Node.js

Lou Marco

Version 1.1
July 2023

Node.js

You:

- Are comfortable with HTML/CSS
- Are fluent in JavaScript

This class:

- Lasts 1 day
- Contains 3 labs

?

QUESTIONS?

? ? ? ?

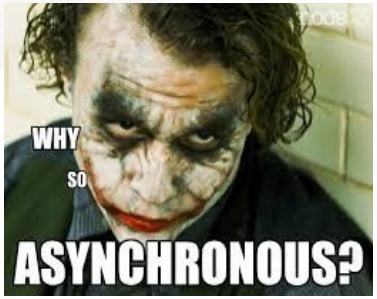
?

Node.js

- What is Node.js?
  - Open source platform that uses JavaScript on the server
  - Built atop Google's V8 virtual machine
    - V8 has various JavaScript extensions available to Node
  - Rich and growing user community
    - Lots of third party stuff out there
  - Current STABLE version: 18.17.1

Node.js

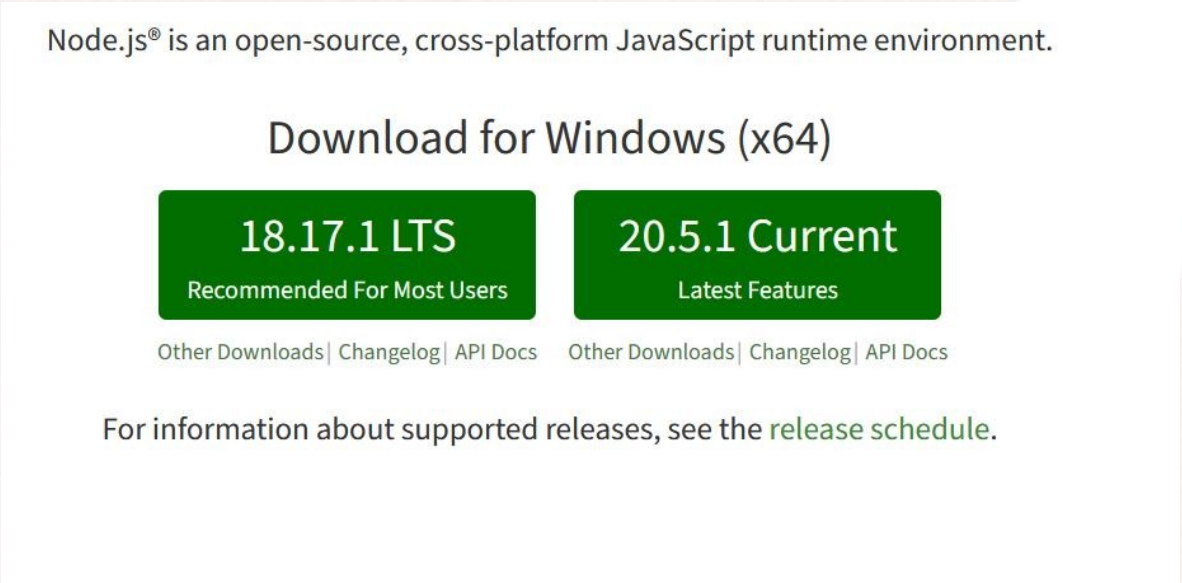
Node.js



- What makes Node.js different?
  - Node uses an event-driven non-blocking I/O model
    - IOW, it's pretty fast and lightweight
  - Stresses *asynchronous operations*
    - JavaScript makes this a bit easier with the common use of it's *callback pattern* and *event emitters*
  - No need for a web app developer to master 'server' based languages like Java, PhP, etc.
    - Now the JavaScript client-side folk can have some fun

Node.js: Getting Started

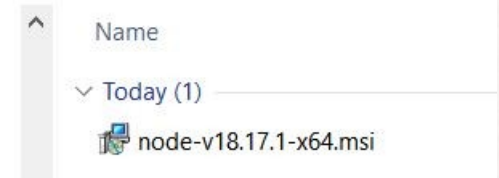
- Download Node.js from www.nodejs.org



- Click that big green button...

Node.js: Getting Started

- For Windows users, you get an *MSI file*

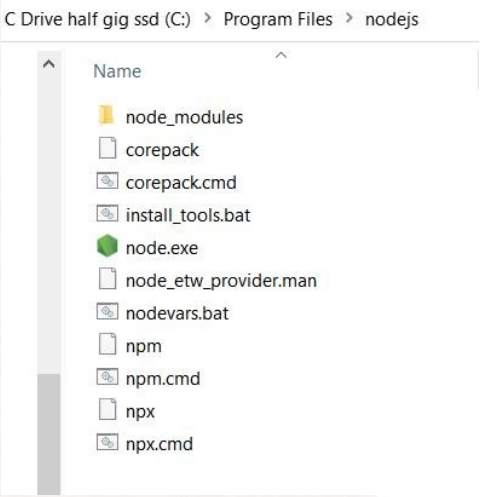


- Opening/running the MSI presents you with the usual install wizard Windows stuff, accept license terms, etc.

