Node JS - Day 1

NorthWestern Mutual

About me - Alain Chautard (or just Al)

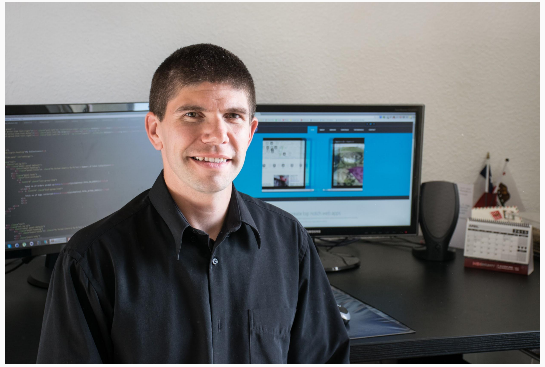
Google Developer Expert in Web technologies / Angular

Java developer since 2006

Angular JS addict since 2011

Web consultant (60%) / trainer (40% of the time)

Organizer of the Sacramento Angular Meetup group



Quick Poll

- How many of you are Java developers? C#, .Net?
- How many of you are developers? Full-stack? Back-end?
- Any experience with Javascript? TypeScript? Node.JS?
- jQuery?
- Any other Javascript framework?

How we're going to work

- Your questions are welcome, anytime!
- Being a web developer requires constant learning
- My goal is to give you the tools to work efficiently with web technologies - We’re going to practice a lot!
- As a result, we will be going through some online documentation when needed

Useful links

- Repository for all labs code + solutions: https://github.com/alcfeoh/di-node-js
- Link to these slides: https://goo.gl/qcqPcB

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Introduction to Node.JS

What is Node.JS?

- Node.JS is a **JavaScript** runtime built on **Chrome's V8 JavaScript engine**.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. As result, most operations in Node.JS are **asynchronous**.
- Node.js' package ecosystem, **npm**, is the largest ecosystem of open source libraries in the world.
- Official website: https://nodejs.org/en

Why Node.JS?

- Unlike older browsers, **Chrome was built for Javascript first**. As a result, its engine is really fast.
- If Javascript runs fast in the browser, why not using it on the server or on any desktop as well?
- As a result, the exported Chrome V8 engine became Node.JS, with the addition of extra libraries to do things that are not allowed in JS: Reading files, connecting to databases, etc.

Lab 1 - Hello Node

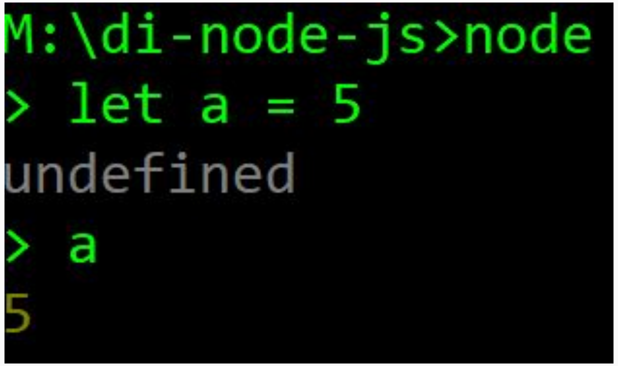
- In this lab, we're going to write a simple Node.JS application that just says "Hello Node"
- **Your mission:** In order to run Node on your machine, first install the LTS version from: https://nodejs.org/en/
- Create a file **`hello-node.js`** and write the following code in it:

```
console.log("Hello Node");
```

- Then open a terminal in the folder where **`hello-node.js`** is located and run: **`node hello-node.js`**

How to use Node.JS?

- Node.JS is added to the path of your operating system. As a result, you can simply use it with the command: **`node [filename]`**
- Node.JS can also be used in interactive mode, just like the browser console of Google Chrome. To do so, simply run: **`node`**



```
M:\di-node-js>node
> let a = 5
undefined
> a
5
```

- To get the current version of Node: **`node --version`**

What we just learnt

Node.JS is a JavaScript runtime built on Chrome's V8 JavaScript engine

Node runs from a simple command line: **node**

Node uses an event-driven, non-blocking I/O model that makes it lightweight and efficient

Node has extra libraries to do things that are not allowed in JS: Reading files, connecting to databases, etc.

