# CIA2503: Web Applications Development

# Lab/Activity 10

Course Learning Outcome: CLO4- Create dynamic web applications using current standards-compliant server-side web technologies

Chapters: Chapter 10

Cognitive Levels: Apply

Aim: The student should be able to:

* Create Node.js modules and express code modularity in an application
* Install and use Express.js and create HTTP request/response Objects.
* Build an HTTP server using the core modules in Node.js
* Create web forms on an HTML page and receive the data in a Noje.js script on the server.
* Validate form data using Node.js module

Tools: Command line Prompt, Nodejs LTS, Visual Studio code, Express

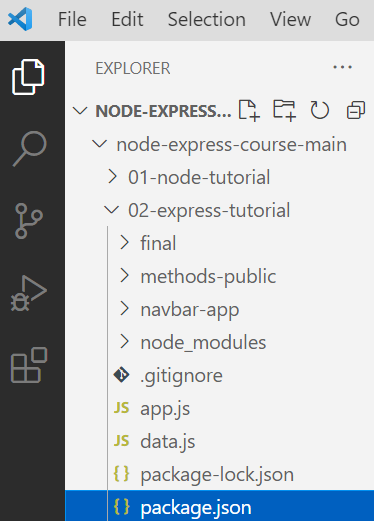
Document Revision Control: 001

# [Example- Understand]

How do we exchange data on the web? User sends the http request message and server send the response. We will build this using node and express. The framework Express will make our life much easier. Web server must make the resource available and we need to make sure that the server is running.

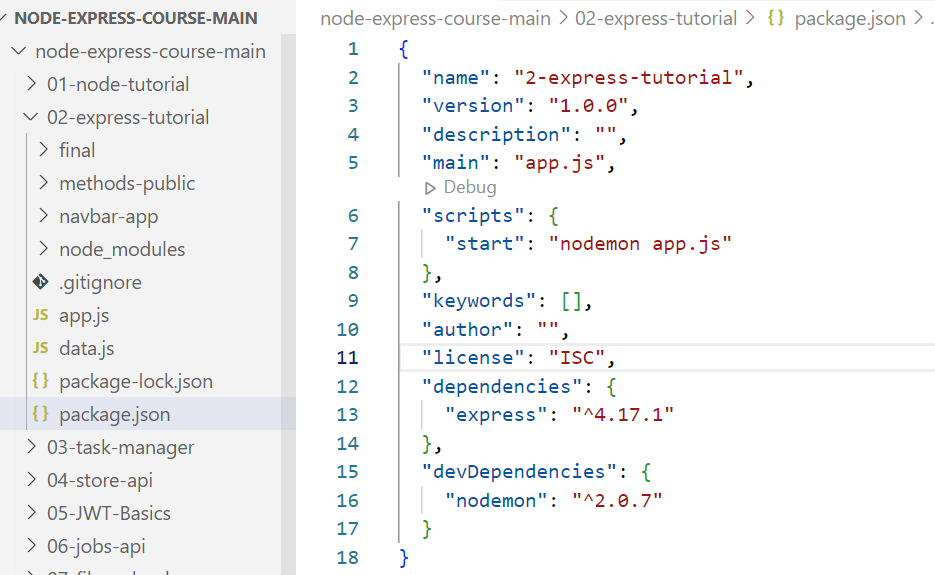
Download and extract the zip file for **02-express-tutorial** and open in VSCode (File located in Week 11). Try to save in a similar location as before. We will use a starter project to demonstrate Express.

Browse to the folder location of 02-express-tutorial and look inside of the **package.json** file.



Several files have already been created inside of the **final** folder which we will use to demonstrate. These will be run using app.js. You can refer to them if you get stuck.

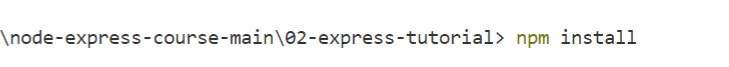
The **package.json** file is used to keep track of dependencies



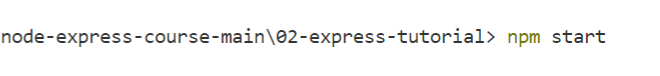
In the terminal window, browse to the tutorial location run command ***npm install*** to set up the needed dependencies.

Then run ***npm start***

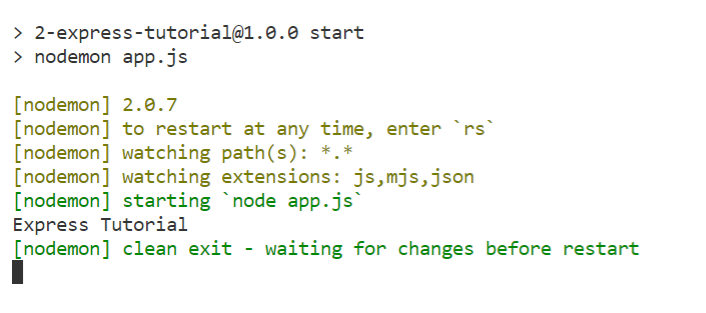
*See example below:*



Followed by:



You should see the following:



As we can see in the Express Tutorial, it seems that everything is working fine.

