NODE.JS

Manuale Completo 2025

Storia • Architettura Event Loop • Sintassi ES2024 • npm & ecosistema • API REST • WebSocket • Database • Cybersecurity • Integrazione Python • AI & Server MCP • Deployment • Performance • Casi d'uso reali

Versione 2025 • Node.js LTS 22.x • Tutte le fonti citate

SOMMARIO

SOMMARIO 2

1. STORIA E ORIGINI DI NODE.JS 10

1.1 Ryan Dahl e la nascita nel 2009 . . . 10

1.2 Timeline storica completa . . . 11

1.3 Il fork io.js e la governance moderna . . . 12

1.4 Deno e Bun: la concorrenza moderna . . . 12

2. INSTALLAZIONE E SETUP DELL'AMBIENTE 13

2.1 Metodi di installazione . . . 13

Metodo 1 — Installer ufficiale (principianti) . . . 13

Metodo 2 — nvm: Node Version Manager (consigliato) . . . 13

Metodo 3 — fnm: Fast Node Manager (Rust) . . . 13

2.2 Struttura di un progetto Node.js . . . 14

2.3 package.json: anatomia completa . . . 14

2.4 Editor e strumenti di sviluppo . . . 16

3. ARCHITETTURA: EVENT LOOP, V8 E LIBUV 17

3.1 I componenti fondamentali . . . 17

3.2 L'Event Loop: funzionamento dettagliato . . . 17

3.3 I/O non bloccante: il vantaggio competitivo . . . 18

3.4 Il thread pool di libuv . . . 18

3.5 Worker Threads: parallelismo CPU in Node.js . . . 19

4. SINTASSI JAVASCRIPT MODERNA IN NODE.JS 21

4.1 Tipi di dati, variabili e scope . . . 21

4.2 Funzioni: dichiarazioni, arrow, closure . . . 22

4.3 Async/Await, Promise e gestione errori . . . 23

4.4 Classi, Moduli ES e nuove sintassi ES2022-2024 . . . 25

4.5 Iteratori, Generatori e Async Iterators . . . 27

5. NPM E L'ECOSISTEMA DI PACCHETTI 29

5.1 Comandi npm fondamentali . . . 29

5.2 Semantic Versioning (SemVer) . . . 30

5.3 package-lock.json: perché è fondamentale . . . 30

5.4 Alternative a npm: yarn e pnpm . . . 30

5.5 Pacchetti essenziali per ogni categoria . . . 31

6. CORE MODULES: I MODULI BUILT-IN 32

6.1 fs: filesystem sincrono e asincrono . . . 32

6.2 path: gestione cross-platform dei percorsi . . . 32

6.3 events: EventEmitter pattern . . . 33

6.4 stream: elaborazione dati in streaming . . . 34

6.5 Altri core modules importanti . . . 36

7. EXPRESS.JS E API REST 37

7.1 Server Express completo con best practice . . . 37

7.2 CRUD API RESTful completa . . . 39

7.3 Middleware: validazione, autenticazione, errori . . . 41

8. DATABASE: MONGODB, POSTGRESQL E ORM 43

8.1 MongoDB con Mongoose . . . 43

8.2 PostgreSQL con Prisma ORM . . . 45

9. CYBERSECURITY IN NODE.JS 48

9.1 OWASP Top 10 in Node.js: mappatura completa . . . 48

9.2 NoSQL Injection (MongoDB) . . . 49

9.3 Command Injection e eval() . . . 49

9.4 ReDoS: Denial of Service tramite Regex . . . 50

9.5 Supply Chain Attacks e sicurezza npm . . . 51

9.6 Protezione completa: header, HTTPS, secrets . . . 52

9.7 Checklist di sicurezza Node.js per produzione . . . 54

10. WEBSOCKET E APPLICAZIONI REAL-TIME 55

10.1 WebSocket nativo (Node.js 22) . . . 55

10.2 Socket.IO: real-time con fallback . . . 57

10.3 Server-Sent Events (SSE) per push unidirezionale . . . 59

11. INTEGRAZIONE NODE.JS CON PYTHON 60

11.1 Pattern 1: child_process — esecuzione script Python . . . 60

11.2 Pattern 2: HTTP microservizi (Flask/FastAPI) . . . 62

11.3 Pattern 3: ZeroMQ per comunicazione ad alte performance . . . 64

11.4 Confronto pattern di integrazione . . . 65

12. AI E LLM IN NODE.JS 66

12.1 Anthropic SDK: integrazione con Claude . . . 66

12.2 OpenAI SDK in Node.js . . . 68

12.3 Vercel AI SDK: astrazione multi-provider . . . 70

13. MODEL CONTEXT PROTOCOL (MCP) 73

13.1 Cos'è MCP e perché è importante . . . 73

13.2 Costruire un server MCP in Node.js . . . 73

13.3 Configurazione in Claude Desktop . . . 76

13.4 Server MCP HTTP per deployment remoto . . . 77

14. TESTING IN NODE.JS 78

14.1 Jest: setup e test unitari . . . 78

14.2 Integration Test con Supertest . . . 80

15. PERFORMANCE, MONITORING E DEPLOYMENT 83

15.1 Profiling e ottimizzazione delle performance . . . 83

15.2 Cluster mode: tutti i core CPU . . . 83

15.3 Docker: containerizzazione . . . 84

15.4 PM2: process manager in produzione . . . 86

16. CASI D'USO REALI E ARCHITETTURE 88

16.1 Netflix: streaming a 250 milioni di utenti . . . 88

16.2 PayPal: 2x performance con la metà delle righe di codice . . . 88

16.3 Uber: real-time matching su 100M+ utenti . . . 88

16.4 LinkedIn: 27x meno server con Node.js . . . 89

16.5 Architettura microservizi con Node.js: pattern . . . 90

17. EVOLUZIONI FUTURE DI NODE.JS E L'ECOSISTEMA 91

17.1 TypeScript: il futuro dello sviluppo Node.js . . . 91

17.2 Edge Computing e Workers . . . 92

17.3 Deno 2: il ritorno di Ryan Dahl . . . 93

17.4 Node.js e AI: il futuro dell'integrazione . . . 93

17.5 Roadmap Node.js 2025-2027 . . . 94

18. APPENDICE: CHEATSHEET E RISORSE 95

18.1 Cheatsheet: comandi npm e Node.js . . . 95

18.2 Cheatsheet: API Node.js pià usate . . . 95

18.3 Glossario dei termini chiave . . . 96

18.4 Risorse di riferimento . . . 98

19. STREAMS AVANZATI E BUFFER **99**

19.1 Quattro tipi di stream 99

19.2 Readable stream personalizzato 99

19.3 Transform stream: CSV parser personalizzato 100

19.4 Buffer: dati binari in Node.js 102

20. PATTERN DI DESIGN IN NODE.JS **104**

20.1 Singleton e Dependency Injection 104

20.2 Repository Pattern con Node.js 105

20.3 Observer / EventEmitter avanzato 107

20.4 Circuit Breaker pattern 107

21. GRAPHQL IN NODE.JS **109**

21.1 Confronto REST vs GraphQL 109

21.2 Apollo Server: schema e resolver completi 109

22. GRPC E MICROSERVIZI AVANZATI **114**

22.1 gRPC vs REST vs GraphQL 114

22.2 Definizione Protobuf e server gRPC 114

23. LOGGING, MONITORAGGIO E OBSERVABILITY **118**

23.1 Logging strutturato con Pino 118

23.2 Metrics con Prometheus e Grafana 120

23.3 Distributed Tracing con OpenTelemetry 122

24. TESTING AVANZATO E TDD **123**

24.1 TDD: Test-Driven Development in Node.js 123

24.2 Test E2E con Playwright 125

24.3 Test di carico con k6 127

25. TYPESCRIPT AVANZATO IN NODE.JS 130

25.1 Tipi avanzati e utility types . . . 130

25.2 Generic avanzati e infer . . . 132

25.3 Decoratori e metadata (TypeScript 5) . . . 134

26. TOOLCHAIN MODERNA, MONOREPO E CI/CD 135

26.1 Monorepo con pnpm workspaces e Turborepo . . . 135

26.2 ESLint 9 e configurazione moderna . . . 136

26.3 GitHub Actions CI/CD per Node.js . . . 137

27. NODE.JS PER IOT E AUTOMAZIONE INDUSTRIALE 140

27.1 MQTT: il protocollo IoT per eccellenza . . . 140

27.2 Modbus TCP con Node.js . . . 142

27.3 OPC-UA con Node.js . . . 144

28. BEST PRACTICE E ANTI-PATTERN 146

28.1 I 10 anti-pattern più pericolosi . . . 146

28.2 Checklist produzione Node.js . . . 147

28.3 Node.js in produzione: architettura di riferimento . . . 147

29. SICUREZZA AVANZATA: AUTH, SESSIONI E ATTACCHI WEB 149

29.1 JWT avanzato: best practice e vulnerabilità . . . 149

29.2 CSRF: Cross-Site Request Forgery . . . 151

29.3 XSS: Cross-Site Scripting in API Node.js . . . 152

30. PERFORMANCE TUNING E MEMORY MANAGEMENT 154

30.1 Analisi memory leak con heap snapshot . . . 154

30.2 V8 optimization: JIT e Deoptimization . . . 156

30.3 Database query optimization . . . 157

31. CASO PRATICO: SISTEMA E-COMMERCE CON NODE.JS 159

31.1 Architettura del sistema 159

31.2 Gestione ordini: saga pattern 159

31.3 Integrazione Stripe e webhook 161

32. APPENDICE TECNICA ESTESA 164

32.1 Middleware Express riutilizzabili 164

32.2 Utility functions indispensabili 166

32.3 Tabella completa: status code HTTP 169

32.4 Configurazioni .env complete per ogni ambiente 169

32.5 Script npm utili per ogni progetto 171

32.6 Tabella performance: confronto soluzioni 173

32.7 Comandi Git indispensabili per il workflow Node.js 173

32.8 Regex patterns più utili in Node.js 175

33. OAUTH 2.0 E OPENID CONNECT 176

33.1 Flussi OAuth 2.0 principali 176

33.2 Authorization Code con PKCE — implementazione 176

33.3 API-to-API con Client Credentials 178

34. JOB QUEUE AVANZATA CON BULLMQ 181

34.1 Architettura BullMQ 181

34.2 Setup completo con Worker e Scheduler 181

34.3 Bull Board: dashboard di monitoraggio 183

35. INTERNAZIONALIZZAZIONE E LOCALIZZAZIONE (I18N) 185

35.1 i18next: la libreria standard per Node.js 185

35.2 Formattazione date, numeri e valute 187

35.3 Email multilingue con template 188

36. MIGRAZIONE E INTEGRAZIONE CON SISTEMI LEGACY 191

36.1 Strangler Fig Pattern 191

36.2 Integrazione con Java via REST e gRPC 192

36.3 Migrazione database graduale 192

37. CONCLUSIONI: NODE.JS NEL 2025 E OLTRE 195

37.1 Mappa delle competenze Node.js 195

37.2 Il percorso di apprendimento consigliato 195

37.3 Risorse per continuare ad imparare 197

37.4 L'ecosistema Node.js nel 2025: stato dell'arte 197

37.5 Domande frequenti (FAQ) 198

37.6 Ecosistema npm: pacchetti per categoria (2025) 200

1. STORIA E ORIGINI DI NODE.JS

Node.js è una delle tecnologie più influenti nella storia dello sviluppo software moderno. Nata nel 2009 da un'idea radicale — eseguire JavaScript lato server con un modello di I/O non bloccante — ha ridefinito il modo in cui si costruiscono applicazioni web scalabili, API, microservizi e strumenti di backend.

1.1 Ryan Dahl e la nascita nel 2009

Ryan Dahl, sviluppatore americano, presentava Node.js il **8 novembre 2009** alla European JSConf di Berlino. La demo in tempo reale — un server HTTP che gestiva migliaia di connessioni simultanee con pochissimo codice — stupì il pubblico. Il problema che Dahl voleva risolvere era concreto: i server web tradizionali (Apache, modello thread-per-request) sprecavano risorse bloccando il thread di esecuzione in attesa di operazioni I/O come letture disco o risposte di rete.

L'intuizione chiave fu combinare tre componenti: **V8** (il motore JavaScript di Google Chrome, scritto in C++, compilatore JIT velocissimo), **libuv** (libreria C per I/O asincrono cross-platform con event loop) e le API JavaScript già note agli sviluppatori web. Il risultato fu un runtime JavaScript che poteva gestire decine di migliaia di connessioni simultanee su hardware modesto, usando un solo thread con un event loop non bloccante.

1.2 Timeline storica completa

Anno	Versione / Evento	Significato
2009	Node.js 0.1 — European JSConf	Prima presentazione pubblica di Ryan Dahl; motore V8 + libuv + event loop
2010	npm 0.1 (Isaac Schlueter)	Nasce il Node Package Manager: ecosistema di pacchetti riutilizzabili
2011	Node.js 0.4 — Windows support	Prima versione stabile su Windows; supporto Microsoft; crescita enterprise
2012	Node.js 0.8 — Ryan Dahl lascia	Dahl cede la leadership a Isaac Schlueter; Joyent gestisce il progetto
2013	Ghost, PayPal, LinkedIn in prod.	Adozione enterprise massiccia; PayPal riduce codice del 33%, +2x velocita
2014	io.js fork	Comunita' si divide: io.js per releases piu' veloci vs Node.js di Joyent
2015	Node.js Foundation; merge con io.js	Fondazione indipendente; riunificazione; nasce ciclo LTS/Current
2015	Node.js 4.0 LTS	Prima Long Term Support; ES6 parziale; stabilita' enterprise
2016	Node.js 6 — npm left-pad incident	Crisi ecosistema: 11 righe di codice rimosse rompono migliaia di progetti
2018	Node.js 10 — npm audit	Sicurezza: npm audit; V8 6.6; performance massicce su I/O
2019	Node.js 12 — ES Modules sperimentali	Supporto nativo ESM; worker_threads stabili; TLS 1.3
2020	Node.js 14 LTS — ESM stabile	ES Modules ufficialmente stabili; top-level await; optional chaining
2021	Node.js 16 / 17 — Apple Silicon	Supporto ARM64 nativo; corepack; fetch API sperimentale
2022	Node.js 18 LTS — fetch nativo	fetch() globale senza dipendenze; test runner integrato; V8 10.2
2023	Node.js 20 LTS — permissions model	Nuovo sistema di permessi (sperimentale); test coverage; SEA
2024	Node.js 22 LTS — stable fetch + WebSocket	WebSocket client nativo; require(ESM); glob integrato; performance V8 12.4
2024	Ryan Dahl crea Deno 2.0	Dahl torna con Deno: sicurezza by default, TypeScript nativo, compatibile npm

1.3 Il fork io.js e la governance moderna

Nel dicembre 2014, un gruppo di sviluppatori insoddisfatti del ritmo di sviluppo di Joyent creò il fork **io.js**. La motivazione era tecnica e politica insieme: rilasciare versioni più frequenti, aggiornare V8 più rapidamente e adottare un modello di governance aperta con un Technical Steering Committee. Il fork evidenziò le tensioni tra sviluppo corporate-driven e open source community-driven.

La crisi si risolse nel settembre 2015 con la fondazione della **Node.js Foundation** — oggi parte di **OpenJS Foundation** — e la riunificazione dei due progetti. Node.js 4.0, primo rilascio unificato, includeva V8 4.5, la stessa versione di io.js 3.x. Da allora, il progetto segue un ciclo di rilascio formale: versioni **Current** (ultime feature, 6 mesi), versioni **Active LTS** (supporto 18 mesi) e **Maintenance LTS** (12 mesi aggiuntivi di patch critiche).

1.4 Deno e Bun: la concorrenza moderna

Runtime	Autore / Anno	Punti di forza	Compatibilità Node.js
Node.js	Ryan Dahl, 2009; OpenJS Foundation	Ecosistema immenso; stabilità enterprise; npm	Nativa — è il riferimento
Deno	Ryan Dahl, 2018; Deno Land Inc.	TypeScript nativo; sicurezza permessi; Standard Library; Deno KV	Deno 2.0: compatibile con npm e molti moduli Node
Bun	Jarred Sumner, 2022; Oven.sh	Velocissimo (Zig + JavaScriptCore); bundler integrato; test runner; compatibile npm	Alta compatibilità API Node.js; drop-in per molti progetti

■ **INFO** Node.js rimane il leader indiscusso con oltre il 60% di market share tra i runtime JavaScript server-side (Stack Overflow Survey 2024). npm conta oltre 2,5 milioni di pacchetti e 40 miliardi di download mensili.

Fonti: nodejs.org/en/about/history; Ryan Dahl JSConf EU 2009 (youtube.com/watch?v=ztspvPYybIY); OpenJS Foundation openjs.foundation; Stack Overflow Developer Survey 2024 survey.stackoverflow.co/2024; npmjs.com/about (statistiche npm 2024)

2. INSTALLAZIONE E SETUP DELL'AMBIENTE

Prima di scrivere una sola riga di codice Node.js occorre configurare un ambiente di sviluppo robusto. Questa sezione copre tutte le piattaforme e i metodi di installazione, con le best practice per gestire versioni multiple.

2.1 Metodi di installazione

Esistono tre approcci principali per installare Node.js, ognuno con vantaggi diversi a seconda del contesto:

Metodo 1 — Installer ufficiale (principianti)

■ BASH | Verifica installazione Node.js

```bash
# Scarica da: https://nodejs.org/en/download
# Scegli sempre la versione LTS (Long Term Support)
# Attuale LTS: Node.js 22.x (Codename: Jod)

# Verifica installazione dopo installer:
node --version      # output: v22.x.x
npm --version       # output: 10.x.x
npx --version       # incluso con npm 5.2+
```

Metodo 2 — nvm: Node Version Manager (consigliato)

nvm (Node Version Manager) è lo strumento consigliato per sviluppatori professionali: permette di installare, gestire e passare tra versioni diverse di Node.js senza conflitti di sistema.

■ BASH | Installazione e uso di nvm (Node Version Manager)

```bash
# ■■ Linux / macOS ■■■
```

■ BASH | fnm: alternativa moderna a nvm scritta in Rust

```
# fnm e' scritto in Rust: installazione istantanea, uso rapidissimo
# Compatibile con .nvmrc e .node-version

# macOS (Homebrew)
brew install fnm

# Linux / Windows (script)
curl -fsSL https://fnm.vercel.app/install | bash

# Uso (identico a nvm)
fnm install 22
fnm use 22
fnm default 22
```

2.2 Struttura di un progetto Node.js

■ BASH | Struttura progetto Node.js consigliata

```
# Crea directory e inizializza progetto
mkdir mio-progetto
cd mio-progetto
npm init -y   # crea package.json con valori default

# Struttura consigliata per progetti Express/API:
mio-progetto/
■■■ src/
■   ■■■ index.js          # entry point
■   ■■■ routes/           # definizioni rotte Express
■   ■   ■■■ users.js
■   ■   ■■■ products.js
■   ■■■ controllers/      # logica business
■   ■■■ models/           # definizioni dati (Mongoose/Sequelize)
■   ■■■ middleware/       # middleware custom
■   ■■■ services/         # logica riutilizzabile
■   ■■■ utils/            # utility functions
■■■ tests/                # test con Jest/Vitest
■■■ .env                  # variabili d ambiente (NON committare!)
■■■ .env.example          # template variabili (da committare)
■■■ .gitignore
■■■ package.json
■■■ package-lock.json     # lock file (committare sempre)
■■■ README.md
```

2.3 package.json: anatomia completa

■ JSON | package.json commentato con tutte le sezioni principali

```
{
  "name": "mio-progetto",
  "version": "1.0.0",
  "description": "Descrizione del progetto",
  "main": "src/index.js",
  "type": "module",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js",
    "test": "jest --coverage",
    "test:watch": "jest --watch",
    "lint": "eslint src/",
    "lint:fix": "eslint src/ --fix",
    "build": "tsc",
    "docker:up": "docker-compose up -d"
  },
  "keywords": ["nodejs", "api", "rest"],
  "author": "Mario Rossi <mario@example.com>",
  "license": "MIT",
  "engines": {
    "node": ">=22.0.0",
    "npm": ">=10.0.0"
  },
  "dependencies": {
    "express": "^4.18.2",
    "dotenv": "^16.4.5",
    "mongoose": "^8.3.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.0",
    "jest": "^29.7.0",
    "eslint": "^9.3.0"
  }
}
```

2.4 Editor e strumenti di sviluppo

Strumento	Tipo	Scopo	Note
VS Code	Editor	IDE principale	Extensioni: ESLint, Prettier, REST Client, GitLens
Nodemon	Dev tool	Riavvio automatico su modifica file	npm install -D nodemon
ESLint	Linter	Analisi statica codice JS/TS	Configurazione con eslint.config.js
Prettier	Formatter	Formattazione codice consistente	Integra con ESLint
Jest / Vitest	Testing	Unit e integration test	Vitest piu' veloce con ESM
Thunder Client	HTTP client	Test API REST in VS Code	Alternativa a Postman
dotenv	Libreria	Carica .env in process.env	npm install dotenv
Docker Desktop	Container	Ambienti isolati e deploy	Fondamentale per produzione

✓ **BEST PRACTICE** Usa sempre un file **.nvmrc** nella root del progetto con il numero di versione Node.js (es. '22'). Questo garantisce che tutti i collaboratori usino la stessa versione eseguendo semplicemente nvm use.

Fonti: nodejs.org/en/download; github.com/nvm-sh/nvm; github.com/Schniz/fnm; docs.npmjs.com/cli/v10/configuring-npm/package-json; code.visualstudio.com

3. ARCHITETTURA: EVENT LOOP, V8 E LIBUV

Capire l'architettura di Node.js è fondamentale per scrivere codice performante e per diagnosticare problemi. Node.js non è semplicemente 'JavaScript lato server': è un ecosistema di componenti C++ e JavaScript che cooperano per offrire I/O non bloccante ad altissima concorrenza.

3.1 I componenti fondamentali

Componente	Linguaggio	Ruolo	Versione (Node.js 22)
V8 Engine	C++	Compila ed esegue JavaScript (JIT compiler); garbage collection	V8 12.4
libuv	C	Event loop, I/O asincrono cross-platform, thread pool	libuv 1.48
Node.js bindings	C++ / JS	Bridge tra API JS e funzioni native C++	Built-in
npm / node_modules	JavaScript	Gestione pacchetti; 2,5M+ pacchetti disponibili	npm 10.x
V8 Turbofan	C++	Compilatore JIT ottimizzante; hotspot detection	Integrato in V8 12.4
Orinoco GC	C++	Garbage collector generazionale concorrente	Integrato in V8 12.4

3.2 L'Event Loop: funzionamento dettagliato

L'event loop è il cuore di Node.js. A differenza dei server multi-thread tradizionali che assegnano un thread a ogni richiesta, Node.js usa **un singolo thread** con un ciclo continuo che processa eventi e callback. Le operazioni I/O (rete, filesystem, database) vengono delegate al thread pool di libuv o al kernel OS (epoll/kqueue/IOCP) e non bloccano mai il thread principale.

L'event loop di Node.js è diviso in **sei fasi**, ognuna con una propria coda di callback:

Fase	Coda	Cosa processa
1 — timers	Timer queue	Callback di setTimeout() e setInterval() con delay scaduto
2 — pending callbacks	I/O callback queue	Callback I/O differite dalla fase precedente (errori TCP, ecc.)
3 — idle, prepare	Interno	Solo uso interno di Node.js
4 — poll	Poll queue	Recupera nuovi eventi I/O; esegue le callback I/O. Se vuota: attende
5 — check	Check queue	Callback di setImmediate()

Fase	Coda	Cosa processa
6 — close callbacks	Close queue	Callback di chiusura socket ('close' events)
Tra ogni fase	microtask queue	process.nextTick() e Promise.then() — priorita' massima

■ JS | event-loop-demo.js — ordine di esecuzione delle code

```
// Dimostrazione ordine di esecuzione dell event loop
console.log('1 - Sincrono');

setTimeout(() => console.log('5 - setTimeout (timers)'), 0);

setImmediate(() => console.log('4 - setImmediate (check)'));

Promise.resolve().then(() => console.log('3 - Promise microtask'));

process.nextTick(() => console.log('2 - nextTick (micro)'));

console.log('1b - Sincrono ancora');

// Output (SEMPRE in questo ordine):
// 1 - Sincrono
// 1b - Sincrono ancora
// 2 - nextTick (micro)      <-- microtask prima di tutto
// 3 - Promise microtask     <-- microtask
// 4 - setImmediate (check)  <-- check phase
// 5 - setTimeout (timers)   <-- timers phase
```

3.3 I/O non bloccante: il vantaggio competitivo

■ JS | confronto bloccante vs non-bloccante

```
// ■■ MODELLO BLOCCANTE (PHP, Python sincrono) ■■■■■■■■■■■■■■■■■■■■■
// Pseudocode: ogni richiesta blocca il thread
function handleRequest(req) {
  const data = readFileSync('/etc/data.txt');  // BLOCCA: 50ms
  const result = queryDB(data);               // BLOCCA: 100ms
  return result;
}
// 1000 richieste concorrenti = 1000 thread = ~150ms * 1000 = 150s

// ■■ MODELLO NON BLOCCANTE Node.js ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
async function handleRequest(req) {
  const data = await fs.readFile('/etc/data.txt');  // NON BLOCCA
  const result = await db.query(data);              // NON BLOCCA
  return result;
}
// 1000 richieste concorrenti = 1 thread = ~150ms totali
// Il thread e' libero durante l attesa I/O
```

3.4 Il thread pool di libuv

Non tutte le operazioni possono essere delegate al kernel OS. Le operazioni su filesystem, la crittografia (crypto), la compressione (zlib) e i DNS lookup vengono eseguite nel **thread pool di libuv**: un pool di thread C++ che lavorano in parallelo e restituiscono il risultato alla coda dell'event loop.

■ JS | thread-pool.js — configurazione e misura UV_THREADPOOL_SIZE

```
// Thread pool size: default 4, configurabile con UV_THREADPOOL_SIZE
// Linux/macOS: esporta prima di avviare node
// UV_THREADPOOL_SIZE=8 node server.js

// Oppure via codice (prima di usare crypto/fs):
process.env.UV_THREADPOOL_SIZE = '8';

const crypto = require('crypto');
const { performance } = require('perf_hooks');

// Con UV_THREADPOOL_SIZE=1: operazioni sequenziali
// Con UV_THREADPOOL_SIZE=4: 4 hash in parallelo
const NUM_OPS = 4;
const start = performance.now();

const promises = Array.from({ length: NUM_OPS }, () =>
  new Promise(resolve =>
    crypto.pbkdf2('password', 'salt', 100000, 64, 'sha512',
      (err, derivedKey) => resolve(derivedKey.toString('hex'))
    )
  )
);

Promise.all(promises).then(() => {
  const elapsed = (performance.now() - start).toFixed(0);
  console.log(`${NUM_OPS} PBKDF2 completate in ${elapsed}ms`);
});
// Con size=1: ~2000ms; con size=4: ~500ms
```

3.5 Worker Threads: parallelismo CPU in Node.js

Node.js 10+ include il modulo worker_threads (stabile da Node.js 12) per eseguire JavaScript in thread separati. Utile per operazioni CPU-intensive che bloccerebbero l'event loop: calcoli matematici, parsing di file enormi, compressione, machine learning.

■ JS | worker-main.js — calcolo CPU-intensive in Worker Thread

```
// worker-main.js
const { Worker, isMainThread, parentPort, workerData }
  = require('worker_threads');

if (isMainThread) {
  // Thread principale: crea worker
  const worker = new Worker(__filename, {
    workerData: { n: 40 }  // dato passato al worker
  });

  worker.on('message', result => {
    console.log(`Fibonacci(${workerData?.n || 40}) = ${result}`);
  });
  worker.on('error', err => console.error(err));
  worker.on('exit', code => {
    if (code !== 0) console.error(`Worker uscito con codice ${code}`);
  });

} else {
  // Thread worker: esegue calcolo CPU-intensive
  function fib(n) {
    if (n <= 1) return n;
    return fib(n - 1) + fib(n - 2);
  }
  parentPort.postMessage(fib(workerData.n));
}
```

Fonti: nodejs.org/en/docs/guides/event-loop-timers-and-nexttick; libuv.org; v8.dev; nodejs.org/api/worker_threads.html; Bert Belder 'Everything You Need to Know About Node.js Event Loop' 2016

4. SINTASSI JAVASCRIPT MODERNA IN NODE.JS

Node.js supporta JavaScript moderno (ES2015+, ES2024) attraverso il motore V8. Questa sezione copre la sintassi fondamentale con esempi pratici, partendo dalle basi e arrivando alle feature più recenti.

