00 · Repository Guide

Flutter Practice Notes

Contents

00 · Repository Guide

Introduction

This is the **master index and operating manual** for the Flutter Architect Handbook — a 60-module curriculum that takes you from writing your first `main()` to designing an enterprise super-app and passing a Staff-level system design interview.

This guide file tells you **what exists, in what order to read it, and to what standard every file is written.** Read it once fully, then use it as your map.

Why this repository exists

Most Flutter learning material stops at "here's a `ListView`, here's `setState`." That produces developers who can ship a screen but cannot:

- explain **why** a widget rebuild is cheap but a layout pass is not,
- reason about **isolates vs async** when the UI jank starts,
- defend an **architecture** to a skeptical staff engineer,
- or scale a codebase past ~50 screens without it collapsing.

This handbook closes that gap. It is deliberately **internals-first and WHY-first**, because senior engineering is about understanding tradeoffs, not memorizing APIs.

Who this is for

Level	You are here if...	Start at
Absolute beginner	New to Dart/Flutter	Module 01
Junior → Mid	Can build screens, shaky on internals/architecture	Module 06, revisit 01–05
Senior candidate	Building features, prepping for product-company interviews	Modules 10, 40, 48, 55
Architect	Designing systems, mentoring, making tech-selection calls	Modules 45–58

The complete module index

Legend for **Level**: 🟢 Foundations · 🔵 Flutter Core · 🟡 Applied/Integrations · 🟠 Architecture · 🔴 Senior/Architect.

#	Module	Level	What it makes you able to do
00	**Repository Guide**	—	Navigate the handbook, follow the standard
01	Dart Fundamentals	🟢	Read/write idiomatic, null-safe Dart
02	Advanced Dart	🟢	Async, isolates, generics, extensions, streams
03	Object Oriented Programming	🟢	Model domains with encapsulation & polymorphism
04	SOLID Principles	🟢	Write code that survives change
05	Design Patterns	🟢	Apply Creational/Structural/Behavioral patterns
06	Flutter Fundamentals	🔵	Understand what Flutter *is* and how it runs
07	Widgets	🔵	Compose UI from the widget catalog
08	Widget Lifecycle	🔵	Reason about `State` and rebuilds
09	Rendering Pipeline	🔵	Explain build→layout→paint→composite→raster
10	Flutter Architecture	🔵	Map framework/engine/embedder layers
11	State Management	🔵	Choose & implement setState→Riverpod→BLoC
12	Navigation	🔵	Imperative & declarative navigation
13	Routing	🔵	Deep links, guards, nested routers
14	Dependency Injection	🟠	Wire dependencies testably
15	Local Storage	🟡	Preferences, secure storage, files
16	Networking	🟡	REST/GraphQL/gRPC/WebSockets
17	Authentication	🟡	Sessions, tokens, OAuth, biometrics
18	Firebase	🟡	Auth, Firestore, Storage, Functions, FCM
19	Offline First	🟠	Sync, conflict resolution, cache strategy
20	Database	🟡	SQLite/Drift/Isar/Hive modeling
21	Performance	🔴	Kill jank, optimize memory & frames
22	Animations	🔵	Implicit/explicit/physics-based motion
23	Custom Painting	🔵	`CustomPainter`, canvas, shaders
24	Responsive UI	🔵	Layouts across sizes
25	Adaptive UI	🔵	Platform-correct UI (Material/Cupertino)
26	Platform Channels	🟡	Bridge to native code
27	Native Android	🟡	Kotlin interop, plugins
28	Native iOS	🟡	Swift interop, plugins
29	Device Features	🟡	Camera, location, sensors, BLE, NFC

#	Module	Level	What it makes you able to do
30	Google Maps	🟡	Maps, markers, geolocation
31	Payments	🟡	Razorpay/Stripe/UPI integration
32	Notifications	🟡	Local & push, deep-linked
33	Background Services	🟡	WorkManager, isolates, background fetch
34	File Handling	🟡	Upload/download, ZIP, CSV, encryption
35	PDF	🟡	Generate & render PDFs
36	Excel	🟡	Read/write spreadsheets
37	Security	🔴	Storage, transport, obfuscation, App Check
38	Error Handling	🟠	Failure modeling, `Result`, global handlers
39	Logging	🟠	Structured logs, observability hooks
40	Clean Architecture	🟠	Layers, boundaries, dependency rule
41	MVC	🟠	The classic separation
42	MVP	🟠	Presenter-driven UI
43	MVVM	🟠	ViewModel + reactive binding
44	Feature First Architecture	🟠	Organize by feature, not by layer
45	Modular Architecture	🔴	Multi-package, enforced boundaries
46	Domain Driven Design	🔴	Entities, value objects, aggregates
47	Scalable Applications	🔴	Patterns for 100+ screen apps
48	System Design	🔴	HLD/LLD for mobile systems
49	Testing	🟠	Unit/widget/integration/golden, TDD
50	CI CD	🔴	Pipelines, signing, store automation
51	Deployment	🟡	Play Store/App Store/web/desktop
52	Monitoring	🔴	Crashlytics, analytics, performance traces
53	Flutter Web	🟡	Web renderers, SEO, deployment
54	Flutter Desktop	🟡	Windows/macOS/Linux targets
55	Flutter Interview Preparation	🔴	Topic- & company-wise question banks
56	Machine Coding Rounds	🔴	Timed build challenges
57	Enterprise Projects	🔴	Full production-grade builds
58	Senior Architect Notes	🔴	Tech selection, mentoring, decision records
59	Computer Science Foundations	🔴	OS, memory, networking, DNS, HTTP/TLS, DB, caching, crypto — the substrate under Flutter

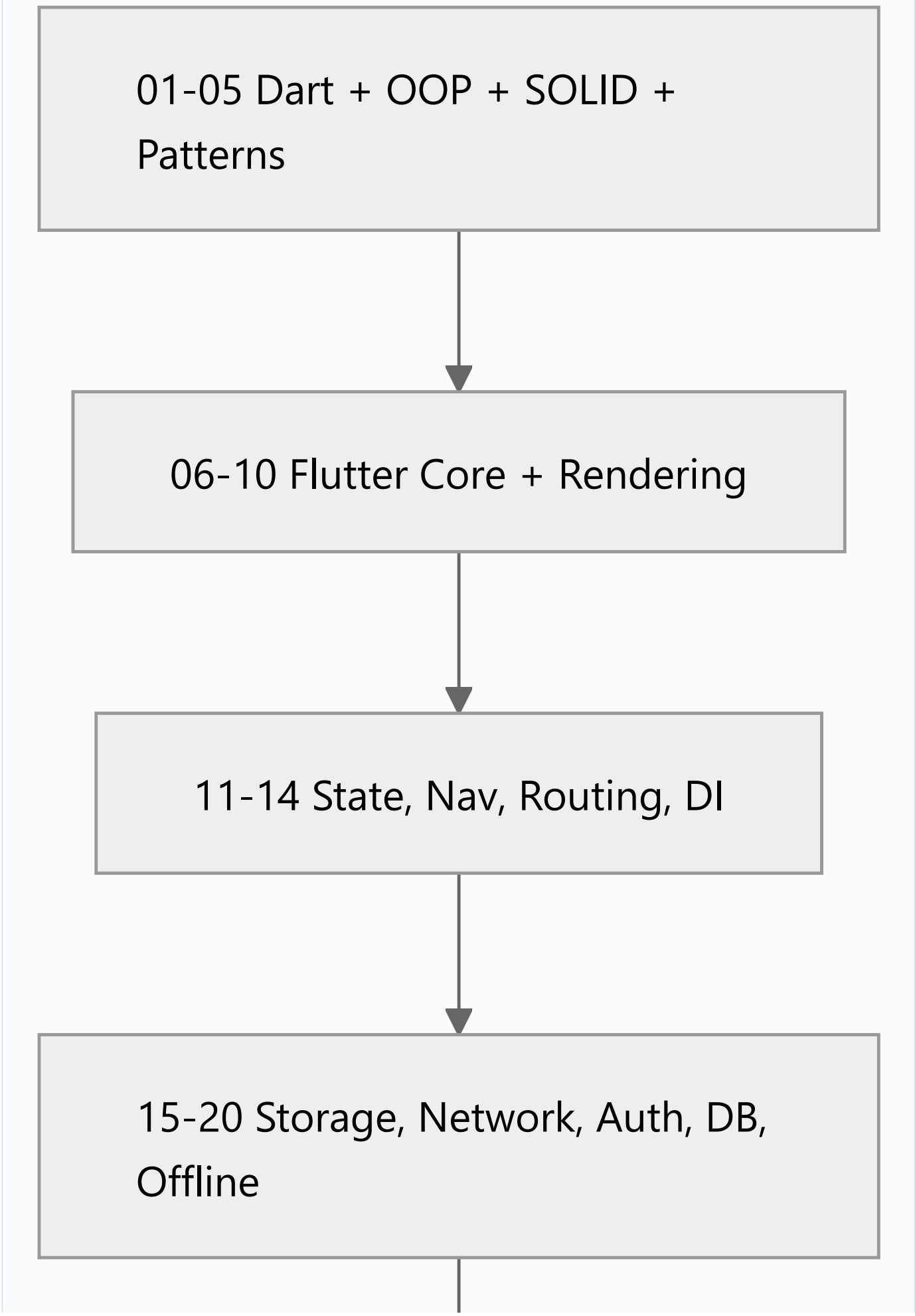
#	Module	Level	What it makes you able to do
60	Software Engineering Practices	🔴	Git, Agile, code review, RFCs, delivery & release, observability as a discipline