- No need to restart Windows to use Node after install

Node.js: Getting Started

- Default install for windows is in c:\program files\nodejs



- Install MSI creates %PATH% entry,

```
C:\Windows\System32>echo %PATH%
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\Javasecurity
ilsInstaller\Git\cmd;C:\RailsInstaller\Ruby1.9.3\bin;C:\Ruby200-x64\bi
Shell\v1.0\;C:\Program Files (x86)\Microsoft ASP.NET\ASP.NET Web Pages
erver\110\Tools\Binn\;C:\Program Files\Java\jdk1.7.0_02\bin;C:\scala-2
Performance Toolkit\;C:\Program Files\Microsoft SQL Server\110\DTS\Bin
osoft SQL Server\110\Tools\Binn\ManagementStudio\;C:\Program Files (x8
osoft SQL Server\110\DTS\Binn\ C:\Program Files\nodejs\;C:\Users\Lou\A
```

Node.js: Getting Started

- Open a *command window*, enter **node -v**

```
C:\Users\Owner>node -v
v18.17.1

C:\Users\Owner>
```

- You see this.. you're good to go

Node.js: Getting Started

- Create a folder that will contain some node scripts/programs

- Create and save a file named *hello.js* in this folder containing one line:

```
console.log( “Hello World” ) ;
```

- Open a command window at your folder location and enter:

```
node hello.js
```

```
C:\nodestuff>node hello.js
Hello World

C:\nodestuff>
```

Node.js: The REPL

- Node users have access to the REPL
  - Read-Evaluate-Print-Loop
- The REPL is a shell that allows you to execute JavaScript scripts and execute JavaScript code interactively
- Handy way to experiment and figure out 'interesting' JavaScript behaviors
- As an aside, several programming languages have a REPL (or a functional equivalent)

Node.js: Using the REPL

- Merely enter *node* at the command line

```
C:\nodestuff>node
>
```

You're ready to enter JavaScript commands

- The > is the basic prompt
- Enter commands/expressions; get an immediate result.

```
C:\nodestuff>node
> 1 + 1
2
> _ + 100
102
>
```

Fetch last returned result with _

Node.js: Using the REPL

- Entering a variable as*signment without the var keyword* saves and returns the value

```
> x = 10
10
> var y = 5
undefined
> x + y
15
>
```

Using *var* saves but does not return the value

- console.clear(); clears the screen

Node.js: Using the REPL

- Not limited to entering single lines in REPL; can enter *code blocks*

```
> arr = [1, 2, 3] ;
[ 1, 2, 3 ]
> arr.forEach( function( elem ) {
... console.log("array elem is " + elem ) ;
... }) ;
array elem is 1
array elem is 2
array elem is 3
undefined
>
```

REPL responds with … when a code block is not complete

- Since we did not assign the value returned by the *forEach* function, the REPL also lists *undefined*

- Exit the REPL by entering .exit or hitting CTRL-C twice

Node.js: Using the REPL

- Fetch command line arguments by using the *process.argv array*

```
C:\nodestuff>node listargs.js 10 20 30 40
Argument passed: C:\Program Files\nodejs\node.exe
Argument passed: C:\nodestuff\listargs.js
Argument passed: 10
Argument passed: 20
Argument passed: 30
Argument passed: 40

C:\nodestuff>
```

C/C++ programmers familiar with the argv array. Usually skip the first 2 elements

- Here's the code in *listargs.js* used above:

```
process.argv.forEach( function( elem )
    { console.log('Argument passed: ' + elem )
    ;
}) ;
```

Node.js: An HTTP Server Example

- Contents of *firsthttpserver.js*

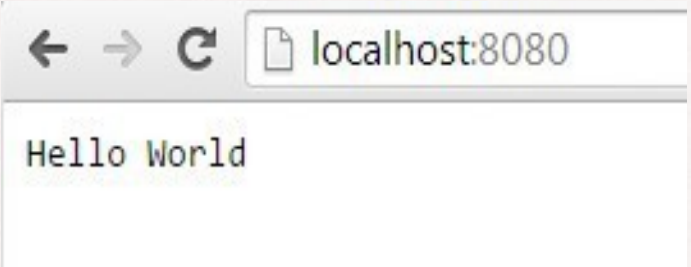
```
var http = require('http');
var server = http.createServer( ) ;

server.on( 'request', function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.write('Hello World') ;
  res.end( ) ;
}) ;
var port = 8080 ; server.listen( port );
server.once( 'listening', function( ) {
   console.log('Hello World server listening on port %d', port ) ;
} );
```

