




CodeSchool building-blocks-of-express-js Level 5

5.2 | Route Instance 250 PTS

Let's rewrite our cities routes using a Route Instance.

Help Me 

Task 1/5  

1

2

3

4

5

Create a new *Route Instance* for the `"/cities"` URL path and assign it to the `citiesRoute` variable.

```
var citiesRoute = app.route("/cities");
```

Task 2/5  



2

3

4

5

Move the code from our previous `app.get()` route to a new **GET** route on the `citiesRoute` object.

```
8 var citiesRoute = app.route("/cities");
9
10 // GET route for /cities
11 citiesRoute.get(function (request, response) {});
12
13 // POST route for /cities
14 citiesRoute.post(parseUrlencoded, function (request, response) {});
```

Task 3/5 < >



Move `app.post()` to `citiesRoute`.

```
10 // GET route for /cities
11 citiesRoute.get(function (request, response) {});
18
19 // POST route for /cities
20 citiesRoute.post(parseUrlencoded, function (request, response) {});
28
```

Task 4/5 < >



Now, let's get rid of the `citiesRoute` temporary variable and use chaining function calls.

```
8 app.route("/cities")
9
10 // GET route for /cities
11 .get(function (request, response) {});
18
19 // POST route for /cities
20 .post(parseUrlencoded, function (request, response) {});
28
```

Task 5/5 < >

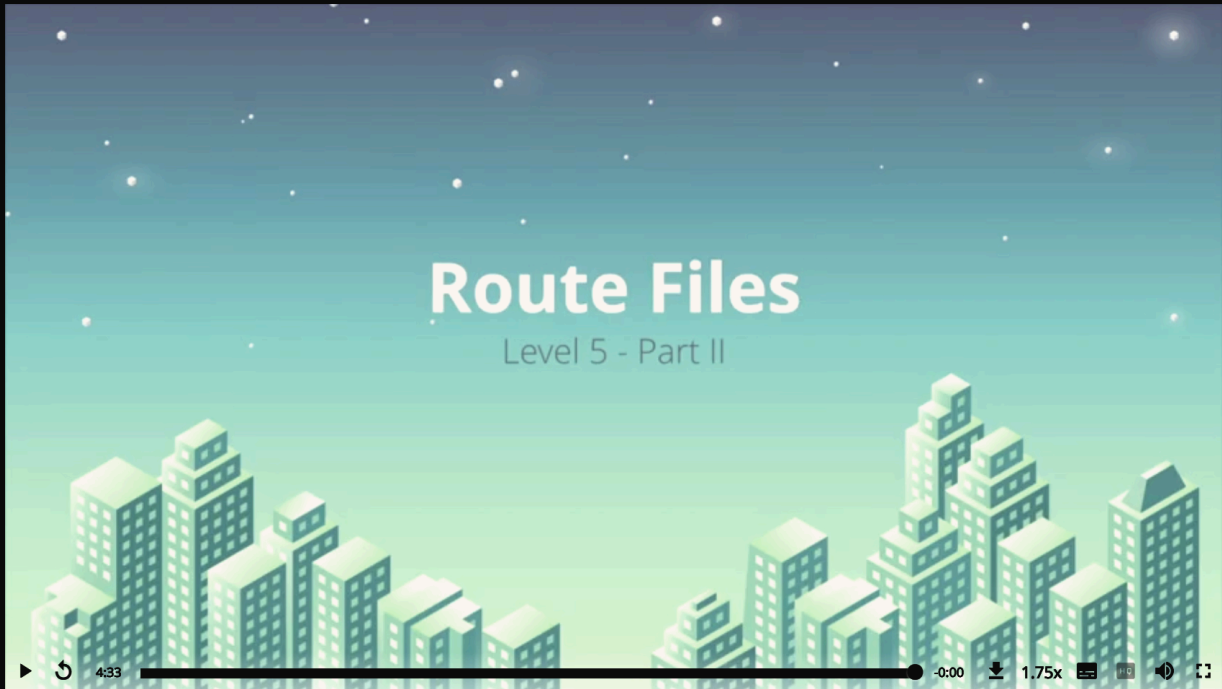


Finally, let's move the old routes for the `'/cities/:name'` URL path to use the new Route Instance API.

