SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC-UGC with 'A' Grade

Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

Course Code and Name : 19TS601 FULL STACK DEVELOPMENT

Unit 3 : NODEJS AND EXPRESS

Topic : Node.js Web server

NodeJS web server

- A NodeJS web server is a server built using NodeJS to handle HTTP requests and responses.
- Unlike traditional web servers like Apache or Nginx, which are primarily designed to give static content.
- NodeJS web servers can handle both static and dynamic content while supporting real-time communication.
- It uses JavaScript (the same language used for client-side scripting) on the server side, making it a popular choice for full-stack developers.

Setting Up a NodeJS Web Server

Step 1:

- If you haven't installed NodeJS in your system,to Install NodeJS
- To verify the installation, open your terminal or command prompt and type:

```
node –v
```

This will display the installed NodeJS version.

Step 2: Create Your Project Directory

- Create a new directory for your project and navigate into it:
- mkdir node-server
- cd node-server

Step 3: Initialize the Project

Create a package.json file, which contains metadata about your project:

npm init –y

Step 4: Create a Basic Server

Create a file named server.js

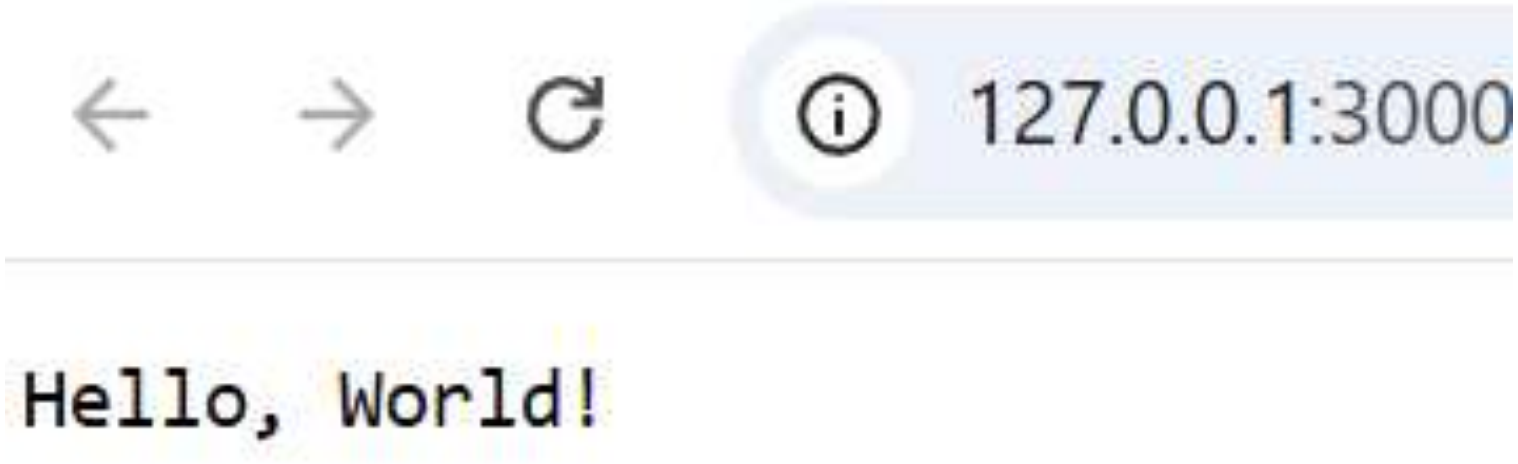
Open server.js in your code editor and add the following code to create a simple HTTP server:

```
const http = require('http');
const server = http.createServer((req, res) => {
    res.writeHead(200, { 'Content-Type': 'text/plain' });
    res.end('Hello, World!');
});
const port = 3000;
const host = 'localhost';
server.listen(port, host, () => {
    console.log(`Server running at http://${host}:${port}/`);
});
```

- Output

Run the server by using the below command

node server.js



In this example

- **http.createServer():** Creates an HTTP server that listens for requests.
- **res.writeHead():** Sends a response header with the status code 200 (OK).
- **res.end():** Ends the response and sends the message “Hello, World!” to the client.
- **server.listen():** Starts the server on the specified host (localhost) and port (3000).

