ANECO ACADEMY

NODEJS -NPM

What is Nodejs

A Server-side Library: JavaScript on the Server-side

Listen to Requests and Send Responses

Execute Server-side Logic

Interact with Databases and Files

An Alternative to PHP, Ruby on Rails, Java etc. Is rarely used Standalone!

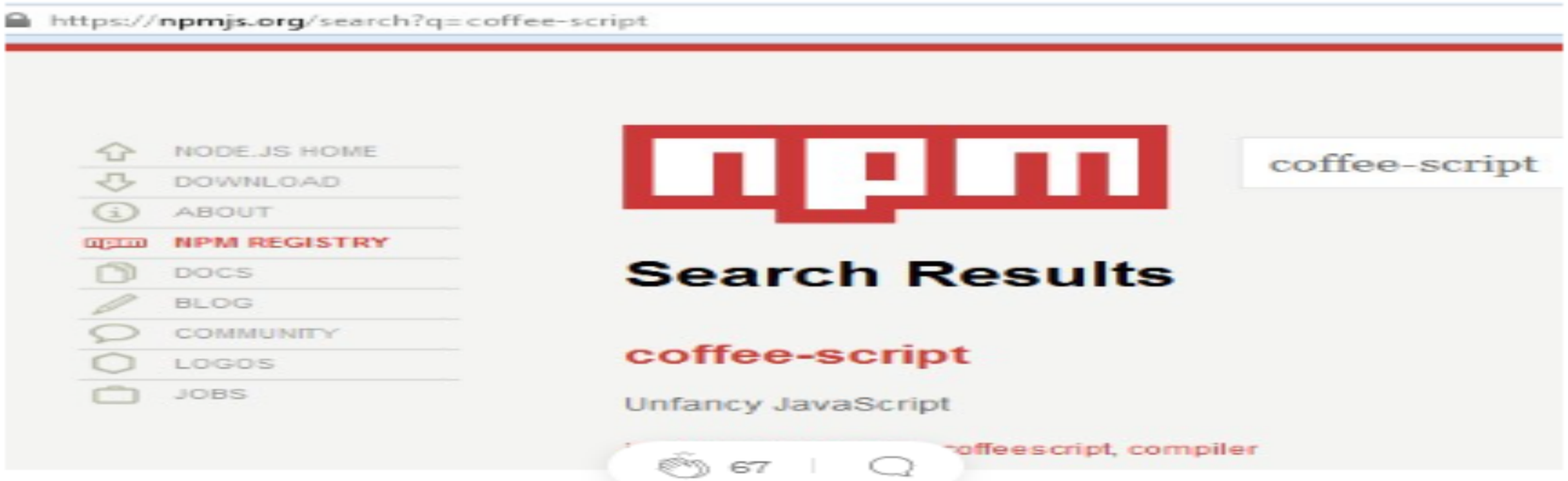
Javascript on Browser and Nodejs

What is NPM

What is NPM?

NPM stands for Node Package Manager and is one of the multiple package managers (others include Yarn, Bower, etc.) available for the Javascript programming language. Furthermore, it's the default package manager for Node.js.

NPM has a command-line client, which is generally how you install packages. What seems magical is how NPM knows what to download. All you do is give the name of the package you'd like, and NPM knows where to get it from. This is because NPM has an online database of public and paid-for private packages called the ***NPM registry***.



The NPM registry can be accessed online

What purpose Does NPM Server?

Like mentioned before, NPM helps with managing packages for our project. Since NPM automates managing packages, developers can focus more on developing and less on packages' maintenance.

So it acts as an assistant to the developer. How a doctor diagnoses a problem and tells his or her assistant to get a specific tool, you will be able to diagnose a bug, determine the best possible solution for it and have NPM retrieve the packages, otherwise known as the tools.

High-Level Overview of NPM

As a general rule, any project that is using Node.js will need a package.json file. The package.json file is basically the control panel for NPM. This is where you tell NPM which libraries you want to import, information about source control, project metadata, and more.

Whenever you run `npm install`, NPM will look to your package.json file and look at what libraries you want to import.

Before we go any further, though, we should understand the difference between a local and global installation. There is a slight distinction that plays a part in the package.json file.

Local Installation

A local installation downloads the package files into your project's folder. The package files are installed "locally" to your project.

You may sometimes notice a `node_modules` folder in your projects. This is the folder that holds those downloaded files. As you add more packages to your project, this folder will get heavier.

