Koa - Practice Sheet

10 Q&A on async middleware, ctx object, routing, and error handling

Q1. What is Koa and how does it improve on Express's middleware model?

A: Koa is built by the Express team using async/await instead of callbacks. Its cascade middleware uses async functions and await next(), giving you clean before/after control without nesting. Koa has no bundled routing or body parsing use separate packages.

Q2. How does Koa's cascading middleware work in practice?

A: Middleware runs sequentially until next() is awaited, then in reverse after. Code before await next() runs on the way in; code after runs on the way out. This makes it trivial to measure duration: record time before next(), log delta after.

Q3. What is the Koa ctx (context) object?

A: ctx bundles the Node.js request and response: ctx.request (Koa's enhanced request), ctx.response, ctx.params (router), ctx.state (app-level data store), and shortcuts like ctx.body, ctx.status, ctx.get(), ctx.set(). Middleware reads and writes via ctx.

Q4. What is the difference between ctx.body and ctx.response.body?

A: They are the same underlying property ctx.body is an alias for ctx.response.body. Koa delegates many properties from ctx directly to ctx.request and ctx.response as shortcuts. Setting ctx.body to an object automatically sends JSON with the correct Content-Type.

Q5. How do you handle routing in Koa with koa-router?

A: const router = new Router(); router.get('/users/:id', async ctx => {ctx.body = await getUser(ctx.params.id)}). Use router.routes() and router.allowedMethods() as middleware on the Koa app. Groups can share prefixes with router.prefix('/api').

Q6. How does centralized error handling work in Koa?

A: Add an error-catching middleware as the first in the stack: try { await next() } catch (err) { ctx.status = err.status || 500; ctx.body = {error: err.message}; ctx.app.emit('error', err, ctx) }. It catches errors from all downstream middleware.

Q7. How do you parse request bodies in Koa?

A: koa-bodyparser adds ctx.request.body for JSON and URL-encoded bodies. Register it before routes: app.use(bodyParser()). For multipart file uploads use @koa/multer or busboy-based middleware. koa-body combines both.

Q8. How do you manage sessions in Koa?

A: koa-session adds ctx.session to store per-user data in a signed cookie. For large payloads configure an external store (Redis). app.keys = ['secret'] must be set for cookie signing. Session data is loaded on request and saved on response.

Q9. How do you serve static files in Koa?

A: koa-static(root, options) creates middleware that checks if the URL maps to a file in root, reads it, and sets the correct Content-Type. Register it early so static file requests bypass the router entirely. Set maxage for browser caching.

Q10. How do you test a Koa application?

A: Use supertest: const server = app.callback(); await request(server).get('/users').expect(200). app.callback() returns a plain http.RequestListener without starting a server, allowing tests to use ephemeral ports managed by supertest.