

***W2*** *PRACTICE*

Native HTTP and Manual Routing

# At the end of this practice, you can

* **Create** and run a native Node.js HTTP server
* **Manually implement** route handling using conditionals.
* Serve **static files** using fs.
* Parse form data from POST requests.
* Debug and enhance server code using console outputs.

# Get ready before this practice!

* **Read** the following documents to understand Nodejs built-in HTTP module: https://nodejs.org/api/http.html
* **Read** the following documents to understand Anatomy of an HTTP Transaction: https://nodejs.org/en/learn/modules/anatomy-of-an-http-transaction

# How to submit this practice?

* + Once finished, push your **code to GITHUB**
  + Join the **URL of your GITHUB** repository on LMS



*EXERCISE 1 –* ***ANALYZE***

### Goal

* Identify and fix the bug.
* Understand the request-response cycle in Node.js using the http module.
* Explain the role of res.write() and res.end() in sending data back to the client.

 For this exercise, you are provided with a minimal server.js file. Read and run the code and observe how it behaves.

// server.js

const http = require('http');

const server = http.createServer((req, res) => { res.write('Hello, World!');

return res.endd();

});

server.listen(3000, () => {

console.log('Server running on [http://localhost:3000');](http://localhost:3000/)

});

**Q1 –** What error message do you see in the terminal when you access

[http://localhost:3000?](http://localhost:3000/) What line of code causes it?

Syntax error or res.endd is not a function. The line that causes it is return res.endd

**Q2 –** What is the purpose of res.write() and how is it different from res.end()?

Write data to the response body, can be called many times. res.end will end the request and send a response, can optionally have content.

**Q3 –** What do you think will happen if res.end() is not called at all?

The server will always wait for the response causing an infinite loop and the request will be always be marked as pending.

**Q4 –** Why do we use http.createServer() instead of just calling a function directly?

Because we need to initialize the server that can listen for requests.

**Q5 –** How can the server be made more resilient to such errors during development?

By using try catch to catch errors that can happen when creating the server or other unexpected outputs.

*EXERCISE 2 –* ***MANIPULATE***

### Goal

* Practice using req.url and req.method.
* Understand how manual routing mimics what frameworks (like Express) automate.
* Serve both plain text and raw HTML manually.

 For this exercise you will start with a **START CODE (EX-2)**

**TASK 1 -** Update the code above to add custom responses for these routes:

|  |  |  |
| --- | --- | --- |
| **ROUTE** | **HTTP METHOD** | **RESPONSE** |
| /about | **GET** | About us: at CADT, we love  node.js! |
| /contact-us | **GET** | You can reach us vai email… |
| /products | **GET** | Buy one get one… |
| /projects | **GET** | Here are our awesome projects |

Use VS Code’s Thunder Client (or other tools (POSTMAN, INSOMIA) of your choice or curl on your terminal to make request.

Example output

curl <http://localhost:3000/about> >

About us: at CADT, we love node.js!

curl <http://localhost:3000/contact-us> -------------------------

-> You can reach us vai email…

**TASK 2 –** As we can see the complexitiy grow as we add more routes. Use switch statement to arrange the code into more organized structure.

### Reflective Questions

1. What happens when you visit a URL that doesn’t match any of the three defined?

* It will send an error page with a 404 not found response.

1. Why do we check both the req.url and req.method?

* Because we need to the see the link/url that they are trying to access and the methos because one url can have other methods.

1. What MIME type (Content-Type) do you set when returning HTML instead of plain text?

* Text/html

1. How might this routing logic become harder to manage as routes grow?

* The number of lines of code will expand and harder to track.

1. What benefits might a framework offer to simplify this logic?

* -less boiler plate code
* Enforces clear separation and structure
* Have an MVC architecture to separate models, from routes and the controller for logic.

*EXERCISE 3 –* ***CREATE***

### Goal

* Practice handling POST requests.
* Parse URL-encoded form data manually.
* Write and append to local files using Node.js' fs module.
* Handle async operations and errors gracefully.

 For this exercise you will start with a **START CODE EX-3**

**TASK 1 -** Extend your Node.js HTTP server to handle a **POST request** submitted from the contact form. When a user submits their name, the server should:

1. **Capture the form data** (from the request body).
2. **Log it to the console**.
3. **Write it to a local file** named submissions.txt.

Testing, go to /contact on broswer and test

## Requirements

* + Handle POST /contact requests.
  + Parse raw application/x-www-form-urlencoded data from the request body.
  + Write the name to a new line in submissions.txt.
  + Send a success response to the client (HTML or plain text).

## Discussion Questions

1. Why do we listen for data and end events when handling POST?

* Because we are working with data and the data comes in chunks so we have to liten for it until it ends and we can build a complete body.

1. What would happen if we didn’t buffer the body correctly?

* The data will be incomplete and malgformed leading to an exception when JSON.parse.

1. What is the format of form submissions when using the default browser form POST?

* application/x-www-form-urlencoded

1. Why do we use fs.appendFile instead of fs.writeFile?

* Because we want to add a new line to the file instead of overwriting it.

1. How could this be improved or made more secure?

* Validate the input, prevent code injection

## Bonus Challenge (Optional)

* + Validate that the name field is not empty before saving.
  + Send back a small confirmation HTML page instead of plain text.
  + Try saving submissions in JSON format instead of plain text.