HTTP Server Implementation

Lecture 05: Implementation of HTTP Server & Use of URL Module

Course: Advanced Web Technologies (CSC337)
Time: 2 Hours
Lecture Type: Theory

Learning Objectives:

By the end of this lecture, students should be able to:

1. Understand the fundamentals of HTTP servers.
2. Implement an HTTP server using Node.js.
3. Utilize the URL module for handling requests in Node.js.

---

1. Introduction to HTTP Server

What is an HTTP Server?

An HTTP server is software that listens for requests over HTTP and responds with web pages, data, or other resources. It enables client-server communication over the internet.

Key Components of an HTTP Server:

1. **Client (Browser or API Request)** - Sends an HTTP request.
2. **Server (Node.js, Apache, Nginx, etc.)** - Processes the request and sends back a response.
3. **Response** - Includes HTML, JSON, images, or any other data.

Real-Life Example:

Think of a restaurant:

- You (the client) order food (send a request).
- The waiter (server) processes your request and brings back food (response).

---

2. Implementing an HTTP Server in Node.js

Node.js provides a built-in `**http**` module to create an HTTP server.

Step-by-Step Implementation

1. **Load the HTTP Module:**

```
javascript

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

2. **Create a Server:**

```
javascript

const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World!');
});
```

3. **Start Listening on a Port:**

```
javascript

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

4. **Run the Server:**
   - Save the file as `**server.js**`.
   - Run using: `**node server.js**`.
   - Open browser and visit `**http://localhost:3000**`.

Explanation:

- `**http.createServer()**` creates a server instance.
- `**res.writeHead(200, { 'Content-Type': 'text/plain' })**` sets the response type.
- `**res.end('Hello, World!')**` sends the response and ends the connection.

---

3. Understanding the URL Module

The URL module in Node.js helps parse and manipulate URLs.

Example of URL Parsing

```
javascript

const url = require('url');

const myUrl = 'http://localhost:3000/?name=John&age=25';
const parsedUrl = url.parse(myUrl, true);
```

```
console.log(parsedUrl.query.name);  // Output: John
console.log(parsedUrl.query.age);   // Output: 25
```

How It Works?

- `url.parse()` breaks the URL into components.
- The `query` object contains parameters (`name=John`, `age=25`).

Practical Use Case:

When users enter data in a form (e.g., search query in Google), the server extracts and processes the information.

---

4. Handling Different Routes in HTTP Server

A basic server should be able to handle different routes dynamically.

Example: Creating Different Routes

```
javascript

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

const server = http.createServer((req, res) => {
    const parsedUrl = url.parse(req.url, true);
    if (parsedUrl.pathname === "/") {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end('<h1>Welcome to Home Page</h1>');
    } else if (parsedUrl.pathname === "/about") {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.end('<h1>About Us</h1>');
    } else {
        res.writeHead(404, { 'Content-Type': 'text/html' });
        res.end('<h1>404 - Page Not Found</h1>');
    }
});

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

Output:

- Visiting `http://localhost:3000/` shows **Welcome to Home Page**.
- Visiting `http://localhost:3000/about` shows **About Us**.
- Any other URL returns **404 - Page Not Found**.

---

5. Class Activity (30 mins)

Objective: Implement an HTTP server with at least three different routes.
Task:

1. Create a Node.js server that responds to:
    - `/` → “Welcome to My Website”

- `/contact` → "Contact us at info@example.com"
- `/services` → "Our services include Web Development, SEO, and Marketing"

2. Use the `http` and `url` modules.
3. Run the server and test the routes.

---

6. Summary & Discussion

Key Takeaways:

- An HTTP server listens for client requests and responds accordingly.
- Node.js provides the `http` module to create servers.
- The `url` module helps extract and manipulate URL data.
- Routing enables different responses based on request paths.

Discussion Questions:

1. How does the Node.js HTTP server differ from Apache or Nginx?
2. Why is URL parsing important in web applications?
3. What happens if a user requests a non-existent route?

---

7. Quiz (15 mins)

Q1: What module is used in Node.js to create an HTTP server?
A) `fs`
B) `http`
C) `path`
D) `os`

Q2: What does `res.writeHead(200, { 'Content-Type': 'text/html' })` do?
A) Closes the server
B) Sets response status and headers
C) Parses the URL
D) Logs a message to the console

Q3: What is the default HTTP port number?
A) 80
B) 8080
C) 443
D) 3000

Q4: What will `url.parse('http://example.com/?id=10', true).query.id` return?
A) `{ id: 10 }`
B) `undefined`
C) `10`
D) `example.com`

---

8. Assignment

Task:

- Implement a Node.js HTTP server with at least **5 different routes**.
- Use the `` `url` `` module to extract query parameters.
- Return different responses based on user input.
- Submit your code and output screenshots.

References:

1. Lim, G. (2019). *Beginning Node.js, Express & MongoDB Development*. Amazon.
2. Doglio, F. (2018). *REST API Development with Node.js*. APress.
3. Ethan Brown. (2019). *Web Development with Node and Express*. O'Reilly Publishing.

Next Lecture Preview:

- **Topic:** Node.js File System (FS) - Handling Files & Streams
- **Concepts:** Reading & Writing Files, Streams, and File Operations

---