Nodejs Cheatsheet https://winstonbrown.me

Shortcut / Concept	Context	Description	Example
Destructuring Assignment	JavaScript / Node.js	Extract values from arrays or properties from objects quickly.	`const {name, age} = user;`
Default Parameters	JavaScript / Node.js	Define default values for function parameters to prevent undefined arguments.	`function greet(name = "Guest") { console.log(name); }`
Async/Await	JavaScript / Node.js	Manage asynchronous code in a cleaner, more readable way than Promises.	`const data = await fetchData();`
Module Imports	JavaScript / Node.js	Use ES6 `import` and `export` syntax (Node >= 12), also applicable to most JavaScript environments including browsers for a cleaner module structure. for a cleaner module structure.	`import fs from 'fs';`
Event Emitters	Node.js	Create custom events and listen to them using the built-in EventEmitter module.	`const EventEmitter = require('events'); class MyEmitter extends EventEmitter {}`
Path Manipulation	Node.js	Manage paths with the `path` module for cross-platform compatibility.	`const path = require('path'); path.join(__dirname, 'file.txt');`
File System Promises	Node.js	Use `fs.promises` to handle file operations with async/await (introduced in Node.js version 10), making it easier to manage asynchronous file operations.	`await fs.promises.readFile('file.txt', 'utf8');`
Environment Variables	Node.js	Securely handle sensitive data by storing it in environment variables using `process.env`.	`const apiKey = process.env.API_KEY;`
Cluster Module	Node.js	Leverage multiple CPU cores to handle requests in parallel for better performance.	`const cluster = require('cluster'); cluster.fork();`

Shortcut / Concept	Context	Description	Example
Worker Threads	Node.js	Run CPU-bound tasks in separate threads for better performance (introduced in Node.js version 12), without blocking the event loop.	`const { Worker } = require('worker_threads');`
Streams	Node.js	Handle large data processing in chunks to avoid memory overloads, especially for file and network I/O. Streams are particularly useful for handling large files or data chunks efficiently without loading the entire content into memory.	`fs.createReadStream('file.txt').pipe(process.stdout);`
Buffer	Node.js	Work with binary data directly, which is useful for file processing, cryptography, etc.	`const buffer = Buffer.from('hello');`
HTTP/HTTPS Module	Node.js	Create servers and handle requests without third-party dependencies.	`http.createServer((req, res) => { res.end('Hello'); }).listen(3000);`
Middleware in Express	Node.js (Express)	Create reusable middleware functions to streamline request processing in Express apps.	`app.use((req, res, next) => { console.log('Time:', Date.now()); next(); });`
Global Exception Handling	Node.js	Handle uncaught exceptions and rejections for application-level error management. Using `process.on` for exception handling is specific to Node.js and helps in managing unexpected errors gracefully.	`process.on('uncaughtException', (err) => console.error('Error:', err));`
Caching with Redis	Node.js	Use Redis as a caching layer to improve app performance by storing frequent data.	`const redis = require('redis'); const client = redis.createClient();`
JWT Authentication	JavaScript / Node.js	Secure your API with JSON Web Tokens for stateless, scalable authentication.	`const jwt = require('jsonwebtoken'); jwt.sign(payload, secret);`

Shortcut / Concept	Context	Description	Example
Rate Limiting	Node.js (Express)	Prevent abuse and denial-of-service attacks by limiting the number of requests from a user/IP.	`app.use(rateLimit({ windowMs: 15 * 60 * 1000, max: 100 }));`
Data Validation	JavaScript / Node.js	Validate data inputs using libraries like Joi or express-validator to ensure API reliability.	`const { body, validationResult } = require('express-validator');`
Graceful Shutdown	Node.js	Handle server shutdown gracefully by closing open connections. Graceful shutdown ensures that all pending requests are completed before terminating the server, preventing data loss or corruption.	`process.on('SIGTERM', () => { server.close(() => console.log('Closed.')); });`
Babel for ES6+ Features	JavaScript / Node.js	Use Babel to transpile modern JavaScript to compatible versions, especially in legacy environments.	`npx babel src --out-dir dist`
Testing with Jest	JavaScript / Node.js	Write unit and integration tests with Jest to improve code reliability.	`test('should add numbers', () => { expect(add(1, 2)).toBe(3); });`