## \*\*\*TODO:

Add Webstorm Introduction and install

* **Use terminal window**
* Run node (later)
* **Set “show line number” by default (use settings, search for “line number” to find parms)**
* Set up mongo server (later) – MongoDB Plugin (maybe? Or just use shell?
* Add plug-in for AngularJS (later?)
* REST tester
* **Free for 30 days**
* **JavaScript editing, much of what you expect to have in an IDE for Java, including spotting missing variables, missing ‘:’, refactoring with changing of references, other…**

## BEAM

In the beginning there was Netscape Navigator interpreting HTML and simple forms with simple server scripting languages running on Apache web servers. For advanced websites you might have a relational database. Soon after came Java and JavaScript. Then Applets and Servlets. Next CSS, XML, XSD, XSLT, CSS2, AJAX and JSON. Php, python, Rails, Grails, EJB, GWT, GXT, ORBs, ORMs, GORMS, CSS3, HTML5, REST, Cloud computing, NoSQL databases, and mobile devices.

Throw enough monkeys in with typewrites and they’ll write Shakespeare. Eventually.

Give enough programmers enough computers and eventually they’ll write something coherent and consistent? Maybe.

**B**ootstrap, aka Twitter Bootstrap, a simple way to write responsive and mobile web interfaces.

**E**xpress, a package that runs under Node.js, a simple server to implement RESTful routing.

**A**ngularJS – an easy way to generate HTML using JSON and tie it all to client side controllers via client side routing.

**M**ongoDB – an easy to install and easy to use NoSQL database that stores data in an extended version of JSON.

Maybe the monkeys have finally produced Shakespeare?

Simple - a consistent theme of maintaining an underlying simplicity

- REST with hard linking (both AngularJS and Express under Node.js)

- Speak a common language - JavaScript and JSON.

Both Express and AngularJS use "routes" to route REST like requests to the appropriate bit of code.

All the way from MongoDB -> Express -> AngularJS you can pass JSON formatted data structures.

And of course the JavaScript is used throughout, love it or hate it. This makes it easier to share code across a client and server as well as allowing a single programming language. Even MongoDB uses JavaScript in its map reduce function.

There are some "gotchas" along the way. It's still a lot to learn and absorb - even if each piece is fairly simple. It is JavaScript, for better or for worse, and a lot might initially think for worse. It lacks some of the goodness of more static object oriented programming languages, though used correctly JavaScript can still provide much of the utility of object oriented and functional programming.

Over the course of the book we’ll be developing an application to allow display and update of data from a MongoDB. The application will actually eventually be quite functional, but we’ll be building it up one piece at a time. For now, all you’ll need to know is that we’ll be starting with the simplest possible application and adding to the application as we illustrate a new point about the BEAM stack.

## SimpleNode.js Hello World Web Server

Although we’d like to begin with an example AngularJS web page, security restrictions require that we first set up a simple web server. Since we’ll eventually be using Node.js and Express, we’ll start there.

For our first project library we’ve created a project directory named “01\_node”. Within that directory we’ll create two subdirectories, “client” and “server”. Now let’s set up a simple Node.js Express “Hello World” server.

First you’ll need to have Node.js installed on your computer. Although you might be able to use any version, this book uses Node.js version 0.10.15 and Express version 3.3.8. To install Node.js see the Node.js site at <http://nodejs.org/download/>

After installing Node.js, you’ll be using the file named “package.json” in the “01\_node” project directory to install our project dependencies. This file is a simple JSON formatted data structure.

{

"name": "01\_node",

"description": "twitter bootstrap hello world app",

"version": "0.0.1",

"private": true,

"dependencies": {

"express": "3.x"

}

}

The “package.json” file defines the dependencies our project has on various packages, in this case, Express 3.x. The file also contains other project information including the project name, description and version. To use the “packages.json” file, open a console window, navigate to the in the “01\_node” project directory and run the command:

> npm install

The Node Package Manager (NPM) will then read the “package.json” file and install the necessary packages into a “01\_node/node\_modules” directory. We can rerun the install at any time, picking up the latest or a specified version of a package, or adding new packages as needed. In our case we’re installing only the “express”.

For our Node.js “Hello World” app we’ve create a Node.js server definition file “01b\_hello.js” in the project directory (“01\_node”). In that script you’ll find only seven lines of code. The first two create a new instance of an express application.

var express = require('express');

var app = express();

The next three lines specify a “route” of “/hello.txt” and specify the action to take place when that route is received.

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

res.send('Hello World');

});

Finally, two more lines start the server and write a console message to let us know the application is running.

app.listen(3000);

console.log('Listening on port 3000');

To run the server, open a command window, navigate to the“01\_node” project directory and run the command:

> node 01b\_hello.js

This should display a console message “listening on port 3000”.

To test the server, simply point your web browser at <http://localhost:3000/hello.txt> and you should see the message “Hello World”. When you’re done, use ^c to stop the server.

Note that the line

res.send('Hello World');

is an Express shortcut for the three lines

var body = 'Hello World';

res.setHeader('Content-Type', 'text/plain');

res.setHeader('Content-Length', body.length);

res.end(body);

You can read more about this initial example server at <http://expressjs.com/guide.html>

If you want to read more about the Express API, you can go to <http://expressjs.com/api.html> and the res.send() api at <http://expressjs.com/api.html#res.send>

Additionally, Express is an extension of another package, Connect, created by Sencha Labs. You can see more about the Connect API, all of which are available within Express, at <http://www.senchalabs.org/connect/>

More on the Node.js api’s can be found at <http://nodejs.org/api/all.html>

Next we’ll create our first Twitter Bootstrap web page and serve it from our new Node.js server.

