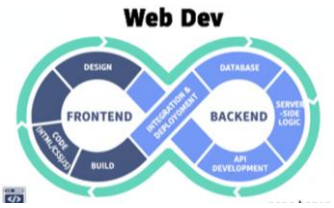
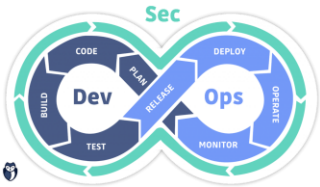




Day 42

"Web Development + Security"

Backend, Node.js & npm:

What is Backend? (Basic Definition)

- The backend is the server-side part of a web application.
- It handles logic, databases, authentication, APIs, and communication with the frontend.
- Example: When you submit a form, backend code processes and stores the data in a database.

Frontend (client) → sends request → Backend (server) → sends response back

What is Node.js?

- Node.js is a JavaScript runtime environment that lets you run JS outside the browser (like on your computer or server).
- It uses Google Chrome's V8 engine internally.
- With Node.js, you can build:
  - Web servers
  - APIs
  - Command-line tools
  - Real-time apps (like chat or streaming)

What is npm (Node Package Manager)?

- npm is the default package manager for Node.js.
- It lets you install, update, or manage external libraries (modules) for your project.
- Every Node project uses a file called package.json — which stores project details and dependencies.

Difference between running JS in the browser console vs. in Node.js (terminal):

Environment Difference

Feature	**Browser Console (Frontend JS)**	**Node.js (Backend JS)**
Environment	Runs inside a web browser (like Chrome)	Runs inside the Node.js runtime on your computer/server
Scope	Attached to the window object	Attached to the global object
APIs available	Has access to DOM, window, document, alert()	Has access to file system (fs), path, http, process)
Use case	Used for frontend logic, testing, UI manipulation	Used for backend logic, servers, APIs, file handling

Now, let's see a basic example where we used node to run the javascript code:

Server.js:

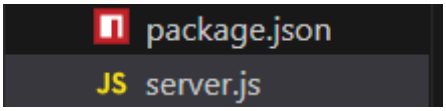
```
JS server.js
1   console.log("Hey");

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS E:\FullStackDevelopment\Day41-50\Day42> node .\server.js
Hey
```

Now, we are initialising the npm in this: for which we will use "npm init". Following text will come in the terminal, fill as per your wish:

```
PS E:\FullStackDevelopment\Day41-50\Day42> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (day42)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS E:\FullStackDevelopment\Day41-50\Day42> npm init
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (day42) test-npm
version: (1.0.0)
description: This is created just for testing the package creation on the npm
entry point: (01_solution.js)
test command:
git repository:
keywords: test-npm npm test
author: Aditya
license: (ISC)
About to write to E:\FullStackDevelopment\Day41-50\Day42\package.json:

{
  "name": "test-npm",
  "version": "1.0.0",
  "description": "This is created just for testing the package creation on the npm",
  "main": "01_solution.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "keywords": [
    "test-npm",
    "npm",
    "test"
  ],
  "author": "Aditya",
  "license": "ISC"
}


Is this OK? (yes)
```

Write yes to confirm: following package.json will automatically get generated.



We can see the content of it:

```
FOL...
~$Day42.docx
~WRL0997.tmp
01_solution.js
07_solution.html
Day42.docx
package.json
server.js

01_solution.js   07_solution.html   server.js   package.json
package.json > ...
1  {
2    "name": "test-npm",
3    "version": "1.0.0",
4    "description": "This is created just for testing the package creation on the npm",
5    "main": "01_solution.js",
       Debug
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "start": "node server.js"
9    },
10   "keywords": [
11     "test-npm",
12     "npm",
13     "test"
14   ],
15   "author": "Aditya",
16   "license": "ISC"
17 }
```

Now, installing packages using the npm: npm install <package-name>

For example:

```
PS E:\FullStackDevelopment\Day41-50\Day42> npm install slugify

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
PS E:\FullStackDevelopment\Day41-50\Day42>
```

Clearly, we can see this in the folder:

```
node_modules
  slugify
  .package-lock.json
```

CommonJs Vs EcmaScript Modules:

What Are Modules in JavaScript?

A module means splitting code into multiple files, so each file handles one specific part — making code reusable and organized.

Example idea:

- math.js → handles calculations
- server.js → runs server logic

In JS, there are two main module systems:
-> CommonJS (CJS) — older, used in Node.js (by default)
-> ECMAScript Modules (ESM) — newer, standard in modern JS

CommonJS (CJS)

