**Basic routing**

***Routing*** refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Each route can have one or more handler functions, which are executed when the route is matched.

Route definition takes the following structure:

app.METHOD(PATH, HANDLER)

Where:

* app is an instance of express.
* METHOD is an [HTTP request method](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods), in lowercase.
* PATH is a path on the server.
* HANDLER is the function executed when the route is matched.

Respond with Hello World! On the homepage:

app.get('/', (req, res){

res.send('Hello World!')

})

Respond to POST request on the root route (/), the application’s home page:

app.post('/', (req, res)=>{

res.send('Got a POST request')

})

Respond to a PUT request to the /user route:

app.put('/user', function (req, res) {

res.send('Got a PUT request at /user')

})

Respond to a DELETE request to the /user route:

app.delete('/user', function (req, res) {

res.send('Got a DELETE request at /user')})

* You define routing using methods of the **Express app object** that correspond to **HTTP methods**; for example, app.get() to handle GET requests and app.post to handle POST requests. For a full list, see app.METHOD. You can also use app.all() to handle all HTTP methods and app.use() to specify middleware as the callback function.
* These routing methods specify a callback function (sometimes called “Route handler”), the application “listens” for requests that match the specified route(s) and method(s), and when it detects a match, it calls the specified callback function.
* In fact, the routing methods can have more than one callback function as arguments. With multiple callback functions, it is important to provide next as an argument to the callback function and then call next() within the body of the function to hand off control to the next callback.
* **Express supports methods that correspond to all HTTP request methods: get, post, and so on.**
* There is a special routing method, app.all(), used to load middleware functions at a path for **all** HTTP request methods. For example, the following handler is executed for requests to the route “/secret” whether using GET, POST, PUT, DELETE, or any other HTTP request method
* Route paths can be strings, string patterns, or regular expressions.

## Response methods

The methods on the response object (res) in the following table can send a response to the client, and terminate the request-response cycle. If none of these methods are called from a route handler, the client request will be left hanging.

| **Method** | **Description** |
| --- | --- |
| [res.download()](https://expressjs.com/en/4x/api.html#res.download) | Prompt a file to be downloaded. |
| [res.end()](https://expressjs.com/en/4x/api.html#res.end) | End the response process. |
| [res.json()](https://expressjs.com/en/4x/api.html#res.json) | Send a JSON response. |
| [res.jsonp()](https://expressjs.com/en/4x/api.html#res.jsonp) | Send a JSON response with JSONP support. |
| [res.redirect()](https://expressjs.com/en/4x/api.html#res.redirect) | Redirect a request. |
| [res.render()](https://expressjs.com/en/4x/api.html#res.render) | Render a view template. |
| [res.send()](https://expressjs.com/en/4x/api.html#res.send) | Send a response of various types. |
| [res.sendFile()](https://expressjs.com/en/4x/api.html#res.sendFile) | Send a file as an octet stream. |
| [res.sendStatus()](https://expressjs.com/en/4x/api.html#res.sendStatus) | Set the response status code and send its string representation as the response body. |

## app.route()

You can create chainable route handlers for a route path by using app.route(). Because the path is specified at a single location, creating modular routes is helpful, as is reducing redundancy and typos.

Here is an example of chained route handlers that are defined by using app.route().

app.route('/book')

.get(function (req, res) {

res.send('Get a random book')

})

.post(function (req, res) {

res.send('Add a book')

})

.put(function (req, res) {

res.send('Update the book')

})

**Middleware**

**Middleware** functions are functions that have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

Middleware functions can perform the following tasks:

* Execute any code.
* Make changes to the request and the response objects.
* End the request-response cycle.
* Call the next middleware in the stack.

If the current middleware function does not end the request-response cycle, it must call next() to pass control to the next middleware function. Otherwise, the request will be left hanging.

|  |  |
| --- | --- |
| https://expressjs.com/images/express-mw.png | HTTP method for which the middleware function applies.  Path (route) for which the middleware function applies.  The middleware function.  Callback argument to the middleware function, called "next" by convention.  HTTP response argument to the middleware function, called "res" by convention.  HTTP request argument to the middleware function, called "req" by convention. |

* **To load the middleware function, call app.use(),** specifying the middleware function. For example, the following code loads the myLogger middleware function before the route to the root path (/).

var express = require('express')

var app = express()

var myLogger = function (req, res, next) {

console.log('LOGGED')

next()

}

app.use(myLogger)

app.get('/', function (req, res) {

res.send('Hello World!')

})

app.listen(3000)

* If myLogger is loaded after the route to the root path, the request never reaches it and the app doesn’t print “LOGGED”, because the route handler of the root path terminates the request-response cycle.
* The middleware function myLogger simply prints a message, then passes on the request to the next middleware function in the stack by calling the next() function.

## Application-level middleware

* Bind application-level middleware to an instance of the app object by using the app.use() and app.METHOD() functions, where METHOD is the HTTP method of the request that the middleware function handles (such as GET, PUT, or POST) in lowercase.
* This example shows a middleware function with no mount path. The function is executed every time the app receives a request.

var app = express()

app.use(function (req, res, next) {

console.log('Time:', Date.now())

next()

})

app.use('/user/:id', function (req, res, next) {

console.log('Request URL:', req.originalUrl)

next()

}, function (req, res, next) {

console.log('Request Type:', req.method)

next()

})

Route handlers enable you to define multiple routes for a path. The example below defines two routes for GET requests to the /user/:id path. The second route will not cause any problems, but it will never get called because the first route ends the request-response cycle.

## Router-level middleware

Router-level middleware works in the same way as application-level middleware, except it is bound to an instance of express.Router().

var router = express.Router()

Load router-level middleware by using the router.use() and router.METHOD() functions.