📘 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!