app.js

```
1 var express = require('express');
2 var app = express();
3 var bodyParser = require('body-parser');
4 var parseUrlencoded = bodyParser.urlencoded({ extended: false });
5 // In memory store for the cities in our application
6 var cities = {};
7
8 app.route("/cities")
9
10 // GET route for /cities
11 .get(function (request, response) {});
12
13 // POST route for /cities
14 .post(parseUrlencoded, function (request, response) {});
15
16 app.route("/cities/:name")
17 // GET route for /cities/:name
18 .get(function (request, response) {});
19
20 // DELETE route for /cities/:name
21 .delete(function (request, response) {});
22
23 // Searches for keyword in description and returns the city
24 function citySearch(keyword) {}
25
26 // Adds a new city to the in memory store
27 function createCity(name, description) {}
28
29 app.listen(3000);
```

5.3 | Route Files





5.4 | Using a Router Instance

250 PTS

Let's refactor `app.js` to use a `Router` object.

Help Me 

Task 1/4  

1

2

3

4

Create a new router object and assign it to the `router` variable.

```
20  
21 var router = express.Router();  
22
```

Task 2/4 < >



When we are done, our `router` will be mounted on the `/cities` path. With this in mind, change `app.route('/cities')` to use `router` and map requests to the root path.

```
router.route('/')  
  .get(function (request, response) {↔})  
  
  .post(parseUrlencoded, function (request, response) {↔});
```

Task 3/4 < >



Likewise, let's move our `'/cities/:name'` route to our `router`. Remember to update the path.

```
42  
43 router.route('/:name')  
44 ▶ .get(function (request, response) {↔})  
52
```

Task 4/4 < >



Our `router` is now ready to be used by `app`. Mount our new `router` under the `/cities` path.

app.js

```
1 var express = require('express');
2 var app = express();
3
4 var bodyParser = require('body-parser');
5 var parseUrlencoded = bodyParser.urlencoded({ extended: false });
6
7 // In memory store for the
8 // cities in our application
9 var cities = {};
16
17 app.param('name', function (request, response, next) {});
20
21 var router = express.Router();
22
23 router.route('/')
24 .get(function (request, response) {})
31
32 .post(parseUrlencoded, function (request, response) {});
40
41 router.route('/:name')
42 .get(function (request, response) {})
50
51 .delete(function (request, response) {});
59
60 app.use('/cities', router);
61
62 // Searches for keyword in description
63 // and returns the city
64 function citySearch(keyword) {}
72
73 // Adds a new city to the
74 // in memory store
75 function createCity(name, description){}
79
80 // Uppercase the city name.
81 function parseCityName(name){}
85
86 app.listen(3000);
87
```

5.5 | All HTTP Verbs

250 PTS

What function would you call to match all HTTP verbs?

☐ `app.use()`

☒ `app.all()`

☐ `app.route()`

5.6 | Using All

250 PTS

Let's use the `app.all()` method to handle the `name` parameter instead of `app.param()`.

Help Me 

Task 1/3 < >



2

3

Add a call to `all()` for our router's `('/:name')` route. Pass a callback function that accepts `request`, `response`, and `next`.

```
▼ .all(function(request, response, next){  
  
  })
```

Task 2/3 < >



2

3

Now let's take our logic from the callback function passed to `app.param()` and move it to our `all()` callback.

```
▼ .all(function(request, response, next){  
  request.cityName = parseCityName(request.params.name);  
})
```

