**Nodejs basics BEE Syllabus**

1. **Intro to node js and web server**
2. **NPM modules Request and response lifecycle**
3. **End point and request methods**
4. **Creating routes in raw node**
5. **Read data from request**
6. **FS module**
7. **Intro to node js and web server:**

1. Import the 'http' module to create an HTTP server.

2. Import the 'url' module to parse URL and query parameters.

3. Create an HTTP server to handle incoming requests.

4. Parse the URL to extract query parameters:

- Parse the URL of the incoming request using the 'url' module.

- Extract the query parameters from the parsed URL object using `parsedUrl.query`.

5. Set the response HTTP headers with a 200 OK status and Content-Type text/plain (or other content types depending on the content being served).

6. Check whether the path part of the URL in the `req.url` variable starts with the string '/greet'. This is done using the `startsWith` method on the URL path, which is a common way to determine the route or endpoint being requested.

7. If the URL path starts with "/greet," do the following:

- Check if there is a "name" query parameter. If not, use the default value 'Guest.'

- Send a personalised greeting response in HTML format with inline CSS for styling.

8. If the URL path does not match "/greet," handle other routes or paths with an "Invalid route" response.

9. Specify the port (e.g., port 3000) where the server will listen for incoming requests.

10. Start the server and have it listen on the specified port.

You can access the '/greet' route with the specified query parameter using a URL like: `<http://localhost:3000/greet?name=Nischal>`.

code :

const http = require('http');

const url = require('url');

const server = http.createServer((req, res) => {

const parsedUrl1 = url.parse(req.url, true);

const queryParams = parsedUrl1.query;

res.writeHead(200, { 'Content-Type' : 'text/html' });

if (req.url.startsWith('/greet')){

const name = queryParams.name || 'Guest';

res.end(`

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<style>

body{

text-align:center;

margin-top: 50px;

}

.greeting{

color : red;

}

</style>

</head>

<body>

<div class="greeting">Welocme To ${name} BEE Training @ Chitkara </div>

</body>

</html>

`);

} else{

res.end('Invalid url or param given in the route\n');

}

});

const port = 3030;

server.listen(port, () => {

console.log(`Server is running on port ${port}`)

});

// <http://localhost:3030/greet?name=NischalAremanda>

**2. NPM modules Request and response lifecycle:**

1. You import the `http` module to create an HTTP server.

2. You create an HTTP server using the `http.createServer()` method. It takes a callback function with two arguments: `req` (the incoming request) and `res` (the response object).

3. Inside the request callback function, you handle different routes:

- For the root path `'/'`, you respond with a text/plain message.

- For the `/html` route, you respond with an HTML message.

- For the `/redirect` route, you send a 302 (Found) status code and redirect to the `/html` route.

4. You attach an event listener to the response object to log the request URL and response status code when the response is finished.

5. You handle unknown routes by checking if headers have been sent (i.e., if a response has already been initiated). If not, you respond with a 404 status code and a "Not Found" message.

6. Finally, you start the server and have it listen on port 4000. When the server starts, it logs a message to the console.

To test your server, you can open your web browser and visit the following URLs:

- <http://localhost:4000/> (It should display "Welcome to the root path!")

- <http://localhost:4000/html> (It should display an HTML message.)

- <http://localhost:4000/redirect> (It should redirect to the `/html` route.)

Code:

const http = require('http');

const server = http.createServer((req, res) => {

// Request event

// Example 1 -> send a basic response for the root path '/'

if (req.url === '/') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.write('This is "/" -> the root path of our URL');

res.end();

}

// Example 2 -> HTML response

else if (req.url === '/html') {

res.writeHead(200, { 'Content-Type': 'text/html' });

res.write('<html><body><h1>This is an HTML response coming from "/html" URL</h1></body></html>');

res.end();

} else if (req.url === '/redirect') {

res.writeHead(302, { 'Location': '[http://example.com](http://example.com/)' });

res.end();

}

});

// Event listener for the 'finish' event

server.on('request', (req, res) => {

res.on('finish', () => {

console.log(`Requested URL: ${req.url}`);

console.log(`Response HTTP Status Code: ${res.statusCode}`);

});

});

const port = 4000;

server.listen(port, () => {

console.log(`Server is Running Successfully on port ${port}`);

});

[12:14](https://chitkaraunive-n7r9809.slack.com/archives/C063DTFPLMA/p1699125277835789)

- <http://localhost:4000/> (It should display "Welcome to the root path!")

- <http://localhost:4000/html> (It should display an HTML message.)

- <http://localhost:4000/redirect> (It should redirect to the `/html` route.)

**3. End point and request methods:**

1. Import the `http` module to create an HTTP server.

2. Create an HTTP server using `http.createServer()`.

3. Extract the HTTP request method and request URL from the incoming request.

4. Handle different endpoints and request methods:

- If the method is `GET`, handle the following:

- For the root path `/`, respond with a 200 status code and a "GET request to the root path" message.

