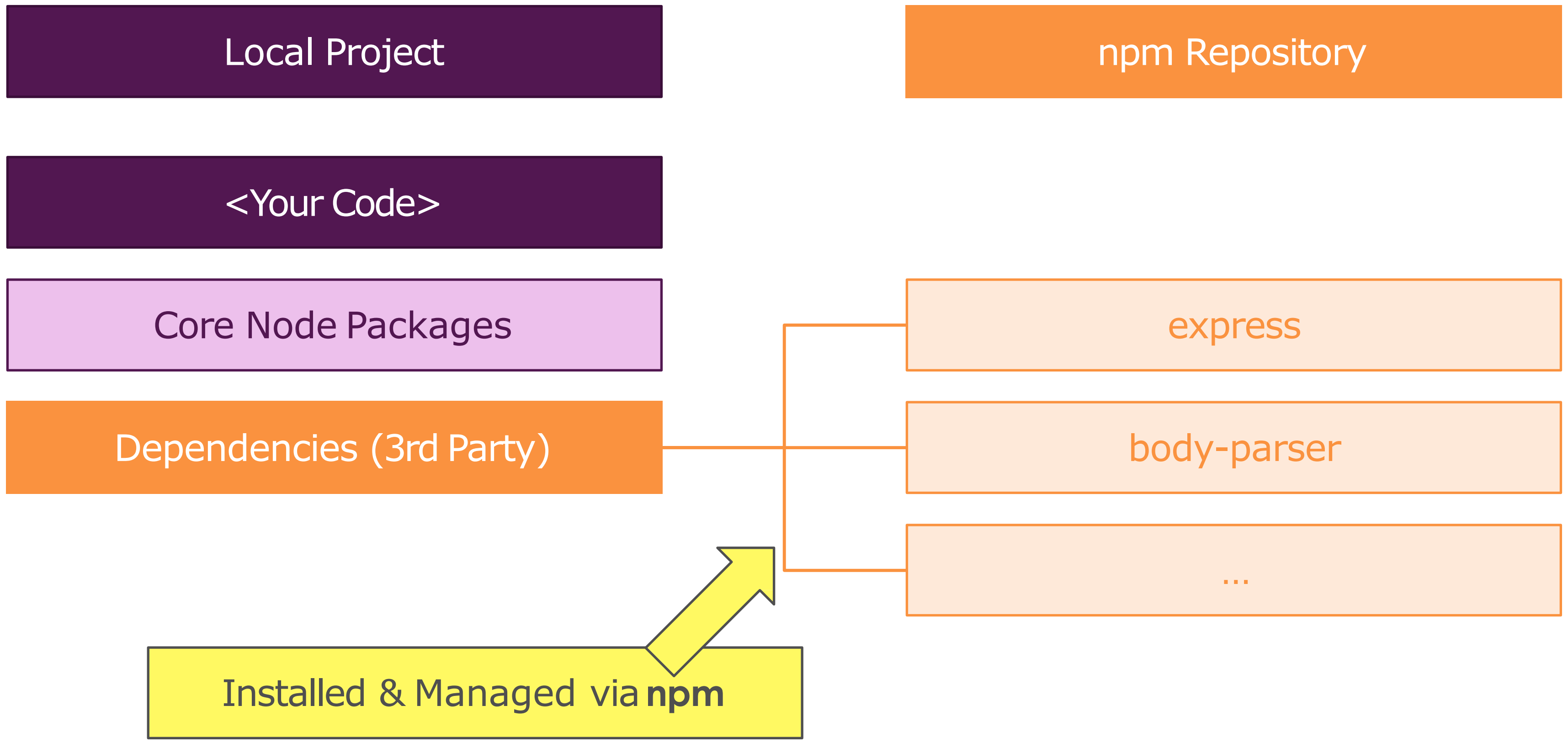
NPM & Modules

Rujuan Xing

npm & packages Intro



What is npm?

