**NODE SERVER EXAMPLE**

In this Project we will create a simple Node Server. Let’s start by creating a server that returns plain text to the user.

1.

Create a file with name server.js. Import Node.js http module first. (The http module contains the function to create the server.)

2.

Next, define two constants, the host and port that our server will be bound to:

const host = 'localhost';

const port = 8000;

3.

Let’s add a special function, which in Node.js we call a *request listener*. This function is meant to handle an incoming HTTP request and return an HTTP response. This function must have two arguments, a request object and a response object.

const requestListener = function (req, res) {

res.writeHead(200);

res.end("My first server!");

};

4.

Create web server (by using createServer method) and assign it to a variable server

const server = http.createServer(requestListener);

5.

At the end of the page, listen server for any incoming requests and console a message for server running port

server.listen(port, host, () => {

console.log(`Server is running on http://${host}:${port}`);

});

6.

Run your server with “node server.js”.

In the console, we will see this output:

Server is running on http://localhost:8000

Open your browser on runnig port. It will show the following output:

My first server!

We’ve now set up a server and got our server response.

Step 2 — Serving HTML

7.

Modify the requestListener() function to return the appropriate Content-Type header for an HTML response. The Content-Type header is used to indicate the format of the data, also known as media type. Add this line of code:

res.setHeader("Content-Type", "text/html");

8.

Now, let’s return HTML content to the user.

res.end(`<html><body><h1>This is HTML</h1></body></html>`);

We will see “Server is running on <http://localhost:8000>” when your program has started.

Now go into the browser and visit http://localhost:8000.

Step 3 — Returning Different Types of Content

9.

Create a two new files called jsonServer.js. , csvServer.js.

Copy same code inside files as we did above steps.

10.

Let’s modify the requestListener() function to return the appropriate header all JSON responses. Add this line of code:

res.setHeader("Content-Type", "application/json");

11.

Modify end() method like this;

res.end(`{"message": "This is a JSON response"}`);

Run your server with “node jsonServer.js”. See the outputs.

12.

The *Comma Separated Values* (CSV) file format is a text standard that’s commonly used for providing tabular data.

In your csvServer.js file; add the following lines to our requestListener() function:

res.setHeader("Content-Type", "text/csv");

res.setHeader("Content-Disposition", "attachment;filename=samplescvfile.csv");

………

res.end(`id,name,email\n1,John Doe,doejohn@sample.com`);

If we go to http://localhost:8000 in our browser, a CSV file will be downloaded. Its file name will be samplecsvfile.csv

## Step 4 — Serving an HTML Page From a File

13.

We can serve HTML as strings in Node.js to the user, but it’s preferable that we load HTML files and serve their content.

Now, create a new file htmlServer.js and copy your server.js code inside it.

14.

To serve HTML files, we will load the HTML file with the [fs module](https://nodejs.org/api/fs.html" \l "fs_file_system) and use its data when writing our HTTP response. Let’s begin by importing the [fs module](https://nodejs.org/api/fs.html#fs_file_system).

const fs = require('fs').promises;

This module contains a readFile() function that we’ll use to load the HTML file in place.

15.

Modify requestListener() to read the file:

const requestListener = function (req, res) {

fs.readFile(\_\_dirname + "/index.html")

.then(contents => {

res.setHeader("Content-Type", "text/html");

res.writeHead(200);

res.end(contents);

}) };

16.

The fs.readFile() method can fail at times, so we should handle this case when we get an error. Add this to the requestListener() function

.catch(err => {

res.writeHead(500);

res.end(err);

return;

});

Run our server with the command: node htmlFile.js

In the web browser, visit http://localhost:8000. You will see the web page.

## Serving HTML Efficiently

17.

Instead of loading the HTML for every request, in this step we will load it once at the beginning. Begin by adding a new variable.

let indexFile;

18.

Now, readjust the requestListener() function.

const requestListener = function (req, res) {

res.setHeader("Content-Type", "text/html");

res.writeHead(200);

res.end(indexFile);

};

19.

Make the following changes as we create the server

fs.readFile(\_\_dirname + "/index.html")

.then(contents => {

indexFile = contents;

server.listen(port, host, () => {

console.log(`Server is running on http://${host}:${port}`);

});

})

.catch(err => {

console.error(`Could not read index.html file: ${err}`);

process.exit(1);

});

The code that reads the file is similar to what we wrote in our first attempt. However, when we successfully read the file we now save the contents to our global indexFile variable.

We’ve now created different web servers that return various types of data to a user.

## Step 4 — Managing Routes

Most websites we visit or APIs we use usually have more than one endpoint so we can access various resources. A good example would be a book management system.

Let’s create a new server for a small library, which will return two different types of data. If the user goes to our server’s address at /books, they will receive a list of books in JSON. If they go to /authors, they will receive a list of author information in JSON

20.

Now, create a new file routes.js and copy the code of json.js.

Open your server. In the browser “http://localhost:8000” you will see:

{"message": "This is a JSON response"}

After pressing “http://localhost:8000/todos” you will see the same result:

We have not built any special logic in our requestListener() function to handle a request for different URLs.

21.

Let’s begin by storing our JSON data in variables before the requestListener() function.

const books = JSON.stringify([

{ title: "The Alchemist", author: "Paulo Coelho", year: 1988 },

{ title: "The Prophet", author: "Kahlil Gibran", year: 1923 }

]);

const authors = JSON.stringify([

{ name: "Paulo Coelho", countryOfBirth: "Brazil", yearOfBirth: 1947 },

{ name: "Kahlil Gibran", countryOfBirth: "Lebanon", yearOfBirth: 1883 }

]);

22.

Now, we want to return the right JSON depending on the URL path the user visits. Let’s create a [switch statement](https://www.digitalocean.com/community/tutorials/how-to-use-the-switch-statement-in-javascript) on the request’s URL.

const requestListener = function (req, res) {

res.setHeader("Content-Type", "application/json");

switch (req.url) {}

}

23.

Let’s continue by adding a case for when the user visits web page.

case "/":

res.writeHead(200);

res.end(`{"message": "This is a JSON response"}`);

break

24.

Let’s add a case for when the user wants to get our list of books.

case "/books":

res.writeHead(200);

res.end(books);

break

25.

Now let’s add another case  for our authors.

case "/authors":

res.writeHead(200);

res.end(authors);

break

26.

We want to return an error if the user tries to go to any other path. Let’s add the default case to do this.

default:

res.writeHead(404);

res.end(JSON.stringify({error:"Resource not found"}));

Finally, test your server to see if it behaves as we expect.

**Congragulations…**