

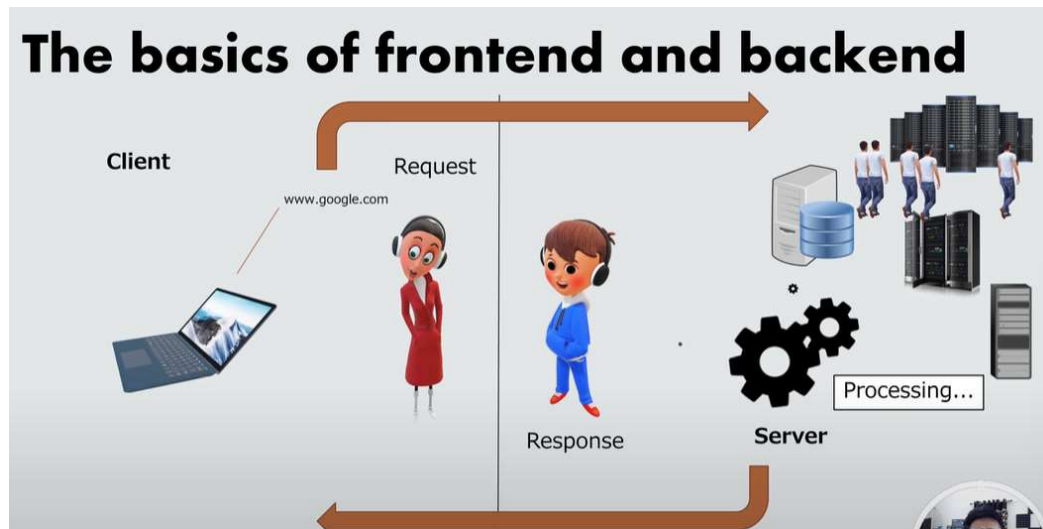
# NodeJS

05 October 2023 12:59 PM

## Client Side vs Server Side

Client requests something to server

Server does some processing and sends the response



NodeJS is used in Backend.

HTML, CSS, JS used in Frontend.

NodeJS is a JS library only so we do not need to learn any extra language.

Install NodeJS

JS works well in Browser

NodeJS is JS which is executable outside browser.

NodeJS developers, combined Chrome browser's V8 engine and C++ and made node.exe which is executable outside browser.

We have a NodeJS Repl (Read Evaluate Print Loop)

```
Node.js
Welcome to Node.js v14.17.1.
Type ".help" for more information.
> 5
5
> 5 + 5
10
> console.log(45)
45
undefined
> const a = 34;
undefined
> a
34
> localStorage
Uncaught ReferenceError: localStorage is not defined
> document.getElementById('er4')
Uncaught ReferenceError: document is not defined
>
```

```
Node.js
Welcome to Node.js v14.17.1.
Type ".help" for more information.
> 5
5
> 5 + 5
10
> console.log(45)
45
undefined
> const a = 34;
undefined
> a
34
> localStorage
Uncaught ReferenceError: localStorage is not defined
> document.getElementById('er4')
Uncaught ReferenceError: document is not defined
>
```

JS executing in Browser is little different than JS executing in Terminal.

Let us make our first NodeJS program.

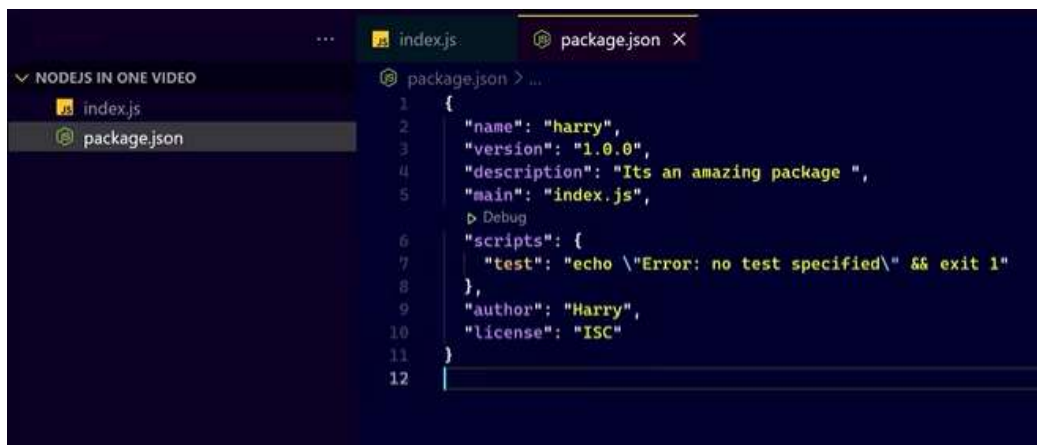
In NodeJS, Single thread can manage multiple connections