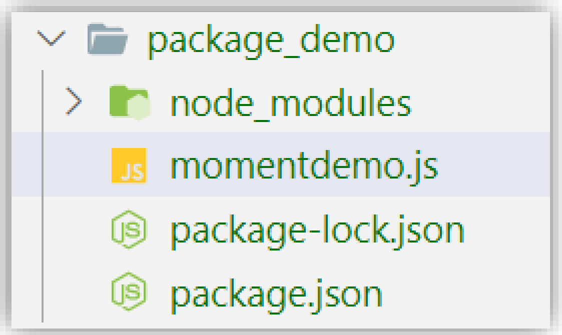
- **npm** is the standard package manager for Node.js. It also manages downloads of dependencies of your project.
- www.npmjs.com hosts thousands of free packages to download and use.
- The NPM program is installed on your computer when you install Node.js.
  - `npm -v // will print npm version`

- 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
```



```
package_demo
  node_modules
  momentdemo.js
  package-lock.json
  package.json
```

- 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
```

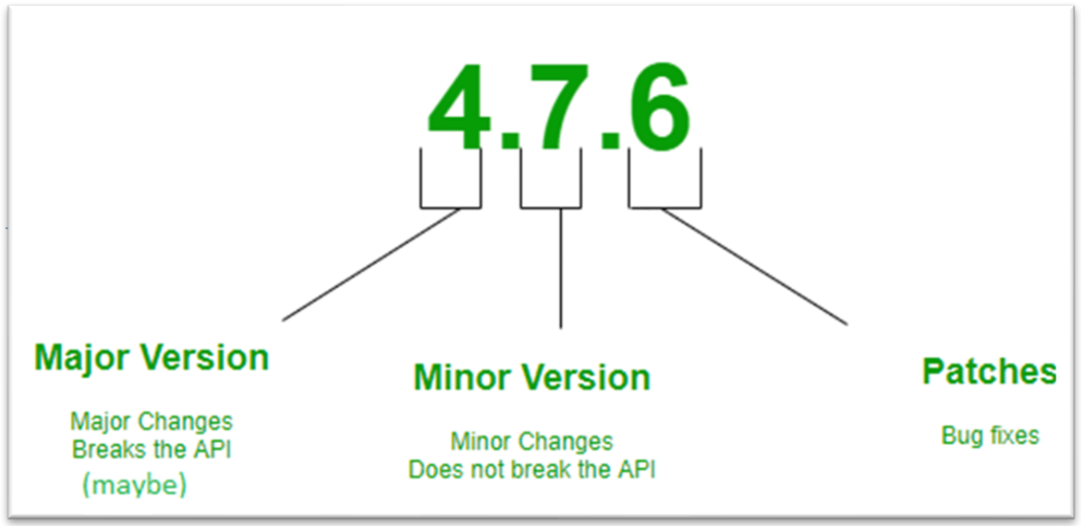
package.json Manifest

- 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

```json
{
    "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

More details about 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

```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/34/2ky5
Vw4B9S446EtIhodAzkFCcR4dQg=="
    }
  }
}
```

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 l // 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
```

HTTP

Node as a Web Server

- Node started as a Web server and evolved into a much more generalized framework.
- Node `http` module is designed with streaming and low latency in mind.
- Node is very popular today to create and run Web servers.

Web Server Example

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

`http.IncomingMessage`
Implements `ReadableStream` Interface

`http.ServerResponse`
Implements `WritableStream` Interface

```
server.on('request', function(req, res) {
                          res.writeHead(200, {'Content-Type': 'text/plain'});
                          res.write('Hello World!');
                          res.end();
                    });
server.listen(3000);
```