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Introduction to NPM

What is NPM?

- **N**ode **P**ackage **M**anager is two things:
  - Online repository for publishing open-source code
  - Command-line utility for interacting with said repository that aids in package installation, version management, and dependency management
- Npm is automatically installed with Node.js
- Npm is the most popular package manager for the Web

Npm install

- **`npm install`** is the most used command with npm

- For instance, running **`npm install typescript`** will download Typescript and all of its dependencies into a **`node_modules`** folder created in the current folder

- As a result, most installs are specific to one project

- In order to install a package globally, you can use the **-g** option:

```
npm install -g typescript
```

Npm install, npm update, npm uninstall

- It is also possible to specify which version we want to install:

```
npm install my-package@1.2.0
```

- You can also use npm update to get the latest version of all packages and dependencies of your current project:

```
npm update
```

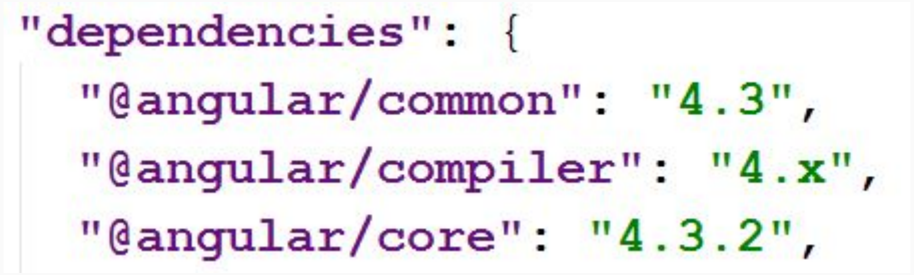
- How to uninstall?

```
npm uninstall my-package
```

package.json

- Production projects have several dependencies with specific version requirements that have to be enforced for all developers

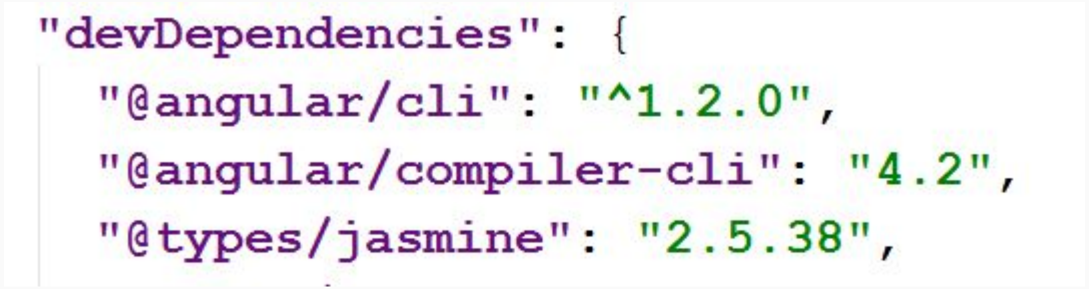
- **`package.json`** is a JSON file that can store all of these versions in one single reference file:



```
"dependencies": {
  "@angular/common": "4.3",
  "@angular/compiler": "4.x",
  "@angular/core": "4.3.2",
```

package.json

- **`package.json`** can be generated by running **`npm init`**
- When running **`npm install`** in a folder, **`npm`** will look for a **`package.json`** file, and if there is one, will use it to download all of the dependencies listed in it.
- Note that **`package.json`** also has a specific section for development-specific dependencies that should not be part of your production code:



```
"devDependencies": {
  "@angular/cli": "^1.2.0",
  "@angular/compiler-cli": "4.2",
  "@types/jasmine": "2.5.38",
```

Npm install with package.json

- It is possible to persist our installs in package.json automatically:

```
npm install my-package --save
```

- The same also applies to dev-only dependencies:

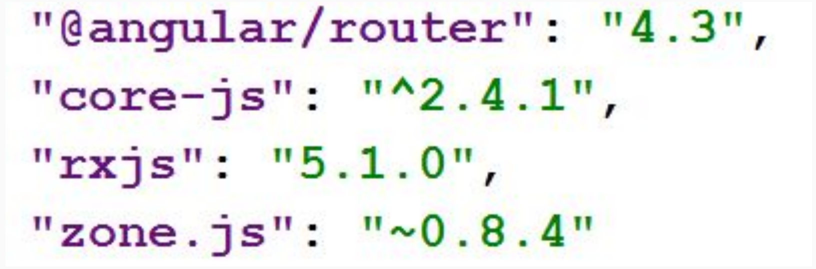
```
npm install my-package --save-dev
```

- And this works with uninstall as well:

```
npm uninstall my-package --save
```

Specifying version constraints with NPM

- If we want an exact version: **`1.2.4`**
- If we want patch releases: **1.2** or **1.2.x** or **~1.2.4**
- If we want minor releases: **1** or **1.x** or **^1.2.4**
- If we want major releases: ***** or **x** or **latest**



```
"@angular/router": "4.3",
"core-js": "^2.4.1",
"rxjs": "5.1.0",
"zone.js": "~0.8.4"
```

Lab 2 - Getting dependencies with NPM

- In this lab, we’re going to write a simple Node.JS application that just says “Hello Node” using a styling library called **Chalk**
- **Your mission:** Install Chalk with npm and use it to display “Hello Node” in blue color over a white background.
- Create a file **`hello-chalk.js`** and write the following code in it:

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

- Then use the Chalk documentation to achieve the desired styling: https://www.npmjs.com/package/chalk

What we just learnt

NPM is both an online repository for publishing open-source code and a command-line utility to interact with that repository

NPM comes with Node.JS

`npm install` is the command used to download packages

`package.json` can be used to store all of your dependency constraints in one place

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Command line apps with Node.js

How to pass arguments to a Node.JS app?

- Any arguments passed to a Node.JS app can be accessed via: **`process.argv`**
- For instance, given the following command:

```
node myapp.js aParamValue
```

- **`process.argv[0]`** is equal to **`C:\Path\to\nodejs\node.exe`**
- **`process.argv[1]`** is equal to **`C:\Path\to\myapp\myapp.js`**
- **`process.argv[2]`** is equal to **`aParamValue`**
- Any other parameters would be added to that array at indexes 3, 4, etc.

Lab 3 - A simple command line app

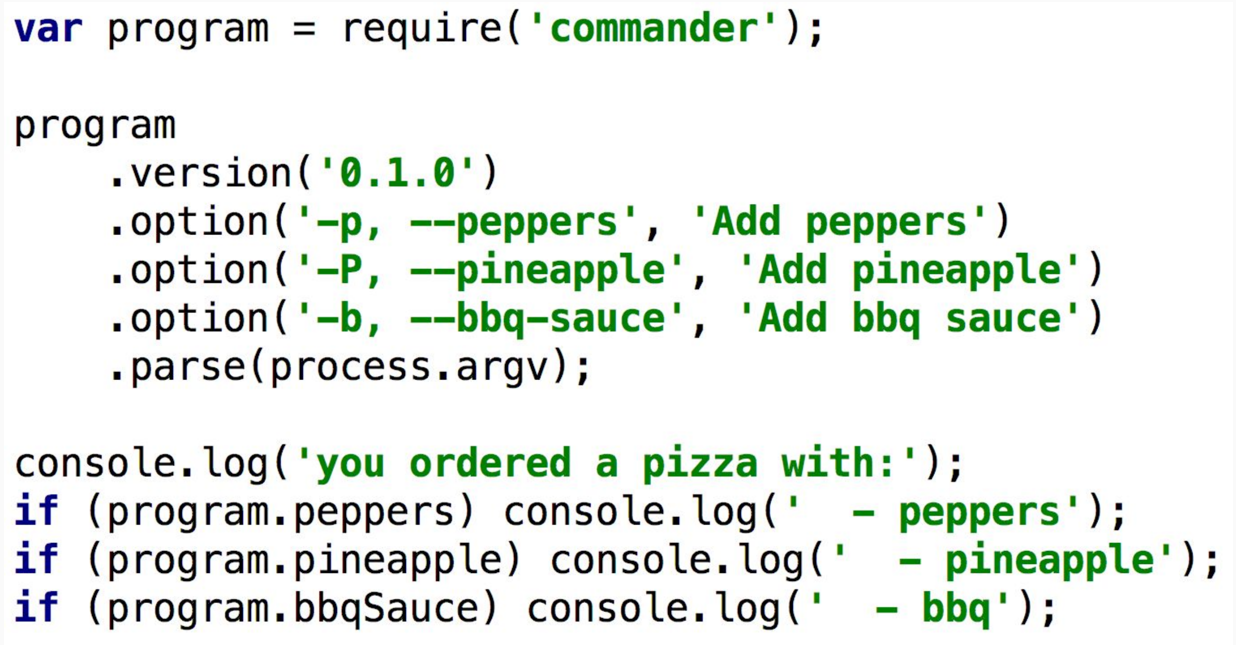
- In this lab, we're going to make a simple command line app that listens to file updates on a given file. Every time the file is edited, it should display "The file has changed"