Task 3/3 < >



3

Now remove the original call to `app.param()`.

```
app.js  
1 var express = require('express');  
2 var app = express();  
3  
4 var bodyParser = require('body-parser');  
5 ▼ var parseUrlencoded = bodyParser.urlencoded({ extended: false });  
6  
7 // In memory store for the cities in our application  
8 ▶ var cities = {};  
15  
16 // Searches for keyword in description and returns the city  
17 ▶ function citySearch(keyword) {}  
25  
26 // Adds a new city to the in memory store  
27 ▶ function createCity(name, description) {}
```



```

31
32 // Uppercase the city name.
33 ▶ function parseCityName(name) {↵}
37
38 var router = express.Router();
39
40 router.route('/')
41 ▶ .get(function (request, response) {↵})
48
49 ▶ .post(parseUrlencoded, function (request, response) {↵});
57
58 router.route('/:name')
59 ▼ .all(function(request, response, next){
60   request.cityName = parseCityName(request.params.name);
61 })
62
63 ▼ .get(function (request, response) {
64 ▼   var cityInfo = cities[↵];
65 ▼   if(cityInfo) {
66     response.json(cityInfo);
67 ▼   } else {
68     response.status(404).json("City not found");
69   }
70 })
71
72 ▼ .delete(function (request, response) {
73 ▶   if(cities[request.cityName]) {↵} else {
77     response.sendStatus(404);
78   }
79 });
80
81 app.use('/cities', router);
82
83 app.listen(3000);
84


```

5.7 Creating a Router Module

250 PTS

Our single application file is growing too long. It's time we extract our routes to a separate Node module under the **routes** folder.

Help Me 

Task 1/3  

1 2 3

Move our **router** and its supporting code from **app.js** to **routes/cities.js**.

```
app.js routes/cities.js
1 var express = require('express');
2
3 var bodyParser = require('body-parser');
4 var parseUrlencoded = bodyParser.urlencoded({ extended: false });
5
6 // In memory store for the
7 // cities in our application
8 var cities = {
9   'Lotopia': 'Rough and mountainous',
10  'Caspiana': 'Sky-top island',
11  'Indigo': 'Vibrant and thriving',
12  'Paradise': 'Lush, green plantation',
13  'Flotilla': 'Bustling urban oasis'
14 };
15
16 var router = express.Router();
17
18 router.route('/')
19 .get(function (request, response) {
20   if(request.query.search){
21     response.json(citySearch(request.query.search));
22   }else{
23     response.json(cities);
24   }
25 })
26
27 .post(parseUrlencoded, function (request, response) {
28   if(request.body.description.length > 4){
29     var city = createCity(request.body.name, request.body.description);
30     response.status(201).json(city);
31   }else{
32     response.status(400).json('Invalid City');
33   }
34 });
35
```

```

36 router.route('/:name')
37 .all(function (request, response, next) {
38     request.cityName = parseCityName(request.params.name);
39 })
40
41 .get(function (request, response) {
42     var cityInfo = cities[request.cityName];
43     if(cityInfo){
44         response.json(cityInfo);
45     }else{
46         response.status(404).json("City not found");
47     }
48 })
49
50 .delete(function (request, response) {
51     if(cities[request.cityName]){
52         delete cities[request.cityName];
53         response.sendStatus(200);
54     }else{
55         response.sendStatus(404);
56     }
57 });
58
59 // Searches for keyword in description
60 // and returns the city
61 function citySearch(keyword) {
62     var regexp = RegExp(keyword, 'i');
63     var result = cities.filter(function (city) {
64         return city.match(regexp);
65     });
66
67     return result;
68 }
69
70 // Adds a new city to the
71 // in memory store
72 function createCity(name, description){
73     cities[name] = description;
74     return name;
75 }
76
77 // Uppercase the city name.
78 function parseCityName(name){
79     var parsedName = name[0].toUpperCase() + name.slice(1).toLowerCase();
80     return parsedName;
81 }
82

```

Task 2/3 < >



export our `router` object so other files can have access to it. Remember, Node - therefore Express - uses the *CommonJS* module specification.

```
77 // Uppercase the city name.  
78 function parseCityName(name){  
79   var parsedName = name[0].toUpperCase()  
80   return parsedName;  
81 }  
82  
83 module.exports = router;
```

Task 3/3 < >



Our cities routes module is now ready to be used from *app.js*. Require the new *routes/cities* module from `app.js` and assign it to a variable called `router`;

app.js

routes/cities.js

```
1 var express = require('express');
2 var app = express();
3
4 var router = require("routes/cities");
5
6 app.use('/cities', router);
7 app.listen(3000);
8
```

