Let's go step by step with a simple **Node.js server tutorial**. By the end, you'll have a basic server running that can respond to HTTP requests.

---

1. Prerequisites

- Install **Node.js**: https://nodejs.org
- A code editor like **VS Code**.

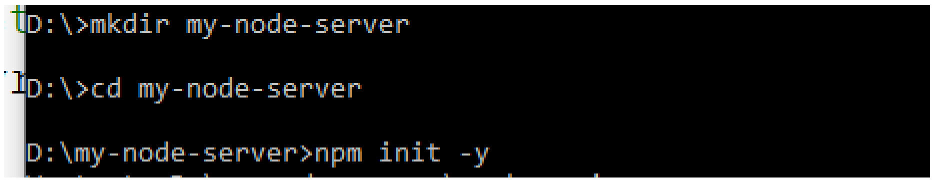
---

2. Initialize a Node.js project

Open a terminal and run:

```
mkdir my-node-server
cd my-node-server
npm init -y
```

This creates a `package.json` file.



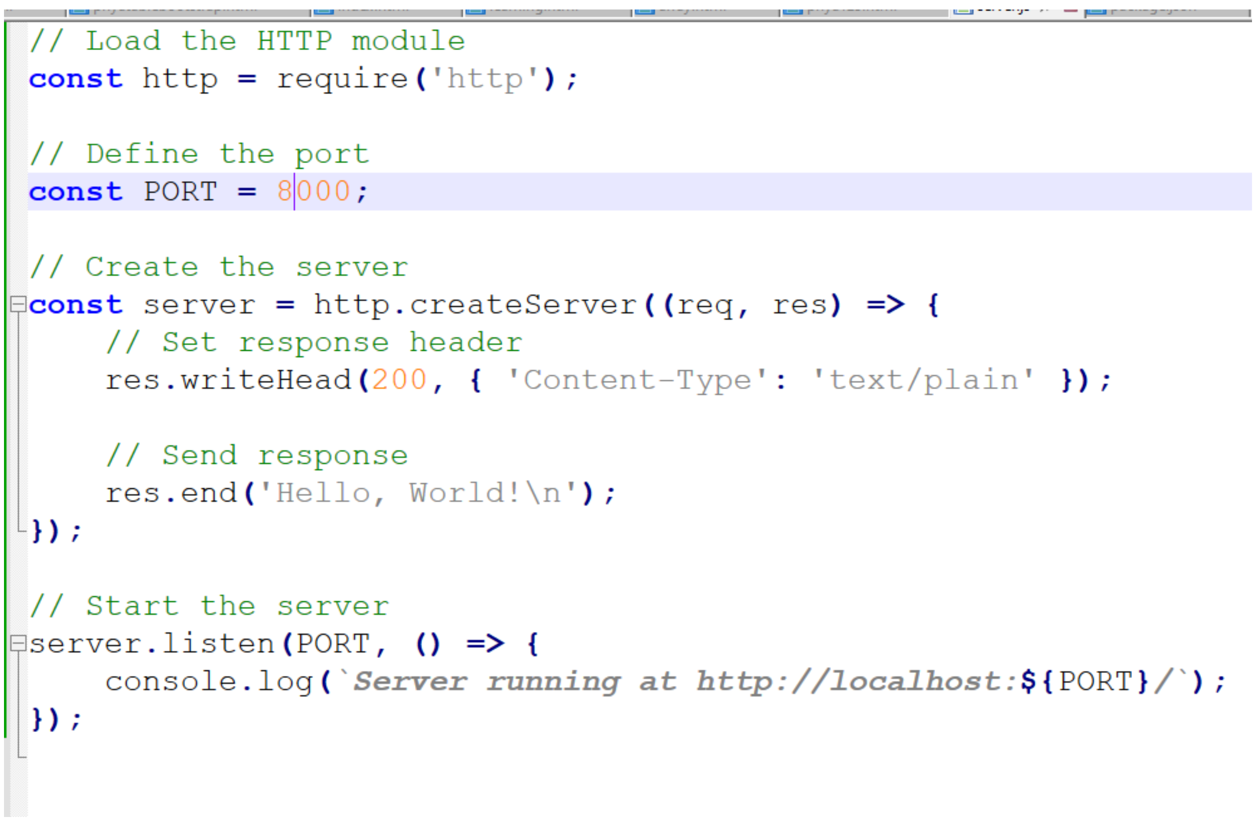
```
D:\>mkdir my-node-server

D:\>cd my-node-server

D:\my-node-server>npm init -y
```

3. Create a server using the built-in HTTP module

Create a file called `server.js`:

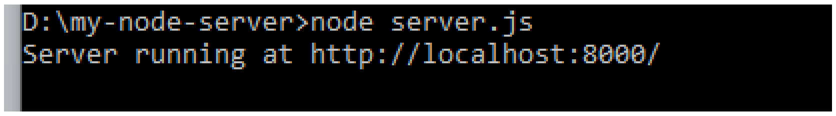
```
// Load the HTTP module
const http = require('http');

// Define the port
const PORT = 8000;

// Create the server
const server = http.createServer((req, res) => {
    // Set response header
    res.writeHead(200, { 'Content-Type': 'text/plain' });

    // Send response
    res.end('Hello, World!\n');
});

// Start the server
server.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}/`);
});
```

Open notepad and type above code in notepad++ as shown below and save it as server.js file:-



```
// Load the HTTP module
const http = require('http');

// Define the port
const PORT = 8000;

// Create the server
const server = http.createServer((req, res) => {
    // Set response header
    res.writeHead(200, { 'Content-Type': 'text/plain' });

    // Send response
    res.end('Hello, World!\n');
});

// Start the server
server.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}/`);
});
```

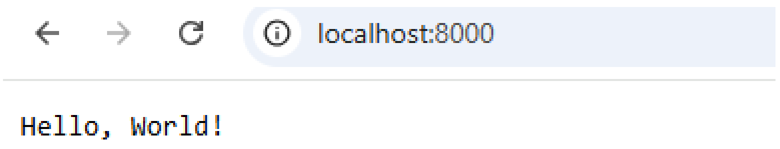
Run the server



```
D:\my-node-server>node server.js
Server running at http://localhost:8000/
```

```
node server.js
```

Then open your browser and go to http://localhost:8000.
You should see:



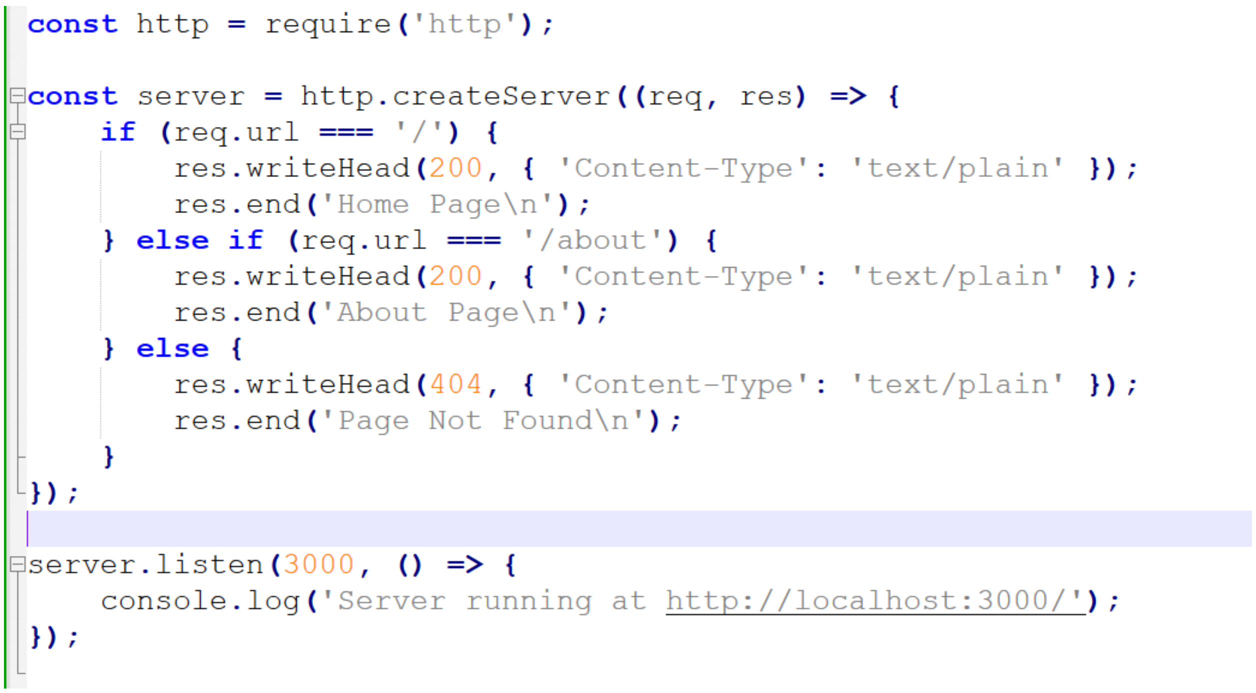
```
Hello, World!
```

5. Adding routing (optional):-

You can handle different URLs like this server.js file :

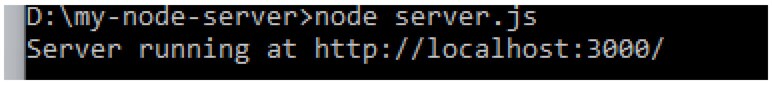
```
const http = require('http');

const server = http.createServer((req, res) => {
    if (req.url === '/') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('Home Page\n');
    } else if (req.url === '/about') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('About Page\n');
    } else {
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Page Not Found\n');
    }
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

Open notepad++ above code as shown below and save it as server.js:-



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

const server = http.createServer((req, res) => {
    if (req.url === '/') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('Home Page\n');
    } else if (req.url === '/about') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.end('About Page\n');
    } else {
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.end('Page Not Found\n');
    }
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000/');
});
```

Run it node server.js as shown below



```
D:\my-node-server>node server.js
Server running at http://localhost:3000/
```

6. Using Express (simpler way)

Express.js is one of the most popular frameworks for Node.js, and its features make building web applications and APIs much easier and more organized. Here's a detailed breakdown:

---

1. Minimal and Flexible

- Express provides a **thin layer over Node.js HTTP module**.
- It doesn't force a particular project structure, so you can design your app the way you want.
- You get **maximum flexibility** without unnecessary overhead.

