Fiber - Practice Sheet

10 Q&A on Fasthttp, routing, middleware, context, and testing

Q1. What is Fiber and why is it built on Fasthttp instead of net/http?

A: Fasthttp reuses request/response objects from a pool and avoids allocations. It is ~10x faster than net/http for raw throughput. Fiber wraps Fasthttp with an Express-inspired API, trading standard library compatibility for maximum performance.

Q2. How do you define parameterized routes in Fiber?

A: app.Get('/users/:id', handler) captures :id as c.Params('id'). Wildcards: /files/*filepath. Optional: /users/:id?. Fiber's routing is case-insensitive by default. Grouped routes: api := app.Group('/api') api.Get('/users', handler).

Q3. How do you write and chain middleware in Fiber?

A: A middleware returns nothing and uses c.Next() to pass control: app.Use(func(c *fiber.Ctx) error { fmt.Println('before'); err := c.Next(); fmt.Println('after'); return err }). app.Use('/api', authMiddleware) scopes to a prefix.

Q4. What does the Fiber Ctx object provide?

A: c.Params('id'), c.Query('q'), c.Body() (raw bytes), c.BodyParser(&req) (JSON/form decode), c.Locals('user', val) / c.Locals('user') to pass data, c.JSON(data) to respond with JSON, c.Status(400).SendString('Bad') for explicit status + body.

Q5. How does Fiber parse and validate request bodies?

A: c.BodyParser(&payload) deserializes the request body into a struct based on Content-Type (JSON, XML, form). Fiber uses encoding/json internally. For validation, pair BodyParser with go-playground/validator.v10 and call validate.Struct(payload).

Q6. What are the tradeoffs of using Fasthttp (and thus Fiber)?

A: Fasthttp does not implement net/http interfaces, so standard net/http middleware (Negroni, Alice) and testing helpers (httptest.NewRecorder) are incompatible. Use app.Test() for testing. Libraries expecting *http.Request cannot be used directly.

Q7. How do you implement JWT authentication middleware in Fiber?

A: Use gofiber/contrib/jwt. Register jwtware.New(jwtware.Config{SigningKey: []byte(secret)}) on a Group. It validates the Bearer token and stores the parsed jwt.Token in c.Locals('user'). Retrieve claims with c.Locals('user').(*jwt.Token).Claims.

Q8. How do you connect Fiber to PostgreSQL?

A: Use jackc/pgx or gorm with a PostgreSQL driver. Create a pgxpool.Pool or gorm.DB at startup and store it in the Fiber App Locals: app.Use(func(c *fiber.Ctx) error { c.Locals('db', pool); return c.Next() }). Handlers retrieve it with c.Locals('db').

Q9. How do you test Fiber handlers?

A: Use app.Test(req, timeout). Build a standard *http.Request: req, _ := http.NewRequest('GET', '/users/1', nil). Fiber processes it in-process and returns a *http.Response. Assert resp.StatusCode and read body with ioutil.ReadAll(resp.Body).

Q10. What is Fiber's Recover middleware and why is it important?

A: Recover catches panics in handlers and middleware, logs the stack trace, and returns a 500 response instead of crashing the goroutine. Without it, a panic in a handler would terminate the goroutine and could leak connections. Register it first with app.Use(recover.New()).