4.1 Tipi di dati, variabili e scope

■ JS | variabili-tipi.js — variabili, tipi, destructuring, spread

```
// ■■ DICHIARAZIONE VARIABILI ■■■
```

■ JS | funzioni.js — dichiarazioni, arrow, closure, HOF

```
// ■■ FUNCTION DECLARATION (hoisted) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
function somma(a, b) { return a + b; }

// ■■ FUNCTION EXPRESSION ■■■■■■■■■■■■■■■■■■■
```

■ JS | async-await.js — Promise, async/await, gestione errori

```
// ■■ PROMISE BASE ■■■
```

```
// In file .mjs o con "type":"module" nel package.json:
// const data = await fetchDati();  // nessun wrapping in async needed
```

4.4 Classi, Moduli ES e nuove sintassi ES2022-2024

■ JS | modern-js.js — classi, ES Modules, optional chaining, ES2022-2024

```
// ■■ CLASSI ES6 ■■■
```

```
arr.at(-1);   // 5 (ultimo elemento)
arr.at(-2);   // 4

Object.hasOwn(user, 'profilo');  // true (piu' sicuro di hasOwnProperty)

// ■■ Error cause (ES2022) ■■■
```

Fonti: tc39.es/ecma262/ (ECMAScript spec); developer.mozilla.org/en-US/docs/Web/JavaScript; node.green (compatibilita' ES in Node.js); nodejs.org/api/esm.html (ES Modules in Node.js)

5. NPM E L'ECOSISTEMA DI PACCHETTI

npm (Node Package Manager) è molto più di uno strumento per installare librerie: è il più grande registry di pacchetti software al mondo, con oltre **2,5 milioni di pacchetti** e più di **40 miliardi di download mensili**. Padroneggiare npm è essenziale per qualsiasi sviluppatore Node.js.

5.1 Comandi npm fondamentali

■ BASH | Comandi npm fondamentali per ogni scenario

```
# ■■ INIZIALIZZAZIONE PROGETTO ■■■
```

5.2 Semantic Versioning (SemVer)

npm usa il **Semantic Versioning**: ogni versione è MAJOR.MINOR.PATCH. MAJOR: breaking changes; MINOR: nuove feature backward-compatible; PATCH: bug fix. I prefissi nel package.json controllano l'aggiornamento automatico:

Sintassi	Significato	Esempio	Aggiorna a
4.18.2	Versione esatta	express@4.18.2	Solo 4.18.2
^4.18.2	Compatible (caret)	"^4.18.2"	4.18.x fino a <5.0.0
~4.18.2	Approximately (tilde)	"~4.18.2"	4.18.x (solo patch)
>=4.0.0	Greater or equal	">=4.0.0"	4.0.0 o superiore
4.x	Wildcard	"4.x"	Qualsiasi 4.y.z
*	Any	"*"	Ultima versione (pericoloso!)
latest	Tag npm	"latest"	Ultimo tag 'latest'

5.3 package-lock.json: perché è fondamentale

■ **INFO** Il **package-lock.json** fissa le versioni esatte di TUTTE le dipendenze (dirette e transitive). Va SEMPRE committato nel repository. Garantisce che tutti i collaboratori e il server CI/CD installino esattamente le stesse versioni. Usa npm ci invece di npm install nelle pipeline CI per installazioni deterministiche.

5.4 Alternative a npm: yarn e pnpm

Tool	Anno	Vantaggi principali	Installazione
npm	2010	Built-in con Node.js; standard de facto; audit integrato	Incluso con Node.js
yarn classic	2016 (Facebook)	Installazioni parallele; offline cache; workspaces	npm install -g yarn
yarn berry (3+)	2020	PnP (no node_modules); zero-install; plugin system	corepack enable && yarn set version stable
pnpm	2017	Installazioni veloci; disco condiviso tra progetti; strict isolation	npm install -g pnpm
bun install	2023	Velocissimo (Zig); compatibile package.json	Installato con Bun runtime

5.5 Pacchetti essenziali per ogni categoria

Categoria	Pacchetto	Versione	Uso principale
Web framework	express	4.x	API REST, middleware, routing classico
Web framework	fastify	4.x	API ad alte performance, schema validation
Web framework	hono	4.x	Ultraleggero, edge-ready, TypeScript nativo
ORM/Database	mongoose	8.x	MongoDB ODM con schema e validazione
ORM/Database	prisma	5.x	ORM multi-DB con type-safety e migrations
ORM/Database	drizzle-orm	0.3x	ORM leggero type-safe per SQL
Auth	passport	0.7.x	Strategie autenticazione (JWT, OAuth, local)
Auth	jose	5.x	JWT signing/verification senza dipendenze
Validazione	zod	3.x	Schema validation TypeScript-first
Validazione	joi	17.x	Schema validation con API fluente
HTTP Client	axios	1.x	HTTP client con intercettori e progress
HTTP Client	got	14.x	HTTP client moderno solo ESM
Logging	winston	3.x	Logging flessibile con transport multipli
Logging	pino	9.x	Logging ultra-performante JSON
Testing	jest	29.x	Test runner completo, mocking integrato
Testing	vitest	1.x	Piu' veloce di Jest, ESM nativo, watch mode
Caching	ioredis	5.x	Client Redis con pipeline e cluster
Queue	bullmq	5.x	Job queue con Redis, retry, scheduling
Email	nodemailer	6.x	Invio email SMTP, template, allegati
Websocket	socket.io	4.x	WebSocket + fallback; rooms; namespace

Fonti: npmjs.com/about; semver.org; yarnpkg.com; pnpm.io; docs.npmjs.com/cli/v10/commands; npmtrends.com (statistiche download comparate)

6. CORE MODULES: I MODULI BUILT-IN

Node.js include una libreria standard ricca di moduli built-in (core modules) disponibili senza installare nulla. Questa sezione copre i moduli più usati con esempi pratici.

6.1 fs: filesystem sincrono e asincrono

■ JS | fs-examples.js — filesystem: read, write, dir, watch, stream

```
const fs = require('fs');
const fsp = require('fs').promises;   // fs/promises (Node 10+)
// oppure: import { readFile, writeFile } from 'fs/promises';

// ■■ LETTURA FILE ■■■
```

■ JS | path-examples.js — path.join, resolve, dirname, basename

```
const path = require('path');

// REGOLA: usare sempre path.join(), MAI concatenare con '/'
// (su Windows il separatore e' '' )

path.join('/home', 'user', 'file.txt');  // /home/user/file.txt
path.join(__dirname, '..', 'config.json'); // percorso relativo al file corrente

path.resolve('src', 'index.js');  // percorso assoluto dalla CWD
path.dirname('/home/user/file.txt');  // /home/user
path.basename('/home/user/file.txt');  // file.txt
path.basename('/home/user/file.txt', '.txt');  // file
path.extname('/home/user/file.txt');  // .txt
path.parse('/home/user/file.txt');
// { root:'/', dir:'/home/user', base:'file.txt', ext:'.txt', name:'file' }

// __dirname e __filename (solo CommonJS)
console.log(__dirname);   // directory del file corrente
console.log(__filename);  // percorso completo del file corrente

// Equivalente in ES Modules:
import { fileURLToPath } from 'url';
import { dirname } from 'path';
const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
```

6.3 events: EventEmitter pattern

■ JS | events.js — EventEmitter: on, once, off, emit, error

```
const { EventEmitter } = require('events');

// ■■ EVENT■■ITTER BASE ■■■
```

■ JS | streams.js — Transform, pipeline, Readable.from, for-await-of

```
const { Transform, pipeline } = require('stream');
const { promisify } = require('util');
const fs = require('fs');
const zlib = require('zlib');

const pipelineAsync = promisify(pipeline);

// ■■ TRANSFORM STREAM: trasforma dati on-the-fly ■■■■■■■■■■■■■■■■■■■■
class ContaParole extends Transform {
  constructor() {
    super({ objectMode: false });
    this.conteggio = 0;
  }

  _transform(chunk, encoding, callback) {
    const parole = chunk.toString().split(/\s+/).filter(Boolean);
    this.conteggio += parole.length;
    callback(null, chunk);  // passa il chunk originale
  }

  _flush(callback) {
    this.push(`
--- Totale parole: ${this.conteggio} ---
`);
    callback();
  }
}

// ■■ PIPELINE: composizione streams ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
async function comprimi(input, output) {
  await pipelineAsync(
    fs.createReadStream(input),
    new ContaParole(),
    zlib.createGzip(),
    fs.createWriteStream(output)
  );
  console.log(`Compresso: ${input} -> ${output}`);
}

// ■■ READABLE STREAM da array/generator ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
const { Readable } = require('stream');
const readableFromArray = Readable.from(['riga1
', 'riga2
', 'riga3
']);
for await (const chunk of readableFromArray) {
  process.stdout.write(chunk);
}
```

6.5 Altri core modules importanti

Modulo	Uso principale	Esempio chiave
http / https	Server HTTP/HTTPS nativo senza framework	http.createServer((req,res) => res.end('OK'))
net	TCP server/client raw	net.createServer(socket => socket.pipe(socket))
crypto	Hash, HMAC, cifratura, chiavi	crypto.createHash('sha256').update(data).digest('hex')
os	Informazioni sistema operativo	os.cpus(), os.totalmem(), os.hostname()
url / querystring	Parsing URL e query string	new URL('https://example.com/path?q=1')
child_process	Esecuzione comandi shell e processi figlio	exec('ls -la', callback); spawn('python3', ['script.py'])
cluster	Multi-processo per usare tutti i core CPU	cluster.fork() per ogni CPU
util	Utility: promisify, inspect, format, deprecate	util.promisify(fs.readFile)
assert	Asserzioni per testing e invarianti	assert.strictEqual(a, b)
buffer	Dati binari raw (base64, hex, UTF-8)	Buffer.from('hello', 'utf8').toString('base64')
dgram	Socket UDP	dgram.createSocket('udp4')
dns	Risoluzione DNS asincrona	dns.resolve4('nodejs.org', callback)
readline	Lettura input riga per riga da stream	readline.createInterface({input: process.stdin})
timers/promises	Timer come Promise (Node 15+)	await timers.setTimeout(1000); await timers.setImmediate()

Fonti: nodejs.org/api/ (documentazione completa di tutti i core modules); nodejs.org/api/stream.html; nodejs.org/api/events.html; nodejs.org/api/fs.html; nodejs.org/api/crypto.html

7. EXPRESS.JS E API REST

Express.js è il framework web più usato nell'ecosistema Node.js con oltre **30 milioni di download settimanali** (npm, 2024). Minimalista e flessibile, fornisce routing, middleware e gestione delle richieste HTTP con pochissimo overhead.

7.1 Server Express completo con best practice

■ JS | src/index.js — server Express completo con helmet, cors, rate limiting

```
// src/index.js — Server Express con struttura professionale
const express = require('express');
const helmet = require('helmet');
const cors = require('cors');
const morgan = require('morgan');
const rateLimit = require('express-rate-limit');
require('dotenv').config();

const userRoutes = require('./routes/users');
const { errorHandler } = require('./middleware/errorHandler');

const app = express();
const PORT = process.env.PORT || 3000;

// ■■ MIDDLEWARE GLOBALI ■■■
```

```
  server.close(() => {
    console.log('Server chiuso');
    process.exit(0);
  });
});

module.exports = app; // per i test
```

7.2 CRUD API RESTful completa

■ JS | routes/users.js + controllers/usersController.js — CRUD completo

```
// src/routes/users.js
const express = require('express');
const router = express.Router();
const { validateUser } = require('../middleware/validate');
const { requireAuth } = require('../middleware/auth');
const UsersController = require('../controllers/usersController');

// GET /api/v1/users — lista utenti con paginazione
router.get('/', UsersController.getAll);

// GET /api/v1/users/:id — singolo utente
router.get('/:id', UsersController.getById);

// POST /api/v1/users — crea utente (richiede validazione)
router.post('/', validateUser, UsersController.create);

// PUT /api/v1/users/:id — aggiorna (auth richiesta)
router.put('/:id', requireAuth, validateUser, UsersController.update);

// PATCH /api/v1/users/:id — aggiornamento parziale
router.patch('/:id', requireAuth, UsersController.partialUpdate);

// DELETE /api/v1/users/:id — elimina (auth richiesta)
router.delete('/:id', requireAuth, UsersController.delete);

module.exports = router;

// src/controllers/usersController.js
const User = require('../models/User');

class UsersController {
  // GET tutti con paginazione
  static async getAll(req, res, next) {
    try {
      const page = Math.max(1, parseInt(req.query.page) || 1);
      const limit = Math.min(100, parseInt(req.query.limit) || 20);
      const skip = (page - 1) * limit;
      const filter = req.query.role ? { role: req.query.role } : {};

      const [users, total] = await Promise.all([
        User.find(filter).skip(skip).limit(limit).select('-password'),
        User.countDocuments(filter),
      ]);

      res.json({
        data: users,
        pagination: {
          page, limit, total,
          pages: Math.ceil(total / limit),
          hasNext: page * limit < total,
          hasPrev: page > 1,
        },
      });
    } catch (err) {
      next(err);  // passa all error handler centralizzato
    }
  }

  static async getById(req, res, next) {
    try {
      const user = await User.findById(req.params.id).select('-password');
      if (!user) {
        return res.status(404).json({ error: 'Utente non trovato' });
```

```
      }
      res.json({ data: user });
    } catch (err) {
      next(err);
    }
  }

  static async create(req, res, next) {
    try {
      const { nome, email, password, role } = req.body;
      const user = new User({ nome, email, password, role });
      await user.save();
      // Non esporre la password nella risposta
      const { password: _, ...userSafe } = user.toObject();
      res.status(201).json({ data: userSafe });
    } catch (err) {
      if (err.code === 11000) {
        return res.status(409).json({ error: 'Email gia' in uso' });
      }
      next(err);
    }
  }

  static async delete(req, res, next) {
    try {
      const user = await User.findByIdAndDelete(req.params.id);
      if (!user) return res.status(404).json({ error: 'Non trovato' });
      res.status(204).send();
    } catch (err) { next(err); }
  }
}

module.exports = UsersController;
```

7.3 Middleware: validazione, autenticazione, errori

■ JS | middleware — validazione Zod, autenticazione JWT, error handler

```
// src/middleware/validate.js — validazione con Zod
const { z } = require('zod');

const userSchema = z.object({
  nome: z.string().min(2).max(50),
  email: z.string().email(),
  password: z.string().min(8).regex(
    /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/,
    'Password: almeno 1 maiuscola, 1 minuscola, 1 cifra'
  ),
  role: z.enum(['user', 'admin', 'moderator']).default('user'),
});

const validateUser = (req, res, next) => {
  const result = userSchema.safeParse(req.body);
  if (!result.success) {
    return res.status(400).json({
      error: 'Dati non validi',
      details: result.error.issues.map(i => ({
        campo: i.path.join('.'),
        messaggio: i.message,
      })),
    });
  }
  req.body = result.data;  // dati sanitizzati
  next();
};

// src/middleware/auth.js — JWT Authentication
const jose = require('jose');

const requireAuth = async (req, res, next) => {
  try {
    const header = req.headers.authorization;
    if (!header?.startsWith('Bearer ')) {
      return res.status(401).json({ error: 'Token mancante' });
    }
    const token = header.slice(7);
    const secret = new TextEncoder().encode(process.env.JWT_SECRET);
    const { payload } = await jose.jwtVerify(token, secret);
    req.user = payload;
    next();
  } catch (err) {
    res.status(401).json({ error: 'Token non valido o scaduto' });
  }
};

// src/middleware/errorHandler.js — gestione centralizzata errori
const errorHandler = (err, req, res, next) => {
  console.error(err);
  const status = err.status || err.statusCode || 500;
  res.status(status).json({
    error: err.message || 'Errore interno del server',
    ...(process.env.NODE_ENV === 'development' && { stack: err.stack }),
  });
};

module.exports = { validateUser, requireAuth, errorHandler };
```

Fonti: expressjs.com; npmjs.com/package/helmet; npmjs.com/package/express-rate-limit; zod.dev; github.com/panva/jose (JWT); npmjs.com/package/morgan; npmjs.com/package/cors

8. DATABASE: MONGODB, POSTGRESQL E ORM

Node.js si integra con tutti i principali database. Questa sezione copre le due combinazioni più diffuse: **MongoDB con Mongoose** (database documentale, schema flessibile) e **PostgreSQL con Prisma** (database relazionale, type-safe ORM).

8.1 MongoDB con Mongoose

■ JS | mongoose.js — connessione, schema, hooks, CRUD, aggregation

```
// npm install mongoose
const mongoose = require('mongoose');

// ■■ CONNESSIONE ■■■
```

```
// Hook pre-save: hash password prima del salvataggio
userSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const bcrypt = require('bcrypt');
  this.password = await bcrypt.hash(this.password, 12);
  next();
});

// Metodo di istanza
userSchema.methods.verificaPassword = async function(candidata) {
  const bcrypt = require('bcrypt');
  return bcrypt.compare(candidata, this.password);
};

const User = mongoose.model('User', userSchema);

// ■■ OPERAZIONI CRUD ■■■
```

■ PRISMA | schema.prisma — modello dati con relazioni e enum

```
// prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique
  nome      String
  password  String
  role      Role     @default(USER)
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
  posts     Post[]
  profilo   Profilo?
}

model Post {
  id        Int      @id @default(autoincrement())
  titolo    String
  contenuto String?
  pubblicato Boolean @default(false)
  autore    User    @relation(fields: [autoreId], references: [id])
  autoreId  Int
  tags      String[]
  createdAt DateTime @default(now())
}

model Profilo {
  id     Int    @id @default(autoincrement())
  bio    String?
  user   User   @relation(fields: [userId], references: [id])
  userId Int    @unique
}

enum Role {
  USER
  ADMIN
  MODERATOR
}
```

■ JS | prisma-client.js — CRUD, relazioni, transazioni, aggregazioni

```
// Applica migrazioni al DB
// npx prisma migrate dev --name init
// npx prisma generate   (genera client TypeScript)

// src/lib/prisma.js — singleton client
const { PrismaClient } = require('@prisma/client');

const globalForPrisma = global;
const prisma = globalForPrisma.prisma ?? new PrismaClient({
  log: process.env.NODE_ENV === 'development'
    ? ['query', 'info', 'warn', 'error']
    : ['error'],
});
if (process.env.NODE_ENV !== 'production') globalForPrisma.prisma = prisma;

module.exports = prisma;

// ■■ OPERAZIONI CRUD CON PRISMA ■■■
```

9. CYBERSECURITY IN NODE.JS

La sicurezza è una responsabilità critica per qualsiasi applicazione Node.js esposta su internet. Questa sezione analizza le vulnerabilità più comuni, gli attacchi documentati e le contromisure concrete, seguendo le linee guida OWASP Top 10 2023 e le best practice della Node.js Security Working Group.

■ **SICUREZZA** Le tecniche offensive descritte in questa sezione sono presentate a scopo educativo e difensivo. L'utilizzo non autorizzato su sistemi altrui costituisce reato.

9.1 OWASP Top 10 in Node.js: mappatura completa

OWASP 2023	Vulnerabilità Node.js	Impatto	Priorità
A01 — Broken Access Control	Mancata verifica JWT; IDOR su MongoDB _id; route non protette	Accesso dati non autorizzati	CRITICA
A02 — Cryptographic Failures	MD5/SHA1 per password; dati sensibili in log; env in repository	Furto credenziali; data breach	CRITICA
A03 — Injection	NoSQL injection (MongoDB); SQL injection; Command injection (child_process)	Esfiltrazione dati; RCE	CRITICA
A04 — Insecure Design	Rate limiting assente; endpoint di debug esposti; CORS * in produzione	Brute force; DoS; data leak	ALTA
A05 — Security Misconfiguration	Variabili debug attive; stack trace esposta; header X-Powered-By	Info disclosure	ALTA
A06 — Vulnerable Components	Pacchetti npm con CVE note; typosquatting; supply chain attacks	RCE; data breach	ALTA
A07 — Auth Failures	JWT senza scadenza; sessioni non invalidate; password deboli permesse	Account takeover	ALTA
A08 — Software Integrity	npm senza lock file; script di install malevoli; CI/CD non firmato	Supply chain attack	MEDIA
A09 — Logging Failures	Password nei log; log strutturati con dati sensibili; log injection	Data breach; non-repudiation	MEDIA
A10 — SSRF	fetch() su URL forniti dall utente; webhook senza whitelist	Port scan interno; metadata cloud	MEDIA

9.2 NoSQL Injection (MongoDB)

La **NoSQL Injection** è l'equivalente MongoDB della SQL injection. Sfrutta il fatto che MongoDB accetta oggetti JavaScript nelle query: un attaccante può inserire operatori MongoDB ($ne, $gt, $where) per manipolare le query e bypassare autenticazioni.

■ JS | nosql-injection.js — vulnerabilità, attacco e protezione

```
// ■■ VULNERABILE: login senza sanitizzazione ■■■■■■■■■■■■■■■■■■■■■■■■
app.post('/login-vulnerabile', async (req, res) => {
  const { email, password } = req.body;
  // PERICOLOSO: se req.body = { email: {$ne: ''}, password: {$ne: ''} }
  // trova QUALSIASI utente nel DB -> bypass autenticazione!
  const user = await User.findOne({ email, password });
  if (user) res.json({ token: 'jwt...' });
});

// ■■ ATTACCO: body inviato dall attaccante ■■■■■■■■■■■■■■■■■■■■■■■■■■■
// POST /login-vulnerabile
// Content-Type: application/json
// {
//   "email": { "$ne": "" },
//   "password": { "$ne": "" }
// }
// Risultato: accede come primo utente del DB!

// ■■ SICURO: validazione + hashing separato ■■■■■■■■■■■■■■■■■■■■■■■■■■
const mongoSanitize = require('express-mongo-sanitize');
app.use(mongoSanitize());  // rimuove $ e . dalle chiavi request

app.post('/login', async (req, res) => {
  // 1. Valida il tipo (string, non object)
  const { email, password } = req.body;
  if (typeof email !== 'string' || typeof password !== 'string') {
    return res.status(400).json({ error: 'Input non valido' });
  }

  // 2. Query con email (stringa sanitizzata) — password mai in query DB
  const user = await User.findOne({ email: email.toLowerCase() })
    .select('+password');
  if (!user) {
    // Stesso tempo di risposta se utente non esiste (timing attack)
    await bcrypt.hash('dummy', 12);
    return res.status(401).json({ error: 'Credenziali non valide' });
  }

  // 3. Verifica password con bcrypt (mai confronto in chiaro!)
  const ok = await bcrypt.compare(password, user.password);
  if (!ok) return res.status(401).json({ error: 'Credenziali non valide' });

  res.json({ token: generaJWT(user) });
});
```

9.3 Command Injection e eval()

■ JS | command-injection.js — exec vulnerabile vs spawn sicuro + eval

```
// ■■ PERICOLOSO: exec con input utente ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
const { exec } = require('child_process');

app.get('/ping-vulnerabile', (req, res) => {
  const host = req.query.host;
  // CRITICO: l utente invia host = "8.8.8.8; rm -rf /"
  exec(`ping -c 1 ${host}`, (err, stdout) => res.send(stdout));
});

// ■■ ATTACCO: URL malevolo ■■■■■■■■■■■■■■■■■■■
```

■ JS | redos.js — ReDoS: vulnerabilità, impatto e prevenzione

```
// ReDoS (Regular Expression Denial of Service)
// Regex con backtracking eccessivo su input malevolo

// ■■ VULNERABILE ■■■
```

■ BASH | Sicurezza npm: audit, supply chain, best practice

```bash
# ■■ AUDIT SICUREZZA NPM ■■■
```

■ JS | sicurezza.js — helmet, header CSP, HSTS, variabili d'ambiente sicure

```
// ■■ HELMET.JS: header di sicurezza HTTP ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
const helmet = require('helmet');

app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'nonce-abc123'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"],
      connectSrc: ["'self'"],
      frameSrc: ["'none'"],
      objectSrc: ["'none'"],
    },
  },
  hsts: { maxAge: 31536000, includeSubDomains: true, preload: true },
  noSniff: true,
  xssFilter: true,
  referrerPolicy: { policy: 'strict-origin-when-cross-origin' },
}));

// Rimuovi X-Powered-By (svela che usiamo Express)
app.disable('x-powered-by');

// ■■ VARIABILI D AMBIENTE SICURE ■■■■■■■■■■■■■■■■■■■
```

9.7 Checklist di sicurezza Node.js per produzione

Area	Controllo	Tool / Metodo
Dipendenze	npm audit senza CVE critiche	npm audit; Snyk; Socket.dev
Autenticazione	JWT con scadenza breve + refresh token	jose; passport
Password	bcrypt/argon2 con salt factor >= 12	bcrypt; argon2
Validazione input	Schema validation su OGNI endpoint	zod; joi
Injection	Nessun input utente in exec/eval/query raw	eslint-plugin-security
Header HTTP	CSP, HSTS, X-Frame, nosniff attivi	helmet
Rate limiting	Limite richieste per IP e per endpoint	express-rate-limit; upstash
Logging	Nessun dato sensibile nei log	winston con redact
Secrets	Variabili in env, mai nel codice	dotenv; Vault; AWS Secrets Manager
CORS	Origin whitelist in produzione	cors() con origin specifico
Error handling	Stack trace non esposta in produzione	NODE_ENV=production
Dipendenze aggiornate	Nessun pacchetto con CVE noto	Dependabot; Renovate
Processo	Non eseguire come root	user: node in Dockerfile
Syscall	Permissions model Node.js 20+	--experimental-permission

Fonti: owasp.org/Top10/; nodejs.org/en/docs/guides/security/; cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html; snyk.io/vuln; socket.dev; cve.mitre.org (CVE Database)

10. WEBSOCKET E APPLICAZIONI REAL-TIME

Node.js è particolarmente adatto alle applicazioni real-time grazie al suo modello event-driven e non bloccante. WebSocket, SSE e Socket.IO sono le tecnologie principali per la comunicazione bidirezionale e il push di dati ai client.

