Express Middleware

In this lab, you will use several Express middleware components in the creation of a simple login form.

# Objectives

In this lab, you will learn to

* use several built in Express middleware components together, and
* implement custom middleware.

# Set things up

You have been provided a minimal Express application to start the lab but first we need to install the required modules.

## Install dependencies

1. Run npm install in the lab directory.

This will install the dependencies for our application from the package.json file. Our dependencies are cookie-parser, cookies, and express-session.

While not required, you may want to install nodemon (npm –g nodemon). nodemon automatically restarts your application when a file system change is detected and will save time so you don’t have to manually restart Node to view your edits.

# Add a logger

Open app.js in a text editor and you can see that currently our application doesn't do anything aside from listen to port 3000. Let's begin with one of the most common middleware components you will encounter – logging.

## Add logger middleware

1. Add the logger middleware to the application with an app.use() statement.

Use the provided LOGGER variable to format the log messages by passing it as the argument to morgan(). It contains the default logging format plus a couple of additional arguments that we will use later to examine cookies.

## Test the logger middleware is working

1. Navigate to <http://localhost:3000> and verify that a logging statement appears in the console. The browser will contain an error because we have not yet defined any working routes so let's do that next!

## Add bodyParser middleware

1. Add the body-parser middleware after the logger. This will ensure that our application and correctly interpret application/x-www-form-urlencoded data used in the login form. Pass it the option: {extended:true}. (This option uses node’s qs module that allows parsing of nested objects within a query string.)

## Add session middleware

1. After the bodyParser middleware, add the session() middleware. The session middleware requires a secret to be passed in the format app.use(session({ resave: false, saveUninitialized: false, secret: 'super secret string' })).

## Add methodOverride middleware

1. Add the methodOverride middleware to the application after the bodyParser. This middleware will allow us to send a form with a DELETE method and Connect will populate the req.method object. Pass ‘\_method’ to it.

## Create routes to our pages

1. Add the routes to our pages

Each route should look like:

app.METHOD\_NAME(**PATH\_STRING**, **function**(req, res) {  
 res.sendfile(**RELATIVE\_PATH\_NAME\_TO\_HTML\_FILE**);  
});

The table below indicates the routes:

|  |  |  |  |
| --- | --- | --- | --- |
| Path | Method | File | Logic |
| / | GET | ./views/index.html | none |
| /login | GET | ./views/login.html | none |
| /login | POST | ./views/success.html OR  ./views/failure.html | If the username AND the password exist in the req.body, then display the success page else display the failure page. |
| /logout | GET | ./views/index.html | none |

## Test the login form

1. Navigate to <http://localhost:3000> and make sure that your login form appears. You should be able to see a successful login message with the correct credentials or an error message. This login form isn't very interesting without a session so let's add that next.

## Test the application

1. Navigate to <http://localhost:3000> and verify that the ability to log in and out of the session works properly.

Once everything is working properly, this lab is now complete!