

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


```

**Q1** – What error message do you see in the terminal when you access `http://localhost:3000`? What line of code causes it?

`TypeError: res.endd is not a function`  
`at Server.<anonymous> (E:\Doc\CADT-Y2T3\Backend\Lab`  
`2\StartCode\StartCode\ex1.js:3:94)`  
`at Server.emit (node:events:507:28)`  
`at parserOnIncoming (node:_http_server:1153:12)`  
`at HTTPParser.parserOnHeadersComplete (node:_http_common:117:17)`

The line that causes the issue is `return res.endd();`

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

`res.write()` sends part of the response to the client.

`res.end()` signals that you're finished sending the response.

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

If the `res.end()` is not called at all, the browser will keep waiting for a response that never finishes.

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

Because `http.createServer()` creates an HTTP server object that listens for requests, calls handler function and manage low-level HTTP protocols.

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

User try/catch blocks to handle runtime errors, use automatic restarters.

## 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 via 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 via email...
```

**TASK 2** – As we can see the complexity 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?  
*If the URL doesn't match, it will execute the default case in the switch block.*
2. Why do we check both the `req.url` and `req.method`?  
*req.url tells us which path the user requested.  
req.method tells us which type of HTTP method.*
3. What MIME type (`Content-Type`) do you set when returning HTML instead of plain text?  
*'Content-Type': 'text/html'*
4. How might this routing logic become harder to manage as routes grow?  
*The switch statement becomes long and hard to read, hard to debug the codes, hard to maintain the code.*
5. What benefits might a framework offer to simplify this logic?

*A framework helps by making code shorter, more organized, and easier to scale, especially when your project becomes bigger.*

## 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 browser 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 when a user submits a form the POST, the data is sent in chunks.**
2. What would happen if we didn't buffer the body correctly?  
**If we didn't buffer the body correctly, we will only get partial data or nothing from the data.**
3. What is the format of form submissions when using the default browser form POST?  
**`application/x-www-form-urlencoded`**
4. Why do we use `fs.appendFile` instead of `fs.writeFile`?  
**Because `fs.appendFile` adds new content to the end of the file without deleting what's already in there.**
5. How could this be improved or made more secure?

**This could be improved by using input validation, limit input size, file security, use HTTPS.**

## 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.