- For the `/info` endpoint, respond with a 200 status code and a "GET request to the /info endpoint" message.

- If the method is `POST` and the URL is `/create`, respond with a 201 status code and a "POST request to create a resource" message.

- If the method is `PUT` and the URL is `/update`, respond with a 200 status code and a "PUT request to update a resource" message.

- If the method is `DELETE` and the URL is `/delete`, respond with a 204 status code and a "DELETE request to delete a resource" message.

5. If the request method is not supported or the route is not found, respond with a 405 status code and a "Method Not Allowed" message.

6. Define the port where the server will listen (e.g., port 5000).

7. Start the server and have it listen on the specified port.

You can use tools like Postman to send requests to the following endpoints:

Handling a GET request to the root path ('/'):

<http://localhost:5000/>

Handling a GET request to the '/info' endpoint:

<http://localhost:5000/info>

Handling a POST request to the '/create' endpoint: <http://localhost:5000/create>

Handling a PUT request to the '/update' endpoint: <http://localhost:5000/update>

Handling a DELETE request to the '/delete' endpoint: <http://localhost:5000/delete>

Code:

// Import the http module to create an HTTP server.

const http = require('http');

// Create an HTTP server that will handle incoming requests.

const server = http.createServer((req, res) => {

// Extract the HTTP request method and the request URL.

const method = req.method;

const url = req.url;

// Handle different endpoints and request methods.

// If the request method is GET, handle the following:

if (method === 'GET') {

// Example 1: Handling a GET request for the root path '/'

// For the root path /, respond with a 200 status code and a "GET request to

// the root path" message.

if (url === '/') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('GET request to the root path.');

} else if (url === '/info') {

// Example 2: Handling a GET request for '/info'

// For the /info endpoint, respond with a 200 status code and a "GET request

// to the /info endpoint" message.

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('GET request to the /info endpoint.');

}

// If the request method is POST

} else if (method === 'POST') {

// Example 3: Handling a POST request

// respond with a 201 status code and a "POST request to create a resource"

// message.

if (url === '/create') {

res.writeHead(201, { 'Content-Type': 'text/plain' });

res.end('POST request to create a resource.');

}

// If the request method is PUT

} else if (method === 'PUT') {

// Example 4: Handling a PUT request

// /update, respond with a 200 status code and a "PUT request to update a

// resource" message.

if (url === '/update') {

res.writeHead(200, { 'Content-Type': 'text/plain' });

res.end('PUT request to update a resource.');

}

// If the method is DELETE

} else if (method === 'DELETE') {

// Example 5: Handling a DELETE request

// /delete, respond with a 204 status code and a "DELETE request to delete a

// resource" message.

if (url === '/delete') {

res.writeHead(204, { 'Content-Type': 'text/plain' });

res.end('DELETE request to delete a resource.');

}

} else {

// Handling unsupported methods or routes with a 405 response.

// If the request method is not supported or the route is not found, respond

// with a 405 status code and a "Method Not Allowed" message.

res.writeHead(405, { 'Content-Type': 'text/plain' });

res.end('Method Not Allowed');

}

});

// the server will listen on port no specified

const port = 5000;

// Start the server and listen on the specified port.

server.listen(port, () => {

console.log(`Server is listening on port ${port}`);

});

// use postman to check url output

// Handling a GET request to the root path ('/') and '/info'.

// <http://localhost:5000/>

// <http://localhost:5000/info>

// Handling a POST request to '/create'.

// <http://localhost:5000/create>

// Handling a PUT request to '/update'.

// <http://localhost:5000/update>

// Handling a DELETE request to '/delete'.

// <http://localhost:5000/delete>

**4. Creating routes in raw node:**

1. Import the `http` and `url` modules.

2. Create an HTTP server to handle incoming requests.

3. Parse the request URL using the `url.parse` method to extract the path.

4. Define different routes and their corresponding handlers:

- For the root path `'/'`, respond with a 200 status code and a "Welcome to the root route!" message.

- For the `/about` path, respond with a 200 status code and a "This is the about route." message.

- For the `/contact` path, respond with a 200 status code and a "Contact us at example@email.com" message.

- For any other path, respond with a 404 status code and a "Route not found" message.

5. Define the port (e.g., port 6000) where the server will listen.

6. Start the server and have it listen on the specified port.

You can access these routes by using the specified URLs:

- Root route: `<http://localhost:6000/>`

- About route: `<http://localhost:6000/about>`

- Contact route: `<http://localhost:6000/contact>`

- Route not found (for any other path): `<http://localhost:6000/anything>`

**5. Read data from request:**

1. Import the 'http' and 'url' modules to create an HTTP server and parse URLs and query parameters.

2. Create an HTTP server to handle incoming requests.

3. Parse the request URL to extract the path and query parameters:

- Use the 'url' module to parse the request URL.

