COURSEWARE

Middleware

Contents

Overview

Middleware is software that facilitates communication between two different technologies (such as a database and an application).
Middleware often provides extra functionality to the software it is placed between.

Using Middleware

Middleware in Express *is just a function* - these functions should all have a similar structure with three parameters: **req**, **res** and **next**.

```
app.use((req, res, next) => {
    //do something
    next();
})
```

This syntax should be very familiar to you from *request handling*.

next

A handler function that calls the next piece of middleware in the chain.
Every middleware function should either terminate with **res.send()** or call **next()**, otherwise the request *will hang*.

Creating middleware

Remember that middleware *is just a function*, creating middleware is as simple as making a function that follows the convention and then **use()**-ing it.

For example, here is a simple piece of middleware that logs the current date to the console:

```
const logger = (req, res, next) => {
    console.log(new Date());
    next();
}

app.use(logger);
```

Now, whenever a route further down is hit this function will be called first, log the date and then hand over to the next function in the chain.

Nesting middleware

So far we have applied middleware to *every* route in the chain but sometimes we only want to apply middleware to *some* routes - we can do this by *nesting* the middleware.

```
const logger = (req, res, next) => {
    console.log(new Date());
    next();
}

app.get('/', logger, (req, res) => {
    res.send('Hello, world!');
});
```

In this example the logger middleware is *only* applied to that route and any other routes in the express app are unaffected.

Common middleware

Some pieces of middleware are so common that the Expressjs team have their own managed versions (note that they still need to be installed via npm as they are separate *modules*), a few examples of these are:

cors

Used to disable restrictions on *Cross Origin Resource Sharing*.

```
const cors = require('cors');

app.use(cors());
```

body-parser

Allows for the parsing of request bodies into JS objects. Comes with several body parsers but json is probably the most useful.

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

app.use(bodyParser.json());
```

Tutorial

There is no tutorial for this module.

Exercises

1. Go back to your *request_handling*.
   Implement some middleware *before* the routing that logs the current time to the console *but still passes the request on to the next function.*

▶ Show solution