- Enter *node firsthttpserver.js* at command window

```
C:\nodestuff>node firsthttpserver.js
Hello World server listening on port 8080
```

Node.js: An HTTP Server Example

- Open a browser, enter *localhost:8080*



Node.js: An HTTP Server Example

```
let http = require('http');
```

- We *require* the core http module
  - *require* is how we tell Node we want to use functions, data from other JS/node modules
- Node core API contains several modules
  - We'll look at some of them later
- Doesn't matter what we name the target of the assignment
- Subsequent statements will use methods/functions of this *http object*

Node.js: An HTTP Server Example

```
var server = http.createServer( );
```

- Now we have a server; we'll fill in needed details in subsequent statements

```
server.on( 'request', function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.write('Hello World\n');
  res.end( ) ;
}) ;
```

- Bind a function to the server's *request* event
  - Function called for each request
  - Receives an HTTP request and response object The *res*
  - (response) is what 'goes back' to the client

Node.js: An HTTP Server Example

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

- Tell the client we're sending back plain text

```
res.write('Hello World\n');
```

- Write a string to the browser

```
res.end( );
```

- Tell the browser we're done with our response

Node.js: An HTTP Server Example

```
var port = 8080 ; server.listen( port );
```

- Server ready to receive requests on port 8080
  - Server emits *listening event*
  - When server is available it will respond to our response

```
server.once( 'listening', function( ) {
    console.log('Hello World server listening on port %d', port );
```

- The ***once*** method is like the *on* method but only fires when the event is first triggered
  - No need to have the server telling us constantly it is listening on the port!

Node.js: The 'important' core modules

- Our server example relied on a core module named *http*
  - Compiled into the node binaries
- Node comes with many core modules
- Access core (and other) modules with the *require* statement
  - Usually coded atop your node programs

Node.js: Some core modules:

```
net      : For creating TCP clients and servers
http     : For creating and consuming HTTP services
fs       : For accessing and manipulating files
dns      : For using the DNS service
events   : For creating event emitters
stream   : For creating streams

os       : For accessing some local operating system statistics
assert   : For assertion testing

util     : For miscellaneous utilities
```

Node.js: Modules

- Node supports three module types
  - Core
  - Third-party
  - User
- A module is simply a .js file
- User modules are those that you create

Node.js: Creating/Using Modules

- Put this line in *firstmodule.js*

```
module.exports = 'just a string' ;
```

- Put this in *usefirst.js*

```
const simple = require( './firstmodule.js' ) ;
console.log( simple ) ;
```

- Run *usefirst.js*

```
C:\nodestuff>node usefirst.js
just a string

C:\nodestuff>_
```

  - *Better! Run in VSCode. Watch and learn...*

Node.js: Another example

- Put this code in *firstmodule.js* (replace what's there)

```
var x = 5;
var addX = function(value) {
   return value + x;
};
module.exports.x = x;
module.exports.addX = addX;
```

- Put this in *usefirst.js*

```
var simple    = require( './firstmodule' ) ; // Or ./firstmodule.js
console.log( simple.x ) ;
console.log('Using function in module : ' +  simple.addX( 200 ) ) ;
```

- Run *usefirst.js*

```
C:\nodestuff>node usefirst.js
5
Using function in module : 205

C:\nodestuff>
```

Node.js: Creating/Using Modules

- Use *require* to use modules
  - *require* creates an object containing exported references
  - Use the object to access variables/functions exported from 'required' module
- Module location identified by relative pathing
- Avoid absolute pathing for the obvious reason

Node.js – Getting Started

Check out the PDF in the *labs/Getting Started With NodeJS* and follow its instructions

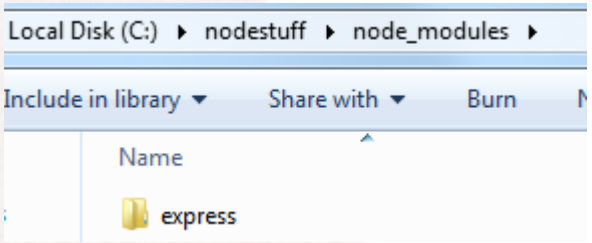
Node.js: Acquiring 3rd Party Modules, Packages

- Let's call a group of related modules a *package*
  - A module may contain packages, packages may contain modules
- Use the ***npm*** application to load and install third party node packages in your installation
- npm comes with your node distribution
- Lots of commands/options; we'll discuss what we'll need to install node packages

