Node.js & Express – Complete Mastery (Basic → Advanced → Production)

> **Single-source documentation covering Node.js & Express.js from basics to advanced concepts, production patterns, file uploads, cloud integrations (AWS S3, Cloudinary), caching with Redis, middleware, authentication, error handling, and common case studies.**

This document is designed for **senior backend engineers** to master **Node.js & Express at production level**.

1️⃣CORE CONCEPTS OF NODE.JS

1.1 Node.js Basics

- Event-driven, non-blocking I/O
- Single-threaded but asynchronous
- Modules (`require` / `import`)
- `package.json`, npm/yarn, semantic versioning

1.2 Core Modules

- `fs`, `http`, `path`, `crypto`, `os`, `events`
- Example: Reading a file asynchronously

```
const fs = require('fs');
fs.readFile('data.txt', 'utf-8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

1.3 Event Loop & Async Patterns

- `setTimeout`, `setImmediate`, `process.nextTick`
- Promises, async/await
- Streams & buffers

2️⃣EXPRESS.JS BASICS

2.1 Creating Server

```
const express = require('express');
const app = express();
app.use(express.json());
app.get('/', (req, res) => res.send('Hello World'));
app.listen(3000, () => console.log('Server running'));
```

2.2 Routing & Middleware

- Middleware order matters
- `app.use()`, `router.use()`, custom middleware

  ```
  app.use((req, res, next) => {
    console.log(`${req.method} ${req.url}`);
    next();
  });
  ```

2.3 Error Handling

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: err.message });
});
```

3️⃣DATABASE INTEGRATION

3.1 MongoDB / Mongoose

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/mydb');
const User = mongoose.model('User', new mongoose.Schema({ name: String,
email: String }));
const users = await User.find({});
```

3.2 PostgreSQL / Sequelize / pg

```
const { Pool } = require('pg');
const pool = new Pool({ user: 'user', host: 'localhost', database: 'db',
password: 'pass', port: 5432 });
const result = await pool.query('SELECT * FROM users');
```

4️⃣FILE UPLOADS & CLOUD STORAGE

4.1 Using Multer for local uploads

```
const multer = require('multer');
const upload = multer({ dest: 'uploads/' });
app.post('/upload', upload.single('file'), (req, res) => {
  res.send(req.file);
});
```

4.2 AWS S3 Upload

```
const AWS = require('aws-sdk');
const s3 = new AWS.S3({ accessKeyId, secretAccessKey });
const uploadFile = async (file) => {
  const params = { Bucket: 'bucket-name', Key: file.originalname, Body:
file.buffer };
  return await s3.upload(params).promise();
};
```

4.3 Cloudinary Upload

```
const cloudinary = require('cloudinary').v2;
cloudinary.config({ cloud_name, api_key, api_secret });
const result = await cloudinary.uploader.upload(file.path);
```

5️⃣CACHING WITH REDIS

5.1 Connecting Redis

```
const redis = require('redis');
const client = redis.createClient({ url: 'redis://localhost:6379' });
await client.connect();
```

5.2 Set & Get cache

```
await client.set('users', JSON.stringify(users), { EX: 3600 });
const cachedUsers = JSON.parse(await client.get('users'));
```

5.3 Cache Middleware Example

```
const cacheMiddleware = async (req, res, next) => {
  const data = await client.get(req.originalUrl);
  if (data) return res.json(JSON.parse(data));
  next();
};
```

6️⃣AUTHENTICATION & AUTHORIZATION

6.1 JWT Token Auth

```
const jwt = require('jsonwebtoken');
const token = jwt.sign({ userId: user._id }, 'secret', { expiresIn: '1h' });
```

6.2 Protect Routes

```
const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  try {
    req.user = jwt.verify(token, 'secret');
    next();
  } catch (e) {
    res.status(401).json({ message: 'Unauthorized' });
  }
};
```

7️⃣ADVANCED EXPRESS PATTERNS

7.1 Async Error Handling

```
const asyncHandler = fn => (req, res, next) => Promise.resolve(fn(req, res,
next)).catch(next);
app.get('/users', asyncHandler(async (req, res) => {
  const users = await User.find();
  res.json(users);
}));
```

7.2 Structuring Large Apps

- MVC or feature-based folder structure
- Separate routes, controllers, services, models
- Environment configs using dotenv

7.3 Logging

- Use **winston** or **pino** for production logging
- Log levels, transports, rotation

8️⃣PRODUCTION & PERFORMANCE

- Use **PM2** for process management
- Enable **compression** middleware
- Apply **rate limiting & helmet** for security
- Use **Redis cache** and **DB indexing** for performance
- Error tracking via **Sentry or LogRocket**

9️⃣REAL CASE STUDIES / SCENARIOS

9.1 File Upload & Response

- Upload multiple files to S3
- Save URLs to MongoDB
- Serve files with signed URLs

9.2 Feed API with Pagination & Cache

- Redis cache for first N pages
- Cursor-based pagination
- Aggregate posts with comments and likes

9.3 Payment Transaction API

- Use MongoDB transactions / PostgreSQL ACID
- Deduplicate requests using unique indexes
- Emit events via RabbitMQ / Kafka

9.4 Analytics Pipeline

- MongoDB aggregation with `$group`, `$match`, `$lookup`
- Generate weekly/monthly reports
- Store summary in Redis for dashboard

---

1 0️⃣FINAL MENTAL MODEL

- Node.js handles async, Express routes handle HTTP, DB handles data integrity
- Cloud uploads (S3/Cloudinary) decouple storage
- Redis handles caching & performance
- JWT for stateless authentication
- Production patterns: structured code, logging, monitoring, security, scaling

📌 **This document gives you everything from basic → advanced → complex production-ready Node.js & Express patterns with cloud and cache integrations in one place.**