Moduling in React

React depends on NodeJS module system. NodeJS has mainly two ways to create and consume modules:

1. Common JS modules
2. ES6 modules

CommonJS modules are the older way to create and consume modules and they depend on the `require` function which is a global function given by nodejs.

ES6 module , is the newer way to handle modules, and provide import and export keywords to create and consume modules.

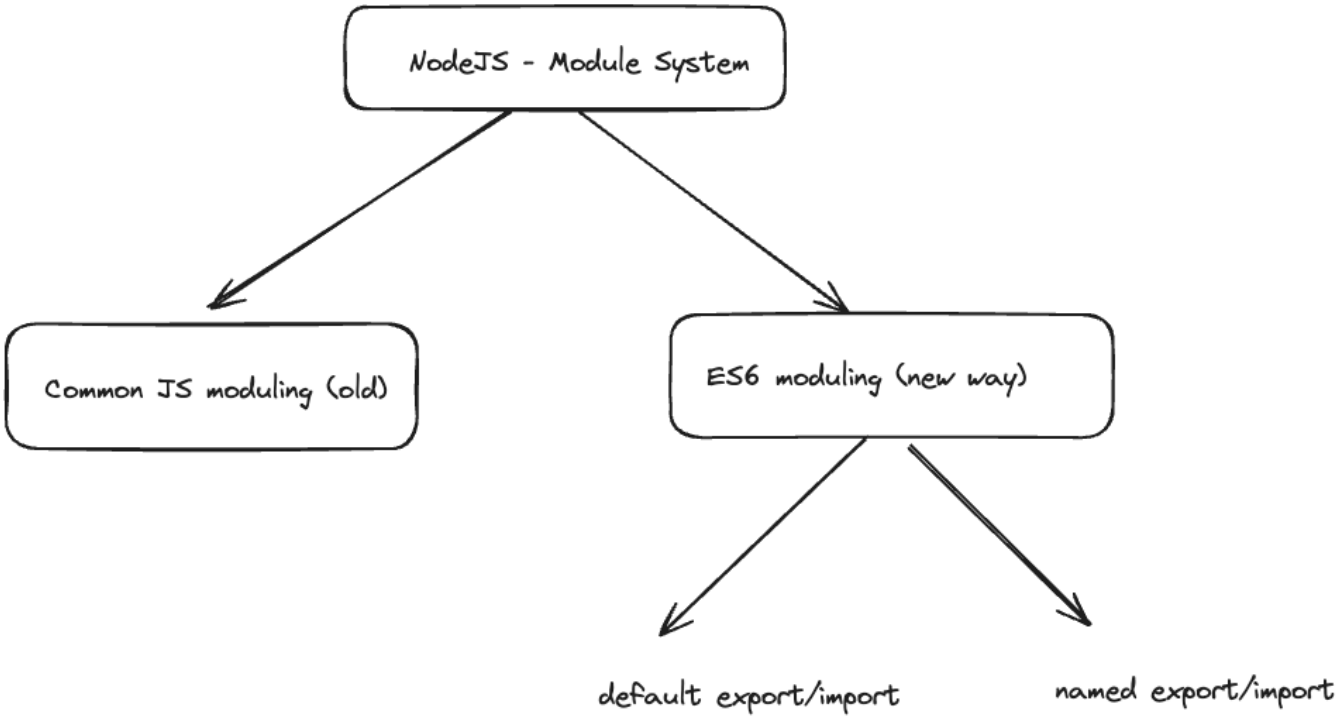
Modern react is mainly dependent on Es6 modules only.

ES6 modules

ES6 modules are a modern import export keyword bases module mechanism.

There are mainly two ways to do ES6 modules:

1. Default import / export
2. Named import / export



import keyword helps us to import other modules in our files.
export keyword helps us to expose our functionalities to other modules.

Default import / export

In default import and export we use syntax like `export default something;` . Saying the default keyword makes an export default. But what does it mean ?

We can only have one default export per file, this makes the default exported variable / function / class as a primary export of that file.
While import a default exported entity, we can use any name to import it not necessarily the original name altogether.

```
function fun() {
        return 10;
}

export default fun; // this is a default export
```

```
import funFunction from './fun'; // this is a default import

// some implementation
```

Note

While importing we use the from keyword to mention the path of the module from where we want to import. This path can be of two types:

1. Relative
2. Absolute

Named export/import

In named export, we don't use the default keyword. Instead just directly say `export something`.
We can do named export of more than one things, and put named export at multiple places of the file.

While importing, as we have multiple exports, we need to use object destructuring. We get the named export inside an object where the key of the object is having the same name as of the entity exported.

We use an object destructuring mechanism to put all these key value pairs inside their individual variables. We can also use the `as` keyword to give a nick name to the variables unpacked.

```
// file.js
export const x = 10;
export const fun = () => {

}
```

```
import {x, fun as somefunction} from '../file.js'
```

Note

- if we see import path as './something' we are looking in the same directory
- if we see import path as '../something' we are looking one folder out.
- If we see import path as '../../something' we are looking 2 folder out.

- and so on ....