Node.js: Acquiring 3rd Party Modules, Packages

- Use the ***npm install <package-name>*** to load and install <package-name>

```
C:\nodestuff>npm install express
express@4.11.1 node_modules\express
├── escape-html@1.0.1
├── methods@1.1.1
├── merge-descriptors@0.0.2
├── cookie@0.1.2
├── utils-merge@1.0.0
├── fresh@0.2.4
├── range-parser@1.0.2
├── cookie-signature@1.0.5
├── finalhandler@0.3.3
├── media-typer@0.3.0
├── vary@1.0.0
├── parseurl@1.3.0
├── content-disposition@0.5.0
├── serve-static@1.8.1
├── path-to-regexp@0.1.3
├── depd@1.0.0
├── on-finished@2.2.0 (ee-first@1.1.0)
├── qs@2.3.3
├── debug@2.1.1 (ms@0.6.2)
├── etag@1.5.1 (crc@3.2.1)
├── send@0.11.1 (destroy@1.0.3, ms@0.7.0, mime@1.2.11)
├── proxy-addr@1.0.5 (forwarded@0.1.0, ipaddr.js@0.1.6)
├── accepts@1.2.2 (negotiator@0.5.0, mime-types@2.0.7)
└── type-is@1.5.5 (mime-types@2.0.7)

C:\nodestuff>
```



npm does a *local install*. Note npm put the installed module in the *node_modules* subdirectory of the directory present when npm was run

Node.js: Acquiring 3rd Party Modules, Packages

npm
Outlook.com - loumarco
npm Inc. [US] https://www.npmjs.com

noncollinear perpendicular microcrystalline | private npm | npm Enterprise | documentation | blog | npm weekly | support

npm

find packages

sign up or log in

npm is the package manager for javascript.

121,321 total packages
41,510,308 downloads in the last day
213,280,128 downloads in the last week
785,326,777 downloads in the last month

packages people 'npm install' a lot

Check out those numbers!!!!!!!

Node.js: Local versus Global Install

- When you install a package *locally*, it's installed in the current directory where you're working
  - Usually your project directory

Installing npm packages, local and global

- Local installations are specific to a particular project
  - Different projects can have different versions of the same package without conflicting

- To install a package locally:

```
npm install package-name
```

Node.js: Local versus Global Install

- Advantages of local installations:
  - Isolation: Each project can have its own set of dependencies without interfering with other projects

  - Version Control: *Dependencies* are listed in the project's **package.json** file, making it easier to share and replicate the project's environment
    - More on package.json soon

Node.js: Local versus Global Install

- When you install a package *globally*, it's installed in the system directory
- Global installations can be accessed from any directory in your command line interface
  - Often used for command-line tools or utilities that you want to use across multiple projects
- To install a package globally:

```
npm install -g package-name
```

Installing npm packages, local and global

Node.js: Local versus Global Install

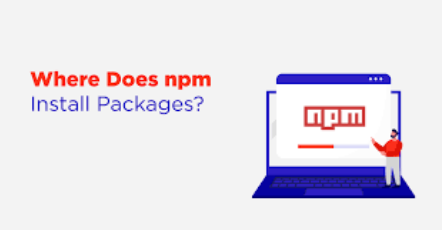
- Advantages of global installations:
  - Convenience: You can use the installed package's command-line tools from any directory
  - Avoid Repetition: You don't need to install the same utility for each project
- Disadvantages of global installations:
  - Version Conflicts: Different projects might depend on different versions of the same package, causing conflicts
  - Less Isolation: Global packages are shared across all projects, which might lead to unintended behavior

Node.js: Local versus Global Install

- In general, prefer *local install* for project-specific dependencies
  - Local install allows dependency management by using the *package.json* manifest

- Install *globally* for tools or utilities that you want to use across projects

Node.js: Where are Installed Modules Stored?



- Global packages are installed in a system-wide directory
  - On Unix-like systems (Linux, macOS), typically stored in */usr/local/lib/node_modules*
  - On Windows, usually *C:\Users\<your_username>\AppData\Roaming\npm\node_modules*
- Local packages are installed within your project's directory in a subfolder named *node_modules*
  - Each project has its own *node_modules* directory to keep its dependencies isolated from other projects

Node.js: The Package.json Mainfest

- Robust node apps use several packages and modules
- Node provides a mechanism to determine if a node app requires modules or packages **and their dependencies**
  - Often, an app requires a *particular release of a dependent package or module*
- The mechanism is to code the module/package names and any required release in its **package.json file** (called a *manifest)*

Node.js: The Package.json Mainfest

- Create a manifest file for modules you're 'serious' about

```
{
  "name": "myapp",
  "version": 1.0.0,
  "dependencies": {
    "request": "*",
    "async": "*"
  }
}
```

- This represents the 1st version of 'myapp' that depends on **any *release*** *of the* request *and async* modules

- Look at the install of the express package on a few slides back, note the dependencies and release (version) information displayed

Node.js: The Package.json Mainfest

- The package.json file contains:
  - All modules needed for the app and installed versions
  - Project metadata (author, license, etc.)
  - Scripts that may automate tasks within the project

