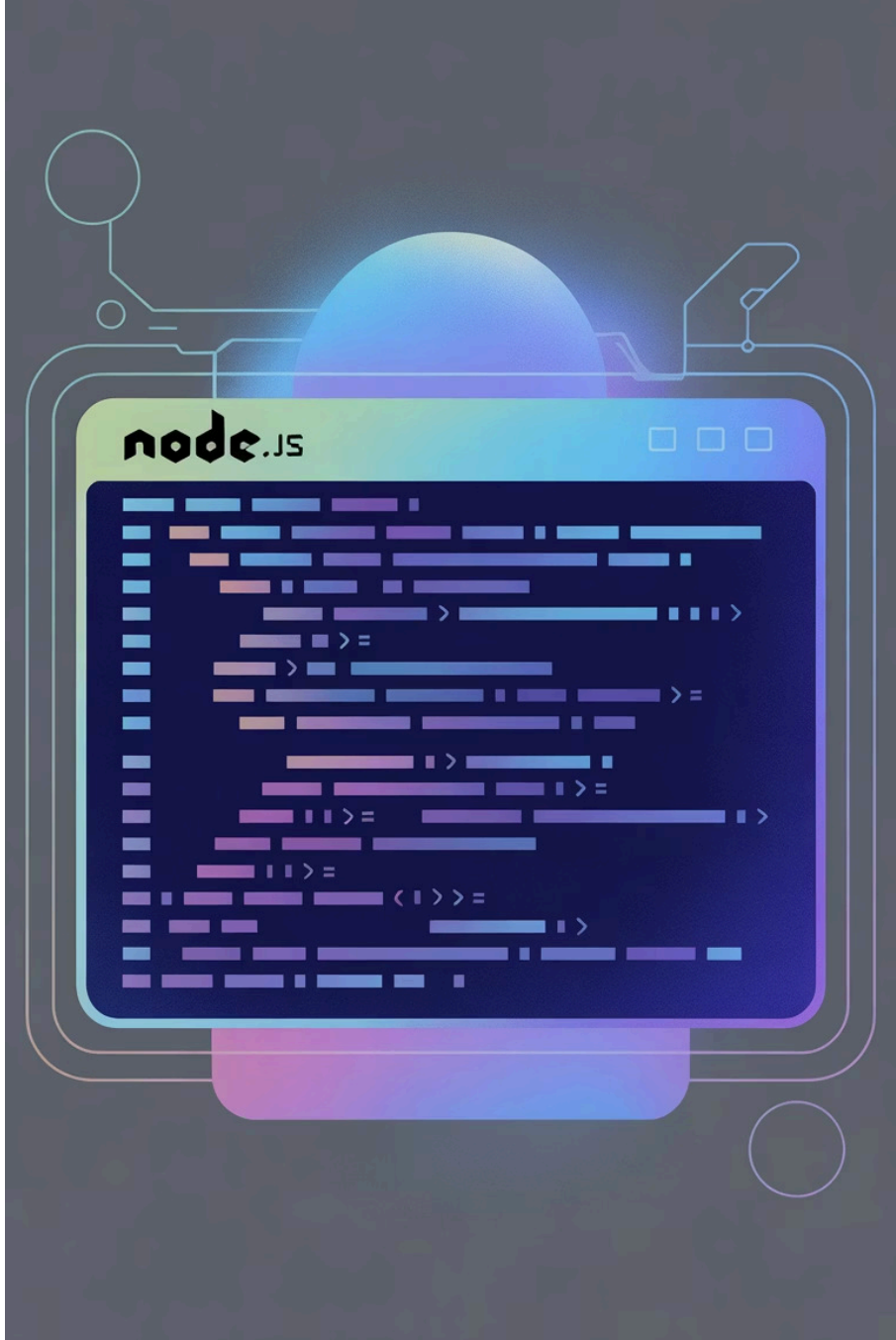
Node.js Fundamentals

Master the essentials of Node.js - the runtime environment that revolutionised server-side JavaScript development and powers millions of applications worldwide.



What is Node.js?

Node.js is a **runtime environment** that allows you to execute JavaScript code outside the browser. Built on Google Chrome's powerful V8 JavaScript engine, it brings JavaScript to the server side, opening up endless possibilities for full-stack development.

What makes Node.js truly special is its **event-driven, non-blocking I/O architecture**. This means your applications can handle thousands of concurrent connections efficiently, making it perfect for modern web applications that demand high performance and scalability.

Runtime Environment

Execute JavaScript anywhere, not just in browsers

V8 Engine

Lightning-fast performance with Google's engine

Server-Side Power

Build robust backend applications with JavaScript

Key Features That Make Node.js Powerful

Node.js stands out from traditional server technologies through its unique architectural choices. Understanding these core features is essential for leveraging Node.js effectively in your projects.

Asynchronous & Non-blocking

Handles multiple requests simultaneously without waiting for each operation to complete. Your server stays responsive even under heavy load, processing thousands of requests efficiently.