# Exercise 1- APPLY – Create and HTTP Server with Request and Response Objects

We will now set up the server in our project using **http module** in the app.js file

1. Open app.js and type or copy & paste the following code:

const http = require('http') // http built in module

//use parameters req and res for our server as we need access to these

const server = http.createServer((req,res)=>{

    console.log('user hit the server');

})  //a callback function will be called every time a user hits the sever

server.listen(5000) // listen on port 5000

Text

Description automatically generated

1. Browse to localhost:5000
2. Then return to your Terminal window in VScode to see the message confirming the user hit the server. See below:

Text

Description automatically generated

Noting really happened in the client side. The method **response.end(),** MUST be called on each response.

1. Add the code highlighted below

const http = require('http')

const server = http.createServer((req,res)=>{

    console.log('user hit the server');

    res.end('homepage')

})

server.listen(5000)

1. Save your code in app.js and take a look in the localhost:5000 and refresh.

Graphical user interface, text, application

Description automatically generated

We are done with the basics of creating an server.

# Exercise 2- APPLY – Response Codes

Unfortunately we don’t have any data about what is being sent back from the server. We want to provide more information to the browser by sending a status code (200 – OK, etc.).

1. Using the same file as in previous exercise, add the highlighted code as follows:

const http = require('http')

const server = http.createServer((req,res)=>{

res.writeHead(200,{'content-type':'text/html'}); //return status code

    res.write('<h1>homepage</h1>')

    res.end()

})

server.listen(5000)

1. Save the code and then refresh the browser to see the following:

Graphical user interface, application

Description automatically generated

1. Think about the status codes which we could return. We learned this in the beginning of the course.

MIME (Multipurpose Internet Mail Extensions) type is **a standard way of describing a data type in the body of an HTTP message or email**. Hopefully express will take care of this.

1. Let’s think about our request object in our code. Add the highlighted code to return the method used for the request.

const http = require('http')

const server = http.createServer((req,res)=>{

    console.log(req.method)

    res.writeHead(200,{'content-type':'text/html'}) //return code status

    res.write('<h1>homepage</h1>')

    res.end()

})

server.listen(5000)

1. Save app.js then take a look at the terminal window. It should show that request method used is GET.

Text

Description automatically generated

1. Add the highlighted code and save.
2. Browse to **localhost:5000/about** and see the request for the url being returned

const http = require('http')

const server = http.createServer((req,res)=>{

console.log(req.method)

console.log(req.url)

    res.writeHead(200,{'content-type':'text/html'}) //return status code

    res.write('<h1>homepage</h1>')

    res.end()

})

server.listen(5000)

Text

Description automatically generated

1. Update the code to send back 3 different responses (status codes) based on the page which the user requested. Test in the browser. Remember to save the app.js file.

const http = require('http')

const server = http.createServer((req,res)=>{

const url = req.url;

    //home page

    if(url === '/'){

        res.writeHead(200,{'content-type':'text/html'}) //return status code

res.write('<h1>homepage</h1>')

        res.end()

    }

    //about page

    else if(url === '/about'){

        res.writeHead(200,{'content-type':'text/html'})  //return status code

        res.write('<h1>about page</h1>')

        res.end()

    }

    //404 resource not found

    else{

        res.writeHead(404,{'content-type':'text/html'})  //return status code

        res.write('<h1>page not found</h1>')

        res.end()

    }

})

server.listen(5000)

What the user sees returned in the browser will be dependent on what request has been made.

# Exercise 3 - APPLY – Referencing and Presenting a Webpage as Response

1. Now let’s go a little further. Within the 02-express-tutorial, create a new basic **index.html** page with a couple of simple tags, a heading and a paragraph. Remember to save. See below

<html>

<p>Hello World</p>

</html>

It should appear under the app.js and the data.js files. See below:

Graphical user interface, application, Word

Description automatically generated

1. Update the code in the app.js file. Instead of presenting the message, let’s reference and present our index page (homePage) which we have just created. See the code changes below.

const http = require('http')

const { readFileSync } = require('fs')

//get all files

const homePage = readFileSync('./index.html')

const server = http.createServer((req,res)=>{

    const url = req.url;

    //home page

    if(url === '/'){

     res.writeHead(200,{'content-type':'text/html'}) //return status code, header

        res.write(homePage);

        res.end()

