

Nodejs

- Nodejs is easily employed as a server-side proxy where it can handle a large amount of connection in a non-blocking manner

In Short

- It is JS which execute on the server.

Installing Nodejs and VS Code

- You can install nodejs from Nodejs.org
 - You can download VS code from code.visualstudio.com.
- ⇒ We can run nodejs by using repl on VS code.

Remember: Some JS command such as `document.getElementById` by id... and `localStorage` won't run in Nodejs.

1) Running a Program

- Create `index.js`
- Write some JS code
- Click on ▶ to run or write `node index.js` in terminal

Initializing a Node project

In terminal, write `npm init`,
Fill up the detail and write Yes.
⇒ `package.json` will be created.

Package.json

- It records important metadata about a project which is required before publishing to NPM and also defines functional attributes of a project that npm uses to install dependencies, run JS etc.

What is npm?

- ⇒ Full form: Node Package manager
- It puts modules in place so that node can find them and ~~node~~ is used to publish, install and develop Node program.
- ⇒ Run `npm help` to get list of

available command.

Eg. To check npm version use:

`npm --version`

Installing Package

i) Express.js

To install it write:

`npm install express --save`

⇒ You will see that "express" version written in your package.json after installing it.

⇒ A Node module folder also gets created. ↳ (size is too large)

⇒ And package-lock.json is created.

(package-lock.json is a large list of each dependency listed in your package.json, the specific version and location of module etc)

To uninstall package write:
`npm uninstall express`

ii) Nodemon package install
To install: `npm i -g nodemon`

⇒ It automatically display the new result if input values are changed.

Eg. `console.log("Hello")`
↳ Print Hello in terminal

On changing
`console.log("Hello World");`
↳ Automatically displays Hello world

Dev dependency
Packages that are used during development.

To install package as dev dependency
`npm install --save-dev nodemon`
Name ↴

iii) Angular cli package
To install: `npm i @angular/cli`

(Check out more packages by Yourself.)

Page No. _____
Date / /

⇒ You can take help from node.js documentation

Import and Export by CJS
Create two files i) Index.js
ii) Second.js

⇒ Firstly we will export code from second.js and then import it in Index.js

In Second.js

```
Harry = {  
  name: "Pika",  
  fav num: 36,  
  developer: true}
```

```
module.export = Harry;
```

↳ Name of object

In Index.js

↳ Any name

↳ location

```
const lovish = require("./second");
```

```
console.log("Hello world", lovish)
```

⇒ Both object get printed.

Module Wrappen function

- ⇒ Give some basic info
- ⇒ Actually it wraps the entire code inside a function before execution. This function is termed as module wrapper function.

```
(function(exports, require, module,  
  - filename, - dirname){
```

```
  console.log(exports, require, module,  
    - filename, - dirname) } );
```

- ⇒ Value of each variable get printed.

Modules

i) OS

It is a built in module.

To import:

- ⇒ Create a file index2.js
- ⇒ Write `const OS = require('OS');`

Example:

⇒ `console.log(os.freemem())`

↳ Gives info about available free memory

There are many function, Check out by yourself:

`os.platform()`

`os.homedir()`

`os.hostname()`

`os.release()`

and much more:

Path module

Create: `pathmodule.js`

Import: `const path = require('path');`

Some example:

`const a = path.basename('C:\\temp\\myfile.html');`

`console.log(a)`

⇒ `myfile.html` get printed.

- (You can check out more function from documentation of Node) like:

- `path.dirname(path)`
- `path.extname(path)`
- `path.relative(from, to)`
- and much more...

FS module

To import: `const fs = require('fs');`

Example

`fs.readFile('file.txt', 'utf8', (err, data))`

\Rightarrow { `console.log(err, data)` }
 You have to create this file.

});

Call back function is used

• To get \Rightarrow instead of \Rightarrow use signature font

\Rightarrow Generally, the output will be null this is a file
 \hookrightarrow For err \hookrightarrow For data

* Asynchronous non blocking io model.

Blocking is when the execution of additional Javascript in the

Node.js process must wait until a non-JS operation completes. But this doesn't happen in Node.js until it is done intentionally.

Example:

```
fs.readFile('File.txt', 'utf8', (err, data) => { console.log(err, data) })
```

```
console.log("Finished reading file")
```

Output: Finished reading file
Null This is a file

→ The second statement gets printed first as first one is time consuming and node.js allowed the second to run.

