Current Skill Middleware

Middleware Introduction

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 that, 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.

Writing middleware (1/2)

Here is a simple example of a middleware function

```
//Simple request time logger
const myLogger = function (req, res, next) {
  console.log("A new request received at " + Date.now());
   next();
}
```

To load the middleware function, call app.use(), specifying which 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();
```

```
//Simple request time logger
const myLogger = function (req, res, next) {
  console.log("A new request received at " + Date.now());
   next();
}

app.use(myLogger);

app.get('/', function (req, res) {
  res.send('Hello World!')
})

app.listen(3000);
```

The above middleware is called for every request on the server. So after every request, we will get the following message in the console:

```
A new request received at 1584954785016
```

Writing middleware (2/2)

To restrict it to a specific route (and all its subroutes), provide that route as the first argument of app.use(). For Example,

```
const express = require('express');
const app = express();

//Middleware function to log request protocol
app.use('/things', function(req, res, next){
   console.log("A request for things received at " + Date.now());
   next();
});


// Route handler that sends the response
```

```
app.get('/things', function(req, res){
    res.send('Things');
});

app.listen(3000);
```

Now, whenever you request any subroute of '/things', that will be the instance where it will log the time.

Error-handling middleware

Express JS comes with default error handling parameters. We can define error-handling middleware functions in the same way as other middleware functions, except error-handling functions have four arguments instead of three:

```
app.use(function (err, req, res, next) {
  console.error(err.stack)
  res.status(500).send('Something broke!')
})
```

In order to call an error-handling middleware, you simply pass the error to next(), like this:

```
app.get('/', (req, res, next) => {
  next(new Error('I am passing you an error!'));
});
```

If you pass anything to the next() function (except the string 'route'), Express regards the current request as being an error and will skip any remaining non-error handling routing and middleware functions.

Third Party Middlewares

A list of third party middlewares for Express are available here. The following are some of the most commonly used middleware. We will also learn how to mount and use them.

body-parser: parse incoming request bodies in a middleware before your handlers. It's available under the req.body property.

To mount body parser, we need to install it using

```
npm install --save body-parser
```

and to mount it, include the following lines in your index.js

```
const bodyParser = require('body-parser');



//To parse URL encoded data
app.use(bodyParser.urlencoded({ extended: false }))



//To parse json data
app.use(bodyParser.json())
```

To view all available options for the body-parser middleware, visit its GitHub page.

cookie-parser: It parses Cookie header and populates req.cookies with objects that have cookie names as keys. To mount cookie parser, we need to install it using

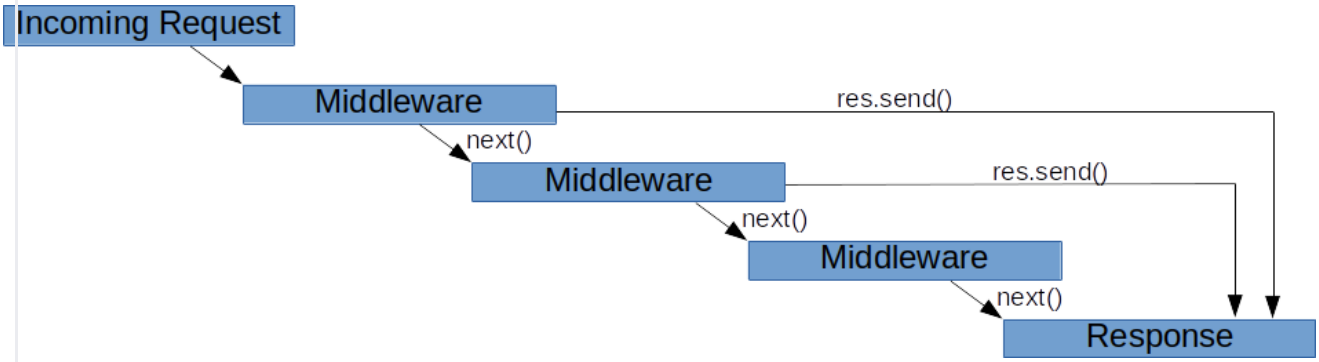
```
npm install --save cookie-parser
```

and to mount it, include the following lines in your index.js

```
var cookieParser = require('cookie-parser');
app.use(cookieParser())
```

Middleware Order is Important

When a request is received by Express, each middleware that matches the request is run in the order it is initialized in until there is a concluding action (like a response being sent).



So if an error occurs, all middlewares that are assigned to handle errors will be called in order until one of them does not call the next() function call.

Previous next