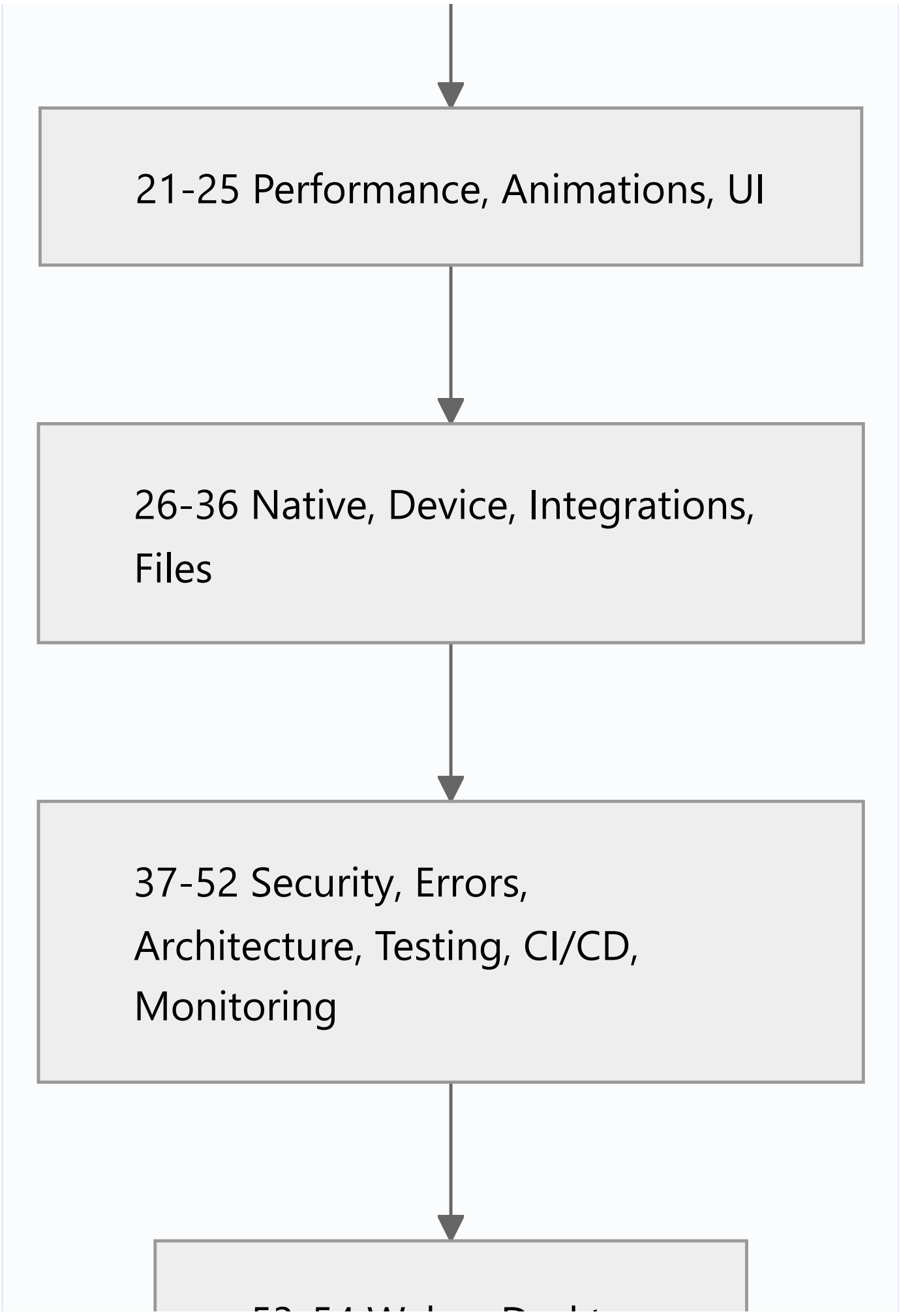
> **Modules 59–60** are the *platform-agnostic* CS and software-engineering layer a senior/staff engineer is expected to know beyond Flutter APIs. They teach each concept vendor-neutrally and cross-link to the Flutter-specific modules rather than duplicating them.

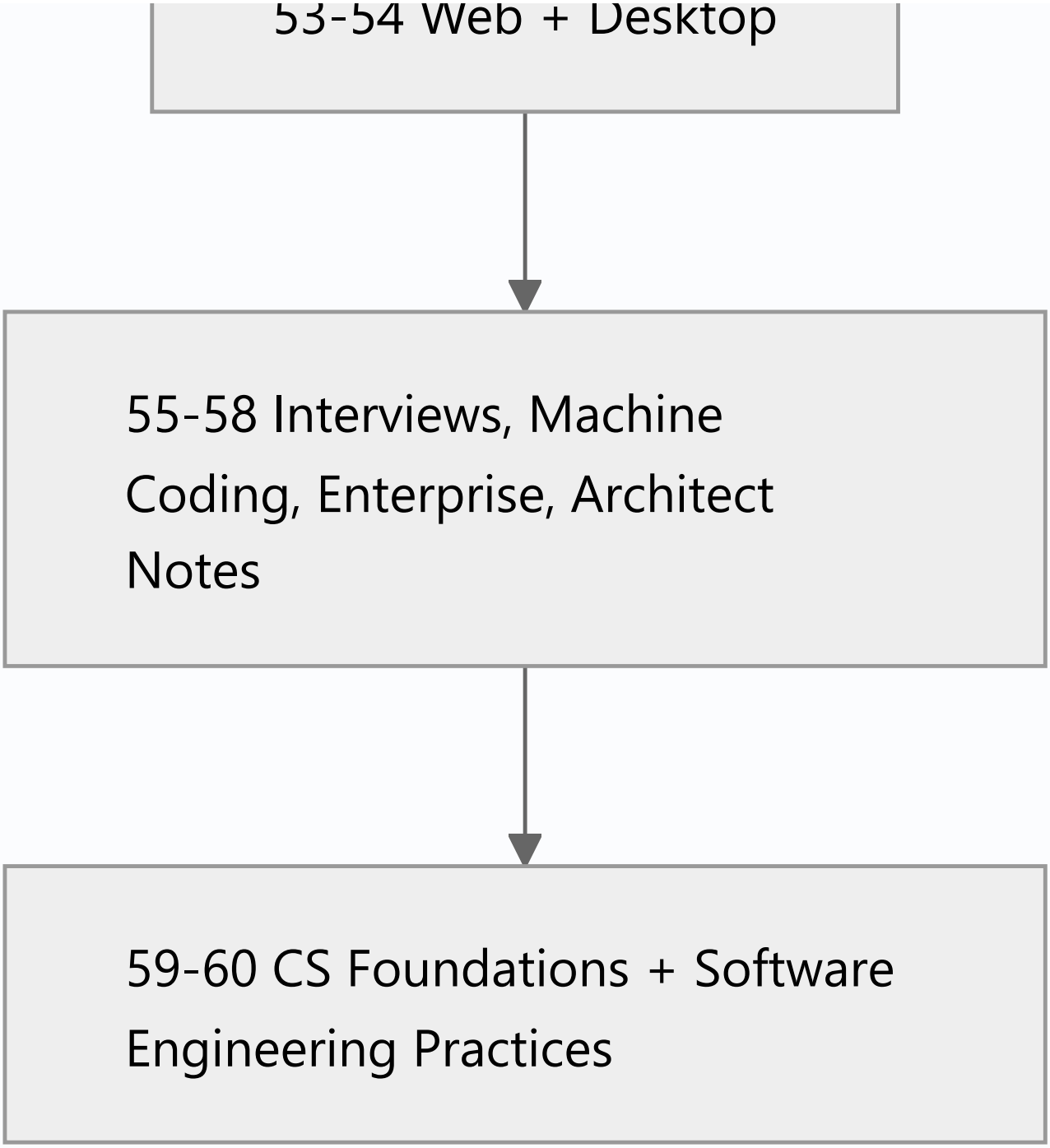
> Each module folder contains a `README.md` (module overview + file index) plus one `.md` per topic, every file following `FILE_TEMPLATE.md`.

