**Week 7 Notes (Server-Side with Node.js)**

**RE-INTRODUCTION TO NODE**

Node.js

* Adds a bit to what JavaScript can do
  + Several packages within Node.js

*Express*

* Framework for making web applications. Adds functionality around mapping URLs from HTTP requests to resources and adds to the Request and Response objects to facilitate handling HTTP requests and sending back responses

*Express Handlebars*

* Simplifies task of injecting content into HTML pages.
* Server side interactions generally help make HTML that is specific to the client requesting info
  + Handlebars is a templating engine which lets you have a template where you can replace specific fields with different data

*Express Sessions*

* Package that works with Express to save data about client accessing the server over several requests
* Lets you have client log ins, remember my login, etc

*Body Parser*

* Used to parse the bodies of requests sent by the client

*MySQL*

* Lets you connect to a MySQL database

\*\*\* Note: This course used to use PHP for server-side interactions, but switched to Node.js. PHP is still pretty ubiquitous in web programming

Node.js on Your Server

Hello world program:

var http = require('http');

http.createServer(function(req,res){

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('Hello world!');

}).listen(3000);

console.log('Server started on localhost:3000; press Ctrl-C to terminate....');

require(‘http’); imports the http module (similar to #include in C/C++ or import in Python)

http.createServer is a function that takes a single arg, a function that serves as an event listener for incoming requests (which in turn takes two args every time a request is sent in by the a request object and response object.

**INTRODUCTION TO EXPRESS.JS**

* Node.js is I/O, input/output (minimal stuff)
* Framework ‘Express’ helps you build web applications to help with requests and responses

Hello Express

* Use the npm ‘express’ module

var express = require('express');

var app = express();

app.set('port', 3000);

app.use(function(req,res){

res.type('text/plain');

res.status(404);

res.send('404 - Not Found');

});

app.use(function(err, req, res, next){

console.error(err.stack);

res.type('plain/text');

res.status(500);

res.send('500 - Server Error');

});

app.listen(app.get('port'), function(){

console.log('Express started on http://localhost:' + app.get('port') + '; press Ctrl-C to terminate.');

});

set allows setting of various properties

use mounts middleware at a specified path

listen uses the app’s get method to retrieve port value and pass that in as first argument

Making Thing Marginally More Useful (ok???)

\*\*\*Note on learning frameworks

* Generally frameworks help you avoid all the annoying low level server stuff
* But you might get confused about stuff as you go

**Adding actual pages**

var express = require('express');

var app = express();

app.set('port', 3000);

app.get('/',function(req,res){

res.type('text/plain');

res.send('Welcome to the main page!');

});

app.get('/other-page',function(req,res){

res.type('text/plain');

res.send('Welcome to the other page!');

});

app.use(function(req,res){

res.type('text/plain');

res.status(404);

res.send('404 - Not Found');

});

app.use(function(err, req, res, next){

console.error(err.stack);

res.type('plain/text');

res.status(500);

res.send('500 - Server Error');

});

app.listen(app.get('port'), function(){

console.log('Express started on http://localhost:' + app.get('port') + '; press Ctrl-C to terminate.');

});

Above has *routes*, basically, a mapping of a URL to a resource (in this case, the root URL maps to a main page that says ‘Welcome to the main page!’, and the URL ending in ‘/other-page’ lands you at a page that returns text saying ‘Welcome to the other page!’)

\*\*Note: We use the app’s get method, so GET requests are directed at this page. Could use POST instead.

app.get and app.post are essentially wrappers for app.use.

**INTRODUCTION TO THE HANDLEBARS TEMPLATING SYSTEM**

* Handlebars is a templating system based on the Mustache template system
* Instead of having to build potentially giant strings in our applications, Handlebars lets us have external HTML pages
* You can reuse sections of HTML and use JavaScript to substitute certain content

How Handlebars Helps

Pages can have specific sections which take input from app but otherwise HTML content is generally static

A basic template:

<!doctype html>

<html>

<head>

<title>My Page</title>

</head>

<body>

{{{body}}}

</body>

</html>

* Everything is fixed except for the body of the document
* Handlebars will replace {{{body}}} with whatever is passed when page is rendered in JS
* Kind of a poor example

Example of book template:

<div class="entry">

<h1>{{title}}</h1>

<h2>By {{author.name}}</h2>

<div class="body">

{{body}}

</div>

</div>

Setting Up Handlebars

var express = require('express');

var app = express();

var handlebars = require('express-handlebars').create({defaultLayout:'main'});

app.engine('handlebars', handlebars.engine);

app.set('view engine', 'handlebars');

app.set('port', 3000);

app.get('/',function(req,res){

res.render('home.handlebars') //We can omit the .handlebars extension as we do below

});

app.get('/other-page',function(req,res){

res.render('other-page');

});

app.use(function(req,res){

res.status(404);

res.render('404');

});

app.use(function(err, req, res, next){

console.error(err.stack);

res.type('plain/text');

res.status(500);

res.render('500');

});

app.listen(app.get('port'), function(){

console.log('Express started on http://localhost:' + app.get('port') + '; press Ctrl-C to terminate.');

});

First, requires express-handlebars which can be installed via npm. The app’s engine method sets the handlebars.engine to what handles any file extensions with the .handlebars extension, this lets us omit the file extension when calls are made later.

Instead of res.send() we call res.render() and pass a view rather than a hard coded string

**Dynamically changing content**

Example:

function genContext(){

var stuffToDisplay = {};

stuffToDisplay.time = (new Date(Date.now())).toLocaleTimeString('en-US');

return stuffToDisplay;

}

app.get('/time',function(req,res){

res.render('time', genContext());

});

We can take data that comes from the server and generate HTML that we can give to the client and change it before it is sent

**SERVER-SIDE FORM HANDLING**

* Handling forms is a large part of why stuff is done on the server. Get data, do something with it, send it back to the user

Form Processing

**GETs**

Handling them is baked into express (no modules needed!)

Example:

app.get('/show-data',function(req,res){

var context = {};

context.sentData = req.query.myData;

res.render('show-data', context);

});

Which has a view that looks like this:

<h1>The data you sent in the variable myData is: {{sentData}}</h1>

This is a simple form handler that gets a field called myData from a GET request and prints it back for the user