Express | GET Methods - Route Params & Query String

Learning Goals

After this lesson you will be able to:

- Understand how GET methods works
- Learn how to send data from the browser to the server
- Learn how Route Params works
- Learn how Query Strings works
- Understand the difference between route and query params

Introduction

Simple web applications are a one-way communication from the browser to the server. The browser emits a request, and the server sends back a resource (HTML, CSS, JS), this is what we call `GET` requests. You can think of `GET` requests as an HTTP request where you want to *get* something.

In this lesson, we are going to learn how also to send data from the browser to the server, and how the server receives those parameters and uses them to perform operations.

Set the environment

For this learning we are going to need an app to practice all the concepts, go ahead and create a folder `express-get-params` and an `app.js` file inside it. We will also need some **node modules**, so you should run the `npm init` command.

After creating the `app.js` we need the `views` directory and an `index.hbs` file inside.

```
$ mkdir express-get-params; cd express-get-params
$ npm init
$ touch app.js
$ mkdir views; cd views; touch index.hbs
```

```
$ code .
```

On the `app.js` file, copy/paste the following code, and install the `express` and `hbs` modules using **`npm install express hbs`**.

```
// app.js
const express = require('express')
const app     = express()
const hbs     = require('hbs')

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'hbs');

app.get('/', function (req, res) {
  console.log(req);
})

app.listen(3000, () => console.log('App listening on port 3000!'))
```

And finally, let's add `nodemon`, so we do not need to restart the server for every change. On the `package.json` file, add the following line inside the **scripts** object:

```
"start": "nodemon app.js",
```

The package.json should look like this:

```
{
  "name": "get-params",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.2",
    "hbs": "~4.0.1"
  }
}
```

> The version may not be the same! Don't worry about that!

We are ready! Go to the terminal and run the **`npm start`** command. You should see the following:

```
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Example app listening on port 3000!
```

Our server is up and running!

GET Request

Every time we navigate to an URL, we are making a GET request to the server, and there are some ways we can send data to our server to use that info. We will learn two techniques of delivering data to our server through GET requests:

- Route Params
- Query Params

Route Params

Route parameters are **named URL segments** that are used to capture the values specified at their position in the URL. The obtained values are populated in the **req.params** object, with the name of the route parameter specified in the path as **their respective keys**.

To define routes with route parameters, simply specify the route parameters in the path of the route as shown below. Let's add the following code to our `app.js` file:

```
app.get('/users/:username', (req, res, next) => {
  res.send(req.params);
})
```

Let's navigate to **`http://localhost:3000/users/techgrounds`** on our browser!

{ "username": "techgrounds" }

Notice that the `:username` is populated on the **req.params** Object as a `key` and the `string` we send on that position of the URL is the `value` of that `key`. That means that we can replace the `:username` with anything we want. For example, let's receive a `bookId` this time:

```
app.get('/books/:bookId', (req, res, next) => {
  res.send(req.params);
})
```

Let's navigate to **`http://localhost:3000/books/13112kj3h$j3h1jk2`** on our browser!

```
{"bookId":"13112kj3h$j3h1jk2"}
```

More Route Params

Sometime we will need to send more than one parameter, in those cases we can do something like the following:

```
app.get('/users/:username/books/:bookId', (req, res, next) => {
  res.send(req.params)
})
```

So, if now we navigate to **`http://localhost:3000/users/techgrounds/books/8989`** on our browser, we should see the following:

{ "username": "techgrounds", "bookId": "8989" }

Query String

Sometimes we will find useful to send params on the URL as a string, and for that purpose Query Params are fantastic.

In simple terms, the **query string** is the part of a URL after the question mark `?` and usually contains key value pairs separated by `&` and `=`. These **key/value** pairs can be used by the server as arguments to query a database, or maybe to filter results.

You will receive the info on the `req.query` object of our routes.

Let's check an example:

Add the following code on the `app.js` file:

```
app.get('/search', (req, res, next) => {
  res.send(req.query)
})
```

Same as route params, **query params** are key-value pairs object. Everything that comes after the `?` it's going to be pair as key-value using the `=` as a separator.

Go ahead and navigate to **`http://localhost:3000/search?city=Barcelona`**

So in our case we will have:

```
console.log(req.query);
// => { "city" : "Barcelona" }
```

It's super important to notice the difference between **route params** and **query strings**. On query strings, the URL we create on the `app.js` does not include the params.

More Query String

When we are doing some searches or filters, is common to have more info to send to the server. In that case, we can use the `&` to attach more params.

If we want to add a `start-date` for example, let's navigate to **`http://localhost:3000/search?city=Barcelona&start-date=2018-01-18`**

```
{ "city" : "Barcelona", "start-date" : "2018-01-18" }
```

Query String from Forms

Super-frequent use for the `query string` is to use them when sending info from a form, for example, a user search. Let's imagine we are doing an app where the users can search for Hotel Rooms to rent in different cities.

In our `app.js` let's add the following code:

```
app.get('/', (req, res, next) => {
  res.render('index');
})
```

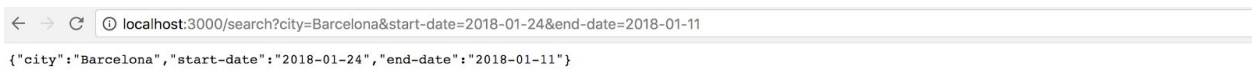
And on our `index.hbs` file, add the following:

```
<form action="/search" method="GET">
  <label for="">City</label>
  <input type="text" name="city" id="">
  <label for="">Start-Date</label>
  <input type="date" name="start-date" id="">
  <label for="">End-Date</label>
  <input type="date" name="end-date" id="">
  <button type="submit">SEARCH</button>
</form>
```

Wait for a second! Let's review some key stuff about our code:

- The `form` method is **GET**. By default, it will always be a **GET** method.
- The `form` action is **/search**, means that when we click on the **SEARCH** button, we will make a **GET request to `/search` route**
- On each input, what we put in the `name` field will be each of the `keys` on the `req.query` object, and what's on the input will be the `value`.

So, let's navigate to **`http://localhost:3000`** , complete the form and click on the **SEARCH** button. We should see something like this:



Awesome huh?

Cheat Table 😉

Given the URL: `http://localhost:3000/products/1345?show=reviews`, and the route `/products/:id`, we can destructure the `req` object in the following fields:

HTTP Request	**Express `req` object**
Method	req.method // `GET`
URL Path	req.path // `/products/1345`
URL Params	req.params.id // `1345`
URL Query String	req.query.show // `reviews`
All headers	req.headers['Accept-Language'] // `"en-US,en;q=0.9`

Route Params vs. Query Strings

Concepts must be more evident by now, especially knowing the difference between both **route** and **query** params, but for discussing a little more about the subject we list some examples, and you have to choose which approach will fit better:

- Visit a user profile on Facebook
- Search an apartment on Airbnb
- See flights Madrid-Miami on 21-Oct on Skyscanner
- Check news on CNN web page

Summary

We learned how we could send info to the server using GET method. Using **route** params or **query** strings we should be able to pass all the info we need on our web apps.

We also have seen the difference between both ways of passing the info, and how we need to structure our code to get that info on each of both approaches.