- As you install modules via *npm*, node updates package.json as needed
  - You may directly edit the manifest but it is not often necessary to do so

Let's Create a Node Project with a Manifest

- Create a directory, navigate to it
- Enter *npm init*
  - At prompts, feel free to enter anything or take defaults
- npm creates a *package.json* file

```
D:\node-project>dir/w
 Volume in drive D is D Drive 4TB
 Volume Serial Number is FC58-56F4

 Directory of D:\node-project

[.]            [..]           package.json
               1 File(s)            226 bytes
               2 Dir(s)  3,339,130,265,600 bytes free
```

```
D:\>mkdir node-project

D:\>cd node-project

D:\node-project>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (project) demo
version: (1.0.0)
description: show how npm init works
entry point: (index.js)
test command:
git repository:
keywords:
author: Lou
license: (ISC)
About to write to D:\node-project\package.json:

{
  "name": "demo",
  "version": "1.0.0",
  "description": "show how npm init works",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lou",
  "license": "ISC"
}


Is this OK? (yes) y

D:\node-project>
```

Let's Create a Node Project with a Manifest

- The *package.json* file contains the info you supplied at the npm prompts:

```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "show how npm init works",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lou",
  "license": "ISC"
}
```

Let's Install a Package and Examine the Manifest

- Do a *local install* of *express* and examine the manifest after install
  - *npm install express* in case you forgot

```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "show how npm init works",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lou",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

You may see various symbols in the manifest

Next OH

Symbols appearing in the Manifest

- The *symbols* in the manifest specify *package version ranges*
  - Versions show the *major.minor.patch*

• No symbol or =	Exact version
• Hyphen -	Range
• 1.3.2-2.4.1	Includes endpoints
• * 1.* 1.2.*	Wildcard (can also use x)
• ~	Allow patches but not different minor versions
• ^	Allow patches AND minor versions

- Full docs on package.json (including all symbols) in official npm site:

https://docs.npmjs.com/cli/v7/configuring-npm/package-json

What's with this *package-lock.json* file?

- After you installed *express*, node created a *package-lock.json file*

package.lock.json is created for locking the dependency with the installed version

```
D:\node-project>dir/w
 Volume in drive D is D Drive 4TB
 Volume Serial Number is FC58-56F4

 Directory of D:\node-project

[.]                [..]               [node_modules]      package-lock.json   package.json
               2 File(s)         22,556 bytes
               3 Dir(s)  3,339,129,856,000 bytes free
```

  - Without *package-lock.json*, there might be some differences in installed versions in different environments

- Include *package-lock.json* in source control with package.json file
  - Users that clone the project and install dependencies package-lock.json ensures the close will have the exact same dependencies as in package-lock.json

What's with this *package-lock.json* file?

- Don't manually edit *package-lock.json*!
  - Node updates package-lock.json when you take actions that change *package.json* (install packages, e.g.)

package.json	package-lock.json
It contains basic information about the project.	It describes the exact tree that was generated to allow subsequent installs to have the identical tree.
It is mandatory for every project.	It is automatically generated for those operations where npm modifies either node_modules tree or package.json.
It records important metadata about the project.	It allows future devs to install the same dependencies in the project.
It contains information such as name, description, author, script, and dependencies.	It contains the name, dependencies, and locked version of the project.

Node.js – Using npm

Follow the steps shown in the slides to create a Node project and install the *express* package locally

When done, peek at the *package.json* and the *package-lock.json* files

The Node Console

- The console object allows for string outputs via console.*log*, console.*warn* and console.*trace*
- console.log and console.warn allow for *string Interpolation and template strings*

```
let a = {1: true, 2: false};
// Old-style interpolated strings (you may see this from time to time)
console.log(‘Number: %d, string: %s, object (JSON): %j',42,'Hello',a);
// OUTPUT:
Number: 42, string: Hello, object (JSON) (“1”: true, “2”, false}
```

- console.trace() takes no arguments

```
console.trace();

Trace:
at [object Context]:1:9
at Interface. (repl.js:171:22)
at Interface.emit (events.js:64:17)
at Interface._onLine (readline.js:153:10)
```

Node.js: HTTP Module Methods

- The smallest HTTP server I've ever seen:

```
   require('http').createServer(function(req, res) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      res.end('Hello World!'); // Yeah – res.end works
}).listen(4000);
```

- We'll look at SOME methods for the *request* and *response* objects (req, res) in the above example

Node.js: ServerRequest Methods

- Use *method* to get the HTTP method (GET, POST, etc)
- Use *headers* to examine the headers sent by the server

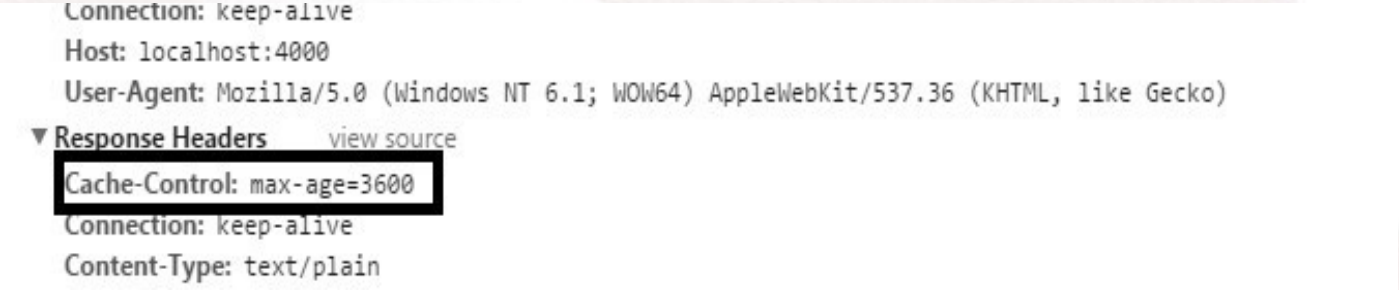
```
var util = require('util' ) ;
require('http').createServer(function(req, res) {
   res.writeHead(200, {'Content-Type': 'text/plain'});
   res.end( util.inspect( req.headers ) ) ;
}).listen(8080);
```

```
Hello World
{ host: 'localhost:8080',
  connection: 'keep-alive',
  accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
  'user-agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Ge
  'accept-encoding': 'gzip, deflate, sdch',
  'accept-language': 'en-US,en;q=0.8' }
```

Node.js: ServerResponse Methods

- Write *headers* with *writeHead*

```
var util = require('util' ) ;
require('http').createServer(function(req, res) {
   res.writeHead(200, {
       'Content-Type': 'text/plain',
       'Cache-Control': 'max-age=3600'
    });
   res.end('Hello World!');
 }).listen(4000);
```



Node.js: ServerResponse Methods

- Set or change headers with set*Header*
- Remove headers with *removeHeader*
  - Both take (hName, hValue) as arguments
  - MUST do this BEFORE writing to response
- Write to the response with *write* or *end*
  - Can write string(s) or buffer(s)

Node.js: HTTP Client Methods

- Although Node was designed with server-side processing in mind, it's flexible enough to act/use as an HTTP client
- Here's an example of a Node client issuing a GET

```
var http = require('http');
var options = {

host: 'www.google.com',  //Try 12345.com
port: 80,
path: '/index.html'
};
http.get(options, function(res) {
  console.log('got response: ' + res.statusCode + " from " + options.host);
}).on('error', function(err)
  { console.log('got error: ' +
  err.message)
});
```

```
C:\nodestuff>node client1.js
got response: 302 from www.12345.com

C:\nodestuff>node client1.js
got response: 200 from www.google.com

C:\nodestuff>
```

Node.js: HTTP Client Methods

- Create a request using the *request* method

```
http.request( options, callback ) ;
```

Options argument to request:

- *host*: A domain name or IP address of the server to issue the request to.
- *port*: Port of remote server.
- *method*: A string specifying the HTTP request method.
  Possible values: ‘GET’ (default), ‘POST’, ‘PUT’, and ‘DELETE’.
- *path*: Request path. Should include query string and fragments if any.
  E.G. ‘/index.html?page=12’
- *headers*: An object containing request headers.

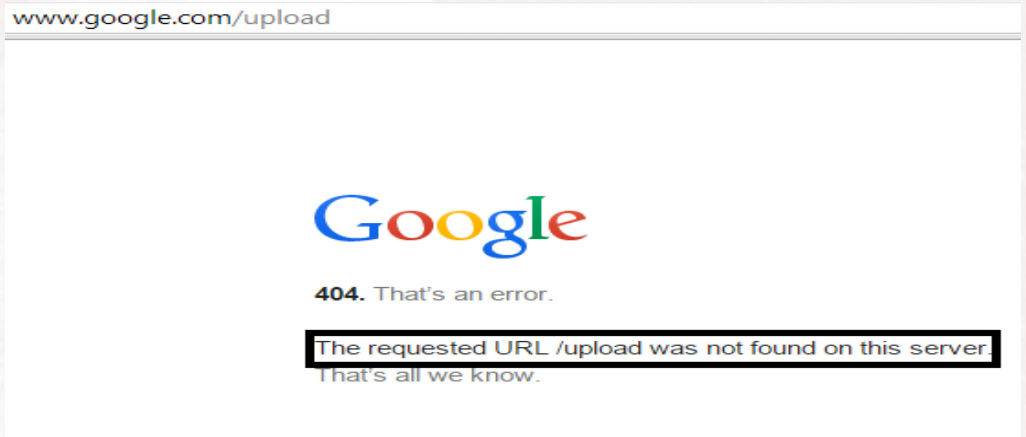
Node.js: HTTP Client Methods

- Example *request* usage

