Node.js Architecture Overview

Lecture 04: Node.js Architecture and Event Handling

Course: Advanced Web Technologies (CSC337)

Duration: 2 Hours

Lecture Type: Theory

Reading Material: *Beginning Node.js, Express & MongoDB Development* by Greg Lim (Chapters 3 & 4)

Learning Outcomes

By the end of this lecture, students should be able to:

1. Understand the architecture of Node.js.
2. Explain the **Event Loop**, **Callbacks**, and **Event Emitters**.
3. Use the **NPM (Node Package Manager)** for package management.
4. Execute commands using **NPM CLI**.

---

1. Introduction to Node.js Architecture

Node.js follows a **non-blocking, event-driven architecture**, which makes it efficient and scalable. Unlike traditional web servers that handle each request with a separate thread, Node.js uses a **single-threaded event loop** to handle multiple requests asynchronously.

1.1 Key Features of Node.js Architecture

- **Single-Threaded**: Uses a single main thread to handle multiple client requests.
- **Non-Blocking I/O**: Uses asynchronous functions, preventing the system from waiting for I/O operations.
- **Event-Driven**: Uses an event loop to handle operations efficiently.
- **V8 Engine**: Built on Google's V8 JavaScript engine, providing high-speed execution.
- **NPM (Node Package Manager)**: Helps manage libraries and dependencies.

1.2 How Node.js Handles Requests

1. A request comes to the server.
2. If it's a **non-blocking** request (e.g., fetching data from a database), it is assigned a callback and moved to a queue.
3. The **event loop** continuously checks if the request is completed.
4. Once completed, the callback function executes and the response is sent to the client.

Real-Life Example:
Think of Node.js as a **fast-food restaurant**. When you order food, the chef does not wait for each dish to be prepared before taking the next order. Instead, multiple orders are taken, processed asynchronously, and served when ready.

---

2. Understanding the Event Loop

The **event loop** is the core of Node.js that enables asynchronous programming. It continuously checks for pending tasks and executes them when ready.

2.1 Event Loop Phases

1. **Timers Phase**: Executes callbacks scheduled by `setTimeout()` and `setInterval()`.
2. **I/O Callbacks Phase**: Executes callbacks from I/O operations (e.g., network requests, file system operations).
3. **Idle/Prepare Phase**: Internal phase for Node.js.
4. **Poll Phase**: Fetches new I/O events and executes them.
5. **Check Phase**: Executes `setImmediate()` callbacks.
6. **Close Callbacks Phase**: Executes cleanup operations (e.g., `socket.on('close')`).

Example:

```
javascript


console.log("Start");
setTimeout(() => {
    console.log("Timeout");
}, 0);
console.log("End");
```

Output:

```
sql


Start
End
Timeout
```

Even though `setTimeout` is set to 0 milliseconds, it runs **after** the synchronous code (`Start` and `End`) due to the event loop.

---

3. Callbacks in Node.js

A **callback** is a function that is passed as an argument to another function and executed later.

Example:

```
javascript

function greet(name, callback) {
    console.log("Hello " + name);
    callback();
}
greet("Alice", function () {
    console.log("This is a callback function.");
});
```

Output:

```
vbnet

Hello Alice
This is a callback function.
```

- The `greet` function executes the `callback()` function after logging "Hello Alice".

---

4. Event Emitters in Node.js

Node.js uses the **EventEmitter** class to handle events.

4.1 Creating an Event Emitter

```
javascript

const EventEmitter = require('events');
const eventEmitter = new EventEmitter();

eventEmitter.on('greet', () => {
    console.log('Hello, this is an event!');
});

eventEmitter.emit('greet');
```

Explanation:

- `on('greet')`: Listens for an event named `greet`.
- `emit('greet')`: Triggers the event.

Real-Life Example of Event Emitters

Consider a **fire alarm system**:

- The alarm **(event listener)** is always active.
- If smoke is detected, it **emits** an event.
- The alarm rings **(event handler)**.

---

5. Node Package Manager (NPM)

NPM is used to install and manage Node.js packages.

5.1 Installing a Package

```
bash

npm install lodash
```

- Installs the `lodash` package.

5.2 Installing Packages Globally

```
bash

npm install -g nodemon
```

- The `-g` flag installs the package globally.

5.3 Uninstalling a Package

```
bash

npm uninstall lodash
```

5.4 Checking Installed Packages

```
bash

npm list
```

Activity: Hands-on Practice (10 Minutes)

Task: Create a Simple Event Emitter

1. Create a new Node.js file `app.js`.
2. Use the `events` module to create a custom event.
3. Emit and listen to the event.

Quick Quiz (10 Minutes)

1. What is the main advantage of Node.js being **single-threaded**?
2. What are the six phases of the **Event Loop**?
3. How does **asynchronous execution** improve performance?
4. What is the difference between `setTimeout()` and `setImmediate()`?
5. What does `npm install -g` do?

Summary

- Node.js is a **single-threaded, non-blocking** JavaScript runtime.

- The **Event Loop** handles asynchronous tasks efficiently.
- **Callbacks** allow execution of code after another function completes.
- **Event Emitters** are used for handling events in Node.js.
- **NPM** helps manage packages in Node.js applications.

Next Lecture Preview

In the next lecture, we will:

- Implement a basic **HTTP server** using Node.