10.1 WebSocket nativo (Node.js 22)

■ JS | ws-server.js — WebSocket server con rooms e heartbeat

```
// Node.js 22+ include WebSocket client nativo (no dipendenze)
// Per il SERVER: usa la libreria 'ws'
// npm install ws

const { WebSocketServer } = require('ws');
const http = require('http');

const server = http.createServer();
const wss = new WebSocketServer({ server });

// Mappa per gestire rooms/canali
const rooms = new Map();

wss.on('connection', (ws, req) => {
  const clientId = crypto.randomUUID();
  console.log(`Client connesso: ${clientId}`);

  ws.clientId = clientId;
  ws.isAlive = true;

  // Ping/pong per mantenere connessione attiva
  ws.on('pong', () => { ws.isAlive = true; });

  ws.on('message', (data) => {
    try {
      const msg = JSON.parse(data.toString());

      switch (msg.type) {
        case 'join-room':
          if (!rooms.has(msg.room)) rooms.set(msg.room, new Set());
          rooms.get(msg.room).add(ws);
          ws.room = msg.room;
          ws.send(JSON.stringify({ type: 'joined', room: msg.room }));
          break;

        case 'message':
          // Broadcast alla room
          const room = rooms.get(ws.room);
          if (room) {
            const outgoing = JSON.stringify({
              type: 'message',
              from: clientId,
              text: msg.text,
              ts: Date.now(),
            });
            room.forEach(client => {
              if (client !== ws && client.readyState === ws.OPEN) {
                client.send(outgoing);
              }
            });
          }
          break;
      }
    } catch (err) {
      ws.send(JSON.stringify({ type: 'error', msg: 'Messaggio non valido' }));
    }
  });

  ws.on('close', () => {
    if (ws.room && rooms.has(ws.room)) {
      rooms.get(ws.room).delete(ws);
    }
    console.log(`Client disconnesso: ${clientId}`);
```

```
  });
});

// Heartbeat ogni 30s per rilevare connessioni zombie
const interval = setInterval(() => {
  wss.clients.forEach(ws => {
    if (!ws.isAlive) return ws.terminate();
    ws.isAlive = false;
    ws.ping();
  });
}, 30000);

wss.on('close', () => clearInterval(interval));
server.listen(3001, () => console.log('WebSocket server su :3001'));
```

10.2 Socket.IO: real-time con fallback

■ JS | socket-io-server.js — Socket.IO con auth JWT, rooms, history, typing

```
// npm install socket.io
// server.js — Socket.IO con autenticazione e namespace
const { createServer } = require('http');
const { Server } = require('socket.io');
const express = require('express');

const app = express();
const httpServer = createServer(app);
const io = new Server(httpServer, {
  cors: { origin: process.env.CLIENT_URL, methods: ['GET','POST'] },
  pingTimeout: 60000,
  pingInterval: 25000,
});

// Middleware di autenticazione Socket.IO
io.use(async (socket, next) => {
  try {
    const token = socket.handshake.auth.token;
    const payload = await verificaJWT(token);
    socket.userId = payload.sub;
    socket.userName = payload.name;
    next();
  } catch (err) {
    next(new Error('Autenticazione fallita'));
  }
});

// Namespace /chat
const chat = io.of('/chat');

chat.on('connection', (socket) => {
  console.log(`${socket.userName} connesso`);

  // Unisciti a una room
  socket.on('join-room', async (roomId) => {
    await socket.join(roomId);
    socket.to(roomId).emit('user-joined', {
      userId: socket.userId,
      name: socket.userName,
    });

    // Invia ultimi 50 messaggi al nuovo utente
    const history = await Message.find({ room: roomId })
      .sort({ createdAt: -1 }).limit(50).lean();
    socket.emit('history', history.reverse());
  });

  // Nuovo messaggio
  socket.on('send-message', async ({ roomId, text }) => {
    const msg = await Message.create({
      room: roomId, userId: socket.userId,
      userName: socket.userName, text,
    });

    // Broadcast a tutti nella room (incluso mittente)
    chat.to(roomId).emit('new-message', msg);
  });

  // Indicatore di digitazione
  socket.on('typing', ({ roomId, isTyping }) => {
    socket.to(roomId).emit('user-typing', {
      userId: socket.userId,
      name: socket.userName,
```

```
      isTyping,
    });
  });

  socket.on('disconnect', (reason) => {
    console.log(`${socket.userName} disconnesso: ${reason}`);
  });
});

httpServer.listen(3000);
```

10.3 Server-Sent Events (SSE) per push unidirezionale

■ JS | sse.js — Server-Sent Events: notifiche push server→client

```
// SSE: più semplice di WebSocket per push server→client (es. notifiche, log)
// Usa HTTP standard, nessun upgrade di protocollo

app.get('/sse/events', requireAuth, (req, res) => {
  // Header SSE
  res.setHeader('Content-Type', 'text/event-stream');
  res.setHeader('Cache-Control', 'no-cache');
  res.setHeader('Connection', 'keep-alive');
  res.setHeader('X-Accel-Buffering', 'no');  // Nginx: disabilita buffering
  res.flushHeaders();

  // Invia heartbeat ogni 15s (evita timeout proxy)
  const heartbeat = setInterval(() => {
    res.write(':heartbeat

');
  }, 15000);

  // Invia evento
  const sendEvent = (eventType, data) => {
    res.write(`event: ${eventType}
`);
    res.write(`data: ${JSON.stringify(data)}

`);
  };

  sendEvent('connected', { userId: req.user.sub, ts: Date.now() });

  // Registra listener per questo utente
  const userId = req.user.sub;
  eventBus.on(`notify:${userId}`, sendEvent);

  // Cleanup alla disconnessione
  req.on('close', () => {
    clearInterval(heartbeat);
    eventBus.off(`notify:${userId}`, sendEvent);
    res.end();
  });
});

// Trigger notifica da qualsiasi parte del codice
eventBus.emit(`notify:${userId}`, 'order-update', { orderId: 123, status: 'shipped' });
```

Fonti: socket.io/docs/v4/; github.com/websockets/ws; developer.mozilla.org/docs/Web/API/Server-sent_events; nodejs.org/api/http.html#event-upgrade (WebSocket nativo Node.js 22)

11. INTEGRAZIONE NODE.JS CON PYTHON

Node.js e Python sono spesso usati insieme nello stesso stack: Node.js per le API web e i microservizi, Python per machine learning, data science, scripting e automazione. Esistono più pattern di integrazione, ognuno con trade-off diversi.

11.1 Pattern 1: child_process — esecuzione script Python

■ JS | python-child.js — esecuzione script Python da Node.js con spawn()

```
// Il metodo più semplice: Node.js lancia script Python come processo figlio
// Ideale per script batch, ML inference, conversioni dati

const { spawn } = require('child_process');
const path = require('path');

function eseguiPython(scriptPath, args = [], inputData = null) {
  return new Promise((resolve, reject) => {
    const py = spawn('python3', [scriptPath, ...args], {
      cwd: path.dirname(scriptPath),
      env: { ...process.env, PYTHONPATH: '/usr/local/lib/python3' },
    });

    let stdout = '';
    let stderr = '';

    py.stdout.on('data', chunk => { stdout += chunk.toString(); });
    py.stderr.on('data', chunk => { stderr += chunk.toString(); });

    if (inputData !== null) {
      // Invia dati allo stdin di Python
      py.stdin.write(JSON.stringify(inputData));
      py.stdin.end();
    }

    py.on('close', code => {
      if (code !== 0) {
        reject(new Error(`Python exited ${code}: ${stderr}`));
        return;
      }
      try {
        resolve(JSON.parse(stdout));
      } catch {
        resolve(stdout.trim());
      }
    });

    // Timeout di sicurezza
    setTimeout(() => {
      py.kill('SIGTERM');
      reject(new Error('Python script timeout'));
    }, 30000);
  });
}

// Uso: analisi sentiment con Python/transformers
app.post('/api/sentiment', async (req, res, next) => {
  try {
    const { text } = req.body;
    const result = await eseguiPython(
      './python/sentiment.py',
      [],
      { text }
    );
    res.json(result);
  } catch (err) {
    next(err);
  }
});
```

■ PYTHON | python/sentiment.py — analisi sentiment con transformers HuggingFace

```python
# python/sentiment.py — script Python richiamato da Node.js
import sys
import json
from transformers import pipeline

def analizza_sentiment(testo):
    classificatore = pipeline(
        'sentiment-analysis',
        model='nlptown/bert-base-multilingual-uncased-sentiment'
    )
    risultato = classificatore(testo[:512])[0]
    return {
        'label': risultato['label'],
        'score': round(risultato['score'], 4),
        'testo': testo[:100]
    }

if __name__ == '__main__':
    # Leggi input da stdin (inviato da Node.js)
    dati = json.load(sys.stdin)
    output = analizza_sentiment(dati['text'])
    # Scrivi JSON su stdout (letto da Node.js)
    print(json.dumps(output, ensure_ascii=False))
```

11.2 Pattern 2: HTTP microservizi (Flask/FastAPI)

■ PYTHON | ml_service.py — FastAPI microservice ML richiamato da Node.js

```python
# python/ml_service.py — FastAPI microservice
# pip install fastapi uvicorn transformers torch
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import torch
from transformers import pipeline

app = FastAPI(title='ML Microservice', version='1.0')

# Carica modello una volta all avvio
sentiment_pipe = pipeline('sentiment-analysis', device=0 if torch.cuda.is_available()
  else -1)
ner_pipe = pipeline('ner', grouped_entities=True)

class AnalisiRequest(BaseModel):
    testo: str
    task: str = 'sentiment'  # sentiment | ner | summarize

class AnalisiResponse(BaseModel):
    risultato: dict | list
    modello: str
    tempo_ms: float

@app.post('/analizza', response_model=AnalisiResponse)
async def analizza(req: AnalisiRequest):
    import time
    start = time.time()

    if req.task == 'sentiment':
        r = sentiment_pipe(req.testo[:512])[0]
        risultato = {'label': r['label'], 'score': round(r['score'], 4)}
        modello = 'distilbert-sentiment'
    elif req.task == 'ner':
        risultato = ner_pipe(req.testo[:512])
        modello = 'bert-ner'
    else:
        raise HTTPException(400, f'Task non supportato: {req.task}')

    return AnalisiResponse(
        risultato=risultato,
        modello=modello,
        tempo_ms=round((time.time() - start) * 1000, 2)
    )

# Avvio: uvicorn ml_service:app --host 0.0.0.0 --port 8000
```

■ JS | ml-client.js — Node.js chiama il microservizio FastAPI Python

```
// Node.js chiama il microservizio Python via HTTP
// npm install axios  (o usa fetch nativo Node.js 22)

class MLServiceClient {
  constructor(baseUrl = 'http://localhost:8000') {
    this.baseUrl = baseUrl;
    this.timeout = 10000;
  }

  async analizza(testo, task = 'sentiment') {
    const controller = new AbortController();
    const timeout = setTimeout(() => controller.abort(), this.timeout);

    try {
      const res = await fetch(`${this.baseUrl}/analizza`, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ testo, task }),
        signal: controller.signal,
      });

      if (!res.ok) {
        const err = await res.json().catch(() => ({}));
        throw new Error(err.detail || `HTTP ${res.status}`);
      }

      return await res.json();
    } finally {
      clearTimeout(timeout);
    }
  }
}

const mlClient = new MLServiceClient(process.env.ML_SERVICE_URL);

app.post('/api/analizza', async (req, res, next) => {
  try {
    const { testo, task } = req.body;
    const risultato = await mlClient.analizza(testo, task);
    res.json(risultato);
  } catch (err) {
    next(err);
  }
});
```

11.3 Pattern 3: ZeroMQ per comunicazione ad alte performance

■ PYTHON | python/worker.py — ZeroMQ worker Python per calcoli numerici

```
# Comunicazione inter-process ad alte performance senza HTTP overhead
# pip install pyzmq
# npm install zeromq

# python/worker.py — worker Python che processa task
import zmq
import json
import numpy as np

context = zmq.Context()
socket = context.socket(zmq.REP)  # Reply socket
socket.bind('tcp://*:5555')

print('Python worker in ascolto su :5555')

while True:
    # Ricevi task da Node.js
    raw = socket.recv()
    task = json.loads(raw)

    # Esegui elaborazione (es. calcolo matrice)
    if task['tipo'] == 'matrix_multiply':
        A = np.array(task['A'])
        B = np.array(task['B'])
        risultato = (A @ B).tolist()
        socket.send(json.dumps({'ok': True, 'risultato': risultato}).encode())
    else:
        socket.send(json.dumps({'ok': False, 'error': 'Unknown task'}).encode())
```

11.4 Confronto pattern di integrazione

Pattern	Latenza	Complessità	Scenario ideale
child_process spawn()	Alta (startup + IPC)	Bassa	Script batch; ML inference occasionale; script una-tantum
HTTP microservice (FastAPI)	Media (rete locale)	Media	Servizi ML persistenti; team separati; scalabilità indipendente
ZeroMQ (zeromq)	Bassa (IPC/TCP diretto)	Media	Alta frequenza messaggi; calcoli numerici; pipeline dati
Redis Queue (BullMQ + Celery)	Media (Redis round-trip)	Alta	Task asincroni; retry automatico; monitoraggio job
gRPC	Molto bassa (binary proto)	Alta	Microservizi production; contratti tipizzati; streaming bidirezionale
Shared filesystem	Molto bassa (su stessa macchina)	Bassa	File grandi; batch processing; prototyping rapido

Fonti: fastapi.tiangolo.com; flask.palletsprojects.com; zeromq.org; grpc.io; github.com/zeromq/zeromq.js; huggingface.co/docs/transformers (HuggingFace Transformers)

12. AI E LLM IN NODE.JS

Node.js è diventato una piattaforma di prima scelta per integrare modelli linguistici di grandi dimensioni (LLM). La sua natura asincrona è perfetta per gestire le risposte in streaming dei modelli AI, e l'ecosistema npm offre SDK ufficiali per tutti i principali provider.

12.1 Anthropic SDK: integrazione con Claude

■ BASH | Installazione Anthropic SDK

```
# npm install @anthropic-ai/sdk
# Variabile d ambiente richiesta: ANTHROPIC_API_KEY
```

■ JS | anthropic-examples.js — Claude: streaming, tool use, agente

```
// anthropic-examples.js
const Anthropic = require('@anthropic-ai/sdk');

const client = new Anthropic({
  apiKey: process.env.ANTHROPIC_API_KEY,
});

// ■■ RISPOSTA COMPLETA ■■■
```

```
      },
    },
  ];

  const messages = [{ role: 'user', content: userQuery }];

  while (true) {
    const response = await client.messages.create({
      model: 'claude-opus-4-6',
      max_tokens: 4096,
      tools,
      messages,
    });

    if (response.stop_reason === 'end_turn') {
      return response.content.find(b => b.type === 'text')?.text;
    }

    if (response.stop_reason === 'tool_use') {
      messages.push({ role: 'assistant', content: response.content });

      const toolResults = [];
      for (const block of response.content) {
        if (block.type !== 'tool_use') continue;

        let result;
        if (block.name === 'get_weather') {
          result = await fetchWeatherAPI(block.input.city);
        } else if (block.name === 'search_database') {
          result = await db.search(block.input.query, block.input.limit);
        }

        toolResults.push({
          type: 'tool_result',
          tool_use_id: block.id,
          content: JSON.stringify(result),
        });
      }
      messages.push({ role: 'user', content: toolResults });
    }
  }
}

// ■■ EXPRESS ENDPOINT ■■■
```

■ JS | openai-sdk.js — OpenAI: streaming, embeddings, vision, JSON output

```
// npm install openai
const OpenAI = require('openai');

const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

// Chat completion con streaming
async function* streamChat(messages) {
  const stream = await openai.chat.completions.create({
    model: 'gpt-4o',
    messages,
    stream: true,
    temperature: 0.7,
    max_tokens: 2048,
  });

  for await (const chunk of stream) {
    const delta = chunk.choices[0]?.delta?.content;
    if (delta) yield delta;
  }
}

// Embeddings per vector search
async function getEmbedding(text) {
  const response = await openai.embeddings.create({
    model: 'text-embedding-3-small',
    input: text,
    dimensions: 1536,
  });
  return response.data[0].embedding;
}

// Vision: analisi immagini
async function analizzaImmagine(imageUrl, domanda) {
  const response = await openai.chat.completions.create({
    model: 'gpt-4o',
    messages: [{
      role: 'user',
      content: [
        { type: 'image_url', image_url: { url: imageUrl, detail: 'high' } },
        { type: 'text', text: domanda },
      ],
    }],
    max_tokens: 500,
  });
  return response.choices[0].message.content;
}

// Structured output (JSON con schema garantito)
const risposta = await openai.chat.completions.create({
  model: 'gpt-4o',
  messages: [{ role: 'user', content: 'Elenca 3 capitali europee in JSON' }],
  response_format: {
    type: 'json_schema',
    json_schema: {
      name: 'capitali',
      schema: {
        type: 'object',
        properties: {
          capitali: {
            type: 'array',
            items: { type: 'string' }
          }
        }
```

```
      }
    }
  }
});
const dati = JSON.parse(risposta.choices[0].message.content);
```

12.3 Vercel AI SDK: astrazione multi-provider

■ BASH | Installazione Vercel AI SDK

```
# npm install ai @ai-sdk/anthropic @ai-sdk/openai @ai-sdk/google
# Vercel AI SDK: interfaccia unificata per Anthropic, OpenAI, Google, Mistral
```

■ JS | ai-sdk.js — Vercel AI SDK: multi-provider, streaming, structured output, agent

```
// ai-sdk-examples.js — Vercel AI SDK
const { generateText, streamText, generateObject } = require('ai');
const { anthropic } = require('@ai-sdk/anthropic');
const { openai } = require('@ai-sdk/openai');
const { z } = require('zod');

// Stessa API per diversi modelli
async function generaTesto(prompt, provider = 'anthropic') {
  const model = provider === 'anthropic'
    ? anthropic('claude-opus-4-6')
    : openai('gpt-4o');

  const { text } = await generateText({ model, prompt });
  return text;
}

// Streaming con Vercel AI SDK (ottimo per Next.js / Express)
app.post('/api/stream', async (req, res) => {
  const { prompt } = req.body;

  const result = streamText({
    model: anthropic('claude-opus-4-6'),
    prompt,
    system: 'Sei un esperto di Node.js.',
  });

  // Pipe diretto alla risposta HTTP
  result.pipeTextStreamToResponse(res);
});

// Structured output tipizzato con Zod
async function estraiDati(testo) {
  const { object } = await generateObject({
    model: openai('gpt-4o'),
    schema: z.object({
      nome: z.string(),
      eta: z.number(),
      competenze: z.array(z.string()),
      esperienza_anni: z.number(),
    }),
    prompt: `Estrai le informazioni dal CV: ${testo}`,
  });
  return object;  // tipo garantito da Zod
}

// Agente con tool use (Vercel AI SDK)
const { streamText: streamAgent } = require('ai');

async function agente(query) {
  const result = await streamText({
    model: anthropic('claude-opus-4-6'),
    tools: {
      cercaDB: {
        description: 'Cerca nel database',
        parameters: z.object({ q: z.string(), limit: z.number().default(10) }),
        execute: async ({ q, limit }) => db.search(q, limit),
      },
      inviaEmail: {
        description: 'Invia email a un utente',
        parameters: z.object({ to: z.string().email(), subject: z.string() }),
        execute: async ({ to, subject }) => emailService.send(to, subject),
      },
    },
```

```
    maxSteps: 5,  // max iterazioni tool use
    prompt: query,
  });
  return result;
}
```

Fonti: docs.anthropic.com/sdk/node; platform.openai.com/docs/libraries/node-js-library; sdk.vercel.ai/docs; github.com/anthropics/anthropic-sdk-python (confronto Python SDK)

13. MODEL CONTEXT PROTOCOL (MCP)

Il **Model Context Protocol (MCP)** è uno standard aperto sviluppato da Anthropic (novembre 2024) che definisce come i modelli AI possono comunicare con strumenti e fonti dati esterne. Node.js è il runtime di riferimento per costruire server MCP, grazie all'SDK ufficiale @modelcontextprotocol/sdk.

13.1 Cos'è MCP e perché è importante

Prima di MCP, ogni applicazione AI doveva implementare integrazioni custom per ogni fonte dati o tool: un'integrazione per GitHub, una per il database, una per i file locali, ecc. MCP standardizza questa comunicazione con un protocollo client-server basato su JSON-RPC 2.0, permettendo ai modelli AI di accedere a **risorse** (dati) e **tool** (azioni) in modo uniforme e riutilizzabile.

Concetto MCP	Descrizione	Esempio
Server MCP	Processo che espone risorse e tool via JSON-RPC	Server che espone file system, database, API GitHub
Client MCP	L applicazione host che si connette al server	Claude Desktop, IDE, applicazione custom
Resource	Dati leggibili dal modello (URI-based)	file:///, database://tabella, git://repo/branch
Tool	Azione eseguibile dal modello (con parametri)	crea_file, esegui_query, crea_issue_github
Prompt	Template riutilizzabili con placeholder	Template per code review, analisi dati, ecc.
Transport	Meccanismo di comunicazione	stdio (locale), HTTP SSE (remoto), WebSocket

13.2 Costruire un server MCP in Node.js

■ BASH | Installazione MCP SDK per Node.js

```
# npm install @modelcontextprotocol/sdk zod
```

■ JS | mcp-server.js — server MCP completo con resources, tools e sicurezza

```
// mcp-server.js — Server MCP completo in Node.js
const { Server } = require('@modelcontextprotocol/sdk/server/index.js');
const { StdioServerTransport } = require(
  '@modelcontextprotocol/sdk/server/stdio.js'
);
const {
  CallToolRequestSchema,
  ListResourcesRequestSchema,
  ReadResourceRequestSchema,
  ListToolsRequestSchema,
} = require('@modelcontextprotocol/sdk/types.js');
const { z } = require('zod');
const fs = require('fs').promises;
const path = require('path');

// Crea il server MCP
const server = new Server(
  { name: 'nodejs-filesystem-server', version: '1.0.0' },
  { capabilities: { resources: {}, tools: {} } }
);

// ■■ RISORSE: esponi file del filesystem ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
server.setRequestHandler(ListResourcesRequestSchema, async () => {
  const cwd = process.cwd();
  const entries = await fs.readdir(cwd, { withFileTypes: true });

  return {
    resources: entries
      .filter(e => e.isFile())
      .map(e => ({
        uri: `file:///${path.join(cwd, e.name)}`,
        name: e.name,
        mimeType: getMimeType(e.name),
        description: `File: ${e.name}`,
      })),
  };
});

server.setRequestHandler(ReadResourceRequestSchema, async (req) => {
  const filePath = new URL(req.params.uri).pathname;

  // Sicurezza: solo file nella directory corrente
  const cwd = process.cwd();
  const absPath = path.resolve(filePath);
  if (!absPath.startsWith(cwd)) {
    throw new Error('Accesso non autorizzato fuori dalla directory');
  }

  const contenuto = await fs.readFile(absPath, 'utf8');
  return {
    contents: [{
      uri: req.params.uri,
      mimeType: getMimeType(filePath),
      text: contenuto,
    }],
  };
});

// ■■ TOOL: azioni che il modello puo' eseguire ■■■■■■■■■■■■■■■■■■■■■■
server.setRequestHandler(ListToolsRequestSchema, async () => ({
  tools: [
    {
      name: 'crea_file',
```

```
      description: 'Crea o sovrascrive un file con il contenuto specificato',
      inputSchema: {
        type: 'object',
        properties: {
          percorso: { type: 'string', description: 'Percorso relativo del file' },
          contenuto: { type: 'string', description: 'Contenuto del file' },
        },
        required: ['percorso', 'contenuto'],
      },
    },
    {
      name: 'esegui_comando',
      description: 'Esegui un comando shell nella directory corrente',
      inputSchema: {
        type: 'object',
        properties: {
          comando: { type: 'string', description: 'Comando da eseguire' },
        },
        required: ['comando'],
      },
    },
    {
      name: 'cerca_file',
      description: 'Cerca file per nome o estensione nella directory',
      inputSchema: {
        type: 'object',
        properties: {
          pattern: { type: 'string', description: 'Pattern glob (es. *.js)' },
          recursive: { type: 'boolean', default: true },
        },
        required: ['pattern'],
      },
    },
  ],
}));

server.setRequestHandler(CallToolRequestSchema, async (req) => {
  const { name, arguments: args } = req.params;

  if (name === 'crea_file') {
    const percorso = path.resolve(args.percorso);
    if (!percorso.startsWith(process.cwd())) {
      throw new Error('Percorso non consentito fuori dalla directory');
    }
    await fs.mkdir(path.dirname(percorso), { recursive: true });
    await fs.writeFile(percorso, args.contenuto, 'utf8');
    return {
      content: [{ type: 'text', text: `File creato: ${args.percorso}` }],
    };
  }

  if (name === 'esegui_comando') {
    const { execSync } = require('child_process');
    // Blocca comandi pericolosi
    const BLOCKLIST = ['rm -rf', 'format', 'del /f', 'shutdown'];
    if (BLOCKLIST.some(b => args.comando.includes(b))) {
      throw new Error('Comando non consentito per sicurezza');
    }
    try {
      const output = execSync(args.comando, {
        cwd: process.cwd(),
        timeout: 10000,
        maxBuffer: 1024 * 1024,
        encoding: 'utf8',
```

```
      });
      return { content: [{ type: 'text', text: output }] };
    } catch (err) {
      return {
        content: [{ type: 'text', text: `Errore: ${err.message}` }],
        isError: true,
      };
    }
  }

  throw new Error(`Tool non trovato: ${name}`);
});

// ■■ HELPER ■■■
```

13.4 Server MCP HTTP per deployment remoto

■ JS | mcp-http-server.js — server MCP via HTTP+SSE per deployment remoto

```
// mcp-http-server.js — MCP via HTTP+SSE per deployment remoto
const express = require('express');
const { Server } = require('@modelcontextprotocol/sdk/server/index.js');
const { SSEServerTransport } = require(
  '@modelcontextprotocol/sdk/server/sse.js'
);

const app = express();
const transports = new Map();

app.get('/sse', async (req, res) => {
  const transport = new SSEServerTransport('/messages', res);
  const sessionId = transport.sessionId;
  transports.set(sessionId, transport);

  const server = createMCPServer();  // factory function
  await server.connect(transport);

  res.on('close', () => {
    transports.delete(sessionId);
  });
});

app.post('/messages', express.json(), async (req, res) => {
  const sessionId = req.query.sessionId;
  const transport = transports.get(sessionId);
  if (!transport) {
    return res.status(404).json({ error: 'Session not found' });
  }
  await transport.handlePostMessage(req, res);
});

app.listen(8080, () => {
  console.log('MCP HTTP Server su http://localhost:8080/sse');
});
```

Fonti: modelcontextprotocol.io; github.com/modelcontextprotocol/typescript-sdk; anthropic.com/news/model-context-protocol (annuncio, novembre 2024); npmjs.com/package/@modelcontextprotocol/sdk

14. TESTING IN NODE.JS

Il testing è una pratica fondamentale per garantire la qualità del codice Node.js. Questa sezione copre unit test, integration test e test E2E con gli strumenti più usati nell'ecosistema.

14.1 Jest: setup e test unitari

■ BASH | Setup Jest

```bash
# npm install --save-dev jest @types/jest
# Aggiungi a package.json: "test": "jest --coverage"
# Per ESM: "test": "node --experimental-vm-modules node_modules/.bin/jest"
```

■ JS | mathUtils.test.js — test unitari con Jest: expect, mock, async

```
// src/utils/mathUtils.js
function somma(a, b) { return a + b; }
function dividi(a, b) {
  if (b === 0) throw new Error('Divisione per zero');
  return a / b;
}
async function fetchDato(id) {
  const res = await fetch(`https://api.example.com/items/${id}`);
  if (!res.ok) throw new Error(`HTTP ${res.status}`);
  return res.json();
}
module.exports = { somma, dividi, fetchDato };

// tests/mathUtils.test.js
const { somma, dividi, fetchDato } = require('../src/utils/mathUtils');

describe('somma()', () => {
  test('somma due numeri positivi', () => {
    expect(somma(2, 3)).toBe(5);
  });
  test('somma con numero negativo', () => {
    expect(somma(-1, 4)).toBe(3);
  });
  test('somma con zero', () => {
    expect(somma(0, 0)).toBe(0);
  });
});

describe('dividi()', () => {
  test('divisione corretta', () => {
    expect(dividi(10, 2)).toBe(5);
  });
  test('lancia errore per divisione per zero', () => {
    expect(() => dividi(5, 0)).toThrow('Divisione per zero');
  });
});

describe('fetchDato() con mock', () => {
  // Mock di fetch globale
  beforeEach(() => {
    global.fetch = jest.fn();
  });

  afterEach(() => {
    jest.restoreAllMocks();
  });

  test('ritorna dati quando fetch ha successo', async () => {
    const datoMock = { id: 1, nome: 'Test' };
    fetch.mockResolvedValue({
      ok: true,
      json: () => Promise.resolve(datoMock),
    });

    const risultato = await fetchDato(1);
    expect(risultato).toEqual(datoMock);
    expect(fetch).toHaveBeenCalledWith('https://api.example.com/items/1');
  });

  test('lancia errore se fetch fallisce', async () => {
    fetch.mockResolvedValue({ ok: false, status: 404 });
    await expect(fetchDato(99)).rejects.toThrow('HTTP 404');
  });
```

```
});
```

14.2 Integration Test con Supertest

■ JS | users.integration.test.js — test API REST con Supertest e MongoDB

```
// npm install --save-dev supertest
// tests/api/users.integration.test.js
const request = require('supertest');
const app = require('../../src/index');
const mongoose = require('mongoose');
const User = require('../../src/models/User');

describe('API /api/v1/users', () => {
  let authToken;
  let testUserId;

  // Setup: connetti al DB di test prima di tutti i test
  beforeAll(async () => {
    await mongoose.connect(process.env.MONGODB_TEST_URI);
    await User.deleteMany({});  // pulizia DB test

    // Crea utente admin per i test auth
    const admin = await User.create({
      nome: 'Admin Test', email: 'admin@test.com',
      password: 'Admin@123', role: 'admin',
    });
    const loginRes = await request(app)
      .post('/api/v1/auth/login')
      .send({ email: 'admin@test.com', password: 'Admin@123' });
    authToken = loginRes.body.token;
  });

  afterAll(async () => {
    await User.deleteMany({});
    await mongoose.connection.close();
  });

  describe('POST /api/v1/users', () => {
    test('crea utente con dati validi', async () => {
      const res = await request(app)
        .post('/api/v1/users')
        .set('Authorization', `Bearer ${authToken}`)
        .send({ nome: 'Mario', email: 'mario@test.com', password: 'Mario@123' });

      expect(res.status).toBe(201);
      expect(res.body.data).toMatchObject({
        nome: 'Mario',
        email: 'mario@test.com',
      });
      expect(res.body.data.password).toBeUndefined();  // no password in response
      testUserId = res.body.data._id;
    });

    test('rifiuta email duplicata con 409', async () => {
      const res = await request(app)
        .post('/api/v1/users')
        .set('Authorization', `Bearer ${authToken}`)
        .send({ nome: 'Dup', email: 'mario@test.com', password: 'Dup@123' });
      expect(res.status).toBe(409);
    });

    test('valida dati mancanti con 400', async () => {
      const res = await request(app)
        .post('/api/v1/users')
        .set('Authorization', `Bearer ${authToken}`)
        .send({ email: 'mancante@test.com' });
      expect(res.status).toBe(400);
      expect(res.body.details).toBeDefined();
```

```
    });
  });

  describe('GET /api/v1/users', () => {
    test('restituisce lista paginata', async () => {
      const res = await request(app)
        .get('/api/v1/users?page=1&limit=10')
        .set('Authorization', `Bearer ${authToken}`);

      expect(res.status).toBe(200);
      expect(res.body.data).toBeInstanceOf(Array);
      expect(res.body.pagination).toMatchObject({
        page: 1, limit: 10,
      });
    });
  });
});
```

Fonti: jestjs.io; vitest.dev; github.com/ladjs/supertest; testing-library.com; playwright.dev; nodejs.org/api/test.html (test runner nativo Node.js 18+)

15. PERFORMANCE, MONITORING E DEPLOYMENT

Portare un'applicazione Node.js in produzione richiede attenzione a performance, monitoring, scalabilità e strategie di deployment. Questa sezione copre le tecniche più importanti.

