COMP1531

🌏 Projects - Package Management

Lecture 2.1

Author(s): Hayden Smith, Dr. Yuchao Jiang

In This Lecture

- **Why?** 🤔
  - To utilise javascript fully, we need to understand how to install other modules that aren't on our system
  - We need to know how to manage our installations on multi-user projects
- **What?** 
  - Problems with packages
  - NPM and how it works
  - How to manage packages
  - Custom scripts

 Disclaimer: Environment Change

Beginning from lecture 2.1, to run this code you may require some other files in your repo - simply copy the lecture content or use a lab in a following week to run any code you see.

😳 Problems That We Face

Sometimes we might want to use a library in NodeJS, but this library wasn't built-in by us. Someone else on the internet wrote it.

An example of this might be you googling "javascript how do I check if a date is valid" and coming across some code. And we find this snippet that looks good... so we try it out!

```
1 import { isValid } from 'date-fns';
2
3 console.log(isValid(new Date('2021, 02, 30')));
```

😳 Problems That We Face

So now we turn it into a function that looks good for our purposes.

```
1 import { isValid } from 'date-fns';
2
3 function dateIsValid(year, month, day) {
4   return isValid(new Date(year, month, day));
5 }
6
7 console.log(dateIsValid('2022', '14', '02'));
```

😳 Problems That We Face

So now we turn it into a function that looks good for our purposes.

```
1 import { isValid } from 'date-fns';
2
3 function dateIsValid(year, month, day) {
4   return isValid(new Date(year, month, day));
5 }
6
7 console.log(dateIsValid('2022', '14', '02'));
```

But when we run it we get this error...

😳 Problems That We Face

So now we turn it into a function that looks good for our purposes.

```
1 import { isValid } from 'date-fns';
2
3 function dateIsValid(year, month, day) {
4   return isValid(new Date(year, month, day));
5 }
6
7 console.log(dateIsValid('2022', '14', '02'));
```

But when we run it we get this error...

```
1 SyntaxError: Cannot use import statement outside a module'
```

😳 Problems That We Face

So now we turn it into a function that looks good for our purposes.

```
1 import { isValid } from 'date-fns';
2
3 function dateIsValid(year, month, day) {
4   return isValid(new Date(year, month, day));
5 }
6
7 console.log(dateIsValid('2022', '14', '02'));
```

But when we run it we get this error...

```
1 SyntaxError: Cannot use import statement outside a module'
```

We need **tools** to solve this...

🧩 NPM: Node Package Manager

NPM (Node Package Manager) is a tool that is automatically installed alongside NodeJS to manage dependencies/modules/libraries (all the same thing) for NodeJS (Javascript) projects.

It's command on terminal is:

`npm`

🧩 NPM: Node Package Manager

The most common usage of NPM is to allow you to download external libraries.

The external libraries that you're able to download with NPM are found on the npmjs website.

Let's have a look for our `date-fns` library!

NPM: Node Package Manager

To setup a code repository to use `npm`, all we need to do is run `npm init` (if it hasn't already been run). It will ask you a few questions (don't stress about getting them right, you can change them later).

Once this is done you should now see a `package.json` in your repository. This is essentailly your projects NPM configuration file.

```
1  {
2    "name": "example",
3    "version": "1.0.0",
4    "description": "",
5    "type": "module",
6    "main": "index.js",
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "author": "",
11   "license": "ISC"
12 }
```

package_simple.json

Sometimes we need to configure further. For example for 1531 reasons we added `"type": "module"` too.

🎬 Managing Packages

We can install dependencies with

`npm install [dependency]`

For example:

`npm install date-fns`

You will see that this command automatically adds the most recent stable version of `date-fns` to our `package.json`.

Let's inspect `package.json`.

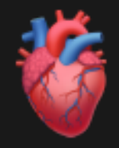
Take note of the `~` and `^` symbols.

🎬 Managing Packages

Our code will now run successfully.

```
1 import { isValid } from 'date-fns';
2
3 function dateIsValid(year, month, day) {
4   return isValid(new Date(year, month, day));
5 }
6
7 console.log(dateIsValid('2022', '14', '02'));
```

date_fns.js

 Anatomy Of NPM

The structure of NPM involves a few key files and folders.

package.json

Where we store meta data about our project including a list of dependencies to install

package-lock.json

Where we store versioning information about dependencies to ensure everyone has the right versions (oversimplification)

node_modules/

Where the dependencies are installed locally.

 Anatomy Of NPM

The structure of NPM involves a few key files and folders.

package.json

Changes are always commited to git.

package-lock.json

Changes are always commited to git.

node_modules/

Changes are never commited to git.

 Anatomy Of NPM

By committing this information we ensure that our environment is completely reproducable on others' machines + our own.

If you fresh clone the repo, the `node_modules` folder won't exist which means the dependencies won't work.

However, if you run `npm install` it will read the `package.json` and `package-lock.json` file to install the appropriate dependencies.

Custom Scripts

A useful feature of `npm` that we will explore in another lecture is the ability to add **scripts** to the `package.json`.

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
};
```

Custom Scripts

We can add our own command and run it with:

`npm run hayden.`

```
1 "scripts": {
2   "test": "echo \"Error: no test specified\" && exit 1",
3   "hayden": "echo 'Hi Hayden!'"
4 };
```

Scripts use a mixture of `bash` and `json`. We will cover these more later in the course.

For now we'll tell you exactly how to modify this stuff.

👂 Feedback

Or go to the form here.