Live Coding, Router & Module

Module Pattern:

It is important to get the distinction. We should be familiar with how module system works in ES6. The module system we learnt is ES6, and before this NodeJS already has its own module pattern. It is called common JS module.

We have 2 files, **main.js** and **math.js**

In **main.js**, lets assume we are accepting certain values from the user. It can either be using the server call or some other way. To get the square of the number we can **`console.log(userInput**2)`** will give us the square of 10.

```
const userInput = 10
```

But to understand module system we will put this into a function format in **math.js** file.

```
function square(num1){
    return num1 * num1
}

function cube(num1) {
    return num1 * num1 * num1
}
```

We need to import the functions of **math.js** into **main.js** file. In ES6 modules we were able to do it using **export**. Unless NodeJS directly supports this, we are at loss. So, in-order to do it we need to understand what is module. When we do console.log(module) and see how module object looks like;

```
PS C:\Users\user\Desktop\prc> node math.js
Module {
  id: '.',
  path: 'C:\\Users\\user\\Desktop\\prc',
  exports: {},
  parent: null,
  filename: 'C:\\Users\\user\\Desktop\\prc\\math.js',
  loaded: false,
```

```
  children: [],
  paths: [
    'C:\\Users\\user\\Desktop\\prc\\node_modules',
    'C:\\Users\\user\\Desktop\\node_modules',
    'C:\\Users\\user\\node_modules',
    'C:\\Users\\node_modules',
    'C:\\node_modules'
  ]
}
```

Module object has paths, file name, children, and exports. In NodeJS every javascript file is a module in itself. This keyword module is referring to the current **math.js** file. If we do console.log in **main.js**, the path would be **main.js**

```
Module {
  id: '.',
  path: 'C:\\Users\\user\\Desktop\\prc',
  exports: {},
  parent: null,
  filename: 'C:\\Users\\user\\Desktop\\prc\\main.js',
  loaded: false,
  children: [],
  paths: [
    'C:\\Users\\user\\Desktop\\prc\\node_modules',
    'C:\\Users\\user\\Desktop\\node_modules',
    'C:\\Users\\user\\node_modules',
    'C:\\Users\\node_modules',
    'C:\\node_modules'
  ]
}
```

Notice in for **main.js** and **math.js** has exports empty. Module.exports is an empty object and this export object is what NodeJS uses to export a particular function from a file.

```
function square(num1){
    return num1 * num1
}

function cube(num1) {
    return num1 * num1 * num1
}

module.exports = {
    square: square,
    cube: cube
}

console.log(module)
```

In the terminal we can see,

```
Module {
  id: '.',
  path: 'C:\\Users\\user\\Desktop\\prc',
  exports: { square: [Function: square], cube: [Function: cube] },
  parent: null,
  filename: 'C:\\Users\\user\\Desktop\\prc\\math.js',
  loaded: false,
  children: [],
  paths: [
    'C:\\Users\\user\\Desktop\\prc\\node_modules',
    'C:\\Users\\user\\Desktop\\node_modules',
    'C:\\Users\\user\\node_modules',
    'C:\\Users\\node_modules',
    'C:\\node_modules'
  ]
}
```

We are now defining what should be exported from this module. This is how we will be exporting stuff from our module. We have added the functions to our exports so we can import it into our **main.js** file. However, import does not work in NodeJS v14 and less.

So,

```
const mathOperations = require('./math')

console.log(mathOperations)
const userInput = 10
```

the object that is getting attached is,

{ square: [Function: square], cube: [Function: cube] }

The same object that we were getting in exports. So, NodeJS is linking everything. We can now say,

```
const mathOperations = require('./math')

const userInput = 10

const sq = mathOperations.square(userInput)
console.log(sq);
```

We are importing math.js file using require(), which inherently is a module in itself and we are adding an object to module.exports, which is math.js. We can do in **main.js**, but for now, we are not using **main.js** to get different functionalities into another file. If we change the property name that would be reflected as well.

