📘 Backend Developer Notes (JWT, Redis, MongoDB, Async/Await, Promises)

✅ 1. MongoDB Connection Setup

```js
// db.js
const mongoose = require('mongoose');

const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URI, {
      useNewUrlParser: true,
      useUnifiedTopology: true,
    });
    console.log('MongoDB connected');
  } catch (err) {
    console.error(err);
    process.exit(1); // fallback (exit on failure)
  }
};

module.exports = connectDB;
```

✅ 2. JWT Middleware (Authentication)

```js
// middleware/auth.js
const jwt = require('jsonwebtoken');

const authMiddleware = (req, res, next) => {
  const token = req.headers['authorization'];
  if (!token) return res.status(401).json({ message: 'No token provided' });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded; // attaching decoded user to request
    next(); // proceed
  } catch (err) {
    return res.status(401).json({ message: 'Invalid token' });
  }
};

module.exports = authMiddleware;
```

✅ 3. Redis Setup & CRUD

```js
// redisClient.js
const redis = require('redis');
const client = redis.createClient();

client.on('error', (err) => console.log('Redis Error:', err));
client.connect();

module.exports = client;
```

Create / Read / Delete from Redis

```js
// cacheService.js
const client = require('./redisClient');

// SET
const cacheData = async (key, value) => {
  await client.set(key, JSON.stringify(value), { EX: 3600 }); // expires in 1hr
};

// GET
const getCachedData = async (key) => {
  const data = await client.get(key);
  return data ? JSON.parse(data) : null; // fallback: return null
};

// DEL
const clearCache = async (key) => {
  await client.del(key);
};

module.exports = { cacheData, getCachedData, clearCache };
```

✅ 4. Fetching Subcategories by Category ID (MongoDB)

```js
// models/Category.js
```

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

const subCategorySchema = new mongoose.Schema({
  name: String,
});

const categorySchema = new mongoose.Schema({
  name: String,
  subcategories: [subCategorySchema],
});

module.exports = mongoose.model('Category', categorySchema);
```

```js
// controller.js
const Category = require('./models/Category');

const getSubcategories = async (req, res) => {
  const { categoryId } = req.params;
  try {
    const category = await Category.findById(categoryId);
    if (!category) return res.status(404).json({ message: 'Category not found' });
    res.json(category.subcategories);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
};
```

✅ 5. async/await, Promise, Callback, Fallback Concepts

🔹 async/await:
- **async**: declares function that returns a **Promise**
- **await**: pauses execution until Promise resolves

```js
const getUser = async (id) => {
  try {
    const user = await User.findById(id); // non-blocking
    return user;
  } catch (err) {
    console.error('Error:', err); // fallback
  }
};
```

🔹 Promise:
```js
const fetchData = () => {
  return new Promise((resolve, reject) => {
    setTimeout(() => resolve('data loaded'), 1000);
  });
};

fetchData().then(console.log);
```

🔹 Callback:
```js
function greet(name, cb) {
  console.log('Hello', name);
  cb(); // callback (runs after greet)
}

greet('Ali', () => console.log('Callback executed'));
```

🔹 Fallback:
```js
const fetchFromCache = async (key) => {
  const cached = await getCachedData(key);
  if (cached) return cached;
  const freshData = await fetchFromDB();
  await cacheData(key, freshData);
  return freshData;
};
```

✅ 6. Node.js Architecture (Simple)

- **Single-threaded** (one main thread runs JS)
- **Event-driven** (executes code on events like requests)
- **Non-blocking** (doesn't wait for long-running tasks)
- Uses **Event Loop** to manage async calls

✅ 7. Steps to Implement JWT Auth (Recap)

1. Install: `npm install jsonwebtoken`
2. Generate Token:

```js
const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: '1h' });
```
3. Save to frontend/localStorage
4. Use `authMiddleware` to protect routes
5. Decode token and access `req.user`

✅ 8. Deployment (Basic)

- Use `dotenv` for secrets
- Dockerize the app (optional)
- Use PM2 for Node process management:
```bash
npm install pm2 -g
pm run build
pm start
```
- Set up Nginx reverse proxy
- Use services like **Render, Railway, or EC2**

✅ Let me know if you want Swagger docs, role-based auth, or multi-db support next!

Creating **microservices** means breaking a big application into **small, independent services** — each responsible for a specific business feature — and they **communicate via APIs or messages**.

🧠 1. What Is a Microservice? (In Simple Words)

- Think of your app like a **pizza shop**:
  - One team bakes.
  - One delivers.
  - One handles payment.

Each team works **independently**, but **together make the shop work** — that's microservices.

✅ 2. Core Concepts

Concept	Meaning (Simple)
Service	A small, self-contained app doing one job (e.g., Auth service, Product service)
API communication	Services talk to each other using **HTTP REST** or **message queues**
Database per service	Each service manages **its own DB**, no shared tables
Stateless	Doesn't remember past state; all info must come with the request
Independent deploy	You can update 1 service without breaking the others

---

🛠️ 3. Steps to Create Microservices in Node.js + Express

🎯 Example Use Case: E-commerce App

We will break it into:

Microservice	Responsibility
Auth Service	Login, Register, JWT
Product Service	Products CRUD, Pricing
Order Service	Place Orders, Status

🧱 4. Basic Folder Structure

