Node.js

COEN 161

What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Why Node.js?

Node.js uses asynchronous programming!

- A common task for a web server can be to open a file on the server and return the content to the client
- Here is how PHP handles a file request
    1. Sends the task to the computer's file system.
    2. Waits while the file system opens and reads the file.
    3. Returns the content to the client.
    4. Ready to handle the next request.

Why Node.js?

Node.js uses asynchronous programming!

- Here is how Node.js handles a file request
    1. Sends the task to the computer's file system.
    2. Ready to handle the next request.
    3. When the file system has opened and read the file, the server returns the content to the client.
- Node.js eliminates the waiting, and simply continues with the next request
- Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Getting Started

- Download Node.js
- Create a Node.js file named "myfirst.js", and add the following code

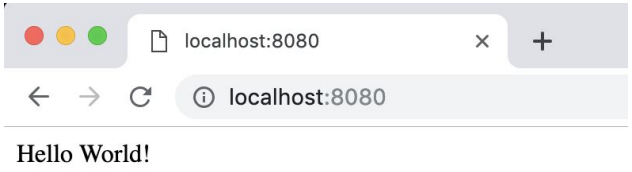
```
var http = require('http');

http.createServer(function (req, res) {

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

  res.end('Hello World!');

}).listen(8080);
```

Command Line Interface

- Node.js files must be initiated in the "Command Line Interface" program of your computer
- Start your command line interface, write **`node myfirst.js`** and hit enter

```
$ node myfirst.js
```

- Now, your computer works as a server!
- If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return



Node.js Modules

- Consider modules to be the same as JavaScript libraries, a set of functions you want to include in your application
- Node.js has a set of built-in modules which you can use without any further installation
  - events - handles events
  - fs - handles the file system
  - http - makes Node.js act as an HTTP server
  - https - makes Node.js act as an HTTPS server.

Include Modules

- To include a module, use the **`require()`** function with the name of the module

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

- Now your application has access to the HTTP module, and is able to create a server

```
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Create Your Own Modules

- You can create your own modules, and easily include them in your applications
- Use the **`exports`** keyword to make properties and methods available outside the module file

```
// myfirstmodule.js

exports.myDateTime = function () {

  return Date();

};
```

Include Your Own Module

- Now you can include and use the module in any of your Node.js files

```
var http = require('http');
var dt = require('./myfirstmodule'); // relative path with './'

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("The date and time are currently: " + dt.myDateTime());
  res.end();
}).listen(8080);
```

Node.js HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP)
- To include the HTTP module, use the **`require()`** method

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

Node.js as a Web Server

- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client
- Use the **`createServer()`** method to create an HTTP server

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

//create a server object:
http.createServer(function (req, res) {
  res.write('Hello World!'); //write a response to the client
  res.end(); //end the response
}).listen(8080); //the server object listens on port 8080
```

Add an HTTP Header

- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type
- The first argument of the **`res.writeHead()`** method is the status code, 200 means that all is OK
- The second argument is an object containing the response headers

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Read the Query String

- The function passed into the **`http.createServer()`** has a **`req`** argument that represents the request from the client, as an object
- This object has a property called "**`url`**" which holds the part of the url that comes after the domain name

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

Split the Query String

- There are built-in modules to easily split the query string into readable parts, such as the URL module

```
var http = require('http');
var url = require('url'); // new module

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query; // read url
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

Node.js as a File Server

- The Node.js file system module allows you to work with the file system on your computer

```
var fs = require('fs');
```

- Common use for the File System module:
  - Read files
  - Create files
  - Update files
  - Delete files
  - Rename files

Read Files

- Assume we have the following HTML file called demofile.html

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Read Files

- The following Node.js file reads the HTML file, and returns the content
- The **`fs.readFile()`** method is used to read files on your computer

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    res.end();
  });
}).listen(8080);
```

Create Files

- The File System module has methods for creating new files
- The **`fs.appendFile()`** method appends specified content to a file
- If the file does not exist, the file will be created

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!',
function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Create Files

- The `fs.open()` method takes a "flag" as the second argument
- If the flag is "w" for "writing", the specified file is opened for writing
- If the file does not exist, an empty file is created

```
var fs = require('fs');

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
```

Create Files

- The `fs.writeFile()` method replaces the specified file and content if it exists
- If the file does not exist, a new file, containing the specified content, will be created

```
var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'Hello content!', function
(err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Update Files

- The File System module has methods for updating files
- The **`fs.appendFile()`** method appends the specified content at the end of the specified file

```
var fs = require('fs');

fs.appendFile('mynewfile1.txt', ' This is my text.',
function (err) {
  if (err) throw err;
  console.log('Updated!');
});
```

Update Files

- The `fs.writeFile()` method replaces the specified file and content

```
var fs = require('fs');

fs.writeFile('mynewfile3.txt', 'This is my text',
function (err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

