ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

502070

WEB APPLICATION DEVELOPMENT USING NODEJS

LESSON 01 – INTRODUCTION TO NODEJS

OUTLINE

OUTLINE

1. 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!
- Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

How PHP or ASP handles a file request	How Node.js 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.	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.

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"

OUTLINE

Node.js Get Started

- helloworld.js

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

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

Node.js Get Started

- Download Node.js: https://nodejs.org
- Node.js files must be initiated in the "Command Line Interface" program of your computer.
- Start your command line interface, write “node helloworld.js” and hit enter.
- Start your internet browser, and type in the address: http://localhost:8080

OUTLINE

Node.js Modules

- Consider modules to be the same as JavaScript libraries.
- Node.js has a set of built-in modules which you can use without any further installation.
- To include a module, use the require() function with the name of the module:

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

Create Your Own Modules

- You can create your own modules, and easily include them in your applications.
- Create a module that returns the current date and time:

```
exports.myDateTime = function () {
  return Date();
};
```

- Use the exports keyword to make properties and methods available outside the module file.
- Save the code above in a file called "myfirstmodule.js"

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');

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);
```

OUTLINE

The Built-in 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:

```
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 (http.IncomingMessage 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);
```

- Visit: http://localhost:8080/summer

OUTLINE

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');

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

- Visit: `http://localhost:8080/?year=2020&month=December`

The Built-in URL Module

- The URL module splits up a web address into readable parts.
- To include the URL module, use the require() method:

```
var url = require('url’);
```

- 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

- summer.html:

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

Node.js File Server

- winter.html:

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

Node.js File Server

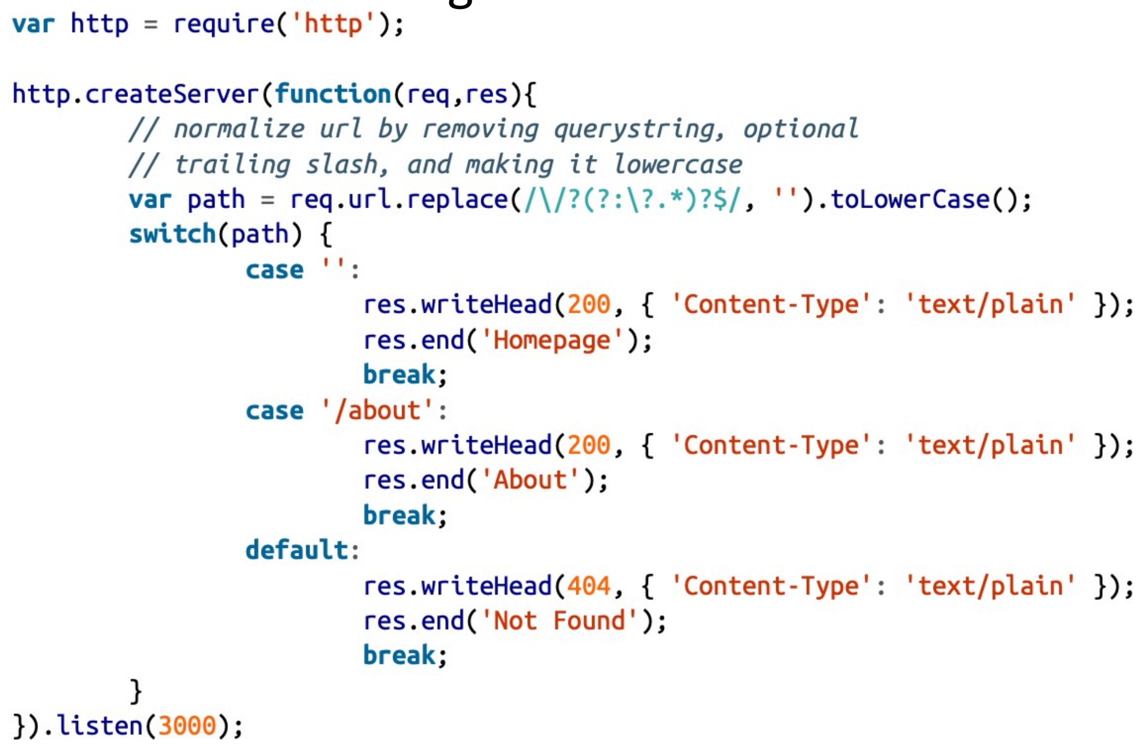
- Create a Node.js file that opens the requested file and returns the content to the client. If anything goes wrong, throw a 404 error::

```
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);
```

Routing

- Routing refers to the mechanism for serving the client the content it has asked for.



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

http.createServer(function(req,res){
        // normalize url by removing querystring, optional
        // trailing slash, and making it lowercase
        var path = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
        switch(path) {
                case '':
                        res.writeHead(200, { 'Content-Type': 'text/plain' });
                        res.end('Homepage');
                        break;
                case '/about':
                        res.writeHead(200, { 'Content-Type': 'text/plain' });
                        res.end('About');
                        break;
                default:
                        res.writeHead(404, { 'Content-Type': 'text/plain' });
                        res.end('Not Found');
                        break;
        }
}).listen(3000);
```