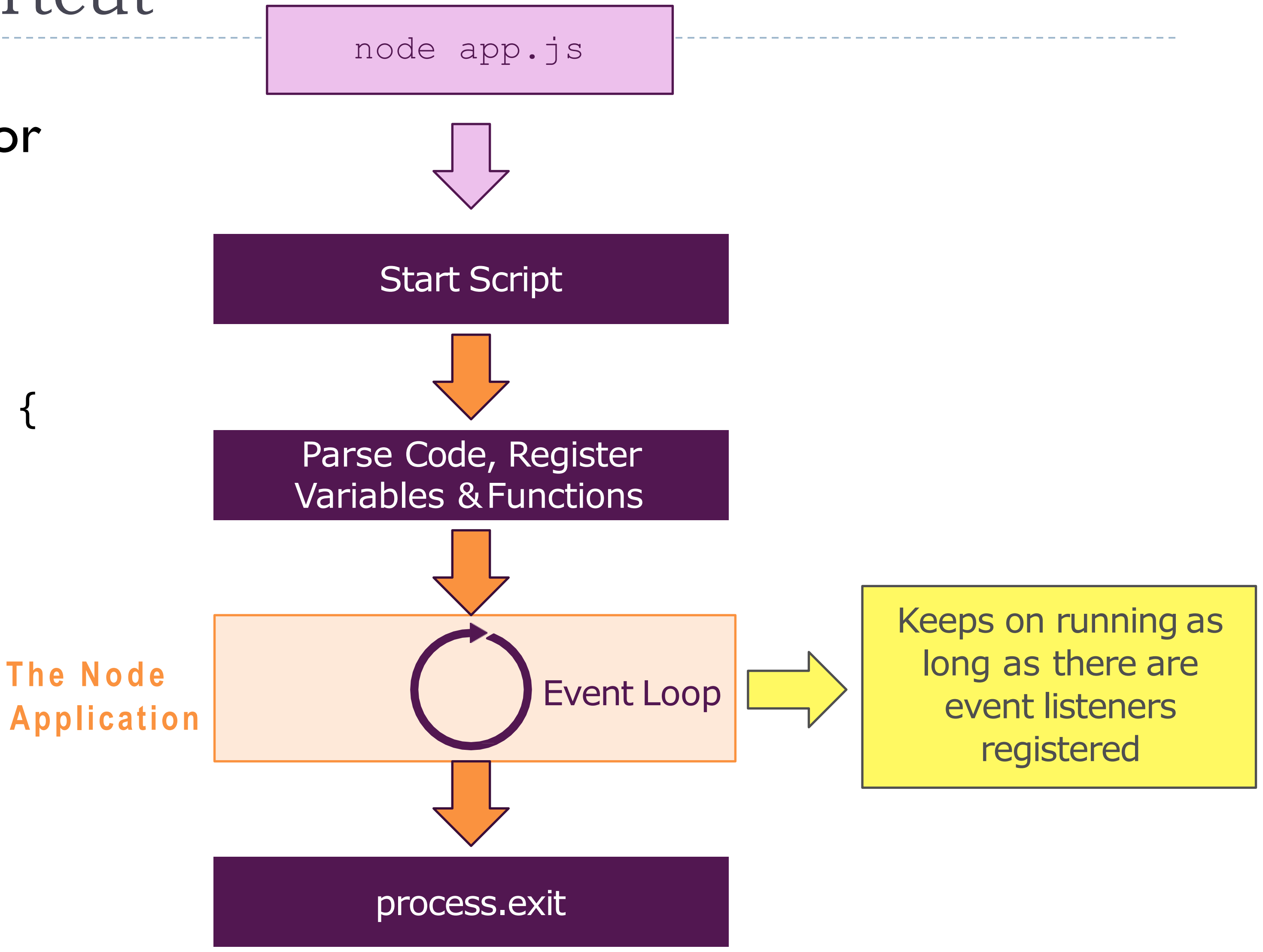
After we run this code. The node program doesn't stop.. it keeps waiting for request

Web Server Example Shortcut

- Passing a callback function to `createServer()` is a shortcut for listening to "request" event.

