Fastify - Practice Sheet

10 Q&A on performance, plugin system, schema validation, and hooks

Q1. What makes Fastify faster than Express.js?

A: Fastify uses a radix-tree (find-my-way) router that is O(1) per segment vs Express's regex iteration. JSON serialization uses fast-json-stringify which generates compiled serializers from JSON Schema, skipping generic JSON.stringify entirely.

Q2. How does Fastify's plugin system and encapsulation work?

A: fastify.register(plugin, options) adds a plugin in an encapsulated scope. Decorators and hooks added inside a plugin are scoped to it and its children. fastify-plugin() opts out of encapsulation, making decorators visible to the parent scope.

Q3. How does Fastify's JSON Schema validation protect routes?

A: Attach a schema object to route options: {schema:{body:{type:'object',required:['name'],properties:{name:{type:'string'}}}}}. Fastify uses ajv to validate incoming requests before the handler runs and returns 400 with a structured error if validation fails.

Q4. What are Fastify lifecycle hooks and when do they run?

A: Hooks run at named stages: onRequest (first, before parsing), preParsing, preValidation, preHandler (after validation), preSerialization, onSend (before response), onResponse (after). Use fastify.addHook('preHandler', fn) or route-level hooks.

Q5. How do you implement JWT authentication in Fastify?

A: @fastify/jwt decorates the instance with fastify.jwt.sign(payload) and fastify.jwt.verify(token). Register fastify.authenticate as a preHandler hook on protected routes. The plugin reads the Bearer token from the Authorization header automatically.

Q6. How does Fastify's response serialization work?

A: Define a response JSON Schema in the route schema ({schema:{response:{200:{...}}}}). Fastify generates a compiled serializer with fast-json-stringify at startup. This is ~3x faster than JSON.stringify and strips fields not in the schema, preventing data

Q7. How do you add security headers and rate limiting in Fastify?

A: @fastify/helmet sets security headers (CSP, X-Frame-Options). @fastify/rate-limit tracks requests per IP using in-memory or Redis counters and returns 429 when limits are exceeded. Both are registered as plugins with fastify.register().

Q8. What are Fastify decorators and how do you use them?

A: fastify.decorate('redis', redisClient) adds a property to the Fastify instance. fastify.decorateRequest('user', null) adds a property to every Request object. This allows plugins to attach shared state (DB connections, auth context) accessible in handlers.

Q9. How do you integrate PostgreSQL in a Fastify application?

A: @fastify/postgres wraps pg pool and adds fastify.pg to the instance. For an ORM, use Prisma (prismaClient as a decorator) or Drizzle. Always close connections on shutdown using fastify.addHook('onClose', () => pool.end()).

Q10. How do you write tests for Fastify routes?

A: fastify.inject({method:'POST', url:'/users', payload:{name:'Alice'}}) sends a request in-process without a TCP connection. It resolves with a LightMyRequest response. Use Jest or tap assertions on response.statusCode and JSON.parse(response.body).