

**DAY 1: Node.js (Theory-Based Questions)**

NAME - SOHAM SARKAR

COLLEGE NAME – MCKV INSTITUTE OF ENGINEERING



June 6, 2025

### **🟡 ASSIGNMENT – DAY 1: Node.js (Theory-Based Questions)**

#### **✅ Q1: What is Node.js?**

Explain what Node.js is. Mention its key features and why it is popular for backend development.

**Node.js** is an open-source, cross-platform **JavaScript runtime environment** that allows developers to run JavaScript code **outside the browser**, typically on the **server-side**. It is built on **Google Chrome's V8 JavaScript engine**, which makes it extremely fast in execution.

1. **Asynchronous and Event-Driven:**
   * All APIs in Node.js are non-blocking (asynchronous).
   * It uses an event-driven architecture, which makes it efficient and suitable for I/O-heavy operations like reading files, databases, or APIs.
2. **Single-Threaded but Highly Scalable:**
   * Uses a single-threaded event loop model to handle multiple clients concurrently without creating new threads for each request.
3. **Fast Execution:**
   * Powered by the V8 engine, Node.js converts JavaScript to machine code quickly, making it very fast.
4. **NPM (Node Package Manager):**
   * Comes with the largest ecosystem of open-source libraries through **npm**, which simplifies development and speeds up the process.
5. **Cross-Platform:**
   * Runs on Windows, macOS, and Linux, making it widely accessible.
6. **Real-time Capabilities:**
   * Perfect for real-time applications like chat apps, gaming, and collaborative tools due to its WebSocket and event-based model.

**Why Node.js is Popular for Backend Development:**

* **Uses JavaScript** on both frontend and backend, allowing full-stack development with one language.
* **Lightweight and Fast**, ideal for microservices and APIs.
* **Scalability**, suitable for modern, high-traffic applications like Netflix, LinkedIn, Uber.
* **Vibrant Community**, regular updates, and extensive documentation.

#### **✅ Q2: How does a basic Node.js server work?**

Describe the process of how Node.js creates an HTTP server. What modules are required? How does it handle requests and responses?

A **basic Node.js server** works by using the built-in http module to **listen for incoming HTTP requests**, **process** them, and **send responses** back to the client.

**Required Module:**

* **http module**: This core module is used to create the server and handle HTTP requests and responses.

**🛠️ Basic Steps in Creating a Node.js HTTP Server:**

1. **Import the http module**
2. **Create the server using http.createServer()**
3. **Provide a callback function to handle requests and send responses**
4. **Listen on a specific port using server.listen()**

* // Step 1: Import the http module
* const http = require('http');
* // Step 2: Create server
* const server = http.createServer((req, res) => {
* // Step 3: Handle incoming request
* res.writeHead(200, { 'Content-Type': 'text/plain' }); // Set response header
* res.write('Hello, World!'); // Send response
* res.end();
* });
* // Step 4: Server listens on port 3000
* server.listen(3000, () => {
* console.log('Server running at http://localhost:3000/');
* });

#### **✅ Q3: What is the difference between http.createServer() and using Express.js?**

Explain the difference between building a server with the core http module vs using the Express framework. What are the advantages of Express?

**Core http Module vs Express.js**

| **Feature** | **http.createServer() (Core Node.js)** | **Express.js (Framework)** |
| --- | --- | --- |
| **Level** | Low-level | High-level abstraction over http module |
| **Routing** | Manual routing with if-else or switch | Built-in routing with app.get(), app.post() etc. |
| **Middleware Support** | Not built-in, must be implemented manually | Rich middleware support (e.g., body-parser, cors) |
| **Code Simplicity** | More boilerplate | Cleaner and shorter code |
| **Static Files Serving** | Needs custom implementation | Built-in with express.static() |
| **Community & Ecosystem** | Smaller, mostly Node.js core utilities | Huge community, thousands of middleware & extensions |
| **Learning Curve** | Good for learning raw Node.js | Easier for building real-world apps quickly |

**Advantages of Using Express.js**

* ✅ **Simplified routing**
* ✅ **Built-in middleware support**
* ✅ **Cleaner, modular code**
* ✅ **Faster development with less boilerplate**
* ✅ **Massive ecosystem** (e.g., session handling, authentication, file uploads)

#### **✅ Q4: How can you serve static HTML and CSS files using Node.js?**

Describe the process of serving static HTML and CSS files using both:

* Core Node.js (http and fs modules), and
* Express framework (express.static()).

**1. Core Node.js (http + fs modules)**

**Required Modules:**

* http – to create the server.
* fs – to read HTML and CSS files.
* path – to resolve file paths.

### **📁 Folder Structure:**

**project/**

**├── index.html**

**├── style.css**

**└── server.js**

**server.js Example:**

* const http = require('http');
* const fs = require('fs');
* const path = require('path');
* const server = http.createServer((req, res) => {
* if (req.url === '/') {
* // Serve HTML file
* fs.readFile(path.join(\_\_dirname, 'index.html'), (err, data) => {
* if (err) {
* res.writeHead(500);
* return res.end('Error loading HTML');
* }
* res.writeHead(200, { 'Content-Type': 'text/html' });
* res.end(data);
* });
* } else if (req.url === '/style.css') {
* // Serve CSS file
* fs.readFile(path.join(\_\_dirname, 'style.css'), (err, data) => {
* if (err) {
* res.writeHead(500);
* return res.end('Error loading CSS');
* }
* res.writeHead(200, { 'Content-Type': 'text/css' });
* res.end(data);
* });
* } else {
* // 404 Not Found
* res.writeHead(404);
* res.end('Page not found');
* }
* });
* server.listen(3000, () => {
* console.log('Server running on http://localhost:3000');
* });

**2. Using Express.js (express.static)**

### **📁 Folder Structure:**

project/

├── public/

│ ├── index.html

│ └── style.css

└── app.js

**app.js Example:**

* const express = require('express');
* const app = express();
* // Serve static files from "public" folder
* app.use(express.static('public'));
* app.listen(3000, () => {
* console.log('Server running on http://localhost:3000');
* });

Now visit http://localhost:3000 to view index.html  
and http://localhost:3000/style.css to view the CSS file.

#### **✅ Q5: What is the role of middleware in Express.js?**

Define middleware in Express.js. What is its purpose? Give examples of commonly used middleware.

**Middleware** in Express.js is a **function** that has access to the:

* req (request object)
* res (response object)
* next (function to pass control to the next middleware)

It **sits between the request and the response** and can:

* Modify request or response objects
* End the request-response cycle
* Call the next middleware in the stack

**Syntax of Middleware:**

* function middlewareName(req, res, next) {
* // Logic or modifications
* next(); // Pass control to the next middleware
* }

Use it with:

* app.use(middlewareName);

**Purpose of Middleware:**

* Execute any code before request reaches route handler
* Modify request/response objects
* Perform tasks like authentication, logging, error handling
* Organize and simplify code

### **Examples of Common Middleware:**

| **Middleware** | **Purpose** |
| --- | --- |
| express.json() | Parses incoming JSON payloads |
| express.urlencoded() | Parses URL-encoded form data |
| express.static() | Serves static files (HTML, CSS, images, etc.) |
| Custom Middleware | Logging, authentication, validation, etc. |
| cors | Enables Cross-Origin Resource Sharing |
| morgan | Logs HTTP requests to console |
| helmet | Secures app by setting HTTP headers |
| cookie-parser | Parses cookies attached to the client request |