

NPM and Modules

CS477 – Server-side Programming

Maharishi International University

Department of Computer Science

M.S. Thao Huy Vu

Maharishi International University - Fairfield, Iowa

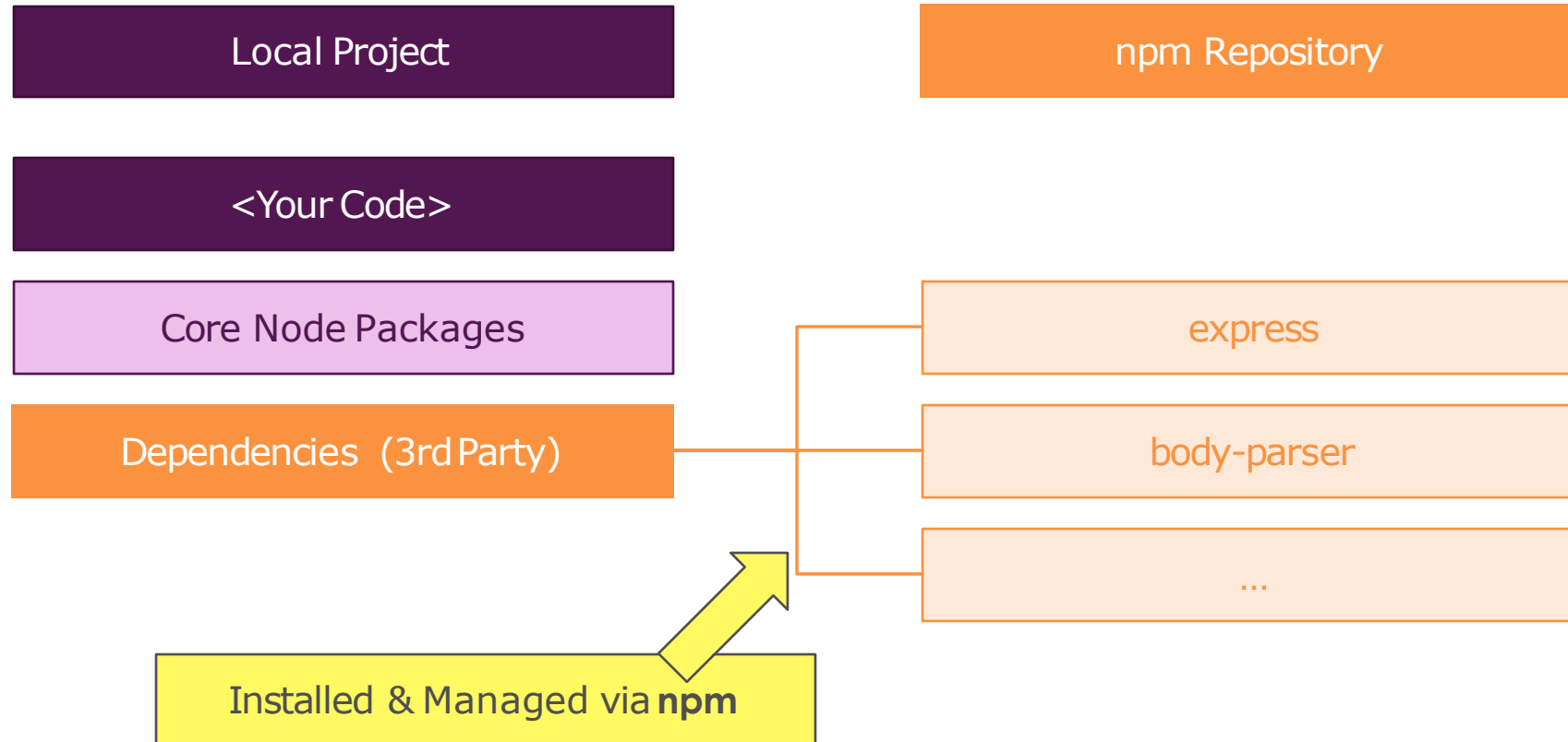


All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi University of Management.

