A Webserver using NodeJS and Express

- Writing a webserver to serve a web application
- Many parts
- We'll learn each as we go

Setup

Create a work folder named `hello-node`

- Not within repository
- `kebab-case` name matters (more soon)

Create a `hello.js` file

```
'use strict'; // Don't forget!
const message = 'Hello';

console.log( `${message} World!` );
```

- Ensure it runs: `node hello.js`

`require()` loads an outside file of code

Node runs on a system and not a "page"

- Can easily load additional files
- "modules" will "export" code
- Requiring a module gets exported value
    - Object, String, Function, etc

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

assert.strictEqual(1, 1); // "assert" is object with method
console.log('it only gets this far if assert is happy');
```

- This is **CommonJS**
    - NOT **ES Modules (ESM)**
    - More on ESM later

The file you `require()`

- Code in file will run once
    - Even if `require()`ed multiple times
- `module.exports` defines exported value

```
// message.js
'use strict'; // Each file should have this
console.log('message.js ran once');

const message = 'Hi there';

module.exports = message; // Exports value, not variable
```

You can export any JS value

```
module.exports = {
  one: 1,
  two: 2,
};

module.exports = 'boring';

module.exports = [ 'a', 'b', 'c' ];

module.exports = function( word ) {
    return word.toLowerCase().replace(/\s/g, '-');
};

module.exports = function() { // a "closure"
  const count = 1;
  return function() {
    return count++;
  };
};
```

`require()` returns value

- You might get part of the exported value:
- What do these lines imply about exported values?

```
const foo = require('./foo').cat;

const bar = require('./bar')();

const { somePart, anotherPart } = require('./baz');
```

Our example

```
// hello.js
'use strict'; // Don't forget!

const message = require('./message');

console.log( `${message} World!` );
```

`require()` is given a path + filename

- No path implies installed/built-in **library**
- **Explicit path** is a **local file**
    - `.` is same folder as this `.js` file
    - `..` is parent folder ("up a level")
- `.js` file extension allowed, not required

Examples

- `./cat` is `cat.js` in this folder
- `./hungry/cat` is `cat.js` in `hungry` subfolder
- `cat` is a separate library called `cat`
- `cat.js` is a separate library called `cat.js`

Each module is a separate variable scope

```
// jorts.js
const name = 'Jorts';
module.exports = name;

// jean.js
const friend = require('./jorts');
const name = 'Jean';
module.exports = name;

// cats.js
const name = require('./jorts');
const rescuer = require('./jean');

console.log(name); // 'Jorts'
console.log(rescuer); // 'Jean'
```

Module-related Names

- Module filenames are generally **kebab-case**
  - ex: `cat-names`
- Imported variables are generally **camelCase**
  - `const catNames = require('./cat-names');`
  - **MixedCase** for constructors/JS classes
    - Later: React Components
  - **CONSTANT_CASE** for actual constants

Overlapping concepts

package vs **module** vs **library** vs **framework**

- **package** - can run `npm install` on it
- **module** - can `require()` or `import` (later)
- **library** - provides code for use
- **framework** - creates app type; has code demands

Any file:

- Part one of these
- Part of more than one of these
- None of these