More NPM

Presented by
Tom Isles

Canva

```
rt_tree = require('../util/convert-tree
til = require('../util/time-util.js');
nst = require('../util/app-const.js');

exports.getRender = f(req, res){
  if(req.query.rootId){
    res.render('tree', {
      title: req.i18n.__("gnr.title"),
      rootId: req.query.rootId,
      currentLocale: req.i18n.getLocale()
    });
  } else {
    res.render('tree', {
      title: req.i18n.__("gnr.title"),
      rootId: null,
      currentLocale: req.i18n.getLocale()
    });
  }
};
```

Recap

- NPM is a package management system for javascript applications. It allows you to install dependencies / libraries from the web.
- You can install new dependencies using *npm install*, start applications using *npm start*, and so on.
- *package.json* is a file that contains metadata about your application and exists in the root of your directory.
- Yarn is an alternative to NPM that was released by facebook and has a few advantages.

package.json

the package.json construct is tied to NPM, or more specifically, to **npmjs.org**, which is where javascript packages are hosted.

Many fields are related to the publication of packages.

Other fields relate to dependency management, browser support, and custom scripts.

For the purposes of this lecture we will talk about NPM, but these can be used interchangeably.

```
{
  "name": "test-app",
  "author": "Tom Isles <my_email@example.com> (https://example.com)",
  "contributors": [
    "Other Contributor <other@example.com> (https://other.com)"
  ],
  "bugs": "https://github.com/repository/package/issues",
  "homepage": "example.com/test-app",
  "version": "0.1.0",
  "license": "MIT",
  "keywords": [
    "demo",
    "unsw"
  ],
  "description": "A package to demonstrate proper package.json",
  "repository": "github:example/example",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^4.2.4",
    "@testing-library/react": "^9.3.2",
    "@testing-library/user-event": "^7.1.2",
    "react": "^16.13.1",
    "react-dom": "^16.13.1",
    "react-scripts": "3.4.3"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

package.json

Many fields in package.json are not related to your application per se, but define metadata related to it.

If you publish a package on *npmjs.org,* fields like *name, author, contributors, bugs, homepage, license, keywords, description* and *repository* are all used to generate your package's page.

Example Package:

https://www.npmjs.com/package/react

License is a special case here. While it is displayed on the NPM site it also determines how your code is licensed and has legal ramifications depending on the license.

```
{
  "name": "test-app",
  "author": "Tom Isles <my_email@example.com>
(https://example.com)",
  "contributors": [
    "Other Contributor <other@example.com> (https://other.com)"
  ],
  "bugs": "https://github.com/repository/package/issues",
  "homepage": "example.com/test-app",
  "license": "MIT",
  "keywords": [
    "demo",
    "unsw"
  ],
  "description": "A package to demonstrate proper
package.json",
  "repository": "github:example/example",
```

package.json

***version* is a specification of your package's current version. If you make a change to a hosted package you should also change the version.**

Versions are used by other developers who wish to depend on your package.

Versions use semantic versioning*:

MAJOR.MINOR.PATCH, where:

A change in MAJOR version means you changed the API in a breaking way.

A change in MINOR version means you added functionality in a backwards compatible manner.

A change in PATCH version means you made backwards compatible bug fixes.

**semver.org*

```
"version": "0.1.0",
```

package.json

***private* prevents accidental publication of packages.** If you have a package that you never wish to upload to NPM, then private should be set to true.

```
"private": true,
```

package.json

Dependencies list the packages that your package relies on.

When specifying a dependency, we also specify a corresponding *version*. Like in our version field, we use semantic versioning to specify which version of the package we require.

Dependency versions do not have to be straight versions, they can include ***semver ranges***.

Some examples:

>1.2.3 Any package with a more recent version than 1.2.3, including major releases.

^1.2.3 Any version of the package with a minor version greater than 2 and a patch version greater than 3. Cannot exceed major version **1**. Get most updated major version of package.

~ 1.2.3 Any version with a patch version greater than 3. Cannot exceed major version **1** or minor version **2.** Get most updated minor version of package.

https://nodesource.com/blog/semver-a-primer/

```
"dependencies": {
  "@testing-library/jest-dom": "^4.2.4",
  "@testing-library/react": "^9.3.2",
  "@testing-library/user-event": "^7.1.2",
  "react": "^16.13.1",
  "react-dom": "^16.13.1",
  "react-scripts": "3.4.3"
},
```

package.json

package.json has numerous dependency types:

- dependencies
- devDependencies
- peerDependencies
- optionalDependencies

```
"dependencies": {
  "@testing-library/jest-dom": "^4.2.4",
  "@testing-library/react": "^9.3.2",
  "@testing-library/user-event": "^7.1.2",
  "react": "^16.13.1",
  "react-dom": "^16.13.1",
  "react-scripts": "3.4.3"
},
```

Dependency Types

dependencies
Modules required by the application during runtime.
Common examples: *react*

devDependencies
Modules required during development of the application.
Common examples: *babel, eslint, jest (testing library)*

Dependency Types

peerDependencies
If a package has a peer dependency, any application that consumes it must also install the peer dependency. The peer dependency won't be installed transitively.

An example of this is *react-dom*. r*eact* lists it as a peer dependency and so react applications must depend both on *react* and *react-dom.*

optionalDependencies
Any dependency where, if it fails to install, npm / yarn will say installation was still successful.

package.json

scripts allows you to specify command line scripts to run with npm.

You have already seen commands like *npm install*. NPM and Yarn both have a set of reserved keywords which link to inbuilt commands - like *install* - but any other keyword is fair game.

The scripts object is a simple key value object. The key is your command, which you run with npm, ie *npm <key>*. The value is the command that will be run when you call out to it. The command should be written in a syntax parseable by your default shell.

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test",
  "eject": "react-scripts eject"
},
```

Demo

package.json

package.json is a JSON file, so it can be arbitrarily extended.

In this case, an *eslintConfig* field has been added. It's not part of a standard package.json specification, but is used by eslint to configure linting for your application.

```
"eslintConfig": {
  "extends": "react-app"
},
```

package.json

browserslist is another example of an option that is referenced by certain tools and is specified by the browserslist plugin (npmjs.com/package/browserslist).

It specifies what browser types are supported by the application.

```
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

Yarn vs NPM

What's the difference between Yarn and NPM?

Yarn was created at Facebook to solve their problems with NPM at scale. At time of release, Yarn was:

- Significantly faster.
- Used a deterministic install algorithm. At the time, NPM could install dependencies in a different order leading to different *node_modules* folder structures given the same *package.json* between developers.
- Used lockfiles.

Performance

Performance Improvements

In Yarn, packages that are not dependent on each other are installed in separate threads, meaning install times are cut down compared to NPM.

NPM version 5 achieved significant performance improvements which have closed the performance gap between the two tools

Lockfiles

When specifying dependencies in *package.json,* version ranges can be specified using the semver syntax (^, >, *, etc).

This means that different developers developing the same application can end up using different versions of their packages. This leads to 'works on my computer' problems.

A **lockfile** records the exact versions of each dependency that has been installed.

Lockfiles

We commit a **lockfile** into version control and this ensures that the next time the repository is checked out and installed, the same versions are installed.

In yarn, **yarn.lock** is the lockfile and is generated each time dependencies are installed.

In NPM, the **npm shrinkwrap** command creates an **npm-shrinkwrap.json** file which performs the same function.

Good Work!