Node-L8: Express Middleware

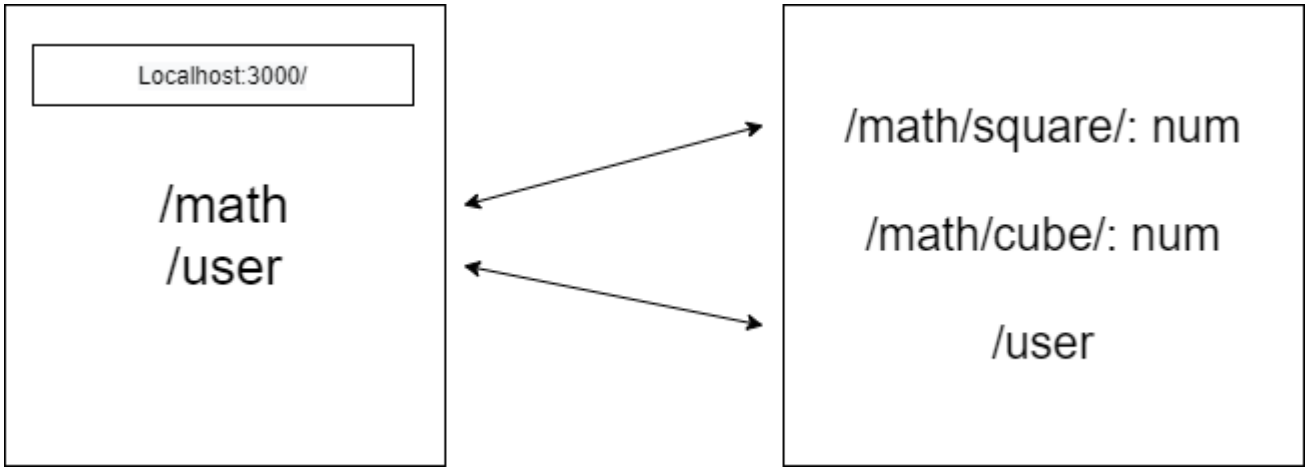
Request Response Cycle:

Yesterday's session, we had different routes and the basic idea was to separate math operations and user operations in different files and run them from one **index.js** file.

In our client side, we have the address bar, and there are different routes that a user can go for. In the NodeJS Server, we have one specific route for math and one for user. So, when we input localhost:3000/math, a set of functionality gets triggered.

The base URL is localhost:3000. After that we have sub-URL or different paths. So, we have **'/math'** and **'/user'**. These are the two paths we are handling. Internally, in our server we have configured to handle this particular URL matching. As long as the server is running, we have defined our routes or paths for different cases. In server anything which matches, **/math/square/:num** or **/math/cube/:num** or **/users**.

Whenever user hits a particular URL in the client side, based on that the matching happens. Whenever the particular URL is written in any address bar or browser, that will get matched to the server where we defined that specific route. So, localhost:3000 get converted to 127.0.0.1



Going into the server we will search if the particular path exists or not. We have only defined /math/square/:num by splitting the router. So, this route gets matched into the server and whatever function get matched to the URL, that function gets executed. Similarly if it matches with **/math/cube/:num** then cube() would get executed.