    }

    //about page

    else if(url === '/about'){

        res.writeHead(200,{'content-type':'text/html'})  //provide status code, header

        res.write('<h1>about page</h1>')

        res.end()

    }

    //404 resource not found

    else{

        res.writeHead(404,{'content-type':'text/html'})  //provide status code, header

        res.write('<h1>page not found</h1>')

        res.end()

    }

})

server.listen(5000)

1. Now browse to the localhost port 5000 as below and see the referenced index.html page appear:

Graphical user interface, text, application

Description automatically generated

1. Let’s use a more complex index page which we can find in the folder navbar-app. In the same code, update the path of the homepage to the following:

const homePage = readFileSync('./navbar-app/index.html')

1. Now browse to the localhost port 5000 and see the new index.html page appear:

Graphical user interface, text

Description automatically generated

1. Not everything on the page is appearing unfortunately as expected. We need to request all missing resources which are required by the new index.html such as the css, etc. Update your code (see additional codes highlighted) in the **app.js** file.

const http = require('http')

const { readFileSync } = require('fs')

//get all files

const homePage = readFileSync('./navbar-app/index.html')

const homeStyles = readFileSync('./navbar-app/styles.css')

const homeImage = readFileSync('./navbar-app/logo.svg')

const homeLogic = readFileSync('./navbar-app/browser-app.js')

const server = http.createServer((req,res)=>{

    const url = req.url;

    console.log(url)

    //home page

    if(url === '/'){

        res.writeHead(200,{'content-type':'text/html'})  //provide the status code, header

        res.write(homePage);

        res.end()

    }

    //about page

    else if(url === '/about'){

        res.writeHead(200,{'content-type':'text/html'})  //provide the status code, header

        res.write('<h1>about page</h1>')

        res.end()

    }

    //styles

    else if(url === '/styles.css'){

        res.writeHead(200,{'content-type':'text/css'})

        res.write(homeStyles)

        res.end()

    }

    //image/logo

    else if(url === '/logo.svg'){

        res.writeHead(200,{'content-type':'image/svg+xml'})

        res.write(homeImage)

        res.end()

    }

    //logic

    else if(url === '/browser-app.js'){

        res.writeHead(200,{'content-type':'text/javascript'})

        res.write(homeLogic)

        res.end()

    }

    //404 resource not found

    else{

        res.writeHead(404,{'content-type':'text/html'})  //provide the status code, header

        res.write('<h1>page not found</h1>')

        res.end()

    }

})

server.listen(5000)

1. Save then run in the browser. You should see the following if all of the files/resources have been properly accessed:

Graphical user interface, application

Description automatically generated

1. You can inspect your file to make sure that all of the resources have been accessed as expected. To do this right click and inspect and select resources button.

Graphical user interface, application, Word

Description automatically generated

Now this is a lot of hard work. There must be a better way!

# Exercise 4 - APPLY – Installing Expressjs Framework for Node.js

Browse to ***expressjs.com***. Express is a fast, minimalist web framework for nodejs. It makes life so much easier to create apps than using nodejs alone.

1. See below:

Graphical user interface, website

Description automatically generated

1. To install a stable version of Expressjs then use 4.17.1. Write the following command in the terminal.

**npm install express@4.17.1 –save**

# Exercise 4 - APPLY – Installing Expressjs Framework for Node.js

1. Remove/delete all of the code in app.js file. Don’t worry as this has been saved in the folder final under 2-http-app.js. We are now going to use express.
2. Import and then invoke the express module which we have just installed.
3. Set up the listener with a callback on port 5000 and output a message.
4. Save app.js. Then use ***npm start*** to see the console log message in the terminal as follows:

Graphical user interface, text, application, email

Description automatically generated

1. What about the other methods such as:

GET Read Data

POST Insert Data

PUT Update Data

DELETE Delete Data

1. Test the **GET** method with a simple message being send back. Add the highlighted code and test in the browser localhost: 5000

const express = require('express');  //import the module

const app = express(); // invoke the express module

app.get('/', (req,res)=>{

    res.send('Home Page')

})

app.listen(5000,()=>{

    console.log('server is listening onport 5000')

})

Graphical user interface, text, application

Description automatically generated

1. Add more to the latest code in **app.js**. Let the browser search for an **about page**. If the requested resource is found, it will send a status code of **200** and send the message ‘**About Page**’. If a resource requested is not found then a status code of **404** will be returned with the heading message ‘**Resource not found**’.

const express = require('express');  //import the module

const app = express(); // invoke the express module

app.get('/', (req, res)=>{

    console.log('user hit the resource');

    res.status(200).send('Home Page')

})

app.get('/about', (req, res)=>{

    res.status(200).send('About Page')

})

app.all('\*',(req, res)=>{

    res.status(404).send('<h1>resource not found</h1>')

})

app.listen(5000,()=>{

    console.log('server is listening on port 5000')

