NODE js

Audience

The audience for Node.js is mainly developers (JavaScript, backend, full-stack) working on real-time, scalable web applications and startups or small businesses looking to build fast, lightweight, and scalable services.

Pre-requisites

To get started with Node.js, you should have:

1. A solid understanding of JavaScript.
2. Basic web development knowledge (HTML, CSS, HTTP).
3. Familiarity with CLI commands and npm.
4. Understanding of asynchronous programming.
5. Optionally, knowledge of Git, databases, and frameworks like Express.js.

About Node.js

Node.js is a JavaScript runtime built on Chrome's V8 engine that allows you to run JavaScript code outside the browser

Executing Node.js

```
node <filename.js>
```

Features of Node.js

Asynchronous and Event-Driven: Handles multiple operations concurrently without blocking the execution.

Single-Threaded: Uses a single thread to handle requests through event looping .

Fast and Scalable: Built on Chrome's V8 engine, it provides high performance and scalability.

Cross-Platform: Works across various operating systems like Windows, Linux, and macOS.

NPM (Node Package Manager): Access to a large library of open-source packages for rapid development.

Built-in Libraries: Provides essential modules like http, fs, and path for server-side development.

Environment Setup

Node.js Environment Setup

Follow these steps to set up Node.js on your system:

1. Download & Install Node.js

- Go to Node.js Official Website
- Download LTS (Long-Term Support) version
- Install it (includes npm automatically)

2. Verify Installation

Check if Node.js and npm are installed:

sh
CopyEdit
node -v   # Check Node.js version
npm -v   # Check npm version

Package Manager (NPM)

NPM is the default package manager for Node.js. It helps in installing, managing, and sharing JavaScript packages (libraries).

Global vs Local Installation (NPM)

- Local Installation (npm install package-name) → Installs the package inside the project folder (node_modules), accessible only within that project.
- Global Installation (npm install -g package-name) → Installs the package system-wide, making it available for all projects and command-line use.

Using package.json

package.json is a configuration file in a Node.js project that stores important project details such as metadata, dependencies, scripts, and configurations. It helps in managing and automating tasks efficiently.

Attributes of package.json

Understanding Attributes of `package.json` (With One-Line Description)

1. `name` – The project name (must be lowercase).
2. `version` – The project version (follows semantic versioning).
3. `description` – A short description of the project.
4. `main` – The entry point file (default: `index.js`).
5. `scripts` – Custom commands to run the project (e.g., `npm start`).
6. `dependencies` – Packages required for production.
7. `devDependencies` – Packages needed only for development.
8. `keywords` – Keywords related to the project for searchability.
9. `author` – The creator of the project.
10. `license` – Specifies the licensing type (e.g., MIT).
11. `engines` – Specifies required Node.js version.

📌 **Purpose:** `package.json` helps manage dependencies, scripts, and project metadata in a structured way! 🚀

Uninstalling Modules

```
npm uninstall  < package-name >
```

Updating Modules in Node.js (NPM)

npm update package-name

Searching for Modules in NPM (One-Line Command)

🔍 Search for a package in the NPM registry:

```sh
npm search package-name
```

Copy Edit

or directly visit **https://www.npmjs.com/** to explore packages. 🚀

REPL Terminal

The repl module in Node.js provides a Read-Eval-Print Loop (REPL) interface. It allows you to execute JavaScript code interactively, similar to what you might do in a browser console or other interactive environments.

Starting the REPL

1. Starting the Node.js REPL

To enter the REPL:

1. Open your terminal.
2. Type:

```bash
node
```

This will start the Node.js REPL.

You'll see a prompt like this:

```bash
>
```

REPL Commands

- .help - Shows all available REPL commands.
- .exit - Exits the REPL session.
- .editor - Opens multi-line editing mode.
- .clear - Clears the current REPL context.'
- .load <filename> - Loads and executes a JavaScript file in the REPL.
- .save <filename> - Saves the current REPL session to a file.

Stopping the REPL

To stop the REPL:

Type `.exit` and press Enter Or press Ctrl + C twice.

Callbacks Concept

A callback function is a function that is passed as an argument to another function. It is executed after the completion of a task within the main function.

We use callbacks in Node.js to handle asynchronous operations without blocking the program. They allow functions to execute tasks and then run code once the task is completed, such as reading files or making HTTP requests.

- **Synchronous Callback**: Executes immediately and blocks further code.
- **Asynchronous Callback**: Executes after the task is completed and doesn't block other operations.

Event Loop

"Node.js can handle multiple tasks and asynchronous operations concurrently because of the Event Loop. Even though Node.js is single-threaded, the Event Loop enables it to manage non-blocking operations efficiently. It continuously checks and executes pending tasks like I/O operations, timers, and callbacks, ensuring that multiple tasks run without blocking the main thread. This makes Node.js highly efficient for handling asynchronous tasks."

Props Validation in Node.js (Theory & Implementation)

📌 What is Props Validation in Node.js?

