Native Modules

Native modules refers to the modules that are written outside of JavaScript, modules that are written in C++ (C++ addons) for example and embedded into JavaScript using things like N-API (Node-API).

In node, most modules are written in javascript. Some modules, like the fs module are written in C/C++, as you can't edit files from plain javascript. IIRC, these modules are called 'native' because the code for these modules is slightly different depending on the OS node runs on.

We can Easily find Node JS Native Modules in it's Documentation.

https://nodejs.org/docs/latest-v20.x/api/

File System : To handle file operations like creating, reading, deleting, etc., Node.js provides an inbuilt module called FS (File System). Node.js gives the functionality of file I/O by providing wrappers around the standard POSIX functions. All file system operations can have synchronous and asynchronous forms depending upon user requirements. To use this File System module, use the require() method:

var fs = require("fs");

Common use for File System module:

- Read Files - https://nodejs.org/docs/latest-v20.x/api/fs.html#fsreadfilepath-options-callback
- Write Files –

https://nodejs.org/docs/latest-v20.x/api/fs.html#fswritefilefile-data-options-callback

- Append Files
- Close Files
- Delete Files

npm (Node Package Manager)

npm is a package manager for the JavaScript programming language maintained by npm, Inc. npm is the default package manager for the JavaScript runtime environment Node.js and is included as a recommended feature in the Node.js installer.

- **npm** is the world's largest **Software Registry**.
- The registry contains over 800,000 **code packages**.
- **Open-source** developers use **npm** to **share** software.
- Many organizations also use npm to manage private development.

Software Package Manager

The name **npm** (Node Package Manager) stems from when npm first was created as a package manager for Node.js.
All **npm** packages are defined in files called **package.json**.
The content of package.json must be written in **JSON**.
At least two fields must be present in the definition file: **name** and **version**.

npm can manage **dependencies**.
npm can (in one command line) install all the dependencies of a project.
Dependencies are also defined in **package.json**.

Getting Started With **npm**

npm comes pre bundled with node, so we don't need to install it separately

Commands

> npm init (Initialize npm)

This will create a package.json file with the following information shown below in images.

```
PS C:\Web Development Projects> cd 'c:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM'
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (1.4-npm) learning-npm
version: (1.0.0)
description: I'm learning npm
entry point: (index.js)
test command:
git repository:
keywords:
author: Naveen Sharma
license: (ISC)
About to write to C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM\package.json:

{
  "name": "learning-npm",
  "version": "1.0.0",
  "description": "I'm learning npm",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Naveen Sharma",
  "license": "ISC"
}


Is this OK? (yes) y
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM>
```

After this we can find a package.json file in our cd

Installing Packages:

```
>npm install <package>
```

We can easily find npm packages in https://www.npmjs.com/ website.

To learn about npm packages, we are taking example of npm “sillyname” package which generates silly names for you.
Link to “Sillyname” - https://www.npmjs.com/package/sillyname

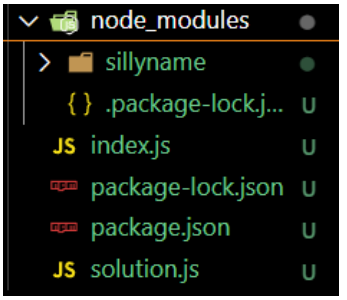
Copy the npm install command from website then run it on terminal of cd. After that see commands in images below.

```
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM> npm i sillyname

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM>
```

```
"dependencies": {
  "sillyname": "^0.1.0"
}
}
```



After installing “sillyname” package it’ll create Dependencies information in package.json file and also create a folder named with “npm_modules” which have all the package code written in it.

Using Packages: Go to the documentation of npm “sillyname” package and see the example here.

eg. First require the package by using following commands

var generateName = require(“sillyname”);
var sillyname = generateName();

Code & Output:

```
1  var generateName = require("sillyname");
2  var sillyname = generateName();
3  console.log("My name is " + sillyname + ".");
4
```

```
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM> node .\index.js
My name is Amberant Ape.
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM> node .\index.js
My name is Titaniumbear Singer.
PS C:\Web Development Projects\2. Back End Development\1. Node JS\1.4 NPM>
```

CJS

CJS is short for CommonJS. Here is what it looks like:

```
//importing
const doSomething = require('./doSomething.js');

//exporting
module.exports = function doSomething(n) {
  // do something
}
```

- Some of you may immediately recognize CJS syntax from node. That's because node uses CJS module format.
- CJS imports module synchronously.
- You can import from a library node_modules or local dir. Either by const myLocalModule = require('./some/local/file.js') or var React = require('react'); works.
- When CJS imports, it will give you a copy of the imported object.
- CJS will not work in the browser. It will have to be transpiled and bundled.

ESM

ESM stands for ES Modules. It is Javascript's proposal to implement a standard module system. I am sure many of you have seen this:

•Works in many modern browsers

•It has the best of both worlds: CJS-like simple syntax and AMD's async

•Tree-shakeable, due to ES6's static module structure

•ESM allows bundlers like Rollup to remove unnecessary code, allowing sites to ship less codes to get faster load.

```
import React from 'react';
```

```
import {foo, bar} from './myLib';

...

export default function() {
  // your Function
};
export const function1() {...};
export const function2() {...};
```

```
<script type="module">
  import {func1} from 'my-lib';

  func1();
</script>
```

Note:- To make sure ESM work in your index.js file make sure to do below changes in your “package.json” file.