NODE.JS

Dynamic Web Pages Server Side

- There are a number of technologies that allow us to create web pages on the fly on the server. Usually, there is some data-base that will back up the pages
- Example: You do a google search and get back a constructed web-page that has your search results.

Node Architecture

- Event loop based server provides responses for requests.
- The response can be html or other types like plain text.
- Runs on a single process so avoids some concurrency issues. Does not spin off a thread for each request.
- I/O operations are non-blocking (asynchronous). When the I/O operation finishes, continue with the code after the I/O.
- Good scalability

Node Architecture

- Uses a familiar language (JavaScript)
- Runs on the V8 JavaScript engine from Chrome
- Not on a browser so no DOM or any objects provided by the browser
- The developer controls the APIs that will be used.

Installing NodeJS/Express

- The official documentation and download
- The long term stable (LTS) version is fine for our purposes.
- You will want all the options.
- You will want to select installing the tools for the native modules, which will trigger a second installation script after the first completes.
  - *This is also install chocolaty*

A Simple Server

- We want to build a program that listens on given port.
  - *List of common ports and what listens there.*
- Requests come in and it returns a response.
- The response will be plain text.

A Simple Server

```
const http = require('http')  //Pull in a useful node package
                              //Try http.
const hostname = '127.0.0.1'  //Local host
const port = 3001 //Not assigned

const server =
  http.createServer( //Creates the response loop
      (req,res)=> { //Anonymous function to handle the request
        res.statusCode = 200 //code for OK
        res.setHeader('Content-Type', 'text/plain') //Set the mime type
        res.end('Hello World') //Close the response and provide content
      } //No return needed, we modified the res object we got
  )

server.listen(port, hostname, () => { //Start the server
console.log(`Server running at http://${hostname}:${port}/`)
         //Log the start
})
```

Notice:

- This uses template literals
  - *Uses backtick pairs*
  - *Allows us to do string interpolation where an expression is evaluated and slotted into the string. ${expression}.*
  - *A little bit cleaner than doing concatenation.*
  - *Reference*
- Local Host (loop back) – an IP address that is associated internally with our computer.
  - *Useful for testing purposes.*
  - *Finding our machines IP address can be tricky*

Running the server

- Assuming it is saved in the file app.js
- The .js file has code but is not HTML, so no tags.
- It is running in node.
- Control-C to stop it

In a command line.
node app.js

Or to start up a read-eval-print loop do
node

NPM

- Node Package Manager
- Used to manage more complicated projects

JSON

- JavaScript Object Notation
- XML and JSON are popular string based representations of objects.
  - *Used to send structured information from a source to a receiver.*
  - *Serialization/Deserialization*
- Mostly already familiar to you.
  - *Use {} for an object and [] for an array*
- Uses string as the name/key for a mapping in an object.
- Functions are not one of the supported data types.

package.json

- Manifest and meta data for a project
- Uses JSON
- Gives dependencies
  - *Allows one to split into dependencies that are used in production and development.*
- Reference

Generator for package.json

- Name
- version
- Description
- Entry point
- Test command
- Git repo
- Keywords
- Author
- License

In a command line.
In the directory where your project lives.

npm init

Installing express

- An application framework to make it easier to develop a server that handles multiple kinds of requests for a URL

In a command line.

```
npm install express --save
```

The save will add express to the dependencies in the package.json.

A Less Simple Server

```
const http = require('http') //Pull in a useful node package
const hostname = process.env.hostname || '127.0.0.1' //get our ip address
from the environment
const port = process.env.port || 3001 //and the port

const server =
  http.createServer( //Creates the response loop
    (req,res)=> { //Anonymous function to handle the request
    res.statusCode = 200 //code for OK
    res.setHeader('Content-Type', 'text/html') //Set the mime type HTML

    res.write('<html> <head> <title> Served </title> </head>')
    res.write('<body>')
    res.write('Content \n')
    res.write('More content \n')
    res.write('Hello World')
    res.end('</body></html>')
    //Close the response
}
)

server.listen(port, hostname, () => { //Start the server
console.log(`Server running at http://${hostname}:${port}/`) //Log the
start
})
```