})

1. Test and browse in the localhost as shown in the 2 images below:

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

1. This exercise is stored in the **final/04-express-app.js,** if you want to revisit later.

# Exercise 5 - APPLY – Referring to Multiple Static Asset Files in a Public Folder

1. Update the app.js file by deleting some codes. The previous exercise code has been prior created and saved in the final folder for you to rerun later. Your **app.js** file code template for this exercise should start in the following way:

Graphical user interface, text

Description automatically generated

1. Create a new public folder inside of the 02-express-tutorial and add 4 resources needed for your app. You will then refer to the public folder from your code so that it renders all required resources on the browser. Copy and paste the 4 resources from the **navbar-app** folder. See below:

Graphical user interface, application

Description automatically generated

1. ***Note:*** If we have many images in our website, then we can place them all into the public folder. These are considered to be static assets. We can also add the index.html to our static assets public folder. This is the beauty of express.
2. Update your app.js code as highlighted below:

const express = require('express');  //import the module

const path = require('path'); // use the pre-installed path module

const app = express(); // invoke the express module

//set up static and middleware

app.use(express.static('./public'))

//With express place all static resources into a folder and place copy of resources to public folder

app.get('\*',(req,res)=>{

 res.status(400).send('resource not found')

})

app.listen(5000,()=>{

    console.log('server is listening on port 5000....')

})

1. Save the **app.js** file and then refresh in the **localhost:/5000**. Everything should appear as expected with correct styling as it has access to all files including the css. See below:

Graphical user interface, text, application, Word

Description automatically generated

1. This is already stored in the **final/05-static-all.js** if you want to rerun again later.

# Exercise 6 - APPLY – API construct using Express set ups to interact with some data

**API (HTTP interface to interact with our data) VS. SSR (server side rendering)**

API – JSON(JavaScript Object Notation) SSR - TEMPLATE

SEND DATA SEND TEMPLATE

RES.JSON() (Response and heavy lifting in the app) RES.RENDER() (Server side rendering)

We will create an API construct using more advanced Express set ups to interact with some data.

1. Update the **app.js** file by deleting some codes. Now add the highlighted codes so that it reads from a json data structure and send a response. See below:

const express = require('express');  //import the module

const app = express(); // invoke the express module

app.get('/',(req,res)=>{

    res.json([{name:'john' },{name:'susan' }])

})

app.listen(5000,()=>{

    console.log('server is listening on port 5000....')

})

1. Save the **app.js file** and browse to **localhost:5000**. You should see the following:

Graphical user interface, text, application

Description automatically generated

1. We can effectively use data now within our app. This is very simple so we should be able to do something much more meaningful. In 02-express-tutorial, take a look inside the file named **data.js**. See below:

A picture containing text

Description automatically generated

1. The **data.js** is simply a file with some arrays of products or people. At the end of the file you will notice that both of the arrays will be exported. We are now going to attempt to import **products** from **data.js** into the **app.js.** Update the **app.js** code as highlighted below then save and refresh in your **localhost:5000**

const express = require('express');  //import the module

const app = express(); // invoke the express module

// uses the data.js file to import into the products

const { products } = require('./data')

app.get('/',(req,res)=>{

//send response with the constant data products which we have just imported from data.js

    res.json(products)

})

app.listen(5000,()=>{

    console.log('server is listening on port 5000....')

})

1. Save the app.js file and refresh in the localhost. You should see all of the product’s data imported from the data.js.