Recommended reading order

Read numerically on the **first pass** — the modules are dependency-ordered. For targeted prep, jump using the `LEARNING_PATH.md` phase map.


01-05 Dart + OOP + SOLID + Patterns
06-10 Flutter Core + Rendering
11-14 State, Nav, Routing, DI
15-20 Storage, Network, Auth, DB, Offline


21-25 Performance, Animations, UI
26-36 Native, Device, Integrations, Files
37-52 Security, Errors, Architecture, Testing, CI/CD, Monitoring



> **Modules 59–60 are cross-cutting** — read them whenever you want, but they land hardest after Module 21, once you have enough Flutter context to see the substrate underneath it.

How every topic file is structured

To guarantee that **reading a single file gives complete understanding**, every topic `.md` follows the fixed section order defined in `FILE_TEMPLATE.md`:

`Title → Introduction → Why this concept exists → Real-world analogy → Problem Statement → Internal Working → Memory Representation → Compiler Behavior → Runtime Behavior → Flutter Engine Behavior → Dart VM Behavior → Examples → Diagrams → Common Mistakes → Best Practices → Performance → Advantages → Disadvantages → Interview Questions → Senior Engineer Tips → Architect Perspective → Summary → Revision Notes → Practice Questions → Coding Questions → Mini Project`

Sections marked *(if applicable)* — Flutter Engine / Dart VM — are included only when relevant, and the file says so explicitly rather than padding.

Production & writing standards

All content adheres to `STANDARDS.md`, which enforces:

- **WHY before HOW**, internals wherever they exist.
- **Mermaid diagrams** for anything with structure or flow.
- **Tables** for comparisons, tradeoffs, and API surfaces.
- **Runnable, null-safe, lint-clean** code samples following Effective Dart.
- **Interview questions + coding exercises + a mini project** in every topic.
- **References** to official dart.dev / docs.flutter.dev.

Capstone strategy

Every **major module** ships four capstones — Beginner, Intermediate, Advanced, Enterprise — and the flagship projects (Module 57) are built **five ways** (Provider, Riverpod, GetX, BLoC, Cubit) and compared on folder structure, performance, scalability, boilerplate, maintainability, and interview value. See `LEARNING_PATH.md`.

Summary

- This handbook is an internals-first, WHY-first path from beginner to architect across **60 modules** (plus this guide), including the platform-agnostic CS and software-engineering layer in modules 59–60.
- Read numerically first; revise by index; drill with the phase map.
- Every file is self-contained and follows one strict template and quality bar.

Revision Notes

- 60 modules, 5 levels (🟢 🔵 🟡 🟠 🔴); modules 59–60 add the vendor-neutral CS + software-engineering foundations.

- One template for every topic file; one standards doc for quality.
- Capstones at four difficulties; flagship projects built five ways.
- First pass = numeric order; interviews = Revision Notes + Module 55/56.

Next step

Folder **00 is complete**. On your confirmation I will author `01 Dart Fundamentals` in full (README + one file per topic: variables, data types, operators, functions, collections, null safety, records/patterns, generics, and the rest listed in the Dart coverage spec), each following the template.

File Template (Mandatory Structure)

Introduction

Every **topic** `.md` file in this handbook uses the exact section order below. This guarantees the promise that *reading a single file gives complete understanding*. Copy this template when authoring a new topic file.

> Module `README.md` files are exempt (they are indexes). All *topic* files follow this template. Sections marked *(if applicable)* are included only when the topic genuinely touches them — and the file states "Not applicable — because …" rather than being dropped silently or padded.

The template (copy below the line)

```
# {Title}

> One-sentence definition a senior could quote verbatim.

## Introduction

What this is, in 2–4 sentences. No fluff.

## Why this concept exists

The problem it solves and what the world looked like before it. WHY before HOW.

## Real-world analogy

A concrete, non-code analogy that makes the mental model click.

## Problem Statement

A precise scenario the reader will solve by the end of the file.

## Internal Working

How it actually works under the hood — data structures, algorithms, the flow.

Use a Mermaid diagram here whenever there is structure or a sequence.

## Memory Representation

Stack vs heap, references vs values, what lives where, allocation/retention.
```

```
## Compiler Behavior

What the Dart compiler does: inference, canonicalization, tree-shaking, errors.

## Runtime Behavior

What happens while executing: dispatch, exceptions, scheduling, timing.

## Flutter Engine Behavior (if applicable)

Skia/Impeller, compositor, raster thread, platform thread interaction.

## Dart VM Behavior (if applicable)

JIT vs AOT, isolates, GC generational behavior, snapshots.

## Examples

Progressive, runnable, null-safe, lint-clean samples — from minimal to real.

## Diagrams

Mermaid diagrams (flow/sequence/class/state) that visualize the concept.

## Common Mistakes

Real bugs beginners and mids hit, each with the fix.

## Best Practices

Effective-Dart / official-guidance-aligned recommendations.

## Performance

Cost model, hotspots, benchmarks or Big-O where meaningful.

## Advantages

When and why to reach for this.

## Disadvantages

Honest tradeoffs and when NOT to use it.

## Interview Questions

6–12 questions with crisp model answers, tagged by difficulty.

## Senior Engineer Tips

Non-obvious knowledge that separates senior from mid.
```

```
## Architect Perspective

System-level implications, tech-selection tradeoffs, scaling impact.


## Summary

Tight recap of the whole file.


## Revision Notes

Bulleted, skimmable, interview-morning-ready.


## Practice Questions

Conceptual questions to self-test (no code required).


## Coding Questions

Hands-on exercises with clear acceptance criteria.


## Mini Project

A small, complete build applying the concept end-to-end.
```

Authoring rules for each section

Section	Rule
Title + quote	The quote must be a standalone, correct one-liner.
Why this concept exists	Always precede HOW. If you can't state the problem, don't write the file.
Internal Working	Include a diagram if there is any structure/flow.
Compiler/Runtime/Engine/VM	Prefer facts you can cite; mark *Not applicable* explicitly when so.
Examples	Must compile against current stable Dart/Flutter and pass `dart analyze`.
Interview Questions	Provide the answer, not just the prompt. Tag 🟢/🟡/🔴.
Common Mistakes	Show the wrong code and the corrected code.
Mini Project	Must be completable in under ~2 hours by a reader at that level.

Quality bar checklist (per file)

- [ ] WHY stated before HOW.
- [ ] At least one Mermaid diagram (if the topic has structure/flow).
- [ ] At least one comparison table.
- [ ] Code is null-safe and lint-clean.

- [ ] ≥ 6 interview questions with answers.
- [ ] Coding exercise + mini project present.
- [ ] Revision Notes are skimmable in under 60 seconds.
- [ ] Official docs referenced where relevant.

Summary

- One fixed section order for all topic files.
- *(if applicable)* sections are declared, never silently dropped.
- A per-file checklist enforces the quality bar.

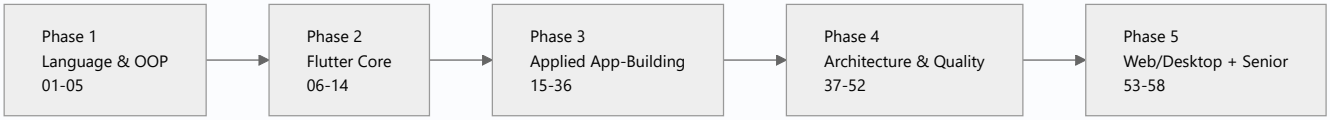
Learning Path — Foundations to Architect

Introduction

The 58 modules are dependency-ordered, but real learning happens in **phases** with checkpoints. This document maps modules into a phased roadmap with rough time estimates, prerequisite gates, and the capstone strategy.

> Time estimates assume ~10 focused hours/week. Halve them if full-time, double them if casual. They measure *understanding + building*, not just reading.

The five phases



Phase 1 — Language & OOP foundations (Modules 01–05)

Goal: Write idiomatic, null-safe, well-structured Dart. Understand OOP, SOLID, and core patterns before touching Flutter.

Module	Est.	Gate to pass
01 Dart Fundamentals	15h	Explain `final` vs `const` vs `late`; use collections + null safety fluently
02 Advanced Dart	20h	Explain the event loop, microtask vs event queue, isolates, streams
03 OOP	12h	Model a domain with inheritance, mixins, composition; deep vs shallow copy
04 SOLID	10h	Refactor a violating class to satisfy each principle
05 Design Patterns	18h	Implement Factory, Strategy, Observer, Repository, Singleton in Dart

Phase checkpoint: Build a CLI or pure-Dart library (no Flutter) applying SOLID + one pattern.

Phase 2 — Flutter core & internals (Modules 06–14)

Goal: Understand how Flutter *renders and rebuilds*, and wire up state, navigation, and DI.

Module	Est.	Gate to pass
06 Flutter Fundamentals	10h	Explain widget/element/render trees
07 Widgets	15h	Compose complex layouts from the catalog
08 Widget Lifecycle	8h	Trace `initState` → `build` → `dispose`; when `didChangeDependencies` fires
09 Rendering Pipeline	12h	Explain build→layout→paint→composite→raster; constraints go down, sizes go up
10 Flutter Architecture	8h	Diagram framework/engine/embedder
11 State Management	25h	Implement the same feature in setState, Provider, Riverpod, BLoC
12 Navigation	8h	Imperative + declarative navigation
13 Routing	8h	Deep links, guards, nested navigators
14 Dependency Injection	8h	Wire `get_it`/Riverpod DI, keep it testable

Phase checkpoint: Beginner + Intermediate capstones (Todo, Notes, Weather).

Phase 3 — Applied app-building & integrations (Modules 15–36)

Goal: Ship real features — storage, networking, auth, device capabilities, files, payments.

Grouped tracks (do the tracks your target projects need first):

- **Data track:** 15 Local Storage · 16 Networking · 20 Database · 19 Offline First
- **Identity track:** 17 Authentication · 18 Firebase · 37 Security (preview)
- **UX track:** 22 Animations · 23 Custom Painting · 24 Responsive · 25 Adaptive
- **Native track:** 26 Platform Channels · 27 Android · 28 iOS · 29 Device Features · 30 Maps
- **Feature track:** 31 Payments · 32 Notifications · 33 Background · 34–36 Files/PDF/Excel

Est. **90–120h** total depending on tracks chosen.

Phase checkpoint: Advanced capstone (Chat App or Food Delivery) with real backend, offline cache, and auth.

Phase 4 — Architecture, quality & operations (Modules 37–52)

Goal: Make it enterprise-grade — architecture, testing, security, CI/CD, monitoring.

Track	Modules
Reliability	37 Security · 38 Error Handling · 39 Logging
Architecture	40 Clean · 41 MVC · 42 MVP · 43 MVVM · 44 Feature-First · 45 Modular · 46 DDD · 47 Scalable · 48 System Design
Quality & Ops	49 Testing · 50 CI/CD · 51 Deployment · 52 Monitoring

Est. **100h+**. This is the phase that separates senior from mid.

Phase checkpoint: Enterprise capstone (Banking or E-Commerce) with Clean Architecture + full test suite + CI/CD.

Phase 5 — Platforms & senior mastery (Modules 53–58)

Module	Focus
53 Flutter Web / 54 Desktop	Multi-platform delivery
55 Interview Preparation	Topic- & company-wise banks
56 Machine Coding Rounds	Timed builds
57 Enterprise Projects	Flagship builds (5-way state comparison)
58 Senior Architect Notes	Tech selection, ADRs, mentoring

Capstones

Every major module ships four capstones:

Tier	Scope	Example
Beginner	Single feature, one screen	Todo (local only)
Intermediate	Multi-screen + state + storage	Notes with search + tags
Advanced	Backend + auth + offline + animations	Chat app
Enterprise	Clean arch + tests + CI/CD + monitoring	Banking / E-commerce

Each capstone specifies: Requirements · Folder Structure · Architecture · UI Design · API · Database · Authentication · Error Handling · Offline Support · Caching · Animations · Testing · Deployment.

The 5-way flagship comparison

Flagship projects (Module 57) are built **five times** and compared:

Criterion	Provider	Riverpod	GetX	BLoC	Cubit
Folder structure					
Performance					
Scalability					
Boilerplate					
Maintainability					
Interview value					

(Filled in within Module 57 with measured results, not opinions.)

Interview-prep fast track

Short on time before an interview? Read in this order:

1. **Revision Notes** of Modules 01, 02, 08, 09, 11.
2. Module **40 Clean Architecture** + **43 MVVM**.
3. Module **48 System Design** + **55 Interview Preparation** (target company section).
4. Module **56 Machine Coding** — do two timed builds.

Summary

- Five phases: Language → Flutter Core → Applied → Architecture/Quality → Platforms/Senior.
- Each phase has a gate and a capstone checkpoint.
- Flagship projects are built five ways and compared on six axes.
- A dedicated fast track exists for imminent interviews.

Standards — Writing, Code, Diagrams, Production

Introduction

This document is the **quality contract** for the handbook. Every file is written to these standards so the repository reads as one coherent, production-grade body of knowledge rather than a pile of blog posts.

---

1. Writing standards

- **WHY before HOW, internals whenever they exist.** State the problem a concept solves before its API.
- **No dumbing down.** Simplify *explanation*, never the *concept*. If something is genuinely hard, say so and explain it properly.
- **Second person, active voice.** "You dispatch an event" not "an event is dispatched."
- **Define jargon on first use**, then use it freely.
- **Cite sources** — link dart.dev and docs.flutter.dev for authoritative claims; avoid unverifiable performance claims.