Why Use NodeJS for Web Servers?

1. High Performance
2. Single Language Stack
3. Scalability
4. Real-Time Data
5. Large Ecosystem

1.High Performance

- NodeJS is designed for speed. Its non-blocking I/O model and the V8 engine allow for fast execution of JavaScript code, making it ideal for handling high-concurrency scenarios, such as APIs or real-time applications.

2. Single Language Stack

- NodeJS enables developers to use JavaScript both on the client side and server side.
- This eliminates the need to switch between different programming languages and allows for better integration between the front-end and back-end.

3. Scalability

- NodeJS is scalable by design due to its event-driven architecture.
- It can easily handle more requests with minimal overhead by adding more processes or utilizing a load balancer.

4. Real-Time Data

- NodeJS is especially well-suited for applications that require real-time data, such as chat applications, live notifications, and collaborative platforms.
- Its WebSocket support allows for full-duplex communication between the client and server.

5. Large Ecosystem

- NodeJS has an extensive ecosystem with over a million open-source libraries available through npm.
- These libraries make it easy to add functionality such as authentication, routing, data handling, and much more.

Benefits of Using a NodeJS Web Server

- **Real-Time Capabilities:** NodeJS can be used in the real-time applications like chat apps or live data streaming due to its non-blocking architecture and event-driven model.
- **High Performance:** The V8 engine provides fast execution of JavaScript, making NodeJS a great choice for applications requiring high performance and low latency.
- **Scalability:** NodeJS is highly scalable, thanks to its event loop and non-blocking I/O. It can handle a large number of simultaneous requests efficiently.
- **Lightweight and Efficient:** With its single-threaded, non-blocking nature, NodeJS reduces the overhead of traditional multi-threaded servers, making it lightweight and ideal for handling concurrent requests.

How Does a NodeJS Web Server Work?

- **Event Loop:** The server listens for incoming requests and processes them in a non-blocking way.
- When a request comes in, it is passed to the event loop, which decides how to handle it.
- **Non-Blocking I/O:** While the server is processing a request (like querying a database or reading from a file), it does not block other incoming requests.
- This is handled asynchronously through callbacks, promises, or async/await syntax.

- **Request and Response**: Once the event loop processes the request, the server sends a response back to the client (browser, API consumer, etc.). The response could be in HTML, JSON, or other data format.
- **Web Servers**: NodeJS can serve static files (like images or stylesheets) and dynamic content (like API responses) using built-in modules like http, fs, and path.

Routing in NodeJS Web Server

In real-world applications, your server needs to handle different routes (URLs) and respond differently based on the request.

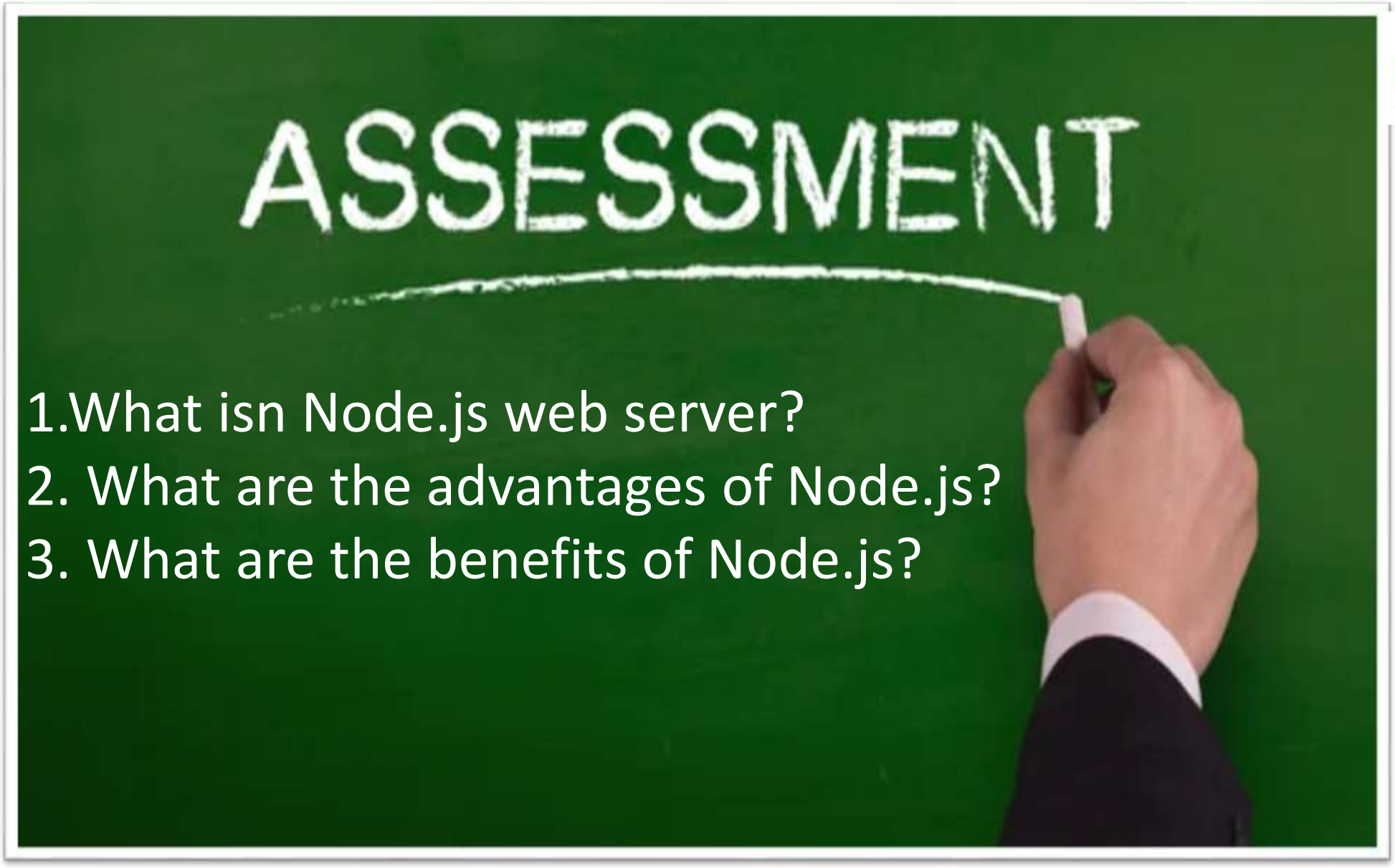
```
// server.js
const http = require('http');
const hostname = '127.0.0.1';
const port = 3000;
const server = http.createServer((req, res) => {
    if (req.url === '/') {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('Welcome to the Homepage!');
    } else if (req.url === '/about') {
```

```
res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('This is the About Page');
    } else if (req.url === '/contact') {
        res.statusCode = 200;
        res.setHeader('Content-Type', 'text/plain');
        res.end('This is the Contact Us Page');
    }
    else {
        res.statusCode = 404;
```

```
 res.setHeader('Content-Type', 'text/plain');
        res.end('Page Not Found');
    }
});
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
});
```

In this example

- The http.createServer() method initializes the server.
- res.writeHead() sends a response header with the status code (e.g., 200 OK) and any other headers.
- res.statusCode sets the HTTP status code for the response. In this case, 200 means the request was successful.
- res.setHeader() sets the content type of the response. In this case, text/plain means the server is sending plain text back to the client.
- server.listen() binds the server to the specified port and hostname (in this case, localhost:3000).
- When the server successfully starts, the callback function is executed, logging a message (Server running at http://127.0.0.1:3000/) to the console.


ASSESSMENT
1.What isn Node.js web server?
2. What are the advantages of Node.js?
3. What are the benefits of Node.js?

Text Book:

1.Pro MERN Stack, Full Stack Web App Development with Mongo, Express, React, and Node, Vasan Subramanian, A Press Publisher, 2019.

Reference:

David Flanagan, “Java Script: The Definitive Guide”, O’Reilly Media, Inc, 7 th Edition, 2020

2. Matt Frisbie, “Professional JavaScript for Web Developers” Wiley Publishing, Inc, 4th Edition, ISBN: 978-1-119-36656-0, 2019


Thank
You!