Non blocking I/O model

One connection can manage many connections easily. As it is async in nature so it does not block flow of program.

NodeJS uses callback functions to perform above function.

Now we do npm init.



```
index.js  package.json X
package.json > ...
1  {
2    "name": "harry",
3    "version": "1.0.0",
4    "description": "Its an amazing package ",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Harry",
10   "license": "ISC"
11 }
12
```

Packages are some pre-written JS code which we can use in our code.

To install packages.

We do **npm install express --save**

--save means add it in dependency. It is optional.

```

package.json > {} dependencies
{
  "name": "harry",
  "version": "1.0.0",
  "description": "Its an amazing package ",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Harry",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}

```

Node modules can be created again easily by npm i

Node modules has all the dependencies.

Now we install npm i -g nodemon

Dev Dependency

Dependencies which we use only during development.

Package.lock.json has all the transitive dependencies [Dependency of dependency].

Package.lock.json takes snapshots of all dependencies with their version.

Now we make second.js

We want to access second.js from index.js



```

harry = {
  name: "Harry",
  favNum: 3,
  developer: true
}

```

To import second.js in index.js

We use require('./second.js')

We need to export from second.js also.

These modules are called common JS modules.

Second.js is called common JS module.

```
second.js > ...
1  harry = {
2    name: "Harry",
3    favNum: 3,
4    developer: true
5  }
6
7  module.exports = harry;
```

```
index.js > ...
1  const lovish = require("./second");
2
3  console.log("Hello world", lovish)
4
```

```
PS D:\MyData\Business\code playground\NodeJs in one video> node index.js
Hello world { name: 'Harry', favNum: 36, developer: true }
```

## Module Wrapper Function

Whenever NodeJS calls an module, it wraps it inside this function

```
function(exports, require, module, __filename, __dirname){
```

NodeJS does it automatically.

```
(function(exports, require, module, __filename, __dirname){
  harry = {
    name: "Harry",
    favNum: 36,
    developer: true
  }

  module.exports = harry;
})
```

`__dirname` means folder we are in.

`__filename` means file we are working in.

NodeJS has so many Modules like OS module, Path Module etc etc.

Let us make an "osmodule.js"

While importing built-in module, we do not specify path, we directly import them.

```
const os = require('os');
```

Os.freemem tells your free memory space in your OS

```
console.log(os.freemem())  
console.log(os.homedir())  
console.log(os.hostname())  
console.log(os.platform())  
console.log(os.release())
```

```
if one video (osmodule.js)  
57241743360  
C:\Users\Harry  
DESKTOP-UBQMKH7  
Win32  
10.0.19042
```

We can also get Total memory, Up-time of our PC, Type of OS and so on..

## Path Module

It helps in playing with path.

We make pathmodule.js

```
const path = require('path');  
  
const a1 = path.basename('C:\\temp\\myfile.html');  
const a2 = path.dirname('C:\\temp\\myfile.html');  
console.log(a1)  
console.log(a2)  
const a3 = path.extname(__filename)  
console.log(a3)
```

```
pathmodule.js  
myfile.html  
C:\temp  
.js
```

## File System Module

We make fsmodule.js

To play with files, read or write in them. We use this module.

We read a file called 'file.txt'

So let us make this file.

```
fsmodule.js > ...
1  const fs = require('fs');
2
3  fs.readFile('file.txt', 'utf8', (err, data) => {
4    console.log(err, data)
5  })
```

```
odule.js
null this is a file
```

ReadFile is a async function, It does not wait, It fires a callback which gets executed when operation is successful till that, it goes to next line.

Call stack stores next operations and callback queue keeps the callback function. As soon as Call stack is empty, callback goes to call stack and gets executed.

There is another function readFileSync which read the file but is synchronous.

```
module.js > ...
const fs = require('fs');

// fs.readFile('file.txt', 'utf8', (err,
//   console.log(err, data)
// })

const a = fs.readFileSync('file.txt')
console.log(a.toString())

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

```
odule.js"
this is a file
Finished reading file
```

"Finished reading file" comes after "this is a file" because this function is synchronous.

We also have **writeFile and writeFileSync function**

```
fs.writeFile('file2.txt', "This is a data", () => {
  console.log("Written to the file")
});

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

```
odule.js"
Finished reading file
Written to the file
```

WriteFileSync also does the same just that it is synchronous.

**Common JS module vs ES6 modules**



To use ES6 modules we need to use .mjs extension

Common modules are imported using require() and exported like module.exports

ES6 modules are imported using import and exported using export.

We can also make anything as module by specifying **type = module** in package.json

```
package.json > ...
  ▶ Debug
  6   "scripts": {
  7     "test": "echo \"Error: no test specified\" && exit 1"
  8   },
  9   "author": "Harry",
 10   "type": "module",
 11   "license": "ISC",
 12   "dependencies": {
 13     "@angular/cli": "^12.0.5",
 14     "express": "^4.17.1"
 15   },
 16   "devDependencies": {}
 17 }
 18
```

We cannot import .mjs module in normal common JS module using require()

We need to import it like

Inside modulesecond.mjs

```
modulesecond.mjs > simple
1  export function simple(){
2    console.log("Simple is Complex")
3  }
4
5
6  export default function simple2(){
7    console.log("Simple2 is Complex")
8  }
9
```

Inside modulefirst.js

```
modulefirst.js
1  import {simple} from "./modulesecond.mjs"
2  // const simple2 = require("./modulesecond.mjs");
3
4  simple()
```

We can also import simple2 as simple and use simple in our program.

```
modulefirst.js
1  import {simple2 as simple} from "./modulesecond.mjs"
2  // const simple2 = require("./modulesecond.mjs");
3
4  simple()
```

If we export something by default we do not use {} and name may be different also.

Otherwise we need to import using {} and name should be same

We can can more than one imports.

We can also import as \*

Import \* as Obj from "./moduleSecond.mjs"

Means we have put all imports inside Obj and now can access using Obj.something

```
import * as a2 from "./modulesecond.mjs"
// const simple2 = require("./modulesecond.mjs");

// simple23()
console.log(a2.simple())
```

## URL Module

We make a `urlmodule.js`

```
urlmodule.js > ...
1 import url from 'url';
2
3
4 const myURL = new URL('https://example.org');
5 myURL.pathname = '/a/b/c';
6 myURL.search = '?d=e';
7 myURL.hash = '#fgh';
8
9 console.log(myURL)
```

```
URL {
  href: 'https://example.org/a/b/c?d=e#fgh',
  origin: 'https://example.org',
  protocol: 'https:',
  username: '',
  password: '',
  host: 'example.org',
  hostname: 'example.org',
  port: '',
  pathname: '/a/b/c',
  search: '?d=e',
  searchParams: URLSearchParams { 'd' => 'e' },
  hash: '#fgh'
}
```

We can store an URL inside a JS object.

## Events in NodeJS

NodeJS works on event-driven architecture

We can fire an event from anywhere and we take action based on firing an event.

We can make event emitters

We make `nodejsevent.js`

Let us say in the below example we are emitting an event on firing of an event called 'waterfall'



Now our other scripts will run fine but the code inside myEmitter will only work when we emit that event.