Main Points

- ▶ NPM (Node Package Manager)
- ▶ Versioning
- ▶ HTTP/HTTPS

npm & packages Intro



NPM

- ▶ **npm** is the standard package manager for Node.js. It also manages downloads of dependencies of your project.
- ▶ 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.
 - ▶ `npm -v` // will print npm version
- ▶ 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.
 - ▶ A package contains:
 - ▶ JS files
 - ▶ `package.json` (manifest)
 - ▶ `package-lock.json` (maybe)

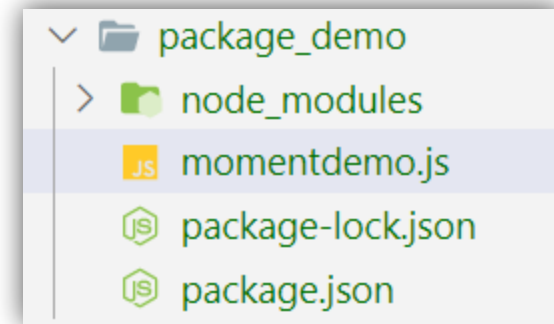
Create & use a new package

```
npm init // will create package.json
```

```
npm install moment --save
```

// moment is a package that parse, validate, manipulate and display dates

- ▶ When we install a package:
 - ▶ Notice dependencies changes in `package.json`
 - ▶ notice folder: `node_modules`
 - ▶ This structure separate our app code to the dependencies. Later when we share/deploy our application, there's no need to copy `node_modules`, run: `npm install` will read all dependencies and install them locally.



```
momentdemo.js
var moment = require('moment');
console.log(moment().format("LLLL")); //Sunday, June 13,
2021 6:24 PM
```

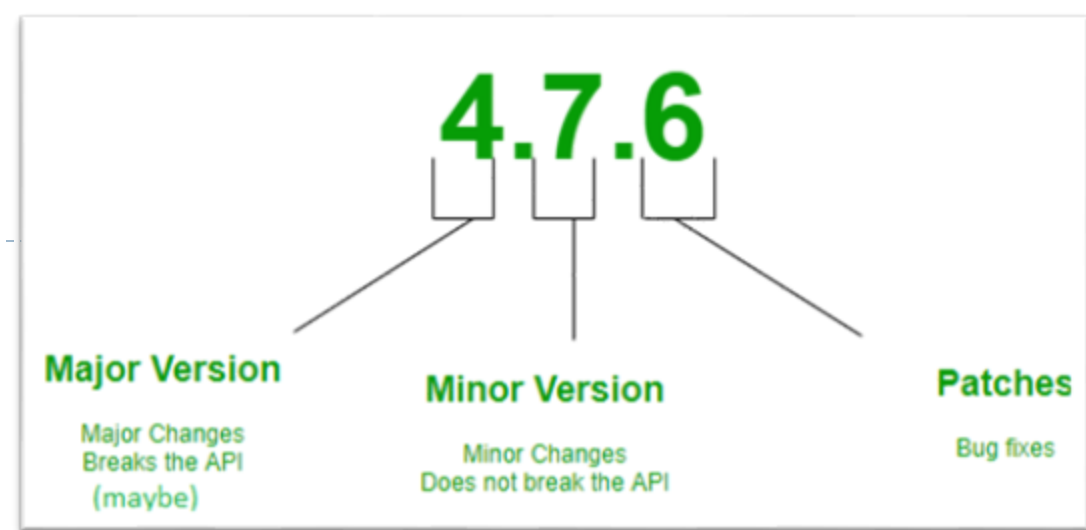
package.json Manifest

- ▶ The `package.json` file is kind of a manifest for your project.
 - ▶ It can do a lot of things, completely unrelated.
 - ▶ It's a central repository of configuration for installed packages.
 - ▶ The only requirement is that it respects the JSON format.
-
- ▶ `version`: indicates the current version
 - ▶ `name`: the application/package name
 - ▶ `description`: a brief description of the app/package
 - ▶ `main`: the entry point for the application
 - ▶ `scripts`: defines a set of node scripts you can run
 - ▶ `dependencies`: sets a list of npm packages installed as dependencies
 - ▶ `devDependencies`: sets a list of npm packages installed as development dependencies

```
{  
  "name": "package_demo",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "start": "node momentdemo.js"  
  },  
  "author": "Rujuan Xing",  
  "license": "ISC",  
  "dependencies": {  
    "moment": "^2.29.1"  
  },  
  "devDependencies": {  
    "eslint": "^7.28.0"  
  }  
}
```

Semantic Versioning

- ▶ The Semantic Versioning concept is simple:
all versions have 3 digits: $x.y.z$.
 - ▶ the first digit is the major version
 - ▶ the second digit is the minor version
 - ▶ the third digit is the patch version
- ▶ When you make a new release, you don't just up a number as you please, but you have rules:
 - ▶ you up the **major** version when you make incompatible API changes
 - ▶ you up the **minor** version when you add functionality in a backward-compatible manner
 - ▶ you up the **patch** version when you make backward-compatible bug fixes



