bien dinámico, a varios.
- Una o más funciones controladoras. Teniendo en cuenta que un mismo recurso puede ser accedido mediante distintos métodos HTTP como GET, PUT, POST, PATCH, etc., Express permite indicar un controlador para cada uno de estos métodos o bien el mismo para todos ellos. Según nuestra necesidad.

Es importante tener muy claro que una aplicación puede disponer de varios encaminadores. Uno de ellos se conoce como encaminador principal (*main router*) y se encarga de procesar, en primera instancia, las peticiones dinámicas, esto es, aquellas que no son procesadas por los componentes que sirven contenido estático como, por ejemplo, serve-static o serve-favicon. Este encaminador puede tener asociados otros encaminadores para determinadas rutas. Así, por ejemplo, todas las rutas que cuelguen de /band/ pueden ser atendidas por un encaminador y las rutas que cuelguen de /user/ por otro. Todas las demás, por el principal.

Por otra parte, tenemos el concepto de punto de montaje (*mount point*), aquel *path* a partir del cual el encaminador responde. Cuando tenemos un único encaminador, el principal, su punto de montaje es el punto de montaje de la aplicación. Ahora bien, cuando tenemos varios encaminadores, generalmente, cada uno de ellos tiene su propio punto de montaje. Todas las rutas que cuelguen del punto de montaje, se remitirán al encaminador correspondiente. Lo importante es tener dos cosas claras:

- Todo encaminador tiene un punto de montaje.
- Toda ruta definida en un encaminador es relativa a su punto de montaje.

Encaminador principal

Recordemos que toda aplicación Express tiene un encaminador principal. Aquel que atiende las peticiones, en primer lugar, y que contiene e invoca a los demás encaminadores de la aplicación cuando es necesario. Podemos acceder al objeto de este encaminador mediante la propiedad router de la aplicación.

Encaminadores secundarios

Las aplicaciones pueden tener tantos encaminadores como sea necesario. El encaminador principal lo crea la propia aplicación cuando invocamos la función express(). Mientras que el resto, los secundarios, los podemos crear nosotros mismos mediante la función express.Router() y, a continuación, registrarlos mediante el método use() de la aplicación. Como con cualquier función de *middleware*.

Creación de encaminadores secundarios

A diferencia de otros componentes de *middleware* como serve-static o serve-favicon, que tienen sus paquetes específicos, el componente de encaminamiento viene de fábrica en el paquete express. Se puede crear un encaminador secundario mediante la función Router() de express:

```
function Router() : function
function Router(options) : function
```

Parámetro	Tipo de datos	Descripción
options	object	Opciones del encaminador: • caseSensitive (boolean). ¿Activar el modo de sensibilidad a mayúsculas y minúsculas en las rutas? • mergeParams (boolean). ¿Mantener req.params del encaminador padre? Valor predeterminado: false. • strict (boolean). ¿Activar el encaminamiento estricto? Valor predeterminado: false.

Veamos un ejemplo ilustrativo:

```
router = express.Router();
```

Registro de encaminadores secundarios

Una vez creado el encaminador secundario, debemos registrarlo en la aplicación mediante el método use() de la aplicación. Un aspecto que todavía no hemos visto, y ha llegado el momento de hacerlo, es que las funciones de *middleware* se pueden registrar para que se ejecuten para cualquier petición HTTP recibida o bien sólo con peticiones de *paths* específicos. He aquí las signaturas del método use():

```
use(middleware)
use(path, middleware)
```

Parámetro	Tipo de datos	Descripción
path	string	Ruta para la que registrar la función de *middleware*.
middleware	function	Función de *middleware*.

A continuación, un ejemplo de registro de encaminador secundario:

```
//imports
import express from express;
…
```



```
//app
const app = express();
const router = express.Router();

app.use(router);
…
```

Registro de rutas

Recordemos que una ruta (*route*) no es más que una dirección o texto mediante el cual se identifica un determinado recurso como, por ejemplo, /, /index.html o /band/nada-surf. Y a la que se asocia una función que procesa sus peticiones. La ruta no es más que la parte de la URL que identifica el recurso dentro de la aplicación, es decir, la ruta de solicitud (*request path*). Recordemos que la URL de un recurso está formada básicamente por los siguientes componentes:

```
protocolo://dominio:puerto/path?cadenaDeConsulta#idFragmento
```

Las rutas configuradas en el encaminador se hacen con respecto al *path* de las peticiones HTTP.

