Perfect 👍 Let's cover **Authentication & Security in Node.js** (a very common interview topic). I'll explain **concepts, implementation patterns, and best practices**.

---

9. Authentication & Security in Node.js

📌 Types of Authentication

1. **Session-based Authentication**
   - Traditional approach (used in PHP/Laravel).
   - Server stores a **session ID** in memory/DB, and client stores a **cookie** with the session ID.
   - On every request → session ID is validated.
   - Not ideal for **scalable APIs** (because server needs session storage).
2. **Token-based Authentication (JWT)**
   - Widely used in Node.js REST APIs.
   - Server generates a **signed token (JWT)** after login.
   - Client sends the token in **Authorization header** (`Bearer <token>`).
   - No session storage required → **stateless & scalable**.
3. **OAuth2 & Social Logins**
   - Used for Google, Facebook, GitHub login.
   - Node.js has `passport.js` and `next-auth` (for Next.js) to handle OAuth flows.

📌 Example: JWT Authentication with Express.js

```
const express = require("express");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");

const app = express();
app.use(express.json());

const users = []; // Dummy DB

// Register
app.post("/register", async (req, res) => {
  const hashedPass = await bcrypt.hash(req.body.password, 10);
  users.push({ username: req.body.username, password: hashedPass });
  res.json({ message: "User registered" });
});

// Login
```

```
app.post("/login", async (req, res) => {
  const user = users.find(u => u.username === req.body.username);
  if (!user || !(await bcrypt.compare(req.body.password, user.password))) {
    return res.status(401).json({ error: "Invalid credentials" });
  }
  const token = jwt.sign({ username: user.username }, "secretKey", { expiresIn:
"1h" });
  res.json({ token });
});

// Protected route
app.get("/profile", authenticate, (req, res) => {
  res.json({ message: `Welcome ${req.user.username}` });
});

// Middleware
function authenticate(req, res, next) {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1];
  if (!token) return res.sendStatus(401);

  jwt.verify(token, "secretKey", (err, user) => {
    if (err) return res.sendStatus(403);
    req.user = user;
    next();
  });
}

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

📌 Security Best Practices in Node.js

1. **Input Validation & Sanitization**

- Use `validator` or `joi` library to prevent **SQL injection, XSS**.
- Always validate user input.

2. **Password Security**

- Never store plain text passwords → use `bcrypt`.
- Use **salting** + hashing.

3. **Protect Against Common Attacks**

- **SQL Injection** → use parameterized queries/ORMs (`sequelize`, `mongoose`).
- **XSS (Cross-Site Scripting)** → sanitize input, use templating engines safely.
- **CSRF (Cross-Site Request Forgery)** → use `csurf` middleware for stateful apps.

4. **Helmet Middleware**

```
const helmet = require("helmet");
app.use(helmet());
```

- Sets secure headers → prevents **Clickjacking, MIME sniffing, XSS**.

5. **Rate Limiting**

```
const rateLimit = require("express-rate-limit");
app.use(rateLimit({ windowMs: 15 * 60 * 1000, max: 100 }));
```

- Prevents **brute-force attacks**.

6. **HTTPS**

- Always use SSL/TLS for production.
- Redirect HTTP → HTTPS.

7. **Environment Variables**

- Never hardcode secrets → use `.env` files + `dotenv`.

📌 Interview Q&A

Q1. Difference between Session-based and JWT authentication?

- **Session** → stored on server, requires session DB, not fully scalable.
- **JWT** → stateless, stored on client, better for microservices & APIs.

Q2. How do you secure passwords in Node.js?

- Use `bcrypt` with **hashing + salting**.

Q3. How to prevent brute-force login attacks?

- Implement **rate limiting**, **captcha**, and account lockouts.

Q4. What's the use of `helmet` in Express?

- Adds **security headers** → prevents XSS, clickjacking, MIME sniffing.

☑ Key Takeaways

- Node.js authentication = mostly **JWT** for APIs.
- Always **hash passwords** and **use HTTPS**.
- Protect apps with **helmet, rate-limiting, validation**.
- Know **common vulnerabilities (XSS, CSRF, SQL Injection)** → interview favorite.

👉 Next, do you want me to move to **10. Testing in Node.js (Mocha, Jest, Supertest)** or **11. Error Handling & Debugging**?