

Node Js

Introduction to Node.js

- The modern web application has really come a long way over the years with the introduction of many popular frameworks such as bootstrap, Angular JS, etc. All of these frameworks are based on the popular JavaScript framework.
- **Node.js** is also based on the JavaScript framework, but it is used for developing **server-based applications**.

What is Node.js?

- Node.js is an open-source, cross-platform, runtime environment used for the development of server-side web applications.
- Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.
- Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications.
- Over a long period of time, the framework available for web development were all based on a stateless model. A stateless model is where the data generated in one session (such as information about user settings and events that occurred) is not maintained for usage in the next session with that user.
- A lot of work had to be done to maintain the session information between requests for a user. But with Node.js, there is finally a way for web applications to have real-time two-way connections, where both the client and server can initiate communication, allowing them to exchange data freely.

Why use Node.js

- Over the years, most of the applications were based on a stateless request-response framework.
- In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.
- But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and either the client or server can start the communication.

Installation of the Node.js

- The first step in using Node.js is the installation of the Node.js libraries on the client system.

Step 1) Download Node.js Installer for Windows

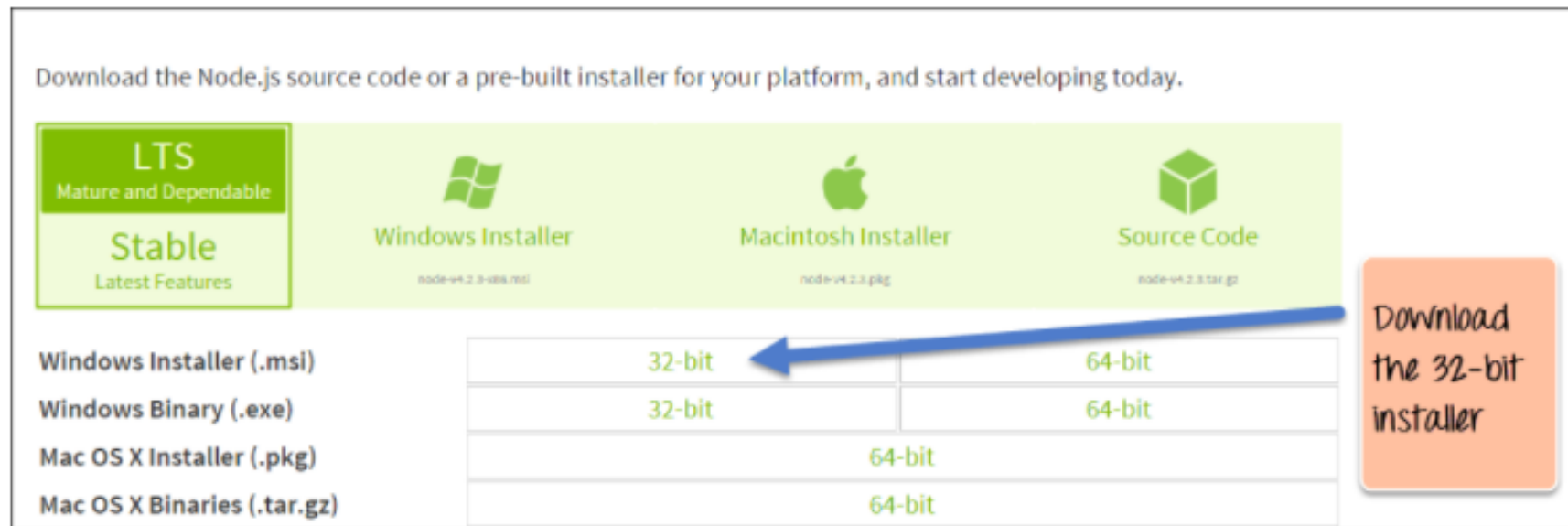
Go to the site <https://nodejs.org/en/download/> and download the necessary binary files.

In our example, we are going to Download Node.js on Windows with the 32-bit setup files.

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

	Windows Installer <small>node-v4.2.3-x86.msi</small>	Macintosh Installer <small>node-v4.2.3.pkg</small>	Source Code <small>node-v4.2.3.tar.gz</small>
Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.exe)	32-bit	64-bit	
Mac OS X Installer (.pkg)		64-bit	
Mac OS X Binaries (.tar.gz)		64-bit	

Download the 32-bit installer




Downloads


Latest LTS Version: **14.17.0** (includes npm 6.14.13)


Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS
Recommended For Most Users

Current
Latest Features


Windows Installer
node-v14.17.0-x86.msi


macOS Installer
node-v14.17.0.pkg


Source Code
node-v14.17.0.tar.gz

Windows Installer (.msi)

Windows Binary (.zip)

macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x64)

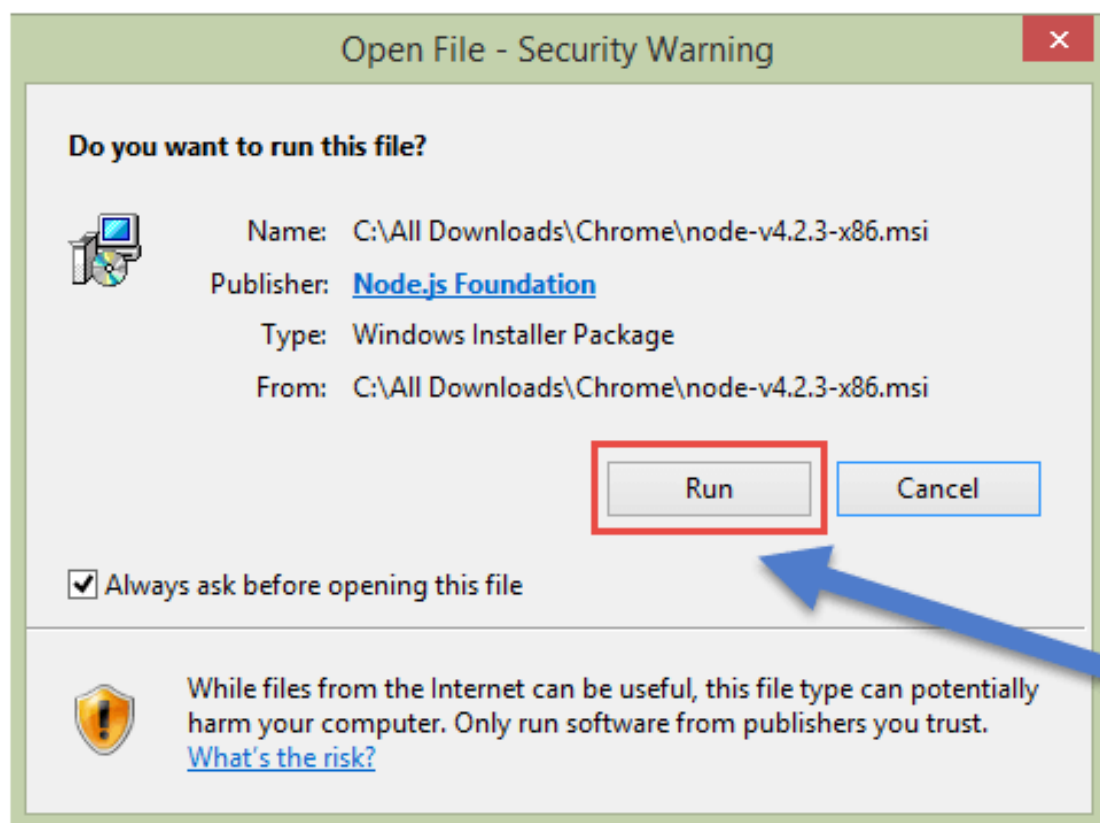
Linux Binaries (ARM)

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8

Step 2) Run the installation

Double click on the downloaded .msi file to start the installation.

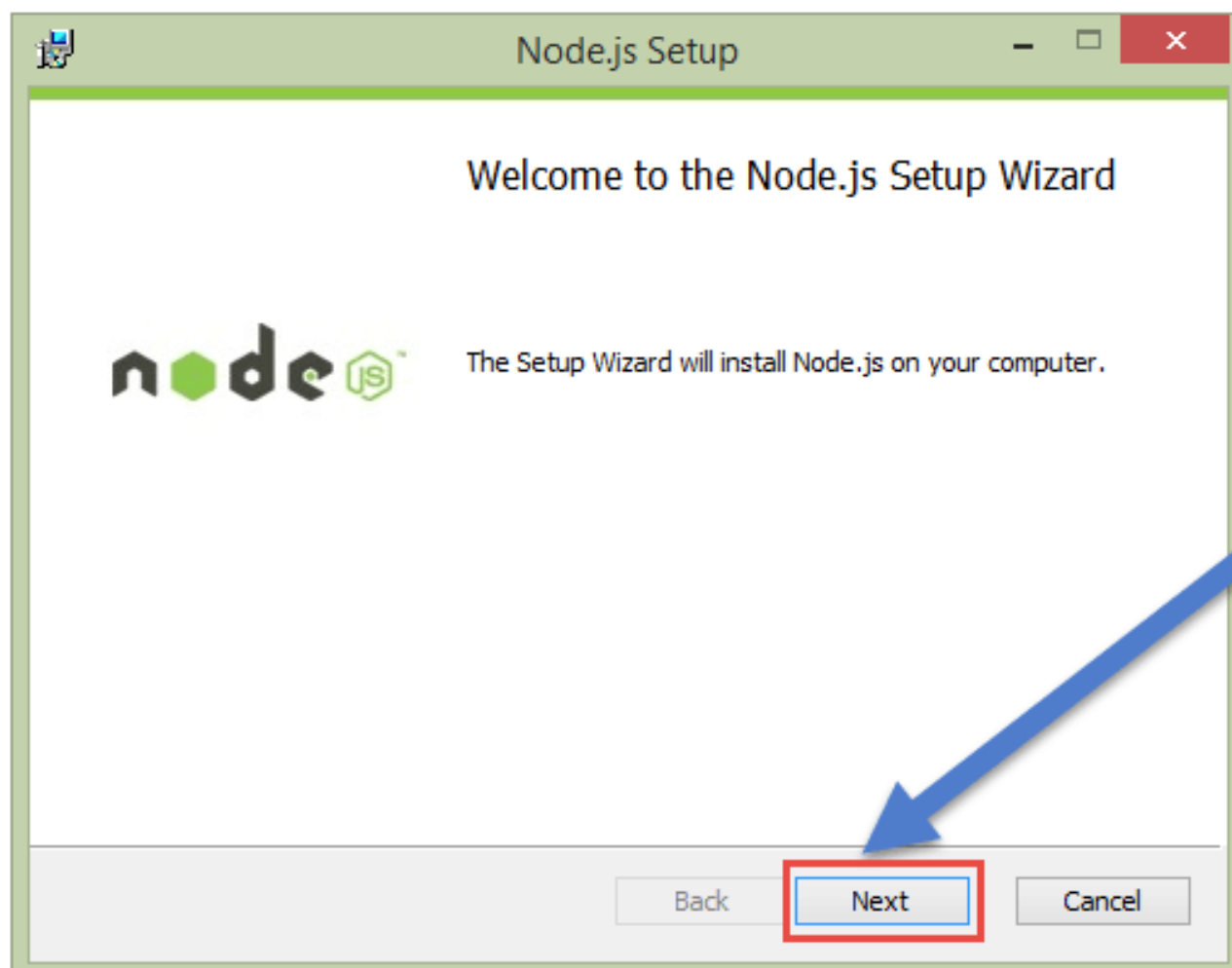
Click the Run button on the first screen to begin the installation.



Click the
Run button

Step 3) Continue with the installation steps

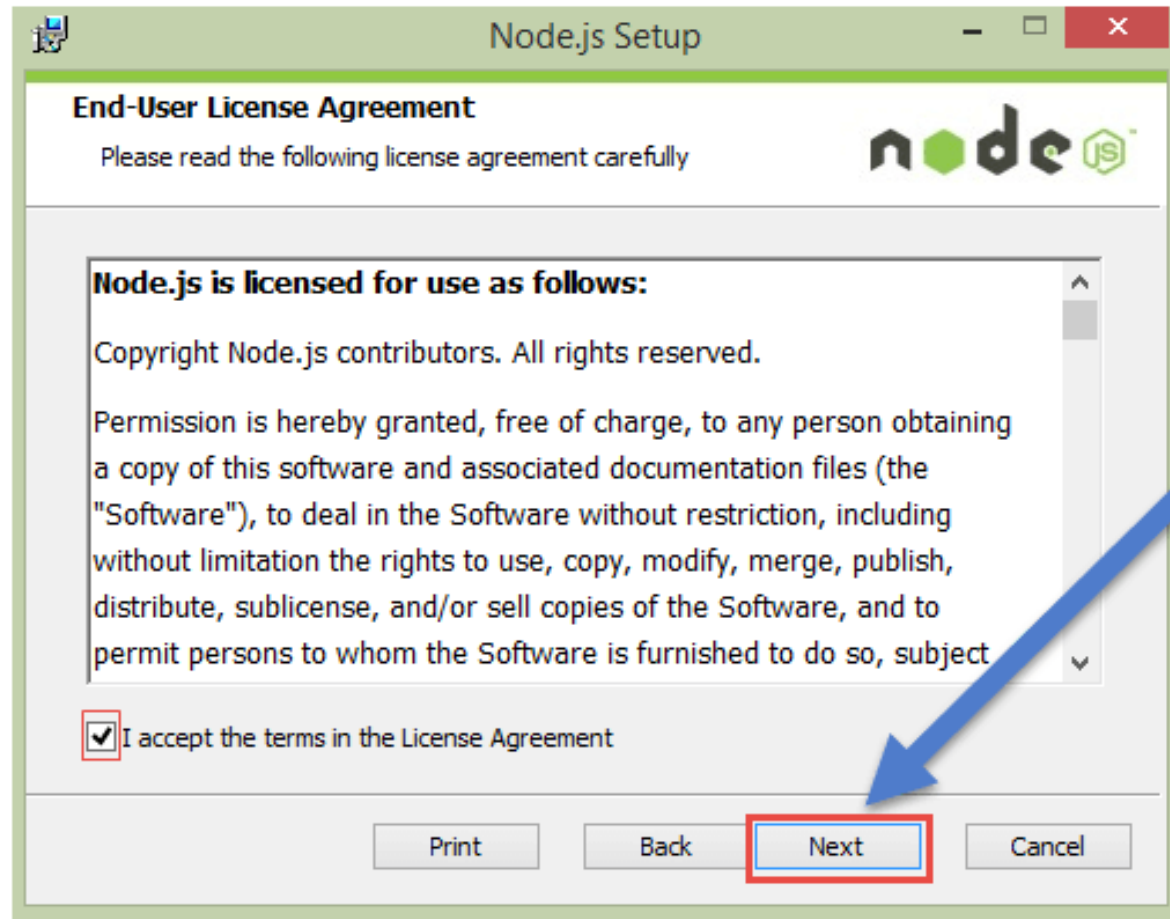
In the next screen, click the "Next" button to continue with the installation



Click the
Next button

Step 4) Accept the terms and conditions

In the next screen, Accept the license agreement and click on the Next button.