No hay que olvidar que un mismo recurso se puede acceder mediante uno o más métodos HTTP como, por ejemplo, GET, POST, PUT, DELETE, PATCH, etc. Podemos definir el mismo controlador de ruta para los métodos soportados por la ruta o bien definir un controlador particular para cada uno de los métodos. Lo más habitual.

La definición de rutas es muy sencilla. Hay que indicarle al encaminador el método HTTP, el *path* y el controlador. Para ir abriendo boca, no hay nada como un ejemplo ilustrativo:

```
router.get("/", function(req, res, next) {
  res.send("Esto es el contenido");
});
```

Si deseamos definir la ruta en el encaminador principal, haremos lo mismo, pero usando el objeto aplicación:

```
app.get("/", function(req, res, next) {
  res.send("Esto es el contenido");
});
```

Es muy probable que el estudiante haya observado que a lo largo del curso ya hemos definido rutas mediante app.get(). Y en efecto, así es. La diferencia es que hemos usado una ruta comodín, *, mediante la cual se indica el controlador de petición para todas las solicitudes. Y hemos usado req.path para especificar qué hacer con determinadas solicitudes. Ahora, ha llegado el momento de ver cómo definir controladores de petición para rutas específicas.

Para cada método HTTP, el encaminador y la aplicación disponen de su correspondiente método. Los métodos hay que invocarlos con el *path* de la ruta en cuestión y su controlador. Así pues, si deseamos que el recurso /ejemplo sea accesible mediante PUT y POST, deberemos registrar el controlador de PUT mediante el método put() y el de POST mediante post().

Todos estos métodos presentan la siguiente signatura:

```
function método(request, response)
function método(request, response, next)
```

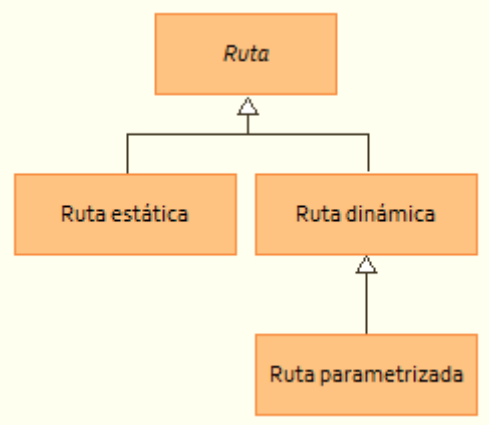
Parámetro	Tipo de datos	Descripción
request	Request	La petición HTTP en procesamiento.
response	Response	La respuesta HTTP a remitir al cliente.
next	function	Función de continuación del flujo de procesamiento.

Actualmente, se dispone de los siguientes métodos para asociar controladores a métodos HTTP: checkout(), copy(), delete(), get(), head(), lock(), merge(), mkactivity(), mkcol(), move(), m-search(), notify(), options(), patch(), post(), purge(), put(), report(), search(), subscribe(), trace(), unlock() y unsubscribe(). Además, disponemos de all() para registrar un controlador para todos los métodos HTTP.

Tipos de rutas

Las rutas las podemos clasificar, atendiendo a cuántos recursos identifican, en estáticas, dinámicas y

parametrizadas.



Una ruta estática (*static route*) es aquella que representa o identifica a un único recurso. Mientras que una ruta dinámica (*dynamic route*) es aquella que representa o identifica varios; tiene una parte con la cual se adapta a cada situación. Finalmente, una ruta parametrizada (*parameterized route*) es una ruta dinámica que contiene parámetros cuyos valores podemos acceder fácilmente en el controlador de petición.

Rutas estáticas

Cuando un controlador de petición debe asociarse a un único recurso, hay que utilizar una ruta estática. No es más que el *path* del recurso.

Veamos un ejemplo:

```
app.get("/band/generationals", function(req, res, next) {
  …
});
```

Rutas dinámicas

Por su parte, una ruta dinámica tiene la capacidad de representar o identificar varios recursos a la vez. Se definen mediante expresiones regulares, lo que permite su adaptación a distintas rutas. Cuando se usa este tipo de ruta, cada vez que un recurso cumpla el patrón dado, se ejecutará la función controladora.