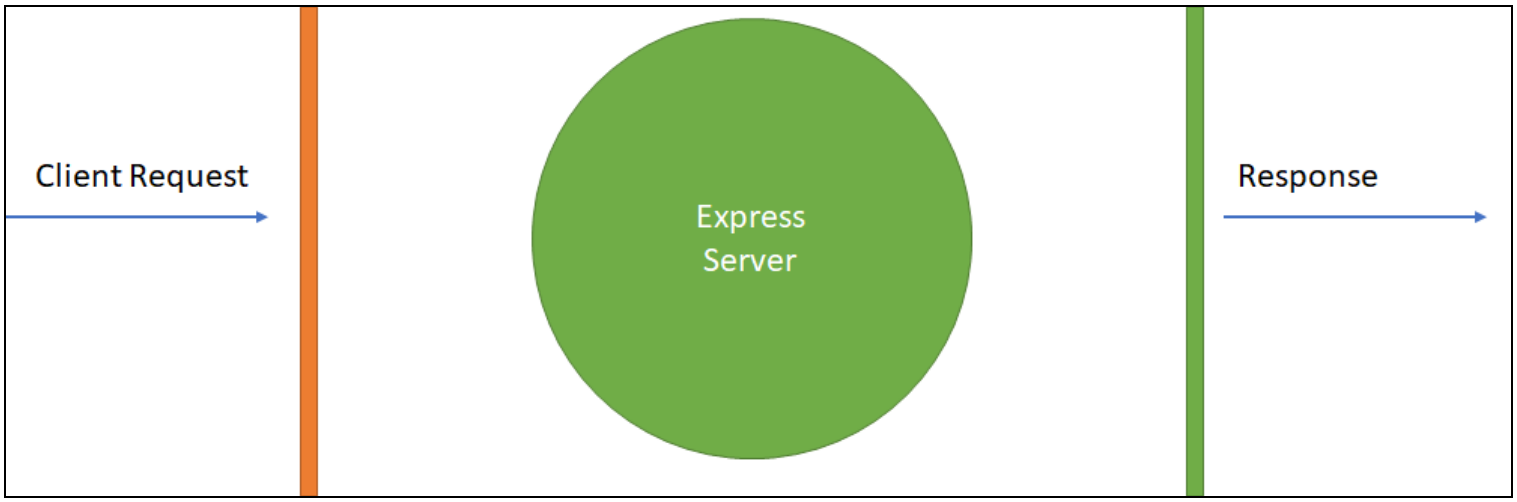
Based on the string pattern, it will execute the function associated with it. This is called the request **response cycle**.

The idea behind a router is to keep things simple.

In brief, client creates a request, sends it over the internet and the server understands that request, do some process accordingly and send the response back to the client. Request-Response Cycle.

Middleware:

There are some cases wherein we need some pre-processing before we handle it. In math operations, we are assuming that the user will give us some number, but what if the user gives a text? So, what can be done to pre-process this request before moving on to the final step? This pre-process work that is being done is what we call Middleware.



Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.

1. As name suggests it comes in middle of something and that is request and response cycle
2. Middleware has access to **request** and **response** object
3. Middleware has access to **next** function of request-response life cycle

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.

We don't have to specify npm install express and axios, because all the data is already listed in our package.json file.

Middleware is a function. We want to tell our express server, that before it goes to the final response cycle, it has to do pre-processing. This we can do using a function. We will create a simple function that it will log in the console every time a request occurred and gives info to us every time during the next change in cycle.

```
function logger() {
    console.log('Request came from the client')
}
```

This is the function that should be executed every time a request comes. Now, we need to tell the express server to use this function. We cannot keep calling the function everywhere and it will cause lot of repetition. Just as we defined a router object for sub-routes, similarly, we will use an app object

```
function logger() {
    console.log('Request came from the client')
}

app.use(logger)
```

We notice that the function is getting executed irrespective of the route that is selected. We have defined one function and express is executing that function whenever a request is coming from the client.

Let us define one more middle ware

```
function logger() {
    console.log('Request came from the client')
}

function prePrcocess() {
    console.log('Pre processing the request')
}

app.use(logger)
app.user(prePrcocess)
```

Now, we have two middleware functions.

The page is not loading and we not getting a response from the server and we see only the logger() getting executed and new middleware function is not being executed. The problem is, the first middleware will always get called. If we reverse the order from logger and preProcess to preProcess and logger then preProcess will be logged in. We need to write some code inside all the middleware function to tell everything is fine

and go to the next middleware function. And if it is the last middleware then it needs to go to the final handler and get the response.

We will re-order so that logger() is executed and then preProcess() is executed. When express calls this middleware function, it passes three parameters to the same function. So, both the middleware function are getting three parameters being passed to them.

- The **first parameter** is the **request** object.
- The **second parameter** is the **response** object.
- The **third parameter** is the **next** function.