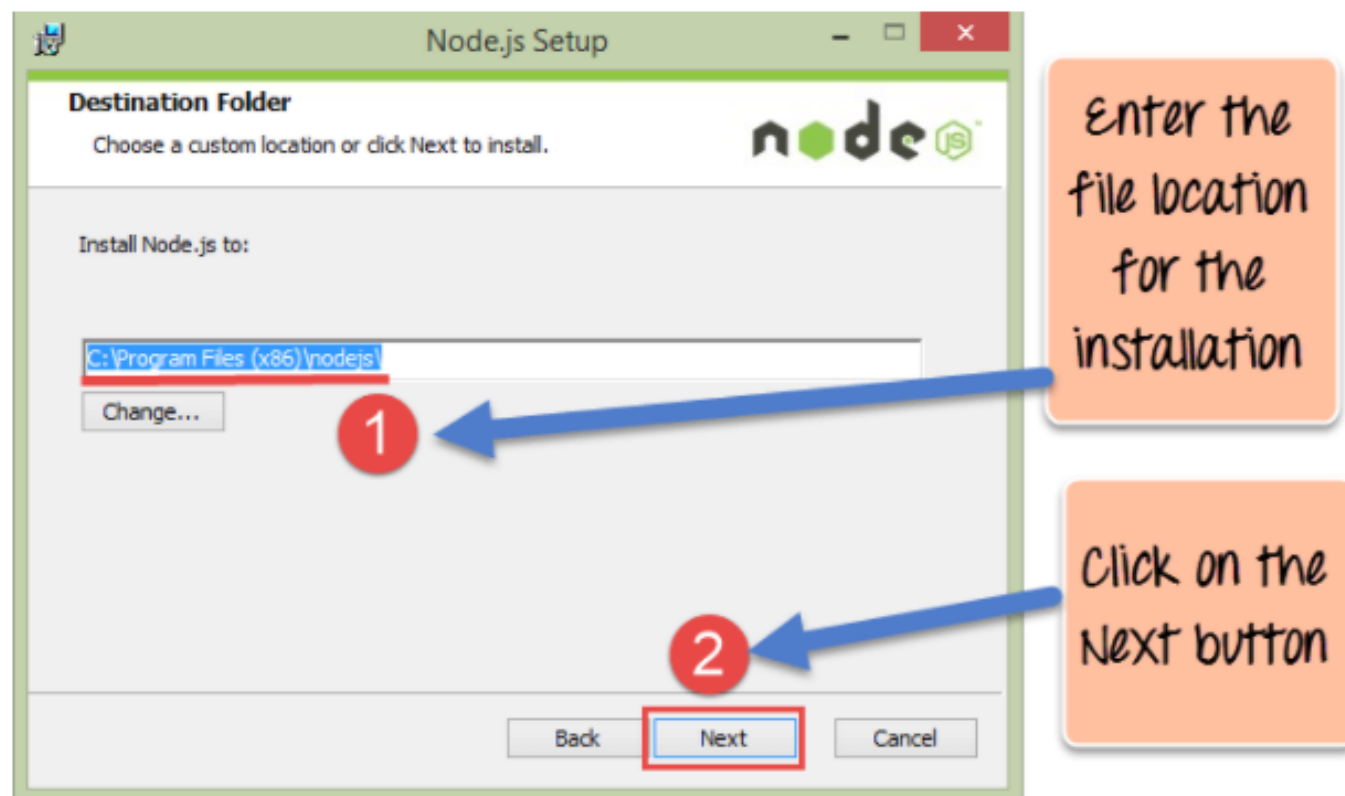


Accept the
license
agreement
and click
the Next

Step 5) Set up the path

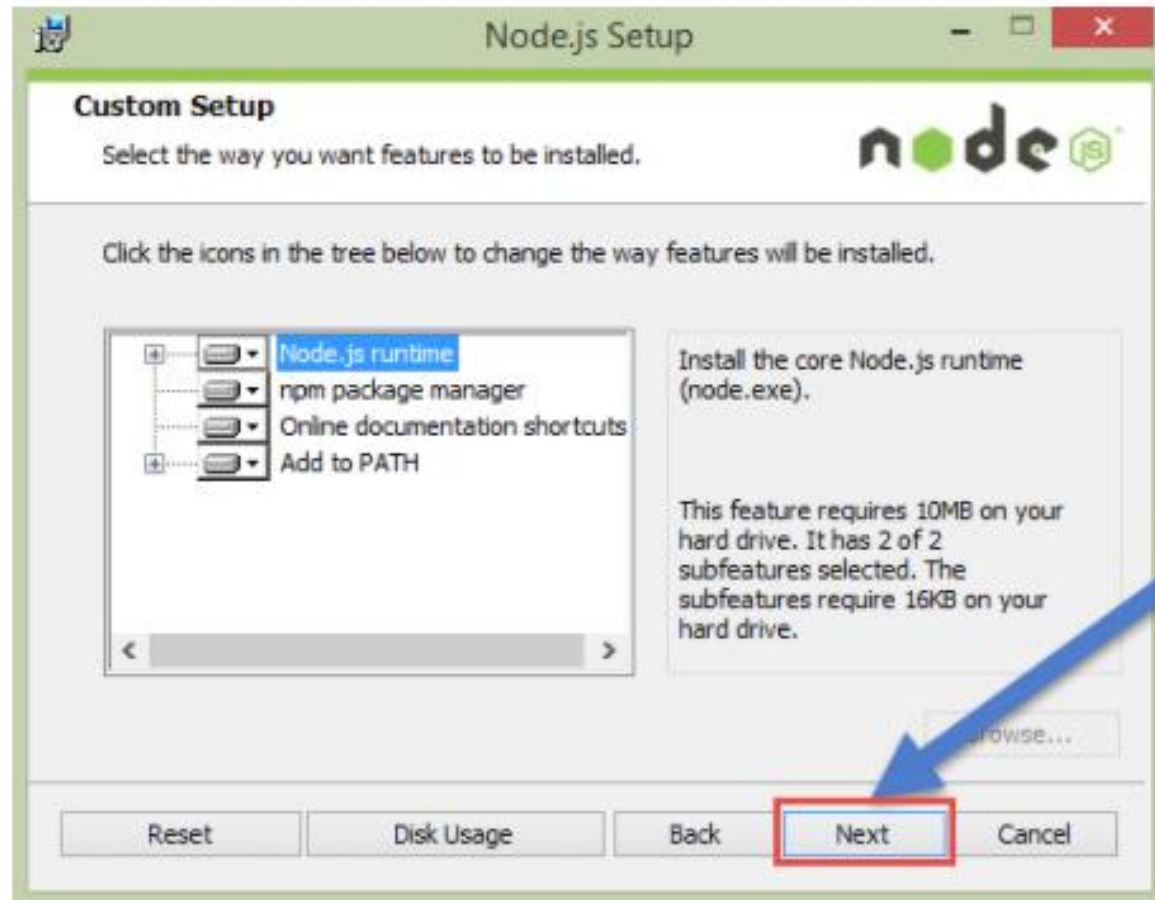
In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

1. First, enter the file location for the installation of Node.js. This is where the files for Node.js will be stored after the installation.
2. Click on the Next button to proceed ahead with the installation.



Step 6) Select the default components to be installed

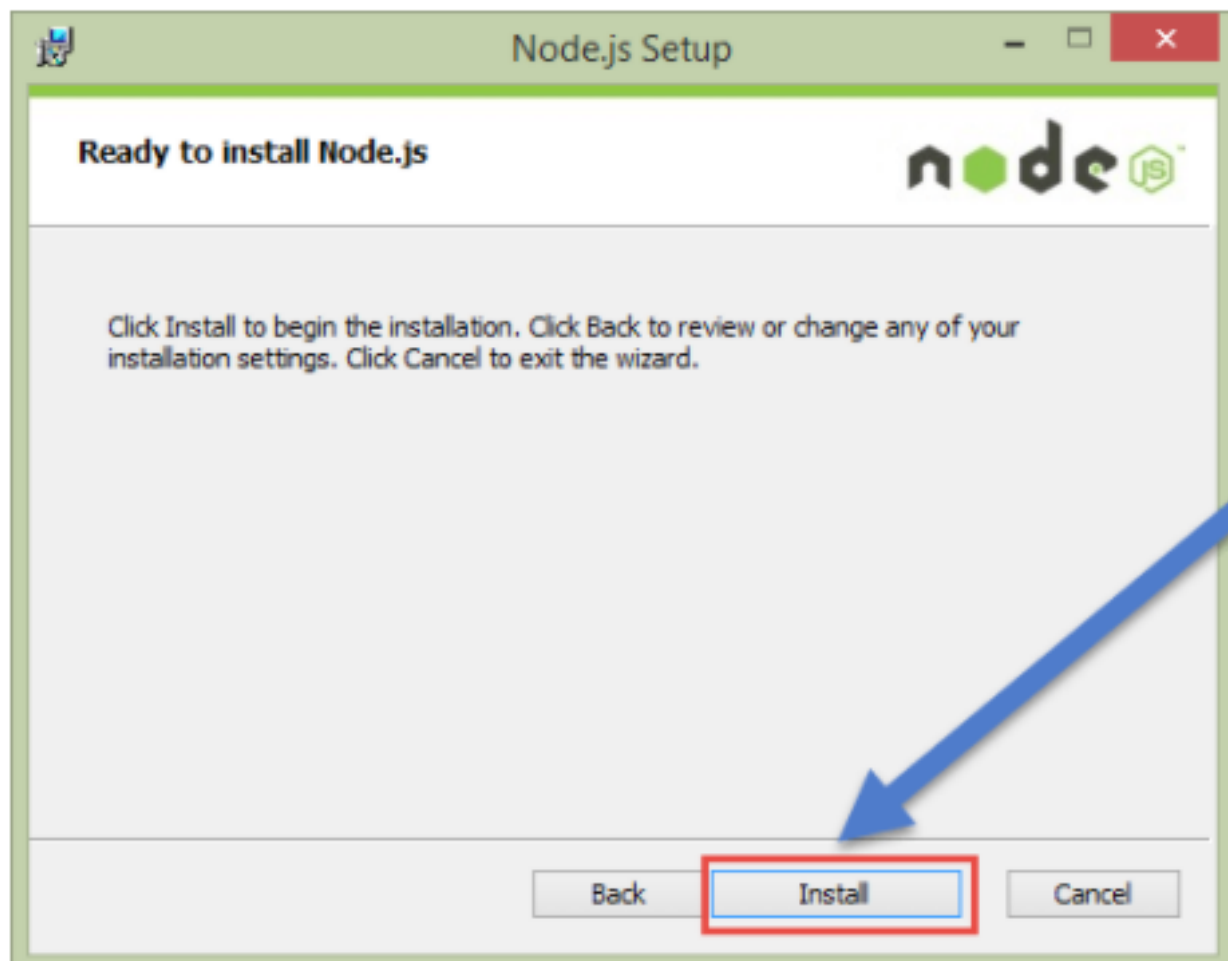
Accept the default components and click on the Next button.



Accept the
default
components
and click on
Next

Step 7) Start the installation

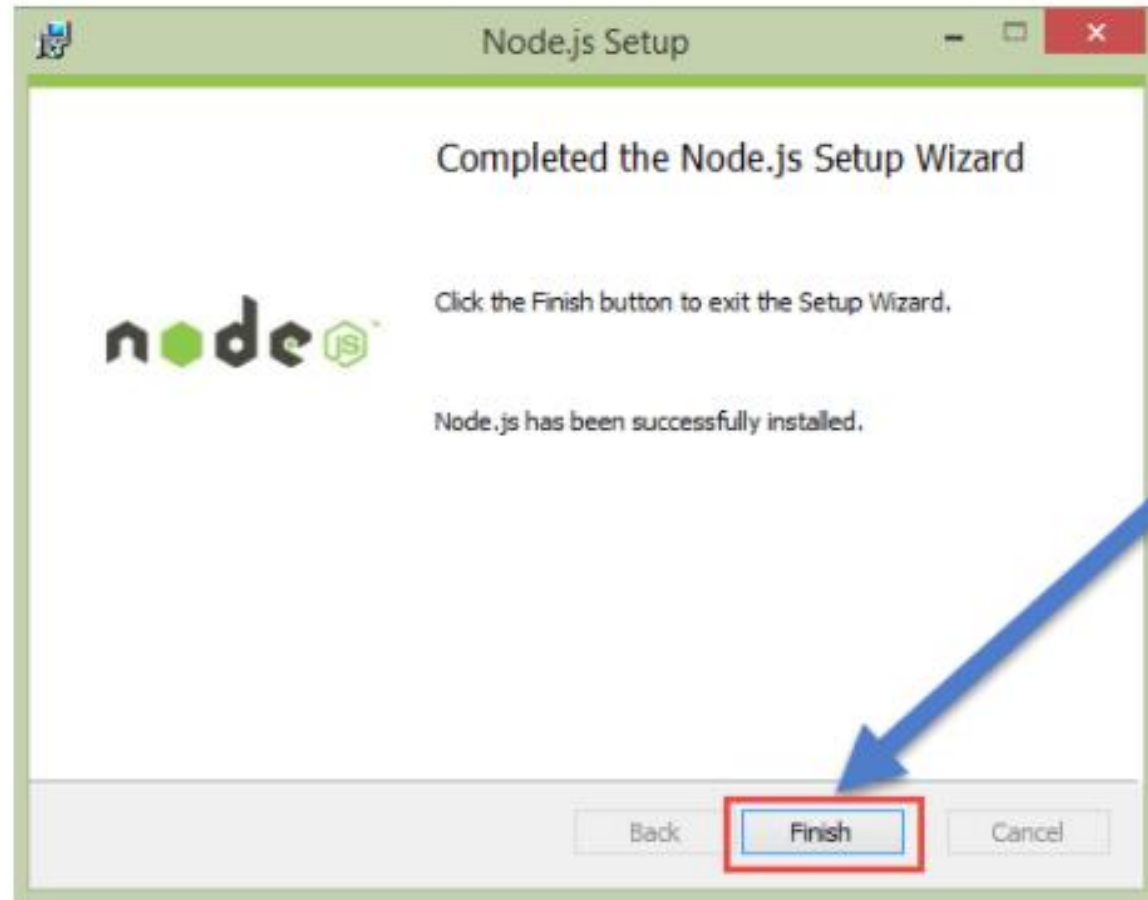
In the next screen, click the Install button to start installing Node.js on Windows.



Click the
Next button
to begin
the
installation

Step 8) Complete the installation

Click the Finish button to complete the installation.



Click the
Finish
button to
complete
the
installation

How to Install NPM on Windows 10/8/7

- **The other way to install Node.js** on any client machine is to use a "package manager."
- On Windows, **the NPM (Node Package Manager)** download is known as Chocolatey. It was designed to be a decentralized framework for quickly installing applications and tools that you need.

