

# Node.js Documentation

## Module 2: Node.js Platform Setup

- **Node REPL**

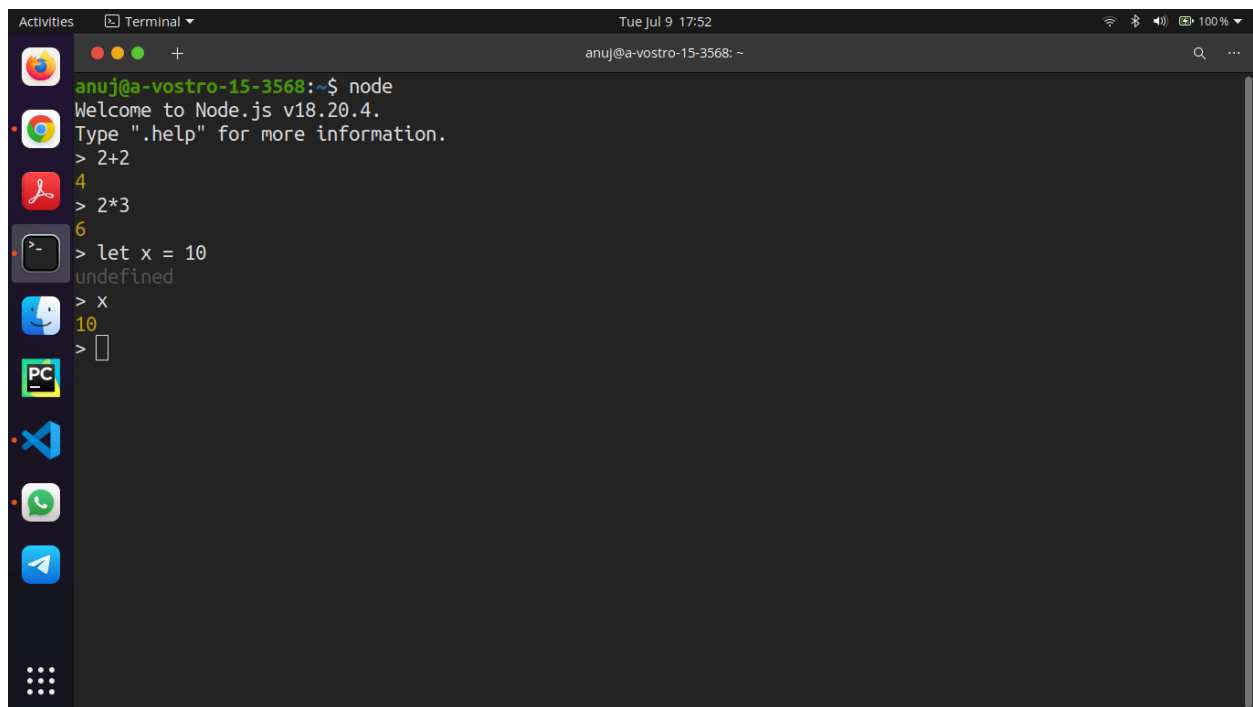
The Node.js REPL (Read-Eval-Print Loop) is an interactive shell that allows you to enter JavaScript code and immediately see the results. It's a useful tool for testing and experimenting with Node.js code snippets in real-time.

### **Starting the Node.js REPL**

To start the Node.js REPL, simply open your terminal or command prompt and type: `node`

This will start the REPL, and you'll see a prompt (`>`) where you can start typing JavaScript code.