- **Your mission:** Create a file **`notify-updates.js`** that can be run as follows: **`node notify-updates.js sample-file.txt`** where **`sample-file.txt`** is a parameter and could be any file name.

- **Hint:** In order to listen to file system changes, use the following documentation: https://nodejs.org/api/fs.html#fs_fs_watchfile_filename_options_listener

Other options for command line apps: Commander

- Commander is a simple package that allows you to define different options for your command line app (https://www.npmjs.com/package/commander):



```
var program = require('commander');

program
    .version('0.1.0')
    .option('-p, --peppers', 'Add peppers')
    .option('-P, --pineapple', 'Add pineapple')
    .option('-b, --bbq-sauce', 'Add bbq sauce')
    .parse(process.argv);

console.log('you ordered a pizza with:');
if (program.peppers) console.log('  - peppers');
if (program.pineapple) console.log('  - pineapple');
if (program.bbqSauce) console.log('  - bbq');
```

Other options for command line apps: Prompt

- Prompt is a simple package that allows you to interact with the user by asking questions (https://github.com/flatiron/prompt):

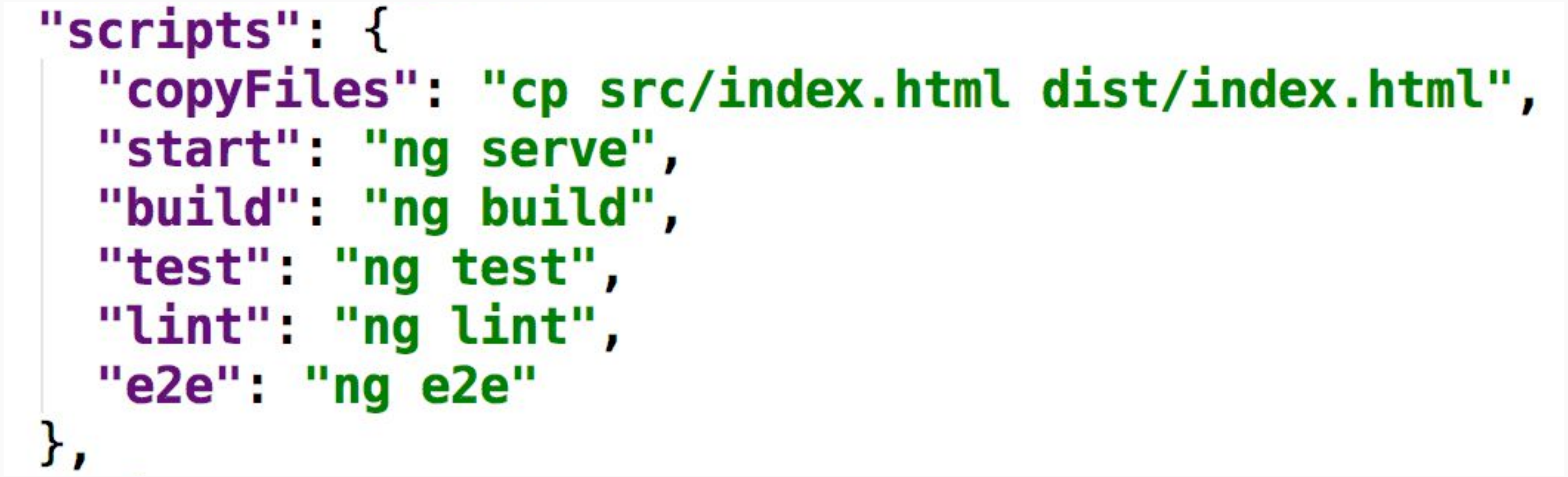
```
var prompt = require('prompt');

prompt.start();

// Get two properties from the user: username and email
prompt.get(['username', 'email'], function (err, result) {
    console.log('Command-line input received:');
    console.log('  username: ' + result.username);
    console.log('  email: ' + result.email);
});
```