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

http.createServer(function (req, res) {
    res.writeHead(200, { 'Content-
Type': 'application/json' });
    const person = {
        firstname: 'Josh',
        lastname: 'Edward'
    };
    res.end(JSON.stringify(person));
}).listen(3000, '127.0.0.1');
```


node app.js
Start Script
Parse Code, Register Variables & Functions
The Node Application
Event Loop
Keeps on running as long as there are event listeners registered
process.exit

Send out an HTML file

- What's the problem with the code below?

```
index.html
<html>
        <head></head>
        <body>
                <h1>{Message}</h1>
        </body>
</html>
```

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

http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/html' });
    let html = fs.readFileSync(path.join(__dirname, 'index.html'), 'utf8');
    html = html.replace('{Message}', 'Hello from Node.js!');
    res.end(html);
}).listen(3000, '127.0.0.1', () => { console.log('listening on 3000...') });
```

Let's create a big file!

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

const file = fs.createWriteStream('./big.file');

for(let i=0; i<= 1e6; i++) {
  file.write('Lorem ipsum dolor sit amet, consectetur adipisicing elit,
  sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
  enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
  aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
  in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
  Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
  officia deserunt mollit anim id est laborum.\n');
}

file.end();
```

Reading the file

- What's going to happened to the Node process in memory? Will this code still work with 2 GB file or more?

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

http.createServer((req, res) => {
    fs.readFile('./big.txt', (err, data) => {
        if (err) throw err;

        res.end(data);
    });
}).listen(3000, () => console.log('listening on 3000'));
```

A Simpler solution – Use Stream

- We can simply use stream.pipe(), which does exactly what we described.

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

const server = require('http').createServer();

server.on('request', (req, res) => {
      const src = fs.createReadStream('./big.file');
      src.pipe(res);
});

server.listen(8000);
```

Understanding Request & Response

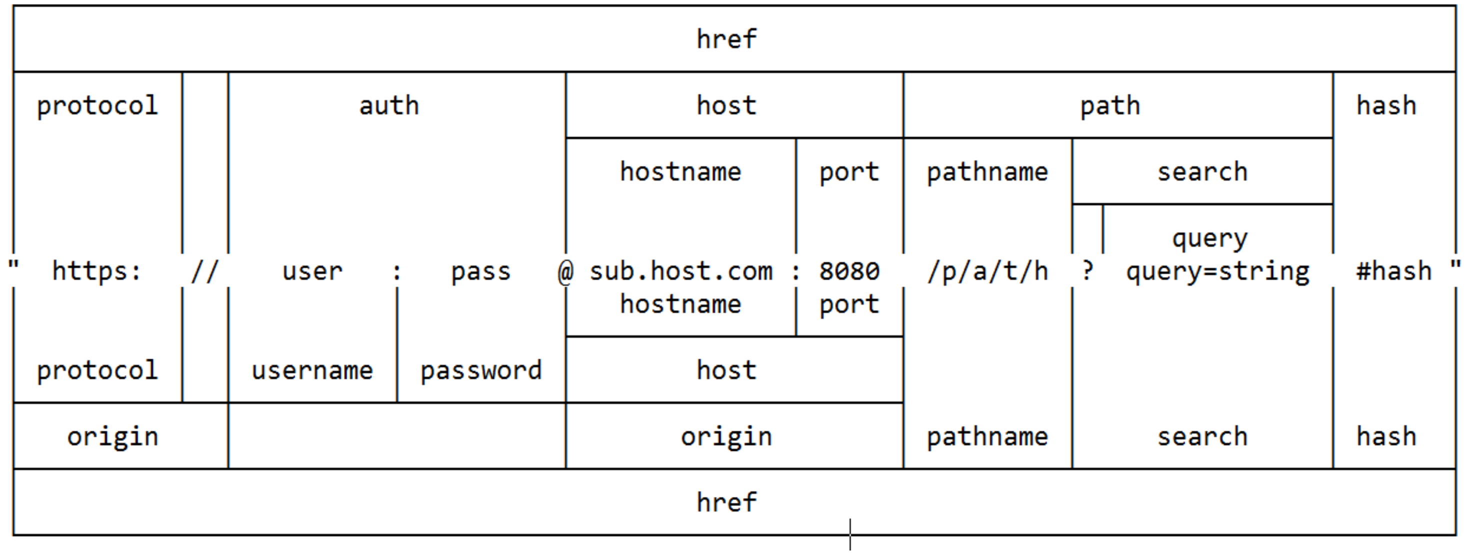
- A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.
- After receiving and interpreting a request message, a server responds with an HTTP response message.