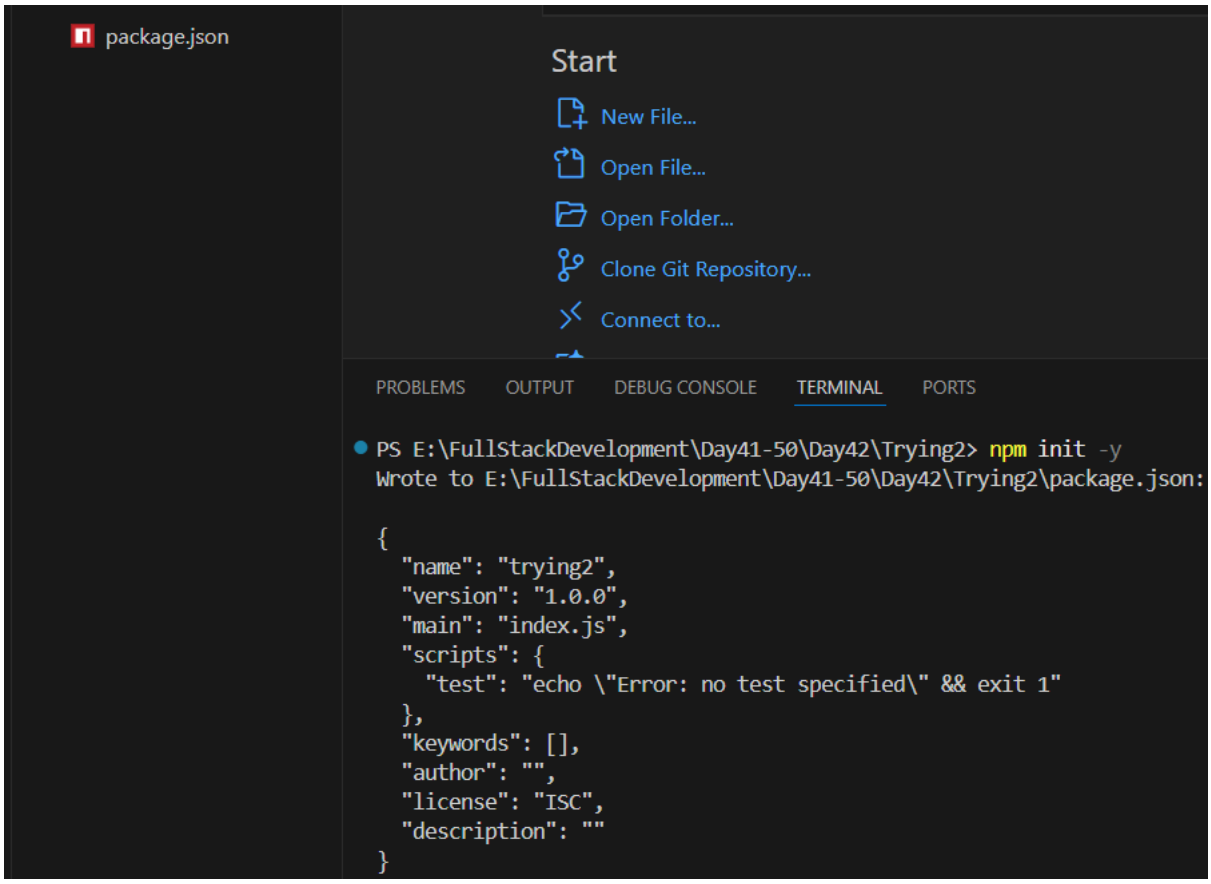
- Used in: Node.js (default)
- File extension: .js
- Uses: require() to import, module.exports to export

ECMAScript Modules (ESM)

- Used in: Modern JavaScript (Frontend + Node 14+)
- File extension: .mjs (or .js with "type": "module" in package.json)
- Uses: import and export keywords

Let’s start from basics:

We can do this to get package.json without specifying the questions: clearly, package.json gets created.



Now, creating a basic server using the node.js: just copy paste the code

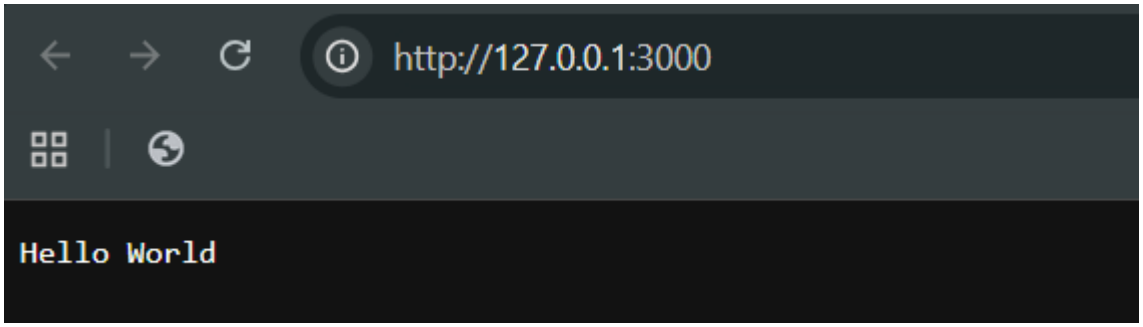
Main.js:

```
JS main.js > server > createServer() callback
const { createServer } = require('node:http');

const hostname = '127.0.0.1';
const port = 3000;

const server = createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Terminal:

```
PS E:\FullStackDevelopment\Day41-50\Day42\Trying2> node main.js
Server running at http://127.0.0.1:3000/
```

Output:



Now, let’s make some changes in the code: just add <h1> there

Main.js:

```
JS main.js > server > createServer() callback
1   const { createServer } = require('node:http');
2
3   const hostname = '127.0.0.1';
4   const port = 3000;
5
6   const server = createServer((req, res) => {
7     res.statusCode = 200;
8     res.setHeader('Content-Type', 'text/html');
9     res.end('<h1>Hello World</h1>');
10  });
11
12  server.listen(port, hostname, () => {
13    console.log(`Server running at http://${hostname}:${port}/`);
14  });
```

Output: