Express: Middleware

Routes as We Know Them

- Typically routes have one string (that could apply to multiple paths) and have one function that triggers for that route
- Sometimes we call `app.use()` to apply some configuration to our app and routes
- But what does use actually do? What's happening in our routes?

Middleware

- What's actually going on in a route is a series of function calls, much like the ones you define in a route, called "middleware"
- Middleware are functions that have access to the Request (req), the Response (res) just like our routes
- They also use a third parameter, the `next()` function, which tells it to move on to the next middleware (Our our final route function)
- This allows us to provide complex reusable functionality to multiple routes, without having to put all of that code in each route

Ways of Using Middleware

- If a middleware wants to provide the route with more information, it can add keys to the `req` object
- If a middleware wants to render something rather than our route function, it can also use `res` to send things to the client
- If the middleware is done with its task, and wants the route to behave as normal, it'll call `next()` to move on

Middleware Examples: Logger

```
function loggerMW(req, res, next) {
    console.log("Received " + req.method + " request at " + req.path);
    next();
}
```

- This middleware simply logs what request it received, at what path
- By calling next, it continues on to the next middleware, or your final route function

Middleware Examples: Location

```
const geoip = require('geoip-lite');

function locationMW(req, res, next) {
    req.location = geoip.lookup(req.ip);
    next();
}
```

- This middleware adds a `location` key to the `req` object, that has the user's location (Based on their IP address)
- Now any routes that use this middleware can use `req.location` to access information about where the user is visiting the site from

Middleware Example: Secure Pages

```
function isLoggedInMW(req, res, next) {
    checkIsLoggedIn(req).then(function() {
        next();
    })
    .catch(function() {
        res.status(403);
        res.render("no-access");
    });
}
```

- This middleware checks if a user is logged in (Don't worry about how it does that, this is just an example)
- If the user is not logged in, rather than running `next()`, we render a "no-access" page instead of what the route was supposed to show
- The `next()` function can be run asynchronously, so middleware can do HTTP requests or talk to your database

Enabling Middleware

- Middleware can be added in a few ways:
    - `app.use()` or `router.use()` can be given a middleware to apply it to ALL routes in the app or router respectively
    - The route functions (`get`, `post`, etc.) can take in multiple functions that run in order
    - They can also take in arrays, that run in order, if you want to bundle multiple middleware

Enabling Middleware Example

```
// Log every endpoint used
app.use(loggingMW);

// Render the homepage using the user's location
app.get("/", locationMW, function(req, res) {
    res.render("home", {
        location: req.location,
    });
});

// Only allow logged in users to see this page
app.get("/secret", isLoggedInMW, function(req, res) {
    res.render("secret");
});
```

Configurable Middleware

- Some middleware might not work the same in all cases, you may want to configure it
- Since all that middleware need to be are functions, we can get functions in more ways than statically defining them
- We can have a function that returns *another, new function* that uses what we passed the first function
- We do this because you can't call the middleware function with your parameters, since it expects `(req, res, next)` to be passed to it, not our configuration

Configurable Middleware Example

```javascript
function queryPasswordMW(password) {
    return function(req, res, next) {
        if (req.query.password !== password) {
            res.status(403);
            res.render("bad-password");
        }
        else {
            next();
        }
    }
}

app.get("/secret", queryPasswordMW("sup3r_s3cr3t"), function(req, res) {
    res.send("You're in!");
});
```

Organizing and Using Your Middleware

- Now that you know how to make middleware, you don't want them all hanging out in the open
- Make a `middleware/` folder where you'll keep them
- Each middleware should be in its own file, and *exported as a module*
- This will allow you to reuse middleware you create more easily!
- While we may not be writing too much of our own middleware, we'll definitely be using third party modules from NPM that are middleware

Additional Reading

- Express' Guide to Writing Middleware
- Express' Guide to Using Middleware
- List of Useful Middlewares Modules

Challenge 1: View Counter Middleware

- Assuming we've built a really popular well trafficked website, we may want to know how many people are visiting us?
- First you'll need to create an Express site with at least **3** routes
  - Feel free to copy and re-use one of your existing express sites
- Create a new middleware in `middleware/` called `traffic.js`
- It should export a middleware function that does the following:
  - Keep track of the total number of requests made, incrementing once per request
  - Keep track of how many times an individual path is hit (You may want to use an object to store this, like `pathTraffic[req.path]`)
  - Expose the traffic count to our routes on the `req` object as `req.totalTraffic` and `req.pathTraffic`
- Apply this middleware to **all** of your routes (In one line of code, don't paste it to all routes)
- Create an admin page route at `/traffic` that renders `req.totalTraffic` and `req.pathTraffic`

Challenge 2: Secure `/traffic`

- Now that we've got an admin page `/traffic`, we don't want it to be public!
- Create and export a configurable middleware function in `middleware/pwform.js` that takes in a password string, and returns a middleware function
- The middleware should do the following:
  - If no password is provided, render a password form instead of the route
  - If the password is incorrect, the form should still be rendered, but show an error, and console.log the failed password attempt
  - If the password is correct, the route should display as normal
- Apply the middleware to the `/traffic` route, but *don't hardcode the password*, use an environment variable instead