**Experiment No.: 07**

**Title:** Handling Routes in Expressjs

**Objectives:**

1. To demonstrate routing in express to handle request according to the method and URL.

**Theory:**

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Routing in express 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, PUT, DELTE, PATCH).

Each route can have one or more handler functions, which are executed when the route is matched. These routing methods specify a callback function called when the application receives a request to the specified route (endpoint) and HTTP method. In other words, 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.

Route definition takes the following structure:

*app.METHOD(PATH, HANDLER)*

Where:

* 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.

The following examples illustrate defining simple routes.

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', (req, res) => {

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

})

Respond to a DELETE request to the /user route:

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

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

})

**Route paths**

Route paths, in combination with a request method, define the endpoints at which requests can be made. Route paths can be strings, string patterns, or regular expressions.

The characters ?, +, \*, and () are subsets of their regular expression counterparts. The hyphen (-) and the dot (.) are interpreted literally by string-based paths.

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

**Routing Parameters**

Route parameters are named URL segments that are used to capture the values specified at their position in the URL. The captured values are populated in the req.params object, with the name of the route parameter specified in the path as their respective 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', (req, res) => {

res.send(req.params)

})

Since the hyphen (-) and the dot (.) are interpreted literally, they can be used along with route parameters for useful purposes.

Route path: /flights/:from-:to

Request URL: http://localhost:3000/flights/LAX-SFO

req.params: { "from": "LAX", "to": "SFO" }

Route path: /plantae/:genus.:species

Request URL: http://localhost:3000/plantae/Prunus.persica

req.params: { "genus": "Prunus", "species": "persica" }

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"}

**Key Concept:** Routing, Express, Method, Callbacks

**Steps:**

1. Create folder with name myapp and execute npm init in the command prompt at the folder location

2. Create index.js file with routing and listen on port no 3000

3. Use postman application to send the request to <http://localhost:3000> with method GET, and POST.

4. Verify the result.