Great, let's move on to **7. Database Integration**. Since you already have Laravel background, I'll explain with comparisons to how you handle DBs in PHP/Laravel.

---

7. Database Integration in Node.js

📌 Overview

Unlike Laravel (which ships with **Eloquent ORM** by default), Node.js gives you **freedom of choice**.
You can connect to **SQL** or **NoSQL** databases using **drivers, query builders, or ORMs**.

📌 Types of Databases with Node.js

1. **Relational Databases (SQL)**
    - MySQL, PostgreSQL, MariaDB, MS SQL Server
    - Tools:
        - **mysql2** → Direct driver
        - **Sequelize / TypeORM / Prisma** → ORMs / Query Builders
2. **NoSQL Databases**
    - MongoDB (most popular in Node world)
    - Tools:
        - **Mongoose** (ODM)
        - Official **MongoDB Node.js Driver**

📌 MySQL Example (Using mysql2)

```
const mysql = require('mysql2');

const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'password',
  database: 'testdb'
});

// Query
connection.query('SELECT * FROM users', (err, results) => {
  if (err) throw err;
  console.log(results);
});
```

📌 PostgreSQL Example (Using pg)

```
const { Client } = require('pg');

const client = new Client({
  user: 'postgres',
  host: 'localhost',
  database: 'testdb',
  password: 'password',
  port: 5432
});

client.connect();

client.query('SELECT * FROM users', (err, res) => {
  if (err) throw err;
  console.log(res.rows);
  client.end();
});
```

📌 MongoDB Example (Using Mongoose ODM)

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

mongoose.connect('mongodb://localhost:27017/testdb')
  .then(() => console.log("MongoDB connected"))
  .catch(err => console.error(err));

// Schema
const UserSchema = new mongoose.Schema({
  name: String,
  email: String,
  age: Number
});

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

// Insert
const user = new User({ name: "John", email: "john@example.com", age: 25 });
user.save().then(() => console.log("User Saved"));
```

📌 Using ORMs in Node.js

1. Sequelize (SQL ORM)

- Similar to Eloquent in Laravel
- Provides models, associations, migrations

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('testdb', 'root', 'password', { dialect:
'mysql' });

const User = sequelize.define('User', {
  name: DataTypes.STRING,
  email: DataTypes.STRING
});

(async () => {
  await sequelize.sync();
  await User.create({ name: 'Alice', email: 'alice@example.com' });
})();
```

2. Prisma (Next-gen ORM)

- Type-safe ORM
- Schema-based (like migrations in Laravel)

📌 Interview-Focused Q&A

Q1: Which database is most commonly used with Node.js?

- MongoDB (because JSON ↔ JS objects = natural fit).

Q2: How does Mongoose differ from Sequelize?

- Mongoose → ODM (for MongoDB only).
- Sequelize → ORM (for SQL databases).

Q3: Why prefer Prisma/TypeORM over raw queries?

- Type safety, migrations, easier to maintain.

Q4: Can Node.js handle transactions?

- Yes, via libraries like Sequelize/TypeORM or raw DB drivers (`BEGIN`, `COMMIT`, `ROLLBACK`).

📌 Key Takeaways

- Node.js supports **both SQL & NoSQL**.
- Popular libraries:
    - **mysql2, pg** for raw SQL

    - **Sequelize, Prisma, TypeORM** for ORM
    - **Mongoose** for MongoDB

- MongoDB is the **most common interview topic** in Node.js context.

👉 Next topic: **8. RESTful APIs with Express** (will cover routing, controllers, middleware, error handling like Laravel).

Do you want me to explain **RESTful APIs** in a **Laravel-to-Express mapping style** (so you see parallels easily), or in a **Node-only deep dive**?