Week 10: Introduction to Node.js and Server-Side JavaScript

Day 1: The Node.js Runtime Environment

Theoretical Overview

Node.js is a cross-platform, open-source JavaScript runtime environment that executes JavaScript code outside of a web browser. Historically, JavaScript was confined to client-side execution within browsers like Chrome or Firefox. Node.js changed this paradigm by leveraging Google's V8 JavaScript engine to run code directly on the server.

Applications and Benefits

- **Scalability**: Node.js is designed to build scalable network applications.
- **JavaScript Everywhere**: It allows developers to use a single language for both frontend (React) and backend (Node) development.
- **Rich Ecosystem**: Through the Node Package Manager (NPM), developers have access to millions of open-source libraries.
- **Performance**: The V8 engine compiles JavaScript into machine code, providing high-speed execution.

Day 2: Architecture and the Event Loop

Traditional Server Models vs. Node.js

Traditional server-side I/O models (like those used in older Java or PHP setups) typically use a multi-threaded approach where each request spawns a new thread. This can lead to high memory consumption and "blocking" where the server waits for one task to finish before starting the next.

Event-Driven, Non-Blocking I/O

Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

- **Single Threaded**: Node.js operates on a single main thread using an Event Loop.
- **Concurrency**: It handles multiple simultaneous connections without the overhead of thread management, making it ideal for data-intensive real-time applications.
- **Event Emitters**: Much of the Node.js core is built around objects that emit events, which are then handled by "listener" functions.

Day 3: Node Package Manager (NPM) and Project Setup

Managing Dependencies

NPM (Node Package Manager) is the world's largest software registry. It is installed automatically with Node.js and allows developers to share and reuse code easily.

- **Initialize Project**: The npm init command creates a package.json file, which tracks all project dependencies and metadata.
- **Installing Packages**: Packages are installed using npm install <package-name>, which stores the code in a node_modules folder.
- **Package.json**: This file serves as the manifest for the project, listing versions of libraries like Express or MongoDB drivers.

Day 4: Creating Your First Node Server

"Hello, Node World!"

Building a basic server in Node.js requires understanding the core modules.

- **The HTTP Module**: Node.js includes a built-in HTTP module that allows it to transfer data over the HyperText Transfer Protocol.

- **Request and Response**: The server listens for a "Request" object from the client and sends back a "Response" object containing data or HTML.
- **How It Works**: You create a server instance, define a port (like 3000), and write a callback function to handle incoming traffic.

Day 5: Web Service and Environment Setup

Practical Implementation

This day focuses on setting up the environment required for the **Gadget API** backend project.

- **Installation**: Installing Node.js and NPM on the local machine.
- **Execution**: Creating a basic web service that responds with JSON data.
- **The Process Object**: Learning to use process.env to manage environment variables, such as database connection strings, which is a security best practice for the **Gadget API**.