15.1 Profiling e ottimizzazione delle performance

■ JS | performance.js — profiling, benchmark, caching in-memory e Redis

```
// ■■ PROFILING CON BUILT-IN NODE.JS ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
// Avvia con profiler V8
node --prof app.js
node --prof-process isolate-*.log > profile.txt

// CPU profiling con clinic.js (npm install -g clinic)
clinic doctor -- node app.js      # diagnostica generale
clinic flame -- node app.js       # flamegraph CPU
clinic bubbleprof -- node app.js  # async profiling

// ■■ BENCHMARK CON autocannon ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
// npm install -g autocannon
autocannon -c 100 -d 30 http://localhost:3000/api/users
// -c 100 = 100 connessioni concorrenti
// -d 30  = 30 secondi di test

// ■■ CACHING PER PERFORMANCE ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
const NodeCache = require('node-cache');
const cache = new NodeCache({ stdTTL: 60, checkperiod: 120 });

app.get('/api/prodotti', async (req, res, next) => {
  const cacheKey = `prodotti:${JSON.stringify(req.query)}`;
  const cached = cache.get(cacheKey);
  if (cached) {
    res.setHeader('X-Cache', 'HIT');
    return res.json(cached);
  }

  const prodotti = await Product.find(req.query).lean();
  cache.set(cacheKey, prodotti);
  res.setHeader('X-Cache', 'MISS');
  res.json(prodotti);
});

// Cache con Redis (distribuita, per cluster)
const { createClient } = require('redis');
const redis = createClient({ url: process.env.REDIS_URL });
await redis.connect();

async function getCached(key, fetcher, ttl = 60) {
  const cached = await redis.get(key);
  if (cached) return JSON.parse(cached);

  const data = await fetcher();
  await redis.setEx(key, ttl, JSON.stringify(data));
  return data;
}
```

15.2 Cluster mode: tutti i core CPU

■ JS | cluster.js — Node.js cluster per utilizzare tutti i core CPU

```
// cluster.js — utilizza tutti i core CPU disponibili
const cluster = require('cluster');
const { cpus } = require('os');
const process = require('process');

const NUM_WORKERS = cpus().length;

if (cluster.isPrimary) {
  console.log(`Master ${process.pid} avviato`);
  console.log(`Creo ${NUM_WORKERS} worker...`);

  // Crea un worker per ogni CPU
  for (let i = 0; i < NUM_WORKERS; i++) {
    cluster.fork();
  }

  // Riavvio automatico dei worker morti
  cluster.on('exit', (worker, code, signal) => {
    console.warn(`Worker ${worker.process.pid} morto (${signal || code})`);
    if (code !== 0) {
      console.log('Riavvio worker...');
      cluster.fork();
    }
  });

  cluster.on('online', (worker) => {
    console.log(`Worker ${worker.process.pid} online`);
  });

} else {
  // Ogni worker carica l applicazione Express
  require('./src/index');
  console.log(`Worker ${process.pid} in ascolto`);
}

// Alternativa moderna: usa PM2 (molto piu' robusto)
// pm2 start app.js -i max   (i = numero worker = max CPU)
```

15.3 Docker: containerizzazione

■ BASH | Dockerfile — multi-stage build ottimizzato per Node.js in produzione

```
# Dockerfile ottimizzato per Node.js in produzione
# Multi-stage build: build stage + produzione stage

# ■■ STAGE 1: dipendenze ■■■
```

■ YAML | docker-compose.yml — stack completo: Node.js, MongoDB, Redis, Nginx

```yaml
# docker-compose.yml — stack completo per sviluppo
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=development
      - MONGODB_URI=mongodb://mongo:27017/mydb
      - REDIS_URL=redis://redis:6379
    volumes:
      - ./src:/app/src   # hot reload in sviluppo
    depends_on:
      mongo:
        condition: service_healthy
      redis:
        condition: service_healthy
    restart: unless-stopped

  mongo:
    image: mongo:7.0
    volumes:
      - mongo_data:/data/db
    healthcheck:
      test: ["CMD", "mongosh", "--eval", "db.runCommand({ping:1})"]
      interval: 10s
      timeout: 5s
      retries: 5

  redis:
    image: redis:7.2-alpine
    command: redis-server --appendonly yes
    volumes:
      - redis_data:/data
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 10s

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/conf.d/default.conf
      - ./nginx/certs:/etc/nginx/certs
    depends_on:
      - app

volumes:
  mongo_data:
  redis_data:
```

15.4 PM2: process manager in produzione

■ BASH | PM2: process manager per Node.js in produzione

```bash
# npm install -g pm2

# ecosystem.config.js — configurazione PM2
module.exports = {
  apps: [{
    name: 'mia-api',
    script: './src/index.js',
    instances: 'max',        // un processo per CPU
    exec_mode: 'cluster',    // mode cluster
    max_memory_restart: '500M',
    env: { NODE_ENV: 'development', PORT: 3000 },
    env_production: { NODE_ENV: 'production', PORT: 3000 },
    error_file: './logs/pm2-error.log',
    out_file: './logs/pm2-out.log',
    merge_logs: true,
    watch: false,
    autorestart: true,
    restart_delay: 1000,
    max_restarts: 10,
  }]
};

# Comandi PM2:
pm2 start ecosystem.config.js --env production
pm2 list                   # stato processi
pm2 logs mia-api           # log in real-time
pm2 monit                  # dashboard CPU/RAM
pm2 reload mia-api         # zero-downtime restart
pm2 save && pm2 startup    # avvio automatico al boot
```

Fonti: nodejs.org/api/cluster.html; pm2.keymetrics.io; docs.docker.com; hub.docker.com/_/node; clinicjs.org; npmjs.com/package/autocannon; redis.io/docs/connect/clients/nodejs

16. CASI D'USO REALI E ARCHITETTURE

Node.js alimenta alcune delle infrastrutture digitali più trafficate al mondo. Questa sezione analizza casi reali documentati con architetture, risultati misurati e pattern applicabili.

16.1 Netflix: streaming a 250 milioni di utenti

Netflix ha migrato il proprio stack di interfaccia utente da un'applicazione Java monolitica a microservizi Node.js nel 2015. Il progetto, documentato nel blog di Netflix Tech, ha prodotto risultati misurabili:

Metrica	Prima (Java)	Dopo (Node.js)	Miglioramento
Tempo avvio server	~40 secondi	<1 secondo	40x più veloce
Richieste/secondo	baseline	+100%	2x throughput
Linee di codice	~700.000	~200.000	-71% codice
Team size per feature	2-3 team	1 team	Produttività maggiore
Deploy frequency	Settimanale	Giornaliero	+7x frequenza

Architettura Node.js di Netflix: edge layer con Node.js gestisce il rendering server-side dell'interfaccia (Falcor + React); ogni team possiede il proprio microservizio Node.js che chiama i servizi Java/Go del backend.

16.2 PayPal: 2x performance con la metà delle righe di codice

PayPal ha pubblicato nel 2013 un case study documentato sulla migrazione della pagina account overview da Java a Node.js, con risultati sorprendenti:

Metrica	Java Spring	Node.js	Delta
File JS (frontend)	Separati per responsabilità	Condivisi frontend/backend	Un solo linguaggio
Linee di codice	~22.000	~14.000	-33%
File sorgente	~109	~26	-76%
Request/second	baseline	+35%	1.35x
Latenza media	baseline	-200ms	Piu' veloce
Team di sviluppo	5 Java + 5 JS	2 full-stack	Meno risorse

16.3 Uber: real-time matching su 100M+ utenti

Uber usa Node.js per il layer di comunicazione real-time tra driver e passeggeri. Il requisito chiave: bassa latenza per la geolocalizzazione in tempo reale e gestione di picchi improvvisi. Node.js gestisce fino a **1 milione di connessioni WebSocket contemporanee** su un singolo cluster, grazie all'event loop non bloccante.

■ JS | uber-realtime.js — geolocalizzazione real-time con Redis Geo + Kafka

```
// Architettura semplificata del real-time matching di Uber
// Pattern: CQRS + Event Sourcing con Node.js + Kafka

// Servizio geolocalizzazione (Node.js + Socket.IO)
const io = require('socket.io')(httpServer, { transport: ['websocket'] });
const redis = require('ioredis').createClient();
const kafka = require('kafkajs').Kafka;

io.on('connection', async (socket) => {
  const { driverId, lat, lng } = socket.handshake.auth;

  // Aggiorna posizione in Redis (geospatial)
  await redis.geoadd('driver:positions', lng, lat, driverId);

  socket.on('location-update', async ({ lat, lng }) => {
    // Aggiorna posizione in Redis
    await redis.geoadd('driver:positions', lng, lat, driverId);

    // Pubblica su Kafka per analytics e matching
    await producer.send({
      topic: 'driver-locations',
      messages: [{ key: driverId, value: JSON.stringify({ driverId, lat, lng, ts:
        Date.now() }) }]
    });
  });

  socket.on('disconnect', async () => {
    await redis.zrem('driver:positions', driverId);
  });
});

// Funzione di matching: trova driver entro 2km
async function trovaNearbyDrivers(passengerLat, passengerLng, radiusKm = 2) {
  const drivers = await redis.georadius(
    'driver:positions', passengerLng, passengerLat,
    radiusKm, 'km', 'WITHCOORD', 'WITHDIST', 'COUNT', 10, 'ASC'
  );
  return drivers;
}
```

16.4 LinkedIn: 27x meno server con Node.js

LinkedIn ha migrato il suo stack mobile API da Ruby on Rails a Node.js nel 2011-2012. I risultati pubblicati nel LinkedIn Engineering Blog: da **30 server Rails** a **3 server Node.js** per gestire lo stesso traffico — un'efficienza 10x superiore. La latenza è scesa del 20%, con picchi di **20.000 connessioni simultanee** per server.

16.5 Architettura microservizi con Node.js: pattern

Pattern	Descrizione	Tool Node.js	Quando usarlo
API Gateway	Single entry point per tutti i microservizi	Express + http-proxy-middleware	Sempre in produzione
BFF (Backend for Frontend)	API layer su misura per ogni tipo di client (mobile, web, TV)	Node.js + Apollo Server	Client con bisogni diversi
Event-Driven	Servizi comunicano tramite eventi asincroni	Kafka (kafkajs); RabbitMQ (amqplib); Redis Pub/Sub	Disaccoppiamento forte; alta scalabilità
CQRS + Event Sourcing	Separazione comandi (scrittura) da query (lettura)	EventStore; PostgreSQL + Kafka	Audit trail; replay eventi; scalabilità lettura
Saga Pattern	Gestione transazioni distribuite tra microservizi	BullMQ + custom orchestrator	Transazioni multi-servizio
Circuit Breaker	Fallback automatico quando un servizio è down	opossum (npm)	Resilienza in produzione

Fonti: netflixtechblog.com/making-netflix-com-faster-f95d15f2e972; medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f; uber.com/blog/node-js-uber/; linkedin.com/blog/engineering/archived/linkedin-moved-from-rails-to-node; npmjs.com/package/opossum (circuit breaker)

17. EVOLUZIONI FUTURE DI NODE.JS E L'ECOSISTEMA

L'ecosistema JavaScript server-side è in rapida evoluzione. Questa sezione analizza le tendenze più significative per i prossimi anni, con focus su TypeScript nativo, edge computing, Deno 2 e l'integrazione crescente con l'AI.

17.1 TypeScript: il futuro dello sviluppo Node.js

TypeScript è già lo standard de facto per i nuovi progetti Node.js professionali. Nel 2024, la OpenJS Foundation ha annunciato lo sviluppo di un supporto TypeScript nativo in Node.js (Type Stripping), disponibile sperimentalmente in Node.js 22.6+.

■ BASH | TypeScript nativo in Node.js 22+ e configurazione tsconfig.json

```
# Node.js 22.6+: esegui TypeScript senza compilare!
# (solo type stripping, non type checking)
node --experimental-strip-types app.ts

# Installazione TypeScript tradizionale (type checking completo)
npm install --save-dev typescript @types/node ts-node tsx

# tsconfig.json ottimizzato per Node.js 22
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "outDir": "./dist",
    "rootDir": "./src",
    "declaration": true,
    "sourceMap": true
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "dist"]
}
```

■ TYPESCRIPT | TypeScript moderno: types, generics, service layer per Node.js

```
// Esempio TypeScript moderno per API Node.js
// src/types/user.ts
export interface User {
  id: string;
  nome: string;
  email: string;
  role: 'admin' | 'user' | 'moderator';
  createdAt: Date;
  profilo?: {
    bio: string;
    avatar?: string;
  };
}

export type CreateUserDTO = Omit<User, 'id' | 'createdAt'>;
export type UpdateUserDTO = Partial<CreateUserDTO>;

// src/services/userService.ts
import { User, CreateUserDTO } from '../types/user';

export class UserService {
  constructor(private readonly db: DatabaseAdapter) {}

  async create(data: CreateUserDTO): Promise<User> {
    // TypeScript garantisce che i tipi siano corretti a compile-time
    const user = await this.db.users.create({
      ...data,
      id: crypto.randomUUID(),
      createdAt: new Date(),
    });
    return user;
  }

  async findById(id: string): Promise<User | null> {
    return this.db.users.findOne({ id });
  }

  // Generic per query tipizzate
  async paginate<T extends Partial<User>>(
    filter: T,
    page: number = 1,
    limit: number = 20,
  ): Promise<{ data: User[]; total: number; pages: number }> {
    const [data, total] = await Promise.all([
      this.db.users.findMany(filter, { skip: (page-1)*limit, take: limit }),
      this.db.users.count(filter),
    ]);
    return { data, total, pages: Math.ceil(total / limit) };
  }
}
```

17.2 Edge Computing e Workers

Il paradigma **edge computing** porta il codice il più vicino possibile all'utente finale, distribuendo l'esecuzione su centinaia di datacenter globali. Cloudflare Workers, Vercel Edge Functions e Deno Deploy usano runtime JavaScript/V8 leggeri compatibili con le Web API standard, parzialmente compatibili con Node.js.

Piattaforma	Runtime	Cold start	Compatibilità Node	Prezzo
Cloudflare Workers	V8 Isolates	<1ms	Parziale (no fs, no net)	Free tier + $5/mese
Vercel Edge Functions	V8 Isolates	<1ms	Parziale	Incluso in Vercel
Deno Deploy	Deno	~50ms	Deno API + npm compatible	Free + $10/mese
AWS Lambda@Edge	Node.js / Deno	~100ms	Completa (Node.js)	Pay-per-use
Netlify Edge Functions	Deno	~50ms	Deno + npm	Free tier
Fastly Compute	WASM	<1ms	WASM compatible	Pay-per-use

17.3 Deno 2: il ritorno di Ryan Dahl

Nel 2024 Ryan Dahl ha rilasciato **Deno 2.0**, una revisione maggiore che ha affrontato le critiche principali di Deno 1.x: la compatibilità con npm e con il codice Node.js esistente. Deno 2.0 è ora compatibile con npm (senza node_modules), supporta le principali API Node.js e si posiziona come alternativa moderna e sicura.

■ BASH | Deno 2.0 — sicurezza, npm compatible, TypeScript nativo, Deno KV

```
# Deno 2.0: sintassi familiare, sicurezza by default
# Installazione: curl -fsSL https://deno.land/install.sh | sh

# Permessi espliciti (default: accesso negato a tutto)
deno run --allow-net --allow-read app.ts

# Importa da npm senza node_modules!
import express from 'npm:express@4';
import { z } from 'npm:zod@3';

# TypeScript nativo (nessun tsconfig necessario)
deno run app.ts   # funziona direttamente

# Formattazione, linting, test: tutto built-in
deno fmt         # formatta il codice
deno lint        # linting
deno test        # test runner integrato
deno compile app.ts  # compila a eseguibile standalone

# Deno KV: database chiave-valore built-in (senza dipendenze)
const kv = await Deno.openKv();
await kv.set(["users", "mario"], { nome: "Mario", eta: 30 });
const entry = await kv.get(["users", "mario"]);
console.log(entry.value);  // { nome: "Mario", eta: 30 }
```

17.4 Node.js e AI: il futuro dell'integrazione

L'integrazione tra Node.js e intelligenza artificiale sta evolvendo rapidamente su più fronti:

Tendenza	Stato 2025	Orizzonte
LLM come microservizi	Già in produzione (OpenAI, Anthropic SDK)	API standard unificata; latenza sub-100ms

Tendenza	Stato 2025	Orizzonte
Agenti AI autonomi	Fase early adopter (LangChain.js, AI SDK agents)	Agent framework maturi; orchestrazione multi-agent
MCP Server proliferazione	In rapida crescita (1000+ server pubblici 2025)	Standard de facto per tool use AI; marketplace server
RAG (Retrieval Augmented Generation)	Produzione (Pinecone, pgvector, Qdrant)	Vector DB integrati in ORM; ricerca semantica default
Inference locale (LLaMA, Mistral)	Sperimentale (node-llama-cpp)	Inference CPU/GPU diretta da Node.js; privacy by default
TypeScript AI-generated	GitHub Copilot, Cursor diffusi	Pair programming AI standard; codebase AI-ready
Test generati da AI	Sperimentale (Copilot, Codestral)	Test coverage automatizzata dall AI

17.5 Roadmap Node.js 2025-2027

Feature	Versione prevista	Stato
Type Stripping TypeScript nativo	Node.js 22.6+ (flag); stabile v24	Sperimentale
require(ESM) stabile	Node.js 22 (parziale); v24 completo	In sviluppo
Permissions model stabile	Node.js 24	Sperimentale
Fetch API stabile	Node.js 22 LTS	Stabile
WebSocket client nativo	Node.js 22 LTS	Stabile
SQLite built-in (node:sqlite)	Node.js 22.5+	Sperimentale
Single Executable Applications (SEA)	Node.js 20+ (stabile v22)	Stabile
test runner con reporter tap/spec	Node.js 20+ (stabile)	Stabile
glob/glob.sync nativo	Node.js 22	Stabile
AbortSignal.any()	Node.js 22	Stabile

Fonti: nodejs.org/en/blog/announcements/v22-release-announce; github.com/nodejs/node/blob/main/doc/changelogs/CHANGELOG_V22.md; deno.com/blog/v2.0 (Deno 2.0 release post); workers.cloudflare.com; vercel.com/docs/functions/edge-functions; github.com/nicolo-ribaudo/tc39-proposal-type-annotations (TypeScript nativo)

18. APPENDICE: CHEATSHEET E RISORSE

18.1 Cheatsheet: comandi npm e Node.js

Comando	Descrizione
node script.js	Esegui file JavaScript
node --watch script.js	Esegui con riavvio automatico (Node.js 18+)
node --inspect-brk script.js	Avvia in modalità debug (Chrome DevTools)
node -e 'console.log(1+1)'	Esegui codice JavaScript inline
node -p 'process.version'	Esegui e stampa risultato
node --trace-warnings	Mostra stack trace per i warning
npm init -y	Crea package.json con valori default
npm install <pkg>	Installa dipendenza
npm install -D <pkg>	Installa dipendenza sviluppo
npm ci	Installa da lock file (CI/CD)
npm run <script>	Esegui script da package.json
npm audit	Controlla vulnerabilità nelle dipendenze
npm list --depth=0	Lista dipendenze top-level
npm outdated	Mostra pacchetti aggiornabili
npx <comando>	Esegui comando npm senza installare globalmente
nvm use 22	Passa a Node.js 22 (con nvm)
nvm install --lts	Installa ultima versione LTS

18.2 Cheatsheet: API Node.js pià usate

■ JS | Cheatsheet API Node.js: fs, fetch, crypto, util, process

```
// ■■ FILESYSTEM ■■■
```

ESM (ES Modules)

Sistema di moduli standard JavaScript: import / export. Stabile in Node.js 14+.

Event Loop

Ciclo continuo che processa eventi e callback in Node.js usando un singolo thread.

Fastify

Framework web Node.js ad alte performance, alternativa a Express con schema validation.

libuv

Libreria C che implementa l'event loop e l'I/O asincrono cross-platform in Node.js.

LTS (Long Term Support)

Versione Node.js con supporto esteso (30 mesi totali). Usare sempre LTS in produzione.

Middleware

Funzione che elabora req/res nell'handler chain di Express prima del controller finale.

MCP (Model Context Protocol)

Standard Anthropic per la comunicazione tra modelli AI e strumenti/dati esterni.

Monorepo

Repository unico con più package/applicazioni. Tool: Turborepo, Nx, pnpm workspaces.

OWASP Top 10

Lista delle 10 vulnerabilità di sicurezza web più critiche, aggiornata ogni 3-4 anni.

Promise

Oggetto che rappresenta il risultato futuro di un'operazione asincrona (pending/fulfilled/rejected).

RAG (Retrieval Augmented Generation)

Tecnica AI che arricchisce il prompt del LLM con dati recuperati da un database vettoriale.

SemVer

Semantic Versioning: MAJOR.MINOR.PATCH per comunicare la natura degli aggiornamenti software.

SSR (Server-Side Rendering)

Rendering dell'HTML sul server (Node.js) invece che nel browser. Usato da Next.js, Nuxt.

Thread Pool

Pool di thread C++ in libuv per operazioni che non possono usare I/O asincrono (crypto, fs).

V8

Motore JavaScript open source di Google (Chrome) che compila JS in codice macchina nativo.

WebSocket

Protocollo di comunicazione full-duplex su TCP per applicazioni real-time.

Worker Thread

Thread JavaScript separato per operazioni CPU-intensive in Node.js (worker_threads module).

18.4 Risorse di riferimento

Risorsa	URL	Tipo
Documentazione Node.js	nodejs.org/api/	Documentazione ufficiale
MDN Web Docs (JS)	developer.mozilla.org/docs/Web/JavaScript	Riferimento JavaScript
npm registry	npmjs.com	Ricerca e gestione pacchetti
ECMAScript spec	tc39.es/ecma262/	Standard JavaScript ufficiale
Node.green	node.green	Compatibilità ES features in Node.js
Fastify docs	fastify.dev	Framework alternativo ad Express
Prisma docs	prisma.io/docs/	ORM per database SQL
Socket.IO docs	socket.io/docs/v4/	WebSocket con fallback
Vercel AI SDK	sdk.vercel.ai	SDK unificato per LLM in Node.js
Anthropic SDK Node	docs.anthropic.com/sdk/node	SDK Claude per Node.js
MCP Protocol	modelcontextprotocol.io	Spec e SDK per server MCP
OWASP Top 10	owasp.org/Top10/	Sicurezza web top vulnerabilità
Node.js Security WG	github.com/nodejs/security-wg	Security Working Group ufficiale
Node.js best practices	github.com/goldbergyoni/nodebestpractices	Repository best practice
clinic.js	clinicjs.org	Profiling e diagnostica Node.js
Jest docs	jestjs.io	Framework testing JavaScript
Vitest docs	vitest.dev	Test runner moderno e veloce
PM2 docs	pm2.keymetrics.io	Process manager produzione
Deno docs	deno.com/manual	Runtime JavaScript/TypeScript alternativo

Fonti: Tutte le risorse sono verificate e attive al momento della pubblicazione. nodejs.org; npmjs.com; developer.mozilla.org; tc39.es; owasp.org; modelcontextprotocol.io; sdk.vercel.ai; docs.anthropic.com

19. STREAMS AVANZATI E BUFFER

Gli stream sono uno dei concetti più potenti di Node.js: permettono di elaborare dati in ingresso/uscita in modo incrementale, senza caricare tutto in memoria. Questa sezione approfondisce stream personalizzati, backpressure, pipeline complesse e la classe Buffer.

19.1 Quattro tipi di stream

Tipo	Classe base	Metodi da implementare	Esempi
Readable	stream.Readable	_read(size)	fs.createReadStream, http.IncomingMessage, process.stdin
Writable	stream.Writable	_write(chunk, enc, cb)	fs.createWriteStream, http.ServerResponse, process.stdout
Duplex	stream.Duplex	_read + _write	net.Socket, TLS socket, TCP connections
Transform	stream.Transform	_transform(chunk, enc, cb)	zlib.createGzip, crypto.createCipher, csv parser

19.2 Readable stream personalizzato

■ JS | db-cursor-stream.js — Readable stream da database con backpressure

```
const { Readable } = require('stream');

// ■■ Readable in modalità oggetto ■■■
```

■ JS | csv-parser-stream.js — CSVParser Transform + pipeline con generator async

```
const { Transform } = require('stream');

class CSVParser extends Transform {
  constructor(options = {}) {
    super({ ...options, objectMode: true });
    this._buffer = '';
    this._headers = null;
    this._separator = options.separator || ',';
  }

  _transform(chunk, encoding, callback) {
    this._buffer += chunk.toString();
    const lines = this._buffer.split('\n');
    // L ultima riga potrebbe essere incompleta
    this._buffer = lines.pop();

    for (const line of lines) {
      if (!line.trim()) continue;

      const values = this._parseLine(line);

      if (!this._headers) {
        this._headers = values;  // prima riga = header
        continue;
      }

      // Crea oggetto dalle intestazioni
      const obj = {};
      this._headers.forEach((h, i) => {
        obj[h] = values[i] || '';
      });
      this.push(obj);  // emetti oggetto JavaScript
    }
    callback();
  }

  _flush(callback) {
    // Processa eventuale ultima riga senza newline
    if (this._buffer.trim() && this._headers) {
      const values = this._parseLine(this._buffer);
      const obj = {};
      this._headers.forEach((h, i) => { obj[h] = values[i] || ''; });
      this.push(obj);
    }
    callback();
  }

  _parseLine(line) {
    // Parser CSV semplice (non gestisce escape, usare csv-parse per produzione)
    return line.split(this._separator).map(v => v.trim().replace(/^"|"$/g, ''));
  }
}

// Pipeline: leggi CSV, filtra, trasforma, scrivi JSON
const { pipeline } = require('stream/promises');
const fs = require('fs');

await pipeline(
  fs.createReadStream('dati.csv', { encoding: 'utf8' }),
  new CSVParser({ separator: ';' }),
  new Transform({
    objectMode: true,
    transform(record, enc, cb) {
```

```
      // Filtra solo record con importo > 1000
      if (parseFloat(record.importo) > 1000) {
        cb(null, record);
      } else {
        cb();  // scarta il record
      }
    }
  }),
  async function*(source) {
    // Generator async come ultimo stage (Node.js 16+)
    let count = 0;
    for await (const record of source) {
      yield JSON.stringify(record) + '\n';
      count++;
    }
    console.log(`${count} record processati`);
  },
  fs.createWriteStream('output.ndjson')
);
```

19.4 Buffer: dati binari in Node.js

■ JS | buffer.js — Buffer: creazione, conversioni, protocolli binari, sicurezza

```
// Buffer: gestione dati binari raw (immagini, file, protocolli)
// NON usare 'new Buffer()' (deprecato): usa Buffer.from/alloc/allocUnsafe

// ■■ CREAZIONE ■■■
```

20. PATTERN DI DESIGN IN NODE.JS

I pattern di design sono soluzioni riutilizzabili a problemi ricorrenti. In Node.js, l'architettura event-driven e la natura asincrona influenzano quali pattern sono più adatti e come si implementano.

