**Experiment No.: 08**

**Title:** Demonstrate middlewares from Expressjs

**Objectives:**

1. To demonstrate middlewares in expressjs.

**Theory:**

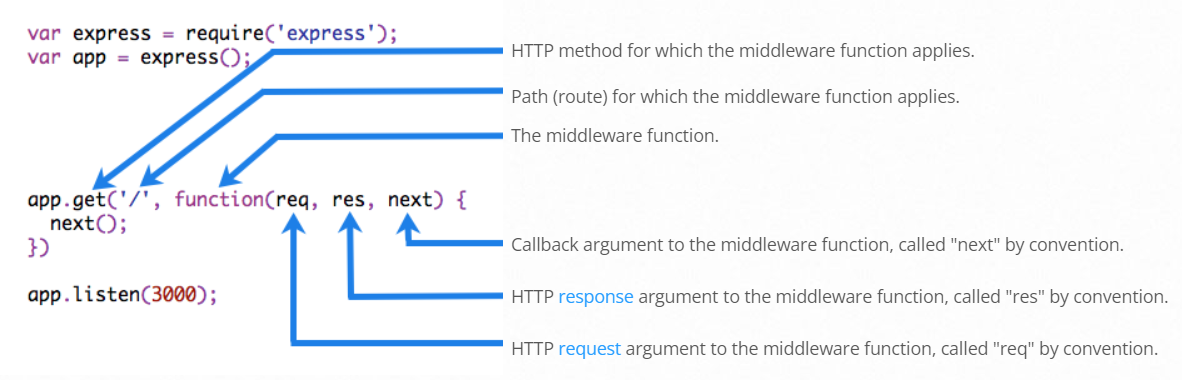
Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application’s request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

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 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.

The following figure shows the elements of a middleware function call:



**Examples:**

**1. Middleware function myLogger**

Here is a simple example of a middleware function called “myLogger”. This function just prints “LOGGED” when a request to the app passes through it. The middleware function is assigned to a variable named myLogger.

*const myLogger = function (req, res, next) {*

*console.log('LOGGED')*

*next()*

*}*

To load the middleware function, call app.use(), specifying the middleware function. For example, the following code loads the myLogger middleware function before the route to the root path (/).

*const express = require('express')*

*const app = express()*

*const myLogger = function (req, res, next) {*

*console.log('LOGGED')*

*next()*

*}*

*app.use(myLogger)*

*app.get('/', (req, res) => {*

*res.send('Hello World!')*

*})*

*app.listen(3000)*

Every time the app receives a request, it prints the message “LOGGED” to the terminal. The order of middleware loading is important: middleware functions that are loaded first are also executed first.

If myLogger is loaded after the route to the root path, the request never reaches it and the app doesn’t print “LOGGED”, because the route handler of the root path terminates the request-response cycle.

The middleware function myLogger simply prints a message, then passes on the request to the next middleware function in the stack by calling the next() function.

**2. Middleware function requestTime**

Next, we’ll create a middleware function called “requestTime” and add a property called requestTime to the request object.

*const requestTime = function (req, res, next) {*

*req.requestTime = Date.now()*

*next()*

*}*

The app now uses the requestTime middleware function. Also, the callback function of the root path route uses the property that the middleware function adds to req (the request object).

*const express = require('express')*

*const app = express()*

*const requestTime = function (req, res, next) {*

*req.requestTime = Date.now()*

*next()*

*}*

*app.use(requestTime)*

*app.get('/', (req, res) => {*

*let responseText = 'Hello World!<br>'*

*responseText += `<small>Requested at: ${req.requestTime}</small>`*

*res.send(responseText)*

*})*

*app.listen(3000)*

When you make a request to the root of the app, the app now displays the timestamp of your request in the browser

**Key Concept:** Middlewares, 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 middlewares 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.