Semantic Versioning

- ▶ Why is that so important?
 - ▶ Because `npm` set some rules we can use in the `package.json` file to choose which versions it can update our packages to, when we run `npm update`.
- ▶ The rules use those symbols:
 - ▶ `^`: it's ok to automatically update to anything within this major release. If you write `^0.13.0`, when running `npm update`, it can update to `0.13.1`, `0.14.2`, and so on, but not to `1.14.0` or above.
 - ▶ `~`: if you write `~0.13.0` when running `npm update` it can update to patch releases: `0.13.1` is ok, but `0.14.0` is not.
 - ▶ `>`: you accept any version higher than the one you specify

package-lock.json

- ▶ Introduced by NPM version 5 to capture the exact dependency tree installed at any point in time.
- ▶ Describes the exact tree
- ▶ Guarantee the dependencies on all environments.
- ▶ Use `npm ci` if you want to use dependencies in package-lock.json file
- ▶ Don't modify this file manually.
- ▶ Always use npm CLI to change dependencies, it'll automatically update package-lock.json

```
{
  "name": "lesson03-demo",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "moment": {
      "version": "2.24.0",
      "resolved": "https://registry.npmjs.org/moment/-/moment-2.24.0.tgz",
      "integrity": "sha512-bV7f+6l2QigeBBZSM/6yTNq4P2fNpSWj/0e7jQcy87A8e7o2nAfP/34/2ky5Vw4B9S446EtIhodAzkFCcR4dQg=="
    }
  }
}
```

More About Packages

- ▶ **Development Dependencies:** Needed only while I'm developing the app. It's not needed for running the app.

- ▶ `npm install mocha --save-dev`
`// notice devDependencies entry now in package.json`

- ▶ **Global Dependencies:** Available to all applications

- ▶ `npm install -g nodemon`
 - ▶ `nodemon app.js` `//auto detects changes and restarts your project`

More npm CLI Commands

```
npm -v // will print npm version
npm init // will create package.json
npm install <package> --S // download & install the code from last commit of git repo
                        // "--save" option will update package.json automatically
                        // other options are: --save-dev (-D) --save-optional (-O)
npm i <package> -g // download & install a package globally
npm i <package> --dry-run
npm ls -g --depth=0 // show all global packages in your system
npm update // check versions in package.json and update
npm i npm -g // update npm
npm outdated -g // show all outdated global packages
npm prune // if a package is installed without --save then delete and clean

npm config list // display the default npm settings
npm config set init-author-name "Josh Edward"
npm config delete init-author-name
npm config set save true // automatically --save (-S)

npm search lint // search online for package with lint in the name
npm home <package> // open browser to package homepage
npm repo <package> // open browser to package repository
```



More about HTTP

Understanding Request & Response

- ▶ A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.
- ▶ After receiving and interpreting a request message, a server responds with an HTTP response message.

```
const http = require('http');
http.createServer((req, res) => {
  console.log(req.url, req.method, req.headers);

  res.setHeader('Content-Type', 'text/html');
  res.write('<html>');
  res.write('<head><title>My First Page</title></head>');
  res.write('<body><h1>Hello From Node.js</h1></body>');
  res.write('</html>');
  res.end();
}).listen(3000);
```

HTTP Request: Reading Get and Post Data

- ▶ Handling basic GET & POST requests is relatively simple with Node.js.
- ▶ We use the `url` module to parse and read information from the URL.
- ▶ The `url` module uses the WHATWG URL Standard (<https://url.spec.whatwg.org/>)

href									
protocol		auth		host		path		hash	
				hostname	port	pathname	search		
		user	pass	@	sub.host.com	:	8080	/p/a/t/h	?
protocol		username	password	host		pathname	search	hash	
origin				origin					
href									

Using URL Module

► Parsing the URL string using the WHATWG API:

```
const url = require('url');  
const myURL =  
    new URL('https://user:pass@sub.host.com:8080/p/a/t/h?course1=nodejs&course2=angular#hash');  
console.log(myURL);
```

```
URL {  
  href: 'https://user:pass@sub.host.com:8080/p/a/t/h?course1=nodejs&course2=angular#hash',  
  origin: 'https://sub.host.com:8080',  
  protocol: 'https:',  
  username: 'user',  
  password: 'pass',  
  host: 'sub.host.com:8080',  
  hostname: 'sub.host.com',  
  port: '8080',  
  pathname: '/p/a/t/h',  
  search: '?course1=nodejs&course2=angular',  
  searchParams: URLSearchParams { 'course1' => 'nodejs', 'course2' => 'angular' },  
  hash: '#hash'
```


