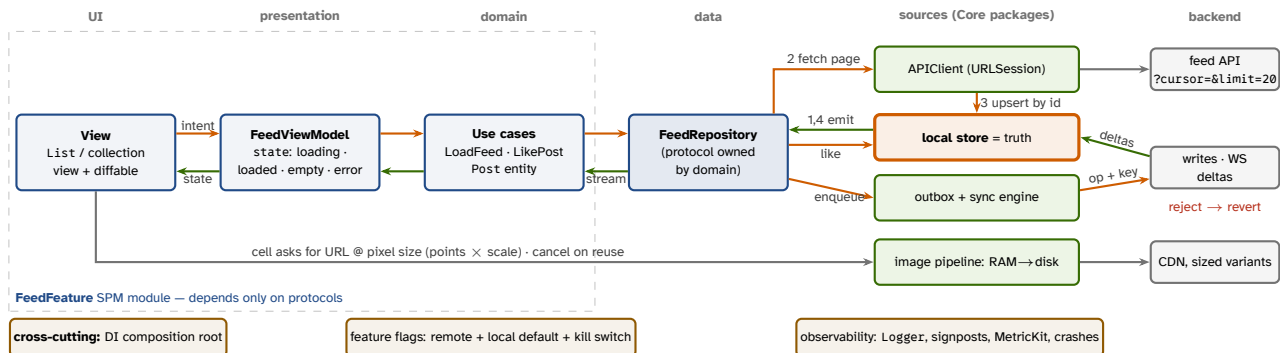


In one line: Drive the dialogue: **clarify** (functional, non-functional, scale) → **API + data model** → **high-level client architecture** → **data flow** (read + write) → **2-3 deep dives** → **trade-offs**. Design the *client*, not the backend: the **local store is the source of truth**, the network *syncs into* it.



How it works — framework on a feed

- **1 Clarify (~5 min).** Functional: scroll, infinite load, pull-to-refresh, like, detail. Out of scope: upload, stories, DMs. Non-functional: cached feed *instantly* on launch, no hitches (16.7/8.3 ms), flaky network/offline, memory-safe images. Ranking + fan-out = backend: say so, *pivot to the client*.
- **2 API + model.** GET /feed?cursor=<opaque>&limit=20 → {items, nextCursor}; items carry multi-resolution image URLs and viewerHasLiked. Normalize: entities by id in one store, the feed = ordered *id list* → one like updates every screen.
- **3 Layers.** View → ViewModel (UI state enum, no UIKit) → use cases → **Repository** (hides remote vs local) → data sources. Dependencies point inward; the domain owns the repository protocol (DIP) → fake repo in tests.
- **4 Data flow.** Read: observe local store (1) → fetch page (2) → upsert + dedup (3) → store emits (4). Write: optimistic store write → outbox → server → ack or **revert**.
- **Offline-first + sync.** Push dirty ops, pull *since cursor*. Retries need **idempotency** (client UUID key; “set” not “increment”). Backoff + jitter; per-item success. Deletes = **tombstones**. Conflicts — *name the policy*: server-wins, LWW (loses edits; clock skew), version/ETag + If-Match (412), field merge, CRDT. Triggers: foreground, NWPathMonitor, BGAppRefreshTask, silent push.
- **Pagination.** *Cursor/keyset*: stable under inserts, opaque, resumable. *Offset*: new posts at the top shift pages → duplicates and gaps. Prefetch the next page N rows before the end.
- **Images.** Memory: NSCache of *decoded, downsampled* images (cost limit, purged under pressure). Disk: size-bounded LRU of bytes. Downsample with ImageIO to points × scale, off main; coalesce duplicate URL requests; **cancel on reuse** and check the cell still shows the same id; prefetch via prefetchItemsAt.
- **Real-time.** WebSocket deltas for *visible* posts in the foreground; APNs when backgrounded. Server count *replaces*, never adds to, the optimistic one.
- **Platform.** SPM feature + Core modules (build time, boundaries). Flags: cached remote config, safe default. Token in **Key-chain**; 401 → refresh + replay.

Example — repository: read local, sync in

```
actor FeedRepository: FeedRepositoryProtocol { // init omitted
  let api: APIClient; let store: PostStore; let outbox: Outbox
  private var cursor: String? // opaque, server-made
  func posts() -> AsyncStream<Post> { store.observeFeed() }
  func loadNext() async throws {
    let page = try await api.feed(after: cursor, limit: 20)
    try await store.upsert(page.items) // dedup by id -> emits
    cursor = page.nextCursor // nil = end of feed
  }
  func like(_ id: Post.ID) async {
    await store.setLiked(id, true) // optimistic, instant
    await outbox.enqueue(.like(id), key: UUID()) // retry-safe
  }
}
```

Trade-offs to say out loud

Choice	Say
SwiftData · GRDB · files	queries + migrations · SQL control · feed cache only
SwiftUI List vs UIKit CV	velocity vs fine reuse / prefetch control
WebSocket · polling · APNs	live in foreground · fallback · app closed
Nuke / Kingfisher	use one — but justify each layer

Interview traps

- ! Designing shards — they asked for the **client**. Network as source of truth → blank screen offline.
- ! **Offset** pagination on a live feed; loading only at the bottom.
- ! Retrying a POST with no idempotency key → double like.
- ! Caching only bytes (decode per display); full-res decode on main.
- ! “It just syncs” — name the conflict policy and how you *detect* a conflict (version, not clock).
- ! try! decoding the page — one bad item kills the feed.

Remember

Clarify → **API** → **layers** → **flow** → **deep dive** → **trade-offs**. Local is truth · cursor · outbox + key · decoded RAM + disk LRU.

Likely questions

1. Why cursor? — inserts shift offsets: dupes/gaps.
2. Offline like? — local write, outbox, keyed replay, revert.
3. Wrong image flashes? — late result after reuse: cancel.
4. Memory vs disk cache? — decoded bitmaps vs bytes.
5. Testable how? — VM against a fake repository.