Scripts in package.json

- It is also possible to create custom scripts in **`package.json`** that can be run with **`npm run [script-name]`**:



```
"scripts": {
  "copyFiles": "cp src/index.html dist/index.html",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
```

What we just learnt

Node provides several tools to create command line apps

Arguments can be passed to Node applications

`Commander` and **`prompt`** are two node packages that can be used to add more options and interactivity in our command line app

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Global and process objects

What is the global object?

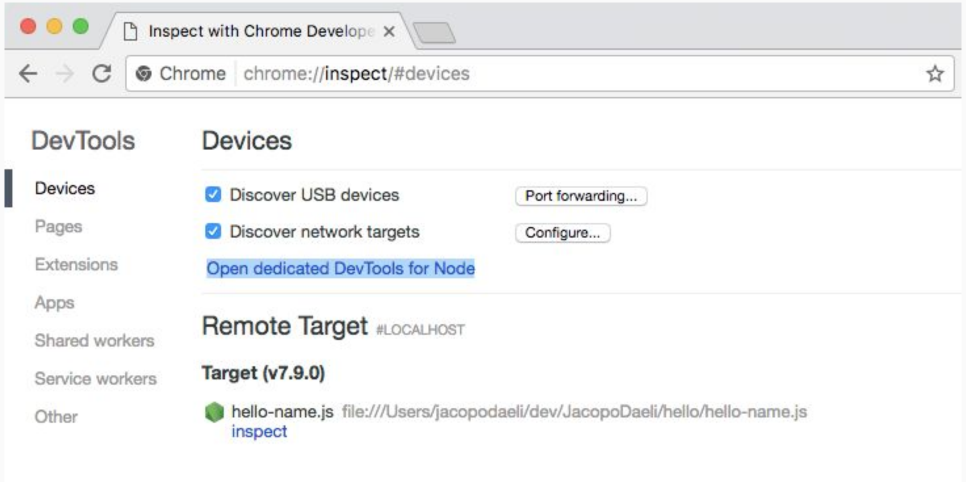
- The global object is a global namespace object
- It contains references to all global objects like:
  - **`console`** - an object used to log some information
  - **`require()`** - the function used to load external modules
  - **`process`** - an object that has information about the current Node process
  - **`setTimeout()`**, **`setInterval()`** - timer functions used to run code asynchronously

What is the process object?

- The **`process`** object provides information about, and control over, the current Node.js process - https://nodejs.org/api/process.html
- As a global, it is always available to Node.js applications without using **`require()`**
- Some examples of properties and methods of that object:
    - **`argv`**- array of parameters passed to the process
    - **`cwd()`** - returns the current working directory
    - **`env`** - returns system environment variables for the current user

How to debug Node.JS applications?

- We can hook Node.JS to the Google Chrome Developer tools in order to debug Node.JS code: https://nodejs.org/api/debugger.html
- To do so, open **`chrome://inspect`** in Chrome
- Then run your Node.js app as follows: **`node --inspect-brk file.js`**



Devices

✓ Discover USB devices

✓ Discover network targets

Remote Target #LOCALHOST

Target (v7.9.0)

hello-name.js file:///Users/jacopodaeli/dev/JacopoDaeli/hello/hello-name.js

Lab 4 - Debugging Node.JS code with Google Chrome

- In this lab, we’re going to debug a simple app to see the contents of both the **`process`** and **`global`** objects.

