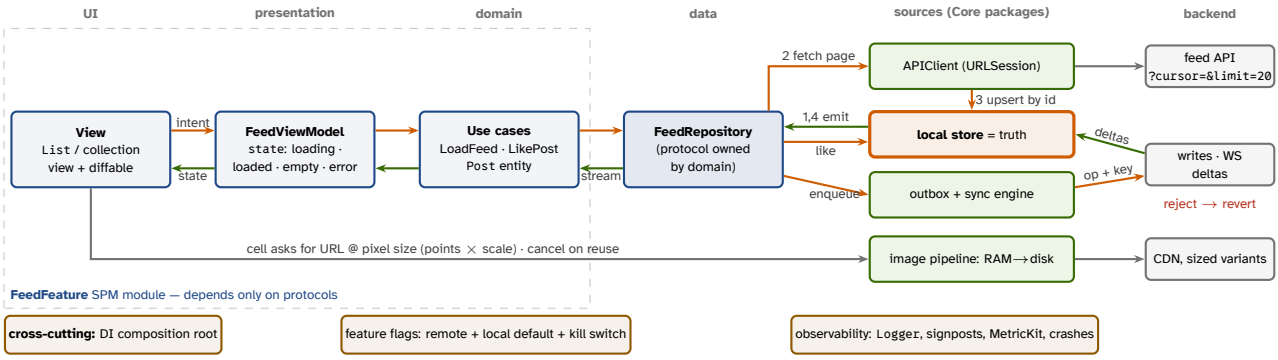
# iOS client system design — a feed, end to end



**In one line:** Order of decisions: **requirements** (functional, non-functional, scale) → **API + data model** → **client layers** → **data flow** (read + write) → the hard parts (sync, pagination, images) → **trade-offs**. On the client the **local store is the source of truth**; the network *syncs into* it.




## The design, step by step

- **1 Requirements.** 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 live on the backend; the client consumes a ranked list.
- **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** (figure): the **Repository** hides remote vs local; dependencies point inward; the domain owns its 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 (the only sure one), `NWPathMonitor`; best effort: `BGAppRefreshTask`, silent push (≤ 2–3/h).
- **Pagination.** *Cursor/keyset*: stable under inserts, opaque, resumable. *Offset*: new posts at the top push seen rows to the next page (duplicates), deletes leave gaps. Prefetch the next page N rows before the end.
- **Images.** Decoded = w×h×4 bytes, whatever the file (590 KB 2048×1536 JPEG → 12 MB). Memory: `NSCache` of *decoded, downsampled* images (cost = those bytes; 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.
- **Auth.** Token in **Keychain**; 401 → *one* shared refresh, then replay. Under rotation a second, parallel refresh presents a spent token, which RFC 9700 treats as theft: the server revokes the live token too.

## 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
  }                 // server rejects -> store.setLiked(id, false)
}
```

## Trade-offs

| Choice | What decides it |
|---|---|
| SwiftData · GRDB · files | queries + migrations · SQL control · feed cache only |
| SwiftUI `List` vs UIKit CV | velocity vs fine reuse / prefetch control |

## Traps

- ! Network as source of truth → blank screen offline and on every cold launch.
- ! **Offset** pagination on a live feed; loading only at the bottom.
- ! Retrying a POST with no idempotency key → double like.
- ! `actor` ≠ one at a time: while `loadNext()` waits at `await`, a second call reads the same cursor and fetches the page again. Keep one in-flight `Task`.
- ! 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).
- ! One bad item fails a whole `[Post]` decode, `try!` or not: decode per item, log and drop it.

## Remember

Local is truth · cursor · outbox + key · decoded RAM + disk LRU.

*Related:* solid (DIP) · mvc-mvp-mvvm · coordinator-repository-di-clean (Repository) · dependency-injection-deep · instruments-performance · observation-and-combine · actors · urlsession-networking · websockets-realtime