Whenever you are working with Git, it is recommended not to push the `node_modules` folder because of how large it can get. Rather the recommended action is to use the `--save` flag whenever installing a package, so it gets tracked in your `package.json` file. Instead of pushing the entire `node_modules` folder, you will push the `package.json` file that contains all the packages to download. Teammates can download the `package.json` file and locally install the `node_modules` folder themselves.

Basic Command

1. `npm init`

   This command will create a `package.json` file in your project. You can manually add packages to this file, or you can use the `npm install` command.

2. `npm install`

   This command will look into your `package.json` file and install all the packages that are listed in the `package.json` file.

3. `npm install <package-name>`

   Installs the latest version of the package to your `node_modules` folder.

What isPackage.json?

This file can contain a lot of meta-data about your project. But mostly it will be used for two things:

- Managing dependencies of your project
- Scripts, that helps in generating builds, running tests and other stuff in regards to your project

Creating a package.json file

Run the command `npm init` and it will ask you some information about your project and will create a basic `package.json` in that directory.

The initial content of that file would look something like this:

```
{
  "name": "npm-example",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

374 | 2

Inside the package.json

Identifier Metadata

These metadata describes the project: what it's called, what is it's purpose, version and other things.

A package.json file must have a `name` and a `version` property. These two properties uniquely identifies your project in npm repository.

name
The name of the project/module that this package.json file is describing

version
Denotes the current version of your module.

```
{
  "name": "npm-example",
  "version": "1.0.0",
}
```

package.1.json hosted with ❤ by GitHub view raw

Some other optional meta properties are `license`, `description` and `keywords`.

Inside the package.json

license
Describes the license of your project. Some common recognised licenses are: MIT, ISC, GPL-3.0

description
Contains a human readable description about the project.

keywords
A collection of keywords that helps identifying the project. These are similar to tags.

```
{
  "name": "npm-example",
  "version": "1.0.0",
  "license": "MIT",
  "description": "This project describes the package.json file in detail and explains what goes ins
  "keywords": ["npm", "package.json", "node.js"]
}
```

package.2.json hosted with ❤ by GitHub view raw

Functional Metadata
These properties help in coordinating with `npm` to perform certain tasks related to execution, build process and other stuff.

Inside the package.json

main
It describes the entry point to the application. When you run this application, or require it as a module in another application, the file described by this property will be included.

```
"main": "index.js",
```

repository
It defines where the source for this application/module is located. Mostly it will be github in case of open source projects. Specify the type of repository it is: `git` or `svn` or anything else, followed by the url to the repository.

```
"repository": {
  "type": "git",
  "url": "https://github.com/skyshader/example.git"
}
```

scripts
The scripts properties takes a list of key-value pairs. Each `key` specifies the name of the tasks the script will perform and the `value` will contain the command to invoke the task. The scripts can be invoked by `npm run <task-name>`. Scripts are mostly used for testing, building and defining some of the commands to work with your module/application.

Inside the package.json

```
"scripts": {
  "build": "node index.js",
  "test": "node index.test.js"
}
```

dependencies
A list of modules that this project depends upon are defined here.

```
"dependencies": {
  "react": "16.2.0",
  "react-dom": "16.2.0",
  "react-redux": "5.0.7",
  "axios": "0.17.1"
}
```

devDependencies
The modules that assist you in creating a build, testing your code, deploying your code and other development level modules.

```
"devDependencies": {
  "webpack": "4.1.1",
  "babel": "v7.0.0-beta.40",
  "jest": "22.4.2"
}
```

Final version of package.json

To check if any module in a project is 'old':

```
npm outdated
```

'outdated' will check every module defined in `package.json` and see if there is a newer version in the NPM registry.

For example, say `xml2js 0.2.6` (located in `node_modules` in the current project) is outdated because a newer version exists (0.2.7). You would see:

```
xml2js@0.2.7 node_modules/xml2js current=0.2.6
```

To update all dependencies, if you are confident this is desirable:

```
npm update
```

Or, to update a single dependency such as `xml2js`:

```
npm update xml2js
```

To update `package.json` version numbers, append the `--save` flag:

```
npm update --save
```

For updating a new and major version of the packages, you must install the npm-check-updates package globally.

```
npm install -g npm-check-updates
```

After installing the package run the following command:

```
ncu
```

It will display the new dependencies in the current directory whereas running this command will list all the global packages which have new releases.

```
ncu -g
```

Now run this command:

```
ncu -u
```

Now run this command:

```
ncu -u
```

After running this command it will result in the upgrading of all the version hints in the `package.json` file, so npm will install the major version by using this method.

Now everything is done. Just run the update command:

```
npm update
```

If you have a new project without any node_modules dependencies and you want to install the new version then run the following command:

```
npm install
```