20.1 Singleton e Dependency Injection

■ JS | patterns-di.js — Singleton con moduli Node.js e Dependency Injection

```
// ■■ SINGLETON: modulo Node.js è già un singleton ■■■■■■■■■■■■■■■■■■
// I moduli CommonJS vengono cacheati: ogni require() restituisce
// la stessa istanza. Sfrutta questo per singleton.

// database.js
const mongoose = require('mongoose');
let connection = null;

async function getConnection() {
  if (connection) return connection;
  connection = await mongoose.connect(process.env.MONGODB_URI);
  return connection;
}

module.exports = { getConnection };
// Ogni require('./database') restituisce lo stesso modulo
// e quindi la stessa connessione.

// ■■ DEPENDENCY INJECTION con factory function ■■■■■■■■■■■■■■■■■■■■■■■
// Evita accoppiamento diretto; facilita il testing

// userService.js - factory con dipendenze iniettate
function createUserService({ userRepository, emailService, logger }) {
  return {
    async register(userData) {
      logger.info('Registrazione utente:', userData.email);

      const existing = await userRepository.findByEmail(userData.email);
      if (existing) throw new Error('Email già in uso');

      const user = await userRepository.create(userData);
      await emailService.sendWelcome(user.email, user.nome);
      return user;
    },

    async getById(id) {
      return userRepository.findById(id);
    },
  };
}

// Produzione: inietta dipendenze reali
const userService = createUserService({
  userRepository: require('./repositories/userRepository'),
  emailService: require('./services/emailService'),
  logger: require('./utils/logger'),
});

// Testing: inietta mock
const testUserService = createUserService({
  userRepository: { findByEmail: jest.fn(), create: jest.fn(), findById: jest.fn() },
  emailService: { sendWelcome: jest.fn() },
  logger: { info: jest.fn(), error: jest.fn() },
});
```

20.2 Repository Pattern con Node.js

■ JS | repository-pattern.js — Repository con MongoDB e InMemory per test

```
// Repository pattern: astrae l accesso ai dati
// Il controller non sa se usa MongoDB, PostgreSQL, o un mock

// src/repositories/userRepository.js
class UserRepository {
  constructor(db) {
    this.db = db;  // db iniettato: Mongoose model o Prisma client
  }

  async findById(id) {
    return this.db.findById(id).lean();
  }

  async findByEmail(email) {
    return this.db.findOne({ email: email.toLowerCase() }).lean();
  }

  async create(data) {
    return this.db.create(data);
  }

  async update(id, data) {
    return this.db.findByIdAndUpdate(id, data, { new: true, lean: true });
  }

  async delete(id) {
    return this.db.findByIdAndDelete(id);
  }

  async paginate({ page = 1, limit = 20, filter = {}, sort = { createdAt: -1 } }) {
    const skip = (page - 1) * limit;
    const [data, total] = await Promise.all([
      this.db.find(filter).sort(sort).skip(skip).limit(limit).lean(),
      this.db.countDocuments(filter),
    ]);
    return {
      data,
      pagination: { page, limit, total, pages: Math.ceil(total / limit) },
    };
  }
}

// Uso con MongoDB
const UserModel = require('../models/User');
const userRepo = new UserRepository(UserModel);

// Uso con in-memory store (testing)
class InMemoryUserRepository {
  constructor() { this.store = new Map(); }
  async findById(id) { return this.store.get(id) || null; }
  async create(data) {
    const user = { ...data, id: Date.now().toString() };
    this.store.set(user.id, user);
    return user;
  }
  async delete(id) {
    const user = this.store.get(id);
    this.store.delete(id);
    return user;
  }
}
```

20.3 Observer / EventEmitter avanzato

■ JS | event-bus.js — Bus di eventi in-process con EventEmitter avanzato

```
const { EventEmitter } = require('events');

// ■■ BUS DI EVENTI CENTRALIZZATO (in-process pub/sub) ■■■■■■■■■■■■■■■
class EventBus extends EventEmitter {
  constructor() {
    super();
    this.setMaxListeners(50);
    this._history = [];
  }

  publish(topic, payload) {
    const event = { topic, payload, ts: Date.now(), id: crypto.randomUUID() };
    this._history.push(event);
    this.emit(topic, event);
    this.emit('*', event);  // wildcard listener
    return event.id;
  }

  subscribe(topic, handler) {
    this.on(topic, handler);
    return () => this.off(topic, handler);  // ritorna unsubscribe fn
  }

  once_async(topic) {
    return new Promise((resolve) => this.once(topic, resolve));
  }
}

const bus = new EventBus();

// Subscriber per notifiche
const unsubscribeOrder = bus.subscribe('order.created', async ({ payload }) => {
  await emailService.sendOrderConfirmation(payload.userId, payload.orderId);
});

const unsubscribeInventory = bus.subscribe('order.created', async ({ payload }) => {
  await inventoryService.reserveItems(payload.items);
});

// Publisher nel controller
async function createOrder(req, res, next) {
  try {
    const order = await orderService.create(req.body);
    bus.publish('order.created', order);  // notifica tutti i subscriber
    res.status(201).json({ data: order });
  } catch (err) { next(err); }
}

// Aspetta un evento specifico (utile nei test)
const orderEvent = await bus.once_async('order.shipped');
console.log('Ordine spedito:', orderEvent.payload.orderId);

// Cleanup quando non necessario
unsubscribeOrder();
unsubscribeInventory();
```

20.4 Circuit Breaker pattern

■ JS | circuit-breaker.js — Circuit Breaker con opossum per resilienza

```
// Circuit Breaker: protegge da cascade failures
// npm install opossum  (libreria production-grade)
const CircuitBreaker = require('opossum');

// Funzione da proteggere (chiamata a servizio esterno)
async function callExternalAPI(payload) {
  const res = await fetch('https://api.external.com/process', {
    method: 'POST',
    body: JSON.stringify(payload),
    signal: AbortSignal.timeout(3000),
  });
  if (!res.ok) throw new Error(`API error: ${res.status}`);
  return res.json();
}

// Configurazione Circuit Breaker
const breaker = new CircuitBreaker(callExternalAPI, {
  timeout: 3000,           // timeout singola chiamata
  errorThresholdPercentage: 50,  // apri circuito se >50% fallisce
  resetTimeout: 30000,     // riprova dopo 30s (half-open state)
  volumeThreshold: 5,      // min 5 chiamate prima di valutare
});

// Fallback: cosa fare quando il circuito è aperto
breaker.fallback((payload) => ({
  status: 'degraded',
  message: 'Servizio temporaneamente non disponibile',
  cachedResult: cache.get(JSON.stringify(payload)),
}));

// Monitoraggio stati
breaker.on('open', () => {
  logger.error('Circuit breaker APERTO: servizio esterno non raggiungibile');
  metrics.increment('circuit_breaker.open');
});
breaker.on('halfOpen', () => logger.warn('Circuit breaker HALF-OPEN: test...'));
breaker.on('close', () => logger.info('Circuit breaker CHIUSO: servizio ripristinato'));

// Uso trasparente
app.post('/api/process', async (req, res, next) => {
  try {
    const result = await breaker.fire(req.body);
    res.json(result);
  } catch (err) { next(err); }
});
```

Fonti: nodejs.org/api/stream.html; nodejs.org/api/buffer.html; npmjs.com/package/opossum (circuit breaker); martinfowler.com/eaaCatalog/repository.html; refactoring.guru/design-patterns (pattern di design con esempi)

21. GRAPHQL IN NODE.JS

GraphQL, sviluppato da Facebook nel 2012 e rilasciato open source nel 2015, è un linguaggio di query per API che offre un'alternativa flessibile alle API REST tradizionali. In Node.js, le implementazioni principali sono **Apollo Server** (la più diffusa) e **GraphQL Yoga** (più leggera, edge-ready).

21.1 Confronto REST vs GraphQL

Aspetto	REST	GraphQL
Endpoint	Multipli (/users, /posts, /comments)	Singolo (/graphql)
Dati ricevuti	Fissi (over/under-fetching comune)	Esatti (il client specifica i campi)
Versioning	Versioni URL (/v1/, /v2/)	Schema evolve con deprecation
Documentazione	OpenAPI/Swagger (manuale)	Introspection automatica (GraphiQL)
Real-time	Richiede SSE o WebSocket separati	Subscription integrata nello schema
Performance	N+1 risolvibile con join DB	N+1 problem richiede DataLoader
Curva apprendimento	Bassa	Media-alta
Quando usarlo	API pubbliche, CRUD semplice, microservizi	BFF, app mobile, dashboard complesse

21.2 Apollo Server: schema e resolver completi

■ BASH | Installazione Apollo Server e dipendenze

```bash
# npm install @apollo/server graphql graphql-scalars
# npm install @prisma/client mongoose  (per i datasource)
```

■ JS | schema.js — Schema GraphQL completo con Query, Mutation, Subscription

```
// src/graphql/schema.js — Schema GraphQL completo
const { gql } = require('graphql-tag');

const typeDefs = gql`
  scalar DateTime
  scalar JSON

  type User {
    id: ID!
    nome: String!
    email: String!
    role: Role!
    createdAt: DateTime!
    posts: [Post!]!
    postsCount: Int!
  }

  type Post {
    id: ID!
    titolo: String!
    contenuto: String
    pubblicato: Boolean!
    autore: User!
    tags: [String!]!
    createdAt: DateTime!
  }

  enum Role {
    USER
    ADMIN
    MODERATOR
  }

  type PaginatedUsers {
    data: [User!]!
    total: Int!
    page: Int!
    pages: Int!
  }

  type Query {
    # Utenti
    users(page: Int, limit: Int, role: Role): PaginatedUsers!
    user(id: ID!): User
    me: User  # utente autenticato

    # Post
    posts(pubblicato: Boolean, autoreId: ID): [Post!]!
    post(id: ID!): Post
  }

  type Mutation {
    # Auth
    login(email: String!, password: String!): AuthPayload!
    register(input: RegisterInput!): AuthPayload!

    # Post
    createPost(input: CreatePostInput!): Post!
    updatePost(id: ID!, input: UpdatePostInput!): Post!
    deletePost(id: ID!): Boolean!
    publishPost(id: ID!): Post!
  }

  type Subscription {
```

```
    postCreated: Post!
    postPublished(autoreId: ID): Post!
  }

  type AuthPayload {
    token: String!
    user: User!
  }

  input RegisterInput {
    nome: String!
    email: String!
    password: String!
  }

  input CreatePostInput {
    titolo: String!
    contenuto: String
    tags: [String!]
  }

  input UpdatePostInput {
    titolo: String
    contenuto: String
    tags: [String!]
  }
`;

module.exports = { typeDefs };
```

■ JS | resolvers.js — Resolver con DataLoader (anti N+1) e autenticazione

```
// src/graphql/resolvers.js — Resolver con autenticazione e DataLoader
const { GraphQLError } = require('graphql');
const DataLoader = require('dataloader');

// DataLoader: risolve il problema N+1
// Invece di N query separate, fa UNA sola query per batch
function createLoaders(db) {
  return {
    userById: new DataLoader(async (ids) => {
      const users = await db.user.findMany({ where: { id: { in: ids } } });
      const userMap = new Map(users.map(u => [u.id, u]));
      return ids.map(id => userMap.get(id) || null);
    }),
    postsByAuthor: new DataLoader(async (authorIds) => {
      const posts = await db.post.findMany({
        where: { autoreId: { in: authorIds } },
      });
      const map = new Map();
      authorIds.forEach(id => map.set(id, []));
      posts.forEach(p => map.get(p.autoreId)?.push(p));
      return authorIds.map(id => map.get(id) || []);
    }),
  };
}

const resolvers = {
  Query: {
    users: async (_, { page = 1, limit = 20, role }, { db, user: ctxUser }) => {
      if (!ctxUser) throw new GraphQLError('Non autenticato', {
        extensions: { code: 'UNAUTHENTICATED' }
      });
      const skip = (page - 1) * limit;
      const where = role ? { role } : {};
      const [data, total] = await Promise.all([
        db.user.findMany({ where, skip, take: limit }),
        db.user.count({ where }),
      ]);
      return { data, total, page, pages: Math.ceil(total / limit) };
    },

    me: (_, __, { user }) => {
      if (!user) throw new GraphQLError('Non autenticato',
        { extensions: { code: 'UNAUTHENTICATED' } });
      return user;
    },
  },

  User: {
    // Resolver per campi nested — usa DataLoader per evitare N+1
    posts: ({ id }, _, { loaders }) => loaders.postsByAuthor.load(id),
    postsCount: async ({ id }, _, { db }) =>
      db.post.count({ where: { autoreId: id } }),
  },

  Post: {
    autore: ({ autoreId }, _, { loaders }) => loaders.userById.load(autoreId),
  },

  Mutation: {
    createPost: async (_, { input }, { db, user }) => {
      if (!user) throw new GraphQLError('Non autenticato',
        { extensions: { code: 'UNAUTHENTICATED' } });
      return db.post.create({
```

```
        data: { ...input, autoreId: user.id },
        include: { autore: true },
      });
    },
  },
};

module.exports = { resolvers, createLoaders };
```

Fonti: apollographql.com/docs/apollo-server/; graphql.org; npmjs.com/package/dataloader (DataLoader di Facebook); the-guild.dev/graphql/yoga-server (GraphQL Yoga); graphql-scalars.com (scalari custom: DateTime, JSON, etc.)

22. GRPC E MICROSERVIZI AVANZATI

gRPC (Google Remote Procedure Call) è un framework RPC open source ad alte performance basato su HTTP/2 e Protocol Buffers. In Node.js è la scelta ideale per comunicazioni inter-servizio in architetture microservizi dove latenza e throughput sono critici.

22.1 gRPC vs REST vs GraphQL

Criterio	REST (HTTP/1.1)	GraphQL	gRPC (HTTP/2)
Protocollo	HTTP/1.1 JSON	HTTP/1.1 JSON	HTTP/2 Protobuf
Serializzazione	JSON (testo)	JSON (testo)	Protobuf (binario, 5-10x piu' compatto)
Streaming	Limitato (SSE, WS separati)	Subscription	Nativo (server/client/bidi)
Latenza	Media	Media	Bassa (multiplexing HTTP/2)
Contratto tipizzato	OpenAPI (opzionale)	Schema GraphQL	Proto file (obbligatorio)
Browser support	Nativo	Nativo	Richiede gRPC-Web proxy
Caso d uso ideale	API pubblica, CRUD	BFF, dashboard	Microservizi interni, streaming

22.2 Definizione Protobuf e server gRPC

■ PROTOBUF | user_service.proto — definizione contratto gRPC con tutti i tipi di RPC

```
// proto/user_service.proto — definizione del contratto
syntax = "proto3";

package userservice;

service UserService {
  // Unary RPC (una richiesta, una risposta)
  rpc GetUser (GetUserRequest) returns (User);
  rpc CreateUser (CreateUserRequest) returns (User);

  // Server streaming (una richiesta, stream di risposte)
  rpc ListUsers (ListUsersRequest) returns (stream User);

  // Client streaming (stream richieste, una risposta)
  rpc BulkCreateUsers (stream CreateUserRequest) returns (BulkResult);

  // Bidirectional streaming
  rpc UserUpdates (stream UpdateRequest) returns (stream User);
}

message User {
  string id = 1;
  string nome = 2;
  string email = 3;
  string role = 4;
  int64 created_at = 5;  // Unix timestamp
}

message GetUserRequest {
  string id = 1;
}

message CreateUserRequest {
  string nome = 1;
  string email = 2;
  string password = 3;
  string role = 4;
}

message ListUsersRequest {
  int32 page = 1;
  int32 limit = 2;
  string role_filter = 3;
}

message BulkResult {
  int32 created = 1;
  int32 failed = 2;
  repeated string errors = 3;
}
```

■ JS | grpc-server.js — Server gRPC con Unary, Server Streaming, Client Streaming

```
// npm install @grpc/grpc-js @grpc/proto-loader
// src/grpc/server.js — Server gRPC Node.js

const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');
const path = require('path');

const PROTO_PATH = path.join(__dirname, '../../proto/user_service.proto');

const packageDef = protoLoader.loadSync(PROTO_PATH, {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
});

const { userservice } = grpc.loadPackageDefinition(packageDef);

// Implementazione servizio
const userServiceImpl = {
  // Unary
  async GetUser(call, callback) {
    try {
      const user = await UserRepository.findById(call.request.id);
      if (!user) {
        callback({
          code: grpc.status.NOT_FOUND,
          message: `Utente ${call.request.id} non trovato`,
        });
        return;
      }
      callback(null, user);
    } catch (err) {
      callback({ code: grpc.status.INTERNAL, message: err.message });
    }
  },

  // Server streaming: invia utenti uno alla volta
  async ListUsers(call) {
    const { page = 1, limit = 50, role_filter } = call.request;
    const filter = role_filter ? { role: role_filter } : {};

    const cursor = UserRepository.stream(filter, { page, limit });
    cursor.on('data', (user) => call.write(user));
    cursor.on('end', () => call.end());
    cursor.on('error', (err) => call.destroy(err));
  },

  // Client streaming: ricevi batch di utenti
  BulkCreateUsers(call, callback) {
    const results = { created: 0, failed: 0, errors: [] };
    const createPromises = [];

    call.on('data', (req) => {
      createPromises.push(
        UserRepository.create(req)
          .then(() => { results.created++; })
          .catch(err => {
            results.failed++;
            results.errors.push(`${req.email}: ${err.message}`);
          })
      );
```

```
    });

    call.on('end', async () => {
      await Promise.all(createPromises);
      callback(null, results);
    });
  },
};

// Avvio server
const server = new grpc.Server();
server.addService(userservice.UserService.service, userServiceImpl);
server.bindAsync(
  '0.0.0.0:50051',
  grpc.ServerCredentials.createInsecure(),  // TLS in produzione
  (err, port) => {
    if (err) throw err;
    console.log(`gRPC server in ascolto su port ${port}`);
  }
);
```

Fonti: grpc.io/docs/languages/node/; npmjs.com/package/@grpc/grpc-js; protobuf.dev (Protocol Buffers documentation); github.com/grpc/grpc-node (gRPC Node.js ufficiale)

23. LOGGING, MONITORAGGIO E OBSERVABILITY

In produzione, la qualità del logging e del monitoraggio determina la capacità di diagnosticare problemi rapidamente. L'observability moderna si basa su tre pilastri: **logs** (eventi strutturati), **metrics** (misure numeriche) e **traces** (percorso di una richiesta attraverso i servizi).

23.1 Logging strutturato con Pino

■ BASH | Installazione Pino

```
# npm install pino pino-pretty pino-http
# Pino è il logger più veloce per Node.js (benchmarks pino.github.io)
```

■ JS | logger.js — Pino: logging strutturato, redact, serializzatori, HTTP middleware

```
// src/utils/logger.js — configurazione Pino production-ready
const pino = require('pino');

const logger = pino({
  level: process.env.LOG_LEVEL || 'info',
  // Redact automatico di campi sensibili (MAI loggare password/token!)
  redact: {
    paths: ['password', '*.password', 'token', 'authorization',
            'req.headers.authorization', '*.creditCard', '*.ssn'],
    censor: '[REDACTED]',
  },
  // Serializzatori personalizzati
  serializers: {
    req: pino.stdSerializers.req,
    res: pino.stdSerializers.res,
    err: pino.stdSerializers.err,
    user: (user) => user ? { id: user.id, role: user.role } : null,
  },
  // Produzione: JSON puro (niente colori, parsing veloce da strumenti)
  // Sviluppo: pino-pretty formatta con colori
  transport: process.env.NODE_ENV !== 'production' ? {
    target: 'pino-pretty',
    options: { colorize: true, translateTime: 'SYS:standard' }
  } : undefined,
  // Campi base su ogni log
  base: {
    env: process.env.NODE_ENV,
    version: process.env.npm_package_version,
    pid: process.pid,
    hostname: require('os').hostname(),
  },
});

module.exports = logger;

// src/middleware/requestLogger.js — log ogni richiesta HTTP
const pinoHttp = require('pino-http');
const logger = require('../utils/logger');

module.exports = pinoHttp({
  logger,
  // Livello per status code
  customLogLevel(req, res, err) {
    if (res.statusCode >= 500 || err) return 'error';
    if (res.statusCode >= 400) return 'warn';
    return 'info';
  },
  // Log body su errori (attenzione a dati sensibili)
  customSuccessMessage(req, res) {
    return `${req.method} ${req.url} ${res.statusCode}`;
  },
  // Escludi health check dal log
  autoLogging: {
    ignore: (req) => req.url === '/health',
  },
});

// Uso nel codice applicativo
const logger = require('./utils/logger');
const child = logger.child({ component: 'UserService', requestId: req.id });

child.info({ userId: user.id, action: 'login' }, 'Utente autenticato');
child.warn({ attempts: 5, ip: req.ip }, 'Troppi tentativi di login');
```

```
child.error({ err, userId }, 'Errore durante la registrazione');
```

23.2 Metrics con Prometheus e Grafana

■ JS | metrics.js — Prometheus: counter, histogram, gauge, middleware Express

```
// npm install prom-client
// prom-client: libreria ufficiale Prometheus per Node.js

const client = require('prom-client');
const express = require('express');

// Abilita metriche default (CPU, memoria, event loop lag)
const register = new client.Registry();
client.collectDefaultMetrics({ register });

// ■■ METRICHE CUSTOM ■■■
```

```
  res.end(await register.metrics());
});

// Aggiorna gauge connessioni WebSocket
wss.on('connection', () => activeConnections.inc());
wss.on('close', () => activeConnections.dec());
```

23.3 Distributed Tracing con OpenTelemetry

■ JS | tracing.js — OpenTelemetry: distributed tracing automatico per Node.js

```
# npm install @opentelemetry/sdk-node @opentelemetry/auto-instrumentations-node
# npm install @opentelemetry/exporter-trace-otlp-http

// src/tracing.js — setup OpenTelemetry (caricare PRIMA di tutto)
const { NodeSDK } = require('@opentelemetry/sdk-node');
const { getNodeAutoInstrumentations } = require(
  '@opentelemetry/auto-instrumentations-node'
);
const { OTLPTraceExporter } = require(
  '@opentelemetry/exporter-trace-otlp-http'
);

const sdk = new NodeSDK({
  serviceName: 'mia-api',
  serviceVersion: process.env.npm_package_version,
  traceExporter: new OTLPTraceExporter({
    url: process.env.OTEL_EXPORTER_OTLP_ENDPOINT || 'http://localhost:4318/v1/traces',
  }),
  // Auto-instrumenta Express, MongoDB, Redis, fetch, http
  instrumentations: [getNodeAutoInstrumentations({
    '@opentelemetry/instrumentation-fs': { enabled: false },
  })],
});

sdk.start();
console.log('OpenTelemetry tracing avviato');

// Graceful shutdown
process.on('SIGTERM', () => {
  sdk.shutdown().then(() => process.exit(0));
});

// Avvio: node -r ./src/tracing.js src/index.js
// Oppure in package.json: "start": "node -r ./src/tracing.js src/index.js"
```

Fonti: pino.github.io; npmjs.com/package/pino; prometheus.io/docs; npmjs.com/package/prom-client; opentelemetry.io/docs/languages/js/; grafana.com/docs/ (Grafana dashboard per Prometheus); jaegertracing.io (tracing backend open source)

24. TESTING AVANZATO E TDD

Il testing è un investimento che ripaga nel tempo. Questa sezione approfondisce il Test-Driven Development (TDD), i test end-to-end con Playwright, e i test di carico con k6, con esempi pratici applicabili a progetti reali.

24.1 TDD: Test-Driven Development in Node.js

Il TDD segue il ciclo **Red-Green-Refactor**: 1) scrivi un test che fallisce (Red), 2) scrivi il codice minimo per farlo passare (Green), 3) refactoring mantenendo i test verdi. Risultato: codice più modulare, testabile e documentato.

■ JS | cart.test.js + cart.js — ciclo TDD Red-Green-Refactor completo

```
// TDD esempio: implementazione di un servizio carrello

// STEP 1 — RED: scrivi il test prima del codice
// tests/cart.test.js
const { createCart } = require('../src/cart');

describe('Cart Service', () => {
  let cart;

  beforeEach(() => {
    cart = createCart();  // questo non esiste ancora!
  });

  test('carrello vuoto ha totale 0', () => {
    expect(cart.total()).toBe(0);
    expect(cart.items()).toHaveLength(0);
  });

  test('aggiunge articolo al carrello', () => {
    cart.add({ id: 'p1', nome: 'Laptop', prezzo: 999.99, quantita: 1 });
    expect(cart.items()).toHaveLength(1);
    expect(cart.total()).toBe(999.99);
  });

  test('aggiorna quantita se articolo già presente', () => {
    cart.add({ id: 'p1', nome: 'Mouse', prezzo: 29.99, quantita: 1 });
    cart.add({ id: 'p1', nome: 'Mouse', prezzo: 29.99, quantita: 2 });
    expect(cart.items()).toHaveLength(1);
    expect(cart.items()[0].quantita).toBe(3);
    expect(cart.total()).toBeCloseTo(89.97);
  });

  test('rimuove articolo dal carrello', () => {
    cart.add({ id: 'p1', nome: 'Tastiera', prezzo: 79.99, quantita: 1 });
    cart.remove('p1');
    expect(cart.items()).toHaveLength(0);
    expect(cart.total()).toBe(0);
  });

  test('lancia errore per prezzo negativo', () => {
    expect(() => cart.add({ id: 'p2', nome: 'Errore', prezzo: -10 }))
      .toThrow('Prezzo non valido');
  });
});

// STEP 2 — GREEN: implementazione minima che passa i test
// src/cart.js
function createCart() {
  const _items = new Map();

  return {
    add(item) {
      if (typeof item.prezzo !== 'number' || item.prezzo < 0) {
        throw new Error('Prezzo non valido');
      }
      if (_items.has(item.id)) {
        const existing = _items.get(item.id);
        existing.quantita += item.quantita || 1;
      } else {
        _items.set(item.id, { ...item, quantita: item.quantita || 1 });
      }
    },

    remove(id) { _items.delete(id); },
```

```
    items() { return Array.from(_items.values()); },

    total() {
      return Array.from(_items.values())
        .reduce((sum, item) => sum + item.prezzo * item.quantita, 0);
    },
  };
}

module.exports = { createCart };
// npm test -> tutti e 5 i test passano (GREEN)
```

24.2 Test E2E con Playwright

■ BASH | Setup Playwright

```
# npm install --save-dev @playwright/test
# npx playwright install  (scarica browser: Chromium, Firefox, WebKit)
```

■ TYPESCRIPT | playwright — test E2E per login UI e API REST su browser multipli

```
// tests/e2e/login.spec.ts — Test E2E con Playwright
import { test, expect } from '@playwright/test';

test.describe('Autenticazione', () => {
  test.beforeEach(async ({ page }) => {
    await page.goto('http://localhost:3000');
  });

  test('login con credenziali corrette', async ({ page }) => {
    await page.getByLabel('Email').fill('admin@example.com');
    await page.getByLabel('Password').fill('Admin@123');
    await page.getByRole('button', { name: 'Accedi' }).click();

    // Attendi navigazione e verifica redirect
    await expect(page).toHaveURL('/dashboard');
    await expect(page.getByText('Benvenuto, Admin')).toBeVisible();
  });

  test('mostra errore con credenziali errate', async ({ page }) => {
    await page.getByLabel('Email').fill('wrong@example.com');
    await page.getByLabel('Password').fill('wrongpass');
    await page.getByRole('button', { name: 'Accedi' }).click();

    const alert = page.getByRole('alert');
    await expect(alert).toBeVisible();
    await expect(alert).toContainText('Credenziali non valide');
  });

  test('API: crea e recupera utente', async ({ request }) => {
    // Test API REST direttamente con Playwright request context
    const createRes = await request.post('/api/v1/users', {
      data: { nome: 'Test E2E', email: 'e2e@test.com', password: 'Test@123' },
      headers: { Authorization: 'Bearer admin-token' },
    });
    expect(createRes.status()).toBe(201);
    const { data: newUser } = await createRes.json();
    expect(newUser.email).toBe('e2e@test.com');

    const getRes = await request.get(`/api/v1/users/${newUser.id}`, {
      headers: { Authorization: 'Bearer admin-token' },
    });
    expect(getRes.ok()).toBeTruthy();
  });
});

// playwright.config.ts
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './tests/e2e',
  timeout: 30000,
  retries: process.env.CI ? 2 : 0,
  use: {
    baseURL: 'http://localhost:3000',
    trace: 'on-first-retry',  // registra trace per debug
    screenshot: 'only-on-failure',
  },
  projects: [
    { name: 'chromium', use: { browserName: 'chromium' } },
    { name: 'firefox',  use: { browserName: 'firefox'  } },
  ],
  webServer: {
    command: 'npm run start:test',
```

```
    port: 3000,
    reuseExistingServer: !process.env.CI,
  },
});
```

24.3 Test di carico con k6

■ JS | load-test.js — test di carico k6 con stages, soglie e metriche custom

```
// load-test.js — Test di carico con k6 (Grafana k6)
// Installazione: https://k6.io/docs/getting-started/installation/
// Esecuzione: k6 run load-test.js

import http from 'k6/http';
import { check, group, sleep } from 'k6';
import { Counter, Rate, Trend } from 'k6/metrics';

// Metriche custom
const authFailures = new Counter('auth_failures');
const apiErrorRate = new Rate('api_errors');
const loginDuration = new Trend('login_duration');

// Configurazione scenario
export const options = {
  stages: [
    { duration: '1m',  target: 10  },  // ramp up: 0 -> 10 VU in 1min
    { duration: '3m',  target: 50  },  // carico normale: 50 VU per 3min
    { duration: '1m',  target: 100 },  // picco: 100 VU
    { duration: '2m',  target: 100 },  // sostieni picco per 2min
    { duration: '1m',  target: 0   },  // ramp down
  ],
  thresholds: {
    http_req_duration: ['p(95)<500', 'p(99)<1000'],  // 95% < 500ms
    http_req_failed: ['rate<0.01'],                   // meno dell 1% errori
    login_duration: ['avg<300'],                       // media login < 300ms
  },
};

const BASE_URL = 'http://localhost:3000/api/v1';

export default function () {
  group('Autenticazione', () => {
    const loginStart = Date.now();
    const loginRes = http.post(`${BASE_URL}/auth/login`, JSON.stringify({
      email: 'test@example.com',
      password: 'Test@123',
    }), { headers: { 'Content-Type': 'application/json' } });

    loginDuration.add(Date.now() - loginStart);

    const ok = check(loginRes, {
      'login status 200': (r) => r.status === 200,
      'token presente': (r) => r.json('token') !== undefined,
    });

    if (!ok) authFailures.add(1);

    const token = loginRes.json('token');

    group('API Users', () => {
      const usersRes = http.get(`${BASE_URL}/users`, {
        headers: { Authorization: `Bearer ${token}` },
      });
      const usersOk = check(usersRes, {
        'users status 200': (r) => r.status === 200,
        'ritorna array': (r) => Array.isArray(r.json('data')),
      });
      apiErrorRate.add(!usersOk);
    });
  });

  sleep(1);  // pausa 1s tra iterazioni
```

```
}

// Esegui: k6 run --out json=results.json load-test.js
// Report HTML: k6 run --out html=report.html load-test.js
```

Fonti: jestjs.io; vitest.dev; playwright.dev; k6.io; martinfowler.com/articles/practical-test-pyramid.html; kentcdodds.com/blog/write-tests (Kent C. Dodds: filosofia testing)

25. TYPESCRIPT AVANZATO IN NODE.JS

TypeScript porta il sistema di tipi nel mondo JavaScript, rilevando errori a compile-time invece che a runtime. Nel 2024, oltre il 70% dei nuovi progetti Node.js professionali usa TypeScript (State of JS 2024). Questa sezione copre i tipi avanzati e i pattern più utili per applicazioni Node.js di produzione.

