Express.js - Practice Sheet

10 Q&A on middleware pipeline, routing, security, and testing

Q1. How does the Express middleware pipeline work?

A: Middleware functions (req, res, next) are executed in registration order. Calling next() passes control to the next function. Calling res.send() or res.json() ends the response. If next(err) is called, Express jumps to the nearest 4-parameter error handler.

Q2. What is the difference between app.use(), app.get(), and app.all()?

A: app.use(path?, fn) runs for every HTTP method matching the path prefix (including sub-paths). app.get(path, fn) matches only GET on the exact path. app.all(path, fn) matches all methods on the exact path. app.use() is the foundation for middleware.

Q3. How do you implement error-handling middleware in Express?

A: Define a function with exactly four parameters (err, req, res, next). Register it last, after all routes. Express detects the 4-param signature and skips it for normal requests. Call next(err) from a route or middleware to activate it.

Q4. How does Express Router modularize routes?

A: const router = express.Router() creates a mini-app. Define routes on router (router.get('/users', handler)). Mount it on the main app with app.use('/api', router). This keeps route logic separated by domain and avoids a monolithic routes file.

Q5. How do you parse different request body types in Express?

A: Use express.json() to parse JSON bodies and express.urlencoded({extended:true}) for HTML form data. Both are built into Express 4.16+. For multipart/form-data (file uploads) use multer middleware, which processes streams and populates req.file / req.files.

Q6. How does session management work in Express?

A: express-session sets a signed session cookie and stores session data in a backend store (MemoryStore for dev, connect-redis or connect-pg-simple for production). req.session.user = user persists data across requests for the session lifetime.

Q7. What does the Helmet middleware do for security?

A: Helmet sets security-relevant HTTP response headers: Content-Security-Policy, X-Frame-Options (clickjacking), X-XSS-Protection, Strict-Transport-Security (HSTS), and others. Call app.use(helmet()) early in the middleware stack before routes.

Q8. How do you connect Express to MongoDB using Mongoose?

A: Call mongoose.connect(uri) once at app startup. Define a Schema and Model (mongoose.model('User', userSchema)). Use User.find({active: true}), User.findById(id), and user.save() to query and persist. Mongoose validates data against the schema before writing.

Q9. How do you implement rate limiting in Express?

A: Use express-rate-limit: const limiter = rateLimit({windowMs: 15*60*1000, max: 100}). Apply with app.use('/api/', limiter). By default it uses in-memory storage; for distributed deployments use rate-limit-redis to share counts across server instances.

Q10. How do you write integration tests for Express routes using Supertest?

A: import request from 'supertest'; const res = await request(app).post('/users').send({name:'Alice'}). Supertest starts the Express app on an ephemeral port, sends the request, and resolves with the response. No need to listen on a real port.