Running your first Hello World application in Node.js

- Create file Node.js with file name firstprogram.js

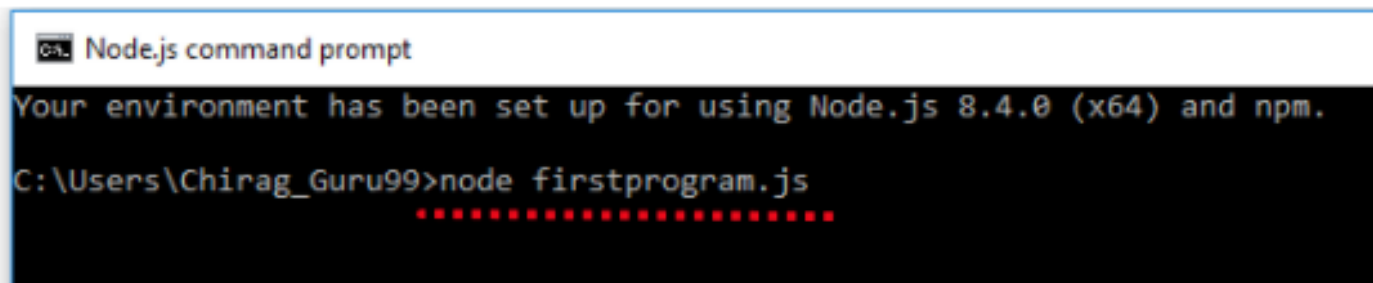
```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

- The basic functionality of the "require" function is that it reads a JavaScript file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of HTTP and we are using the **require(http)** command.
- In this 2nd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application.
- When a request is received, we are asking our function to return a "Hello World" response to the client. The **writeHead** function is used to send header data to the client, and while the end function will close the connection to the client.
- We are then using the **server.listen** function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

Executing the code

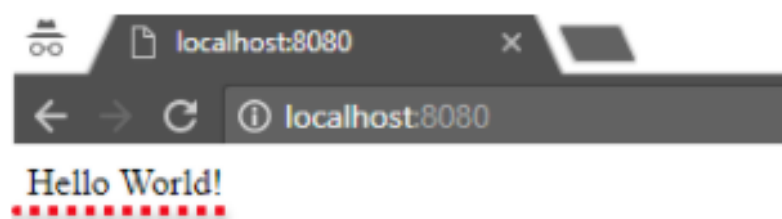
1. Save the file on your computer: C:\Users\Your Name\ firstprogram.js
2. In the command prompt, navigate to the folder where the file is stored. Enter the command Node firstprogram.js



```
Node.js command prompt
Your environment has been set up for using Node.js 8.4.0 (x64) and npm.
C:\Users\Chirag_Guru99>node firstprogram.js
.....
```

1. Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
2. Start your internet browser, and type in the address: <http://localhost:8080>

OutPut



Introduction

- Node.js is an open source server environment.
- Node.js is not a programming language itself. It is a platform which runs JavaScript on server side
- Node.js = Runtime Environment + JavaScript Library
- Its runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js has built-in libraries to handle web requests and responses so we don't need a separate web server or other dependencies.

How Node js work

- Node.js is the JavaScript runtime environment which is based on Google's V8 Engine i.e. with the help of Node.js we can run the JavaScript outside of the browser. Other things that you may or may not have read about Node.js is that it is single-threaded, based on event-driven architecture, and non-blocking based on the I/O model.
- 1) Node.js Architecture:
 - Node.js is made of Chrome V8 engine which is written in C++ and Libuv which is a multi-platform C library that provides support for asynchronous I/O based events on event loops and thread loops. An important thing that we need to remember is that, even though Node.js is made using the V8 engine and Libuv which are written in C or C++, we can still use Node.js in pure JavaScript.
- 2) Node.js Application:
 - A thread in simple terms is basically a set of programming instructions that can be run independently in a computer's processor and every process that we want to run has its own thread to run the programming instructions and the process can have more than one thread. But, the point to remember is, Node.js application runs only on a single thread and by that, It mean whether that Node.js application is being used by 5 users or 5 million users, it will only run on a single thread which makes the Node.js application blockable (which means that a single line of code can block the whole app because an only single thread is being used).
 - So, to keep the Node.js application running, asynchronous code must be used everywhere having callback functions because as we know that asynchronous code keeps on running in the background and the callback gets executed as soon as the promise gets resolved rather than synchronous code which blocks the whole application until it gets finished executing. But, we can still use synchronous code however at someplace in our application and that place is before our application enters Event-loop.
 - Event-loop is what allows Node.js applications to run non-blocking asynchronous I/O based operations i.e, all the asynchronous code is managed and executed within the event-loop and before that, we can use our synchronous code which is in this case known as Top-Level code. So, try to write synchronous code only for those operations which are executed only once at the start of our application and not every time, for Ex: reading some data from your computer memory which later can be requested by some user (many times) in asynchronous code.

How its work ?

- **Extremely fast:** Node.js is built on Google Chrome's V8 JavaScript Engine, its used for converting javascript code to machine code faster
- **I/O Non Blocking and Asynchronous :** All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a **notification mechanism of Events** of Node.js helps the server to **get a response** from the previous API call. It is also a reason that it is very fast.
- **Single threaded:** Node.js follows a single threaded model with **event looping**. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No buffering:** Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.



Difference between Javascript and node js

1. NodeJS :

- NodeJS is a cross-platform and opensource Javascript runtime environment that allows the javascript to be run on the server-side.
- Nodejs allows Javascript code to run outside the browser. Nodejs comes with a lot of modules and mostly used in web development.

2. JavaScript :

- Javascript is a Scripting language. It is mostly abbreviated as JS. It can be said that Javascript is the updated version of the ECMA script.
- Javascript is a high-level programming language that uses the concept of Oops but it is based on prototype inheritance.

Modules of nodejs

- In Node.js, Modules are the blocks of encapsulated code that communicates with an external application on the basis of their related functionality.
- Modules can be a single file or a collection of multiples files/folders.
- The reason programmers are heavily reliant on modules is because of their re-usability as well as the ability to break down a complex piece of code into manageable chunks.
- Modules are of three types:
 - Core Modules
 - local Modules
 - Third-party Modules

- 1) **Core Modules:** Node.js has many built-in modules that are part of the platform and comes with Node.js installation. These modules can be loaded into the program by using the require function.
- Syntax:

```
var module = require('module_name');
```
- The require() function will return a JavaScript type depending on what the particular module returns. The following example demonstrates how to use the Node.js **http** module to create a web server.

```
var http = require('http');  
http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.write('Welcome to this page!');  
  res.end();  
}).listen(3000);
```

- The `require()` function returns an object because the `Http` module **returns its functionality as an object**.
- The function `http.createServer()` method will be executed when someone tries to access the computer on port 3000.
- The `res.writeHead()` method is the status code where 200 means it is OK, while the second argument is an object containing the response headers.

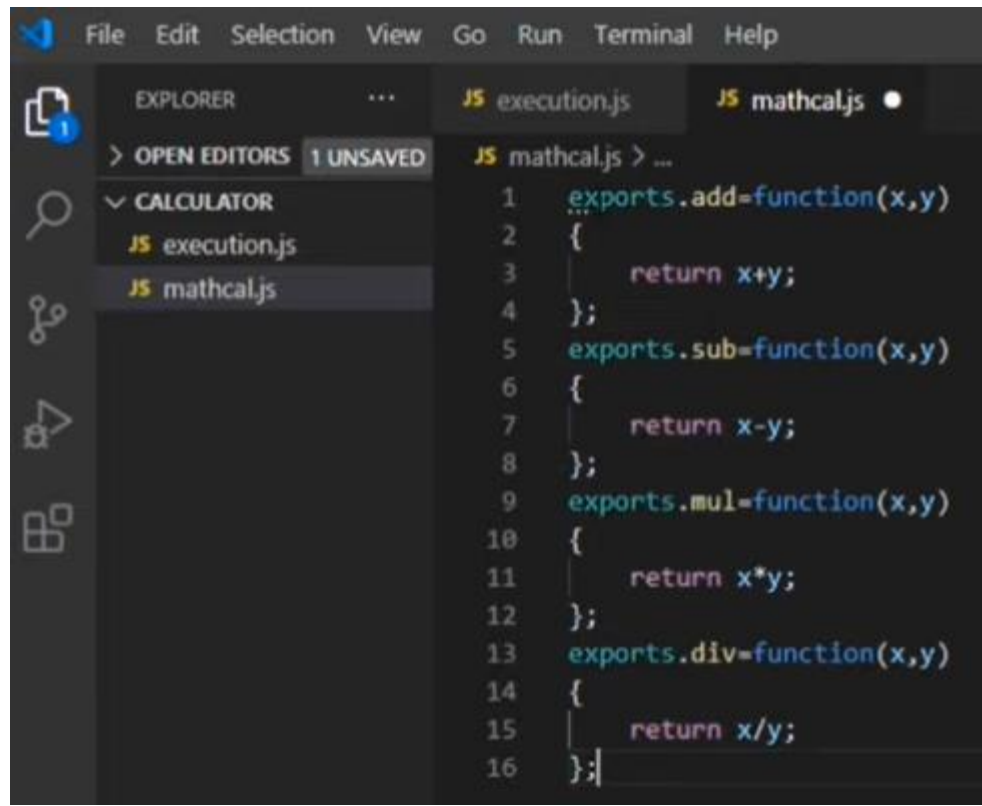
2) Local Modules: Unlike built-in and external modules, local modules are created locally in your Node.js application. For example to create a simple calculating module that calculates various operations. Create a calc.js file that has the following code: Since calc.js file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

Calc.js

```
exports.add = function (x, y) {  
    return x + y;  
};  
  
exports.sub = function (x, y) {  
    return x - y;  
};  
  
exports.mult = function (x, y) {  
    return x * y;  
};  
  
exports.div = function (x, y) {  
    return x / y;  
};
```

Index.js

```
var calculator = require('./calc');  
  
var x = 50, y = 20;  
  
console.log("Addition of 50 and 10 is "  
            + calculator.add(x, y));  
  
console.log("Subtraction of 50 and 10 is "  
            + calculator.sub(x, y));  
  
console.log("Multiplication of 50 and 10 is "  
            + calculator.mult(x, y));  
  
console.log("Division of 50 and 10 is "  
            + calculator.div(x, y));
```



File Edit Selection View Go Run Terminal Help

EXPLORER

> OPEN EDITORS 1 UNSAVED

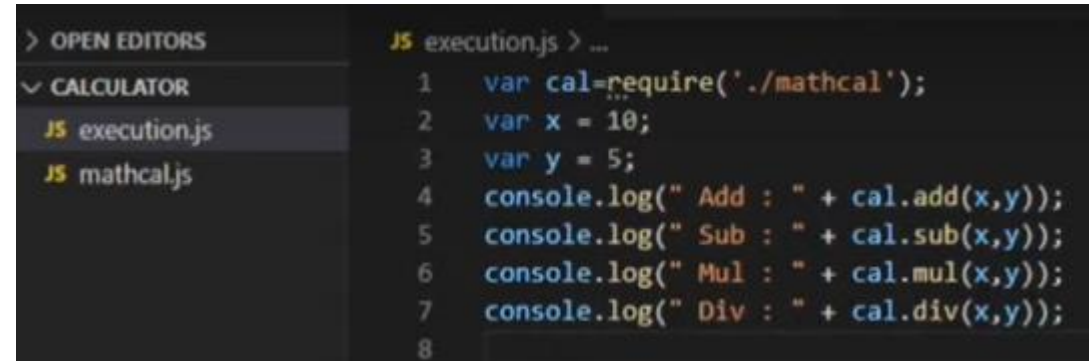
✓ CALCULATOR

JS execution.js

JS mathcal.js

JS mathcal.js > ...

```
1 exports.add=function(x,y)
2 {
3     return x+y;
4 };
5 exports.sub=function(x,y)
6 {
7     return x-y;
8 };
9 exports.mul=function(x,y)
10 {
11     return x*y;
12 };
13 exports.div=function(x,y)
14 {
15     return x/y;
16 };
```



> OPEN EDITORS

✓ CALCULATOR

JS execution.js

JS mathcal.js

JS execution.js > ...

