Node.js and Express.js Guide

1. Node.js Basics

What is Node.js?
Node.js is a JavaScript runtime built on Chrome's V8 engine that allows JavaScript to be used for server-side programming.
- Asynchronous, Non-blocking I/O model.
- Built-in package manager (NPM).
- Uses event-driven architecture.
- Ideal for scalable applications.

Why V8 Engine?
- V8 compiles JavaScript to machine code, making it fast.
- Used in Google Chrome and Node.js.

How Node.js Works?
- Uses event loop to handle multiple requests without creating multiple threads.
- Uses Libuv for asynchronous operations.

2. Node.js Module System

Types of Modules
1. Core Modules - Built-in modules (fs, http, path, os, etc.).
2. Local Modules - Custom modules created by developers.
3. Third-Party Modules - Installed via NPM (e.g., Express, Lodash).

Module.exports & Exports
- module.exports = {} is used to export functions, objects, or variables.
- exports is a reference to module.exports.

Require in Node.js
- require() is used to import modules.

```javascript
const fs = require('fs');
```

3. Express.js Basics

What is Express.js?
Express.js is a minimalist web framework for Node.js that simplifies handling HTTP requests, middleware, and routing.

.use() Method in Express
- Used to add middleware functions.

```javascript
app.use(express.json()); // Middleware to parse JSON
```

View Engine in Express.js (HBS)
- Used to render dynamic HTML templates.
- hbs (Handlebars) is commonly used with Express.

```javascript
app.set('view engine', 'hbs');
```

4. HTTP and Requests

Request-Response Cycle
1. Client sends a request to the server.
2. Server processes the request.
3. Server sends a response back to the client.

HTTP Methods
- GET - Retrieve data.
- POST - Send data.
- PUT - Update existing data.
- DELETE - Remove data.

Request Headers & Response Headers
- req.headers - Access request headers.
- res.setHeader() - Set response headers.

Status Codes
- 200 OK - Success.
- 404 Not Found - Resource not found.
- 500 Internal Server Error - Server failure.

5. Security in Node.js

CORS (Cross-Origin Resource Sharing)
- Controls which domains can access the server.
- Preflight request checks permissions before sending the actual request.

SQL Injection & Prevention
- Vulnerability: Attackers inject malicious SQL queries.
- Fix: Use prepared statements.

```javascript
const query = "SELECT * FROM users WHERE username = ? AND password = ?";
db.query(query, [username, password]);
```

XSS (Cross-Site Scripting)
- Injecting malicious JavaScript into a web page.
- Use input validation and HTML escaping.

CSRF (Cross-Site Request Forgery)
- Attackers perform actions on behalf of authenticated users.
- Use CSRF tokens to prevent this.

6. Advanced Topics

Cluster vs. Worker Threads

- Cluster - Creates multiple child processes to utilize multi-core CPUs.
- Worker Threads - Run JavaScript code in parallel threads for CPU-intensive tasks.

Child Process Methods
- spawn() - Executes a command, returns a stream.
- fork() - Creates a new Node.js process.
- execFile() - Runs an external executable.

Cron Jobs in Node.js
- Used for scheduling tasks at intervals.
- Example using node-cron:

```javascript
const cron = require('node-cron');
cron.schedule('* * * * *', () => {
    console.log('Task runs every minute');
});
```

7. Useful NPM Commands

```bash
npm init          # Initialize package.json
npm install       # Install dependencies
npm install -g    # Install globally
npm install nodemon --save-dev   # Install nodemon as a dev dependency
```

Conclusion

This document covers Node.js fundamentals, Express.js basics, security best practices, and advanced topics like clusters and worker threads. With this guide, you should have a solid foundation in Node.js development.