

THE STACK & A REQUEST'S LIFE

SwiftNIO (event loops, HTTP/1.1 + 2, TLS) → Vapor (Application, router, Request/Response, Content) → Fluent, Leaf, Queues, JWT, Redis, APNs. One async process on macOS and Linux.

accept → app middleware (first added = outermost)
→ route match → group middleware → handler
→ encode → middleware back out → write

TOOLING & COMMAND MAP

```
swiftly install latest      # Swift 6.4 toolchain
brew install vapor         # Toolbox 20
vapor new Bookshelf
swift build [-c release]; swift test
swift run App serve --env production \
  --hostname 0.0.0.0 --port 8080 # default
swift run App routes | migrate [--revert] --yes
swift run App queues [--scheduled]
swift run App seed --books 6
```

Env: --env, else VAPOR_ENV, else development; .env files, Environment.get("X"), LOG_LEVEL.

PACKAGE, LAYOUT, ENTRYPPOINT

```
// swift-tools-version:6.3
platforms: [.macOS(.v13)]
.package(url: "https://github.com/vapor/vapor.git",
  from: "4.122.2") // + fluent, leaf
.executableTarget(name: "App", dependencies: [
  .product(name: "Vapor", package: "vapor")])
.testTarget(name: "AppTests", dependencies: ["App",
  .product(name: "VaporTesting", package: "vapor")])

@main enum Entrypoint {
  static func main() async throws {
    var env = try Environment.detect()
    try LoggingSystem.bootstrap(from: &env)
    let app = try await Application.make(env)
    do { try await configure(app)
      try await app.execute() }
    catch { app.logger.report(error: error)
      try? await app.asyncShutdown(); throw error }
    try await app.asyncShutdown() } }

func configure(_ app: Application) async throws {
  app.databases.use(.postgres(url: url), as: .psql)
  app.migrations.add(CreateBook())
  app.views.use(.leaf)
  try routes(app) }
```

Sources/App/: Controllers, Models, Migrations, DTOs, Jobs, Commands, Middleware, Services; Resources/Views, Public. Share DTOs with iOS in a package without Vapor.

SWIFT 6 CONCURRENCY RULES

- Handlers are @Sendable async throws; bridge old futures with .get(). • Models: final class ... : Model, @unchecked Sendable. • Never block an event loop: req.fileio, app.threadPool.runIfActive or a job. • Shared state: an actor or a Sendable value in app.storage; no mutable globals. • Services are Application/Request extensions; LifecycleHandler for boot and shutdown.

ROUTING & CONTROLLERS

```
app.get("books", ":id") { req async throws
  → BookDTO in
  let id = try req.parameters.require("id",
    as: UUID.self)
  ... // post put patch delete; "*", "**"
app.on(.POST, "up",
  body: .collect(maxSize: "10mb"))
let api = app.grouped("api", "v1")
struct BookController: RouteCollection {
  func boot(routes: any RoutesBuilder) throws {
    let b = routes.grouped("books")
    b.get(use: index); b.post(use: create) } }
try app.register(collection: BookController())
```

Default body limit 16 KB; body: .stream for large uploads.

CONTENT, VALIDATION, ERRORS

```
struct NewBook: Content, Validatable {
  var title: String; var year: Int
  static func validations(_ v: inout Validations) {
    v.add("title", as: String.self, is: !.empty)
    v.add("year", as: Int.self,
      is: .range(1450...2100)) } }
try NewBook.validate(content: req) // 400
let b = try req.content.decode(NewBook.self)
let f = try req.query.decode(Filter.self)
throw Abort(.notFound, reason: "no such book")

Wrong Content-Type → 415. Custom errors conform to
  AbortError.
```

MIDDLEWARE, SERVER, SERVICES

```
struct Timing: AsyncMiddleware {
  func respond(to req: Request, chainingTo next:
    any AsyncResponder) async throws → Response {
    try await next.respond(to: req) } }
app.middleware.use(Timing(), at: .beginning)
app.http.server.configuration.port = 8080
struct SettingsKey: StorageKey {
  typealias Value = Settings }
let r = try await req.client.get("https://x.example")

Defaults: RoutelLoggingMiddleware, then
  ErrorMiddleware. Group middleware runs only on matched
  routes. Built-ins: FileMiddleware (Public/, ETag, ranges),
  CORSMiddleware, SessionsMiddleware. Secrets from the
  environment, fail fast.
```

FLUENT: MODELS, DTOS, MIGRATIONS

```
@ID @Field @OptionalField @Parent @Children(for:)
@Siblings(through: from: to:) @Timestamp @Enum
@Group; drivers: Postgres, SQLite, MySQL, Mongo.

final class Book: Model, @unchecked Sendable {
  static let schema = "books"
  @ID(key: .id) var id: UUID?
  @Field(key: "title") var title: String
  @Parent(key: "author_id") var author: Author
  init() {}
  func toDTO() → BookDTO {
    .init(id: id, title: title) } }
struct BookDTO: Content {
  var id: UUID?; var title: String }
struct CreateBook: AsyncMigration {
  func prepare(on db: any Database) async throws {
    try await db.schema("books").id()
      .field("title", .string, .required)
      .field("author_id", .uuid, .required,
        .references("authors", "id",
          onDelete: .cascade))
      .create() }
  func revert(on db: any Database) async throws {
    try await db.schema("books").delete() } }
```

QUERIES, TRANSACTIONS, RAW SQL

```
try await Book.query(on: req.db)
  .filter(\.year > 2000).sort(\.title)
  .with(\.author).paginate(for: req)
try await req.db.transaction { tx in
  try await a.save(on: tx)
  try await b.save(on: tx) }
try await (req.db as! any SQLiteDatabase)
  .raw("SELECT ... WHERE year > \(bind: y)").all()

Keyset pagination for big tables; @Timestamp(on: .delete)
= soft delete; ModelMiddleware hooks. MongoDB: fluent-
mongo-driver or MongoKitten.
```

LEAF, HTML DSLS, SESSIONS, REDIS

```
#extend("base"): #export("body"):
  #for(b in books):<li>#(b.title)</li>#endfor
#endexport #endextend
try await req.view.render("home", ctx)
app.leaf.tags["price"] = PriceTag() // LeafTag
return HTMLResponse { CatalogPage(books: books) }
app.sessions.use(.redis)
req.session.data["uid"] = id
try await req.cache.set("k", to: v,
  expiresIn: .minutes(5))

#(x) is escaped, #unsafeHTML is not. Elementary + htmx:
typed HTML fragments. Forms: POST-redirect-GET, CSRF
token, re-render with errors.
```

AUTHENTICATION, JWT, PASSWORDS

Basic → login only · Bearer ModelTokenAuthenticatable, hashed tokens → API clients · Session → browsers · JWT → stateless services · Sign in with Apple → verify the identity token.

```
let priv = app.grouped(User.authenticator(),
  User.guardMiddleware())
// 401 when not logged in
let user = try req.auth.require(User.self)
let hash = try await req.password.async.hash(pw)
struct Claims: JWTPayload { var sub: SubjectClaim
  var exp: ExpirationClaim
  func verify(using alg: some JWTAlgorithm)
    async throws {
    try exp.verifyNotExpired() } }
let t = try await req.jwt.sign(claims)
let c = try await req.jwt.verify(as: Claims.self)

Authorization: check ownership per resource; 401 vs 403 vs
  404; guard every write route.
```

SECURITY CHECKLIST

- ☐ CSRF token + Origin check + SameSite cookies
- ☐ CSP, HSTS, nosniff
- ☐ rate limits
- ☐ CORS allow-list
- ☐ body size caps
- ☐ bcrypt, constant-time compares
- ☐ TLS
- ☐ secrets from env
- ☐ short-lived access + rotating refresh tokens
- ☐ OAuth state + PKCE
- ☐ OWASP Top 10.

WEBSOCKETS, SSE, UPLOADS

```
app.websocket("live") { req, ws in
  ws.onText { ws, text in ws.send(text) } }
Response(headers: ["Content-Type":
  "text/event-stream"],
  body: .init(managedAsyncStream: { w in
    try await w.write(.buffer(
      .init(string: "data: hi\n\n"))) } })
struct Up: Content { var file: File } // multipart

Fan out with an actor hub; SSE resumes with Last-Event-ID.
  Uploads: cap size, check type, generate the file name.
```

JOBS, SCHEDULES, APNS, EMAIL, COMMANDS

```
struct Mail: AsyncJob { typealias Payload = MailMsg
  func dequeue(_ c: QueueContext, _ p: MailMsg)
    async throws { ... } }
try app.queues.use(.redis(url: url))
app.queues.add(Mail())
try await req.queue.dispatch(Mail.self, msg)
app.queues.schedule(Nightly()).daily().at(2, 0)
app.asyncCommands.use(Seed(), as: "seed")

APNs: app.apns.configure(.jwt(...)) with a .p8 key; drop
  tokens on 410. Email: an EmailSender over SendGrid,
  Mailgun or SMTP, sent from a job.
```

OPENAPI & IOS CLIENTS

openapi.yaml → OpenAPIGenerator plugin → implement APIProtocol → registerHandlers(on: VaporTransport(routesBuilder: app)); generate the client from the same file. Or share a Codable, Sendable DTO package; call it from an actor with URLSession, tokens in the Keychain.

TESTING

```
@Test func health() async throws {
  try await withApp(configure: configure) { app in
    try await app.testing()
      .test(.GET, "health") { res in
        #expect(res.status == .ok) } } }

.testing env, SQLite .memory,
  autoMigrate/autoRevert, queues.use(.asyncTest),
  .running(port: 0) for a real socket. XCTVapor is legacy.
```

LOGGING, TRACING, METRICS

```
req.logger.info("...", metadata: [:]);
TracingMiddleware and withSpan; swift-metrics Counter;
export via swift-otel or Prometheus; W3C traceparent.
```

DOCKER & DEPLOYMENT

```
FROM swift:6.3-noble AS build
RUN swift build -c release --static-swift-stdlib
FROM ubuntu:noble # copy /staging, USER vapor
ENTRYPOINT ["./App"]
CMD ["serve", "--env", "production",
  "--hostname", "0.0.0.0"]

☐ /health + /ready ☐ migrate --yes once ☐ one
scheduler ☐ non-root ☐ SIGTERM → drain ☐ logs to stdout ☐
trust X-Forwarded-For only from the proxy. Fly.io, VPS +
  systemd, ECS, Cloud Run, Kubernetes.
```

PERFORMANCE, MICROSERVICES, ECOSYSTEM

Stateless instances scale out (sessions in Redis); measure with wrk/oha. Split services only at real boundaries: a req.client gateway, audience-bound service JWTs or mTLS, timeouts, bounded retries. Hummingbird 2 is the lighter alternative; Kitura and Perfect are dormant.

CHANGED SINCE OLDER TUTORIALS

Droplet, Services → Application + extensions; main.swift → @main, Application.make, execute()

EventLoopFuture chains → async/await; models as Content → DTOs; .sql(raw:) → \(\bind:)

Vapor Cloud, Heroku free tier → Docker, Fly.io. Vapor 5 (beta.4, not GA): swift-http-server, macro routes, swift-configuration