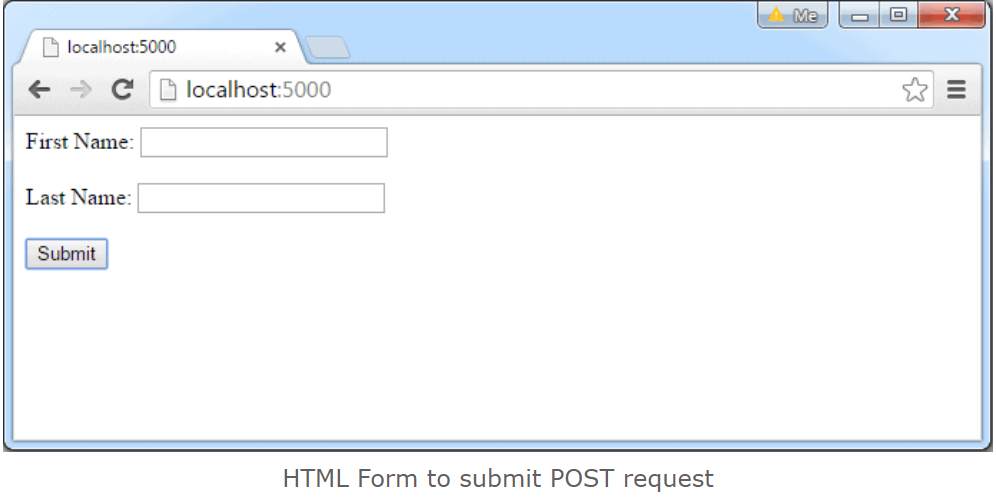
A picture containing scatter chart

Description automatically generated

1. This code will be saved in **02-express-tutorial**/**final/6-basic-json.js**

# Exercise 7 - APPLY - Create Web Form on an HTML page and receive Data in a Node.js Script on Server using Express

We will create a basic form for the client side. The user will be able to enter some simple data (First Name and Last Name) and submit the data which will be received on the server side in a **nodejs** script. You will learn how to handle HTTP POST request and get data from the submitted form.



1. First, create index.html file in the root folder of your application in VS and write the following HTML code in it. (***Note:*** if you already have a file with this name then you can name it index2.html). The index file will place itself underneath the app.js file.

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<form action="/submit-student-data" method="post">

First Name: <input name="firstName" type="text" /><br/><br/>

Last Name: <input name="lastName" type="text" /><br/><br/>

<input type="submit" />

</form>

</body>

</html>

1. To handle HTTP POST request in Express.js version 4 and above, you need to install a middleware module called [body-parser](https://github.com/expressjs/body-parser). The middleware was a part of Express.js earlier but now you have to install it separately. This body-parser module parses the JSON, buffer, string and url encoded data submitted using HTTP POST request. Install body-parser using NPM as shown below.

npm install body-parser –save

Make sure you are in the 02-express-tutorial directory when you install the body-parser

1. Now remove the code which is in your current **app.js** file and add the following server side code to handle the post requests:

var express = require('express');

const path = require('path'); // use the pre-installed path module

var app = express();

var bodyParser = require("body-parser");

app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', function (req, res) {

res.sendFile(path.resolve(\_\_dirname, 'index.html'));

});

app.post('/submit-student-data', function (req, res) {

var name = req.body.firstName + ' ' + req.body.lastName;

res.send(name + ' Submitted Successfully!');

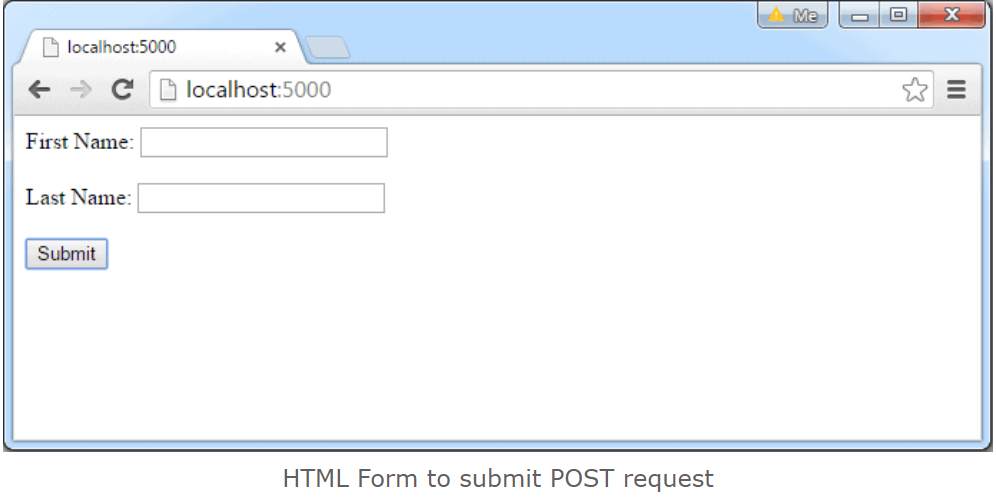
});

var server = app.listen(5000, function () {

console.log('Node server is running..');

});

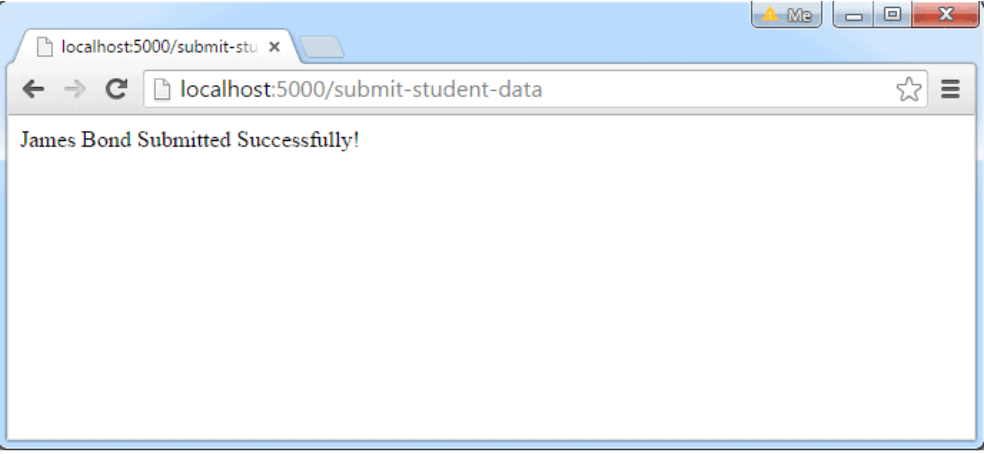
1. Save the new code in the app.js file. Remember to use command **npm start** if you are using nodemon or **node app.js** in the terminal window to make sure that the server has started.
2. Now, run the example by pointing your browser to *http://localhost:5000* and add some data into the text boxes and then submit.