```
1 var cal=require('./mathcal');
2 var x = 10;
3 var y = 5;
4 console.log(" Add : " + cal.add(x,y));
5 console.log(" Sub : " + cal.sub(x,y));
6 console.log(" Mul : " + cal.mul(x,y));
7 console.log(" Div : " + cal.div(x,y));
8
```

3)Third-party modules: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

- Example:
 - `npm install express`
 - `npm install mongoose`

Node Package Manager (npm)

- NPM (Node Package Manager) is the default package manager for Node.js and is written entirely in Javascript. Developed by Isaac Z. Schlueter, it was initially released in January 12, 2010. NPM manages all the packages and modules for Node.js and consists of command line client npm. It gets installed into the system with installation of Node.js. The required packages and modules in Node project are installed using NPM.
- **A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.**
- NPM can install all the dependencies of a project through the package.json file. It can also update and uninstall packages. In the package.json file, each dependency can specify a range of valid versions using the semantic versioning scheme, allowing developers to auto-update their packages while at the same time avoiding unwanted breaking changes.

Nodejs Libraries

- **Body-parser** : Node.js body parsing middleware. Parse incoming request bodies in a middleware before your handlers, available under the req.body property.
- **Mongoose** : It is a Mongodb object modelling tool designed to work in an asynchronous environment. It supports callbacks.
- **Crypto-js** : JavaScript library of crypto standards.
- **NodeMailer** : is a module for Nodejs application that allows you to send emails from Nodejs.
- **Async** : It is a utility module that allows you to work with asynchronous javascript.
- **Moment** : is a lightweight javascript date library for manipulating, formatting and parsing date.
- **Express** : It is flexible and it is a framework for Node js

Benefits

1)Easy to Learn

Node.js has **no steep learning curve**. Coding in Node.js is relatively easy to grasp, once you have mastered JavaScript and Object Oriented Programming basics.

2)Keeping things simple

Applications written in Node.js require fewer files and less code compared to those with different languages for front-end and back-end. You can also reuse and share the code between the front-end and back-end parts of your application, which speed up the development process. **One code, one deployment, everything in one place.**

3)Scalability

Scalability is baked into the core of Node.js. It is one of Node's basic benefits for startups who are planning to grow with the course of time. App-based startups choose it to develop lightweight and fast systems with good real-time response that can be scaled up later, as well as to easily add more modules to the existing ones. Node's scalability is achieved by the *load balancing* and the capability to **handle a huge number of concurrent connections.**

4) MVP

Node.js enables to quickly develop an MVP (minimum viable product) – a piece of software with just enough features (or a single killer feature) so that the product can go to the market and satisfy the first customers. MVP is a rudimentary stage on the way to a full-fledged application.

5) Community

An active community means a lot of support and feedback. Again, thriving Node.js community can help you avoid reinventing a bicycle – they have produced multiple tools and decent instruments that accelerate the development speed.

Middleware

- Middleware functions are functions that have access to the request object (req), the response object (res), and the next middleware function in the application's request-response cycle.
- The next middleware function is commonly denoted by a variable named next.
 - As name suggests it comes in middle of something and that is request and response cycle
 - Middleware has access to request and response object
 - Middleware has access to next function of request-response life cycle



- Middleware functions can perform the following tasks:
 - Execute any code.
 - Make changes to the request and the response objects.
 - End the request-response cycle.
 - Call the next middleware in the stack.
- If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function.
- Otherwise, the request will be left hanging.

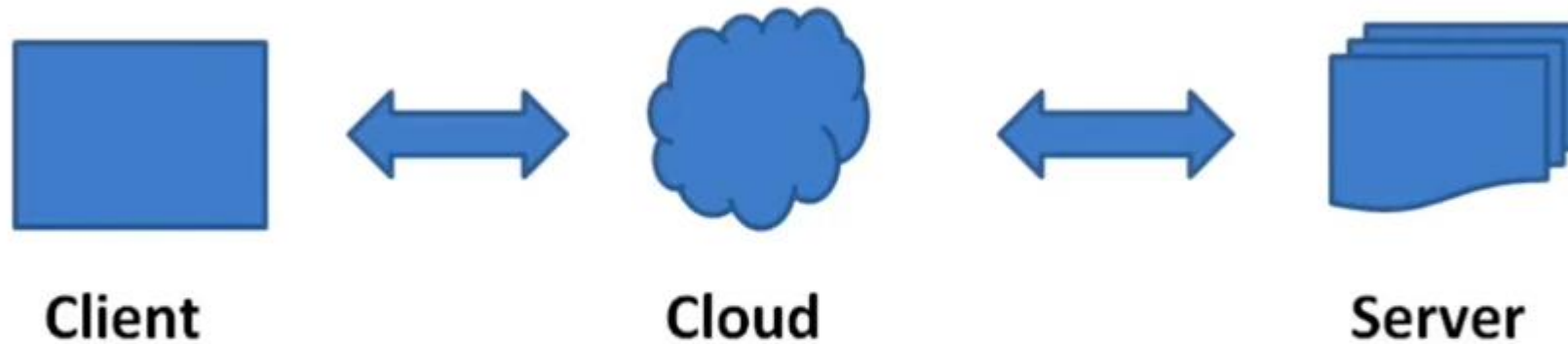
- Application level middleware : `app.use`
- Router level middleware : `router.use`
- Built-in middleware : `express.static`, `express.json`, `express.urlencoded`
- Error handling middleware : `app.use(err, req, res, next)`
- Thirdparty middleware : `bodyparser`, `cookieparser`

Node JS Built-in Modules

- http - To make Node.js act as an HTTP server
- https - To make Node.js act as an HTTPS server.
- fs - To handle the file system
- events - To handle events
- path - To handle file paths
- url - To parse URL strings

How HTTP Works

- **HTTP** stands for **H**yper **T**ext **T**ransfer **P**rotocol
- **WWW** is about communication between web **clients** and **servers**
- Its done by sending **HTTP Requests** and receiving **HTTP Responses**



- A client (a browser) sends an **HTTP request** to the web
- An web server receives the request
- The server runs an application to process the request
- The server returns an **HTTP response** (output) to the browser
- The client (the browser) receives the response.



HTTP MODULE

- **require**

We use the **require** directive to load Node JS modules.

- **Create Server**

HTTP module is a built-in module of node js, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

The HTTP module use the `createServer()` method to create an HTTP server.

- **HTTP Header**

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type.

The first argument of the `res.writeHead()` method is the status code, 200 means that all is OK, the second argument is an object containing the response headers.

- **Port Number**

HTTP server that listens to server ports and gives a response back to the client.

The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 8080.

JS app.js X JS sample.js JS sample1.js X

D: > mekala_f > fall21-22 > Programs > nodejs > JS sample1.js > http.createServer() callback > 'content-type'

```
1 var http=require('http');
2 http.createServer(function(req,res){
3     res.writeHead(200,{ 'content-type': 'text/plain'});
4     res.write("Welcome to VIT");
5     res.end("Thank You");
6 }).listen(8080);
7
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
D:\mekala_f\fall21-22\Programs\nodejs\sample1.js:3
  res.writeHead(200,{ 'content-type':text/plain});
                        ^
```

ReferenceError: text is not defined

```
at Server.<anonymous> (D:\mekala_f\fall21-22\Programs\nodejs\sample1.js:3:39)
at emitTwo (events.js:126:13)
at Server.emit (events.js:214:7)
at parserOnIncoming (_http_server.js:634:12)
at HTTPParser.parserOnHeadersComplete (_http_common.js:117:17)
```

PS D:\mekala_f\fall21-22\Programs\nodejs> node sample1.js

← → ↻ ⓘ localhost:8080

Welcome to VITThank You

To verify whether node installed

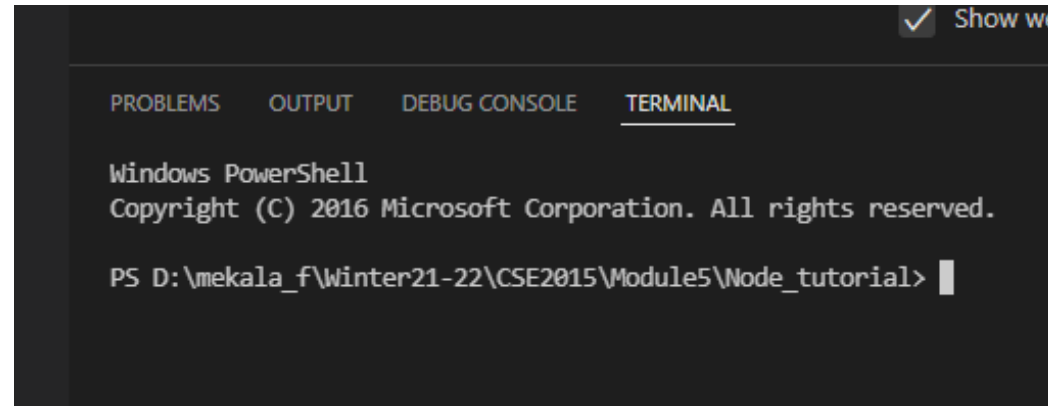
Command Prompt

```
Microsoft Windows [Version 10.0.15063]  
(c) 2017 Microsoft Corporation. All rights reserved.
```

```
C:\Users\Admin>node -v  
v8.12.0
```

```
C:\Users\Admin>
```

- Create new folder Node_tutorial
- Open Vscode
 - File->Open Folder->select Node_tutorial
 - Terminal->new Terminal

A screenshot of a Windows PowerShell terminal window within the Visual Studio Code editor. The terminal title bar shows a checkmark icon and the text "Show we". The terminal interface includes tabs for "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", and "TERMINAL", with "TERMINAL" being the active tab. The terminal content displays the Windows PowerShell prompt, the copyright notice "Copyright (C) 2016 Microsoft Corporation. All rights reserved.", and the current directory path "PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial>" followed by a cursor.

- Enter npm init
 - Press enter key..... Finally it asks the question **Is this OK? (yes)**
 - Type **Y** . A package.json file will be created.



EXPLORER



Get Started X



OPEN EDITORS

X Get Started



NODE_TUTORIAL

{ } package.json



> OUTLINE

Start

Walkthroughs

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

and exactly what they do.

Use ``npm install <pkg>`` afterwards to install a package and save it as a dependency in the package.json file.

Press ^C at any time to quit.

package name: (node_tutorial)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository:

keywords:

author:

license: (ISC)

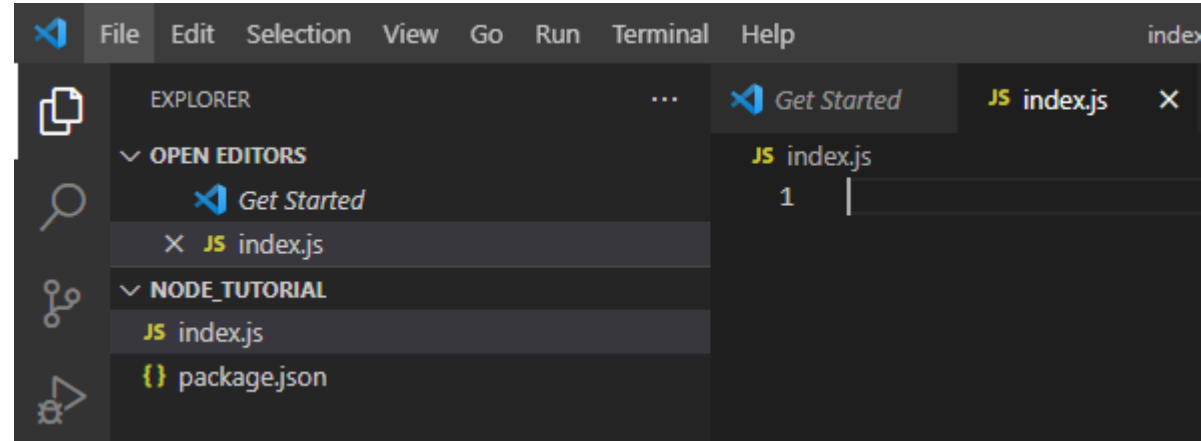
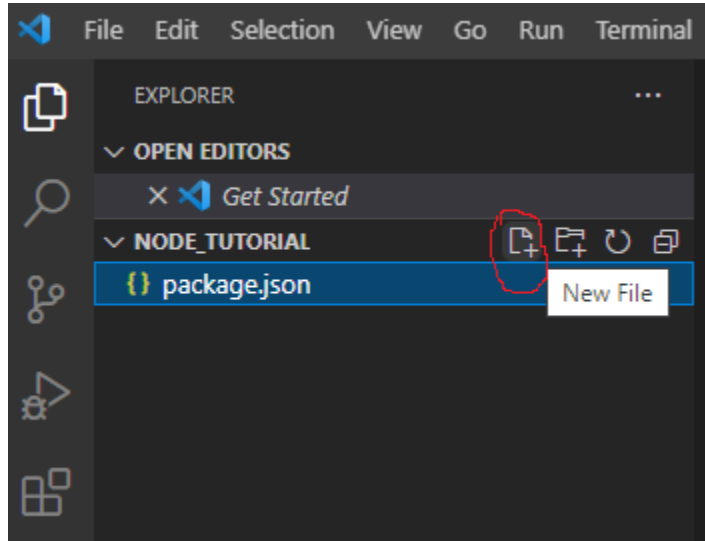
About to write to D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial\package.json:

```
{
  "name": "node_tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

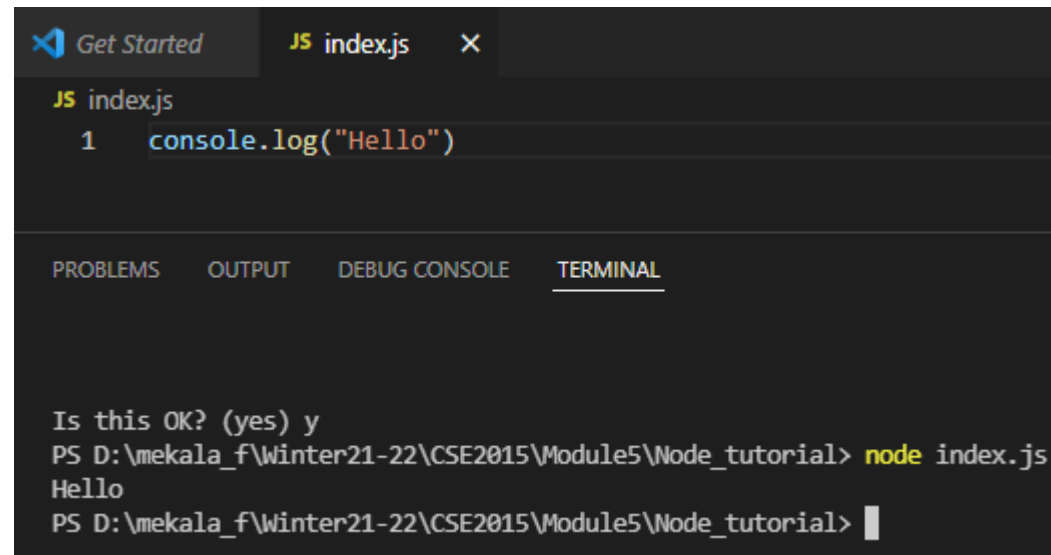
Is this OK? (yes) y

PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial> |

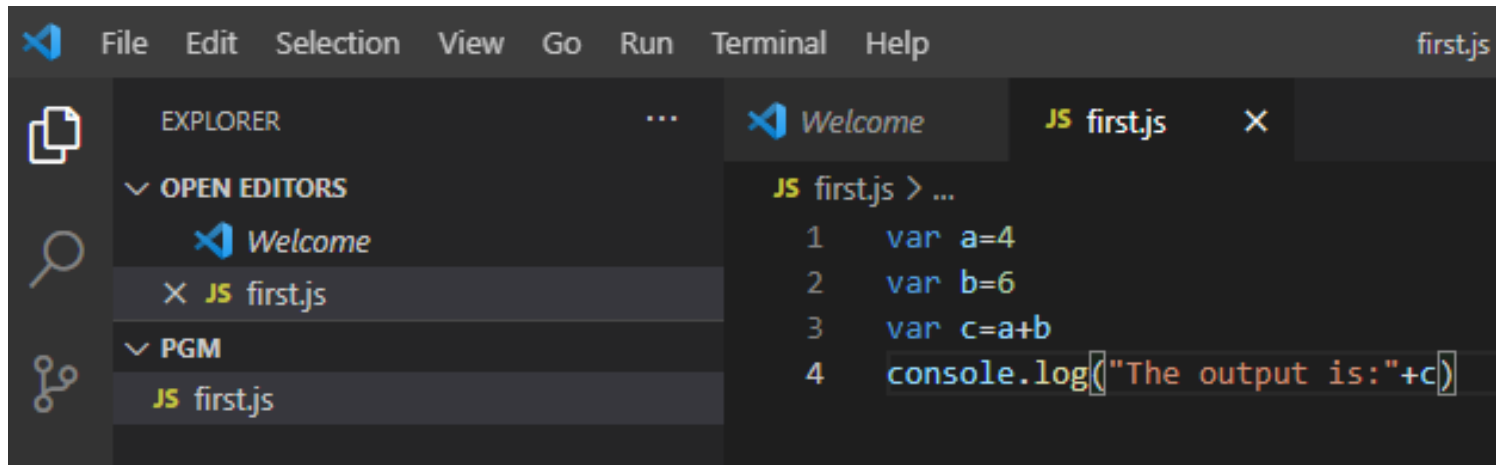
- Click on New File and type index.js



- To execute the program
- Syntax: **node filename.js**



Simple javascript code



```
File Edit Selection View Go Run Terminal Help first.js -
```