Delete Files

- To delete a file with the File System module, use the **`fs.unlink()`** method

```
var fs = require('fs');

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});
```

Rename Files

- To rename a file with the File System module, use the **`fs.rename()`** method

```
var fs = require('fs');

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function
(err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

Node.js URL Module

- The URL module splits up a web address into readable parts
- Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'

var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

Node.js File Server

- Now we know how to parse the query string, and we know how to make Node.js behave as a file server, let's combine the two
- summer.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Summer</h1>
<p>I love the sun!</p>
</body>
</html>
```

- winter.html

```
<!DOCTYPE html>
<html>
<body>
<h1>Winter</h1>
<p>I love the snow!</p>
</body>
</html>
```

Node.js File Server

- The file server returns the requested file, or a 404 error

```
var http = require('http');
var url = require('url');
var fs = require('fs');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

Node.js NPM

- What is NPM?
  - NPM is a package manager for Node.js packages, or modules if you like
  - www.npmjs.com hosts thousands of free packages to download and use
  - The NPM program is installed on your computer when you install Node.js
- What is a Package?
  - A package in Node.js contains all the files you need for a module
  - Modules are JavaScript libraries you can include in your project

Download a Package

- To download an NPM package, open the command line interface and tell NPM to download the package you want
- I want to download a package called "upper-case"

```
$ npm install upper-case
```

- NPM creates a folder named "node_modules", where the package will be placed
- All packages you install in the future will be placed in this folder

Using a Package

- Once the package is installed, it is ready to use
- Include the "upper-case" package the same way you include any other module

```
var http = require('http');
var uc = require('upper-case');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(uc("Hello World!"));
  res.end();
}).listen(8080);
```

Events in Node.js

- Every action on a computer is an event
- Like when a connection is made or a file is opened
- Objects in Node.js can fire events, like the readStream object fires events when opening and closing a file

```
var fs = require('fs');
var rs = fs.createReadStream('./demofile.txt');
rs.on('open', function () {
  console.log('The file is open');
});
```

Events Module

- Node.js has a built-in module, called "Events", where you can create-, fire-, and listen for- your own events
- All event properties and methods are an instance of an EventEmitter object
- To be able to access these properties and methods, create an EventEmitter object

```
var events = require('events');

var eventEmitter = new events.EventEmitter();
```

The EventEmitter Object

- You can assign event handlers to your own events with the EventEmitter object
- To fire an event, use the **`emit()`** method

```
var events = require('events');
var eventEmitter = new events.EventEmitter();

//Create an event handler:
var myEventHandler = function () {
  console.log('I hear a scream!');
}

//Assign the event handler to an event:
eventEmitter.on('scream', myEventHandler);

//Fire the 'scream' event:
eventEmitter.emit('scream');
```

Node.js Upload Files

- The Formidable Module
  - There is a very good module for working with file uploads, called "Formidable"
- After you have downloaded the Formidable module, you can include the module in any application

```
var formidable = require('formidable');
```

Step 1: Create an Upload Form

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write('<form action="fileupload" method="post"
enctype="multipart/form-data">');
  res.write('<input type="file" name="filetoupload"><br>');
  res.write('<input type="submit">');
  res.write('</form>');
  return res.end();
}).listen(8080);
```

Step 2: Parse the Uploaded File

```
var http = require('http');
var formidable = require('formidable');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      res.write('File uploaded');
      res.end();
    });
  } else {
  res.writeHead(200, … }).listen(8080);
```

Step 3: Save the File

```
...
var fs = require('fs');

http.createServer(function (req, res) {
  if (req.url == '/fileupload') {
    var form = new formidable.IncomingForm();
    form.parse(req, function (err, fields, files) {
      var oldpath = files.filetoupload.path;
      var newpath = '/Users/Your Name/' + files.filetoupload.name;
      fs.rename(oldpath, newpath, function (err) {
        if (err) throw err;
        res.write('File uploaded and moved!');
        res.end();
      });
    });
  } else {
  res.writeHead(200, … }).listen(8080);
```

Import/Export in Node.js

- Starting with version 8.5.0+, Node.js supports ES6 modules
- For example, you can have a file, **`circle.js`**

```
const PI = 3.14159265359;

export function area(radius) {
  return (radius ** 2) * PI;
}

export function circumference(radius) {
  return 2 * radius * PI;
}
```

Import/Export in Node.js

- Then you can **`import`** in other files

```
import { area, circumference } from './circle.mjs';

const r = 3;
console.log(area(r));
console.log(circumference(r));
```

Import/Export in Node.js

- This syntax can be used in place of **`require`** and **`exports`**

```
// myfirstmodule.js

export function myDateTime() {

  return Date();

};

// usemymodule.js

import { createServer } from 'http';

import { myDateTime } from './myfirstmodule';
```

References

https://www.w3schools.com/nodejs/nodejs_intro.asp