Single-threaded Event Loop

Uses one main thread with an event loop to manage operations. This eliminates the complexity and overhead of traditional multi-threaded servers whilst maintaining excellent performance.

Cross-platform

Runs seamlessly on Windows, macOS, and Linux. Write once, deploy anywhere - perfect for teams working across different operating systems.

NPM Ecosystem

Access to over one million packages through Node Package Manager. The world's largest software registry provides solutions for virtually any programming challenge.

These features make Node.js particularly excellent for real-time applications like chat systems, live streaming platforms, and collaborative tools where instant communication is crucial.

Installing & Running Node.js

Getting Started

Before diving into Node.js development, you'll need to verify your installation and understand the basic commands. These simple steps will get you up and running quickly.

First, check if Node.js is properly installed on your system by running version commands in your terminal. This will confirm both Node.js and NPM are ready to use.

01

Check Your Installation

```
node -v
npm -v
```

Verify both Node.js and NPM versions are displayed correctly

02

Create Your First File

Create a JavaScript file (e.g., app.js) with some basic Node.js code to test your setup

03

Run Your Application

```
node app.js
```

Execute your JavaScript file directly from the command line

Core Modules: Node.js Built-in Powerhouses

Node.js comes with a rich set of built-in modules that provide essential functionality without requiring external dependencies. These core modules are the building blocks of most Node.js applications.

File System (fs)

Read, write, and manipulate files and directories. Handle file operations synchronously or asynchronously to suit your application's needs.

- Create and delete files
- Read file contents
- Watch for file changes

HTTP Module

Create web servers and handle HTTP requests and responses. The foundation for building web applications and APIs in Node.js.

- Create HTTP servers
- Handle requests and responses
- Manage headers and status codes

Path Module

Work with file and directory paths across different operating systems. Ensures your path operations work consistently everywhere.

- Join path segments
- Extract file extensions
- Resolve absolute paths

```
const fs = require("fs");
fs.writeFileSync("test.txt", "Hello Node.js!");
```

This simple example demonstrates how to use the fs module to create a file with content.

Creating Your First Node.js Server

Building a web server is often the first practical step in Node.js development. With just a few lines of code, you can create a fully functional HTTP server that responds to requests.

```
const http = require("http");

const server = http.createServer((req, res) => {
   res.writeHead(200, { "Content-Type": "text/plain" });
   res.end("Hello, Node.js!");
});

server.listen(3000, () =>
   console.log("Server running on port 3000")
);
```

1

Import HTTP Module

Load the built-in HTTP module to access server creation functionality

2

Create Server

Define how your server responds to incoming requests with custom logic

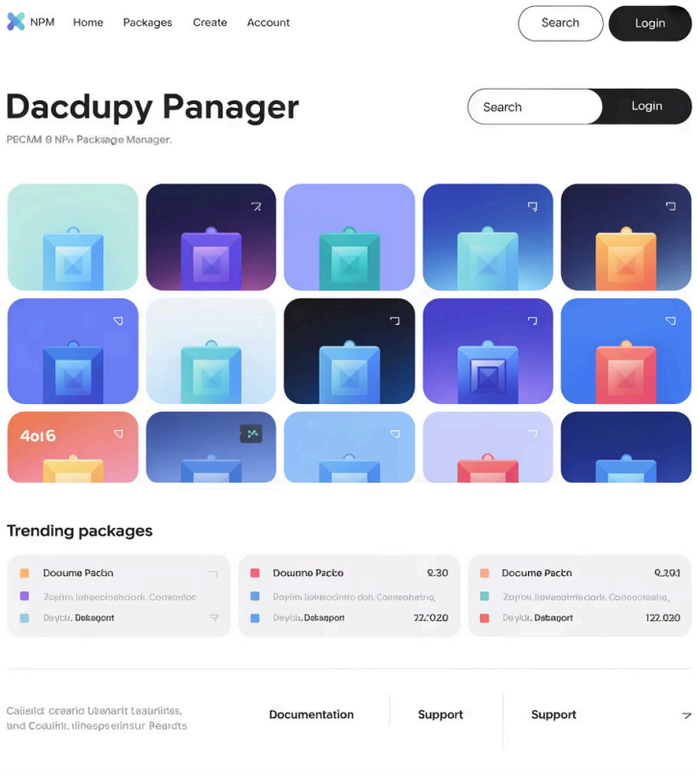
3

Start Listening

Bind to port 3000 and begin accepting connections from clients

Test Your Server: Run this code and visit `http://localhost:3000` in your browser to see your server in action!

NPM: Your Gateway to Node.js Packages



Node Package Manager (NPM) is the world's largest software registry, containing over one million packages. It's your gateway to leveraging the incredible Node.js ecosystem and avoiding the need to reinvent the wheel.

NPM handles dependency management, version control, and package distribution, making it effortless to incorporate powerful libraries into your projects.

01

Initialize Your Project

```
npm init -y
```

Creates a `package.json` file with default settings to manage your project dependencies

02

Install Production Dependencies

```
npm install express
```

Adds packages your application needs to run in production

03

Install Development Dependencies

```
npm install nodemon --save-dev
```

Adds packages only needed during development, like testing tools or build utilities

Module System: Organising Your Code

Node.js uses the CommonJS module system to help you organise code into reusable components. This modular approach promotes clean architecture and code reusability across your applications.

Exporting Modules

Create reusable functions and objects by exporting them from dedicated files. This keeps your code organised and promotes the DRY (Don't Repeat Yourself) principle.

```
// add.js
module.exports = function add(a, b) {
   return a + b;
};
```

You can also export multiple items using object notation for more complex modules.

Importing Modules

Use the `require()` function to import modules into your application. Node.js will automatically look for the file and load its exports.

```
// main.js
const add = require("./add");
console.log(add(2, 3)); // Output: 5
```

The relative path `./` indicates the module is in the same directory.

Single Function Export

Export one main function or class per module

Multiple Exports

Export objects containing multiple functions and properties

Selective Importing

Import only the specific functions you need

Express.js: The Node.js Web Framework

Express.js is the most popular web framework for Node.js, providing a robust set of features for building web applications and APIs. It simplifies common web development tasks while maintaining the flexibility of Node.js.

```
const express = require("express");
const app = express();

app.get("/", (req, res) =>
   res.send("Hello Express!")
);

app.listen(3000, () =>
   console.log("Server running on port 3000")
);
```

1. **Install Express**

   Add Express to your project with NPM

2. **Create App Instance**

   Initialize Express application

3. **Define Routes**

   Set up URL endpoints and handlers

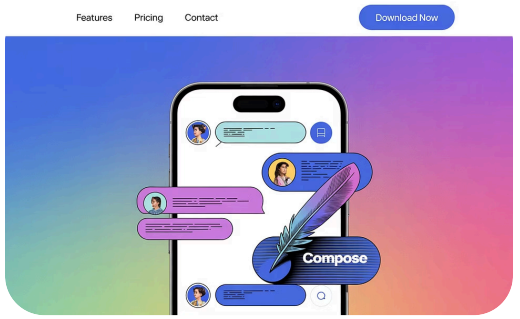
4. **Start Server**

   Begin listening for requests

Express provides powerful features like routing, middleware support, template engines, and much more, making it the go-to choice for Node.js web development.

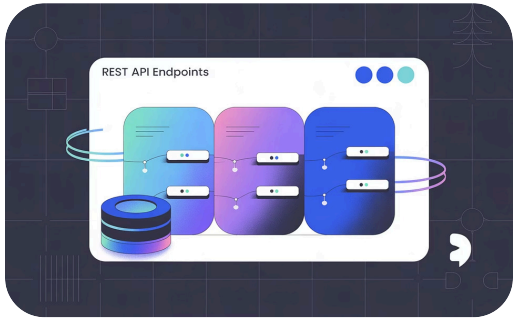
Real-World Node.js Applications

Node.js powers some of the world's most popular applications and services. Its versatility and performance make it ideal for a wide range of use cases, from real-time communication to data-intensive applications.



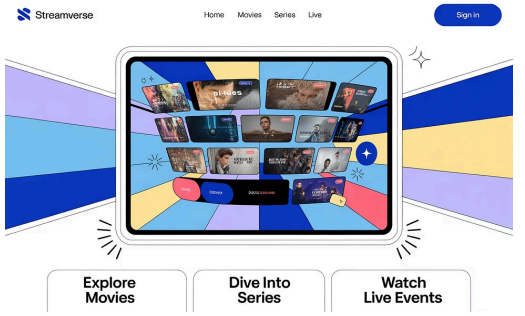
Real-time Chat Applications

Perfect for instant messaging, live chat support, and collaborative communication platforms requiring immediate message delivery.



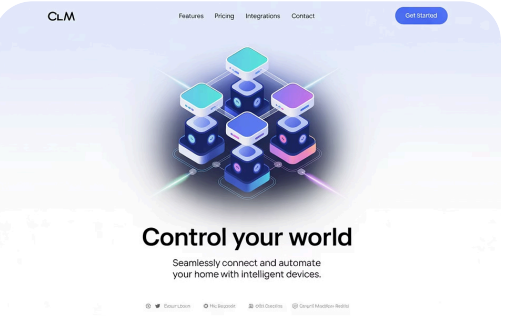
REST APIs & Backend Services

Build scalable APIs that power mobile apps, web applications, and microservices architectures with high throughput.



Streaming Applications

Handle media streaming, live broadcasts, and real-time data feeds with Node.js's excellent I/O performance.



IoT Applications

Connect and manage Internet of Things devices, process sensor data, and build smart home automation systems.

> *"Node.js enables developers to build fast, scalable network applications capable of handling a large number of simultaneous connections with high throughput."*

Companies like Netflix, LinkedIn, Uber, and WhatsApp rely on Node.js for their critical systems, demonstrating its enterprise-grade reliability and performance capabilities.