## Twitter Bootstrap Hello World

What is Twitter Bootstrap? "Sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development." (from <http://getbootstrap.com/>)

Or, in other words, it provides and easy to use way to develop web pages which can then be run on almost any device, such as a smart phone, tablet, or PC. Why care about mobile? This book isn’t about why mobile first, but there are a number of excellent books on the subject including “Mobile First” by Luke Wroblewski and “Responsive Web Design” by Ethan Marcotte. Suffice to say, why not mobile? Even if your primary audience today is not mobile, five to 10 years from now that will likely change. In addition, the mobile first design tends to be simpler to develop and simpler to use. So what’s not to like?

But modern user interfaces which leverage HTML5 and CSS3 tend to have trouble with some of the older browsers, such as Internet Explorer (IE) versions 8 and lower. For that reason we will not deal with customized code that may sometimes be needed to make Bootstrap as well as AngularJS work on older versions of IE. After all, IE 9 was released in March of 2011 and is free of charge to upgrade, so why not upgrade IE or even use another browser such as Chrome or Firefox, all of which are free? For this book we’ve tested code on the following browsers:

* IE 10.0
* Chrome 20.0
* Firefox f23.0.1

For Bootstrap we’ll be using versions available from a Content Distribution Network (CDN). This allows us to download our Bootstrap and AngularJS CSS and JavaScript from a network of computers the provide free access to various released. Using CDNs provides a number of advantages including: easier to select a particular release, less traffic for your web server, and browser caching means that visitors may already have these packages cached on their local computer from visits to other web sites.

In our case, we’ll be using the following tags for Bootstrap and jQuery, which is needed for some of Bootstraps JavaScript.

//netdna.bootstrapcdn.com/bootstrap/3.0.0/css/bootstrap.min.css

//netdna.bootstrapcdn.com/bootstrap/3.0.0/js/bootstrap.min.js

//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js

If you look at the example code in the directory /01\_node/client you’ll see our Bootstrap hello world HTML in the file “01b\_bootstrap\_hello\_world.html”. In this file you’ll see references to the three CDNs noted above and a tag that you may not have seen before:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

This tag is simple a way to tell Bootstrap how to scale the web page to optimize the display no matter which type of device it is being rendered on.

One other point, notice that the “<script>” tags are placed just before the “</body>” tag. This is done intentionally to allow the rest of the page to load and be rendered before the JavaScript is loaded, speeding up the initial page viewing in those cases where the libraries are not already cached in the browser.

Next we’ll make a slight, and very simple change to our Node.js server script to allow it to serve up static files, such as our “01b\_bootstrap\_hello\_world.html” web page. To do this we simply replace the lines:

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

res.send('Hello World');

});

with the lines

app.configure(function(){

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

});

Almost sounds too simple? But if you open a console window, cd into the project directory “01\_node” and running the server command

> node 01b\_hello\_bootstrap.js

You should then be able to navigate to the URI <http://localhost:3000/01b_bootstrap_hello_world.html> and see your newly created Bootstrap web page, running on your newly created Node.js web server. All in about a dozen lines of code!

If you want to experiment a little, try navigating to <http://localhost:3000/hello.txt> - it fails. Why?

To make the “hello.txt” URI work, you can add the three lines from the first server script file, into the “01b\_hello\_bootstrap”

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

res.send('Hello World');

});

Does it make any difference if these lines are before or after the app.configure statement?

If you’re curious about how this will look on different devices, trying running the web page on different devices connected to your local area network. On Windows you can find out the LAN IP address by opening a console window and typing:

> ipconfig

This will show you a number of addresses. Look for the one labeled “IPv4 Address”. The exact number will vary depending on your network configuration, but mine, for example is 192.169.222.6, so if I enter http://192.169.222.6:3000/01b\_bootstrap\_hello\_world.html on my smart phone or tablet, I will see the Bootstrap “Hello World”. Note that you may have to disable any firewalls if they are blocking your web server’s port 3000.

In addition, if you modified my server script as mentioned above, and can display the “hello.txt” results, you’ll notice that on mobile devices it is not very readable. This is an initial demonstration how web sites developed using Bootstrap and its built-in responsive web design capabilities, will find it easy to build websites that run on a variety of devices.

If you were able to access the web server from a smart phone, try creating “01b\_bootstrap\_hello\_world\_2.html” from the original file but remove the viewport tag:

<meta name="viewport" content="width=device-width, initial-scale=1.0">

What happens when you access this page from a smart phone?

Prior to Bootstrap release 3.0, responsive CSS was separate from the main CSS library. As of 3.0, responsive is built-in and you have to do extra work to remove that feature. You can read more about Bootstraps “mobile first” design at <http://getbootstrap.com/css/#overview-mobile>.

So far you’ve installed Node.js and Express, created a static web server, and built your first twitter bootstrap page, one that can run on a variety of devices including mobile phones and tablets. Next we’ll see a bit more about Bootstraps capabilities including menus and tables.