<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#routes_primer>

**[Introducción a las rutas](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes" \l "routes_primer)**

Una ruta es una sección de código Express que asocia un verbo HTTP ( GET, POST, PUT, DELETE, etc.), una ruta/patrón de URL y una función que se llama para manejar ese patrón.

Existen varias formas de crear rutas. Para este tutorial, vamos a utilizar el [express.Router](https://expressjs.com/en/guide/routing.html" \l "express-router" \t "_blank) middleware, ya que nos permite agrupar los controladores de ruta para una parte particular de un sitio y acceder a ellos mediante un prefijo de ruta común. Mantendremos todas nuestras rutas relacionadas con la biblioteca en un módulo de "catálogo" y, si agregamos rutas para gestionar cuentas de usuario u otras funciones, podemos mantenerlas agrupadas por separado.

**Nota:** Analizamos brevemente las rutas de la aplicación Express en [Introducción a Express > Creación de controladores de rutas](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction#creating_route_handlers) . Además de proporcionar un mejor soporte para la modularización (como se explica en la primera subsección a continuación), el uso de *Router* es muy similar a definir rutas directamente en el *objeto de la aplicación Express* .

El resto de esta sección proporciona una descripción general de cómo se Router pueden utilizar para definir las rutas.

**[Definición y uso de módulos de ruta independientes](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes" \l "defining_and_using_separate_route_modules)**

El código siguiente proporciona un ejemplo concreto de cómo podemos crear un módulo de ruta y luego usarlo en una aplicación *Express .*

Primero creamos rutas para una wiki en un módulo llamado **wiki.js.** El código primero importa el objeto de aplicación Express, lo usa para obtener un Router objeto y luego le agrega un par de rutas usando el get() método. Por último, el módulo exporta el Router objeto.

js

// wiki.js - Wiki route module.

const express = require("express");

const router = express.Router();

// Home page route.

router.get("/", function (req, res) {

res.send("Wiki home page");

});

// About page route.

router.get("/about", function (req, res) {

res.send("About this wiki");

});

module.exports = router;

Para utilizar el módulo de enrutador en nuestro archivo de aplicación principal, primero utilizamos require() el módulo de ruta ( **wiki.js** ). Luego, llamamos use() a la aplicación *Express* para agregar el enrutador a la ruta de manejo del middleware, especificando una ruta URL de 'wiki'.

js

const wiki = require("./wiki.js");

// …

app.use("/wiki", wiki);

Las dos rutas definidas en nuestro módulo de ruta wiki son luego accesibles desde /wiki/y /wiki/about/.

[**Funciones de ruta**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#route_functions)

Nuestro módulo anterior define un par de funciones de ruta típicas. La ruta "about" (reproducida a continuación) se define utilizando el Router.get()método, que responde solo a solicitudes HTTP GET. El primer argumento de este método es la ruta URL, mientras que el segundo es una función de devolución de llamada que se invocará si se recibe una solicitud HTTP GET con la ruta.

router.get("/about", function (req, res) {

res.send("About this wiki");

});

La devolución de llamada toma tres argumentos (generalmente nombrados como se muestra: req, res, next), que contendrán el objeto de solicitud HTTP, la respuesta HTTP y la *siguiente* función en la cadena de middleware.

**Nota:** Las funciones de enrutador son [middleware de Express](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction#using_middleware) , lo que significa que deben completar (responder) la solicitud o llamar a la nextfunción en la cadena. En el caso anterior, completamos la solicitud utilizando send(), por lo que no se utiliza el nextargumento (y elegimos no especificarlo).

La función de enrutador anterior acepta una única devolución de llamada, pero puede especificar tantos argumentos de devolución de llamada como desee o una matriz de funciones de devolución de llamada. Cada función es parte de la cadena de middleware y se llamará en el orden en que se agregue a la cadena (a menos que una función anterior complete la solicitud).

La función de devolución de llamada aquí invoca [send()](https://expressjs.com/en/4x/api.html" \l "res.send" \t "_blank)la respuesta para devolver la cadena "Acerca de esta wiki" cuando recibimos una solicitud GET con la ruta (' /about'). Hay [varios otros métodos de respuesta](https://expressjs.com/en/guide/routing.html#response-methods) para finalizar el ciclo de solicitud/respuesta. Por ejemplo, podrías invocar [res.json()](https://expressjs.com/en/4x/api.html" \l "res.json" \t "_blank)para enviar una respuesta JSON o [res.sendFile()](https://expressjs.com/en/4x/api.html" \l "res.sendFile" \t "_blank)para enviar un archivo. El método de respuesta que usaremos con más frecuencia a medida que construimos la biblioteca es [render()](https://expressjs.com/en/4x/api.html#res.render) , que crea y devuelve archivos HTML utilizando plantillas y datos. ¡Hablaremos mucho más sobre eso en un artículo posterior!

[**Verbos HTTP**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#http_verbs)

Las rutas de ejemplo anteriores utilizan el Router.get()método para responder a solicitudes HTTP GET con una ruta determinada.

También Routerproporciona métodos de ruta para todos los demás verbos HTTP, que en su mayoría se utilizan exactamente de la misma manera: post(), put(), delete(), options(), trace(), copy(), lock(), mkcol(), move(), purge(), propfind(), proppatch(), unlock(), report(), mkactivity(), checkout(), merge(), m-search(), notify(), subscribe(), unsubscribe(), patch(), search()y connect().

