COESIONE ITALIA 21-27 PIEMONTE

Cofinanziato dall'Unione europea

REGIONE PIEMONTE

# Appendice

## Programmazione frontend

**Docente:** Shadi Lahham

*Intervento realizzato da*
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

# NPM

Using NPM and installing packages

## Node modules & NPM

Modules are a concept in Node that allow adding functionality written by other developers to applications. They also allow code refactoring for readability and usability.

## NPM

Npm is the default package manager for node. Node has one of the biggest single language code repositories of user-contributed packages.

You can find packages on npmjs.com

### Quick example

```
// create a directory and enter it
mkdir quick-test
cd quick-test

// quick initialize a project and create the package.json file
npm init -y

// install a package
npm install cowsay
```

```
// run the package
npx cowsay hello
```

```
Also try using npm init without the -y flag
npm init

Take a look at the package.json file to understand what init does
```

## Package.json

The **package.json** file holds metadata about the project. This information provides the Node package manager the necessary information to understand how the project should be handled along with its dependencies.

The package.json files contain information such as the project description, the version of the project in a particular distribution, license information, and configuration data. It is normally located at the root directory of a Node.js project.

The package.json serves as:

- A central repository of configuration for tools
- The place where npm stores names and versions of installed packages

```
{
  "name": "grader",
  "version": "1.2.9",
  "description": "An application to manage student grades",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "dependencies": {
    "@angular/common": "~12.2.0",
    "@angular/core": "~12.2.0",
    "@angular/forms": "~12.2.0"
  },
  "devDependencies": {
    "@angular/cli": "~12.2.6",
    "typescript": "~4.3.5"
  },
  "engines": {
    "node": ">= 15.0.0",
    "npm": ">= 7.0.0"
  }
}
```

```
// version: the current (semantic) version
// name: name of the app/package
// description: a brief description
// scripts: node scripts that can be run
// dependencies: list of npm packages installed as dependencies
// devDependencies: list of npm packages installed as dev dependencies
// engines: versions of Node and npm that this app/package works on
```

Note that another file, package-lock.json is automatically generated. You can read about it in the link below.

Reference:
Package.json documentation
Package-lock.json

## Managing packages with NPM

### Install packages

```
// install a package
npm install <package-name>

// alternative
npm i <package-name>

// install a package as a development dependency
npm i --save-dev <package-name>

// alternative
npm i -D <package-name>

// install a package globally
npm i -g <package-name>
```

### Development dependency

--save-dev or -D flag

These are packages that you want to install as development tools for your project but will not be a part of your code

These packages will be added to package.json as **"devDependencies"**

Example: **npm i -D typescript**

### Global packages

-g flag

These are tools that you want to install on your entire system, not specifically for a single project. They are usually command line tools

You will not find these in your project's package.json as they are installed globally. **Think twice** before you install something globally

```
// this installs angular cli
Example: npm i -g @angular/cli
```

#### Deal with package versions

```
// find the latest version of a package
npm show <package-name> version

// list all versions of a package
npm show <package-name> versions

// install a specific version of a package
npm i <package-name>@version
// example
npm i typescript@4.4.3
```

#### See what's installed

```
// list installed packages
npm list

// also useful: see the contents of package.json
cat package.json
// look inside node_modules to see all packages and their dependencies
ls node_modules
```

```
// node_modules is a special directory used by NPM to install modules
// remember to add node_modules to .gitignore
```

#### Uninstall packages

```
// uninstall a package
npm uninstall <package-name>

// uninstall a global package
npm uninstall -g <package-name>
```

List of all npm flags

## Semantic versioning

All Node packages use semantic versioning for version number.

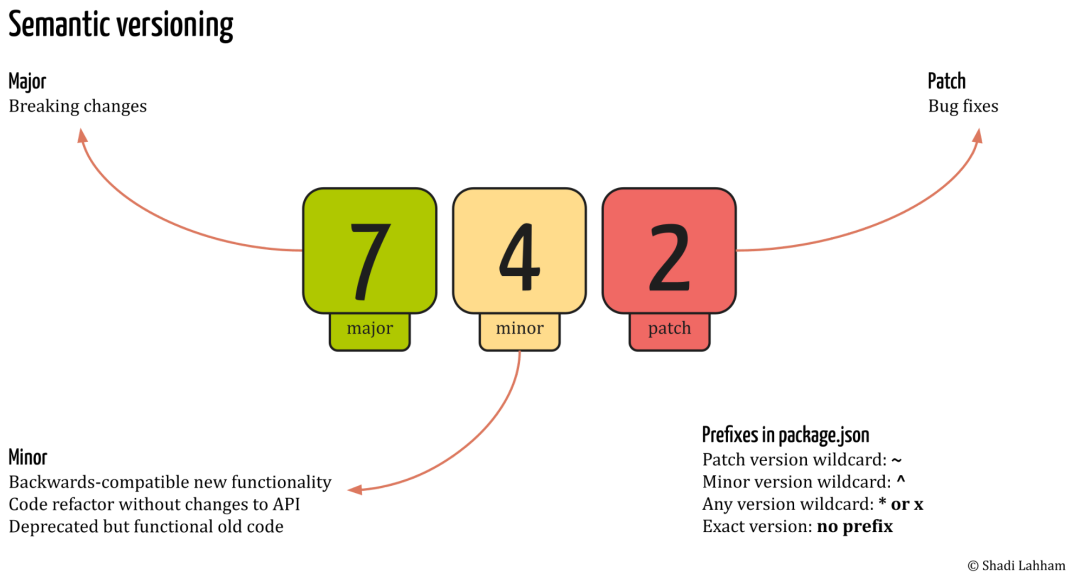
In semantic versioning all versions have 3 digits: x.y.z

- major version: first digit
- minor version: second digit
- patch version: third digit

When making a new release increment:

- major version: when there are incompatible API changes
- minor version: when adding backward-compatible functionality
- patch version: when making backward-compatible bug fixes

These version numbers are used in package.json to indicate how Node updates our packages when we run the command **"npm update"** and there are new versions.




References:
[About semantic versioning]
[npm semantic version calculator]

## NPM scripts & NPX

### NPX

```
Allows running a package even when it's not previously installed or as
a tool even if it's not installed globally

This can be useful in some situations

// run nodemon even if it's not installed globally (runs js in node)
npx nodemon main.js
```

### NPM scripts

```
// define scripts inside package.json


"scripts": {
  "watch": "webpack --watch",
  "start": "webpack-dev-server --open",
  "build": "webpack"
}
```

### NPM scripts

```
// run scripts in terminal
// start and test are special, they don't require 'run'

// start is normally your main script, a.k.a. entry point
npm start
npm test

// use npm run to run other scripts
npm run watch
npm run debug

// you can define your own
npm run special
```

## Evaluating packages

Packages can be found everywhere, especially on npmjs.com and github.com.

### How to evaluate packages

There are many factors when evaluating packages, but the most important is to get an idea if a package is stale or in active development and if you can use it for commercial purposes.

Finding a package is easy, finding a good, safe, stable package is not always simple.

### On npmjs

Look at the **Weekly Downloads**, **License**, **Last publish date** and the number of **Collaborators.**

### On github

Look at the date of **the last commit**, the **number of commits**, read through the **open issues**, and look at the **number of stars.**

### Readme

In both cases, make sure you read the readme on the main page in detail.

## Interesting modules

1. nodemon
2. rimraf
3. http-server
4. static-server
5. nanoid