2. Code standards

Follow **Effective Dart** and these rules:

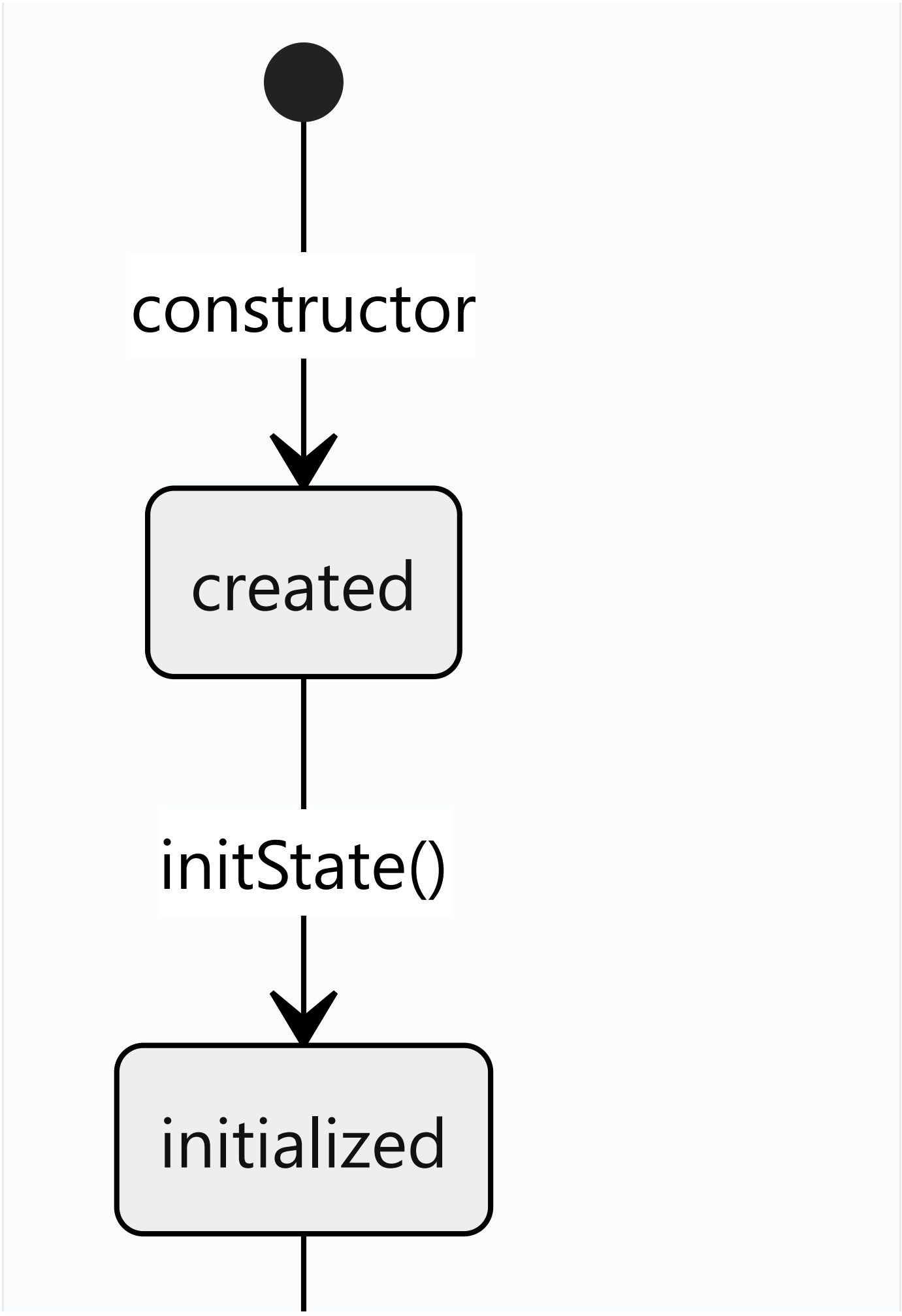
Rule	Requirement
Null safety	All code is sound-null-safe.
Linting	Passes `dart analyze` with `flutter_lints` / `lints`.
Const correctness	Use `const` constructors wherever possible.
Immutability	Prefer immutable models; document mutability when needed.
Naming	`lowerCamelCase` members, `UpperCamelCase` types, `_private` per library.
Formatting	`dart format` applied; no manual misalignment.
Error handling	No empty `catch`; model failures explicitly (see Module 38).
Comments	Explain *why*, not *what*; match surrounding density.