- **Your mission:** Create a file **`env-global.js`** that will get a reference to both **`process`** and **`global`** objects. Then use the Chrome dev tools to inspect the contents of these objects.

- **Hint:** You can add a breakpoint to your code with the following statement:

```
debugger;
```

What we just learnt

Node.JS has two objects the contain global information about the runtime environment

`global` gives use access to regular Javascript features like **`console`** and **`setTimeout()`**

`process` gives us access to system information like the current working directory (**cwd**) and the runtime environment (**env**)

We can debug Node.JS applications using Google Chrome Developer Tools

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Asynchronous programming

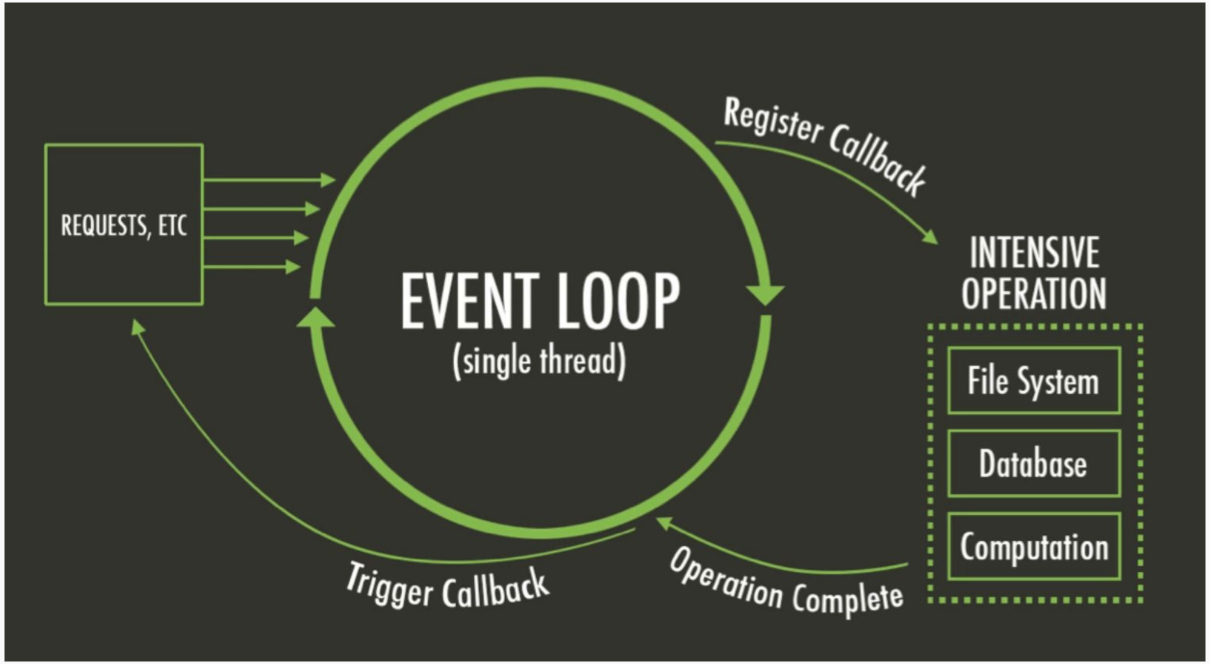
Why is asynchronous programming important?

- Node.JS is **single-threaded**, yet claims to have an event-driven, non-blocking I/O model that makes it lightweight and efficient.
- How can Node be non-blocking if it has a single-thread? How could it possibly handle multiple tasks at once in that case?
- The answer lies in a mechanism called the **event loop**
- Whenever a task is handled **asynchronously**, Node.JS will make it run within the event loop, thus freeing Node’s main thread from that task

What is the event loop?

It’s a mechanism that delegates tasks to worker threads of the OS

Node periodically checks for updates from the event loop to be notified when tasks are completed



How to handle asynchronous code?