---

2. Routing

- Express has a **powerful routing system** to handle different HTTP methods (`GET`, `POST`, `PUT`, `DELETE`) and URLs.

3. Middleware Support

- Middleware are functions that run **before the route handler**, used for:
  - Logging
  - Authentication
  - Request parsing
  - Error handling

4. Template Engines

- Express supports **dynamic HTML rendering** using template engines like:
  - **EJS**

5. RESTful API Support

- Express is perfect for building APIs:
  - Handles JSON requests/responses easily
  - Supports URL parameters, query strings, and headers
  - Compatible with middleware like `body-parser` for request data parsing

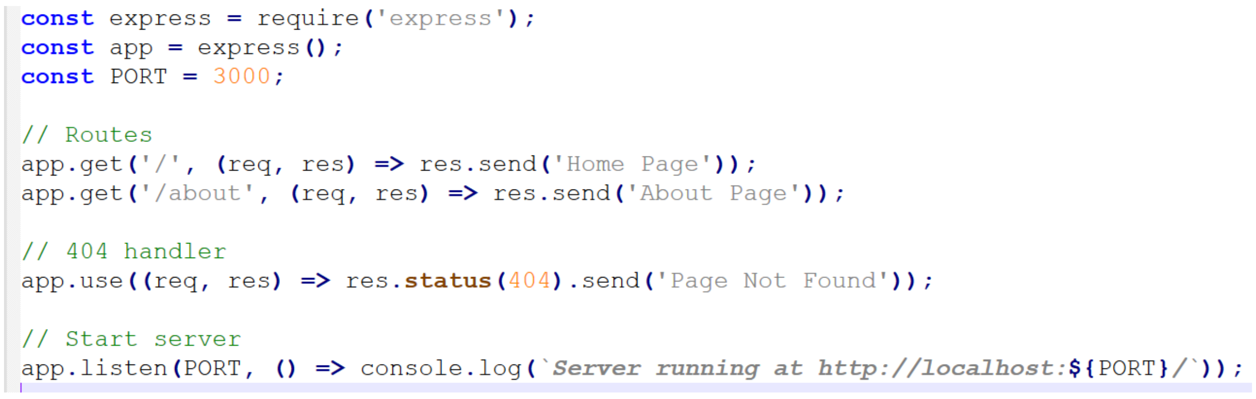
Instead of the raw HTTP module, most Node.js apps use **Express**:

```
npm install express
```

Then create `server.js`:

```
const express = require('express');
const app = express();
const PORT = 3000;

// Routes
app.get('/', (req, res) => res.send('Home Page'));
app.get('/about', (req, res) => res.send('About Page'));

// 404 handler
app.use((req, res) => res.status(404).send('Page Not Found'));

// Start server
app.listen(PORT, () => console.log(`Server running at
http://localhost:${PORT}/`));
```

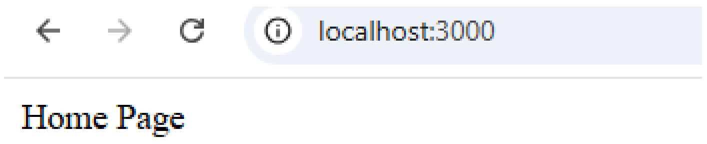
```
open notepad++ and type as shown below and save as server.js  :-
```



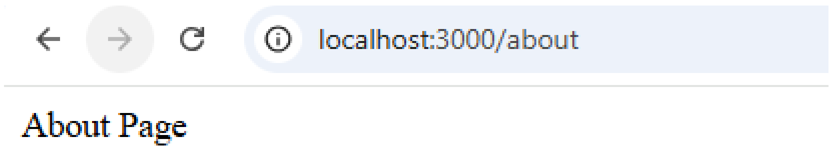
```
const express = require('express');
const app = express();
const PORT = 3000;

// Routes
app.get('/', (req, res) => res.send('Home Page'));
app.get('/about', (req, res) => res.send('About Page'));

// 404 handler
app.use((req, res) => res.status(404).send('Page Not Found'));

// Start server
app.listen(PORT, () => console.log(`Server running at http://localhost:${PORT}/`));
```

Run with:

```
node server.js
```


localhost:3000

Home Page


localhost:3000/about

About Page