Routes Introduction

Routes are one of the core concepts in Express and one of the things that make it really useful.

Routing refers to determining how an application responds to a client request, at a particular endpoint, which is called 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)
```

- app is an instance of express.
- METHOD is an HTTP request method, in lowercase.
- PATH is a path on the server.
- HANDLER is the function executed when the route is matched.

Route methods (1/2)

A route method is derived from one of the HTTP methods and it is attached to an instance of the express class.

The following code is an example of routes that are defined for the GET and the POST methods to the root of the app.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
})
```

```
// POST method route
app.post('/', function (req, res) {
```

```
    res.send('POST request to the homepage');
  })
```

Route methods (2/2)

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 supported in the HTTP module.

```
app.all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
})
```

Express allows us to attach the methods to it.

```
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
}).post('/', function (req, res) {
  res.send('POST request to the homepage');
}).all('/secret', function (req, res, next) {
  console.log('Accessing the secret section ...');
  next(); // pass control to the next handler
}).use(function(req, res, next){
    res.status(404).send('Page introuvable !');
});
```

Route paths (1/2)

The route path is used to define an endpoint where requests can be made. It is like a backend endpoint. In the Express, route paths can be:

- string
- string patterns
- regular expressions.

The characters ?, +, *, and () are subsets of their regular expression counterparts.

The hyphen (-) and the dot (.) are literally interpreted by string-based paths.

If you need to use the dollar character ($) in a path string, enclose it within ([and]). For example, the path string for requests at “/data/$book”, would be “/data/([$])book”.

- Here are some examples of route paths based on strings.

This route path will match requests to /**about**.

```
app.get('/about', function (req, res) {
  res.send('about');
})
```

This route path will match requests to /**random.text**.

```
app.get('/random.text', function (req, res) {
  res.send('random.text');
})
```

Route paths (2/2)

- Here are some examples of route paths based on **string patterns.**

This route path will match acd and abcd.

```
app.get('/ab?cd', function (req, res) {
  res.send('ab?cd');
})
```

This route path will match abcd, abbcd, abbbcd, and so on.

```
app.get('/ab+cd', function (req, res) {
  res.send('ab+cd');
})
```

This route path will match abcd, abxcd, abRANDOMcd, ab123cd, and so on.

```
app.get('/ab*cd', function (req, res) {
  res.send('ab*cd');
```

```
})
```

This route path will match /abe and /abcde.

```
app.get('/ab(cd)?e', function (req, res) {
  res.send('ab(cd)?e');
})
```

- Examples of route paths based on **regular expressions:**

This route path will match anything with an “a” in it.

```
app.get(/a/, function (req, res) {
  res.send('/a/');
})
```

This route path will match butterfly and dragonfly, but not butterflyman, dragonflyman, and so on.

```
app.get(/.*fly$/, function (req, res) {
  res.send('/.*fly$/');
})
```

Route Parameters

The route parameters are used to capture the values that are assigned to a particular position in the URL. They are called the URL segments.

The values obtained are made available in a req.params object, using the name of the route parameter specified in the path as the values' keys.

```
Route path: /users/:userId/books/:bookId
Request URL: http://localhost:3000/users/34/books/8989
req.params: { "userId": "34", "bookId": "8989" }
```

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below.

```
app.get('/users/:userId/books/:bookId', function (req, res) {
  res.send(req.params)
```

```
})
```

To have more control over the exact string that can be matched by a route parameter, you can append a regular expression in parentheses (()):

```
Route path: /user/:userId(\d+)
Request URL: http://localhost:3000/user/42
req.params: {"userId": "42"}
```

Route handlers

We can have multiple request handlers, hence the name middleware. They accept the third parameter/function *next* and calling which (next()) to be executed will switch the execution flow from one handler to the next:

For Example :

```
app.get('/api/v1/stories/:id', function(req,res, next) {
  //do authorization
  //if not authorized or there is an error
  //return next(error);
  //if authorized and no errors
  return next();
}), function(req,res, next) {
 //extract id and fetch the object from the database
 //assuming no errors, save story in the request object
  req.story = story;
  return next();
}), function(req,res) {
 //output the result of the database search
  res.send(res.story);
});
```

Another useful technique is to pass callbacks as items of an array, thanks to the inner workings of arguments. JavaScript mechanism:

```
const authAdmin = function (req, res, next) {
 ...
 return next();
}
const getUsers = function (req, res, next) {
 ...
 return next();
}
const renderUsers = function (req, res) {
 res.end();
}
const admin = [authAdmin, getUsers, renderUsers];
app.get('/admin', admin);
```

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 well as reducing redundancies 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');
  })
```

app.route()

The express.Router middleware allows us to group the route handlers for a particular part of a site together and access them using a common route-prefix.

The code below provides a concrete example of how we can create a route module and then use it in an *Express application*.

First, we create routes for a **post** in a module named **post.js**. Then, the code imports the Express application object, uses it to get a Router object, and adds a couple of routes to it using the get() method. Lastly, the module exports the Router object.

post.js - **post** route module.

```
const express = require('express');
const router = express.Router();

// post page route.
router.route('/post/:slug')
  .all(function(req, res, next) {
     // runs each time
     // we can fetch the post by id from the database
   })
  .get(function(req, res, next) {
     //render post
   })
   .put(function(req, res, next) {
     //update post
   })
   .post(function(req, res, next) {
     //create new comment
   })
   .del(function(req, res, next) {
     //remove post
   });

module.exports = router
```

To use the router module in our main app file we first `require()` the route module (post.js). We then call `use()` on the Express application to add the router to the middleware handling path while specifying the URL path of '/blog'.

```
const post = require('./post');

app.use('/blog, post);
```

The app will now be able to handle requests to /blog/post.

Previous next