COMP 3123

What are Routes?

- Routing determine the way in which an application responds to a client request to a particular endpoint.
- For example, a client can make a GET, POST, PUT or DELETE http request for various URL such as the ones shown below;
  - *http://localhost:3000/Books*
  - *http://localhost:3000/Students*
- If a GET request is made for the first URL, then the response should ideally be a list of books.
- If the GET request is made for the second URL, then the response should ideally be a list of Students.
- ***So based on the URL which is accessed, a different functionality on the webserver will be invoked, and accordingly, the response will be sent to the client. This is the concept of routing.***

Syntax :

```
app.METHOD(PATH, HANDLER)
```

Using Routes (Get)

- We have seen a basic application which serves HTTP request for the homepage. 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).
- Here is a simple example which passes two values using **HTML FORM GET** method. We are going to use **process_get** router inside server.js to handle this input.

```
<html>
<body>
    <form action = "http://127.0.0.1:8081/process_get" method = "GET">
       First Name: <input type = "text" name = "first_name">  <br>
       Last Name: <input type = "text" name = "last_name">
       <input type = "submit" value = "Submit">
    </form>

  </body>
</html>
```

```
var express = require('express');
var app = express();
app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
  res.sendFile( __dirname + "/" + "index.htm" );
})
app.get('/process_get', function (req, res) {
  // Prepare output in JSON format
  response = {
    first_name:req.query.first_name,
    last_name:req.query.last_name
  };
  console.log(response);
  res.end(JSON.stringify(response));
})
var server = app.listen(8081, function () {
  var host = server.address().address
  var port = server.address().port
  console.log("Example app listening at http://%s:%s", host, port)
})
```

Extracting Query and Route Parameters

- **req.query:** directly access the parsed query string parameters
- **req.params:** directly access the parsed route parameters from the path

- A query *string* is the part of a URL (Uniform Resource Locater) *after* the question mark (?).

  https://stackabuse.com/?page=2&limit=3

- The query *parameters* are the actual key-value pairs like page and limit with values of 2 and 3, respectively.

```
app.get('/', function(req, res)
{
        let page = req.query.page;
        let limit = req.query.limit;
});
```

- In any web application another common way to structure your URLs is to place information within the actual URL path, which are simply called route parameters in Express.

  https://stackabuse.com/tag/pritam

- We tell Express that our route is **/tag/:id**, where **:id** is a placeholder for anything. It could be a string or a number.
- So whatever is passed in that part of the path is set as the id parameter.

```
app.get('/tag/:id', function(req, res)
{
        let page = req.params.id;
});
```

Query-strings-and-parameters

Using Routes (POST)

Here is a simple example which passes two values using HTML FORM POST method. We are going to use **process_get** router inside **server.js** to handle this input.

```
<html>
  <body>
    <form action = "http://127.0.0.1:8081/process_post" method = "POST">
       First Name: <input type = "text" name = "first_name"> <br>
       Last Name: <input type = "text" name = "last_name">
       <input type = "submit" value = "Submit">
    </form>
  </body>
</html>
```

```
var express = require('express');
var app = express();
var bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser
var urlencodedParser = bodyParser.urlencoded({ extended: false })
app.use(express.static('public'));
app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})

app.post('/process_post', urlencodedParser, function (req, res) {

   // Prepare output in JSON format
   response = {
      first_name:req.body.first_name,
      last_name:req.body.last_name
   };

   console.log(response);

   res.end(JSON.stringify(response));

})


var server = app.listen(8081, function () {
   var host = server.address().address
   var port = server.address().port
   console.log("Example app listening at http://%s:%s", host, port)
})
```

Using Routes (PUT/DELETE)

- The PUT route is almost the same as the POST route. We will be specifying the ID for the object that'll be updated/created.
- This route will update the object with new details if it exists. If it doesn't exist, it will create a new object.

Using Routes (delete)

```
router.delete('/:id', function(req, res){
   var removeIndex = movies.map(function(movie){
      return movie.id;
   }).indexOf(req.params.id); //Gets us the index of movie with given id.


   if(removeIndex === -1){
      res.json({message: "Not found"});
   } else {
      movies.splice(removeIndex, 1);
      res.send({message: "Movie id " + req.params.id + " removed."});
   }
});
```

Working with route params

- The parameters of **router.param()** are **name** and a **function**.
- Where **name** is the actual name of parameter and **function** is the callback function.
- Basically router.param() function triggers the callback function whenever user routes to the parameter.
- This callback function will be called for only single time in request response cycle, even if user routes to the parameter multiple times.

Syntax:

router.param(name, function)

Parameters of callback function are:

req – the request object
res – the response object
next – the next middleware function
id – the value of name parameter

Using Middleware

- You can also use app.all() to handle all HTTP methods and app.use() to specify middleware as the callback function (See Using middleware for details).

- 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 function 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.

- An Express application can use the following types of middleware:
  - Application-level middleware
  - Router-level middleware
  - Error-handling middleware
  - Built-in middleware
  - Third-party middleware

For examples click middleware

Using Regular Expression

```
/**
 * "/abc" - handles /abc
 * "/ab?cd" - handles /acd or /abcd
 * "/ab+cd" - handles /abcd, /abbbcd, /abbbbbbbcd, etc
 * "/ab*cd" - "/ab" + anything + "cd"
 * /a/ - RegExp: anything that contains "a"
 * /.*man$/ - RegExp: anything that ends with "man"
 *
 */
```

ExpressJS - Serving static files

- Static files are files that clients download as they are from the server.
- Create a new directory, **public** express, by default does not allow you to serve static files. You need to enable it using the following built-in ***middleware***.
- Express provides a built-in middleware **express.static** to serve static files, such as images, CSS, JavaScript, etc.
- You simply need to pass the name of the directory where you keep your static assets, to the express.static middleware to start serving the files directly. For example, if you keep your images, CSS, and JavaScript files in a directory named public.
- Example- ***app.use(express.static('public'));***

Multiple Static Directories

- We can also set multiple static assets directories using the following program −

```
var express = require('express');

var app = express();

app.use(express.static('public'));

app.use(express.static('images'));

app.listen(3000);
```

```
node_modules
server.js
public/
public/images
public/images/logo.png
```

Virtual Path Prefix

We can also provide a path prefix for serving static files. For example, *if you want to provide a path prefix like '/static', you need to include the following code in your index.js file* −

```
var express = require('express');
var app = express();
app.use('/static', express.static('public'));
app.listen(3000);
```

Now whenever you need to include a file, for example, a script file called main.js residing in your public directory, use the following script tag −

```
<script src = "/static/main.js" />
```

This technique can come in handy when providing multiple directories as static files. These prefixes can help distinguish between multiple directories.

Thank You