```
microservices/
│
├── auth-service/
│   ├── app.js
│   ├── routes/
│   └── controllers/
│
```

```
├── product-service/
│   ├── app.js
│   ├── routes/
│   └── controllers/
│
├── order-service/
│   ├── app.js
│   ├── routes/
│   └── controllers/
│
└── gateway/
    └── index.js (API Gateway / Reverse Proxy)
```

🔐 5. Auth Service – Example with JWT

```
// auth-service/routes/auth.js
const express = require("express");
const jwt = require("jsonwebtoken");
const router = express.Router();

router.post("/login", (req, res) => {
  const { email } = req.body;
  const token = jwt.sign({ email }, "SECRET", { expiresIn: "1h" });
  res.json({ token });
});

module.exports = router;
```

🛒 6. Product Service – Example CRUD

```
// product-service/routes/product.js
const express = require("express");
const router = express.Router();

let products = [];

router.post("/", (req, res) => {
  products.push(req.body);
  res.send("Product added");
});

router.get("/", (req, res) => {
  res.json(products);
});

module.exports = router;
```

📦 7. API Gateway (Simple)

```
// gateway/index.js
const express = require("express");
const { createProxyMiddleware } = require("http-proxy-middleware");

const app = express();

app.use("/auth", createProxyMiddleware({ target: "http://localhost:4000", changeOrigin: true }));
app.use("/products", createProxyMiddleware({ target: "http://localhost:5000", changeOrigin: true }));

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

🧪 8. Run Services (Dev Mode)

```
# Run Auth Service
cd auth-service
PORT=4000 node app.js

# Run Product Service
cd ../product-service
PORT=5000 node app.js

# Run Gateway
cd ../gateway
node index.js
```

Now you can access:

- `POST http://localhost:3000/auth/login`
- `GET http://localhost:3000/products`

⚙️ 9. Recommended for Production

Tool	Purpose
Docker	Containerize each microservice
Docker Compose	Run all services together
Kubernetes	Deploy & scale microservices easily
Redis	For caching or token/session storage
RabbitMQ / Kafka	Async communication between services
PM2	Process manager for each service
MongoDB / Postgres	Independent DBs per service

🧠 10. Want To Learn By Doing?

I can give you:

- 🧪 Complete mini-project with 2 services + API Gateway
- 🐳 Docker setup for all services
- 💬 Redis + JWT integration
- 📨 Add RabbitMQ communication (e.g., order notification)

Would you like the full code zip OR a GitHub repo link setup with all of the above?

You've covered a lot already! Here's a quick checklist to see what's left or worth revisiting **for your Node.js Backend Developer interview** (with microservices, Redis, JWT, MongoDB, etc.):

✅ You've Already Covered Well:

- Node.js basics to advanced (event loop, async/await, single-threaded, non-blocking)
- Express.js routing and middleware
- JWT Authentication with middleware
- Redis (caching + CRUD)
- MongoDB: schema design, fetching subcategories via categoryId
- API scalability: sharding, indexing, performance tuning
- Microservices vs tiered architecture
- Deployment concept overview
- Promises, callbacks, fallbacks
- Error handling concepts

🔁 Suggested Final Topics To Quickly Review or Add (if not done yet):

1. Rate Limiting (API Throttling)

- Prevents API abuse (especially if 1M users hit at once)
- Use middleware like `express-rate-limit`
- Can be stored in Redis for distributed systems

```
const rateLimit = require('express-rate-limit');
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100 // limit each IP to 100 requests per 15 mins
});
app.use(limiter);
```

2. API Gateway (if doing microservices)

- Central entry point for all services (e.g., using NGINX or Express Gateway)
- Handles routing, rate-limiting, auth, logging

3. Service-to-Service Communication

- For microservices:
    - REST over HTTP (simpler)
    - Or Message Queues (like RabbitMQ / Kafka) for async and decoupling

4. Docker Basics (if asked for deployment)

- Package app into containers

```
FROM node:18
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "index.js"]
```

5. CI/CD Pipeline

- Even a basic flow:
    - Push → GitHub Actions / GitLab CI → Run Tests → Deploy to Vercel / Heroku / EC2

6. Unit Testing (Optional but Bonus)

- Use Jest for route or logic testing

```
test('Should return user list', async () => {
  const res = await request(app).get('/api/users');
  expect(res.statusCode).toEqual(200);
});
```

7. Logging

- Use `winston` or `morgan` for logging requests and errors.

📦 Optional Microservices Features:

If you're doing microservices, you can optionally touch:

- **Database per service** pattern
- **Centralized logging** (ELK stack)
- **Distributed tracing** (Zipkin or Jaeger)
- **Health checks** for each service (e.g., `/healthz` endpoint)

✅ You're Interview-Ready!

If you want:

- A **mini mock interview** to practice
- OR, a **cheatsheet for quick revision 10 mins before** interview

Just say the word.