So after having the logger() executed, we will call the next function.

```
function logger(req, res, next) {
    console.log('Request came from the client')
    next()
}
```

Now when we call the server,

We can see in terminal that the logger() and preProcess() are being executed. The moment we use next() in the logger(), it will be referring to the next function in the middleware chain. So, the next() would be referring to the preProcess() in our case.

In the preProcess() also we have to call the next function. So,

```
function prePrcocess(req, res, next) {
    console.log('Pre processing the request')
    next()
}
```

When we do this, then we would get the response appropriately.

The logger() function gets executed, then with the next(), the preProcess() is getting executed. In preProcess() as well, we have given the third parameter as a function and it will be getting referred to the response.json(thinuser), will be getting executed.

Similarly, if we use math/square/:12 → 144

In the console we can see, the two console.log statements being logged in.

This is the whole chain of middleware cycle.

So, the complete program in index.js file would be as

```
const express = require('express')
const app = express();
const mathRouter = require('./routes/math');
const userRouter = require('./routes/user');

function logger(req, res, next) {
    console.log('Request came from the client')
    next()
}

function prePrcocess(req, res, next) {
    console.log('Pre processing the request')
    next()
}

app.use(logger)
app.user(prePrcocess)

app.use('./math', mathRouter)
app.use('./user', userRouter)


app.listen(3000, () => console.log('server started'))
```

All we need to do is log some information about the request itself. So to do it,

```
function logger(req, res, next) {
    console.log('Request came from the client')
    console.log(`Path: ${req.path} Method: ${req.method} Time:
${new Date().toLocaleString()}`)
    next()
}
```

Once this is logged then it move to the next().

We will be getting the output as

Path: /math/square/12 **Method**: GET **Time**: 4/14/2021, 10:20:33 PM

To get the base-url and url we can use,

```
function logger(req, res, next) {
    console.log('Request came from the client')
    console.log(req.baseUrl)
    console.log(req.url)
    console.log(`Path: ${req.path} Method: ${req.method} Time:
${new Date().toLocaleString()}`)
    next()
}
```

You will get baseUrl if you have hosted it. and for req.url → /math/square/: 12

So, for every request, the logger() would execute. Coming to the preProcess(),

```
function prePrcocess(req, res, next) {
    console.log('Pre processing the request')
    if (req.path.includes('math')){
        const {} = req.params

        const isNan = Number(num) //Nan

        if(isNan){
            res.send('Wrong Parameters')
            return
        }else {
            next()
        }
    }
}
```

We are not sure about the if-statement being executed. So, we can debug it and check it got executed or not.

We see that the preProcess is not able to distinguish the give path as path parameter. What we can do in order to handle this, is to split path parameter using the index.

```
const param = req.path.split('/') // we will get the string
```

We will get the string and the last element in the string should be the parameter that we have to check.

```
function prePrcocess(req, res, next) {
    console.log('Pre processing the request')

    if (req.path.includes('math')){

        const param = req.path.split('/')
        const num = param[3]
        const {} = req.params

        const isNumber = Number(num) //Nan

        if(!isNumber){
            res.send('Wrong Parameters')
        }else {
            next()
        }
        return
    }
}
```

There is a better way to do this. in the middleware itself, we are asking to do the check or skip the step and move to the next().

```
const express = require('express')
const app = express();
const mathRouter = require('./routes/math');
const userRouter = require('./routes/user');

function prePrcocess(req, res, next) {
    console.log('Pre processing the request')

    if (req.path.includes('math')){

        const param = req.path.split('/')
        const num = param[3]
        const {} = req.params

        const isNan = Number(num) //Nan

        if(isNan){
```

```
            res.send('Wrong Parameters')
        }else {
            next()
        }
        return
    }
}

router.use(prePrcocess)

router.get('/square/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const sq = square(num)
    res.send(`${sq}`)
})

router.get('/cube/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const cu = cube(num)
    res.send(`${cu}`)
})
```

The logger() is global so we would use it. express allows us to define route level middleware. In the math.js, we will define the preProcess(). This is how we have defined app level middleware, and now we need to specify route level middleware. To do it, we will just **`route.use(preProcess)`**

We can see the path processing request came in, but we can see the preProcess was not logged in. So, logger() is working for all the levels and preProcess() is working only for math route.