Parsing the Query String

```
const url = require('url');
const myURL =
    new URL('https://user:pass@sub.host.com:8080/p/a/t/h?course1=nodejs&course2=angular#hash');
let params = myURL.searchParams;
console.log(params);
console.log(params.get('course1'), params.get('course2'));
```

```
URLSearchParams { 'course1' => 'nodejs', 'course2' => 'angular' }
nodejs angular
```

Using querystring module

The `querystring` API is considered Legacy. While it is still maintained, new code should use the `<URLSearchParams>` API instead.

```
const querystring = require('querystring');

const result1 = querystring.stringify({
  firstname: 'Josh',
  lastname: 'Edward'
})

console.log(result1); //firstname=Josh&lastname=Edward

const result2 = querystring.parse('firstname=Josh&lastname=Edward');
console.log(result2); // {firstname: 'Josh', lastname: 'Edward'}
```

HTTP Request: Reading Post Data

- ▶ Handling POST data is done in a **non-blocking way**, by using asynchronous callbacks. Because POST requests can potentially be very large - multiple megabytes in size. Handling the whole bulk of data in one go would result in a blocking operation.
- ▶ To make the whole process non-blocking, Node.js serves our code the POST data in small chunks (**stream**), callbacks that are called upon certain events. These events are `data` (a new chunk of POST data arrives) and `end` (all chunks have been received).
- ▶ We need to tell Node.js which functions to call back to when these events occur. This is done by adding listeners to the `request` object

Reading Post Data & Routing Example

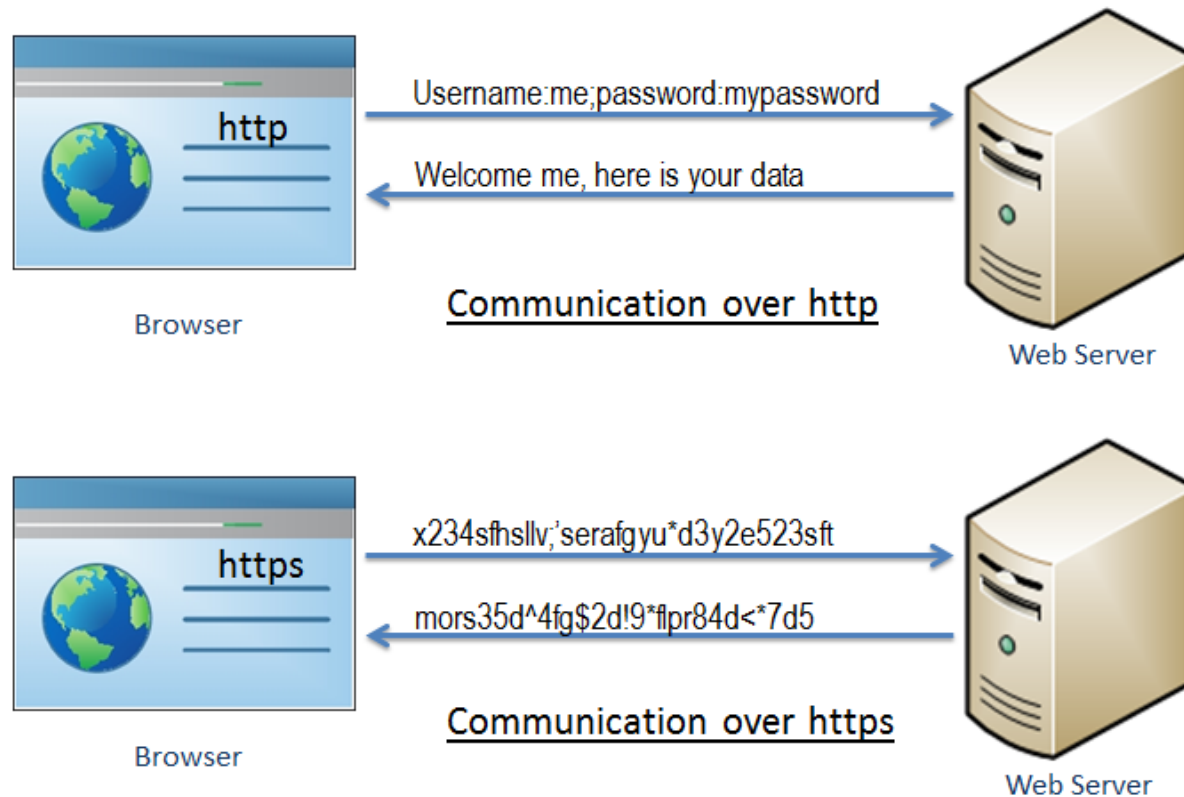
```
const http = require('http');

http.createServer((req, res) => {
  const url = req.url;
  const method = req.method;

  if (url === '/') {
    res.write('<html>');
    res.write('<head><title>Enter Message</title></head>');
    res.write('<body><form action="/message" method="POST">Enter Message: <input name="message"><button type="submit">Send</button></form></body>');
    res.write('</html>');
    res.end();
  } else if (url === '/message' && method === 'POST') {
    const body = [];
    req.on('data', (chunk) => {
      body.push(chunk);
    });
    req.on('end', () => {
      const parsedBody = Buffer.concat(body).toString();
      console.log(parsedBody);
    });
    res.end('Done');
  }
}).listen(3000);
```