EXPLORER

OPEN EDITORS

- Welcome
- JS first.js

PGM

- JS first.js

```
JS first.js > ...
1  var a=4
2  var b=6
3  var c=a+b
4  console.log("The output is:"+c)
```

```
PS D:\mekala_f\fall21-22\CSE3002\ppt\Nodejs\pgm> node first.js
The output is:10
PS D:\mekala_f\fall21-22\CSE3002\ppt\Nodejs\pgm> 
```

Function calling in javascript

- Instead of calling like this

```
function myFunc()  
{  
    console.log("Function Working");  
};  
  
myFunc();
```

- We can call a function like this too.

```
(function myFunc()  
{  
    console.log("Function Working");  
})();
```

File Edit Selection View Go Run Terminal Help one.js - Node_tutorial - Visual Studio Code

EXPLORER

OPEN EDITORS

- Get Started
- JS index.js
- JS one.js

NODE_TUTORIAL

- index.js
- one.js
- package.json

```
JS one.js > myfun2
1 function myfun1()
2 {
3     console.log("Function Working")
4 }
5 function myfun2()
6 {
7     console.log("Function Working")
8 }
9 x=1234
10 y=567
11 //If i need to export more than one function and variable
12 module.exports.myfun1=myfun1;
13 module.exports.myfun2=myfun2;
14 module.exports.x=x;
15 module.exports.y=y;
```

File Edit Selection View Go Run Terminal Help index.js - Node_tutorial - Visual Studio Code

EXPLORER

OPEN EDITORS

- Get Started
- JS index.js
- JS one.js

NODE_TUTORIAL

- index.js
- one.js
- package.json

```
JS index.js > ...
1 //calling local module from one.js
2 const global = require('./one')
3 global.myfun1()
4 global.myfun2()
5 console.log(global.x)
6 console.log(global.y)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial> node index.js
Function Working
Function Working
1234
567
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial>
```

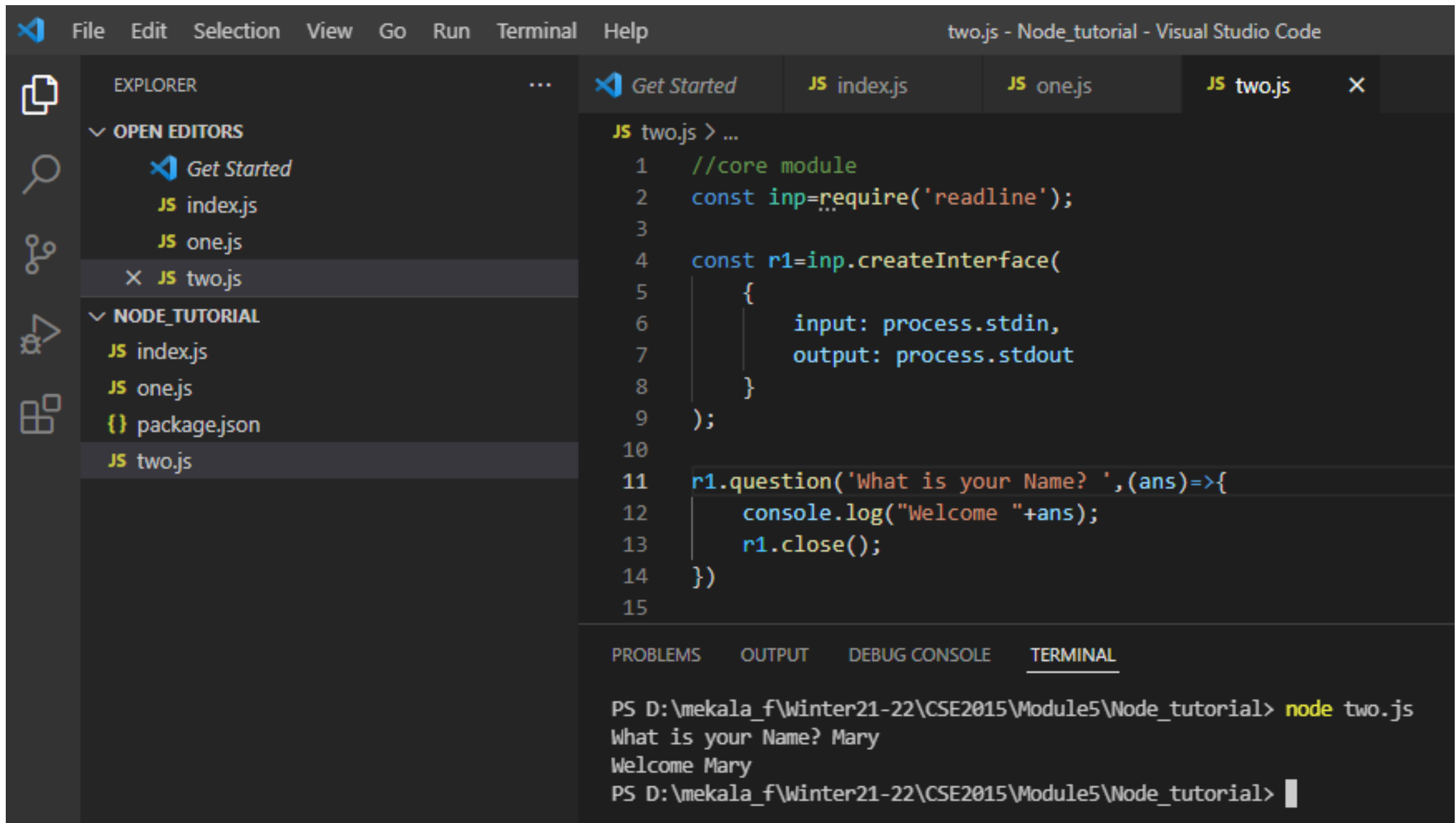
Core module

- **Readline** :The readline module provides an interface for reading data from a Readable stream (such as process.stdin) one line at a time.
- **http**:To make HTTP requests in Node.js, there is a built-in module HTTP in Node.js to transfer data over the HTTP. To use the HTTP server in node, we need to require the HTTP module. The HTTP module creates an HTTP server that listens to server ports and gives a response back to the client.

Syntax: **var http = require('http');**

- We can create a HTTP server with the help of **http.createServer()** method.

Core Module



two.js - Node_tutorial - Visual Studio Code

EXPLORER

OPEN EDITORS

- Get Started
- JS index.js
- JS one.js
- JS two.js

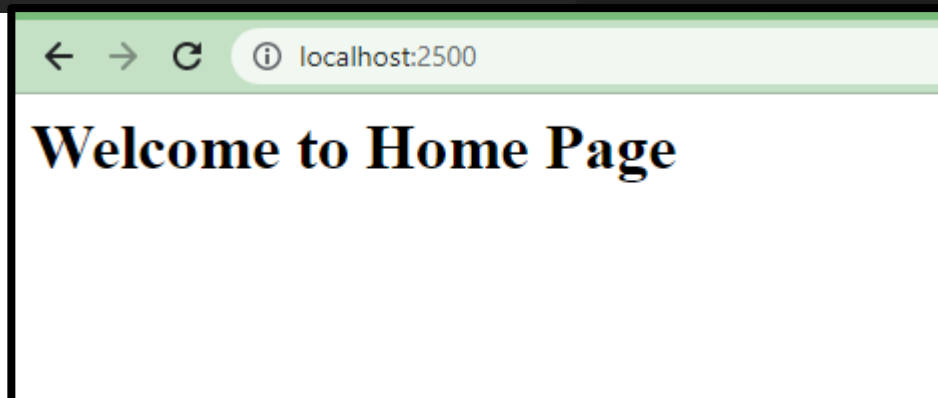
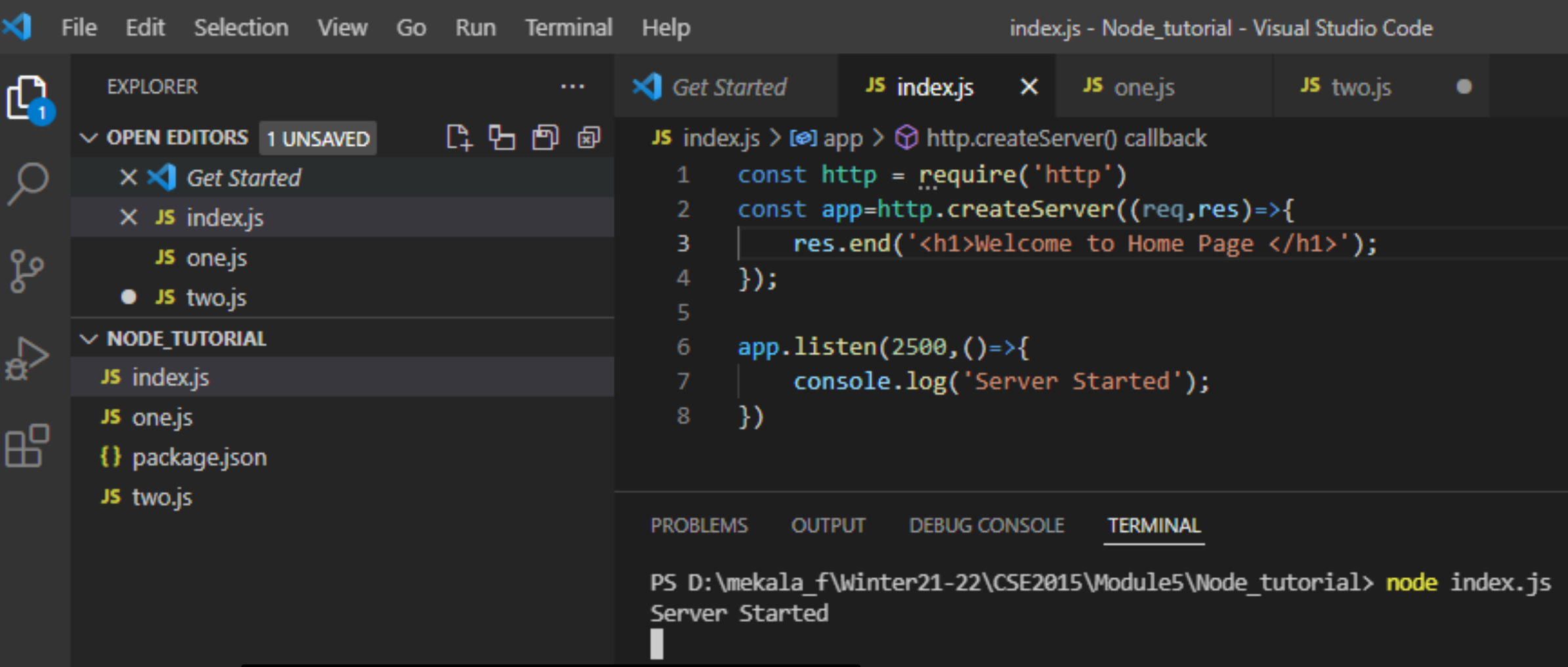
NODE_TUTORIAL

- JS index.js
- JS one.js
- package.json
- JS two.js

```
JS two.js > ...
1  //core module
2  const inp=require('readline');
3
4  const r1=inp.createInterface(
5    {
6      input: process.stdin,
7      output: process.stdout
8    }
9  );
10
11 r1.question('What is your Name? ',(ans)=>{
12   console.log("Welcome "+ans);
13   r1.close();
14 })
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial> node two.js
What is your Name? Mary
Welcome Mary
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial>
```



- Swap **nodemon** instead of **node** to run your code, and now your process will automatically restart when your code changes
- Syntax:

npm i --save-dev nodemon

- **Now click on package.json file , you can see the nodemon dependencies**



EXPLORER



JS index.js

{ } package.json X

JS one.js

JS two.js



OPEN EDITORS 2 UNSAVED

● JS index.js

X { } package.json

JS one.js

● JS two.js



NODE_TUTORIAL

> node_modules

JS index.js

JS one.js

{ } package-lock.json

{ } package.json

JS two.js



{ } package.json > ...

```
1 {
2   "name": "node_tutorial",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "",
10  "license": "ISC",
11  "devDependencies": {
12    "nodemon": "^2.0.15"
13  }
14 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Is this OK? (yes) yes

PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial> npm i --save-dev nodemon

added 116 packages, and audited 117 packages in 15s

16 packages are looking for funding

run `npm fund` for details

- Now remove **“test”** from **scripts** and type

```
"scripts": {  
  "nodemon": "nodemon index.js"  
},
```

YOU TYPE THE NAME OF THE JAVASCRIPT FILE YOU
CREATED FOR EXAMPLE I CREATE A INDEX.JS FILE
THATSWHY I TYPE nodemon index.js YOU TYPE HERE THE
NAME OF FILE YOU CREATED.
AND AFTER THAT TO RUN NODEMON
YOU CAN TYPE IN TERMINAL
npm run nodemon
AND THEN YOUR NODEMON WILL BE START

```
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Node_tutorial> npm run nodemon

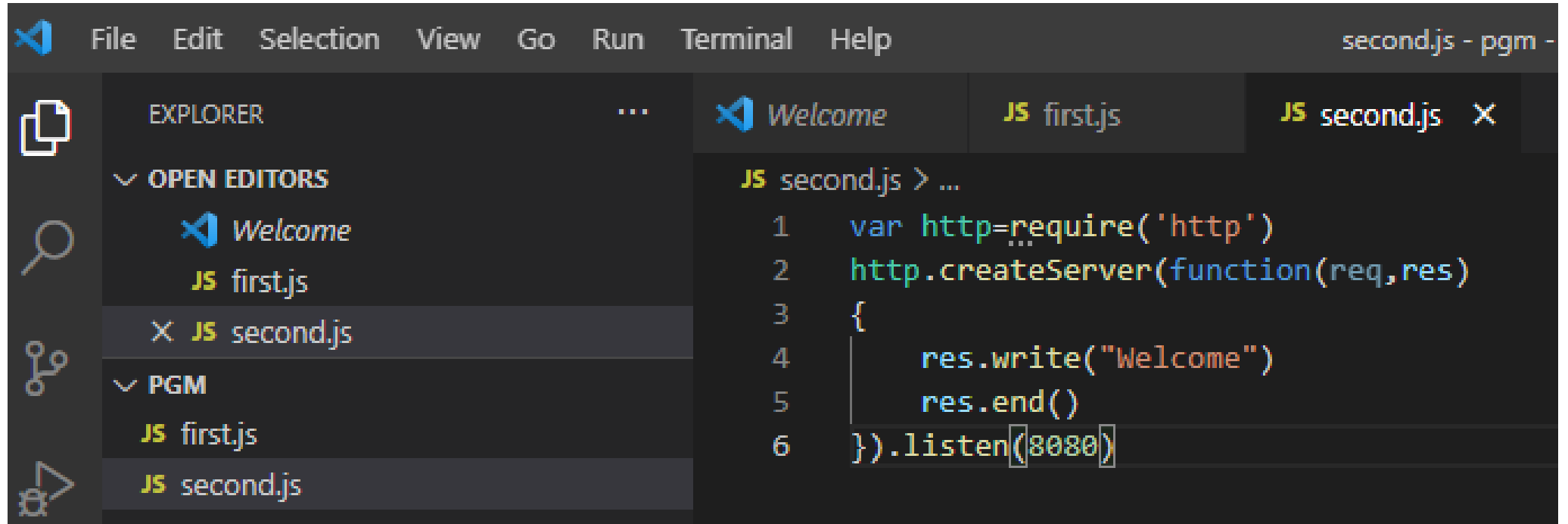
> node_tutorial@1.0.0 nodemon
> nodemon index.js

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server Started
_
```

- Now whatever changes I do in the code, just save it and refresh the browser window. Changes will be reflected.
- So no need to type **node index.js** in the terminal for every change in code.

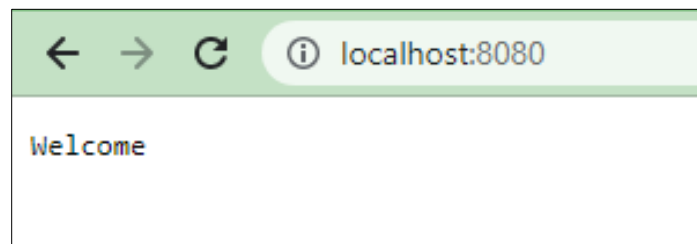
To create http service in nodejs

- To display response message from server

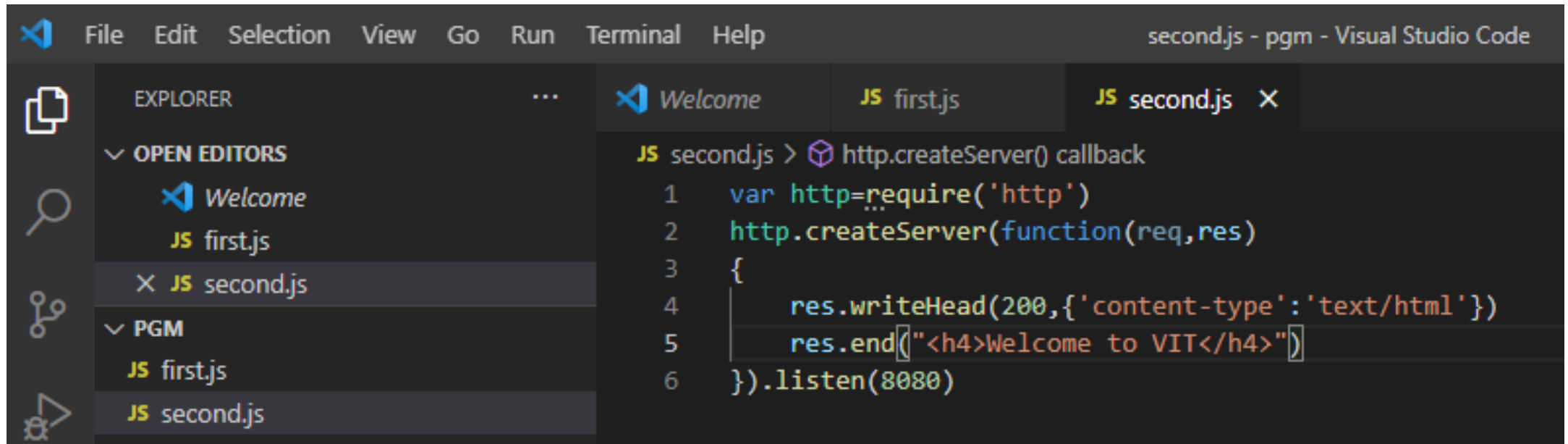


The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows the file structure with 'second.js' selected. The main editor area displays the code for 'second.js':

```
JS second.js > ...  
1  var http=require('http')  
2  http.createServer(function(req,res)  
3  {  
4      res.write("Welcome")  
5      res.end()  
6  }).listen(8080)
```



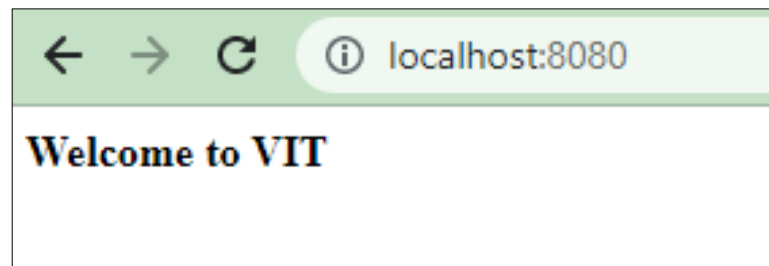
- It is better to add header also.



```
second.js - pgm - Visual Studio Code

EXPLORER
  OPEN EDITORS
    Welcome
    JS first.js
    JS second.js
  PGM
    JS first.js
    JS second.js

JS second.js > http.createServer() callback
1  var http=require('http')
2  http.createServer(function(req,res)
3  {
4      res.writeHead(200,{ 'content-type': 'text/html'})
5      res.end("<h4>Welcome to VIT</h4>")
6  }).listen(8080)
```



Express JS

- Express is a mature, flexible, lightweight server framework.
- It is designed for building single, multi-page, and hybrid web applications.

Express JS Features

- Express quickens the development pace of a web application.
- It also helps in creating mobile and web application of single-page, multi-page.
- Express can work with various templating engines such as Pug, Mustache, and EJS.
- Express follows the Model-View-Controller (MVC) architecture.
- It makes the integration process with databases such as MongoDB, Redis, MySQL effortless.
- Express also defines an error-handling middleware.
- It helps in simplifying the configuration and customization steps for the application.

Install Express JS

- Create new folder Express_tutorial
- Open Vscode
- File->Open Folder->select Express_tutorial
- Terminal->new Terminal
- Enter npm init
- Press enter key..... Finally it asks the question Is this OK? (yes)
- Type Y . A **package.json** file will be created.
- To install express framework in your project type the following command.

npm install express --save

- **package-lock.json** file will be created

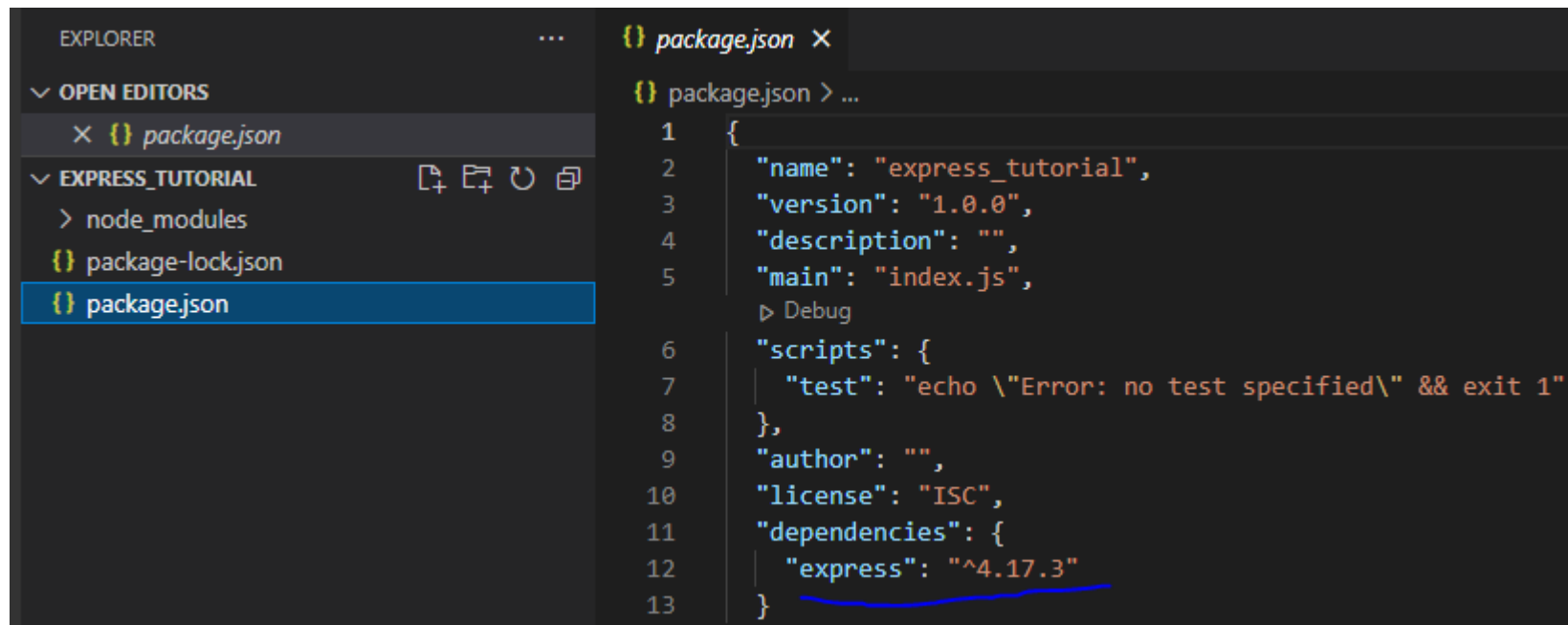
```
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Express_tutorial> npm install express --save

added 50 packages, and audited 51 packages in 7s

2 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Express_tutorial>
```

- After installing look into package.json file, you will find **express** dependencies



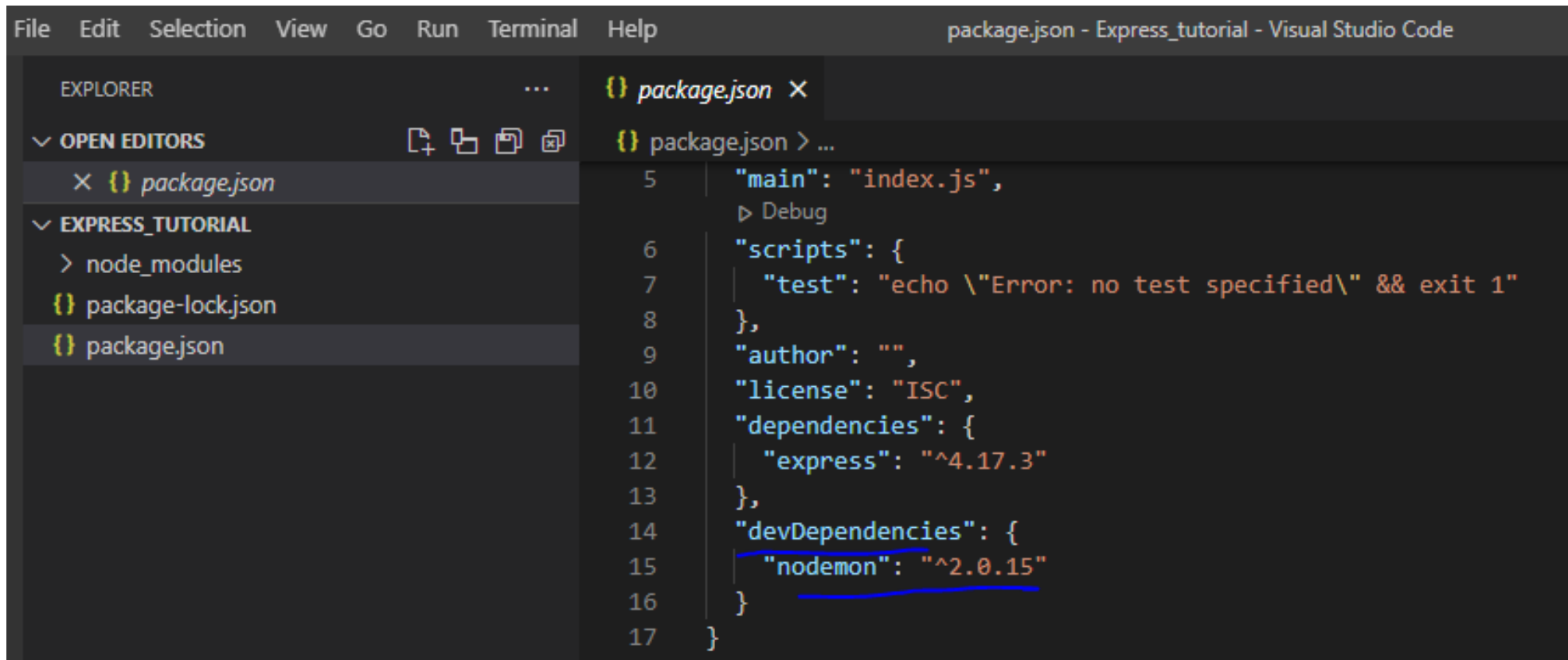
The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the file structure of a project named 'EXPRESS_TUTORIAL'. The 'package.json' file is selected and highlighted in blue. The main editor area shows the content of 'package.json', which is a JSON object with the following properties: 'name', 'version', 'description', 'main', 'scripts', 'author', 'license', and 'dependencies'. The 'dependencies' object contains a single entry for 'express' with the version '^4.17.3'. This entry is underlined in blue. The line numbers 1 through 13 are visible on the left side of the editor.