To stop non-blocking use:

→ fs.readFile Sync

Check out more :

- fs.openSync(path[, flags, mode])
 - fs.rename(oldpath, newpath, callback)
 - fs.unlinkSync(path)
- and more.

- * Tempcode number.js
Automatically gets created, when a portion of code is run.

- * Common JS module. (Import/Export)
Create two file
i) module first.js
ii) module second.js

In module second.js
function name() {
 console.log("Simple is Complex")
}
module.exports = name;

In module first.js
const name = require("./module2")
name()

On being run, it prints Simple is Complex.

ES6 module

Create two files i) modulesecond.mjs
ii) modulefirst.js

⇒ .mjs is used for ES6 module.

In ECMAScript module, require and module.export can't be used.

Import:

i) In package.json
add "type": "module",
to use ES6.

In second.mjs

```
Export function name() {  
  console.log("Hello world");  
}
```

In first.js

```
Import {name} from './modulesecond.mjs'
```

```
name()
```

In terminal, 'Hello world' will be printed.

⇒ In this way, you can import/export function among the files

⇒ More than one function can also be imported/exported.

Eg. In second.mjs

- i) `Export function name() {
 console.log("Simple is complex")`
- ii) `Export function Name ↗ simple2 {
 console.log("Hello")`

In modulefirst.js

```
Import { name, simple2 } from  
  './module second.mjs'
```

Default function

It allows formal parameters to be initialized with default values if no value or undefined is passed.

Eg. `Export default function name() {
 console.log("Hello World")`

Now, if you try to import any function which isn't define then above value i.e Hello world get imported.

⇒ Import * as a2 from './module
second.mjs'

console.log(a2)

* is used to
import/export
all values.

URL module.

To import: Import url from 'url';

You can get many function of URL from documentation which you can use to set a portion of URL.

Eg. i) url.hostname: Gets and sets the host name portion of the URL

ii) url.host: It gets and sets host portion of URL

* difference between the two is that url.hostname doesn't include port.

iii) url.password: Gets and sets password

iv) `url.hash` → Gets and sets fragment portion of the URL.

• There are many more, checkout by yourself.

Eg.

```
const myurl = newurl('https://ex.org');  
const myurl.pathname = '/a/b/c';  
const myurl.search = '?a=e';  
const myurl.hash = '#fgh'
```

```
console.log(myurl)
```

↳ Prints all value of URL individually

```
console.log(myurl.href)
```

↳ Prints your URL.

Events Modules

Every action on a computer is an event

⇒ Node.js has a built in module, called "Events", where you can create, fire, and listen for your own events.

Eg.

```
const EventEmitter = require('events');  
class myEmitter extends EventEmitter {  
  const myEmitter = new myEmitter();  
  When this event is fired, the below code will run.
```

```
  myEmitter.on('Waterfull', () => {  
    console.log('Turn off motor!');  
    set timeout(() => {  
      console.log('Please turn off  
        the motor');  
    }, 3000);  
  });
```

```
  myEmitter.emit('Waterfull');
```

↳ Here, the event is fired.

Output.

In terminal you will get 'Turn off motor' printed and after every three second

'Please turn off motor' will be printed.

* It is very helpful when you are building a real-time application.

Building an HTTP server

Import: `const http = require('http');`

- i) `const port = process.env.PORT || 3000;`
- ↳ You can set the environment variable `port` to tell your web server what port to listen on.
 - ↳ If no port is there, listen on 3000.

* To open: write `localhost:3000` in your browser.

ii) `http.createServer()`

↳ This method turns your computer into an HTTP server.

Syntax

```
http.createServer((request,
  response) => {
    response.end(); })
```

iii) Status code.

It indicates whether a specific HTTP request has been successfully completed.

Eg.

```
const port = process.env.PORT || 3000;  
const server = http.createServer(  
  (req, res) => {  
    res.statusCode = 200;  
    res.setHeader('Content-Type', 'Text/  
                    HTML')  
  })
```

It means request is served as HTML not plain text.

```
res.end('<h1> This is code </h1>');  
});
```

```
server.listen(port, () => { console.log  
  ('server is listening on port  
   $ {port}');  
});
```

- This will create a server, To access it write localhost:3000 in browser

#Pikacoden

Q. Do You Need Hand
Written Notes of
Exponens ?