Introduction to Node.js: Powering Modern JavaScript on the Server

A comprehensive guide to getting started with Node.js - the platform that revolutionized server-side development by bringing JavaScript beyond the browser.

What is Node.js and Why Use It?

Definition & Origin

Created by Ryan Dahl in 2009, Node.js is a cross-platform JavaScript runtime built on Chrome's V8 engine, allowing developers to execute JavaScript code outside web browsers.

Key Benefits

- Unifies front-end and back-end development with a single language
- Lightweight and efficient for data-intensive applications
- Massive ecosystem with reusable packages via npm

Major companies like Netflix, LinkedIn, and PayPal rely on Node.js to power their high-performance services, handling millions of concurrent connections.

Node.js Runtime & Event-Driven Model

Single-Threaded, But Scalable

Node.js handles thousands of concurrent connections with a single-threaded event loop, delegating I/O operations to the system kernel through callbacks.

Non-Blocking Architecture

Unlike traditional servers that create new threads for each request (potentially causing memory issues), Node.js processes events asynchronously without waiting for operations to complete.

This event-driven approach is perfect for real-time applications like chat services, streaming platforms, and APIs with high throughput requirements.

Modules in Node.js: CommonJS vs ES Modules

CommonJS (Traditional)

Uses `require()` and `module.exports`

Synchronous loading, dynamic imports

ES Modules (Modern)

Uses `import` and `export`

Static analysis, top-level `await`

Node.js supports both module systems, though projects typically use one or the other by specifying `"type": "module"` in package.json for ESM or omitting it (defaulting to CommonJS).

Core Built-in Modules Overview

fs (File System)

Read, write, update, and delete files with both synchronous and asynchronous methods.

`fs.readFile()`, `fs.writeFile()`

path

Platform-independent path utilities for joining, resolving, and normalizing file paths.

`path.join()`, `path.resolve()`

os

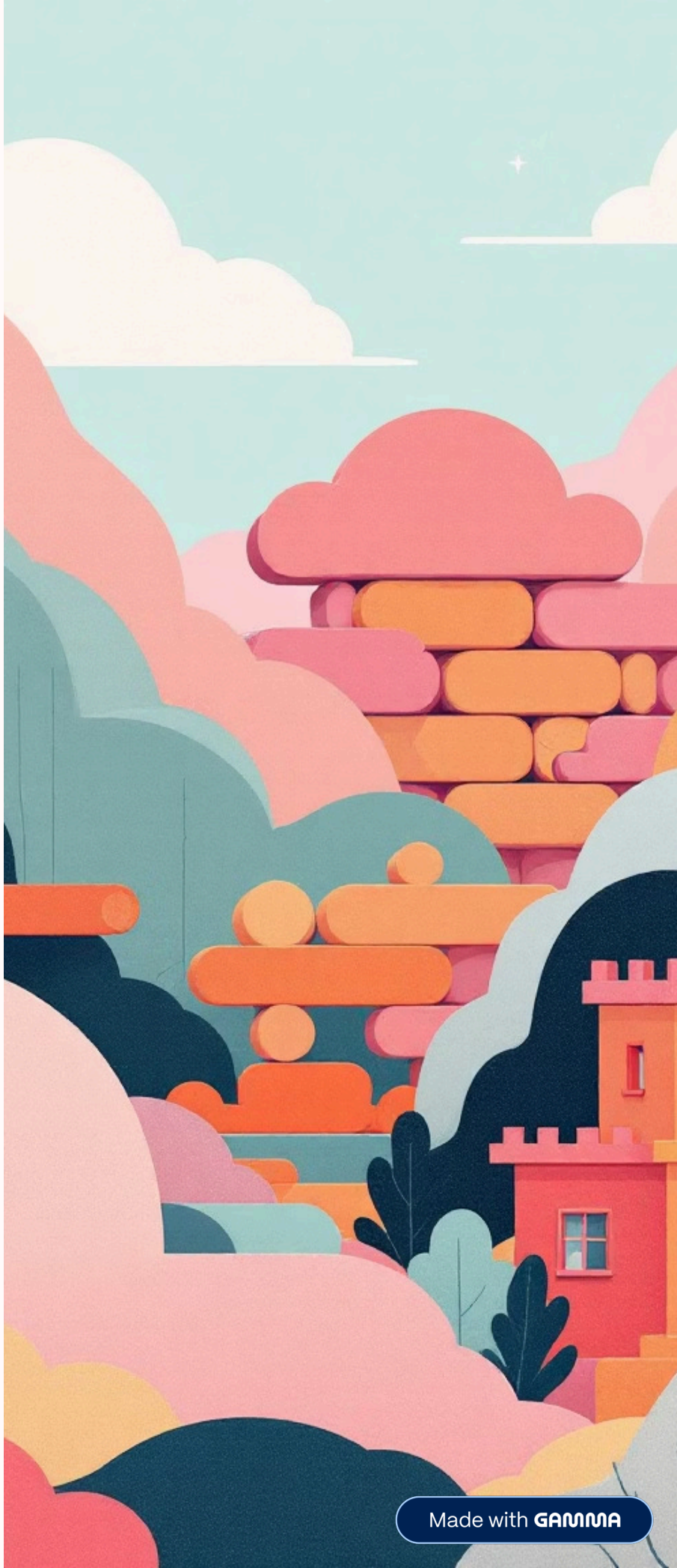
Access operating system information like CPU, memory, and platform details.

`os.platform()`, `os.freemem()`

http

Create web servers and make HTTP requests without external dependencies.

`http.createServer()`, http.request()



npm & package.json Basics

npm: The Node Package Manager

Provides access to over 1.3 million packages and serves over 75 billion downloads per month.

Essential Commands:

- `npm init` - Create a new package.json
- `npm install express` - Add a dependency
- `npm run start` - Execute a script

The package.json file is the heart of any Node.js project, defining metadata, dependencies, scripts, and configuration. Always commit this file to version control!

Mini Task: Create a System Info Logger Script

Requirements:

1. Collect OS information (platform, architecture)
2. Gather CPU details (model, cores, speed)
3. Calculate available memory and total memory
4. Format data in a readable structure
5. Write information to a text file

Modules Needed:

`const os = require('os');` - For system information

`const fs = require('fs');` - For file operations

`const path = require('path');` - For file path handling

This practical task demonstrates how Node.js core modules work together to create useful utilities with minimal code.

Sample Code Snippet for System Info Logger

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

// Collect system information
const systemInfo = {
  platform: os.platform(),
  architecture: os.arch(),
  hostname: os.hostname(),
  cpuModel: os.cpus()[0].model,
  cpuCores: os.cpus().length,
  totalMemory: `${(os.totalmem() / 1024 / 1024 / 1024).toFixed(2)} GB`,
  freeMemory: `${(os.freemem() / 1024 / 1024 / 1024).toFixed(2)} GB`,
  uptime: `${(os.uptime() / 3600).toFixed(2)} hours`
};

// Format the data
const output = Object.entries(systemInfo)
  .map(([key, value]) => `${key}: ${value}`)
  .join('\n');

// Write to file
fs.writeFile('system-info.txt', output, err => {
  if (err) console.error('Error writing file:', err);
  else console.log('System info saved to system-info.txt');
});
```

Try running this script and examine the output file! Experiment by adding more system metrics or formatting the output differently.