****

Bond

James

1. After submitting, both names entered will be confirmed on the browser if it has been successfully handled in by the nodejs server script (app.js). See the following browser output:



***Note:*** *This is how you can handle HTTP get and post requests using Express.js.*

1. Try adding more text boxes to your index.html form and handle the requests in the nodejs script.

# Exercise 8 - APPLY – Validate Simple Data using Nodejs Validator Module

The **Validator** module is popular for validation. Validation is necessary to check whether the data is correct or not, so this module is easy to use and validates data very quickly and easily.

**Installation of validator module:**

1. You can visit the link [Install validator module](https://www.npmjs.com/package/validator). You can install this package by using this command.

**npm install validator**

1. After installing validator module you can check your validator version in command prompt using the command.

**npm version validator**

1. Then create a new folder (in 02-express-tutorial) and add a file for example **index.js** (as shown below), to run this file you need to run the following command.

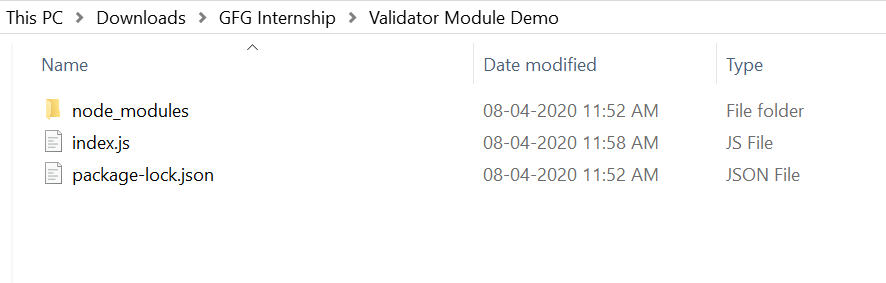
**node index.js**

**Filename: index.js**

|  |
| --- |
| const validator = require(**'validator'**)  **// Check whether given email is valid or not**  var email = **'test@gmail.com'**  console.log(validator.isEmail(email))  **// true**  email = **'test@'**  console.log(validator.isEmail(email))  **// false**    **// Check whether string is in lowercase or not**  var name = **'geeksforgeeks'**  console.log(validator.isLowercase(name))  **// true**  name = **'GEEKSFORGEEKS'**  console.log(validator.isLowercase(name))  **// false**    **// Check whether string is empty or not**  var name = **''**  console.log(validator.isEmpty(name))  **// true**  name = **'geeksforgeeks'**  console.log(validator.isEmpty(name))  **// false**    **// Other functions also available in**  **// this module like isBoolean()**  **// isCurrency(), isDecimal(), isJSON(),**  **// isJWT(), isFloat(), isCreditCard(), etc.** |

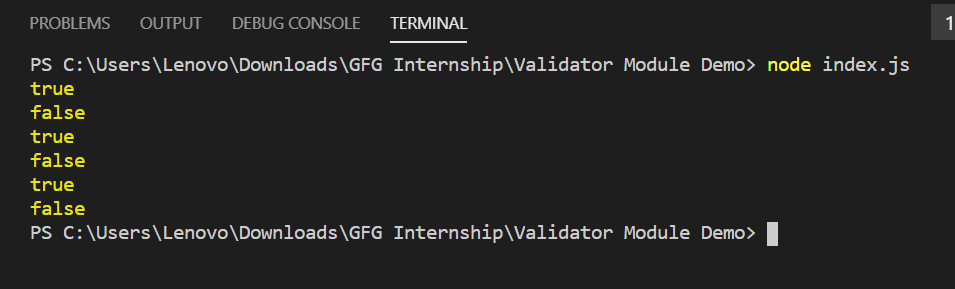
**Steps to run the program:**

The project structure should look something like this depending on where you have saved your file/folder:



1. Run index.js file using below command:

**node index.js**



So this is how you can use the validator module for validation. Notice the console output for the data values hard coded in the program.

There are also other modules available in market for validation like hapi-joi, express-validator, etc.