```
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

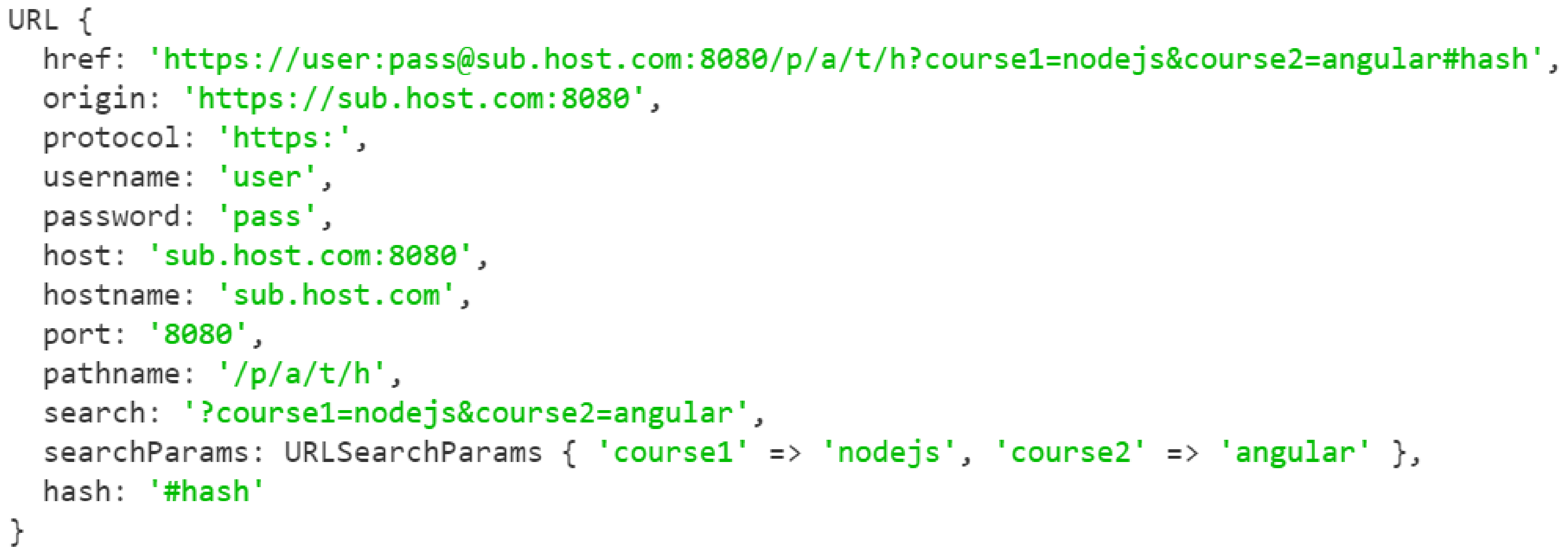
- Handling basic GET & POST requests is relatively simple with Node.js.
- We use the `url` module to parse and read information from the URL.
- The `url` module uses the WHATWG URL Standard (https://url.spec.whatwg.org/)



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=angula
r#hash');
let params = myURL.searchParams;
console.log(params);
console.log(params.get('course1'), params.get('course2’));
```

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

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="/messsage" method="POST">Enter Message: <input name="message"><button type="submit">Send</button></form></body>');
        res.write('</html>');
        res.end();
    } else if (url === '/messsage' && 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);
```