```
let options = {
 host: 'www.google.com', port: 80,
 path: '/upload',
 method: 'POST'
};
let req = require('http').request(options, function(res) {
    console.log('STATUS: ' + res.statusCode);
    console.log('HEADERS: ' + JSON.stringify(res.headers));
    res.setEncoding('utf8');
    res.on('data', function (chunk){
       console.log('BODY: ' + chunk);
    });
});
// write data to request body
req.write("data\n");
req.write("data\n");
req.end();
```

Node.js: HTTP Client Methods

```
C:\nodestuff>node client2.js
STATUS: 404
HEADERS: {"content-type":"text/html; charset=UTF-8","x-content-type-options":"nosniff","date":
1431","x-xss-protection":"1; mode=block","alternate-protocol":"80:quic,p=0.02"}
BODY: <!DOCTYPE html>
<html lang=en>
  <meta charset=utf-8>
  <meta name=viewport content="initial-scale=1, minimum-scale=1, width=device-width">
  <title>Error 404 (Not Found)!!1</title>
  <style>
    *{margin:0;padding:0}html,code{font:15px/22px arial,sans-serif}html{background:#fff;color:
:180px;padding:30px 0 15px}* > body{background:url(//www.google.com/images/errors/robot.png) 1
w:hidden}ins{color:#777;text-decoration:none}a img{border:0}@media screen and (max-width:772px
}}#logo{background:url(//www.google.com/images/errors/logo_sm_2.png) no-repeat}@media only scr
com/images/errors/logo_sm_2_hr.png) no-repeat 0% 0%/100% 100%;-moz-border-image:url(//www.goog
-webkit-min-device-pixel-ratio:2){#logo{background:url(//www.google.com/images/errors/logo_sm_
ay:inline-block;height:55px;width:150px}
  </style>
  <a href=//www.google.com/><span id=logo aria-label=Google></span></a>
  <p><b>404.</b> <ins>That's an error.</ins>
  <p>The requested URL <code>/upload</code> was not found on this server.  <ins>That's all we


C:\nodestuff>
```



Node.js: Quick look at *express*

- Express is a minimal and flexible Node.js web application framework that provides a robust set of features to develop web and mobile applications

- Core features:
  - Allows to set up middleware to respond to HTTP Requests
  - Defines a routing table which is used to perform different actions based on HTTP Method and URL
  - Allows to dynamically render HTML Pages based on passing arguments to templates

Node.js: Quick look at *express*

- Handy helper modules that should be installed via npm when using express:
  - body-parser — This is a node.js middleware for handling JSON, Raw, Text and URL encoded form data
  - Cookie-parser — Parse Cookie header and populate req.cookies with an object keyed by the cookie name

Node.js: HelloWorld

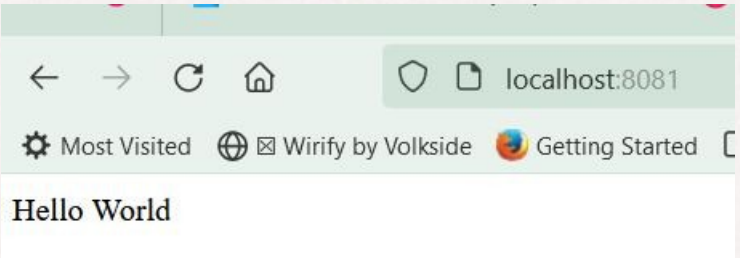
- Starts a server, listens on port 8081 for connection
  - This app responds with Hello World! for requests to the homepage
  - For every other path, it will respond with a 404 Not Found

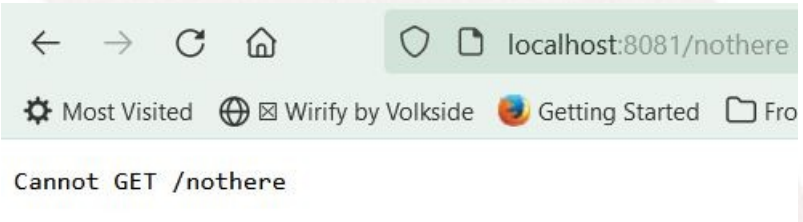
```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
   res.send('Hello World');
})

const server = app.listen(8081, function () {
   const host = server.address().address
   const port = server.address().port

   console.log("Example app listening at http://%s:%s", host, port)
})
```

Node.js: HelloWorld

- Run it!

