Node.js - Day 3: Core Modules (fs, path, http) → File Operations

■ Learning Objectives

- Understand what core modules are in Node.js.
- Import and use built-in modules like fs, path, and http.
- Perform file operations such as reading, writing, appending, and deleting files.
- Build a simple HTTP server serving static text or HTML.

■ Session 1 (45 min) — Understanding Core Modules

1. What Are Core Modules?
Node.js includes several built-in modules available without installing anything.
Common examples:
• fs — File System: work with files and directories
• path — Handle and format file paths
• http — Create web servers
• os — Get system info (CPU, memory, etc.)
You can load a core module using **require()**:
const fs = require('fs'); const path = require('path'); const http = require('http');

■ The fs (File System) Module

Read a File:
const fs = require('fs'); fs.readFile('message.txt', 'utf8', (err, data) => { if (err) return console.error(err); console.log("File content:", data); }); **Write a File:**
fs.writeFile('output.txt', 'Hello Node.js!', (err) => { if (err) throw err; console.log('File created successfully.'); }); **Append to a File:**
fs.appendFile('output.txt', '\nAppended text.', (err) => { if (err) throw err; console.log('Text appended successfully.'); }); **Delete a File:**
fs.unlink('output.txt', (err) => { if (err) throw err; console.log('File deleted.'); });

■■ The path Module

Helps handle file and folder paths safely across platforms.
const path = require('path'); console.log(path.basename(__filename)); // file name console.log(path.dirname(__filename)); // directory console.log(path.extname(__filename)); // extension console.log(path.join(__dirname, 'files', 'data.txt')); // safe path **Why use path.join?** Different OS use different slashes (\ vs /), and path.join() automatically corrects them.

■ Session 2 (45 min) — Using http + File Operations Project

The http Module
Node can act as a web server with no external software.
const http = require('http'); const server = http.createServer((req, res) => { res.writeHead(200, {'Content-Type': 'text/plain'}); res.end('Hello from Node.js server!'); }); server.listen(3000, () => { console.log('Server running at http://localhost:3000'); });

■ Serving File Content with fs + http

Combine both modules to serve HTML files:
const http = require('http'); const fs = require('fs'); http.createServer((req, res) => { fs.readFile('index.html', (err, data) => { if (err) { res.writeHead(404); res.end('File not found!'); } else { res.writeHead(200, {'Content-Type': 'text/html'}); res.end(data); } }); }).listen(3000, () => console.log('Server running...')); **index.html Example:**
Welcome to my Node.js Server! This content is served from a file.

■ Mini Project — File Logger App

Create a Node app that:
1. Asks the user for a message using readline.
2. Appends the message to log.txt with current time.
3. Displays a success message.
const fs = require('fs'); const readline = require('readline'); const rl = readline.createInterface({ input: process.stdin, output: process.stdout }); rl.question('Write a message: ', (msg) => { const log = `${new Date().toLocaleString()}: ${msg}\n`; fs.appendFile('log.txt', log, (err) => { if (err) throw err; console.log('Message saved successfully!'); rl.close(); }); });

■ Recap Discussion

- What are core modules and why are they useful?
- Difference between synchronous and asynchronous fs methods?
- How does http.createServer() work?
- Where could this be used in real-life projects?

■ Homework

Create a script that:
- Reads a file named todo.txt
- Adds a new line (a task) every time it runs
- Displays the full file content in the console
Example output:
- Buy groceries - Finish Node.js homework - Prepare slides for tomorrow