25.1 Tipi avanzati e utility types

■ TYPESCRIPT | typescript-advanced.ts — union, utility types, mapped, conditional, template literal

```
// ■■ TIPI UNION E INTERSECTION ■■■
```

25.2 Generic avanzati e infer

■ TYPESCRIPT | typescript-generics.ts — Repository generico, Builder type-safe, infer

```
// ■■ GENERIC REPOSITORY TYPE-SAFE ■■■
```

```
type FirstParam<T extends (first: any, ...rest: any) => any> =
  T extends (first: infer F, ...rest: any) => any ? F : never;

type FetchUserFirstParam = FirstParam<typeof fetchUser>; // = string
```

25.3 Decoratori e metadata (TypeScript 5)

■ TYPESCRIPT | decorators.ts — decoratori Log e Retry riutilizzabili con TypeScript 5

```
// tsconfig.json per decoratori:
// "experimentalDecorators": true, "emitDecoratorMetadata": true

// ■■ DECORATORI DI CLASSE E METODO ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
// Decoratore di logging automatico
function Log(target: any, propKey: string, descriptor: PropertyDescriptor) {
  const original = descriptor.value;
  descriptor.value = async function(...args: any[]) {
    console.time(`${target.constructor.name}.${propKey}`);
    try {
      const result = await original.apply(this, args);
      console.timeEnd(`${target.constructor.name}.${propKey}`);
      return result;
    } catch (err) {
      console.error(`${propKey} fallito:`, err);
      throw err;
    }
  };
  return descriptor;
}

// Decoratore Retry
function Retry(attempts = 3, delay = 1000) {
  return function(target: any, propKey: string, descriptor: PropertyDescriptor) {
    const original = descriptor.value;
    descriptor.value = async function(...args: any[]) {
      for (let i = 0; i < attempts; i++) {
        try {
          return await original.apply(this, args);
        } catch (err) {
          if (i === attempts - 1) throw err;
          await new Promise(r => setTimeout(r, delay * (i + 1)));
          console.warn(`Retry ${i + 1}/${attempts} per ${propKey}`);
        }
      }
    };
  };
}

// Uso dei decoratori
class PaymentService {
  @Log
  @Retry(3, 500)
  async processPayment(amount: number, userId: string): Promise<string> {
    // Chiama API di pagamento esterna
    const res = await fetch('https://api.stripe.com/v1/charges', { /* ... */ });
    return res.json();
  }
}
```

Fonti: typescriptlang.org/docs/; typescriptlang.org/docs/handbook/2/types-from-types.html (utility types); stateofjs.com/en-US/2024 (State of JS 2024 survey); github.com/type-challenges/type-challenges (esercizi TypeScript avanzati)

26. TOOLCHAIN MODERNA, MONOREPO E CI/CD

La toolchain moderna per Node.js comprende bundler, linter, formatter, e pipeline CI/CD. Per progetti con più package correlati, il monorepo è diventato lo standard nell'industria (usato da Google, Meta, Microsoft, Vercel, Nx).

26.1 Monorepo con pnpm workspaces e Turborepo

■ BASH | Struttura monorepo con pnpm workspaces e Turborepo

```
# Struttura monorepo con pnpm workspaces
monorepo/
■■■ apps/
■   ■■■ api/          # Node.js Express API
■   ■■■ web/          # Next.js frontend
■   ■■■ admin/        # React admin panel
■■■ packages/
■   ■■■ ui/           # Componenti UI condivisi
■   ■■■ types/        # Tipi TypeScript condivisi
■   ■■■ config/       # Configurazioni ESLint, TS, ecc.
■   ■■■ utils/        # Utility condivise
■■■ pnpm-workspace.yaml
■■■ turbo.json
■■■ package.json

# pnpm-workspace.yaml
packages:
  - "apps/*"
  - "packages/*"

# Installazione con pnpm
pnpm install   # installa tutte le dipendenze con hard links condivisi

# Aggiunge dipendenza a un package specifico
pnpm add express --filter api

# Aggiunge dipendenza shared a tutti i package
pnpm add typescript -D -w

# Esegui script in tutti i package
pnpm --filter "*" build
pnpm --filter "api" dev

# turbo.json — pipeline Turborepo con cache
{
  "$schema": "https://turbo.build/schema.json",
  "pipeline": {
    "build": {
      "dependsOn": ["^build"],
      "outputs": ["dist/**", ".next/**"]
    },
    "test": {
      "dependsOn": ["^build"],
      "outputs": ["coverage/**"]
    },
    "dev": {
      "cache": false,
      "persistent": true
    },
    "lint": {}
  }
}

# Esegui build con cache remota (CI 10x più veloce)
turbo build --cache-dir=.turbo
```

26.2 ESLint 9 e configurazione moderna

■ JS | eslint.config.js — ESLint 9 flat config con TypeScript e plugin security

```
// eslint.config.js (flat config ESLint 9)
// npm install eslint @eslint/js typescript-eslint eslint-plugin-security

import js from '@eslint/js';
import typescript from 'typescript-eslint';
import security from 'eslint-plugin-security';

export default typescript.config(
  js.configs.recommended,
  ...typescript.configs.recommended,
  {
    plugins: { security },
    rules: {
      // TypeScript
      '@typescript-eslint/no-explicit-any': 'warn',
      '@typescript-eslint/no-unused-vars': ['error', { argsIgnorePattern: '^_' }],
      '@typescript-eslint/explicit-function-return-type': 'off',
      '@typescript-eslint/no-floating-promises': 'error',  // promise non gestite

      // Sicurezza (eslint-plugin-security)
      'security/detect-object-injection': 'warn',
      'security/detect-non-literal-regexp': 'warn',
      'security/detect-possible-timing-attacks': 'error',

      // Node.js
      'no-console': ['warn', { allow: ['warn', 'error'] }],
      'prefer-const': 'error',
      'no-var': 'error',
    },
  },
  {
    files: ['**/*.test.ts', '**/*.spec.ts'],
    rules: {
      '@typescript-eslint/no-explicit-any': 'off',
    },
  }
);
```

26.3 GitHub Actions CI/CD per Node.js

■ YAML | .github/workflows/ci.yml — Pipeline CI/CD completa con test, lint, Docker

```yaml
# .github/workflows/ci.yml — Pipeline CI/CD completa per Node.js

name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
  NODE_VERSION: "22"
  PNPM_VERSION: "9"

jobs:
  # ■■ QUALITY CHECK ■■■
```

```yaml
# ■■ BUILD E DEPLOY ■■■
```

27. NODE.JS PER IOT E AUTOMAZIONE INDUSTRIALE

Node.js è sorprendentemente adatto all'IoT e all'automazione industriale grazie alla sua natura event-driven, al basso footprint di memoria e all'eccellente supporto per comunicazioni asincrone. Questa sezione analizza i protocolli industriali più usati e pattern pratici per sistemi SCADA e PLC in ambito Node.js.

27.1 MQTT: il protocollo IoT per eccellenza

MQTT (Message Queuing Telemetry Transport) è il protocollo publish-subscribe più usato nell'IoT: leggerissimo (overhead minimo 2 byte), funziona su connessioni instabili con QoS configurabile, ideale per sensori con banda limitata.

■ JS | mqtt-client.js — gateway MQTT per raccolta dati IoT con InfluxDB

```
# npm install mqtt
// mqtt-client.js — client MQTT per raccolta dati sensori

const mqtt = require('mqtt');
const { InfluxDB, Point } = require('@influxdata/influxdb-client');

// Connessione al broker MQTT (es. Mosquitto, HiveMQ, AWS IoT)
const client = mqtt.connect('mqtt://localhost:1883', {
  clientId: `nodejs-gateway-${Date.now()}`,
  username: process.env.MQTT_USER,
  password: process.env.MQTT_PASS,
  keepalive: 60,
  reconnectPeriod: 5000,
  connectTimeout: 30000,
  clean: false,  // mantieni sessione persistente (QoS 1 e 2)
});

// InfluxDB per time-series data
const influx = new InfluxDB({
  url: process.env.INFLUX_URL,
  token: process.env.INFLUX_TOKEN,
});
const writeApi = influx.getWriteApi('org', 'bucket');

client.on('connect', () => {
  console.log('Connesso al broker MQTT');

  // Subscribe a topic gerarchico con wildcard
  client.subscribe([
    'impianto/+/temperatura',    // + = un livello qualsiasi
    'impianto/+/pressione',
    'impianto/#',                 // # = tutti i sottolivelli
  ], { qos: 1 }, (err) => {
    if (err) console.error('Subscription fallita:', err);
  });
});

// Elaborazione messaggi in tempo reale
client.on('message', (topic, payload) => {
  try {
    const [, deviceId, dataType] = topic.split('/');
    const value = parseFloat(payload.toString());

    if (isNaN(value)) {
      console.warn(`Valore non valido da ${topic}: ${payload}`);
      return;
    }

    // Salva su InfluxDB (time-series database)
    const point = new Point(dataType)
      .tag('device', deviceId)
      .tag('impianto', 'linea-1')
      .floatField('value', value)
      .timestamp(new Date());

    writeApi.writePoint(point);

    // Alert in tempo reale
    if (dataType === 'temperatura' && value > 85) {
      console.error(`ALERT: Temperatura critica su ${deviceId}: ${value}°C`);
      // Invia notifica (Telegram, email, SMS...)
    }

    console.log(`[${deviceId}] ${dataType}: ${value}`);
```

```
  } catch (err) {
    console.error('Errore elaborazione messaggio:', err);
  }
});

// Publish comando a dispositivo (es. apri valvola)
function inviaComando(deviceId, comando, valore) {
  const topic = `impianto/${deviceId}/comando`;
  const payload = JSON.stringify({ comando, valore, ts: Date.now() });
  client.publish(topic, payload, { qos: 2, retain: false });
  console.log(`Comando inviato a ${deviceId}: ${comando}=${valore}`);
}

client.on('error', (err) => console.error('MQTT error:', err));
client.on('reconnect', () => console.warn('MQTT riconnessione...'));
client.on('offline', () => console.warn('MQTT offline'));
```

27.2 Modbus TCP con Node.js

Modbus TCP è il protocollo di automazione industriale più diffuso al mondo, usato per comunicare con PLC, variatori di frequenza, sistemi SCADA e sensori industriali. La libreria modbus-serial supporta Modbus RTU (seriale), TCP e ASCII.

■ JS | modbus-client.js — lettura/scrittura PLC via Modbus TCP con polling

```
# npm install modbus-serial
// modbus-client.js — lettura registri da PLC Siemens/Allen-Bradley

const ModbusRTU = require('modbus-serial');

class ModbusGateway {
  constructor(ip, port = 502, slaveId = 1) {
    this.client = new ModbusRTU();
    this.ip = ip;
    this.port = port;
    this.slaveId = slaveId;
    this.connected = false;
  }

  async connect() {
    await this.client.connectTCP(this.ip, { port: this.port });
    this.client.setID(this.slaveId);
    this.connected = true;
    console.log(`Connesso a PLC ${this.ip}:${this.port}`);
  }

  // Leggi Holding Registers (FC03)
  async leggiRegistri(startAddr, count) {
    if (!this.connected) await this.connect();
    const data = await this.client.readHoldingRegisters(startAddr, count);
    return data.data;  // array di interi 16-bit
  }

  // Leggi Input Registers (FC04) - solo lettura
  async leggiInput(startAddr, count) {
    const data = await this.client.readInputRegisters(startAddr, count);
    return data.data;
  }

  // Scrivi Holding Register (FC06) - singolo
  async scriviRegistro(addr, valore) {
    await this.client.writeRegister(addr, valore);
  }

  // Scrivi Multiple Registers (FC16)
  async scriviRegistri(startAddr, valori) {
    await this.client.writeRegisters(startAddr, valori);
  }

  // Leggi Coil (bobina digitale, FC01)
  async leggiCoils(startAddr, count) {
    const data = await this.client.readCoils(startAddr, count);
    return data.data;  // array di boolean
  }

  // Polling ciclico con interval
  startPolling(registri, intervalMs = 1000) {
    return setInterval(async () => {
      try {
        const valori = await this.leggiRegistri(registri.start, registri.count);

        // Converti valori grezzi in grandezze fisiche
        const temperatura = valori[0] / 10;    // es. 235 = 23.5°C
        const pressione   = valori[1] / 100;   // es. 1013 = 10.13 bar
        const velocita    = valori[2];           // RPM diretto

        console.log({ temperatura, pressione, velocita });
        // emit evento o salva su DB...
```

```
      } catch (err) {
        console.error('Errore polling Modbus:', err.message);
        this.connected = false;  // riconnetterà al prossimo ciclo
      }
    }, intervalMs);
  }
}

// Uso
const plc = new ModbusGateway('192.168.1.100', 502, 1);
await plc.connect();

// Polling ogni 500ms dai registri 40001-40010 (0-based: 0-9)
plc.startPolling({ start: 0, count: 10 }, 500);

// Scrivi setpoint temperatura (registro 40020 = indice 19)
await plc.scriviRegistro(19, 750);  // 75.0°C
```

27.3 OPC-UA con Node.js

■ JS | opcua-client.js — lettura e subscription OPC-UA da PLC/SCADA

```
# npm install node-opcua
// opcua-client.js — client OPC-UA per sistemi SCADA/MES

const { OPCUAClient, AttributeIds, TimestampsToReturn } = require('node-opcua');

const endpointUrl = 'opc.tcp://192.168.1.100:4840';

async function readOPCUA() {
  const client = OPCUAClient.create({
    endpointMustExist: false,
    connectionStrategy: {
      initialDelay: 1000,
      maxRetry: 5,
    },
  });

  await client.connect(endpointUrl);
  console.log('OPC-UA connesso');

  const session = await client.createSession();

  // Leggi valore da nodo specifico
  const nodeId = 'ns=2;s=Channel1.Device1.Temperatura';
  const dataValue = await session.readVariableValue(nodeId);
  console.log('Temperatura:', dataValue.value.value, dataValue.value.dataType);

  // Subscription: notifiche su cambio valore
  const subscription = await session.createSubscription2({
    requestedPublishingInterval: 500,
    requestedMaxKeepAliveCount: 10,
    maxNotificationsPerPublish: 100,
    publishingEnabled: true,
    priority: 10,
  });

  const monitoredItem = await subscription.monitor({
    nodeId: nodeId,
    attributeId: AttributeIds.Value,
  }, { samplingInterval: 500, discardOldest: true, queueSize: 100 },
    TimestampsToReturn.Both);

  monitoredItem.on('changed', (dataValue) => {
    console.log(`[${dataValue.serverTimestamp.toISOString()}] Temp:
      ${dataValue.value.value}°C`);
  });

  // Mantieni connessione attiva (in produzione usa loop o daemon)
  await new Promise(resolve => setTimeout(resolve, 30000));
  await session.close();
  await client.disconnect();
}

readOPCUA().catch(console.error);
```

Fonti: npmjs.com/package/mqtt; influxdata.com/docs/influxdb/v2/api-guide/client-libraries/nodejs/; npmjs.com/package/modbus-serial; npmjs.com/package/node-opcua; mqtt.org; opcfoundation.org/about/opc-technologies/opc-ua/

28. BEST PRACTICE E ANTI-PATTERN

Questa sezione raccoglie le best practice più importanti e gli anti-pattern da evitare, sintetizzando l'esperienza di migliaia di progetti Node.js in produzione.

28.1 I 10 anti-pattern più pericolosi

Anti-pattern	Problema	Soluzione
Callback hell	Codice annidato illeggibile; errori difficili da gestire	async/await con try/catch; Promise.all per parallelismo
Bloccare l Event Loop	setTimeout/fs.sync in hot path bloccano tutte le richieste	Usare sempre versioni async; Worker Thread per CPU-intensive
Variabili globali mutable	Race condition; stato condiviso tra richieste diverse	Closures; dependency injection; request-scoped state
require() in loop / hot path	Rallentamento per lookup nel modulo cache	require() sempre top-level; caching esplicito risultati
Error swallowing	catch(err) {} silenzioso; bug invisibili in produzione	Sempre loggare o rilanciare; error handler centralizzato
Nessun graceful shutdown	Dati persi; connessioni DB non chiuse su SIGTERM	process.on(SIGTERM) con server.close() e db.disconnect()
Segreti nel codice	API key nel repository; data breach	process.env + .env; Vault; AWS Secrets Manager
npm install senza lock file	Build non riproducibili; vulnerabilita' non viste	Committare package-lock.json; usare npm ci in CI
Nessun rate limiting	DoS, brute force, abuso API	express-rate-limit; Cloudflare; NGINX limit_req
Logica business nel route handler	Codice non testabile; duplicazione	Service layer + Repository pattern; controller snelli

28.2 Checklist produzione Node.js

Area	Check	Priorità
Sicurezza	npm audit senza CVE critiche; helmet attivo; CORS configurato	CRITICA
Autenticazione	JWT con scadenza + refresh; password con bcrypt/argon2	CRITICA
Variabili ambiente	Nessun segreto nel codice; validazione ENV all avvio	CRITICA
Error handling	Error handler centralizzato; nessuno stack trace esposto	CRITICA
Logging	Logging strutturato (pino/winston); nessun dato sensibile nei log	ALTA
Rate limiting	Limite richieste su tutti gli endpoint pubblici	ALTA
Monitoring	Health check /health; metriche Prometheus; alerting	ALTA
Database	Pool connessioni configurato; indici sulle query frequenti	ALTA
Graceful shutdown	SIGTERM gestito; connessioni chiuse pulitamente	ALTA
Docker	Multi-stage build; utente non-root; healthcheck	MEDIA
CI/CD	Test automatici; lint; coverage; deploy automatizzato	MEDIA
Caching	Redis per dati frequenti; cache headers HTTP corretti	MEDIA
Input validation	Validazione su ogni endpoint; sanitizzazione dati	MEDIA
Dipendenze	Dependabot attivo; revisione periodica dipendenze	MEDIA

28.3 Node.js in produzione: architettura di riferimento

■ BASH | Architettura di riferimento Node.js per sistema production-grade

```
# Architettura production-grade completa per applicazione Node.js

Internet
   |
[Cloudflare CDN + DDoS protection + WAF]
   |
[Load Balancer: Nginx / AWS ALB]
   |         |
[Node.js]  [Node.js]   <-- Cluster PM2 o Kubernetes pods
   |         |
[Redis Cluster]        <-- Sessioni, cache, job queue
   |
[Database Layer]
■■■ MongoDB Atlas      <-- Dati documentali (replica set)
■■■ PostgreSQL         <-- Dati relazionali (RDS Multi-AZ)
■■■ InfluxDB           <-- Time-series (IoT, metrics)
   |
[Message Queue]
■■■ BullMQ + Redis     <-- Job asincroni in-process
■■■ Apache Kafka       <-- Event streaming inter-servizi
   |
[Monitoring Stack]
■■■ Prometheus + Grafana  <-- Metriche
■■■ ELK Stack             <-- Log aggregation
■■■ Jaeger                <-- Distributed tracing

# File struttura src/ consigliata per API production
src/
■■■ config/           # Configurazione ambiente
■■■ middleware/       # Auth, logging, rate limit, error
■■■ routes/           # Definizioni rotte (thin)
■■■ controllers/      # Orchestrazione richieste (thin)
■■■ services/         # Logica business (fat)
■■■ repositories/     # Accesso dati (fat)
■■■ models/           # Schema dati
■■■ events/           # Event bus, handler eventi
■■■ jobs/             # Background job BullMQ
■■■ utils/            # Helper, logger, crypto
■■■ index.js          # Entry point + graceful shutdown
```

✓ **BEST PRACTICE** Il repository **Node.js Best Practices** di Yoni Goldberger (github.com/goldbergyoni/nodebestpractices) raccoglie oltre 80 best practice verificate dalla community. Con 100k+ stelle su GitHub, è la risorsa più completa per sviluppo Node.js professionale.

Fonti: github.com/goldbergyoni/nodebestpractices; twelve-factor.net (metodologia 12-factor app); nodejs.org/en/docs/guides/; snyk.io/blog/nodejs-security-best-practices/; pm2.io/docs/runtime/best-practices/

29. SICUREZZA AVANZATA: AUTH, SESSIONI E ATTACCHI WEB

La sicurezza delle applicazioni web richiede una comprensione profonda degli attacchi più comuni e delle contromisure. Questa sezione approfondisce JWT avanzato, gestione sessioni, CSRF, XSS e SQL injection con esempi pratici di attacco e difesa.

29.1 JWT avanzato: best practice e vulnerabilità

Vulnerabilità JWT	Descrizione	Contromisura
Algorithm confusion (alg:none)	Attaccante imposta alg='none' per bypassare firma	Whitelist algoritmi; usa jose con algoritmo fisso (RS256/ES256)
HS256 con secret debole	Secret corto o prevedibile; attacchi bruteforce	Secret >= 256 bit random; preferire RS256 (chiave asimmetrica)
Token senza scadenza	Token rubato valido per sempre	exp breve (15min); refresh token con rotazione
Payload con dati sensibili	JWT decodificabile senza chiave (base64)	MAI mettere password/card in JWT; solo ID e ruolo
Token non invalidabile	Logout non revoca token esistente	Token blacklist in Redis; rotazione refresh token
JWKS endpoint non verificato	Bypass firma con JWKS esterno malevolo	Fissa issuer e audience; whitelist JWKS URL

■ JS | auth-secure.js — JWT ES256, refresh token con rotazione, cookie httpOnly

```
// auth-secure.js — implementazione JWT sicura con refresh token
const jose = require('jose');
const crypto = require('crypto');

const ACCESS_EXPIRY  = '15m';    // breve: token di accesso
const REFRESH_EXPIRY = '7d';     // lungo: token di rinnovo

// Usa ES256 (asimmetrico) invece di HS256 per maggiore sicurezza
async function generaChiavi() {
  return jose.generateKeyPair('ES256');
}

// Genera access token (breve durata, piccolo payload)
async function generaAccessToken(user, privateKey) {
  return new jose.SignJWT({
    sub: user.id,
    role: user.role,
    email: user.email,
  })
    .setProtectedHeader({ alg: 'ES256' })
    .setIssuedAt()
    .setIssuer('https://mia-api.com')
    .setAudience('https://mia-api.com/api')
    .setExpirationTime(ACCESS_EXPIRY)
    .sign(privateKey);
}

// Genera refresh token (opaco, salvato nel DB)
function generaRefreshToken() {
  return crypto.randomBytes(64).toString('hex');
}

// Endpoint login completo
app.post('/api/auth/login', async (req, res, next) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email }).select('+password');
    if (!user || !(await bcrypt.compare(password, user.password))) {
      await bcrypt.hash('dummy', 12);  // timing attack prevention
      return res.status(401).json({ error: 'Credenziali non valide' });
    }

    const accessToken = await generaAccessToken(user, privateKey);
    const refreshToken = generaRefreshToken();

    // Salva refresh token nel DB (hash per sicurezza)
    const refreshHash = crypto.createHash('sha256')
      .update(refreshToken).digest('hex');
    await RefreshToken.create({
      userId: user.id,
      tokenHash: refreshHash,
      expiresAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000),
      userAgent: req.headers['user-agent'],
      ip: req.ip,
    });

    // Refresh token in cookie httpOnly (non accessibile da JS)
    res.cookie('refreshToken', refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict',
      maxAge: 7 * 24 * 60 * 60 * 1000,
    });
```

```
    res.json({ accessToken });  // access token nel body
  } catch (err) { next(err); }
});

// Endpoint refresh
app.post('/api/auth/refresh', async (req, res, next) => {
  try {
    const refreshToken = req.cookies.refreshToken;
    if (!refreshToken) return res.status(401).json({ error: 'No refresh token' });

    const hash = crypto.createHash('sha256').update(refreshToken).digest('hex');
    const stored = await RefreshToken.findOne({
      tokenHash: hash,
      expiresAt: { $gt: new Date() },
    });

    if (!stored) return res.status(401).json({ error: 'Refresh token non valido' });

    // ROTAZIONE: revoca vecchio token, emetti nuovo
    await RefreshToken.deleteOne({ _id: stored._id });
    const user = await User.findById(stored.userId);
    const newAccessToken = await generaAccessToken(user, privateKey);
    const newRefreshToken = generaRefreshToken();

    const newHash = crypto.createHash('sha256').update(newRefreshToken).digest('hex');
    await RefreshToken.create({ userId: user.id, tokenHash: newHash,
      expiresAt: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000) });

    res.cookie('refreshToken', newRefreshToken, {
      httpOnly: true, secure: true, sameSite: 'strict',
      maxAge: 7 * 24 * 60 * 60 * 1000,
    });
    res.json({ accessToken: newAccessToken });
  } catch (err) { next(err); }
});
```

29.2 CSRF: Cross-Site Request Forgery

■ JS | csrf.js — protezione CSRF con doppio cookie e SameSite=Strict

```
// CSRF: attaccante fa eseguire richieste autenticate a vittima

// ■■ PROTEZIONE CSRF con token doppio cookie ■■■■■■■■■■■■■■■■■■■■■■■■■■
const { randomBytes } = require('crypto');

function generaCSRFToken() {
  return randomBytes(32).toString('hex');
}

// Middleware: imposta token CSRF
app.use((req, res, next) => {
  if (!req.cookies.csrfToken) {
    const token = generaCSRFToken();
    res.cookie('csrfToken', token, {
      httpOnly: false,  // JS deve poterlo leggere
      secure: true,
      sameSite: 'strict',
    });
    req.csrfToken = token;
  } else {
    req.csrfToken = req.cookies.csrfToken;
  }
  next();
});

// Middleware: verifica token per metodi mutanti
function verificaCSRF(req, res, next) {
  if (['GET', 'HEAD', 'OPTIONS'].includes(req.method)) return next();

  const cookieToken = req.cookies.csrfToken;
  const headerToken = req.headers['x-csrf-token'];

  if (!cookieToken || !headerToken || cookieToken !== headerToken) {
    return res.status(403).json({ error: 'CSRF token non valido' });
  }
  next();
}

app.use(verificaCSRF);

// Frontend (React/Vue): invia token nell header
// axios.defaults.headers.common['X-CSRF-Token'] =
//   document.cookie.match(/csrfToken=([^;]+)/)?.[1];

// ■■ PROTEZIONE CSRF con SameSite=Strict (moderna) ■■■■■■■■■■■■■■■■■■
// Cookie con SameSite=Strict non vengono inviati da siti terzi
// Combina: SameSite + CSRF token per protezione massima
// - SameSite=Strict: previene la maggior parte dei CSRF
// - CSRF token: protezione aggiuntiva per browser meno recenti
```

29.3 XSS: Cross-Site Scripting in API Node.js

■ JS | xss.js — DOMPurify, sanitizzazione testo, CSP stretta, HSTS

```
// XSS in contesto API Node.js (anche se non rende HTML, il JSON viene
// consumato da frontend che potrebbe usarlo in innerHTML)

// ■■ SANITIZZAZIONE OUTPUT ■■■
```

30. PERFORMANCE TUNING E MEMORY MANAGEMENT

Le performance di un'applicazione Node.js dipendono da diversi fattori: efficienza dell'event loop, gestione della memoria, utilizzo del garbage collector V8 e ottimizzazione delle query ai database. Questa sezione fornisce strumenti e tecniche concrete.

30.1 Analisi memory leak con heap snapshot

■ JS | memory-analysis.js — heap snapshot, anti-pattern memory leak, LRU cache

```
// memory-analysis.js — tecniche per trovare e correggere memory leak

// ■■ RILEVAMENTO MEMORY LEAK ■■■
```

```
  cache.set(req.url, { data: res.body, ts: Date.now() });
  next();
});

// CORRETTO: usa Map con LRU eviction
const LRUCache = require('lru-cache');
const lruCache = new LRUCache({
  max: 500,          // max 500 entry
  maxSize: 50 * 1024 * 1024,  // max 50MB totali
  sizeCalculation: (value) => JSON.stringify(value).length,
  ttl: 1000 * 60,   // 1 minuto
});
```

30.2 V8 optimization: JIT e Deoptimization

■ JS | v8-optimization.js — JIT, deopt, benchmarking e flag V8 utili

```
// V8 compila JS in codice macchina con JIT (Just-In-Time)
// Alcune pratiche causano "deoptimizzazione" (ritorno a interpretato)

// ■■ PATTERN CHE CAUSANO DEOPT ■■■
```

■ JS | db-optimization.js — explain(), indici, N+1 con DataLoader, EXPLAIN ANALYZE

```
// Query optimization: MongoDB e PostgreSQL

// ■■ MONGODB: explain() e analisi performance ■■■■■■■■■■■■■■■■■■■■■■■■
// Analizza esecuzione query
const explain = await User.find({ role: 'admin', createdAt: { $gt: date } })
  .explain('executionStats');

console.log({
  nScanned:   explain.executionStats.totalDocsExamined,
  nReturned:  explain.executionStats.nReturned,
  executionMs:explain.executionStats.executionTimeMillis,
  indexUsed:  explain.queryPlanner.winningPlan.inputStage.indexName,
});

// Se totalDocsExamined >> nReturned: manca un indice!

// Crea indice composto (ordine campo: equality, range, sort)
await User.collection.createIndex(
  { role: 1, createdAt: -1 },  // role=equality, createdAt=sort
  { background: true, name: 'role_created_idx' }
);

// Indice text per ricerca full-text
await Post.collection.createIndex(
  { titolo: 'text', contenuto: 'text' },
  { weights: { titolo: 10, contenuto: 1 } }
);
const results = await Post.find({ $text: { $search: 'Node.js performance' } });

// ■■ POSTGRESQL con Prisma: EXPLAIN ANALYZE ■■■■■■■■■■■■■■■■■■■■■■■■■■
// Esegui query raw per analisi
const explain2 = await prisma.$queryRaw`
  EXPLAIN ANALYZE
  SELECT u.id, u.nome, COUNT(p.id) as post_count
  FROM "User" u
  LEFT JOIN "Post" p ON p."autoreId" = u.id
  WHERE u.role = 'ADMIN'
  GROUP BY u.id
  ORDER BY post_count DESC
  LIMIT 20
`;

// Indici PostgreSQL consigliati
await prisma.$executeRaw`CREATE INDEX CONCURRENTLY IF NOT EXISTS
  idx_user_role ON "User"(role)`;
await prisma.$executeRaw`CREATE INDEX CONCURRENTLY IF NOT EXISTS
  idx_post_autore_created ON "Post"("autoreId", "createdAt" DESC)`;

// ■■ N+1 PREVENTION con DataLoader ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
// Senza DataLoader: 1 query per user + N query per post = N+1
// Con DataLoader: 1 query + 1 query batch = 2 query totali

const userLoader = new DataLoader(async (ids) => {
  const users = await prisma.user.findMany({ where: { id: { in: ids } } });
  const map = new Map(users.map(u => [u.id, u]));
  return ids.map(id => map.get(id));
});

// Ogni resolver chiama userLoader.load(id) invece di db.findById(id)
// DataLoader raggruppa automaticamente le chiamate in un batch
```

Fonti: v8.dev/docs/profile (V8 CPU profiler); nodejs.org/en/docs/guides/simple-profiling; mongodb.com/docs/manual/reference/explain-results/; postgresql.org/docs/current/using-explain.html; npmjs.com/package/lru-cache; clinicjs.org (clinic doctor/flame per profiling Node.js)

31. CASO PRATICO: SISTEMA E-COMMERCE CON NODE.JS

Questa sezione presenta un caso pratico completo: l'architettura e l'implementazione di un sistema e-commerce con Node.js, integrando i concetti di tutti i capitoli precedenti. Il sistema include: catalogo prodotti, carrello, ordini, pagamenti, notifiche real-time e un pannello admin.

31.1 Architettura del sistema

Servizio	Tecnologia	Responsabilità
API Gateway	Node.js + Express + nginx	Routing, auth, rate limiting, logging
Product Service	Node.js + MongoDB + Redis	Catalogo, varianti, stock, ricerca
Order Service	Node.js + PostgreSQL + Prisma	Ordini, stato, storico
Payment Service	Node.js + Stripe SDK	Checkout, rimborsi, webhook Stripe
Notification Service	Node.js + Socket.IO + nodemailer	Email, push, WebSocket real-time
Image Service	Node.js + Sharp + S3	Upload, resize, CDN
Admin Dashboard	Next.js + React	Backoffice prodotti, ordini, analytics
Background Jobs	Node.js + BullMQ + Redis	Email, report, sincronizzazione stock
Search	Elasticsearch + Node.js client	Full-text search prodotti

31.2 Gestione ordini: saga pattern

■ JS | order-saga.js — Saga pattern con BullMQ per transazioni distribuite

```
// order-saga.js — Gestione transazioni distribuite con Saga pattern
// Problema: creare un ordine richiede azioni su servizi multipli
// Saga: sequenza di step locali con compensazione su fallimento

const { Queue, Worker } = require('bullmq');
const redis = { connection: { host: 'localhost', port: 6379 } };

// Code per ogni step della saga
const sagaQueue = new Queue('order-saga', redis);
const paymentQueue = new Queue('payments', redis);
const inventoryQueue = new Queue('inventory', redis);
const notifyQueue = new Queue('notifications', redis);

// Orchestratore saga: gestisce il flusso
class OrderSaga {
  constructor(orderId, orderData) {
    this.orderId = orderId;
    this.orderData = orderData;
    this.steps = [];
  }

  async execute() {
    try {
      // Step 1: Riserva stock
      await this.reserveStock();

      // Step 2: Processa pagamento
      await this.processPayment();

      // Step 3: Conferma ordine
      await this.confirmOrder();

      // Step 4: Invia notifiche
      await this.sendNotifications();

      console.log(`Ordine ${this.orderId} completato`);
    } catch (err) {
      console.error(`Saga fallita per ordine ${this.orderId}:`, err.message);
      // Compensazione: annulla tutti gli step completati
      await this.compensate();
    }
  }

  async reserveStock() {
    const job = await inventoryQueue.add('reserve', {
      orderId: this.orderId,
      items: this.orderData.items,
    });
    await job.waitUntilFinished(inventoryQueue.events);
    this.steps.push({ name: 'reserveStock', jobId: job.id });
  }

  async processPayment() {
    const job = await paymentQueue.add('charge', {
      orderId: this.orderId,
      amount: this.orderData.total,
      customerId: this.orderData.customerId,
      paymentMethodId: this.orderData.paymentMethodId,
    });
    const result = await job.waitUntilFinished(paymentQueue.events);
    this.steps.push({ name: 'processPayment', jobId: job.id, chargeId: result.chargeId
      });
  }
```

```
  async confirmOrder() {
    await Order.findByIdAndUpdate(this.orderId, {
      status: 'confirmed',
      confirmedAt: new Date(),
    });
    this.steps.push({ name: 'confirmOrder' });
  }

  async sendNotifications() {
    await notifyQueue.add('order-confirmed', {
      orderId: this.orderId,
      customerId: this.orderData.customerId,
      email: this.orderData.customerEmail,
    });
  }

  async compensate() {
    // Annulla in ordine inverso
    for (const step of [...this.steps].reverse()) {
      try {
        if (step.name === 'processPayment') {
          // Rimborso Stripe
          await stripe.refunds.create({ charge: step.chargeId });
        } else if (step.name === 'reserveStock') {
          // Rilascia riserva stock
          await inventoryQueue.add('release', { orderId: this.orderId });
        } else if (step.name === 'confirmOrder') {
          await Order.findByIdAndUpdate(this.orderId, { status: 'cancelled' });
        }
      } catch (compErr) {
        console.error(`Compensazione ${step.name} fallita:`, compErr);
      }
    }
    await notifyQueue.add('order-failed', {
      orderId: this.orderId,
      customerId: this.orderData.customerId,
    });
  }
}

// Trigger dalla route ordine
app.post('/api/orders', requireAuth, async (req, res, next) => {
  try {
    const order = await Order.create({
      ...req.body,
      customerId: req.user.sub,
      status: 'pending',
    });
    const saga = new OrderSaga(order.id, { ...req.body, customerId: req.user.sub });
    saga.execute().catch(console.error);  // esegui asincrono
    res.status(202).json({ orderId: order.id, status: 'processing' });
  } catch (err) { next(err); }
});
```

31.3 Integrazione Stripe e webhook

■ JS | payment-service.js — Stripe: checkout session, webhook, gestione eventi

```
// npm install stripe
// payment-service.js — integrazione Stripe completa

const Stripe = require('stripe');
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY, {
  apiVersion: '2024-04-10',
});

// ■■ CHECKOUT SESSION (redirect-based) ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
app.post('/api/payments/checkout', requireAuth, async (req, res, next) => {
  try {
    const { orderId } = req.body;
    const order = await Order.findById(orderId).populate('items.product');

    const session = await stripe.checkout.sessions.create({
      mode: 'payment',
      customer_email: req.user.email,
      line_items: order.items.map(item => ({
        price_data: {
          currency: 'eur',
          product_data: {
            name: item.product.nome,
            images: [item.product.immagine],
          },
          unit_amount: Math.round(item.product.prezzo * 100),  // centesimi
        },
        quantity: item.quantita,
      })),
      metadata: { orderId: orderId, userId: req.user.sub },
      success_url: `${process.env.CLIENT_URL}/ordini/${orderId}?success=1`,
      cancel_url:  `${process.env.CLIENT_URL}/carrello?cancelled=1`,
    });

    res.json({ checkoutUrl: session.url });
  } catch (err) { next(err); }
});

// ■■ WEBHOOK STRIPE (notifica eventi asincroni) ■■■■■■■■■■■■■■■■■■■■■■■
app.post('/webhooks/stripe',
  express.raw({ type: 'application/json' }),  // raw body necessario!
  async (req, res) => {
    const sig = req.headers['stripe-signature'];
    let event;

    try {
      event = stripe.webhooks.constructEvent(
        req.body,
        sig,
        process.env.STRIPE_WEBHOOK_SECRET
      );
    } catch (err) {
      console.error('Webhook firma non valida:', err.message);
      return res.status(400).send(`Webhook Error: ${err.message}`);
    }

    switch (event.type) {
      case 'checkout.session.completed': {
        const session = event.data.object;
        const { orderId } = session.metadata;
        await Order.findByIdAndUpdate(orderId, {
          status: 'paid',
          stripeSessionId: session.id,
          paidAt: new Date(),
```

```
        });
        await notifyQueue.add('order-paid', { orderId });
        break;
      }
      case 'charge.dispute.created': {
        console.warn('DISPUTE:', event.data.object.id);
        break;
      }
    }

    res.json({ received: true });
  }
);
```

Fonti: stripe.com/docs/api/node; stripe.com/docs/webhooks; microservices.io/patterns/data/saga.html (Saga pattern); npmjs.com/package/bullmq; npmjs.com/package/stripe

32. APPENDICE TECNICA ESTESA

Questa appendice raccoglie snippet di codice pronti all'uso, tabelle di riferimento complete e pattern frequentemente usati che non trovano spazio nei capitoli principali.

32.1 Middleware Express riutilizzabili

■ JS | middleware/index.js — 6 middleware pronti: requestId, timeout, rateLimit, cache

```
// middleware/index.js — raccolta middleware pronti all uso

const crypto = require('crypto');
const { RateLimiterRedis } = require('rate-limiter-flexible');
const redis = require('ioredis');

// ■■ 1. REQUEST ID ■■■
```

```
  if (req.method !== 'GET') return next();

  const key = `cache:${req.originalUrl}`;
  const cached = await redisClient.get(key);
  if (cached) {
    res.setHeader('X-Cache', 'HIT');
    return res.json(JSON.parse(cached));
  }

  const originalJson = res.json.bind(res);
  res.json = (data) => {
    redisClient.setex(key, ttl, JSON.stringify(data)).catch(() => {});
    res.setHeader('X-Cache', 'MISS');
    return originalJson(data);
  };
  next();
};

// ■■ 6. COMPRESSION ■■■
```

■ JS | utils/helpers.js — retry, sleep, debounce, throttle, chunk, token, hash

```
// utils/helpers.js — utility functions pronte all uso

const crypto = require('crypto');

// ■■ RETRY con backoff esponenziale ■■■
```

```
// ■■ DEEP MERGE ■■■
```

32.3 Tabella completa: status code HTTP

Codice	Nome	Quando usarlo in una API REST
200 OK	Success	GET riuscito; risposta con dati
201 Created	Created	POST che crea una risorsa; includi Location header
202 Accepted	Accepted	Richiesta accettata ma elaborazione asincrona (saga, job)
204 No Content	No Content	DELETE o PUT riuscito senza corpo risposta
206 Partial Content	Partial Content	Range request (streaming file, paginazione stream)
301 Moved Permanently	Moved	Redirect permanente (cambiamento URL API)
304 Not Modified	Not Modified	ETag/Cache-Control: risorsa non cambiata
400 Bad Request	Bad Request	Dati input non validi; includi dettagli errore
401 Unauthorized	Unauthorized	Token mancante o non valido; indica come autenticarsi
403 Forbidden	Forbidden	Autenticato ma non autorizzato; ruolo insufficiente
404 Not Found	Not Found	Risorsa non esistente; non usare per errori auth
405 Method Not Allowed	Method NA	Metodo HTTP non supportato su quella route
409 Conflict	Conflict	Email duplicata; versione conflittuale (ottimistic locking)
410 Gone	Gone	Risorsa rimossa permanentemente (diverso da 404)
422 Unprocessable Entity	Unprocessable	Validazione fallita con dettagli specifici
429 Too Many Requests	Rate Limited	Rate limiting; includi Retry-After header
500 Internal Server Error	Server Error	Errore non gestito; MAI esporre stack trace
502 Bad Gateway	Bad Gateway	Servizio upstream non raggiungibile
503 Service Unavailable	Unavailable	Manutenzione programmata; includi Retry-After
504 Gateway Timeout	Timeout	Servizio upstream troppo lento

32.4 Configurazioni .env complete per ogni ambiente

■ BASH | .env.example — configurazione completa per ogni aspetto dell'applicazione

```
# .env.example — template completo per applicazione Node.js production-ready
# Copia in .env e compila con valori reali
# MAI committare .env con valori reali

# ■■ APPLICAZIONE ■■■
```

```
PROMETHEUS_PORT=9090

# ■■ SICUREZZA ■■■
```

■ JSON | package.json scripts — tutti gli script npm necessari in un progetto serio

```
// package.json — scripts completi per ogni esigenza
{
  "scripts": {
    "start":         "node src/index.js",
    "start:prod":    "NODE_ENV=production node -r ./src/tracing.js src/index.js",
    "dev":           "nodemon src/index.js",
    "dev:debug":     "nodemon --inspect-brk src/index.js",
    "build":         "tsc --project tsconfig.json",
    "build:watch":   "tsc --watch",

    "test":          "jest --coverage",
    "test:watch":    "jest --watch",
    "test:e2e":      "playwright test",
    "test:load":     "k6 run tests/load/load-test.js",
    "test:ci":       "jest --coverage --ci --forceExit",

    "lint":          "eslint src/ tests/",
    "lint:fix":      "eslint src/ tests/ --fix",
    "typecheck":     "tsc --noEmit",
    "format":        "prettier --write src/ tests/",
    "format:check":  "prettier --check src/ tests/",

    "db:migrate":    "prisma migrate deploy",
    "db:migrate:dev":"prisma migrate dev",
    "db:seed":       "node src/db/seed.js",
    "db:reset":      "prisma migrate reset",
    "db:studio":     "prisma studio",
    "db:generate":   "prisma generate",

    "docker:build":  "docker build -t mia-api:latest .",
    "docker:run":    "docker-compose up -d",
    "docker:down":   "docker-compose down",
    "docker:logs":   "docker-compose logs -f api",

    "security:audit":"npm audit --audit-level=moderate",
    "security:scan": "npx snyk test",

    "docs:generate": "jsdoc src/ -r -d docs/",
    "changelog":     "conventional-changelog -p angular -i CHANGELOG.md -s",

    "prepare":       "husky install",
    "pre-commit":    "lint-staged",
    "postinstall":   "prisma generate"
  },
  "lint-staged": {
    "src/**/*.{js,ts}": ["eslint --fix", "prettier --write"],
    "src/**/*.{json,md}": ["prettier --write"]
  }
}
```

32.6 Tabella performance: confronto soluzioni

Scenario	Soluzione A	Soluzione B	Raccomandazione
HTTP framework	Express (30k req/s)	Fastify (75k req/s)	Fastify per API ad alte perf; Express per rapidita' sviluppo
Logger	console.log (lento)	pino (800k log/s)	Pino sempre in produzione
ORM SQL	Sequelize (lento, maturo)	Prisma (moderno, type-safe)	Prisma per nuovi progetti
Job queue	node-cron (semplice)	BullMQ + Redis (robusto)	BullMQ per job critici con retry
Caching	node-cache (in-memory)	Redis (distribuito)	Redis per cluster; node-cache per single instance
Validazione	joi (API fluente)	zod (TypeScript-first)	Zod con TypeScript; Joi senza TS
Test runner	Jest (lento startup)	Vitest (veloce, ESM)	Vitest per nuovi progetti; Jest per legacy
HTTP client	axios (popolare)	fetch nativo (Node.js 22)	fetch nativo se Node.js 22+; axios per browser compat
Auth	passport.js (molte strategie)	jose (minimalista JWT)	jose per JWT puro; passport per OAuth complesso
Password	bcrypt (C++, veloce)	argon2 (piu' sicuro)	argon2 per nuovi progetti (vincitore PHC)

32.7 Comandi Git indispensabili per il workflow Node.js

■ BASH | Git workflow completo per Node.js: branching, conventional commits, alias

```
# Git workflow professionale per progetti Node.js

# ■■ SETUP PROGETTO ■■■
```

32.8 Regex patterns più utili in Node.js

Uso	Pattern	Note
Email	/^[^\s@]+@[^\s@]+\.[^\s@]+$/	Validazione base; usa librerie per RFC 5322 completo
URL HTTP/HTTPS	/^https?:\/\/.+/	Validazione URL con schema
UUID v4	/^[0-9a-f]{8}-[0-9a-f]{4}-4[0-9a-f]{3}-[89ab][0-9a-f]{3}-[0-9a-f]{12}$/i	UUID v4
Password forte	/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&]).{8,}$/	Maiusc+minusc+cifra+speciale
Solo alfanumerico	/^[a-zA-Z0-9]+$/	Slug, username, ID
Slug URL	/^[a-z0-9]+(?:-[a-z0-9]+)*$/	URL-friendly slug
IPv4	/^(\d{1,3}\.){3}\d{1,3}$/	Indirizzo IPv4 (validazione superficiale)
Data ISO 8601	/^\d{4}-\d{2}-\d{2}(T\d{2}:\d{2}:\d{2})?/	Data con ora opzionale
Numero di telefono IT	/^(\+39|0039)?[\d\s\-]{9,13}$/	Telefono italiano
Codice fiscale IT	/^[A-Z]{6}\d{2}[A-Z]\d{2}[A-Z]\d{3}[A-Z]$/i	CF italiano

Fonti: Tutte le fonti di questa appendice sono documentazione ufficiale o risorse di riferimento citate nei capitoli precedenti. npmjs.com; nodejs.org/api/; developer.mozilla.org; docs.stripe.com; grafana.com; owasp.org

33. OAUTH 2.0 E OPENID CONNECT

OAuth 2.0 è il protocollo standard per l'autorizzazione delegata: permette a un'applicazione di accedere a risorse di un utente su un altro sistema, senza che l'utente condivida le proprie credenziali. OpenID Connect (OIDC) estende OAuth 2.0 con autenticazione e identità dell'utente.

33.1 Flussi OAuth 2.0 principali

Flusso	Uso	Come funziona
Authorization Code + PKCE	Web app, SPA, mobile	Redirect a provider, code scambiato con token. PKCE evita intercettazione del code.
Client Credentials	Machine-to-machine (API)	Client ID + Secret scambiati direttamente per access token. Nessun utente coinvolto.
Device Code	TV, CLI, IoT	Device mostra codice, utente autorizza su altro device (telefono)
Implicit (DEPRECATO)	SPA legacy	Token nel redirect URL. Non usare: vulnerabile a token leakage.
Resource Owner Password (DEPRECATO)	Legacy	Username/password diretti al server OAuth. Non usare: espone credenziali.

33.2 Authorization Code con PKCE — implementazione

■ JS | oauth-server.js — Google OAuth 2.0 con PKCE, sessioni Redis, logout OIDC

```
// npm install openid-client  (libreria OAuth2/OIDC certificata)
// oauth-server.js — server Node.js con Google OAuth

const { Issuer, generators } = require('openid-client');
const express = require('express');
const session = require('express-session');
const RedisStore = require('connect-redis').default;
const { createClient } = require('redis');

const app = express();

// Sessione con Redis
const redisClient = createClient({ url: process.env.REDIS_URL });
await redisClient.connect();
app.use(session({
  store: new RedisStore({ client: redisClient }),
  secret: process.env.SESSION_SECRET,
  resave: false,
  saveUninitialized: false,
  cookie: {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'lax',
    maxAge: 7 * 24 * 60 * 60 * 1000,
  },
}));

// Scopri configurazione da provider (discovery document)
const googleIssuer = await Issuer.discover('https://accounts.google.com');

const client = new googleIssuer.Client({
  client_id:     process.env.GOOGLE_CLIENT_ID,
  client_secret: process.env.GOOGLE_CLIENT_SECRET,
  redirect_uris: [`${process.env.BASE_URL}/auth/callback`],
  response_types: ['code'],
});

// Step 1: Redirect a Google con PKCE
app.get('/auth/login', (req, res) => {
  const codeVerifier  = generators.codeVerifier();
  const codeChallenge = generators.codeChallenge(codeVerifier);
  const state         = generators.state();

  // Salva in sessione per verifica al callback
  req.session.codeVerifier = codeVerifier;
  req.session.state = state;
  req.session.returnTo = req.query.returnTo || '/dashboard';

  const authUrl = client.authorizationUrl({
    scope: 'openid email profile',
    code_challenge: codeChallenge,
    code_challenge_method: 'S256',
    state,
  });

  res.redirect(authUrl);
});

// Step 2: Callback da Google
app.get('/auth/callback', async (req, res, next) => {
  try {
    const params = client.callbackParams(req);

    // Verifica state e scambia code con token
```

```
    const tokenSet = await client.callback(
      `${process.env.BASE_URL}/auth/callback`,
      params,
      {
        code_verifier: req.session.codeVerifier,
        state: req.session.state,
      }
    );

    // Ottieni dati utente (userinfo endpoint)
    const userInfo = await client.userinfo(tokenSet.access_token);

    // Crea o aggiorna utente nel DB
    const user = await User.findOneAndUpdate(
      { email: userInfo.email },
      {
        nome: userInfo.name,
        avatar: userInfo.picture,
        googleId: userInfo.sub,
        emailVerified: userInfo.email_verified,
      },
      { upsert: true, new: true, runValidators: true }
    );

    // Inizia sessione autenticata
    delete req.session.codeVerifier;
    delete req.session.state;
    req.session.userId = user.id;

    const returnTo = req.session.returnTo || '/dashboard';
    delete req.session.returnTo;
    res.redirect(returnTo);

  } catch (err) {
    next(err);
  }
});

// Logout
app.get('/auth/logout', async (req, res) => {
  const endSessionUrl = client.endSessionUrl({
    id_token_hint: req.session.idToken,
    post_logout_redirect_uri: `${process.env.BASE_URL}/`,
  });
  req.session.destroy();
  res.redirect(endSessionUrl);
});

// Middleware protezione route
const requireAuth = (req, res, next) => {
  if (!req.session.userId) {
    req.session.returnTo = req.originalUrl;
    return res.redirect('/auth/login');
  }
  next();
};

app.get('/dashboard', requireAuth, async (req, res) => {
  const user = await User.findById(req.session.userId);
  res.json({ user });
});
```

33.3 API-to-API con Client Credentials

■ JS | client-credentials.js — Client Credentials flow con token caching automatico

```
// client-credentials.js — autenticazione machine-to-machine
// Usato per comunicazioni tra microservizi

class OAuth2Client {
  constructor({ tokenUrl, clientId, clientSecret, scope }) {
    this.tokenUrl = tokenUrl;
    this.clientId = clientId;
    this.clientSecret = clientSecret;
    this.scope = scope;
    this._token = null;
    this._tokenExpiry = 0;
  }

  async getToken() {
    // Riusa il token se ancora valido (con margine 30s)
    if (this._token && Date.now() < this._tokenExpiry - 30000) {
      return this._token;
    }

    const res = await fetch(this.tokenUrl, {
      method: 'POST',
      headers: { 'Content-Type': 'application/x-www-form-urlencoded' },
      body: new URLSearchParams({
        grant_type:    'client_credentials',
        client_id:     this.clientId,
        client_secret: this.clientSecret,
        scope:         this.scope,
      }),
    });

    if (!res.ok) throw new Error(`Token request fallita: ${res.status}`);

    const data = await res.json();
    this._token = data.access_token;
    this._tokenExpiry = Date.now() + data.expires_in * 1000;
    return this._token;
  }

  async fetch(url, options = {}) {
    const token = await this.getToken();
    return fetch(url, {
      ...options,
      headers: {
        ...options.headers,
        Authorization: `Bearer ${token}`,
      },
    });
  }
}

// Uso nel microservizio
const inventoryClient = new OAuth2Client({
  tokenUrl:     process.env.AUTH_SERVER_URL + '/oauth/token',
  clientId:     process.env.INVENTORY_CLIENT_ID,
  clientSecret: process.env.INVENTORY_CLIENT_SECRET,
  scope:        'inventory:read inventory:write',
});

// Ogni chiamata usa automaticamente il token valido
const stock = await inventoryClient.fetch(
  'http://inventory-service/api/stock/123'
);
```

Fonti: oauth.net/2/ (spec OAuth 2.0); openid.net/connect/ (spec OpenID Connect); npmjs.com/package/openid-client; oauth.net/2/pkce/ (PKCE spec); auth0.com/docs/get-started/authentication-and-authorization-flow

34. JOB QUEUE AVANZATA CON BULLMQ

BullMQ è la libreria di job queue più completa per Node.js, costruita su Redis. Gestisce code di lavori asincrone con retry automatico, scheduling (cron), priorità, batch processing e pannello di monitoraggio integrato.

34.1 Architettura BullMQ

Concetto	Descrizione	Esempio
Queue	Canale named per i job	emails, payments, reports, notifications
Job	Unita' di lavoro con payload JSON e opzioni	{ to: 'user@...', subject: '...' }
Worker	Processo che consuma ed esegue i job	Funzione async che processa un job alla volta
Scheduler	Aggiunge job ripetuti (cron)	Ogni giorno alle 8:00: invia report
Flow	Pipeline di job dipendenti (parent/child)	Ordine -> Pagamento -> Spedizione -> Email
Events	Webhook su completamento, errore, progresso	job.on('completed'), queue.on('failed')
Bull Board	Dashboard web per monitoraggio	UI per vedere job in coda, falliti, completati

34.2 Setup completo con Worker e Scheduler

■ JS | bullmq.js — Queue, Worker con concurrency e rate limit, Scheduler cron, Flow

```
// npm install bullmq ioredis @bull-board/api @bull-board/express

// queues/index.js — definizione code
const { Queue } = require('bullmq');

const connection = {
  host: process.env.REDIS_HOST || 'localhost',
  port: parseInt(process.env.REDIS_PORT || '6379'),
};

const defaultJobOptions = {
  attempts: 3,
  backoff: { type: 'exponential', delay: 2000 },
  removeOnComplete: { count: 100 },    // mantieni ultimi 100 completati
  removeOnFail: { count: 500 },        // mantieni ultimi 500 falliti
};

const emailQueue        = new Queue('emails',        { connection, defaultJobOptions });
const paymentQueue      = new Queue('payments',      { connection, defaultJobOptions });
const reportQueue       = new Queue('reports',       { connection, defaultJobOptions });
const notificationQueue = new Queue('notifications', { connection, defaultJobOptions });

module.exports = { emailQueue, paymentQueue, reportQueue, notificationQueue };

// workers/emailWorker.js — worker per invio email
const { Worker } = require('bullmq');
const nodemailer = require('nodemailer');
const { renderTemplate } = require('../utils/templates');

const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: parseInt(process.env.SMTP_PORT || '587'),
  auth: { user: process.env.SMTP_USER, pass: process.env.SMTP_PASS },
});

const emailWorker = new Worker('emails', async (job) => {
  const { to, subject, template, data } = job.data;

  // Aggiorna progresso (visibile in dashboard)
  await job.updateProgress(10);

  // Renderizza template HTML
  const html = await renderTemplate(template, data);
  await job.updateProgress(50);

  // Invia email
  const info = await transporter.sendMail({
    from: process.env.EMAIL_FROM,
    to,
    subject,
    html,
    text: html.replace(/<[^>]+>/g, ''),
  });

  await job.updateProgress(100);
  return { messageId: info.messageId, accepted: info.accepted };

}, {
  connection: { host: process.env.REDIS_HOST, port: 6379 },
  concurrency: 5,              // 5 email in parallelo
  limiter: { max: 100, duration: 60000 },  // max 100/min (rate limit SMTP)
});

// Event listener del worker
```

```
emailWorker.on('completed', (job, result) => {
  console.log(`Email ${job.id} inviata a ${job.data.to}: ${result.messageId}`);
});

emailWorker.on('failed', (job, err) => {
  console.error(`Email ${job?.id} fallita (tentativo ${job?.attemptsMade}):`,
    err.message);
});

// Scheduler: job ricorrenti (cron syntax)
const { QueueScheduler, RepeatableJob } = require('bullmq');

// Aggiungi job ripetuto
await reportQueue.add(
  'daily-report',
  { type: 'sales', recipients: ['admin@example.com'] },
  {
    repeat: { cron: '0 8 * * *' },  // ogni giorno alle 8:00
    jobId: 'daily-sales-report',    // ID fisso per evitare duplicati
  }
);

// ■■ FLOW: job con dipendenze (parent/child) ■■■■■■■■■■■■■■■■■■■■■■■■■■
const { FlowProducer } = require('bullmq');
const flow = new FlowProducer({ connection });

await flow.add({
  name: 'process-order',
  queueName: 'orders',
  data: { orderId: '123' },
  children: [
    {
      name: 'send-confirmation',
      queueName: 'emails',
      data: { template: 'order-confirmed', orderId: '123' },
    },
    {
      name: 'update-inventory',
      queueName: 'inventory',
      data: { orderId: '123', action: 'reserve' },
    },
  ],
});
// I children si completano prima che il parent venga eseguito
```

34.3 Bull Board: dashboard di monitoraggio

■ JS | bull-board.js — dashboard web per monitoraggio code BullMQ

```
// monitoring/bullBoard.js — pannello web per monitoraggio code
const { createBullBoard } = require('@bull-board/api');
const { BullMQAdapter }   = require('@bull-board/api/bullMQAdapter');
const { ExpressAdapter }  = require('@bull-board/express');
const { emailQueue, paymentQueue, reportQueue } = require('../queues');

const serverAdapter = new ExpressAdapter();
serverAdapter.setBasePath('/admin/queues');

createBullBoard({
  queues: [
    new BullMQAdapter(emailQueue),
    new BullMQAdapter(paymentQueue),
    new BullMQAdapter(reportQueue),
  ],
  serverAdapter,
});

// Aggiungi al server Express (con autenticazione!)
app.use(
  '/admin/queues',
  requireAdminAuth,                 // FONDAMENTALE: proteggi la dashboard
  serverAdapter.getRouter()
);

// Accedi a http://localhost:3000/admin/queues
// Funzionalita': vedere job attivi, completati, falliti; retry manuale;
// statistiche throughput; pause/resume code
```

Fonti: docs.bullmq.io; npmjs.com/package/bullmq; github.com/felixmosh/bull-board (Bull Board UI); npmjs.com/package/nodemailer; redis.io/docs/manual/patterns/bull/

35. INTERNAZIONALIZZAZIONE E LOCALIZZAZIONE (I18N)

L'internazionalizzazione (i18n) è il processo di progettare un'applicazione affinché possa essere facilmente adattata a diverse lingue e culture senza modifiche al codice sorgente. La localizzazione (l10n) è l'adattamento effettivo a una specifica cultura/lingua.

35.1 i18next: la libreria standard per Node.js

■ BASH | Installazione i18next per Node.js

```
# npm install i18next i18next-fs-backend i18next-http-middleware
# Per Express: aggiunge req.t() per le traduzioni
```

■ JS | i18n/index.js — i18next con rilevamento lingua, namespace, interpolazione

```
// i18n/index.js — configurazione i18next per Express

const i18next = require('i18next');
const Backend = require('i18next-fs-backend');
const middleware = require('i18next-http-middleware');
const path = require('path');

await i18next
  .use(Backend)
  .use(middleware.LanguageDetector)
  .init({
    // Lingue supportate
    supportedLngs: ['it', 'en', 'de', 'fr', 'es'],
    fallbackLng: 'en',
    defaultNS: 'common',

    // Carica file JSON da disco
    backend: {
      loadPath: path.join(__dirname, 'locales/{{lng}}/{{ns}}.json'),
    },

    // Rilevamento lingua dall header Accept-Language
    detection: {
      order: ['header', 'querystring', 'cookie'],
      lookupHeader: 'accept-language',
      lookupQuerystring: 'lang',
      lookupCookie: 'i18next',
      caches: ['cookie'],
    },

    interpolation: { escapeValue: false },
  });

// Aggiungi middleware Express
app.use(middleware.handle(i18next));

// locales/it/common.json
const itTranslations = {
  "welcome": "Benvenuto, {{nome}}!",
  "errors": {
    "notFound": "Risorsa non trovata",
    "unauthorized": "Accesso non autorizzato",
    "validation": "Dati non validi: {{campo}}"
  },
  "email": {
    "subject": {
      "welcome": "Benvenuto in {{appName}}!",
      "orderConfirmed": "Ordine {{orderId}} confermato"
    }
  },
  "pagination": {
    "showing": "Mostrando {{from}}-{{to}} di {{total}} risultati"
  }
};

// locales/en/common.json (fallback)
const enTranslations = {
  "welcome": "Welcome, {{nome}}!",
  "errors": {
    "notFound": "Resource not found",
    "unauthorized": "Unauthorized access",
    "validation": "Invalid data: {{campo}}"
  }
```

```
};

// Uso nelle route
app.get('/api/users/:id', async (req, res) => {
  const user = await User.findById(req.params.id);
  if (!user) {
    return res.status(404).json({
      error: req.t('errors.notFound'),  // traduce in base alla lingua della richiesta
    });
  }
  res.json({
    message: req.t('welcome', { nome: user.nome }),
    data: user,
  });
});
```

35.2 Formattazione date, numeri e valute

■ JS | i18n-format.js — Intl API: date, numeri, valute, plurali senza dipendenze

```
// Usa Intl API nativa di Node.js (non serve libreria)

// ■■ DATE E ORE ■■■
```

■ JS | email-templates — email multilingue con MJML, Handlebars e i18next

```
// email-templates/welcome.js — email multilingue con Handlebars
// npm install handlebars mjml

const handlebars = require('handlebars');
const mjml2html = require('mjml');
const fs = require('fs');
const path = require('path');
const i18next = require('i18next');

async function renderEmailTemplate(templateName, data, locale = 'it') {
  // Carica template MJML (framework email responsive)
  const mjmlTemplate = fs.readFileSync(
    path.join(__dirname, `${templateName}.mjml`),
    'utf8'
  );

  // Traduci le stringhe nel template
  const t = i18next.getFixedT(locale);
  const templateData = {
    ...data,
    t,  // passa funzione traduzione al template
    locale,
    year: new Date().getFullYear(),
  };

  // Compila con Handlebars
  const compiled = handlebars.compile(mjmlTemplate)(templateData);

  // Converti MJML in HTML ottimizzato per email client
  const { html, errors } = mjml2html(compiled, {
    validationLevel: 'soft',
  });

  if (errors.length > 0) {
    console.warn('MJML warnings:', errors);
  }

  return html;
}

// templates/welcome.mjml (frammento)
const welcomeMjml = `
<mjml>
  <mj-head>
    <mj-title>{{t "email.subject.welcome" appName=appName}}</mj-title>
  </mj-head>
  <mj-body>
    <mj-section>
      <mj-column>
        <mj-text font-size="24px">
          {{t "welcome" nome=user.nome}}
        </mj-text>
        <mj-text>
          {{t "email.welcomeBody" appName=appName}}
        </mj-text>
        <mj-button href="{{ctaUrl}}">
          {{t "email.cta"}}
        </mj-button>
      </mj-column>
    </mj-section>
  </mj-body>
</mjml>
`;
```

```
// Invio email multilingue
async function inviaEmailBenvenuto(user) {
  const locale = user.preferredLocale || 'it';
  const html = await renderEmailTemplate('welcome', {
    user,
    appName: 'MioApp',
    ctaUrl: `${process.env.BASE_URL}/onboarding`,
  }, locale);

  await transporter.sendMail({
    to: user.email,
    subject: i18next.getFixedT(locale)('email.subject.welcome', {
      appName: 'MioApp'
    }),
    html,
  });
}
```

Fonti: i18next.com/overview/getting-started; npmjs.com/package/i18next-http-middleware; developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Intl; mjml.io/documentation (MJML email framework); w3.org/International/articles/language-tags/

36. MIGRAZIONE E INTEGRAZIONE CON SISTEMI LEGACY

Molte aziende devono integrare Node.js con sistemi legacy scritti in Java, PHP, .NET o COBOL. Questa sezione descrive strategie pratiche per migrazione graduale, strangler fig pattern e coesistenza.

36.1 Strangler Fig Pattern

Il pattern Strangler Fig (Martin Fowler, 2004) permette di migrare un sistema legacy sostituendolo gradualmente con nuovo codice, senza big-bang rewrite. Si aggiunge un proxy davanti al sistema legacy: le nuove funzionalità vengono implementate in Node.js, le vecchie rimangono nel legacy fino alla sostituzione.

■ JS | proxy-router.js — Strangler Fig: proxy tra Node.js (nuovo) e legacy

```
// proxy-router.js — Strangler Fig con http-proxy-middleware
// npm install http-proxy-middleware

const express = require('express');
const { createProxyMiddleware } = require('http-proxy-middleware');

const app = express();

// Nuove route implementate in Node.js (priorità alta)
app.use('/api/v2/users',    require('./routes/users-new'));
app.use('/api/v2/products', require('./routes/products-new'));
app.use('/api/v2/auth',     require('./routes/auth-new'));

// Tutto il resto va al sistema legacy (PHP/Java)
app.use('/', createProxyMiddleware({
  target: process.env.LEGACY_URL || 'http://legacy-server:8080',
  changeOrigin: true,
  on: {
    error: (err, req, res) => {
      console.error('Proxy error:', err.message);
      res.status(502).json({ error: 'Legacy service unavailable' });
    },
    proxyReq: (proxyReq, req) => {
      // Aggiungi header per identificare richieste proxiate
      proxyReq.setHeader('X-Forwarded-By', 'nodejs-proxy');
      proxyReq.setHeader('X-Request-Id', req.id);
    },
  },
  // Log ogni proxy request
  logger: console,
}));

app.listen(3000, () => {
  console.log('Strangler Fig proxy in ascolto su :3000');
  console.log('Legacy system su:', process.env.LEGACY_URL);
});

// Configurazione Nginx (alternativa al proxy Node.js)
// location /api/v2/ {
//   proxy_pass http://nodejs-server:3000;
// }
// location / {
//   proxy_pass http://legacy-server:8080;
// }
```

36.2 Integrazione con Java via REST e gRPC

Metodo	Latenza	Complessità	Quando
REST HTTP	Media (1-50ms)	Bassa	API esistenti; team separati; compatibilità browser
gRPC	Bassa (<1ms LAN)	Media	Microservizi interni; streaming; contratto tipizzato
Apache Kafka	Media (async)	Alta	Event streaming; disaccoppiamento forte; replay
RabbitMQ (AMQP)	Bassa-media	Media	Task queue; routing complesso; ack/nack
Redis Pub/Sub	Molto bassa	Bassa	Notifiche real-time; semplice; no persistenza
Database condiviso	Dipende da DB	Bassa	Migrazione graduale; evitare in nuovi sistemi
File system condiviso	Dipende da I/O	Bassa	Batch processing; file grandi; legacy SFTP

36.3 Migrazione database graduale

■ JS | db-migration.js — dual-write e backfill per migrazione graduale database

```
// db-migration.js — migrazione dati da MySQL legacy a MongoDB/PostgreSQL
// Pattern: dual-write + backfill

const mysql = require('mysql2/promise');
const { PrismaClient } = require('@prisma/client');

const mysqlConn = await mysql.createConnection({
  host: process.env.LEGACY_MYSQL_HOST,
  database: process.env.LEGACY_MYSQL_DB,
  user: process.env.LEGACY_MYSQL_USER,
  password: process.env.LEGACY_MYSQL_PASS,
});

const prisma = new PrismaClient();

// ■■ FASE 1: DUAL-WRITE ■■■
```

```
            legacyId: String(row.id),
            createdAt: row.created_at,
          },
          update: {
            nome: row.nome,
            email: row.email,
          },
        })
      )
    );

    totalMigrati += rows.length;
    offset += batchSize;
    console.log(`Migrati ${totalMigrati} utenti...`);

    // Pausa per non sovraccaricare il legacy
    await new Promise(r => setTimeout(r, 100));
  }

  console.log(`Backfill completato: ${totalMigrati} utenti migrati`);
}

// ■■ FASE 3: CUTOVER ■■■
```

37. CONCLUSIONI: NODE.JS NEL 2025 E OLTRE

Abbiamo percorso insieme un viaggio completo nell'ecosistema Node.js: dalla storia alle architetture moderne, dalla sintassi base alle integrazioni AI, dalla sicurezza al deployment in produzione. Questa sezione finale sintetizza le competenze acquisite e indica i prossimi passi per continuare a crescere.

37.1 Mappa delle competenze Node.js

Livello	Competenze	Capitoli di riferimento
Principiante	Installazione nvm; sintassi JS moderna; npm base; primo server Express; operazioni fs	Cap. 2, 4, 5, 6, 7
Intermedio	API REST con auth JWT; MongoDB/Prisma; testing Jest; Docker; WebSocket; gestione errori	Cap. 7, 8, 10, 14, 15
Avanzato	Event Loop profondo; Streams; performance profiling; cybersecurity OWASP; TypeScript; gRPC	Cap. 3, 9, 19, 25, 22, 30
Expert	Architetture microservizi; MCP server; AI integration; migrazione legacy; monorepo; IoT/Modbus	Cap. 13, 12, 16, 26, 27, 36
Staff Engineer	Roadmap tecnologica; design system-wide; mentoring; contribuzione open source	Cap. 17, 28, 37
Distinguished Engineer	Standard industriali; RFC e specifiche; conferenze; autore librerie diffuse	Community + Cap. 26, 37

37.2 Il percorso di apprendimento consigliato

■ BASH | Percorso apprendimento strutturato: 12 mesi da principiante a expert

```
# Percorso consigliato per diventare sviluppatore Node.js esperto

## MESE 1-2: Fondamenta
- Installa Node.js 22 LTS con nvm
- JavaScript ES2024: async/await, destructuring, ESM
- Costruisci API REST con Express e MongoDB
- Scrivi i primi test con Jest

## MESE 3-4: Produzione
- TypeScript: aggiungi al tuo progetto Express
- Autenticazione JWT completa con refresh token
- Docker: containerizza la tua applicazione
- CI/CD con GitHub Actions

## MESE 5-6: Scalabilita
- PostgreSQL + Prisma: migra da MongoDB
- Redis: caching e BullMQ per job asincroni
- WebSocket con Socket.IO: chat real-time
- Performance profiling con clinic.js

## MESE 7-9: Architettura
- Microservizi: split l applicazione in servizi
- GraphQL: implementa Apollo Server
- gRPC: comunicazione inter-servizio
- Kubernetes: deploy su cluster

## MESE 10-12: Specializzazione (scegli un percorso)
# Percorso AI:
- Anthropic SDK + streaming
- Vector database + RAG
- Server MCP: esponi tool per LLM

# Percorso IoT/Industriale:
- MQTT + Mosquitto broker
- Modbus TCP: leggi PLC
- OPC-UA: integrazione SCADA
- InfluxDB: time-series data

# Percorso Open Source:
- Contribuisci a un progetto Node.js
- Pubblica la tua libreria npm
- Scrivi documentazione e tutorial
```

37.3 Risorse per continuare ad imparare

Tipo	Risorsa	Livello
Documentazione	nodejs.org/api/ — API ufficiale completa	Tutti
Corso	The Odin Project (theodinproject.com) — full-stack gratuito	Principiante
Libro	Node.js Design Patterns (Casciaro, Mammino) — O'Reilly	Intermedio
Blog	Node.js blog (nodejs.org/en/blog) — annunci e articoli ufficiali	Tutti
Community	r/node su Reddit; Discord Node.js; Stack Overflow [node.js]	Tutti
Conferenze	NodeConf EU; OpenJS World; JSNation	Intermedio-Avanzato
Podcast	Syntax.fm; JS Party; Node.js Podcast	Tutti
Newsletter	Node Weekly (nodeweekly.com) — aggiornamenti settimanali	Tutti
Repository	github.com/goldbergyoni/nodebestpractices — 100k+ stelle	Intermedio
Pratica	Exercism.io (track JavaScript); LeetCode; HackerRank	Principiante-Intermedio
Certificazione	OpenJS Node.js Application Developer (JSNAD)	Intermedio
Open Source	Contribuisci a Express, Fastify, Prisma, Socket.IO	Avanzato

37.4 L'ecosistema Node.js nel 2025: stato dell'arte

A fine 2025, Node.js si trova in una posizione di forza solida: oltre **20 milioni di sviluppatori** nel mondo usano Node.js (Stack Overflow Survey 2024), l'ecosistema npm conta **2,5 milioni di pacchetti** e il runtime continua a evolversi con release regolari. Le sfide principali — TypeScript nativo, edge computing, integrazione AI — stanno ricevendo risposte concrete dalla community e dalla Node.js Foundation.

Indicatore	Dato 2025	Trend
Sviluppatori globali	~20 milioni	Crescita stabile (+8% YoY)
Pacchetti npm	2,5 milioni	Crescita costante
Download npm mensili	40+ miliardi	In aumento
Versione LTS corrente	Node.js 22.x (LTS: Jod)	LTS ogni anno pari

Indicatore	Dato 2025	Trend
Utilizzo enterprise	Netflix, PayPal, LinkedIn, Uber, NASA	Adozione crescente
Popolarita' (SO Survey)	62% degli sviluppatori backend	Prima da 10 anni
Concorrenti principali	Deno 2.0, Bun 1.x	Spingono Node.js a migliorare
Uso con AI/LLM	SDK ufficiali Anthropic + OpenAI	In rapida crescita 2024-2025
TypeScript adoption	70%+ nuovi progetti	Standard de facto
Edge deployment	Cloudflare Workers, Vercel Edge	Trend in forte crescita
Package manager	npm 10.x; pnpm 9.x; yarn berry 4.x	pnpm in crescita per monorepo
Runtime alternativi	Deno 2.0 (npm compat); Bun 1.x (velocissimo)	Spingono innovazione Node.js
Sicurezza supply chain	npm provenance; Socket.dev; Dependabot	Standard 2025 per audit

✓ **BEST PRACTICE** Node.js rimane nel 2025 una delle scelte tecnologiche più solide per API, microservizi, strumenti CLI e server real-time. La combinazione con TypeScript, Docker e le API AI lo rende particolarmente potente per applicazioni moderne. Continua a studiare, sperimentare e contribuire alla community: l'ecosistema Node.js è costruito da persone come te.

37.5 Domande frequenti (FAQ)

D: Devo imparare JavaScript prima di Node.js?

R: Si, assolutamente. Node.js esegue JavaScript, quindi le basi del linguaggio (variabili, funzioni, async/await, array, oggetti) sono fondamentali. Dedica almeno 4-6 settimane a JavaScript prima di passare a Node.js.

D: Express o Fastify per un nuovo progetto?

R: Per la maggior parte dei progetti, entrambi vanno bene. Fastify e' piu' veloce e ha schema validation integrata (Zod/JSON Schema), ma Express ha un ecosistema piu' vasto e piu' risorse di apprendimento. Se le performance sono critiche o si usa TypeScript, Fastify e' preferibile.

D: MongoDB o PostgreSQL?

R: Dipende dai dati. MongoDB e' ideale per dati con struttura variabile, prototyping rapido, dati gerarchici. PostgreSQL e' ideale per dati relazionali, transazioni ACID, query complesse. Per la maggior parte delle applicazioni business, PostgreSQL con Prisma e' la scelta piu' robusta.

D: Node.js e' adatto per calcoli CPU-intensive?

R: Node.js con un singolo thread non e' ottimale per operazioni CPU-intensive (ML, encoding video, compressione pesante). Usa Worker Threads per parallelismo in-process, o delega a un microservizio Python/Go. Per ML in produzione, usa un servizio FastAPI/Python dedicato.

D: Conviene usare TypeScript da subito?

R: Si, per qualsiasi progetto che durera' piu' di qualche settimana o avra' piu' di un collaboratore. Il costo iniziale di setup (tsconfig, tipi) viene ripagato rapidamente con meno bug runtime e migliore autocompletamento IDE.

D: Come gestire segreti in produzione?

R: MAI nel codice o nel repository. Per ambienti semplici: variabili d'ambiente del server. Per team: Doppler, AWS Secrets Manager, HashiCorp Vault, GCP Secret Manager. Configura sempre la rotazione automatica dei segreti critici.

D: Node.js e' sicuro per applicazioni finanziarie?

R: Si, con le giuste precauzioni: HTTPS ovunque, validazione input con Zod, rate limiting, audit npm, JWT con scadenza breve, argon2 per le password, logging strutturato senza dati sensibili. Molte banche e fintech usano Node.js in produzione (PayPal, Stripe, Revolut).

D: Kubernetes o PM2 per il deployment?

R: PM2 per applicazioni su singolo server o VPS. Kubernetes per microservizi, alta disponibilita', scaling automatico. Per startup early-stage: PM2 su VPS o Railway/Render e' piu' semplice e economico. Passa a Kubernetes quando la complessita' operativa lo giustifica.

D: Come scegliere la versione Node.js da usare?

R: Sempre la versione LTS (Long Term Support) in produzione. Controlla la roadmap su nodejs.org/releases: le versioni con numero pari (18, 20, 22) diventano LTS. Evita versioni Current in produzione: sono instabili per definizione.

D: Vale la pena imparare Deno o Bun?

R: Deno 2.0 e Bun sono eccellenti da conoscere, ma Node.js rimane il runtime dominante con il 95%+ delle offerte di lavoro. Impara Node.js a fondo prima, poi esplora Deno e Bun come strumenti aggiuntivi. Le competenze JavaScript/TypeScript si trasferiscono facilmente.

37.6 Ecosistema npm: pacchetti per categoria (2025)

Categoria	Consigliato 2025	Alternativa valida	Evitare
Web framework	Fastify 4.x	Express 4.x / Hono 4.x	Koa (inattivo); Sails (pesante)
Database ORM	Prisma 5.x	Drizzle ORM (piu' leggero)	Sequelize (lento; legacy API)
Validazione	Zod 3.x	Joi 17.x; Yup 1.x	Ajv grezzo (verboso)
Auth JWT	jose 5.x	jsonwebtoken (mantenuto)	jwt-simple (abbandonato)
Password hash	argon2 0.3x	bcrypt 5.x	MD5/SHA1 per password (insicuro)
Logger	pino 9.x	winston 3.x	console.log in produzione
HTTP client	fetch nativo (Node.js 22)	axios 1.x; got 14.x	node-fetch (non necessario)
WebSocket	ws 8.x	socket.io 4.x	ws versioni <7 (vulnerabilita')
Job queue	bullmq 5.x	bee-queue (semplice)	kue (non mantenuto)
Email	nodemailer 6.x	Resend SDK	@sendgrid/mail 7.x
Image processing	sharp 0.33x	jimp (pure JS, lento)	gm (legacy GraphicsMagick)
PDF generation	pdfkit 0.14x	puppeteer (headless Chrome)	html-pdf (abbandonato)
CSV parsing	csv-parse 5.x	papaparse (browser/node)	csv (API non intuitiva)
Environment	dotenv 16.x	dotenv-flow (multi-env)	config (complesso)
Process manager	PM2 5.x	Forever (semplice)	nodemon in produzione
Test runner	Vitest 1.x	Jest 29.x	Mocha/Chai (legacy)
Linter	ESLint 9.x (flat config)	Biome (piu' veloce)	JSHint (obsoleto)
Bundler	esbuild	Rollup; Webpack 5	Browserify (obsoleto)
TypeScript compiler	tsc + tsx (dev)	ts-node	ts-node/esm (configurazione complessa)

Fonti: survey.stackoverflow.co/2024 (Stack Overflow Developer Survey 2024); npmjs.com/about; nodejs.org/en/blog; openjs.foundation; github.com/goldbergyoni/nodebestpractices