Gin - Practice Sheet

10 Q&A on routing, middleware, binding, error handling, and testing

Q1. What is Gin and what performance advantages does it have?

A: Gin is a Go HTTP framework built on HttpRouter, a radix-tree router that resolves routes in O(k) where k is the URL depth. It has zero memory allocation per request on the hot path and benchmarks at ~50,000 req/s, significantly faster than standard net/http mux.

Q2. How do you define route groups and apply group-scoped middleware?

A: api := r.Group('/api', authMiddleware). Inside, api.GET('/users', handler) inherits authMiddleware. Nesting groups further: admin := api.Group('/admin', adminOnly). This scopes middleware without repeating it on every individual route.

Q3. How do you write custom Gin middleware?

A: A middleware is a HandlerFunc: func MyMiddleware() gin.HandlerFunc { return func(c *gin.Context) { // before; c.Next(); // after } }. Call c.Abort() or c.AbortWithStatus(401) to stop the chain without calling Next.

Q4. How does JSON binding and validation work in Gin?

A: Define a struct with json and binding tags: Name string `json:'name' binding:'required'`. Call c.ShouldBindJSON(&req). On validation failure ShouldBindJSON returns an error; c.JSON(400, gin.H{'error': err.Error()}) sends the response. Gin uses go-playground/validator.

Q5. How does Gin's Context (c *gin.Context) work?

A: c wraps request and response. c.Param('id') reads path params, c.Query('q') reads query params, c.ShouldBindJSON reads body. c.JSON(200, obj) serializes and writes JSON. c.Set('user', u) and c.Get('user') pass values through the middleware chain.

Q6. How do you handle errors centrally in Gin?

A: Append errors with c.Error(err). After all middleware, a final middleware reads c.Errors and formats a JSON response. Alternatively use c.AbortWithStatusJSON(500, gin.H{'error': err.Error()}) inside a recovery middleware registered with gin.Recovery().

Q7. How do you serve static files and HTML templates in Gin?

A: r.Static('/assets', './static') serves a directory. r.StaticFile('/favicon.ico', './favicon.ico') serves one file. Load templates with r.LoadHTMLGlob('templates/**/*') and render with c.HTML(200, 'index.tmpl', gin.H{'title': 'Home'}).

Q8. How do you configure TLS in a Gin server?

A: Replace r.Run(':8080') with r.RunTLS(':443', 'cert.pem', 'key.pem'). Alternatively pass a pre-configured *tls.Config via r.RunListener(tls.NewListener(l, tlsConf)). Use autotls or certmagic packages for Let's Encrypt automation.

Q9. How do you use gin.Context.Set/Get to pass data between middleware and handlers?

A: c.Set('userID', 42) stores a value in the context for the current request lifecycle. c.Get('userID') returns (interface{}, bool). Use c.MustGet('userID').(int) for typed access. This is the correct way to share auth data from middleware to handlers.

Q10. How do you write tests for Gin routes?

A: Create a test recorder: w := httptest.NewRecorder(). Build a request: req, _ := http.NewRequest('GET', '/users/1', nil). Call router.ServeHTTP(w, req). Assert w.Code and w.Body.String(). No server port needed tests run fully in-process.