```
1 {  
2   "name": "express_tutorial",  
3   "version": "1.0.0",  
4   "description": "",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "author": "",  
10  "license": "ISC",  
11  "dependencies": {  
12    "express": "^4.17.3"  
13  }  
}
```

- Swap **nodemon** instead of **node** to run your code, and now your process will automatically restart when your code changes
- Syntax:

npm i --save-dev nodemon

- Now click on **package.json** file , you can see the **nodemon** dependencies

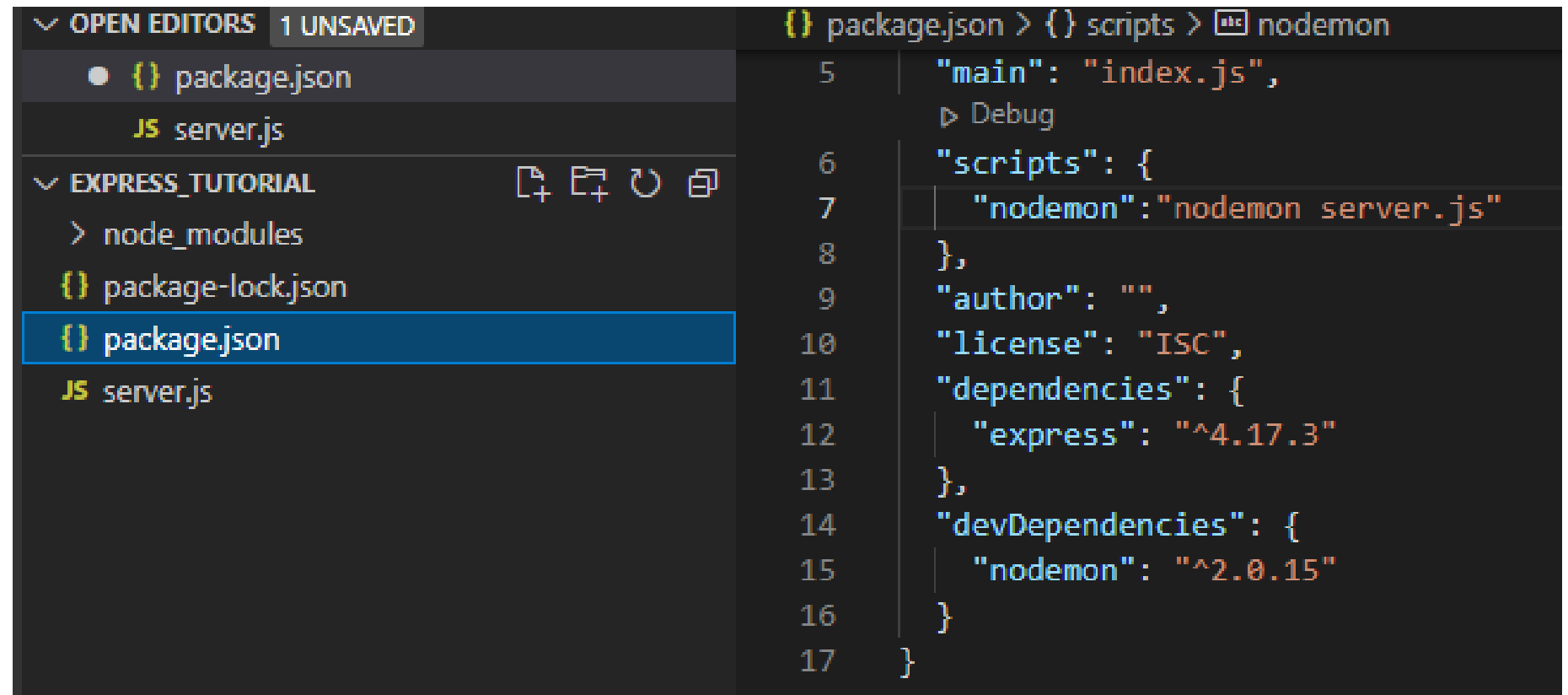


The screenshot shows the Visual Studio Code interface with the `package.json` file open. The Explorer sidebar on the left shows the project structure with `package.json` selected. The main editor area displays the following JSON content:

```
5  "main": "index.js",
6  "scripts": {
7    "test": "echo \"Error: no test specified\" && exit 1"
8  },
9  "author": "",
10 "license": "ISC",
11 "dependencies": {
12   "express": "^4.17.3"
13 },
14 "devDependencies": {
15   "nodemon": "^2.0.15"
16 }
17 }
```

The `"devDependencies": {` section and the `"nodemon": "^2.0.15"` entry are highlighted with a blue underline.

- Now remove **“test”** from **scripts** and type
"scripts": {
 "nodemon": "nodemon server.js"
},
- Now create a new file with a name **server.js**
- Type **npm run nodemon** in the Terminal.



The screenshot shows the VS Code editor interface. On the left, the Explorer sidebar shows the project structure with 'package.json' selected. The main editor area displays the content of 'package.json', which has been updated to include the 'nodemon' script. The terminal at the bottom shows the command 'npm run nodemon' being executed, resulting in a successful start of the application.

```
{} package.json > {} scripts > npm run nodemon
5  "main": "index.js",
6  "scripts": {
7    "nodemon": "nodemon server.js"
8  },
9  "author": "",
10 "license": "ISC",
11 "dependencies": {
12   "express": "^4.17.3"
13 },
14 "devDependencies": {
15   "nodemon": "^2.0.15"
16 }
17 }
```


```
const express=require('express')
const app=express()
app.get("/",(req,res)=>{
  console.log("Here")
  res.send("Hi")
})
app.listen(3000)
```

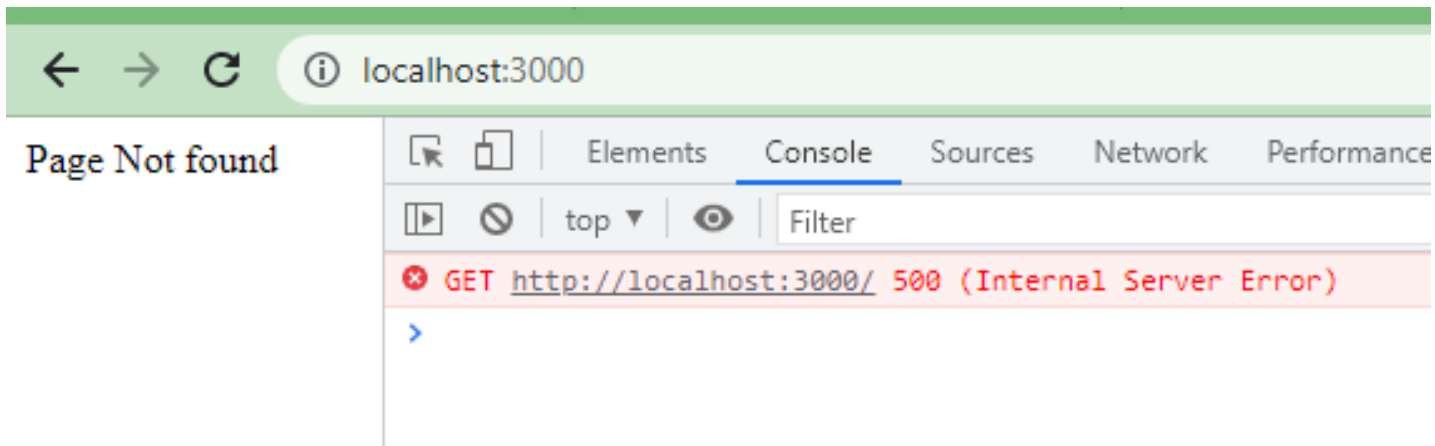
- Send() is used for testing purpose

```
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
Here
█
```

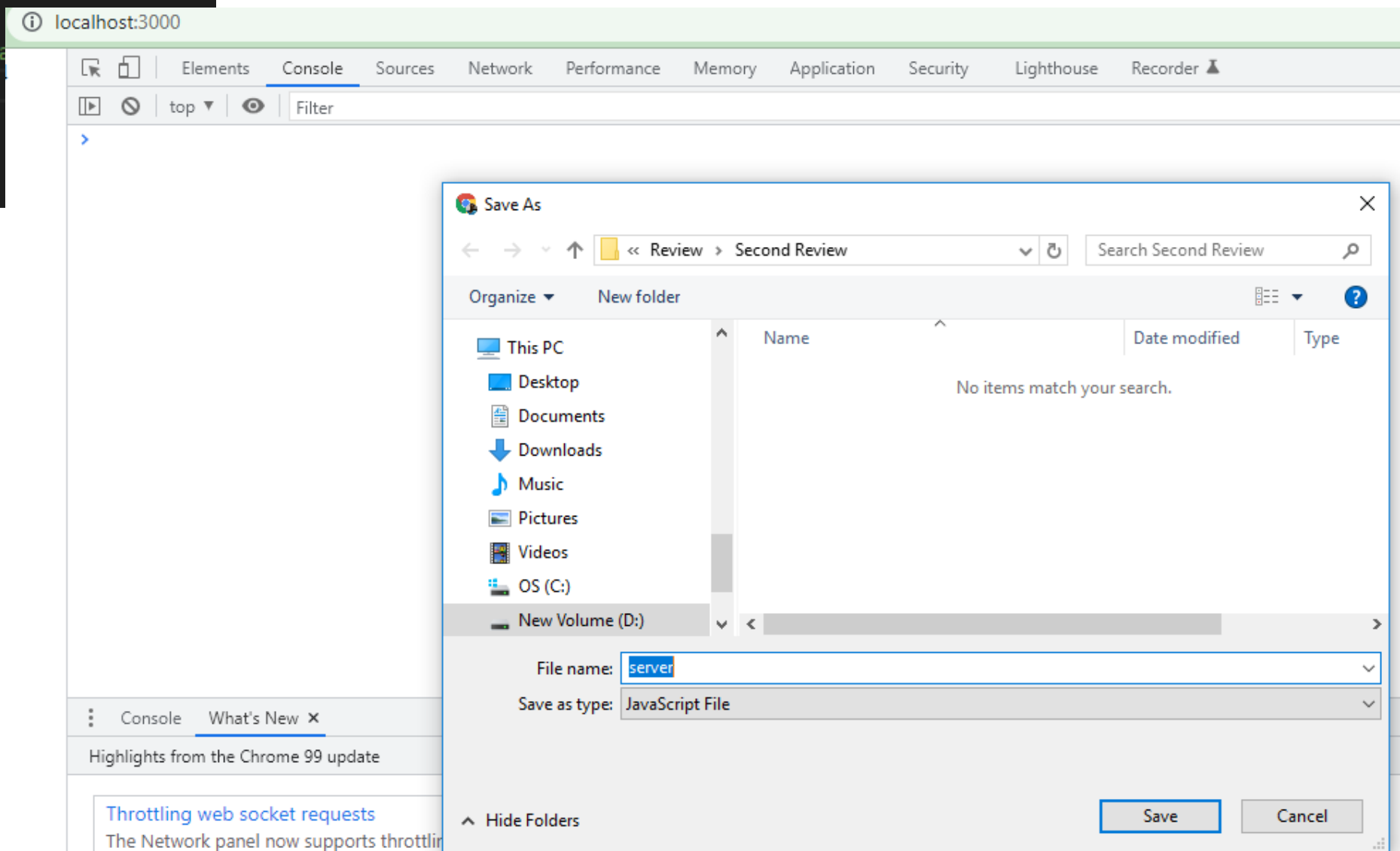
← → ↻ ⓘ localhost:3000

Hi

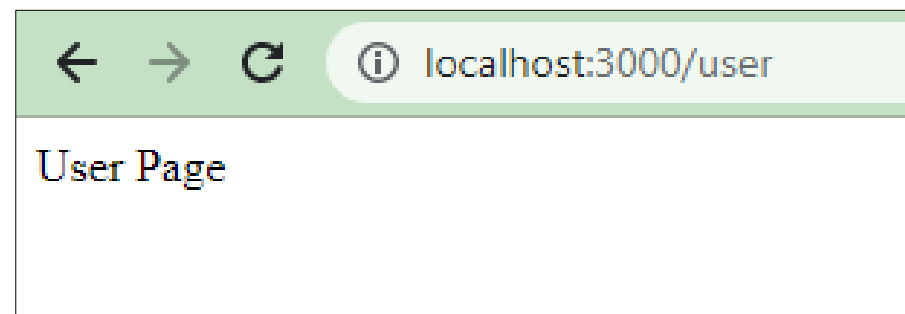
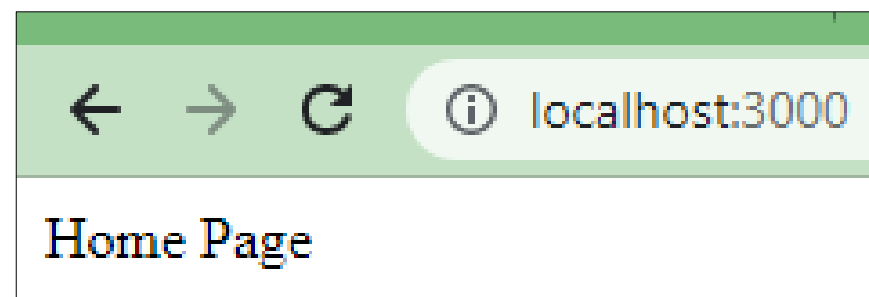
```
JS server.js >  app.get("/") callback
1  const express=require('express')
2  const app=express()
3  app.get("/",(req,res)=>{
4    console.log("Here")
5    res.status(500).send(["Page Not found"])
6  }
7  })
8  app.listen(3000)
```




```
{ } package.json JS server.js X
JS server.js > app.get("/") callback
1 const express=require('express')
2 const app=express()
3 app.get("/",(req,res)=>{
4   console.log("Here")
5   // res.status(500).send("Pa
6   res.download("server.js")
7 })
8 app.listen(3000)
```



```
{ } package.json  JS server.js  X
JS server.js > ...
1  const express=require('express')
2  const app=express()
3  app.get("/",(req,res)=>{
4
5      res.send("Home Page")
6  })
7  app.get("/user",(req,res)=>{
8
9      res.send("User Page")
10 })
11 app.listen(3000)
```



HTTP Methods

- **GET**

The GET method should retrieve the data. It should send limited amount of data only and that data are visible in the url address.

- **POST**

The POST method send large amount of data and that data are not visible in the url address. The data enclosed in the request is accept by the server as a new object/entity of the resource

- **PUT**

The data enclosed in the request is accept by the server as a modification to existing object

- **DELETE**

This method requests the server to delete the specified resource.

Request	Response
req.app	res.app
req.body	res.send
req.params	res.redirect
req.path	res.location
req.query	res.append
req.route	res.attachment
req.cookie	res.cookie
req.secure	res.download
req.subdomains	res.render
req.baseUrl	res.end

Form processing

Index.html

```
<html>
  <head>
    <title>My first Form Processing</title>
  </head>
  <body>
    <form action="http://localhost:8080/home" method="get">
      UserName: <input type="text" name="user"/>
      <button type="submit">Submit</button>
    </form>
  </body>
</html>
```

App.js

```
var express=require('express');
var app=express();
app.get('/',(req,res)=>{
    res.sendFile(__dirname + "/" + "index.html");
});

app.get('/home',(req,res)=>{
    res.send('<h1>Welcome'+ req.query['user'] + '</h1>');
});
app.listen(8080)
```

<> index.html X

JS app.js

{ } package.json

<> index.html >  html


```
1  <html>
2    <head>
3      <title>My first Form Processing</title>
4    </head>
5    <body>
6      <form action="http://localhost:3000/home" method="get">
7        UserName: <input type="text" name="user"/>
8        <button type="submit">Submit</button>
9      </form>
10   </body>
11 </html>
```

<> index.html

JS app.js

X

{ } package.json

JS app.js >  app.get('/home') callback

```
1  var express=require('express');
2  var app=express();
3  app.get('/',(req,res)=>{
4    res.sendFile(__dirname + "/" + "index.html");
5  });
6
7  app.get('/home',(req,res)=>{
8    // res.send("welcome");
9     res.send('<h1>Welcome ' + req.query['user'] + '</h1>');
10  });
11  app.listen(3000)
12
13
```


- If we use post method, the values will be encrypted and send in the hidden manner.
- We can't display or access the encrypted code.
- Express **body-parser** is an npm library used to process data sent through an HTTP request body.

bodyParser.urlencoded([options])


- Returns middleware that only parses urlencoded bodies and only looks at requests where the Content-Type header matches the type option. This parser accepts only UTF-8 encoding of the body and supports automatic inflation of gzip and deflate encodings.
- A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This object will contain key-value pairs, where the value can be a string or array (when extended is false), or any type (when extended is true).

Key-extended

- The extended option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true).

index.html > html > body > form

```
1  <!DOCTYPE html>
2  <html>
3      <body>
4          <form action="http://localhost:8080/home" method="POST">
5              Name: <input type="text" name="username"/>
6              Mail ID : <input type="email" name="mailid"/>
7              <input type="submit" value="submit"/>
8          </form>
9      </body>
10 </html>
```

JS app.js >  app.post('/home') callback

```
1  var express = require('express');
2  var app = express();
3  var bodyParser = require('body-parser');
4  var urlencodedParser = bodyParser.urlencoded({ extended: false})
5  app.get('/',function(req,res){
6    res.sendFile(__dirname + "/"+"index.html");
7  });
8  app.post('/home',urlencodedParser,function(req,res){
9    res.send('<h1>Welcome' + req.body.username+'</h1><br><h2>Mail ID : ' +req.body.mailid+'</h2>');
10 });
11 app.listen(8080);
```

Mongodb

Node JS + MongoDB

- MongoDB is a most popular NoSQL Database
- Node.js can be used in database applications.
- Now, We use MongoDB as a database with Node.js
- If you want to access MongoDB database by using node js then you first install MongoDB in your system.

<https://www.mongodb.com>

- You can download MongoDB by visit the above website
- Once you installed and running MongoDB on your system then you can access it by using Node.js
- So, You also need a MongoDB driver. You download "MongoDB" module from NPM.