- Extract the path using `parsedUrl.pathname`.

- Extract the query parameters using `parsedUrl.query`.

4. Define different routes and their corresponding handlers:

- For the root path `'/'`, respond with a 200 status code and a "Welcome to the root route!" message.

- For the '/about' path, respond with a 200 status code and a "This is the about route." message.

- For the '/contact' path, respond with a 200 status code and a "Contact us at example@email.com" message.

- For the '/data' path:

- If the request method is 'GET', respond with a 200 status code and the query parameters as JSON.

- If the request method is 'POST', read and parse the request body data as JSON, and respond with the parsed JSON data.

- Use the `req.on('data', ...)` event to read the request data in chunks.

- Use the `req.on('end', ...)` event to parse and respond with the request body data.

- Handle errors if the request body is not valid JSON.

5. If the URL path doesn't match any of the defined routes, respond with a 404 status code and a "Route not found" message.

6. Specify the port (e.g., port 7000) where the server will listen for incoming requests.

7. Start the server and have it listen on the specified port.

You can access these routes and send GET and POST requests using the provided URLs:

- Root route: `<http://localhost:7000/>`

- About route: `<http://localhost:7000/about>`

- Contact route: `<http://localhost:7000/contact>`

- Data route (GET request with query parameters): `<http://localhost:7000/data?param1=value1&param2=value2>`

- Data route (POST request with JSON data): `<http://localhost:7000/data>`

- Make sure your Node.js server is running on port 7000 to access these routes and send requests.

POSTMAN STEPS:

To send a POST request with JSON data using Postman, follow these steps:

1. Install Postman :

If you don't have Postman installed, you can download and install it from the [Postman website](https://www.postman.com/downloads/).

2. Open Postman :

Launch Postman on your computer.

3. Create a New Request :

- Click on the "New" button in the top-left corner.

- Click on "Request."

4. Name Your Request :

Give your request a name, for example, "POST Request with JSON Data."

5. Select the Request Type :

- In the request editor, select "POST" from the dropdown next to the URL field.

- Enter the URL of your server, for example, `http://localhost:7000/data`.

6. Set Headers :

- Click on the "Headers" tab.

- Click "Add a header."

- Enter the following header:

- Key: `Content-Type`

- Value: `application/json`

7. Add JSON Data :

- Click on the "Body" tab.

- Select "raw."

- Choose the type as "JSON (application/json)."

- Enter your JSON data in the text editor. For example:

json

{

"key1": "value1",

"key2": "value2"

}

8. Send the Request :

- Click the "Send" button to send the POST request with the JSON data.

9. View the Response :

- You will see the response from your server in the lower part of the Postman window.

- The response should contain the JSON data you sent.

That's it! You have sent a POST request with JSON data using Postman. Make sure your Node.js server is running on http://localhost:7000` as specified in your server code to receive the request.

**6. FS module:**

1. Import the 'http', 'url', and 'fs' modules to create an HTTP server, parse URLs and query parameters, and work with the file system.

2. Create an HTTP server to handle incoming requests.

3. Parse the request URL to extract the path and query parameters:

- Use the 'url' module to parse the request URL.

- Extract the path using `parsedUrl.pathname`.

4. Define different routes and their corresponding handlers:

- For the root path `'/'`, respond with a 200 status code and a "Welcome to the root route!" message.

- For the '/about' path, respond with a 200 status code and a "This is the about route." message.

- For the '/contact' path, respond with a 200 status code and a "Contact us at example@email.com" message.

- For the '/readfile' path, read and respond with the contents of a file using the 'fs' module:

- Use `fs.readFile` to read the contents of 'sample.txt'.

- If the file is not found, respond with a 404 status code and a "File not found" message.

- If the file is found, respond with a 200 status code and the file's contents.

- For the '/writefile' path, write data to a file using the 'fs' module:

- Use `fs.writeFile` to write data to 'sample.txt'.

- If there is an error while writing, respond with a 500 status code and an "Error writing to the file" message.

- If the write operation is successful, respond with a 200 status code and a "Data written to file successfully" message.

5. If the URL path doesn't match any of the defined routes, respond with a 404 status code and a "Route not found" message.

6. Specify the port (e.g., port 8000) where the server will listen for incoming requests.

7. Start the server and have it listen on the specified port.

Root route: <http://localhost:8000/>

About route: <http://localhost:8000/about>

Contact route: <http://localhost:8000/contact>

Read file route: <http://localhost:8000/readfile>

Write file route: <http://localhost:8000/writefile>

You can access these routes, which include reading from and writing to a file, and send requests using the provided URLs while ensuring the Node.js server is running on port 8000.