We emit the event like myEmitter.emit('waterfall')

```
nodejssevents.js > ...
1  const EventEmitter = require('events'),
2
3  class MyEmitter extends EventEmitter {}
4
5  const myEmitter = new MyEmitter();
6
7  myEmitter.on('WaterFull', () => {
8    console.log('Please turn off the motor!');
9    setTimeout(() => {
10      console.log('Please turn off the motor! Its a gentle reminder');
11    }, 3000);
12  });
13
14  console.log("The script is running")
15  console.log("The script is still running")
16  myEmitter.emit('event');
```

```
nodejssevents.js
The script is running
The script is still running
Please turn off the motor!
Please turn off the motor! Its a gentle reminder
```

If we change the flow to

```
console.log("The script is running")
myEmitter.emit('WaterFull');
console.log("The script is still running")
```

The output looks like

```
nodejssevents.js
The script is running
Please turn off the motor!
The script is still running
Please turn off the motor! Its a gentle reminder
```

As setTimeout is non-blocking in nature so whatever inside setTimeout will run at the end.

## HTTP Server in NodeJS

We make a httpserver.js file

```

httpserver.js > [0] server > http.createServer() callback
1  const http = require('http');
2
3  const port = process.env.PORT || 3000;
4
5  const server = http.createServer((req, res) =>{
6    res.statusCode = 200;
7    res.setHeader('Content-Type', 'text/html')
8    res.end('<h1> This is CodeWithHarry</h1> <p> Hey this is the way to rock the world!</p>');
9  })
10
11  server.listen(port, () =>{
12    console.log('Server is listening on port ${port}');
13  });
14

```

Output



Let us make website.js

Let us make a http server and

Let us make some routes inside it

```

website.js > [0] server > http.createServer() callback
1  const { Console } = require('console');
2  const http = require('http');
3
4  const port = process.env.PORT || 3000;
5
6  const server = http.createServer((req, res) =>{
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/html')
9    console.log(req.url)
10
11    if(req.url == '/'){
12      res.end('<h1> This is CodeWithHarry</h1> <p> Hey this is the way to rock the world!</p>');
13    }
14    else if(req.url == '/about'){
15      res.end('<h1> About CodeWithHarry</h1> <p> Hey this is about CodeWithHarry!</p>');
16    }
17  }
18  })
19
20  server.listen(port, () =>{
21    console.log('Server is listening on port ${port}');
22  });
23

```

Let us now run nodemon

Npm install -g nodemon

Nodemon website.js

Nodemon gives us facility of HMR [Hot Module Replacement]

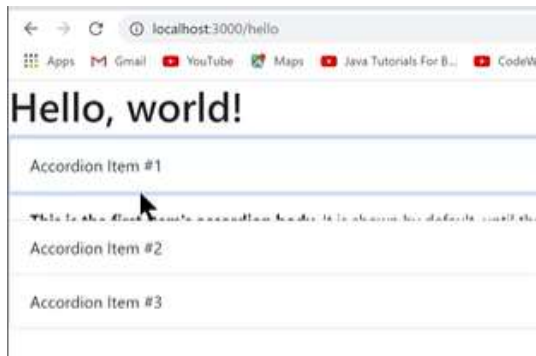
Let us make a index.html and put accordion and some components from bootstrap

We now want to read this file from website.js using NodeJS

We can read the file using readFileSync from file module

```
}  
else if(req.url == '/hello'){  
  res.statusCode = 200;  
  const data = fs.readFileSync('index.html');  
  res.end(data.toString());  
}
```

Output



We can see we need to write so many if-else condition to make routes which makes our code less readable.

So we use express

Npm i express

## Express

### Why express is so famous?

Express is a NodeJS framework

Let us make expressapp.js

```
expressapp.js > ...  
1  const express = require('express')  
2  const app = express()  
3  const port = 3000  
4  
5  app.get('/', (req, res) => {  
6    res.send('Hello World!')  
7  })  
8  
9  app.listen(port, () => {  
10   console.log(`Example app listening at http://localhost:${port}`)  
11 })
```

Production ready apps should be preferably made using Express.

Express makes our work easy.

