Sheryians Coding School

Node.js & Backend Basics

Node.js & Backend Basics

1. How to Run JavaScript Outside the Browser

Till now, you have only run JavaScript inside the browser. But JavaScript is **not a browser-only language**.

Node.js allows us to run JavaScript directly on our computer, without Chrome, without HTML, without React.

That means:

- JavaScript can create servers
- JavaScript can talk to databases
- JavaScript can run scripts
- JavaScript can power backend logic

Technical Definition

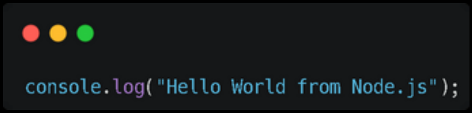
Node.js is a **JavaScript runtime environment** that allows JavaScript to run outside the browser using the V8 engine.

Steps to Run a Script

1. **Install Node.js**
   - Download from official site
   - Verify installation:

```
node -v
```

2. **Create a JavaScript file**
   - File name can be anything
   - Example: `app.js`
3. **Write JavaScript code**



```
console.log("Hello World from Node.js");
```

4.**Open terminal in that folder**
5.**Run the file**

```
node app.js
```

6. Output appears in the terminal (not browser)

Common Pitfalls

❌ Node not installed properly
❌ Running command from wrong folder
❌ Typo in file name
❌ Expecting browser APIs like `window` or `document` (they don't exist in Node)

Interview Questions

- What is Node.js?
- Can JavaScript run without a browser?
- Why do we need Node.js?
- What is the difference between browser JS and Node.js JS?

Optional Tasks

- Run a file that prints your name and age
- Create a file that adds two numbers and logs the result

2. What Are Packages?

A **package is code that you didn't write**.

Some other developer wrote useful code, made it public, and said:

> "Use this instead of writing everything from scratch."

Examples:

- Sending emails
- Encrypting passwords
- Creating servers
- Handling file uploads

You **borrow** this code.

Where Are Packages Published?

Packages are published on **npmjs.com**.

Think of npm as:

- A **store** of JavaScript code
- Where developers share reusable logic

Node.js & Backend Basics

Technical Definition

A package is a reusable block of JavaScript code published on npm that can be installed and used in a Node.js project.

How to Install Packages

Steps

1. Initialize a project

```
npm init -y
```

2. This creates:
- `package.json`

3. **Install a package**

```
npm install package-name

    Example:
    npm install express
```

What Happens Internally?

- Package code goes into:
  - node_modules/
- `package.json`
  - Tracks which packages you installed
  - Tracks versions
- `package-lock.json`
  - Tracks **dependencies of your dependencies**
  - Ensures same install across systems

Important Files

- **node_modules** → actual package code
- **package.json** → your project's dependency list
- **package-lock.json** → exact dependency tree

Common Pitfalls

❌ Deleting `package-lock.json`
❌ Manually editing versions randomly
❌ Pushing `node_modules` to GitHub
❌ Not running `npm install` after cloning project

Interview Questions

- What is npm?
- What is a package?
- Difference between package.json and package-lock.json?
- Why should we not push node_modules?

Optional Tasks

- Initialize a project
- Install any package
- Delete node_modules and reinstall using `npm install`

2.2 How to Use Packages?

Steps

1. Install the package
2. Import it in your file

Example:

```
const express = require("express");
```

3. Use its functionality

Key Rule

You **cannot use a package without installing it first.**

Common Pitfalls

❌ Forgetting to install the package
❌ Wrong import syntax
❌ Version mismatch errors

Common Pitfalls

❌ Deleting `package-lock.json`
❌ Manually editing versions randomly
❌ Pushing `node_modules` to GitHub
❌ Not running `npm install` after cloning project

Interview Questions

- What is npm?
- What is a package?
- Difference between package.json and package-lock.json?
- Why should we not push node_modules?

Optional Tasks

- Initialize a project
- Install any package
- Delete node_modules and reinstall using `npm install`

2.2 How to Use Packages?

Steps

1. Install the package
2. Import it in your file

Example:

```
const express = require("express");
```

3. Use its functionality

Key Rule

You **cannot use a package without installing it first.**

Common Pitfalls

❌ Forgetting to install the package
❌ Wrong import syntax
❌ Version mismatch errors

2.3 What Is a Server?

A **server is a program that listens for requests and sends responses.**

Client asks:
> "Give me data"

Server replies:
> "Here is the data"

Browser, mobile apps, Postman → all are **clients**.

Technical Definition

A server is a software application that listens on a network port and handles incoming HTTP requests by sending responses.

Why Do We Need Servers?

- To store data
- To authenticate users
- To connect frontend with database
- To apply business logic

Interview Questions

- What is a server?
- Difference between client and server?
- Can frontend act as a server?

2.4 Create a Server Using Express

Why Express?

Writing servers using plain Node.js is **painful and verbose**.

Express:

- Simplifies server creation
- Handles routing
- Handles middleware

Node.js & Backend Basics

Steps to Create Server

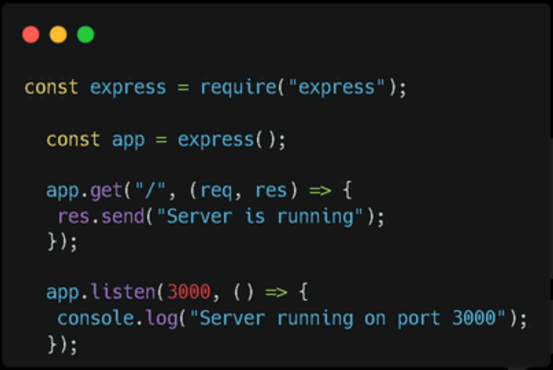
1. Initialize project

```
npm init -y
```

2. Install Express

```
npm install express
```

3. Create `index.js`



```
const express = require("express");

  const app = express();

  app.get("/", (req, res) => {
   res.send("Server is running");
  });

  app.listen(3000, () => {
   console.log("Server running on port 3000");
  });
```

4. Run server

```
node index.js
```

5. Open browser:

```
http://localhost:3000
```

What Is Happening?

- `app.get` → route
- `/` → endpoint
- `req` → request from client
- `res` → response from server
- `listen` → starts server

Node.js & Backend Basics

Common Pitfalls

❌ Port already in use
❌ Forgetting to restart server
❌ Syntax errors crashing server
❌ Using browser-only APIs

Interview Questions

- What is Express?
- Why Express over Node HTTP module?
- What is a route?
- What does `app.listen` do?

Optional Tasks

- Change port number
- Add one more route (`/about`)
- Return JSON instead of text