Another way of doing this, we can directly use the function expression in module.exports

```
module.exports = {
    square: function(num1) {
        return num1 * num1
    },
    cube: function(num1) {
        return num1 * num1 * num1
    }
}
```

Instead of doing math operations, we can directly define the function in the object itself.

We can define all functionalities in another file and import them into another file to execute them.

It is important to separate our project into individual small modules, so that we can use it atomically. It should not in one file and we are getting confused to make changes. To avoid such headache, we separate our project into multiple files.

So **main.js** will be the main file and into it we will be importing different files.

Instead of doing math operations, we can do something like,

```
const { square } = require('./math')

// console.log(mathOperations)
const userInput = 10

const sq = square(userInput)
console.log(sq);
```

We were able to de-structure it since an object is being exported. Even though the object is exporting two functions, but we are able to import one of the functions and the second function was using only one of it. In de-structuring, we are able to import only one function and it is much more memory efficient and the second function cube will be removed from the memory.

As the project increases, we might not know which function is getting imported. So, always import using de-structuring. There are other 3rd party libraries, like web pack and parcel, that when it is not being used it will be removed.

De-bugging:

So, how to debug node application? Directly from the IDE. If want to debug **main.js** file, all we need to do is press F5, and then to the left click on Run & Debug.

It will debug the file. To the left we can see local and global details. To the top right corner, we have the step into, step over and step ____. In any case if there is an error, this is the process to debug. If the configuration is different, it will show a list, select NodeJS from it and run it.

Express – Node:

Using HTTP request, we will try to get square and cube. So,

localhost3000:/square/121 → should give us the output of the square of 121.

```
const express = require('express');
const app = express();

app.get('/square/:num', (req, res) => {
    const {num} = req.params
    console.log(typeof num)
    res.send('Works)')
})

app.listen(3000, () => console.log('server started'))
```

Using nodemon, it will start the node index.js file automatically. In the address bar when we enter,

localhost:3000/square/123

the output would be ‘Works’.

In the terminal, 123 is also logged in. If we check the type of num, then it shows it as string. So to change it from strin g to number we had to add, num = ***Number***(num) so that the value is converted to the integer data typeof.

```
const {squre, cube} = require('./math')
const express = require('express');
const app = express();

app.get('/square/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const sq = square(num)
    res.send(`${sq}`)
})

app.listen(3000, () => console.log('server started'))
```

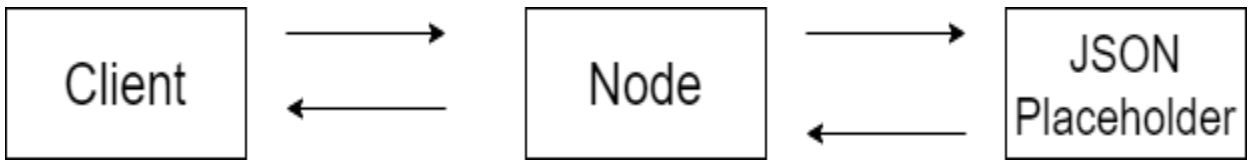
localhost3000:/square/11 → 121

All we have to do now is import our square and cube functions into the **index.js** file.

```
const {squre, cube} = require('./math')
const express = require('express');
const app = express();

app.get('/square/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const sq = square(num)
    res.send(`${sq}`)
})

app.get('/cube/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const cu = cube(num)
    res.send(`${cu}`)
})

app.listen(3000, () => console.log('server started'))
```

our next approach or functionality would be using json placeholder.

In JSON placeholder, we have different resources. In resources, if we select post, it is referenced using userid. Now, when we call /users, we get a list of 10 users with their details. Just as an example, if we are calling from the NodeJS server, we will be calling a HTTP request for json placeholder. Meaning, our server will use NodeJS and call json placeholder



One of the packages is node fetch. There is a better library called axios.

