CADT IDT

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 – REVIEW and 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.

You are provided with a minimal `server.js` file. Read and run the code. 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');
});
```

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

Answer: The error message that display is “res.endd is not a function” and line 4 “res.endd()” is the cause.

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

Answer: res.write() is used to send a chunk of response message and it’s different from res.end() by res.write can be use multiple time but res.end() can only be used once.

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

Answer: If res.end() is not called then the request handler cannot be called and the HTTP response will not be finalized.

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

Answer: We use http.createServer() instead of just calling a function directly because it is used in Node.js to create a HTTP server instance and it serve multiple crucial purposes like Create a Server Object, Handles Networking, Event-Driven Request Handling, …

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

Answer: The server can be made more resilient to such errors during development by using error handler like try-catch, global error listeners, linting/TypeScript, logging, frameworks like Express, automated testing, nodemon, and pre-run validation to catch and recover from errors like res.endd().

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(https://www.thunderclient.com/) 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?
2. Why do we check both the `req.url` and `req.method`?
3. What MIME type (`Content-Type`) do you set when returning HTML instead of plain text?
4. How might this routing logic become harder to manage as routes grow?
5. What benefits might a framework offer to simplify this logic?

Answer:

1. If you visit a URL that doesn't match any of the three defined, it returns the 404 error message.
2. We check both the req.url and res.method because we want to ensure that the server respond the correct endpoint and HTTP method, adhering it to HTTP standards and enable precise routing.
3. When returning HTML instead of plain text, we use Content-Type: text/html.
4. The routing logic became harder to manage as routes grow when the switch statement become unwisely harder to manage, especially with additional methods, dynamic routes, or middleware.
5. The benefits from using framework offer to simplify this logic like by using Express simplify routing, automate Content-Type, support middleware, handle errors centrally, and provide a modular structure, making the code more maintainable and scalable.

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`?
2. What would happen if we didn't buffer the body correctly?
3. What is the format of form submissions when using the default browser form POST?
4. Why do we use `fs.appendFile` instead of `fs.writeFile`?
5. How could this be improved or made more secure?

Answer:

1. We listen for data and end events when handling POST because we needed to collect POST body chunks by using 'data' and process the complete body using 'end' due to streaming nature.
2. If we didn't use buffer the body correctly, it will leads to partial data, errors, client hangs, or resource leaks.
3. The format of form submission when using the default browser from POST by using application/x-www-form-urlencoded.
4. We use fs.appendFile instead of fs.writeFile because appendFile preserves previous data, suitable for logging submissions, while writeFile overwrites.
5. This can be improved or made more secure by using rate limit, use async/await, enhance responses, add CSRF protection, HTTPS, and secure file paths.

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.