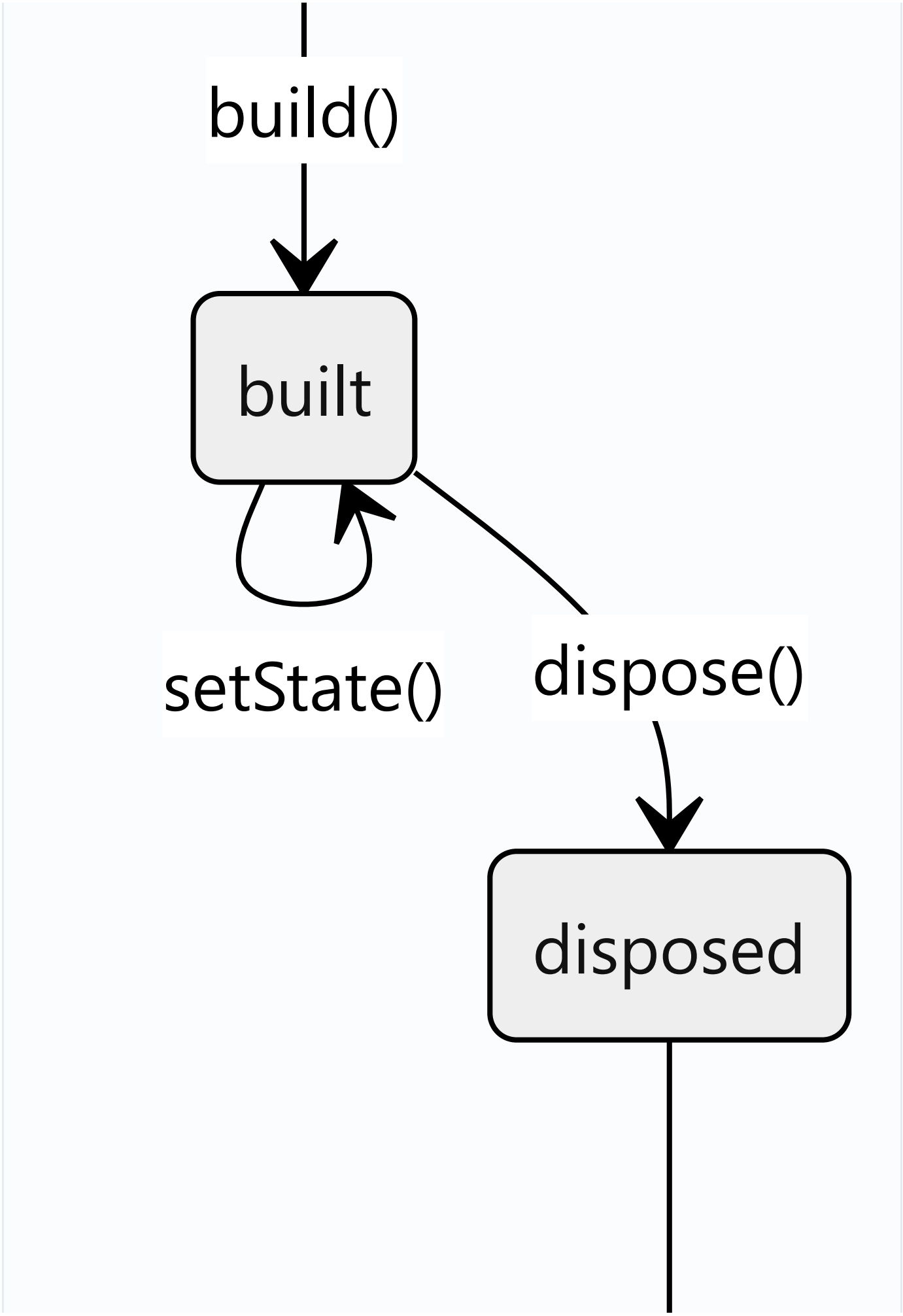
Every non-trivial snippet states its expected output as a comment or trailing note.

3. Diagram standards

- Use **Mermaid** so diagrams render in VS Code and GitHub.
- Choose the right type: `flowchart` (structure/flow), `sequenceDiagram` (interactions over time), `classDiagram` (relationships), `stateDiagram-v2` (lifecycles).
- Keep node labels short; put detail in surrounding prose.

Example — a lifecycle as a state diagram:


constructor
created
initState()
initialized


build()
built
setState()
dispose()
disposed

4. Table standards

Use tables for: comparisons, tradeoff matrices, API surfaces, decision guides, and "when to use X vs Y." Tables beat paragraphs for anything with ≥ 3 parallel items.

5. Production standards (for all sample apps)

- **Architecture:** layered, testable, dependency-rule respected (see Modules 40–47).
- **State:** explicit, predictable, no hidden globals.
- **Errors:** typed failures, user-safe messages, logged internals (Modules 38–39).
- **Security:** no secrets in code, secure storage for tokens, TLS enforced (Module 37).
- **Testing:** unit + widget + at least one integration/golden test (Module 49).
- **Performance:** const widgets, keys where needed, no rebuilds of static subtrees (Module 21).
- **Accessibility:** semantics labels, sufficient contrast, scalable text.
- **CI/CD:** format + analyze + test gates before merge (Module 50).

6. Anti-patterns to always flag

Anti-pattern	Why it's wrong	Correct approach
`setState` in `build`	Infinite rebuild loop	Trigger from callbacks/effects
Business logic in widgets	Untestable, coupled	Move to ViewModel/BLoC/UseCase
Blocking the UI isolate	Jank/ANR	Offload to isolate/async (Module 33)
Catch-all `catch (_) {}`	Swallows bugs	Model failures, log, rethrow selectively
God classes / God widgets	Unmaintainable	Compose, apply SRP (Module 04)
Secrets in source	Security breach	Env/secure storage, obfuscation

7. Google-recommended practices

Where Flutter/Google publish official guidance (e.g., app architecture guide, state management recommendations, performance best practices), the handbook aligns with it and links the source, noting any place where common community practice diverges and why.

Summary

- One writing voice, one code standard (Effective Dart), Mermaid diagrams, tables for comparisons.
- Sample apps meet production bars for architecture, security, testing, performance, a11y, CI/CD.
- Anti-patterns are called out explicitly with corrections.
- Claims are cited to official docs.