Axios is a npm package and can be used inside NodeJS. To perform a GET request; we will first need to install axios. In the dependencies, it would be listed. Then we need to import it as per the documentation.

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

app.get('/users', (req, res) => {
    const rs = axios.get('')
    console.log(rs)

    res.send('Works')
})

app.listen(3000, () => console.log('server started'))
```

await is only used for async function. But the function written is not an async function. So, to make it an async function, we need to add async before the function,

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

app.get('/users', async (req, res) => {
    const rs = await axios.get('https://jsonplaceholder.typicod
e.com/users')
    console.log(rs)

    res.send('Works')
})

app.listen(3000, () => console.log('server started'))
```

We have lot of data attached to the responseObj. Data is attached to response object, so we can say,

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

app.get('/users', async (req, res) => {
    const responseObj = await axios.get('https://jsonplaceholde
r.typicode.com/users')
    console.log(responseObj.data)

    res.send('Works')
})

app.listen(3000, () => console.log('server started'))
```

We can loop over it as it is an array using map.

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

app.get('/users', async (req, res) => {
    const responseObj = await axios.get('https://jsonplaceholde
r.typicode.com/users')

    responseObj.data.map(user) => {
        console.log(user)
    }

    res.send('Works')
})

app.listen(3000, () => console.log('server started'))
```

Each of them is logged individually as there is not square bracket. Now we will use thinuser,

```
app.get('/users', async (req, res) => {
    const responseObj = await axios.get('https://jsonplaceholde
r.typicode.com/users')

    const thinuser = responseObj.data.map((user) => {
        delete user.address
        delete user.company
        return user
    })

    res.send('Works')

})

app.listen(3000, () => console.log('server started'))
```

We get ‘Works’ again on the screen. In the terminal we are only getting id, name, username, email, phone and website. Address and Company are removed. Now that the data is ready, we will send this data as json.

```
app.get('/users', async (req, res) => {
    const responseObj = await axios.get('https://jsonplaceholde
r.typicode.com/users')

    const thinuser = responseObj.data.map((user) => {
        delete user.address
        delete user.company
        return user
    })

    response.json(thinuser)

})

app.listen(3000, () => console.log('server started'))
```

Now, we will get the same result, but we are pre-processing it. This is obviously depending on the project and what we want to achieve. The idea here is to understand that we can create a http request inside NodeJS and call different server. Here the example is server to server. Previously, it was from client to server call.

There are many routes in the json placeholder and in the same way we have also many routes in the **index.js** file. If we keep adding it will be difficult for us to make changes. So, just as we split our functions, we can also define the routes in different files.

There is an object in express called router object. This is exactly what we will use to split our router objects.

We will create two files math-utilities.js and user.js files. In math.js

```
const {squre, cube} = require('../math-utilities')
const express = require('express')
const router = express.Router()

router.get('/square/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const sq = square(num)
    res.send(`${sq}`)
})
```

```
router.get('/cube/:num', (req, res) => {
    let {num} = req.params
    num = Number(num)
    const cu = cube(num)
    res.send(`${cu}`)
})

module.exports = router
```

in **user.js**

```
const express = require('express')
const router = express.Router()
const axios = require('axios')

router.get('/users', async (req, res) => {
    const responseObj = await axios.get('https://jsonplaceholde
r.typicode.com/users')

    const thinuser = responseObj.data.map((user) => {
        delete user.address
        delete user.company
        return user
    })

    response.json(thinuser)

})

module.exports = router
```

The math-utilities.js file is outside one directory, so we need to use ../math-utilities. We have separated our files, and we need to tell our index.js file to import them from other files. So, in **index.js**

```
const express = require('express')
const app = express();
const mathRouter = require('./routes/math');
const userRouter = require('./routes/user');

app.use('./math', mathRouter)
app.use('./user', userRouter)

app.listen(3000, () => console.log('server started'))
```

This is how we can extract our route handlers.