1. About NPM
   1. World largest software registry were open-source developers share and borrow packages.
   2. The 3 components of the npm is the website, the CLI, and the registry.
   3. Could be installed locally (inside a particular project) or globally (available across the machine). For local use: [npm intall package\_name], and for global use [npm install -g package\_name].
2. Npm registry
   1. The public npm registry is a database of JS packages, each compromised of software and the metadata.
   2. Npm registry contains packages, many of which are node modules or contain Node modules.
   3. Package: Is a file or directory that is described by a package.json file. A package must contain this file to be published in the registry. Package.json file is used to keep track of all the package the particular project is using.
   4. Module: A file or directory inside the node\_module directory and that can be loaded by the require() function of the Node.js. All modules aren’t packages unless they contain package.json (plus having the “main” field inside that).
3. When we do npm init the CLI ask for some questions and then make a package.json file for the project which. And then we can put dependencies and required packages inside it.
4. Node\_modules stores all the packages for the project inside it, usually created when we run npm install command for a package. And, this package is also shown in the dependencies in the package.json file.
5. It is also possible the package that was installed by the npm install is dependent on some other package so those will be installed as well. And for some heavy packages it could get very complex and heavy. That’s why node\_module is said to be:

A picture containing diagram

Description automatically generated

1. Semantic Version System
2. Node modules are the JS libraries that have built in function that we can use in our projects, and built-in modules are the modules that we can use without any installation.
3. With modules.exports we have two different ways to export the functions:
   1. If there is only function to be exported we can simply do module.exports = function\_name
   2. By packing more than one function inside an object literal
   3. By using module.exports.function\_name = function\_name (RECOMMENDED)
4. Node.js in the backend do something like this var exports = module.exports as a convenience to provide exports keywork as an alias. Which we can use instead of the full module.exports. So, this means we can do exports.function\_name = function\_name. But we CANNOT do exports = function\_name like we did for module.exports = function\_name, this is because exports is literally an alias which further calls the modules.exports nothing more and if we do exports = function\_name all we are doing is changing the alias for exports to function\_name.
5. <https://www.freecodecamp.org/news/require-module-in-node-js-everything-about-module-require-ccccd3ad383/>