He aquí un ejemplo ilustrativo:

```
app.get("/band/*", function(req, res, next) {
  …
});
```

Rutas parametrizadas

Una ruta parametrizada es una ruta dinámica que define parámetros cuyos valores podemos utilizar posteriormente en el controlador. Estos parámetros se definen en la ruta mediante la siguiente sintaxis:

```
:parámetro
```

Así, por ejemplo, la ruta /band/:name puede adaptarse a recursos como /band/something-happens y /band/telekinesis. Cuando se acceda al recurso /band/something-happens, el parámetro name tendrá el valor something-happens; mientras que con /band/telekinesis, telekinesis.

Ejemplo:

```
app.get("/band/:name", function(req, res, next) {
  …
});
```

El encaminador permite el acceso a los parámetros mediante la propiedad params del objeto petición. Así, por ejemplo, para la ruta /band/:name, el acceso al parámetro name se hará mediante req.params.name. De esta manera, se podrá usar su valor para acceder, por ejemplo, a una base de datos y obtener el contenido sobre la banda que está solicitando el usuario.

Los dos puntos se usan para indicar el parámetro y su nombre. Una misma ruta puede presentar varios parámetros, tal como muestra el siguiente ejemplo:


```
/bands/:name/:member
```

Módulo de rutas

Para facilitar el mantenimiento de las aplicaciones Express, en muchas ocasiones, las funciones controladoras de las rutas no se definen en el archivo principal de la aplicación. En su lugar, por convenio, se crea una carpeta /routes, en la que se definirá un módulo para cada grupo de recursos relacionados. Por ejemplo, si tenemos los recursos y grupos de recursos /, /band/ y /user/, definiremos los siguientes archivos de rutas: routes/index.js, routes/band.js y routes/user.js.

Cada módulo de rutas devolverá un objeto express.Router, el cual deberá registrarse en el archivo principal de la aplicación mediante el método app.use() tal como hemos visto anteriormente. Por ejemplo, el módulo del recurso principal podría ser como sigue:

```
//routes/index.js
//imports
import express from "express";

//api
export const router = express.Router();

//routes and handlers
router.get("/", function(req, res, next) {
  res.send("Contenido del recurso index.html");
});
```

Para registrar este encaminador secundario, tendremos algo como lo siguiente en el archivo app.js:

```
app.use("/", require("./routes/index").router);
```

Como ejemplo de módulo de rutas más completo, veamos por ejemplo el de la carpeta lógica /band:

```
//routes/band.js
//imports
import express from "express";

//api
export const router = express.Router();

//routes and handlers
router.get("/", function(req, res, next) {
  res.send("Lista de todas las bandas disponibles.");
});

router.get("/:name", function(req, res, next) {
  res.send(`Información de ${req.params.name}.`);
});
```

Observe que cada ruta se define relativa al punto de montaje indicado en el registro del encaminador en el archivo principal. Así, por ejemplo, si tenemos el siguiente registro:

```
app.use("/band", require("./routes/band").router);
```

Es como si hubiésemos definido a nivel del encaminador principal de la aplicación lo siguiente:

```
app.get("/band", function(req, res, next) {
  res.send("Lista de todas las bandas disponibles.");
});

app.get("/band/:name", function(req, res, next) {
  res.send(`Información de ${req.params.name}.`);
});
```

Archivo map.js

Existe otra manera similar de hacer las cosas. Consiste en añadir el archivo map.js al directorio de rutas, routes. Este archivo no es más que un módulo que define una función, la cual recibe como argumento el objeto aplicación en el que debe definirse los encaminadores. He aquí un ejemplo:

```
export function map(app) {
  app.use("/", require("./index").router);
  app.use("/one", require("./one").router);
  app.use("/two", require("./two").router);
```



```
}
```

Cada vez que añadamos un componente nuevo, definiremos su correspondiente módulo de rutas en el directorio routes y, a continuación, lo registraremos en la función map(). Ya no hace falta tocar el archivo app.js.

Ahora, en el archivo app.js tendremos únicamente una línea como la siguiente, para el registro de las rutas y encaminadores:

```
//routes
require("./routes/map").map(app);
```

Es muy similar a lo visto hasta ahora, pero todo lo relacionado con las rutas y los encaminadores se define en la carpeta routes. Es simplemente una manera de organizar las cosas.

