Modules, NPM and packages

JS

npm

Express JS

sebinbenjamin

What we'll do learn today

- Recap
- Getting started
  - Executing js code in Node
  - Writing functions
- Importing/Exporting functions and variables
- Node Package Manager
- NPM CLI - init, install, uninstall, list
- Core Modules
  - HTTP Module
- Computer Networking

Review - Modern Javascript concepts

- `var`, `let` vs `const` - https://www.freecodecamp.org/news/var-let-and-const
- => fat arrow notation
  - Curly brackets is optional if only one statement/expression is present in the body of the function
  - Return is optional similarly
- Object Destructuring
  - `const {id, is_verified} = user;`
- Template literalals
  - `console.log(`I love${car}and ${anotherCar}`)`
- Conditional (ternary) operator
  - condition ? exprIfTrue : exprIfFalse

Node.js and modules

- In Node.js, each file is treated as a separate module.
  - All code is executed in V8 as IIIFE
- Each module is an independent entity with its own encapsulated functionality, it can be managed as a separate unit of work.
  - Variables local to the module will be private, because the module is wrapped in a function by Node.js

Importing/Exporting local modules

- Local modules are modules created locally in your Node.js application.
    - These modules include different functionalities of your application in separate files and folders.
- There are two main ways of exporting things from a module.
    - Multiple named exports
    - A single default export
- Used to export functions, objects, or primitive values from the module so they can be used by other programs with the import statement.

Multiple named exports

```
module.exports.someName = functionExpressionName;


    //use require() with a .addMethod
    module.exports.addMethod = add;
```

Default export

```
let fname = ‘sebin’; let lname = ‘benjamin’;
module.exports = {
    First : fname,
     Last : lname
}
```

Importing - require()

- `require()` is a ***special function*** call defined as part of the CommonJS spec. It is NOT a part of the standard JavaScript API but used in Node.js.

  Eg: `app1FnExpression = require('app1.js');`
- `require()` function exposes features by **returning module.exports**, the reference to the module object. It works as our code is wrapped into the module object.
- Require caches modules and returns it wherever it is called, instead of importing the file again.
- Require can be used to import JSON files for reading some data, configurations stored in it etc.

- We could import a folder by adding a index.js file in it. Eg require(‘./folderName’);

    -- folderName

        -- index.js

- Can also be used to import core(native) modules/node api’s

```
const util = require(‘util’); //dont need ./
```

Some core modules

Node.js has several modules compiled into the binary. That are part of the platform and comes with node installation.

- events module
- fs module
- http module
- os module
- path module

Module Resolution

If the module identifier passed to require() is

1. NOT a core module, and
2. ***Does not begin with '/', '../', or './',***

Then Node.js starts at the **parent directory** of the current module, and **adds /node_modules**, and attempts to load the module from that location.

Module Resolution

For example, if the file at '/home/ry/projects/foo.js' called require('bar.js'), then Node.js would look in the following locations, in this order:

- `/home/ry/projects/node_modules/bar.js`
- `/home/ry/node_modules/bar.js`
- `/home/node_modules/bar.js`
- `/node_modules/bar.js`

This allows programs to localize their dependencies, so that they do not clash.

Questions ?

Node Package Manager (NPM)

- Code is shared through something called a package.
- A package contains all the **code being shared** + **a package.json file** (called a manifest) which describes the package.
- NPM ensuring your project's dependencies are under control.
- NPM comes already bundled with your Node.js installation
- Consists of three components:
  - **Website** - to manage various aspects of your npm experience (*https://www.npmjs.com/*)
  - **Command Line Interface** (CLI) - to interact with npm via the terminal
  - **Registry** - to access an extensive public database of JavaScript software.

- Package managers store dependency files into the ***node_modules*** folder
- The **package.json** file in the app root defines what libraries will be installed into node_modules when you run **`npm install`**.
    - Keeps all the relevant metadata associated with the project.
- npm provide an ***autogenerated lock file*** called **package-lock.json** that has the entries of the **exact versions** of the dependencies used in the project.
    - This file **locks the dependencies** to their stipulated versions during the installation process, after establishing the versioning parameters in the package.json file
- From **`node_modules`** the package code can be included into a project

NPM CLI

- `npm init` - Create a package.json file
- `npm install` - Install a package
- `npm outdated` - check the registry to see if any (or, specific) installed packages are currently outdated.
- `npm update` - Update an installed package
- `npm uninstall` - Uninstall a package
- `npm ls | list` - List installed packages
- `npm start` - Start a package
- `npm audit` - Run a security audit
- `npm audit fix` - Run a security audit and fix auto-fixable issues

Global Flag `-g`

Refer - NPM CLI Commands

To do

Do clone the repo and run Monday's example

week-7/day-1/sharp-demo

Thank you