Whenever a task is asynchronous, we register a callback function that will be invoked whenever the task completes.

It is a very common practice in Node.JS: Callbacks are everywhere!

```
fs.readFile('/etc/passwd',
   (err, data) => {
         console.log(data);
   }
);
```

DEMO - To the event loop and back

What we just learnt

Node.JS relies on the event loop for asynchronous work, then callback functions handle the end result

The event loop can delegate tasks to the operating system

As a result, asynchronous programming is the key to efficient, non-blocking Node.JS applications

Callback functions notify us when an asynchronous task completes

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

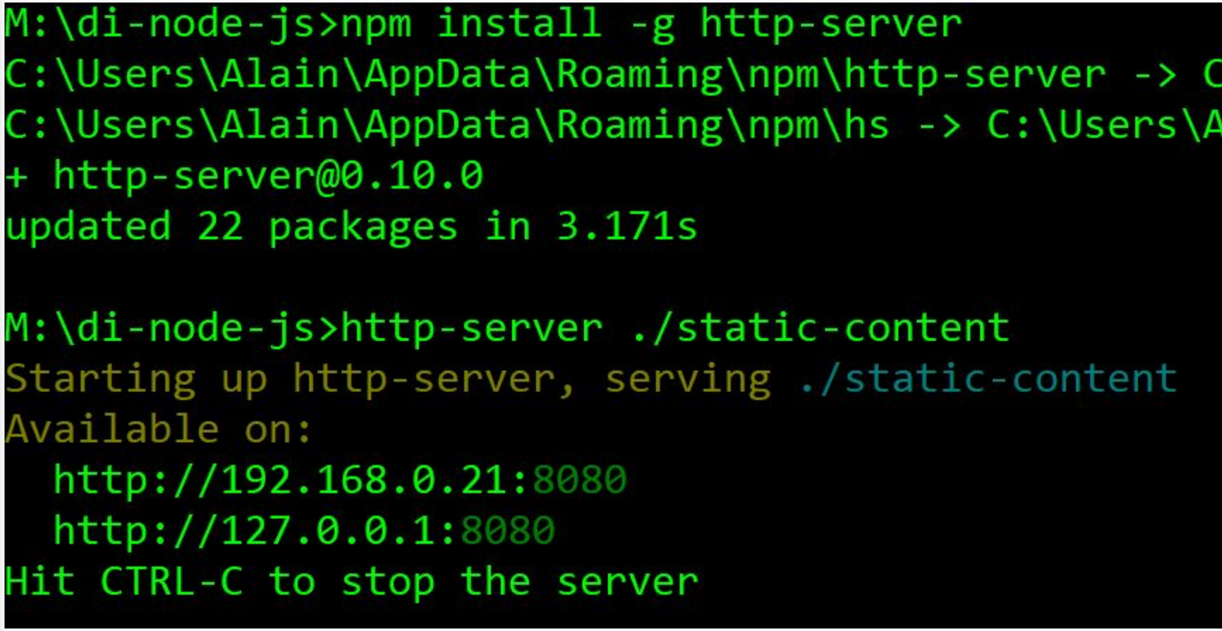
Creating Node modules

Creating HTTP servers / Making HTTP requests

How to serve static files?

The easiest way to serve static content is to use the **`http-server`** module

It starts a local server on port 8080 that serves content from a local folder of your choice



```
M:\di-node-js>npm install -g http-server
C:\Users\Alain\AppData\Roaming\npm\http-server -> C
C:\Users\Alain\AppData\Roaming\npm\hs -> C:\Users\A
+ http-server@0.10.0
updated 22 packages in 3.171s

M:\di-node-js>http-server ./static-content
Starting up http-server, serving ./static-content
Available on:
  http://192.168.0.21:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

How to write a basic HTTP server?

You can use the default **`http`** module from Node.JS

The example on the right would always return the same string no matter the URL

