package.json

package.json is a plain **JSON**(JavaScript Object Notation) text file which **contains all metadata information about nodeJS project or application.** It's like quick summary to- what your project is all about and to what it is actually made up of.

Why package.json is so important?

Because it-

1. Is a great way to manage project dependencies, as it document your every dependency in one file.
2. Specifies the versions of dependency that the project use.
3. Makes your project easily reusable or shareable, with least installation overheads.

Hence, package.json is the most important file and the **great way to manage project dependencies**, for every developer who are developing their projects using nodeJS environment.

How to generate it?

There are two ways to create package.json:

METHOD I:

You can create **your own package.json file** using command -

> npm init

This command will initiate a process of creation of your package.json file along with few queries to be answered-

1. **package name -** The name is what your project is called
2. **version -** differ your updates to the same package
3. **Description -** helps people find your package on npm search
4. **entry point -** name of file, which you want to run when you hit your script.
5. **test command**- command or script that will be used while testing the project or package.
6. **git repository**- specify the place where your code is on git
7. **Keywords** - **array of string** which contains keywords that can be used **to identify your package in npm search tab**.

8. **Author**- default value as- ISC
9. **license**

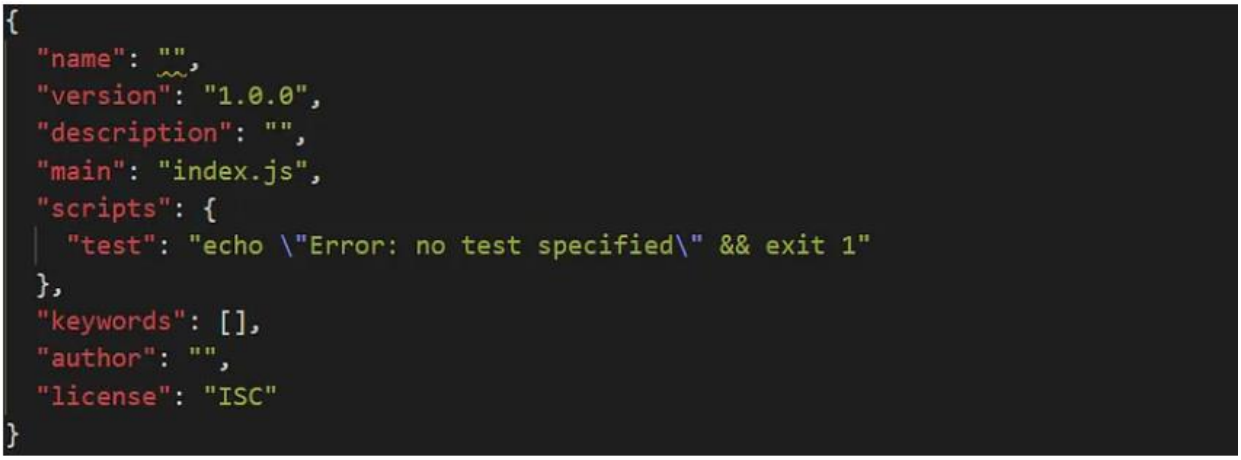
Package name + version = unique package

METHOD II:

Setting **package.json with default values**, by using the following command:

> npm init — — yes

This command will generate default package.json file with default entries, as below:



Scripts

The “scripts”are used for running test or starting any application.It is formed as a key-value pair where, key is the life-cycle event, and the value is the command to run at that point.

```
"scripts": {
      "start": "node index.js",
      "dev": "nodemon"
}
```

Dependencies –

It is specified as normal object containing key-value pairs where,Key- name of the package and Value — is version range with one or more space separated descriptors.We can also pass git URL, local URL i.e the path of package of your system or normal URL as a dependency for the package.

```
"dependencies": {
      "express": "^4.16.4",
    "compression": "~1.7.4"
}
```

Dependencies vs devDependencies

devDependencies contains packages which are **used during development or while creating build**. And are only useful in development phase.

```
"devDependencies": {
      "nodemon": "^1.18.11"
}
```

In general when we run,

npm install :- It **includes both dev and normal dependencies** on our system. But in case when we run -

npm install — — production or npm install “$package” :- It will **install only usual dependencies** for you, and don’t include dev dependencies.

npm install — — dev :- it will install development dependencies for you.

So, **use -dev** flag, **if you are installing packages which are useful only during development**, and don’t if package you are installing is for production.

How to add dependencies?

Using npm as package manager: Use this command to add dependencies in your project. you have to install the dependencies before using it in you project

```
npm install package-name –save
     OR
npm i package-name
```

Using yarn as package manager: When using Yarn as your package manager, you can add a dependency with the following command, replacing “package-name” with the actual package name:

```
yarn add package-name
```

How to remove dependencies?

Using npm as package manager: Execute the following command so that you can remove that dependencies from your project.

```
npm uninstall package-name
```

Using yarn as package manager: When using Yarn as your package manager, you can remove a dependency with the following command, replacing “package-name” with the actual package name:

```
yarn remove package-name
```

How to add a script?

Step 1: Open Your package.json File.

Step 2: Insert a key-value pair into the “scripts” object. The key serves as the identifier for your script, while the associated value is the command to be executed upon running the script.

```
"scripts": {
  "start": "node server.js",
  "test": "mocha test/*.js"
}
```

Step 3: Save the package.json File amd run your script:

```
npm run start
```

or

```
npm run test
```