In **React**, props validation ensures that components receive the correct type of data. Similarly, in **Node.js**, when handling API requests, we must validate the request body, query parameters, or URL parameters before processing them. This prevents errors and ensures data integrity.

Event Emitte

In Node.js, the EventEmitter class from the events module allows you to create and handle custom events.

EventEmitter is a built-in module in Node.js that allows handling and emitting custom events asynchronously. It follows the event-driven architecture, where an event is emitted, and registered listeners respond to it.

How EventEmitter Works

1. Import the `events` module.
2. Create an instance of `EventEmitter`.
3. Register event listeners using `.on()` or `.once()`.
4. Emit events using `.emit()`.

Buffers

In Node.js, Buffer is a built-in object used to handle raw binary data. It is especially useful when dealing with data streams, like reading from or writing to files, handling HTTP requests .

Why Use Buffers in Node.js?

- JavaScript strings are UTF-16 encoded, but when you deal with raw binary data (e.g., images, files, network data), you need to work with Buffer objects because they provide a way to handle data as binary sequences.

Buffers help improve performance when manipulating raw binary data.

1. `Buffer.alloc(size)`
   - **Creates a buffer** and allocates a specified size (in bytes) to it.
2. `Buffer.from(initialization)`
   - **Initializes** a buffer with given data (string, array, etc.).
3. `Buffer.write(data)`
   - **Writes** data to a buffer, replacing its current content.
4. `toString()`
   - **Reads** the data stored in the buffer and converts it to a string.
5. `Buffer.isBuffer(object)`
   - **Checks** if an object is a buffer.
6. `Buffer.length`
   - Returns the **length** (in bytes) of the buffer.
7. `Buffer.copy(buffer, subsection size)`
   - **Copies** data from one buffer to another.
8. `Buffer.slice(start, end=buffer.length)`
   - **Extracts** a portion (subsection) of data from the buffer.
9. `Buffer.concat([buffer, buffer])`
   - **Concatenates** multiple buffers into one.

Streams

In Node.js, a stream is an abstract interface for working with streaming data. It allows you to read or write data piece by piece, instead of loading the entire data into memory at once. This is crucial for handling large amounts of data, as it improves performance, reduces memory consumption, and provides real-time data processing.

File System

> The **File System (fs) module** in Node.js allows you to work with the file system, enabling you to read, write, update, delete, and manage files and directories.

Operation	Asynchronous	Synchronous
Read File	`fs.readFile`	`fs.readFileSync`
Write File	`fs.writeFile`	`fs.writeFileSync`
Append File	`fs.appendFile`	`fs.appendFileSync`
Delete File	`fs.unlink`	`fs.unlinkSync`
Rename File	`fs.rename`	`fs.renameSync`
Copy File	`fs.copyFile`	`fs.copyFileSync`
Create Directory	`fs.mkdir`	`fs.mkdirSync`
Read Directory	`fs.readdir`	`fs.readdirSync`
Remove Directory	`fs.rmdir`	`fs.rmdirSync`
Check File Exists	`fs.access`	-
Get File Stats	`fs.stat`	`fs.statSync`

Child Process Operations

- Child Process Node.js me naye system processes create karne ke liye use hota hai.
- exec() - Chhoti commands ke liye, spawn() - Large output ke liye, fork() - Parallel processing ke liye.
- Agar aapko system commands, dusre scripts ya parallel processing chahiye, to ye module kaafi useful hai.

Global Objects

Global Objects in Node.js

In Node.js, **Global Objects** are built-in objects that are **available in all modules without requiring an import**. These objects can be accessed from anywhere in a Node.js application.

- ✅ `global` - Stores all global variables.
- ✅ `__dirname` - Gets the current directory path.
- ✅ `__filename` - Gets the current file path.
- ✅ `process` - Provides system and environment information.
- ✅ `console` - Used for logging messages.
- ✅ `setTimeout()` - Runs a function after a delay.
- ✅ `setInterval()` - Runs a function repeatedly at an interval.
- ✅ `clearTimeout()` - Cancels a scheduled `setTimeout()`.
- ✅ `clearInterval()` - Cancels a repeating `setInterval()`.
- ✅ `require()` - Used for importing modules.

Utility Modules

📌 Utility Modules in Node.js

Node.js provides several **utility modules** that help developers handle common tasks like debugging, formatting, and working with data structures efficiently. These modules are **built-in**, meaning you don’t need to install them separately.

🔹 List of Important Utility Modules in Node.js

1. **util** – Provides various utility functions
2. **os** – Provides information about the operating system
3. **path** – Handles and manipulates file paths
4. **fs (File System)** – Handles file operations (read/write files)
5. **events** – Implements the EventEmitter class for event handling
6. **crypto** – Provides cryptographic functions (hashing, encryption)
7. **querystring** – Parses and formats URL query strings
8. **url** – Parses and formats URLs
9. **zlib** – Handles file compression and decompression
10. **dns** – Performs domain name resolution