```
npm install mongodb
```

Installation

- Nodejs: <https://nodejs.org/en/>
- Mongodb: <https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/>



Products

Solutions

Resources

Company

Pricing



Sign In

Try Free



Atlas

MongoDB as a service



On-premises

MongoDB locally



Tools

Boost productivity



Mobile & Edge

Realm Datastore

MongoDB Community Server

The Community version of our distributed database offers a flexible document data model along with support for ad-hoc queries, secondary indexing, and real-time aggregations to provide powerful ways to access and analyze your data.

The database is also offered as a fully-managed service with [MongoDB Atlas](#). Get access to advanced functionality such as auto-scaling, serverless instances (in preview), full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

Available Downloads

Version

5.0.6 (current)



Platform

Windows










Package

msi



Download

Copy Link

» This PC » OS (C:) » Program Files » MongoDB » Server » 5.0 » bin					
Name	Date modified	Type	Size		
 InstallCompass	27-01-2022 16:20	Windows PowerS...	2 KB		
 mongo	27-01-2022 18:03	Application	21,780 KB		
 mongod.cfg	11-04-2022 10:47	CFG File	1 KB		
 mongod	27-01-2022 18:02	Application	46,547 KB		
 mongod	27-01-2022 18:02	Program Debug D...	5,19,684 KB		
 mongos	27-01-2022 17:17	Application	29,367 KB		
 mongos	27-01-2022 17:17	Program Debug D...	3,04,908 KB		

- In order to run the mongodb client click **mongo**

- To run mongo and mongod copy the path and paste it in command prompt as follows:

```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd C:\Program Files\MongoDB\Server\5.0\bin

C:\Program Files\MongoDB\Server\5.0\bin>
```

- Now type mongod and press enter. Now the mongodb server will be started.

```
C:\Program Files\MongoDB\Server\5.0\bin>mongod
{"t":{"$date":"2022-04-11T11:44:17.618+05:30"},"s":"I", "c":"NETWORK", "id":
4915701, "ctx":"-", "msg":"Initialized wire specification", "attr":{"spec":{"inc
omingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingIntern
alClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVersion"
:0,"maxWireVersion":13},"isInternalClient":true}}}
```

- Now you will get this error that c:\data\db\ not found..
- So create that folder manually.

Select Command Prompt

```
", "attr": {"error": "NonExistentPath: Data directory C:\\data\\db\\ not found.  
create the missing directory or specify another path using (1) the --dbpath o  
mand line option, or (2) by adding the 'storage.dbPath' option in the confi
```

- After creating manually, once again give the command as **mongod** and press enter. Now you wont find any error.

```
rage": {}, "protocol": "op_msg", "durationMillis": 363}}  
{ "t": { "$date": "2022-04-11T11:52:55.625+05:30" }, "s": "I", "c": "STORAC  
22430, "ctx": "Checkpoint", "msg": "WiredTiger message", "attr": { "me  
649658175:625135][1924:140735622361856], WT_SESSION.checkpoint: [WT_  
POINT_PROGRESS] saving checkpoint snapshot min: 34, snapshot max: 34  
count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0,  
ite gen: 1"}}
```

- Now open another command prompt and To run mongo client copy the path and paste it in command prompt as follows:

```
Command Prompt
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Admin>cd C:\Program Files\MongoDB\Server\5.0\bin

C:\Program Files\MongoDB\Server\5.0\bin>
```

- Now type mongo and press enter. Now the mongod client will be started.
- Now you can start typing the commands.

```
...to permanently c
sableFreeMonitoring()
---
> show dbs
admin          0.000GB
config         0.000GB
local          0.000GB
university    0.000GB
>
```

MySQL	MongoDB
table	collection
insert	insertOne , insertMany
record	document
select	findOne, find
order by	sort
delete	deleteOne, deleteMany
update	updateOne, updateMany

```
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.3
connecting to: mongodb://127.0.0.1:27017
```

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
newdb      0.000GB
students   0.000GB
>
```

- To create collection(table) profile in the database cheetah

```
> use cheetah
switched to db cheetah
> db.createCollection('profile')
{ "ok" : 1 }
```

- To see the collections

```
> show collections
profile
```

- To create more collections and display collections

```
> db.createCollection('student')
{ "ok" : 1 }
> show collections
profile
student
>
```

- To delete the collection

```
> db.profile.drop()
true
> show collections
student
>
```

- To delete the entire database

```
> db.dropDatabase()
{ "dropped" : "cheetah", "ok" : 1 }
>
```

- To insert the values in the collection and to display

```
> use cheetah
switched to db cheetah
> db.profile.insert({Name:"John",Age:"25"})
WriteResult({ "nInserted" : 1 })
> db.profile.find()
{ "_id" : ObjectId("6194f1cb9f2c1a955ffc1ff7"), "Name" : "John", "Age" : "25"
}
>
```


`ObjectId(<hexadecimal>)`

Returns a new `ObjectId` value. The 12-byte `ObjectId` value consists of:

- a 4-byte *timestamp value*, representing the `ObjectId`'s creation, measured in seconds since the Unix epoch
- a 5-byte *random value* generated once per process. This random value is unique to the machine and process.
- a 3-byte *incrementing counter*, initialized to a random value

- To display in json format

```
> db.profile.find().pretty()
{
  "_id" : ObjectId("6194f1cb9f2c1a955ffc1ff7"),
  "Name" : "John",
  "Age" : "25"
}
```

- Id can be generated by the user. But not recommended.

```
> db.profile.insert({_id:1,Name:"Teju",Age:"35"})
WriteResult({ "nInserted" : 1 })
> db.profile.find().pretty()
{
  "_id" : ObjectId("6194f1cb9f2c1a955ffc1ff7"),
  "Name" : "John",
  "Age" : "25"
}
{ "_id" : 1, "Name" : "Teju", "Age" : "35" }
```

- To check number of elements

```
> db.profile.find().count()  
2
```

- To insert many values

```
> db.profile.insertMany([ {Name:"Harry",Age:40}, {Name:"Marry",Age:38} ] )  
{  
  "acknowledged" : true,  
  "insertedIds" : [  
    ObjectId("5daab6986954d938dec231e8"),  
    ObjectId("5daab6986954d938dec231e9")  
  ]  
}
```

- To display the values

```
> db.profile.find().pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
{ "_id" : 1, "Name" : "Doe", "Age" : 20 }
{
  "_id" : ObjectId("5daab6466954d938dec231e7"),
  "Name" : "Ben",
  "City" : "USA"
}
{
  "_id" : ObjectId("5daab6906954d938dec231e8"),
  "Name" : "Harry",
  "Age" : 40
}
{
  "_id" : ObjectId("5daab6906954d938dec231e9"),
  "Name" : "Marry",
  "Age" : 38
}
```

- To display only Ben details Note:{} brackets represents the query

```
> db.profile.find({Name:"Ben"}).pretty()
{
  "_id" : ObjectId("5daab6466954d938dec231e7")
  "Name" : "Ben",
  "City" : "USA"
}
```

```
> db.profile.find({Age:{$gt:38}}).pretty()
{
  "_id" : ObjectId("5daab6906954d938dec231e8"),
  "Name" : "Harry",
  "Age" : 40
}
>
```

```
> db.profile.find({Age:{$lt:38}}).pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
{ "_id" : 1, "Name" : "Doe", "Age" : 20 }
> db.profile.find({Age:{$lte:38}}).pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
{ "_id" : 1, "Name" : "Doe", "Age" : 20 }
{
  "_id" : ObjectId("5daab6906954d938dec231e9"),
  "Name" : "Marry",
  "Age" : 38
}
>
```

```
> db.profile.find({Age:{$eq:38}}).pretty()
{
  "_id" : ObjectId("5daab6906954d938dec231e9"),
  "Name" : "Marry",
  "Age" : 38
}
>
```

update

```
db.profile.update({Name:"Ben"},{$set:{City:"UK"}})  
writeResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.profile.find().pretty()  
{  
  "_id" : ObjectId("5daab5006954d938dec231e6"),  
  "Name" : "John",  
  "Age" : 25  
}  
{ "_id" : 1, "Name" : "Doe", "Age" : 20 }  
{  
  "_id" : ObjectId("5daab6466954d938dec231e7"),  
  "Name" : "Ben",  
  "City" : "UK"  
}  
{  
  "_id" : ObjectId("5daab6906954d938dec231e8"),  
  "Name" : "Harry",  
  "Age" : 40  
}  
{  
  "_id" : ObjectId("5daab6906954d938dec231e9"),  
  "Name" : "Marry",  
  "Age" : 38  
}
```


- And , or

```
> db.profile.find({$and:[{Name:"John"},{Age:25}]}).pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
```

- In

```
> db.profile.find({Name:{$in:["John"]}}).pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
> db.profile.find({Name:{$in:["John","Ben"]}}).pretty()
{
  "_id" : ObjectId("5daab5006954d938dec231e6"),
  "Name" : "John",
  "Age" : 25
}
{
  "_id" : ObjectId("5daab6466954d938dec231e7"),
  "Name" : "Ben",
  "City" : "UK"
}
```


- Open vs code->create anew folder Mongo-> create a new file app.js.
- In the terminal do the following
 - Type npm init -y
 - Type npm install mongodb
- Now type the following code in app.js to **create database connection**

```
var MongoClient =require('mongodb').MongoClient;  
var url="mongodb://localhost:27017/demodb";  
MongoClient.connect(url,function(err,db){  
  if (err) throw err;  
  console.log("Database connected Successfully");  
  db.close();  
})
```
- Execute the file as node app.js
- **Database connected Successfully** message will be displayed in the terminal.
- This shows that database is connected properly.

To create collections:

```
var MongoClient = require('mongodb').MongoClient;
var url="mongodb://localhost:27017/";
MongoClient.connect(url,function(err,db){
  if (err) throw err;
  var dbname=db.db("demodb");//mention the database where we create tables
  dbname.createCollection("students", function(err,result){
    if(err) throw err;
    console.log("collection created");
    db.close();
  })
})
```

- **To insert one record**

```
var MongoClient =require('mongodb').MongoClient;
var url="mongodb://localhost:27017/";
MongoClient.connect(url,function(err,db){
  if (err) throw err;
  var dbname=db.db("demodb");//mention the database where we create tables
  dbname.collection("students").insertOne({Name:"Mary",Place:"Vellore"},
function(err,result){
  if(err) throw err;
  console.log("One Document inserted");
  db.close();
})
})
```

- **To insert more than one record**

```
var MongoClient =require('mongodb').MongoClient;
var url="mongodb://localhost:27017/";
MongoClient.connect(url,function(err,db){
  if (err) throw err;
  var dbname=db.db("demodb");//mention the database where we create tables
  var multipliedata=[
    {Name:"Santhi",Place:"Chennai"},
    {Name:"Selvi",Place:"Chennai"}
]
  dbname.collection("students").insertMany(multipliedata, function(err,result){
    if(err) throw err;
    console.log(" Documents inserted");
    db.close();
  })
})
```

Form Processing with HTML, Node JS, MongoDB

- First in the terminal install the dependencies whatever required.

```
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Mongo> npm install express body-parser mongoose nodemon
added 193 packages, and audited 214 packages in 21s

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\mekala_f\Winter21-22\CSE2015\Module5\Mongo> █
```

EXPLORER

...

JS app.js

package.json X

OPEN EDITORS 1 UNSAVED

JS app.js

package.json

MONGO

node_modules

JS app.js

package-lock.json

package.json

OUTLINE

TIMELINE

package.json > {} scripts > nodemon

```
1 {
2   "name": "mongo",
3   "version": "1.0.0",
4   "description": "",
5   "main": "app.js",
6   "scripts": {
7     "nodemon": "nodemon app.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "body-parser": "^1.20.0",
14    "express": "^4.17.2"
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

28 packages are looking for funding
run `npm fund` for details

found 0 vulnerabilities

PS D:\mekala_f\Winter21-22\CSE2015\Module5\Mongo> npm run nodemon

> mongo@1.0.0 nodemon

> nodemon app.js

[nodemon] 2.0.15

[nodemon] to restart at any time, enter `rs`

[nodemon] watching path(s): *.*

[nodemon] watching extensions: js,mjs,json

[nodemon] starting `node app.js`

Documents inserted

[nodemon] clean exit - waiting for changes before restart

Index.html

```
<html>
  <head>
    <title>Welcome</title>
  </head>
  <body>
    <form action="http://localhost:3000/sign_up" method="POST">
      <h2>Form</h2>
      Name: <input type="text" name="name" required />
      <br>
      Place: <input type="text" name="place" required />
      <br>
      <input type="submit" value="Submit" />
    </body>
</html>
```

app.js

```
var express = require("express")
var bodyParser = require("body-parser")
var MongoClient = require('mongodb').MongoClient;
const app = express()

var bodyParser=require('body-parser')
var urlencodedparser=bodyParser.urlencoded({extended:true})

var url="mongodb://localhost:27017/";
MongoClient.connect(url,function(err,db){
  if (err) throw err;
  var dbname=db.db("demodb");
```



```
app.post("/sign_up",urlencodedparser,(req,res)=>{
    var name = req.body.name;
    var place = req.body.place;

    var data = {
        "Name": name,
        "Place" : place,
    }

    dbname.collection('students').insertOne(data,function(err,result){

        if(err) throw err;

        console.log("Record Inserted Successfully");
        res.send("Record Successfully inserted")
    });

})
```

```
app.get("/",(req,res)=>{  
    res.sendFile(__dirname + "/" + "index.html");  
})  
app.listen(3000);  
})
```

```
console.log("Listening on PORT 3000");
```