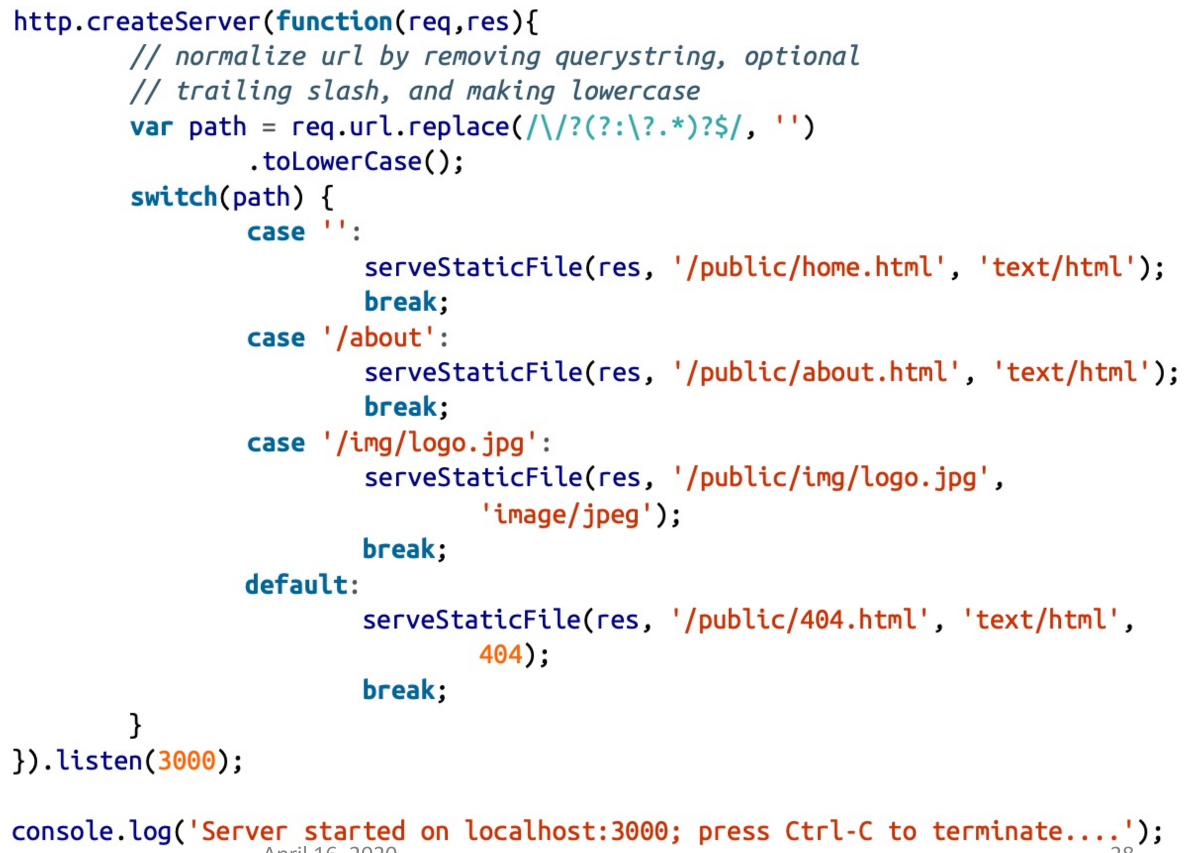
Serving Static Resources

- We need to open the file, reading it, and then sending its contents along to the browser.

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

function serveStaticFile(res, path, contentType, responseCode) {
        if(!responseCode) responseCode = 200;
        fs.readFile(__dirname + path, function(err,data) {
                if(err) {
                        res.writeHead(500, { 'Content-Type': 'text/plain' });
                        res.end('500 - Internal Error');
                } else {
                        res.writeHead(responseCode,
                                { 'Content-Type': contentType });
                        res.end(data);
                }
        });
}
```

Serving Static Resources



```
http.createServer(function(req,res){
        // normalize url by removing querystring, optional
        // trailing slash, and making lowercase
        var path = req.url.replace(/\/?(?:\?.*)?$/, '')
                .toLowerCase();
        switch(path) {
                case '':
                        serveStaticFile(res, '/public/home.html', 'text/html');
                        break;
                case '/about':
                        serveStaticFile(res, '/public/about.html', 'text/html');
                        break;
                case '/img/logo.jpg':
                        serveStaticFile(res, '/public/img/logo.jpg',
                                'image/jpeg');
                        break;
                default:
                        serveStaticFile(res, '/public/404.html', 'text/html',
                                404);
                        break;
        }
}).listen(3000);

console.log('Server started on localhost:3000; press Ctrl-C to terminate....');
```

OUTLINE

1. What is Node.js?
2. Node.js Get Started
3. Node.js Modules
4. Node.js HTTP Module
5. Node.js URL module
6. 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

- Downloading a package is very easy.
- 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 uc = require('upper-case’);
```

- Create a Node.js file that will convert the output "Hello World!" into upper-case letters:

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