```
const http = require('http');
const port = 3000;

const server = http.createServer(
 (req, resp) => {
   console.log(req.url);
   resp.end('Hello Node.js Server!')
  }
);

server.listen(port, (err) => {
   console.log(`server listening on ${port}`)
});
```

Lab 5 - Basic HTTP servers

- **Your mission #1:** Install `http-server` globally with npm and start a server in the `static-content` folder of our labs. Then go to `localhost:8080` in your browser to see the static content being served.

- **Your mission #2:** Create a new file **http.js** and use the **http** module to create a basic server that runs on port 8000. The server should work as follows:
  - When `/hello` is accessed, it should return `Hello Node.js Server!`
  - When `/data` is accessed, it should return the contents of the file `data/plates.json`
  - Any other URL should return `Nothing to see here`

- **Hint:** Use the `fs` module to read a file with Node.JS

How to write a more advanced HTTP server?

Many modules exist to do that. **Hapi** is one possible option.

Hapi is not a default Node.JS module and has to be installed through npm

```
const Hapi = require('hapi');

// Create a server with a host and port
const server = new Hapi.Server();
server.connection({
  host: 'localhost', port: 8000
});

// Add a route
server.route({
   method: 'GET',  path:'/hello',
   handler: function (request, reply) {
       return reply('Hello Hapi World');
   }
});

// Start the server
server.start((err) => {
   console.log('Server running');
});
```

Lab 6 - Using Hapi

- **Your mission:** Create a new file **`hapi.js`** and use the **`hapi`** module (at version **16.x**) to create a basic server that runs on port 8000. The server should work as follows:
  - When **`/hello`** is accessed, it should return **`Hello Hapi Server!`**
  - When **`/data`** is accessed, it should return the contents of the file **`data/plates.json`**
  - Any other URL should return **`Nothing to see here`**

How to make a HTTP request?

The **`http`** module of Node.JS is the default option but is a verbose one.

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

http.get('http://localhost/data', (res) => {

    res.setEncoding('utf8');
    let rawData = '';

    res.on('data', (chunk) => {
        rawData += chunk;
     });

    res.on('end', () => {
        const data = JSON.parse(rawData);
        console.log(data);
     });
});
```

Making a HTTP request with **`node-fetch`**

A much easier solution based on promises

```
const fetch = require("node-fetch");

const url = "http://localhost/data";

fetch(url).then(response => {
       response.json().then(json => {
           console.log(json);
       });
   })
   .catch(error => {
       console.log(error);
   });
```

Lab 7 - Using node-fetch to make a HTTP request

- **Your mission:** Create a new file **`request.js`** and use the **`node-fetch`** module to create a basic HTTP request that gets the data from our **`hapi`** server and outputs it to the **`console`**
- **Hint:** Remember to install **`node-fetch`** first

What we just learnt

Node has a default **`http`** module that can be used to create both HTTP servers and requests

`http-server` is a better option for a simple static file server

`express` is a popular option for more complex HTTP servers

`node-fetch` is a module for easy HTTP requests

Outline for today

Introduction to Node.JS

Introduction to NPM

Command line apps with Node.JS

Global and process objects

Asynchronous programming

Creating HTTP servers / Making HTTP requests

Creating Node modules

Creating node modules

How to create our own Node.JS module?

- Create a new directory and then run **`npm init`** in that directory
- An interactive prompt will ask for information about your module
- Then a default **`package.json`** will be created in that directory
- Create an index.js file with the contents of your Node module
- Once your module is ready to ship, use **`npm publish`** to make your module available to the rest of the world!
- **Note:** You will need to run **`npm login`** before being able to publish, which requires a npm account

Lab 8 - Creating your own node module

- **Your mission:** Create a new module by following the instructions from the previous slide. Your module should have a **`helloWorld`** function that does:

```
console.log('Hello world from NPM module!');
```

- Then publish your module to the npm repository
- Once your module is published, go to a different directory and run npm install to download your module. Write a test.js file that imports your module and calls its **`helloWorld()`** function
- **Hint:** Start with **`npm init`**

What we just learnt

Node modules can be created and published from simple command line instructions

`npm init` creates a default package.json for your module

`npm publish` pushes your module to the public npm repository

Thanks for your attention

I need your feedback before you leave:
http://bit.ly/lsnode2-22-18

See you tomorrow for day 2