# Exercise 8 - APPLY – Validate Data using Joi Module in Node.js

**Joi** module is a popular module for data validation. This module validates the data based on schemas. There are various functions like optional(), required(), min(), max(), etc. which make it easy to use and a user-friendly module for validating the data.

1. You can visit the link [**Install joi module**](https://www.npmjs.com/package/joi). You can install this package by using this command.

**npm install joi**

1. After installing you can check your joi version in command prompt using the command.

**npm ls joi**

1. After that, you can just create a folder and add a file for example **index.js**. In this example we will validate a user as highlighted. To run this file you need to run the following command.

**node index.js**

**Filename: index.js**

const Joi = require('joi')

//User-defined function to validate the user

function validateUser(user)

{

const JoiSchema = Joi.object({

username: Joi.string()

.min(5)

.max(30)

.required(),

email: Joi.string()

.email()

.min(5)

.max(50)

.optional(),

date\_of\_birth: Joi.date()

.optional(),

account\_status: Joi.string()

.valid('activated')

.valid('unactivated')

.optional(),

}).options({ abortEarly: false });

return JoiSchema.validate(user)

}

const user = {

username: 'Pritish',

email: 'pritish@gmail.com',

date\_of\_birth: '2020-8-11',

account\_status: 'activated'

}

response = validateUser(user)

if(response.error)

{

console.log(response.error.details)

}

else

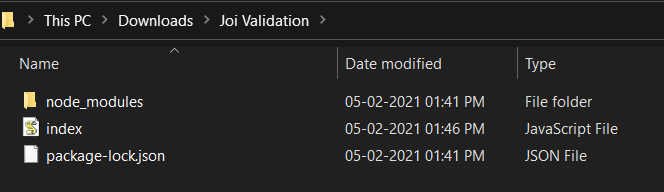
{

console.log("Validated Data")

}

**Note:** In the above program abortEarly is set to false which makes sure that if there are multiple errors then all are displayed in the terminal. If it is set to true then the execution of the program will stop as soon as the first error is encountered and only that error will be displayed in the terminal.

**Steps to run the program:**   
The project structure should look something like this depending on where you have saved your file/folder:



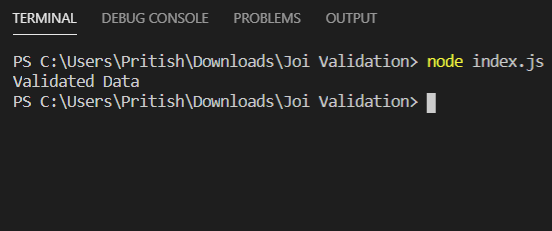
1. Make sure you have installed the joi module by using the following command:

**npm install joi**

1. Run index.js file using below command:

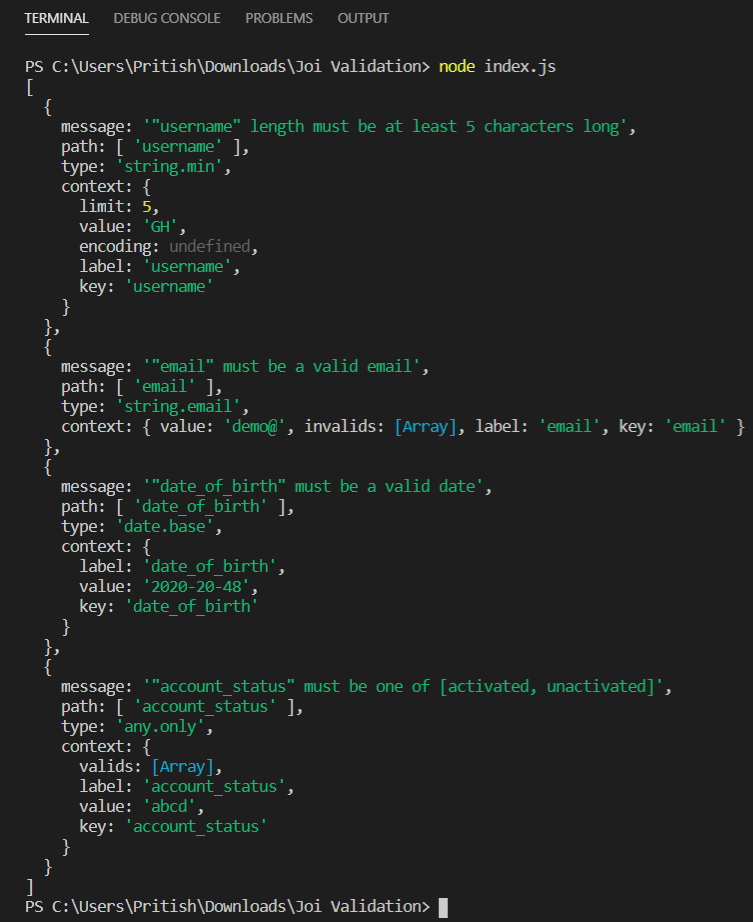
**node index.js**

1. If no error occurs i.e. user data is validate, then following output will be produced:

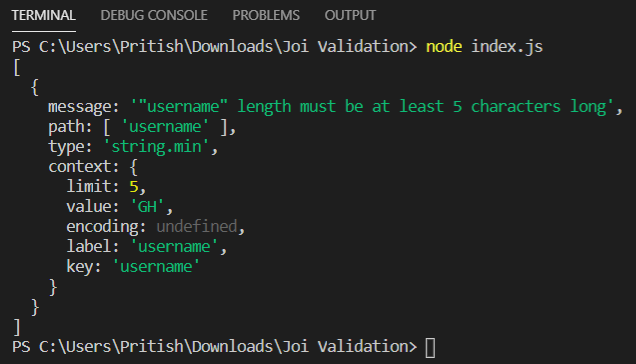


1. Now, if we validate the user against the invalid data as highlighted below (try in your code), then the following output will be produced:

|  |
| --- |
| **var user = {**  **username: 'GH',**  **email: 'demo@',**  **date\_of\_birth: '2020-20-48',**  **account\_status: 'abcd'**  **};** |



If **abortEarly** is set to true the following output will be produced:



So this is how you can validate data using **joi** module. There are other modules in the market for validation like express-validator, etc.

# Exercise 9 - APPLY – Validate Form Data using express-validator Module in Node.js

Validation in node.js can be easily done by using the express-validator module. This module is popular for data validation. There are other modules available in market like hapi/joi, etc but express-validator is widely used and popular among them.

**Steps to install express-validator module:** 

1. You can install this package by using this command.

**npm install express-validator**

1. After installation, you can check your express-validator module version in command prompt using the command.

**npm version express-validator**

1. After that, create a simple form as shown below to send the data to the server. We will use port 3000 for this example.

**Filename: SampleForm.ejs**

<!DOCTYPE html>

<html>

<head>

<title>Validation using Express-Validator</title>

</head>

<body>

<h1>Demo Form</h1>

<form action="saveData" method="POST">

<pre>

Enter your Email : <input type="text" name="email"> <br>

Enter your Name : <input type="text" name="name"> <br>

Enter your Number : <input type="number" name="mobile"> <br>

Enter your Password : <input type="password" name="password"> <br>

<input type="submit" value="Submit Form">

</pre>

</form>

</body>

</html>

1. After that, you can just create a file, for example **index.js** as show below:

**Filename: index.js** 

const { check, validationResult }

= require('express-validator');

const bodyparser = require('body-parser')

const express = require("express")

const path = require('path')

const app = express()

var PORT = process.env.port || 3000

// View Engine Setup

app.set("views", path.join(\_\_dirname))

app.set("view engine", "ejs")

// Body-parser middleware

app.use(bodyparser.urlencoded({ extended: false }))

app.use(bodyparser.json())

app.get("/", function (req, res) {

res.render("SampleForm");

})

// check() is a middleware used to validate

// the incoming data as per the fields

app.post('/saveData', [

check('email', 'Email length should be 10 to 30 characters')

.isEmail().isLength({ min: 10, max: 30 }),

check('name', 'Name length should be 10 to 20 characters')

.isLength({ min: 10, max: 20 }),

check('mobile', 'Mobile number should contains 10 digits')

.isLength({ min: 10, max: 10 }),

check('password', 'Password length should be 8 to 10 characters')

.isLength({ min: 8, max: 10 })

], (req, res) => {

// validationResult function checks whether

// any occurs or not and return an object

const errors = validationResult(req);

// If some error occurs, then this

// block of code will run

if (!errors.isEmpty()) {

res.json(errors)

}

// If no error occurs, then this

// block of code will run

else {

res.send("Successfully validated")

}

});

app.listen(PORT, function (error) {

if (error) throw error

console.log("Server created Successfully on PORT ", PORT)

})

|  |
| --- |
| **Steps to run the program:**  The project structure should look something like this depending on where you have saved your file/folder:  project structure   1. Make sure you have a ‘**view engine**’. We have used “**ejs**” and also make sure that you have already installed express and express-validator, body-parser using following commands:   **npm install ejs**  **npm install express**  **npm install body-parser**  **npm install express-validator**   1. Run index.js file using below command:   **node index.js**  Output of node index.js   1. Open the browser and type this URL *http://localhost:3000/*, the fill this sample form with correct data as shown below:   Signup.ejs   1. Then submit the form and if no error occurs, then you will see the following output:   success output   1. If you try to submit the form with incorrect data, then you will see the error message as shown below:   error-message |