[**Express for node.js – View Engines**](https://colinmackay.scot/2014/11/16/express-for-node-js-view-engines/)

<https://colinmackay.scot/tag/express-ejs-layouts/>

in the last post, [Express Hello World](https://colinmackay.scot/2014/11/15/express-for-node-js-walk-through-hello-world/), I talked about getting started with an Express application in Node.js. In it, the output was rendered by writing HTML directly. In most situations that is undesirable and some sort of view or template engine is better suited to rendering the output.

This post follows on directly from that, so any modifications will be based on what the end result was at the end of the last post. The final output from this post will be shown at the bottom.

There are quite a few view engines out there. A popular one for express is [jade](http://jade-lang.com/), but personally I find it too far from HTML to be useful. This is especially true if you work with graphic designers whose tools will output HTML that the developer has to mould into a view for the application to run. The less rework there the better, in my opinion.

So, I’m going to use [EJS](http://www.embeddedjs.com/) in this post. It is a bit like the ASPX view engine in .NET applications, but its actually a derivative of Ruby’s ERB.

To install EJS add the "ejs" package to the application.

npm install ejs --save

**Configuring the view engine**

Express needs to know which view engine is going to be used, it also needs to know where you are going to store the views. So the following lines are needed in the app.jsfile from the previous post

app.set("view engine","ejs");

app.set("views","./views");

**Changing the route to render the view in the response**

As express has been told to look in a directory called "views" to get the views, the view is going to go into that directory. At this time all that is going into helloworld.ejs is the HTML that was derectly sent form the route in the last post. The helloworld.ejs file now looks like this:

<h1>Hello, World!</h1>

And the route looks like this:

module.exports = function(req, res) {

res.render("helloworld");

};

This will now give the same output as before.

So far, this has done nothing new for us. The power of views is the ability to pass data to them and for them to render it nicely for the user.

**Sending data to the view**

This is relatively easy. All that needs to happen is that the response.render call needs an additional parameter which contains the information.

module.exports = function(req, res) {

res.render("helloworld",{name:"Colin"});

};

In order to render the information the view has to be changed too.

<h1>Hello, <%= name %>!</h1>

**Adding a layout**

EJS, out of the box, does not support layouts (or master pages as .NET’s ASPX view engine calls them). However there is a package that can be added that adds layout support. To install:

npm install express-ejs-layouts --save

And the changes in app.js file

// in the requirements section

var ejsLayouts = require("express-ejs-layouts");

...

// in the set up section

app.use(ejsLayouts);

And that’s it. **Your application can now use layouts, by default it uses the file layout.ejs in the views folder, but this can be changed if you prefer.**

To demonstrate this, here is a layout:

<!DOCTYPE HTML>

<html>

<head>

<title>Express with EJS</title>

<link rel="stylesheet" href="/css/bootstrap.css"/>

<link rel="stylesheet" href="/css/bootstrap-theme.css"/>

</head>

<body>

<div class="container">

<%- body %>

</div>

<script type="application/javascript" src="/js/bootstrap.js"></script>

</body>

**The <%- body %> indicates where the body of the page should go. This is the view that is names in the response.render function call.**

Running the application at this point now shows that the view with its layout are now being rendered. But there is a problem. The styles are not showing up correctly.

**Accessing static files**

By default Express will not serve static files. It needs to be told explicitly where the static files are so that it can render them. While this may seem a little bit of a pain over something like an ASP.NET MVC application, where IIS will server anything in the folder that it recognised (and it recognises a lot), it is actually somewhat comforting that it won’t serve up files by accident.

To set up a folder to be served add the following line to the app.js file:

app.use(express.static(\_\_dirname + '/public'));

What this does is tell Express that a directory called public contains static content and files should be served directly when requested. The \_\_dirname is the directory where the current file is located. So what this means is that public is a directory located in the same directory as the app.js file.

What is interesting here is that the path to the static resource is a full server path. That means the public files can be anywhere on the server, or addressable by the server. The path in the URL will be translated to the full server path as needed. So that means that even although I have a directory called public in my Express application, the browser doesn’t see that directory. It only sees what is in it.

So, from the browser’s perspective the bootstrap.css file is located at /css/bootstrap.css but my application sees it as …/public/css/boostrap.css.

**Summary**

In this post the Hello World application was advanced with the EJS view engine along with layout support and the ability for static files to be rendered to the browser.

The app.js file now looks like this:

// Requirements

var express = require("express");

var http = require("http");

var ejsLayouts = require("express-ejs-layouts");

var hello = require("./routes/hello.js");

// Set up the application

var app = express();

app.set("port", process.env.PORT || 3000);

app.set("view engine","ejs");

app.use(ejsLayouts);

app.set("views","./views");

app.use(express.static(\_\_dirname+"/public"));

app.get("/", hello);

// Run up the server

http.createServer(app).listen(app.get("port"), function(){

console.log("Express server listening on port " + app.get("port"));

});

The routes/hello.js file now looks like this:

module.exports = function(req, res) {

res.render("helloworld",{name:"Colin"});

};

The views/layout.ejs file looks like this:

<!DOCTYPE HTML>

<html>

<head>

<title>Express with EJS</title>

<link rel="stylesheet" href="/css/bootstrap.css"/>

<link rel="stylesheet" href="/css/bootstrap-theme.css"/>

</head>

<body>

<div class="container">

<%- body %>

</div>

<script type="application/javascript" src="/js/bootstrap.js"></script>

</body>

</html>

The views/helloworld.ejs file looks like this:

<div class="row">

<div class="col-md-12">

<h1>Hello, <%= name %>!</h1>

</div>

</div>

And twitter bootstrap was installed into the public directory. The project structure now looks like this:

