Node.js & Express – Advanced Production Mastery

Complete production-level backend reference: optimized APIs, cron jobs, webhooks, caching, mailing, OTPs with Kafka, complex APIs, and best practices. All code examples ready for real-world usage.

1️⃣ PROJECT STRUCTURE (BEST PRACTICES)

```
project/
├── src/
│   ├── controllers/
│   ├── services/
│   ├── models/
│   ├── routes/
│   ├── middlewares/
│   ├── utils/
│   ├── jobs/
│   ├── config/
│   └── app.js
├── package.json
├── .env
├── README.md
```

- Controllers: Handle requests/responses
- Services: Business logic
- Models: DB models (Mongo/PostgreSQL)
- Middlewares: Auth, validation, caching
- Jobs: Cron / scheduled tasks
- Utils: Helpers, mail, Kafka producer/consumer

2️⃣ OPTIMIZED API EXAMPLES

2.1 Caching Middleware (Redis)

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

app.get('/users', cacheMiddleware, async (req, res) => {
  const users = await User.find();
```

```
  await redisClient.set(req.originalUrl, JSON.stringify(users), { EX:
3600 });
  res.json(users);
});
```

2.2 Optimized Pagination API

```
app.get('/posts', async (req, res) => {
  const limit = parseInt(req.query.limit) || 10;
  const lastId = req.query.lastId;
  const query = lastId ? { _id: { $gt: lastId } } : {};
  const posts = await Post.find(query).sort({ _id: 1 }).limit(limit);
  res.json(posts);
});
```

2.3 Complex Aggregation API (MongoDB)

```
app.get('/user-stats', async (req, res) => {
  const stats = await User.aggregate([
    { $lookup: { from: 'orders', localField: '_id', foreignField: 'userId',
as: 'orders' } },
    { $unwind: '$orders' },
    { $group: { _id: '$_id', totalSpent: { $sum: '$orders.total' },
orderCount: { $sum: 1 } } },
    { $sort: { totalSpent: -1 } }
  ]);
  res.json(stats);
});
```

3️⃣ CRON JOBS (NODE-CRON)

```
const cron = require('node-cron');
cron.schedule('0 0 * * *', async () => {
  console.log('Running daily cleanup');
  await OldLogs.deleteMany({ createdAt: { $lt: new Date(Date.now() -
30*24*60*60*1000) } });
});
```

4️⃣ WEBHOOK HANDLER

```
app.post('/webhook/payment', express.json(), async (req, res) => {
  const event = req.body;
  // Verify signature if required
```

```
  await PaymentService.processWebhook(event);
  res.status(200).send('OK');
});
```

5️⃣ MAILING SYSTEM (NODEMAILER)

```
const nodemailer = require('nodemailer');
const transporter = nodemailer.createTransport({
  host: process.env.SMTP_HOST,
  port: process.env.SMTP_PORT,
  auth: { user: process.env.SMTP_USER, pass: process.env.SMTP_PASS }
});

const sendMail = async (to, subject, html) => {
  await transporter.sendMail({ from: 'noreply@example.com', to, subject,
html });
};
```

6️⃣ OTP SYSTEM WITH KAFKA

6.1 Kafka Producer

```
const { Kafka } = require('kafkajs');
const kafka = new Kafka({ clientId: 'otpService', brokers: ['localhost:
9092'] });
const producer = kafka.producer();
await producer.connect();
await producer.send({ topic: 'otp', messages: [{ key: 'userId', value:
JSON.stringify({ otp, userId }) }] });
```

6.2 Kafka Consumer

```
const consumer = kafka.consumer({ groupId: 'otpGroup' });
await consumer.connect();
await consumer.subscribe({ topic: 'otp', fromBeginning: true });
await consumer.run({ eachMessage: async ({ message }) => {
  const data = JSON.parse(message.value.toString());
  await sendMail(data.userEmail, 'Your OTP', `<p>${data.otp}</p>`);
} });
```

7️⃣ ADVANCED MIDDLEWARE & PATTERNS

- **Rate limiting**: Prevent abuse with `express-rate-limit`
- **Input validation**: `Joi` or `express-validator`
- **Async wrapper**: Handle all async errors

```
const asyncHandler = fn => (req, res, next) => Promise.resolve(fn(req, res, next)).catch(next);
```

- **JWT Auth with role-based access**
- **Centralized logging** with Winston or Pino

8️⃣ PERFORMANCE & BEST PRACTICES

- Use Redis caching for frequent queries
- Avoid N+1 queries using aggregation or populate
- Use cursor-based pagination instead of OFFSET
- Compress responses (`compression` middleware)
- Use PM2 or cluster for multi-core utilization
- Health check endpoints
- Environment variables for config
- Monitoring & metrics (Prometheus / Grafana)

9️⃣ FULL PRODUCTION EXAMPLE API FLOW

1. **User requests OTP** → API generates OTP → sends to Kafka
2. **Kafka consumer** → picks OTP message → sends via nodemailer / SMS
3. **User verifies OTP** → updates DB → caches result in Redis
4. **User accesses dashboard** → optimized cached API returns aggregated data
5. **Scheduled tasks** → node-cron cleans up old OTPs & logs
6. **Webhook handler** → listens to payment events, triggers internal service

All layers are **asynchronous, optimized, and ready for scale**.

📌 **This documentation consolidates: complex APIs, optimized Node.js/Express patterns, cron jobs, webhook handling, Redis caching, Kafka OTP/messaging, mailing, production-level practices, and advanced backend architecture**.

It serves as **the ultimate single reference to master Node.js & Express backend development at production level**.