Package Management with NPM

This lesson teaches package management in Node.js with NPM.

WE'LL COVER THE FOLLOWING

Soon after the creation of Node.js, it became apparent that something was missing to orchestrate code sharing and reuse through modules. So npm (Node Package Manager) was born in 2010. It is still the standard package manager for the Node ecosystem, even if it is being challenged by yarn, a more recent alternative. It consists of a command line client, also called npm, and an online database of public packages, called the *npm registry* and accessed by the client.

Over 477,000 packages are now available on the registry, ready to reuse and covering various needs. This makes npm the largest ecosystem of open source libraries in the world. The npm client is used by typing commands in a terminal open in the package's folder. It offers numerous possibilities for managing packages. Let's study two of the most important ones.

Installing Dependencies

To install all the dependencies of a package, you type the following `npm` command.

> npm install

This will read the `package.json` file, look for the packages satisfying the version ranges declared in the `dependencies` field, and download and install them (and their own dependencies) in the `node_modules/ subfolder`.

Adding a New Dependency

There are two ways for adding a new dependency to a package. The first one is to manually edit the `package.json` to declare the dependency and its associated version range. The next step is to run the following npm command.

> ```
> npm update
> ```

This will update all the packages listed to the latest version respecting their version range, and install missing packages.

The other way is to run the following command.

> ```
> npm install <package-id>
> ```

This command will fetch a specific package from the registry, download it in the `node/modules/` subfolder and (since npm 5) update the `package.json` file to add it as a new dependency. The `<package-id>` parameter is usually the dependency's package name.

Using a Dependency

Once external packages have been installed in `node_modules/`, the application

can load them as modules with the `require()` function.

For example, the npm registry has a `semver` package that handles semantic versioning. Assuming this package has been installed as a dependency, it can be used to perform manual version range checks.

```
// Load the npm semver package as a module
// Notice the omission of "./" since the package was installed in node_modules/
const semver = require("semver");

// Check if specific versions satisfy a range
console.log(semver.satisfies("2.19.0", "^2.18.1")); // true
console.log(semver.satisfies("3.0.0", "^2.18.5")); // false
```