Por ejemplo, el código siguiente se comporta igual que la /aboutruta anterior, pero solo responde a solicitudes HTTP POST.

js

router.post("/about", (req, res) => {

res.send("About this wiki");

});

[**Rutas de ruta**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#route_paths)

Las rutas definen los puntos finales en los que se pueden realizar solicitudes. Los ejemplos que hemos visto hasta ahora han sido solo cadenas y se utilizan exactamente como están escritas: '/', '/about', '/book', '/any-random.path'.

Las rutas de ruta también pueden ser patrones de cadenas. Los patrones de cadenas utilizan una forma de sintaxis de expresión regular para definir *patrones* de puntos finales que se coincidirán. La sintaxis se muestra a continuación (tenga en cuenta que el guion ( -) y el punto ( .) se interpretan literalmente por las rutas basadas en cadenas):

* ?:El punto final debe tener 0 o 1 del carácter (o grupo) precedente, por ejemplo, una ruta de '/ab?cd'coincidirá con los puntos finales acdo abcd.
* +:El punto final debe tener 1 o más de los caracteres (o grupo) anteriores, por ejemplo, una ruta de '/ab+cd'coincidirá con los puntos finales abcd, abbcd, abbbcd, y así sucesivamente.
* \*: El punto final puede tener una cadena arbitraria donde \*se coloca el carácter. Por ejemplo, una ruta de '/ab\*cd'coincidirá con los puntos finales abcd, abXcd, abSOMErandomTEXTcd, y así sucesivamente.
* ():Agrupar coincidencias en un conjunto de caracteres para realizar otra operación, por ejemplo, '/ab(cd)?e'realizará una ?coincidencia en el grupo (cd); coincidirá con abey abcde.

Las rutas de acceso también pueden ser [expresiones regulares](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions) de JavaScript . Por ejemplo, la ruta de acceso que se muestra a continuación coincidirá con catfishy dogfish, pero no con catflap, catfishhead, y así sucesivamente. Tenga en cuenta que la ruta de una expresión regular utiliza la sintaxis de expresión regular (no es una cadena entre comillas como en los casos anteriores).

jsCopiar al portapapeles

app.get(/.\*fish$/, function (req, res) {

// …

});

**Nota:** la mayoría de nuestras rutas para LocalLibrary utilizarán cadenas y no expresiones regulares. También utilizaremos parámetros de ruta como se explica en la siguiente sección.

[**Parámetros de ruta**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#route_parameters)

Los parámetros de ruta son *segmentos de URL con nombre* que se utilizan para capturar valores en posiciones específicas de la URL. Los segmentos con nombre tienen como prefijo dos puntos y luego el nombre (p. ej., /:your\_parameter\_name/). Los valores capturados se almacenan en el req.paramsobjeto utilizando los nombres de los parámetros como claves (p. ej., req.params.your\_parameter\_name).

Por ejemplo, supongamos que una URL está codificada para contener información sobre usuarios y libros: http://localhost:3000/users/34/books/8989. Podemos extraer esta información como se muestra a continuación, con los parámetros de ruta userIdy :bookId

jsCopiar al portapapeles

app.get("/users/:userId/books/:bookId", (req, res) => {

// Access userId via: req.params.userId

// Access bookId via: req.params.bookId

res.send(req.params);

});

Los nombres de los parámetros de ruta deben estar formados por "caracteres de palabra" (AZ, az, 0-9 y \_).

**Nota:** La URL */book/create* se corresponderá con una ruta como /book/:bookId(porque :bookIdes un marcador de posición para *cualquier* cadena, por lo tanto createcoincide). Se utilizará la primera ruta que coincida con una URL entrante, por lo que si desea procesar /book/createURL específicamente, su controlador de ruta debe definirse antes que su /book/:bookIdruta.

Eso es todo lo que necesitas para comenzar a usar las rutas. Si es necesario, puedes encontrar más información en la documentación de Express: [Enrutamiento básico](https://expressjs.com/en/starter/basic-routing.html) y [Guía de enrutamiento](https://expressjs.com/en/guide/routing.html) . Las siguientes secciones muestran cómo configuraremos nuestras rutas y controladores para LocalLibrary.

[**Manejo de errores en las funciones de ruta**](https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes#handling_errors_in_the_route_functions)

Las funciones de ruta que se muestran anteriormente tienen argumentos reqy res, que representan la solicitud y la respuesta, respectivamente. Las funciones de ruta también se llaman con un tercer argumento next, que se puede utilizar para pasar errores a la cadena de middleware de Express.

El código siguiente muestra cómo funciona esto, utilizando el ejemplo de una consulta de base de datos que toma una función de devolución de llamada y devuelve un error erro algunos resultados. Si errse devuelve, nextse llama con errcomo valor en su primer parámetro (finalmente, el error se propaga a nuestro código de manejo de errores global). En caso de éxito, se devuelven los datos deseados y luego se utilizan en la respuesta.

jsCopiar al portapapeles

router.get("/about", (req, res, next) => {

About.find({}).exec((err, queryResults) => {

if (err) {

return next(err);

}

//Successful, so render

res.render("about\_view", { title: "About", list: queryResults });

});

});

Eso es una gran cantidad de código repetitivo para agregar a cada función. En cambio, para este tutorial usaremos el módulo [express-async-handler](https://www.npmjs.com/package/express-async-handler" \t "_blank)try...catch . Este define una función contenedora que oculta el bloque y el código para reenviar el error. El mismo ejemplo ahora es muy simple, porque solo necesitamos escribir código para el caso en el que asumimos que se logró el éxito:

js

// Import the module

const asyncHandler = require("express-async-handler");

exports.get(

"/about",

asyncHandler(async (req, res, next) => {

const successfulResult = await About.find({}).exec();

res.render("about\_view", { title: "About", list: successfulResult });

}),

);