HTTPS and Secure Communication

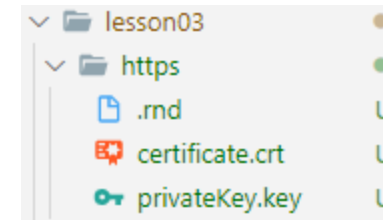
- ▶ HTTPS stands for Hyper Text Transfer Protocol Secure. It is a protocol for securing the communication between two systems e.g. the browser and the web server.
- ▶ The following figure illustrates the difference between communication over http and https:



Generating Keys

1. OpenSSL Windows installer is here
<http://slproweb.com/products/Win32OpenSSL.html>
2. Navigate to the OpenSSL bin directory.
 - ▶ C:\openssl_X64\bin in our example. You can also add openssl.exe as Path environment variable, then you can use command “openssl” under all direcotries.
3. Right-click the **openssl.exe** file and select **Run as administrator**.
4. MacOS has built-in OpenSSL, directly goes to Step 5.
5. Enter the following command to begin generating a certificate and private key:
 - ▶ **req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt**
- ▶ For production environment / deploying to a production server you need to get the keys and certificate from a certification authority (CA) e.g., Verisign, Thawte
<https://helpcenter.gsx.com/hc/en-us/articles/115015960428-How-to-Generate-a-Self-Signed-Certificate-and-Private-Key-using-OpenSSL>

```
OpenSSL> req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 -keyout privateKey.key -out certificate.crt
Loading 'screen' into random state - done
Generating a 2048 bit RSA private key
.....+++
.....+++
writing new private key to 'privateKey.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Iowa
Locality Name (eg, city) []:Fairfield
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MIU
Organizational Unit Name (eg, section) []:MSD
Common Name (eg, YOUR name) []:Rujuan Xing
Email Address []:rxing@miu.edu
```



Test Https Server

1. Download https-demo on Sakai
2. Open in VS Code, run using “node app.js”
3. Open the browser, enter <https://localhost:8443>
4. Click “Advanced” -> “[Proceed to localhost \(unsafe\)](#)”
 - ▶ For Windows user, you’ll be able to see the content sent from https server
 - ▶ For MacOS user, follow the link below to have your certificate added in Keychain access, otherwise doesn’t work.
 - ▶ <https://tosbourn.com/getting-os-x-to-trust-self-signed-ssl-certificates/>

Create HTTPS Server

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

const options = {
  key: fs.readFileSync('./privateKey.key'),
  cert: fs.readFileSync('./certificate.crt')
};

const server = require('https')
  .createServer(options);

server.on('request', (req, res) => {
  res.writeHead(200, { 'content-type': 'text/plain' });
  res.end('Hello from my HTTPS Web server!!!\n');
});

server.listen(443);
```


Resources

▶ Node and NPM

- ▶ [Nodejs.org](https://nodejs.org)
- ▶ [Npmjs.com](https://npmjs.com)
- ▶ [Node API Documentation](https://nodejs.org/en/docs/guides/node-api/documentation)
- ▶ [NPM Documentation](https://docs.npmjs.com)

▶ HTTPS

- ▶ [HTTPS \(Wikipedia\)](https://en.wikipedia.org/wiki/HTTPS)
- ▶ [Public Key Cryptography](https://en.wikipedia.org/wiki/Public-key_cryptography)
- ▶ [Transport Layer Security](https://en.wikipedia.org/wiki/Transport_Layer_Security)
- ▶ [Node HTTPS Server](https://nodejs.org/en/docs/guides/https-server)
- ▶ [Kurose, James F., and Keith W. Ross. Computer networking: a top-down approach. Pearson, 2017, ISBN-10: 0134522206 • ISBN-13: 9780134522203.](#)

▶ Other Resources

- ▶ [Howto: Make Your Own Cert With OpenSSL on Windows](#)
- ▶ [OpenSSL for Windows](#)
- ▶ [How to Use SSL/TLS with Node.js](#)
- ▶ [Adding HTTPS \(SSL\) to Express 4.X Applications](#)
- ▶ [How does HTTPS actually work?](#)