### **Basic Operations in the REPL**

A screenshot of a Linux terminal window. The window title is "Terminal" and the user is "anuj@anuj-vostro-15-3568". The terminal shows the command "node" being executed, which outputs "Welcome to Node.js v18.20.4. Type '.help' for more information." followed by several arithmetic calculations: "> 2+2" returns "4", "> 2\*3" returns "6", and "> let x = 10" returns "undefined". Finally, "> x" returns "10". The terminal is part of a desktop environment with a sidebar on the left containing icons for various applications like a file manager, web browser, and messaging apps. The top status bar shows the date and time as "Tue Jul 9 17:52" and battery level at "100%".

```
anuj@anuj-vostro-15-3568:~$ node
Welcome to Node.js v18.20.4.
Type ".help" for more information.
> 2+2
4
> 2*3
6
> let x = 10
undefined
> x
10
>
```

## ● Download and Install Node.js

To download and install Node.js, follow these steps based on your operating system:

### For Windows

#### 1. Download Node.js Installer:

- Go to the Node.js download page.
- Click the Windows Installer button to download the **.msi** file.

#### 2. Run the Installer:

- Locate the downloaded **.msi** file and double-click to run it.
- Follow the prompts in the Node.js Setup Wizard. Accept the license agreement and choose the installation options.
- Make sure to check the box that says "Automatically install the necessary tools" (if available).

#### 3. Verify Installation:

- Open Command Prompt (**cmd**) or PowerShell.
- Type the following commands to verify the installation: `node -v`
- `npm -v`

## **For macOS**

### **1. Download Node.js Installer:**

- Go to the Node.js download page.
- Click the macOS Installer button to download the **.pkg** file.

### **2. Run the Installer:**

- Locate the downloaded **.pkg** file and double-click to run it.
- Follow the prompts in the Node.js Installer. Accept the license agreement and proceed with the installation.

### **3. Verify Installation:**

- Open Terminal.
- Type the following commands to verify the installation: `node -v`
- `npm -v`

## **● Import Required Modules**

In Node.js, you can import required modules using the **require** function for CommonJS modules, which is the module system used by Node.js by default. Here's how you can import some commonly used modules:

### **Core Modules**

```
const fs = require('fs'); // File system module
const http = require('http'); // HTTP module for creating web servers
const path = require('path'); // Path module for working with file and directory paths
```

```
const os = require('os'); // OS module for getting operating system-related utility
methods and properties
const util = require('util'); // Utilities module
```

## External Modules

External modules are installed via npm (Node Package Manager). For example, to use the `express` module, you first need to install it using npm:

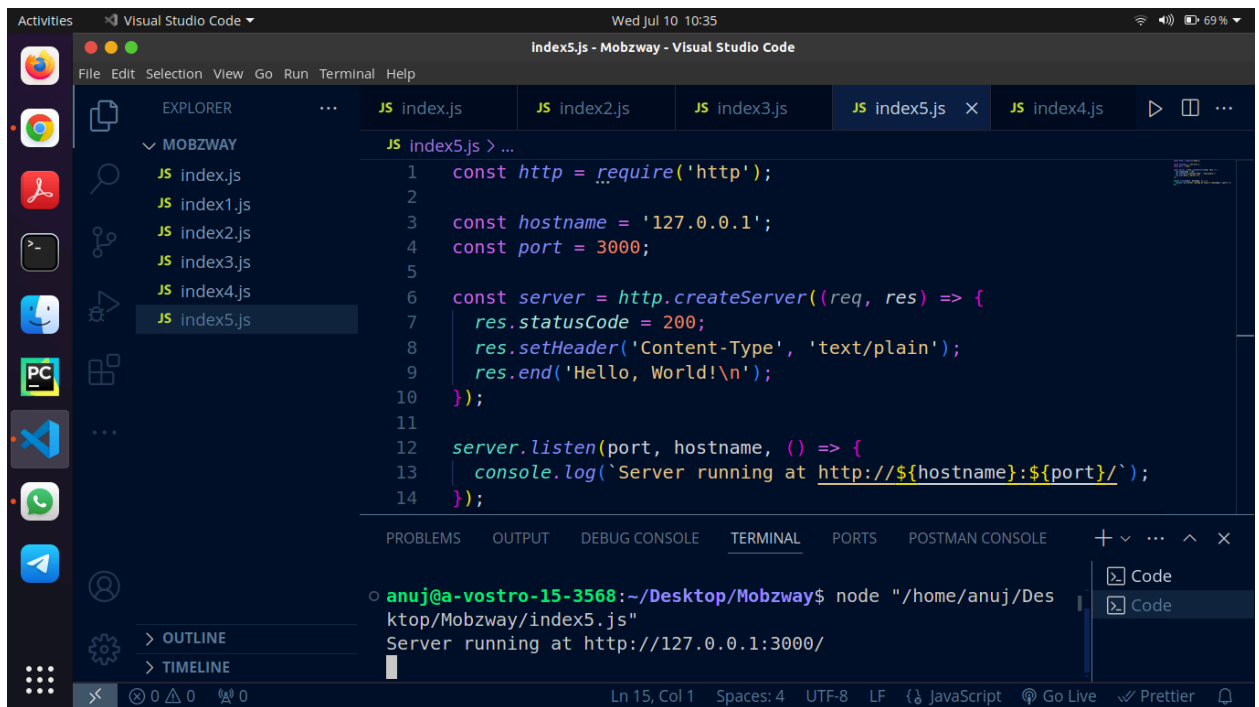
```
npm install express
```

Then, you can import it in your Node.js application.

## ● Creating Web Server

Creating a web server in Node.js is straightforward, I'll show you how to create a web server: using the built-in `http` module.

## Using the `http` Module



The screenshot shows the Visual Studio Code interface with a file explorer on the left displaying a project named 'MOBZWAY' containing files 'index.js' through 'index5.js'. The main editor window shows 'index5.js' with the following JavaScript code:

```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello, World!\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

At the bottom, the TERMINAL panel shows the command executed and the output:

```
anuj@a-vostro-15-3568:~/Desktop/Mobzway$ node "/home/anuj/Desktop/Mobzway/index5.js"
Server running at http://127.0.0.1:3000/
```

The status bar at the bottom indicates the current line and column (Ln 15, Col 1), encoding (UTF-8), and the active language (JavaScript).

let's break down this code step by step:

### Importing the `http` Module

This will import the built-in `http` module, which provides the functionality to create an HTTP server.

### Setting the Hostname and Port

Here, `hostname` is set to `'127.0.0.1'`, which is the loopback IP address (localhost). The `port` is set to `3000`. These are the address and port number where the server will listen for incoming requests.

### Creating the Server

1. `http.createServer((req, res) => { ... })`: This function creates an HTTP server. It takes a callback function as an argument. This callback function is called every time a request is received by the server.
2. `(req, res)`: The callback function takes two arguments:
  - `req`: An object representing the incoming request.
  - `res`: An object representing the response that will be sent to the client.
3. `res.statusCode = 200`;: This line sets the status code of the response to `200`, which means "OK".
4. `res.setHeader('Content-Type', 'text/plain')`;: This line sets the `Content-Type` header of the response to `text/plain`. This tells the client that the response body is plain text.
5. `res.end('Hello, World!\n')`;: This line ends the response and sends the string `'Hello, World!\n'` to the client.

### Starting the Server

1. `server.listen(port, hostname, () => { ... })`: This method starts the server and makes it listen on the specified `port` and `hostname`.

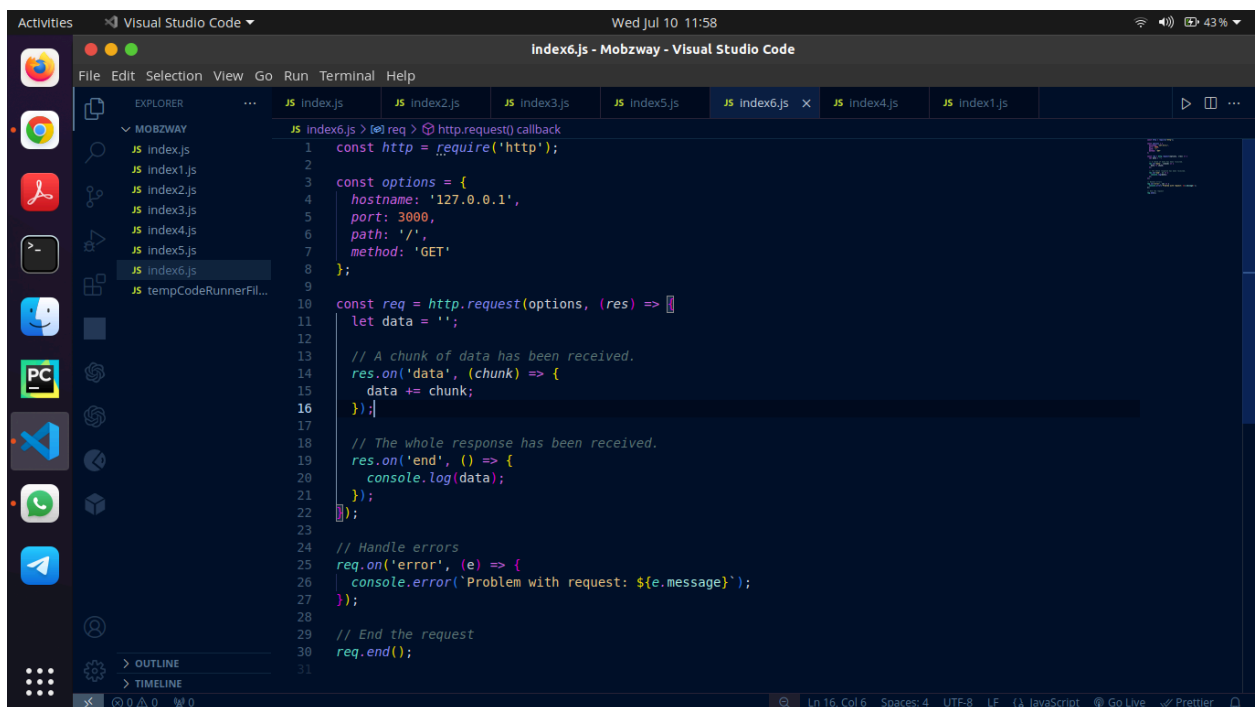
- 
2. `() => { ... }`: This is a callback function that is executed once the server starts successfully.
3. `console.log(Server running at http://${hostname}:${port}/);`: This line logs a message to the console, indicating that the server is running and providing the URL where it can be accessed.

## ● Sending Request

Here is the example of sending HTTP requests using http module in Node.js:

### Using the Built-in http Module

Here's an example of how to send a GET request using the http module:



```
1  const http = require('http');
2
3  const options = {
4    hostname: '127.0.0.1',
5    port: 3000,
6    path: '/',
7    method: 'GET'
8  };
9
10 const req = http.request(options, (res) => {
11   let data = '';
12
13   // A chunk of data has been received.
14   res.on('data', (chunk) => {
15     data += chunk;
16   });
17
18   // The whole response has been received.
19   res.on('end', () => {
20     console.log(data);
21   });
22 });
23
24 // Handle errors
25 req.on('error', (e) => {
26   console.error('Problem with request: ${e.message}');
27 });
28
29 // End the request
30 req.end();
```

### Setting Up Request Options

Here, an `options` object is created to specify the details of the HTTP request:

- **hostname**: The server address. In this case, it's the loopback IP address (localhost).
- **port**: The port number to connect to on the server.
- **path**: The URL path to request from the server.
- **method**: The HTTP method to use for the request (GET, in this case).

## Making the Request

**http.request(options, callback)**: This function creates a new HTTP request using the provided **options** and a callback function that handles the response.

1. **Callback function (res) => { ... }**: This function is called when the server responds. It receives a **res** (response) object.
2. **res.on('data', (chunk) => { ... })**: This event listener handles the incoming data chunks. The data received is appended to the **data** variable.
3. **res.on('end', () => { ... })**: This event listener is called when the entire response has been received. It logs the complete response data to the console.

## Handling Errors

**req.on('error', (callback))**: This event listener is called if there is an error with the request. It logs the error message to the console.

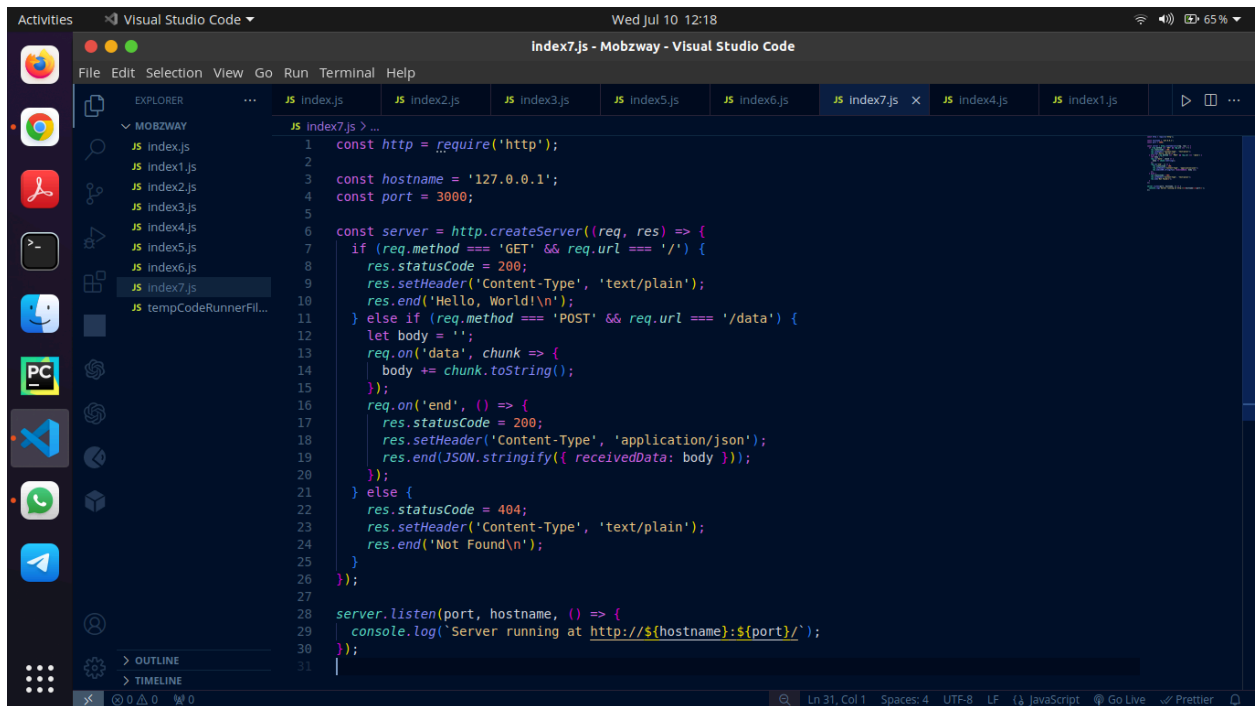
## Ending the Request

**req.end()**: This function signals that the request has been completed and no more data will be sent. It is necessary to call **req.end()** to actually send the request.

- **Handling HTTP requests**

Handling HTTP requests in Node.js typically involves creating a server that listens for incoming requests and responds appropriately. This can be done using the built-in `http` module.

**Using the Built-in `http` Module**