```
node hello-express.js
Example app listening at http://:::8081
```





Node.js: Basic Routing

- Routing refers to determining how an application responds to a client request to a particular endpoint
  - URI (or path) and a specific HTTP request method (GET, POST, et. al)

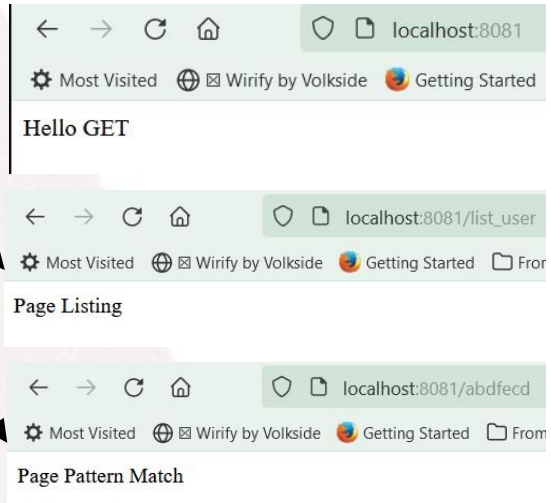
```
const express = require('express');
const app = express();

// This responds with "Hello World" on the homepage
app.get('/', function (req, res) {
   console.log("Got a GET request for the homepage");
   res.send('Hello GET');
})

// This responds a POST request for the homepage
app.post('/', function (req, res) {
   console.log("Got a POST request for the homepage");
   res.send('Hello POST');
})

// This responds a DELETE request for the /del_user page.
app.delete('/del_user', function (req, res) {
   console.log("Got a DELETE request for /del_user");
   res.send('Hello DELETE');
})
// More code...
```

Entire program with additional requests in basic-routing.js

Node.js: Basic Routing

- Run it!

```
node basic-routing.js
Example app listening at http://:::8081
Got a GET request for the homepage
Got a GET request for /list_user
```



Code for /list_user and /ab*cd endpoint requests

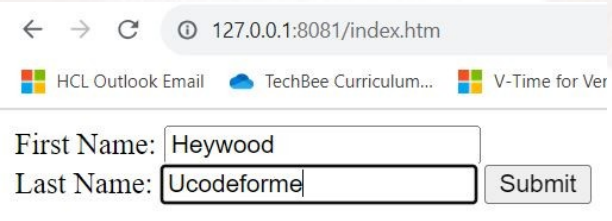
```
// This responds a GET request for the /list_user page.
app.get('/list_user', function (req, res) {
   console.log("Got a GET request for /list_user");
   res.send('Page Listing');
})

// This responds a GET request for abcd, abxcd, ab123cd, and so on
app.get('/ab*cd', function(req, res) {
   console.log("Got a GET request for /ab*cd");
   res.send('Page Pattern Match');
})
```

Node.js: Using Forms

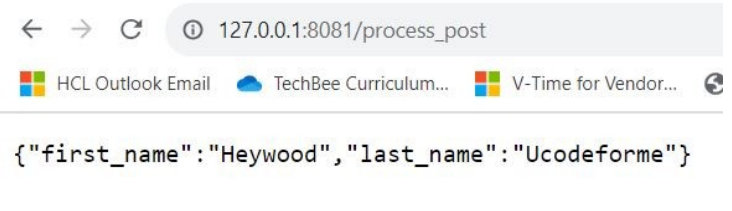
- This example shows express grabbing form data

```
<html>       <!-- index.htm  -->
   <body>
      <form action = "http://127.0.0.1:8081/process_post" method = "POST">
         First Name: <input type = "text" name = "first_name"> <br>
         Last Name: <input type = "text" name = "last_name">
         <input type = "submit" value = "Submit">
      </form>

   </body>
</html>
```

Start server, enter data in form



Click ‘Submit’




```
Example app listening at http://:::8081
{ first_name: 'Heywood', last_name: 'Ucodeforme' }
```

Node console:

Node.js: Using Forms

- The express server:

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');

// Create application/x-www-form-urlencoded parser
const urlencodedParser = bodyParser.urlencoded({ extended: false })

app.use(express.static('public')); // Use static web pages
app.get('/index.htm', function (req, res) {
   res.sendFile( __dirname + "/" + "index.htm" );
})
app.post('/process_post', urlencodedParser, function (req, res) {
   // Prepare output in JSON format
   response = {
      first_name:req.body.first_name,
      last_name:req.body.last_name
   };
   console.log(response);
   res.end(JSON.stringify(response));
})

const server = app.listen(8081, function () {
    const host = server.address().address
    const port = server.address().port

    console.log(`Example app listening at http://${host}:${port}`);

})
```

Recommended; example works without it

Matches action= in form

Use body-parser with urlencoded when using forms

process-form.js

Node.js – Using express

Look in the folder

Using Forms and Express

and check out:

Lab Using Express.pdf