**NODE JS – Introduction**

* **Synchronous vs Asynchronous**

**Synchronous JavaScript:** As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

Let us understand this with the help of an example.

**<script>**

**document.write("Hi"); // First**

**document.write("<br>");**

**document.write("Mayukh") ;// Second**

**document.write("<br>");**

**document.write("How are you"); // Third**

**</script>**

**Asynchronous JavaScript:** Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one. This may not look like a big problem but when you see it in a bigger picture you realize that it may lead to delaying the User Interface.

Let us see an example of how Asynchronous JavaScript runs.

**<script>**

**document.write("Hi");**

**document.write("<br>");**

**setTimeout(() => {**

**document.write("Let us see what happens");**

**}, 2000);**

**document.write("<br>");**

**document.write("End");**

**document.write("<br>");**

**</script>**

* **NVM (Node version manager, install)**

Using nvm (Node.js Version Manager) makes it easier to install and manage multiple versions of Node.js on a single local environment. Even if you only need a single version of Node.js right now, we still recommend using nvm because it allows you to switch between different versions of Node (depending on the requirements of your project) with minimal hassle.

To install or update nvm, you should run the [install script](https://github.com/nvm-sh/nvm/blob/v0.39.1/install.sh). To do that, you may either download and run the script manually, or use the following cURL or Wget command:

**curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash**

**wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash**

* **Install multiple versions of Node through NVM**

One of the most important parts of nvm is, of course, installing different versions of Node.js. For this, nvm provides the nvm install command. You can install specific versions by running this command followed by the version you want. For example:

nvm install 12.14.1

By running the above in a terminal, nvm will install Node.js version 12.14.1.

**NODE JS – Packages**

* **How to select a best suitable package**

This answer is based on a talk a few weeks ago in San Francisco by *Isaac Schlueter*

Isaac's main project now is to improve the NPM to help people figure out the quality of packages.

Before the [npmjs.org](https://npmjs.org/) website gets smarter, here are factors to consider Factors

1. Number of downloads
2. How recently updated
3. History of updates (has it been updated often over a long period of time)
4. Number of contributors
5. Have well-known/trusted developers and maintainers starred in it?
6. Do other important packages depend on it?
7. Is the package well-documented and has its own website?
8. Does the module have test coverage?

* **How to install a specific version of the package**

You can install specific version of an npm package using the @ syntax:

Install version 1.2.0 with:

npm install cowsay@1.2.0

**NODE JS – Package Managers**

* **NPM**

**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.

1. **Yarn**

Yarn, initially released by Facebook in 2016, is another popular package manager for the JavaScript programming language. The intention behind creating Yarn was to address some of the performance and security shortcomings of working with npm (at that time).

Since then, npm has undergone several improvements to fix some of its inefficiencies. As a result, as we’ll demonstrate in this blog post, npm and Yarn are now in a neck-to-neck race over which package manager trumps the other.

In an attempt to take Yarn a notch higher, the core team behind its development released Yarn 2 in January 2020. While Yarn 2 brings several improvements on the table, it has been heavily criticized among the developer community, and even Facebook engineers have publicly washed their hands from using it.

So, for the rest of this article, we’ll be discussing Yarn 1, and simply referring to it as Yarn, unless explicitly stated otherwise.

* **PNPM**

[PNPM](https://pnpm.io/) is an alternative package manager for Node.js which stands for “Performant NPM”.

The main purpose of PNPM is to hold all the packages at a global (centralized) store and use them if needed by other projects too by creating hard links to it.

**NODE JS – Package.json**

The package.json file is kind of a manifest for your project. It can do a lot of things, completely unrelated. It's a central repository of configuration for tools, for example. It's also where npm and yarn store the names and versions for all the installed packages.

package.json

{

"name": "test-project"

}

* **Dependencies**

When you install an npm package using npm install <package-name>, you are installing it as a dependency.

The package is automatically listed in the package.json file, under the dependencies list.

* **Dev-dependencies**

When you add the -D flag, or --save-dev, you are installing it as a development dependency, which adds it to the devDependencies list.

Development dependencies are intended as development-only packages that are unneeded in production. For example testing packages, webpack or Babel.

* **Peer-dependencies**

In the package.json file, there is an object called peerDependencies and it consists of all the packages that are exactly required in the project or to the person who is downloading and the version numbers should also be the same. That is the reason they were named as peerDependencies. The best example is ‘react’ which is common in every project to run similarly.

**Note:** These dependencies are not automatically installed. npm gives a warning message whenever there is a peer Dependency and these are different dependencies compared to the above-discussed dependencies.

* **Scripts**

Package.json has various sections, scripts is one of them, which allows you to write npm script which we can run using npm run <script-name>.

**"scripts"**: {

**"build"**: **"react-scripts build"**,

**"code:check"**: **"yarn code:lint && yarn code:format --check"**,

**"validate"**: **"run-p --print-label lint typecheck test build"**,

**"preversion"**: **"git pull"**,

**"postversion"**: **"git push && git push --tags"**

},

so to run the build we have to do

npm run build

* **NODE JS – package-lock**

package-lock.json is automatically generated for any operations where npm modifies either the node\_modules tree, or package.json. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

This file is intended to be committed into source repositories, and serves various purposes:

* Describe a single representation of a dependency tree such that teammates, deployments, and continuous integration are guaranteed to install exactly the same dependencies.
* Provide a facility for users to "time-travel" to previous states of node\_modules without having to commit the directory itself.
* Facilitate greater visibility of tree changes through readable source control diffs.
* Optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.
* As of npm v7, lockfiles include enough information to gain a complete picture of the package tree, reducing the need to read package.json files, and allowing for significant performance improvements.
* **NODE JS –** [**Building Package**](https://docs.npmjs.com/creating-node-js-modules)

All you need to build(and then publish) an NPM package is the NPM command line tool which also goes by the name npm. npm is distributed with [NodeJS](https://nodejs.org/en/), this means that when you install NodeJS, you automatically get npm installed on your computer.

## **The single and most important file**

After downloading npm, we can go ahead to start creating our package. Due to the simplistic nature of the package, most of the coding would be done in the command line interface(CLI). In the CLI do the following:

**# Create the project directory**

**mkdir random-number-generator**

**# Change into the project directory**

**cd random-number-generator**

**# Initialise an NPM package**

**npm init**

You will be required to answer some questions, these questions are basically about the NPM package you want to create. The command line tool auto-suggests reasonable answers to the questions, if you feel the suggested answer is good enough, just hit enter. If you also don’t have an answer to any of the questions, hit enter and continue, you can always edit it later.

After answering the questions, a package.json file will be created in the root of the project folder, the content will be similar to this:

**{**

**"name": "random-number-generator",**

**"version": "1.0.0",**

**"description": "",**

**"main": "index.js",**

**"scripts": {**

**"test": "echo \"Error: no test specified\" && exit 1"**

**},**

**"author": "",**

**"license": "ISC"**

**}**

**NODE JS – Syntax**

* **Explore different types of syntaxes, es6 specifically. Async awaits and promises included.**

const doSomethingAsync = () => {

return new Promise(resolve => {

setTimeout(() => resolve('I did something'), 3000);

});

};