```
1 const http = require('http');
2
3 const hostname = '127.0.0.1';
4 const port = 3000;
5
6 const server = http.createServer((req, res) => {
7   if (req.method === 'GET' && req.url === '/') {
8     res.statusCode = 200;
9     res.setHeader('Content-Type', 'text/plain');
10    res.end('Hello, World!\n');
11  } else if (req.method === 'POST' && req.url === '/data') {
12    let body = '';
13    req.on('data', chunk => {
14      body += chunk.toString();
15    });
16    req.on('end', () => {
17      res.statusCode = 200;
18      res.setHeader('Content-Type', 'application/json');
19      res.end(JSON.stringify({ receivedData: body }));
20    });
21  } else {
22    res.statusCode = 404;
23    res.setHeader('Content-Type', 'text/plain');
24    res.end('Not Found\n');
25  }
26 });
27
28 server.listen(port, hostname, () => {
29   console.log(`Server running at http://${hostname}:${port}/`);
30 });
31
```

**Handling GET Requests**

1. **Checking the Request Method and URL:** This `if` statement checks if the request method is `GET` and the request URL is `/`.
2. **Setting the Status Code:** `res.statusCode = 200;` sets the HTTP status code of the response to `200`, which means "OK".
3. **Setting the Header:** `res.setHeader('Content-Type', 'text/plain');` sets the `Content-Type` header of the response to `text/plain`, indicating that the response body is plain text.
4. **Ending the Response:** `res.end('Hello, World!\n');` ends the response and sends the string `'Hello, World!\n'` to the client.



## Handling POST Requests

**Checking the Request Method and URL:** This `else if` statement checks if the request method is `POST` and the request URL is `/data`.

### 1. Handling Incoming Data:

- `let body = ''`; initializes an empty string to accumulate the incoming data chunks.
- `req.on('data', chunk => { body += chunk.toString(); });` sets up an event listener for the `data` event, which is emitted when a chunk of data is received. The received chunk is converted to a string and appended to the `body` variable.

### 2. Handling End of Data:

- `req.on('end', () => { ... });` sets up an event listener for the `end` event, which is emitted when all data has been received.
- Inside the `end` event listener:
  - `res.statusCode = 200`; sets the HTTP status code of the response to `200`, indicating success.
  - `res.setHeader('Content-Type', 'application/json');` sets the `Content-Type` header to `application/json`, indicating that the response body is JSON.
  - `res.end(JSON.stringify({ receivedData: body }));` ends the response and sends a JSON string containing the received data back to the client.

- **Write your First Hello-World Program**

```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 2000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello, World!\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
15
```

node "/home/anuj/Desktop/Mobzway/index5.js"  
anuj@a-vostro-15-3568:~/Desktop/Mobzway\$ node "/home/anuj/Desktop/Mobzway/index5.js"  
Server running at http://127.0.0.1:2000/

### Explanation

1. **Import the `http` Module:** The first line imports the built-in `http` module, which is required to create an HTTP server.
2. **Define the Hostname and Port:** The `hostname` is set to `'127.0.0.1'` (localhost), and the `port` is set to `2000`.
3. **Create the HTTP Server:** The `http.createServer` function creates a new HTTP server. The server takes a callback function that is executed each time a request is received.
4. **Start the Server:** The `server.listen` function starts the server and makes it listen on the specified port and hostname. When the server starts, a message is logged to the console.

- **Deploy your node app on any platform like Heroku, Render and firebase**

### Deploying on Heroku

1. **Install Heroku CLI:** If you haven't already, download and install the Heroku CLI.
2. **Login to Heroku:** Open your terminal and run: `heroku login`
3. **Prepare Your Application:** Ensure your `helloWorld.js` and `package.json` are in the same directory. Your `package.json` should look something like this:

```
{  
  "name": "hello-world",  
  "version": "1.0.0",  
  "main": "helloWorld.js",  
  "scripts": {  
    "start": "node helloWorld.js"  
  }  
}
```

4. **Initialize a Git Repository:**

```
git init
```

```
git add .
```

```
git commit -m "Initial commit"
```

5. **Create a Heroku App:**

```
heroku create
```

**6. Deploy Your Code:**

```
git push heroku master
```

**7. Open Your App:**

```
heroku open
```