# *Web Programming III (420-C30-HR)*

# *Lab 6 – Node js Web Servers*

Date assigned: Wednesday, October 5, 2016

Date due: **Wednesday, October 5, 2016, 6:00 p.m.**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will have:

* Create your first node js web server
* Serve html and txt pages from node js

To do:

1. Create a folder called yourusername\_C30\_L05 in your Labs folder for this course. All files will eventually need to be stored in this folder.
2. You can run node from the command line by starting it with Run->All Programs->Node.js->Node.js Command Prompt or by setting it up to run in Notepad++. Follow the steps on this page: <http://blog.aguskurniawan.net/post/notepadjs.aspx> to set up node to run in Notepad++ (I think you will find this easier).

I strongly recommend that you try the WebStorm IDE in today’s lab. Webstorm is an IDE put out by JetBrains and has a very nice JavaScript and Node environment. Start it by going to the Start Menu and selecting JetBrains -> WebStorm. You need to make 2 small configuration changes to make it work better for you.

When the “Welcome to WebStorm” screen is displayed click the Configure dropdown at the bottom of the screen and select Settings. Under Languages and Frameworks choose JavaScript and change the JavaScript language version to ECMAScript6. This way if you use any ES6 features like arrow functions you will not receive errors. Also, under the Node.js and NPM option Enable the Node.js Code library. This will allow you to have smart help on the node modules like http, url, path, etc.

Webstorm will allow you to run the application directly from it (similar to Notepad++) but also provides other tools such as a debugger that can be helpful. To create a project, select Create New Project -> Empty Project and store it in the folder you created in step 1. Do **NOT** create a Node.js Express App as this uses the express module which we have not done yet. You can optionally set up a run configuration for your project, but should not need to do so.

1. You should not need to look this up on the Internet. If you do decide to do that, make sure that you use the techniques from class and no other techniques you will definitely find. I purposely have not yet included the in class examples on Moodle.

Node modules/methods to use.

1. At a minimum you will need the following modules of node and the associated methods/properties:
   1. http – createServer, listen
   2. fs – open, close, readFile
   3. path – basename, extname, dirname (or optionally the parse method which returns all of these values)
   4. url – parse to parse the url string (optional)
2. At a minimum you will need the following parts of the response object (part of http module):
   1. writeHead, setHeader, write, end
3. At a minimum you will need the following parts of the request object (part of http module):
   1. url
4. At a minimum you will also need the currently running directory for node which is: \_\_dirname (two underscores)

HINT: Use console.log – a lot.

**Part A – Node web server 1**

Each of the following steps should be seen as a checkpoint. Complete a number and make sure that it works. This will be much better than attempting to run the entire server at the end.

1. You will need to create at least three separate HTML files for this section. Call them index.html, mypage.html and myother.html. Put some content in them to tell them apart. Also create a text file called text.txt with some information in it (Lorem Ipsum type stuff).
2. Create a file called parta.js. This will be your web server for node. Run the web server on port 3456.
3. Start with a web server that logs a message to the console of the url it received and nothing more. Make sure it is working.
4. When that is working, add writing a header to the page to set the content type (remember writeHead requires a status code as the first parameter). For now, simply set the Content-Type to “text/html”.
5. Add a second header which is your first name with the value of your last name (that is, Allan: McDonald). These must both be set in the writeHead method.
6. Another method of the response object is setHeader which allows you to set header values outside of writeHead. The format is setHeader(“name”, “value”) with both parameters as strings.
   1. Use setHeader method calls to set the value of the header fields:
      1. Cache-Control to no-cache,
      2. Date to be the current date (remember it is a string) and
      3. Warning to be 99 Problems.
7. Use the fs module and methods to try to open the file index.html from the webbase folder and use readFile to read the file. Place the contents in the response object and end the response object to return the file to the browser.
8. Add using basename and dirname from the path module or the parse method of the path module (base and dir are the names corresponding to the methods) which parses the path name and filename from the url.
   1. Use the open method to try and read the associated file
   2. If the file can be opened, close it and the use readFile to read the file and use end (and optionally write) method of the response object to return the file to the browser.
   3. If no file name is included, return index.html.
   4. If a filename is included return that file.
   5. The base path for your webserver (starting path) is the current folder plus /webbase/. That is all files must be in webbase and subsequent folders.
9. Add a check so that if the file cannot be opened an error 404 is returned with the message file not found. This is a TEXT return not HTML. Set the header and Content-Type appropriately. The error code for does not exist is ENOENT. If the open error is NOT 404 return the error code 500 and add the error code to the response message as in “Unknown Server Error: ” + code.
10. Add a check so that if the file cannot be read, but is there return and error 500 with the text Server Error as plain text. At this point you can test with most of the tests at the bottom of this section.
11. Add code to get the extension name using the path module and either extname or parse.
    1. Check the extension of the request. If the extension is html set the content type header to be text/html. If the extension is txt, set the content type header to be text/plain.
12. Test the server using the HTML files above
    1. With no file name, return index.html
    2. With file name, return it
    3. With path and file name, return it
    4. With path and no file name, return index.html in path
    5. With path and incorrect file name, return 404
    6. With file name with txt extension (text.txt) return the file as a text file
    7. With file name with txt extension on the path, return the as a text file
    8. If possible, with path and file name correct but cannot read return 500 (I’m not 100% sure how to accomplish this in Windows)

**Part B – More Robust**

1. Copy parta.js to partb.js. In this part you are going to improve on the web server.
2. Create a folder called errorpages
3. Create two html files called 404.html and 500.html. Add an appropriate message in each of them about the error that occurred.
4. Update your web server to, if one of those errors occurs, read the appropriate file from the errorpages path and return that file.
5. Using the fs.appendFile method to log every path and file name requested and returned (including errors) to a file called logfile.txt in the folder logs.

**To submit**

A zip file of all the files used.